Beyond Blobs:
Recent Advances in Implicit Surfaces

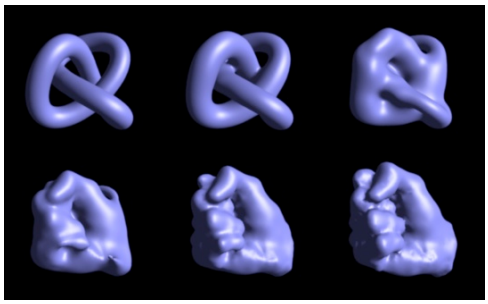
ACM SIGGRAPH 2003

Course 13

Course Organizers:

Terry S. Yoo
National Library of Medicine
National Institutes of Health

Greg Turk
Georgia Institute of Technology



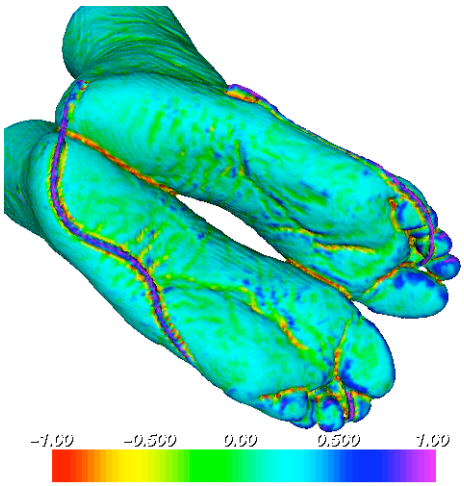
Lecturers:

Jules Bloomenthal
Unchained Geometry

H. Quynh Dinh
Stevens Institute of Technology

John C. Hart
University of Illinois Urbana-Champaign

Ross T. Whitaker
University of Utah



About the Title Page

Top figure: Raycasting view of an implicit surface constructed from the Stanford Bunny model using Compactly Supported Radial Basis Functions (CSRBF). The CSRBF method introduces multiple zero-crossings, a condition that requires some care when raycasting such models. By selecting a transfer function that accents high gradient magnitudes, the desired surface can be rendered. For more information, see the reprint later in these notes, Morse, et al., 2001. Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions, IEEE SMI 2001.

Middle figure: Shape transformation using "variational" implicit surfaces. The use of a thin-plate spline as the radial basis function makes possible interpolation between topologically distant objects. The result is smooth transformation between implicit models, including this transition from knot to fist and the corresponding change in genus. For more information, see the reprint: Turk, Greg and James O'Brien, 1999. Shape Transformation Using Variational Implicit Functions, ACM SIGGRAPH 1999.

Bottom figure: Differential Geometry of complex volumetric data using level sets. The dataset is taken from a portion of the Visible Human Project female dataset. The mean curvature of the isosurface is computed directly from the differential structure of the volume, and is shown here as a color map on the isosurface. For more on isosurfaces, level sets, and measuring curvature from volume samples see the introductory notes by Whitaker.

Forward

We are proud to present this course on recent advances in implicit techniques. We are joined this year by a distinguished panel of lecturers who each bring a different focus and insight to the problems and strengths of designing and reconstructing models using implicit surfaces. Our presentation is intended to convey a growing interest in implicit surfaces throughout our community. The publication of an introductory text by Bloomenthal [Morgan-Kaufman publishers, 1997] on implicit surfaces and the creation of a symposium on implicit surfaces (now merged with Shape Modeling International) are indicators of their growing importance.

As examples of recent advances, consider these developments: Implicit surfaces that interpolate, such as variational implicit surfaces (Turk and O'Brien: SIGGRAPH 1999), permit a smooth transition between polygonal or point data and compact analytical representations using linear algebraic techniques on radial basis functions. Level sets approach some of the same problems, not from polygonal or point data, but rather with strong ties to voxels and volume graphics. Level sets and radial basis functions have re-invigorated the study of implicit surfaces. Level sets use implicit techniques to overcome the problems of intersection detection and topology management. Radial basis functions have emerged as an exciting new method for interpolating scattered point data with an implicit surface. Finally topology and Morse theory persist as leading research areas of implicit surfaces. This course describes these recent tools as well as the fundamentals of implicit surfaces that make these tools work best.

The techniques and applications described in this course have emerged since the publication of the introductory text. Moreover, they are scattered throughout the literature and across many disciplines including modeling, visualization, medicine, and computer vision. We have collected all of these methods into a single, exciting course for SIGGRAPH 2003. We believe that a good set of notes should serve as more than a guide to the course, but it should also retain value as a desk reference for future investigations. Therefore to complement the course, we have collected some of the most important papers of the last few years, including some extraordinary bibliographic resources. We hope that you find both the course and the notes useful.

We welcome you to our presentation of Beyond Blobs: Recent Advances in Implicit Surfaces. We hope this introduction to the field will encourage you to take up your own research and participate in these rewarding efforts.

Terry S. Yoo
Greg Turk
14 April 2003

Contents

Speaker Biographies

Jules Bloomenthal received an MSc in Computer Graphics from the University of Utah and a Ph.D. from the University of Calgary. He has conducted research at the NY Institute of Technology and at Xerox PARC. He edited Introduction to Implicit Surfaces, and has written on related topics, including uniform and adaptive polygonization, polygonization of non-manifolds, convolution of skeletons, bulge elimination in implicit blends, volume/surface blends, definition of branching structures, and interactive design and display techniques. He co-chaired the 1995 Workshop on Implicit Surfaces, and has organized and lectured at several previous SIGGRAPH courses. Today he is president of Unchained Geometry.

H. Quynh Dinh is an Assistant Professor in the Department of Computer Science at the Stevens Institute of Technology. Her research interests are in modeling in computer graphics with emphasis on modeling shape metamorphosis, and computer vision techniques for shape reconstruction. She has co-authored several papers on reconstructing implicit surfaces from data generated by space carving. Quynh received her B.S. from the George Washington University in 1994, and her Ph.D. from the Georgia Institute of Technology in 2002.

John C. Hart is Associate Professor of Computer Science at the University of Illinois Urbana-Champaign. In 1993 he received an NSF award to explore implicit surfaces, and got hooked. He co-chaired the 1996 Eurographics/SIGGRAPH Workshop on Implicit Surfaces, and has organized/lectured in previous SIGGRAPH courses, including several on implicit surfaces. Hart is co-author of *Real-Time Shading* and a contributing author of *Modeling and Texturing: A Procedural Approach*, 3rd edition. Hart is the Editor-in-Chief of ACM Transactions on Graphics. He served five years on the SIGGRAPH Executive Committee and was an executive producer of the documentary "The Story of Computer Graphics."

Greg Turk is an Associate Professor in the College of Computing at the Georgia Institute of Technology, where he is also a member of the Graphics, Visualization and Usability Center (GVU). His research is primarily in the areas of computer graphics modeling and rendering. Greg has authored or co-authored numerous SIGGRAPH papers on topics including reaction diffusion textures, surface simplification, zippered polygonal meshes, and implicit surfaces. Greg received his Ph.D. in Computer Science from UNC Chapel Hill in 1992. Before coming to Georgia Tech, he was a postdoctoral researcher at Stanford and then a research scientist at UNC Chapel Hill.

Ross Whitaker is currently an assistant professor at the University of Utah, Department of Computer Science. His research interests include: computer vision, image processing, medical imaging, and computer graphics/visualization. He received his B.S. degree in Electrical Engineering and Engineering Physics from Princeton University in 1986 and his Ph.D. in Computer Science from the University of North Carolina at Chapel Hill in 1993. Previously he has been a research scientist in the User Interaction and Visualization Group at the European Computer-Industry Research Centre in Munich, Germany and an assistant professor of Electrical and Computer Engineering at the University of Tennessee.

Terry S. Yoo is a Computer Scientist in the Office of High Performance Computing and Communications, National Library of Medicine, NIH, where he explores the processing and visualizing of 3D medical data, interactive 3D graphics, and computational geometry. Previously as a professor of Radiology, he managed a research program in Interventional MRI with the University of Mississippi. Terry holds an A.B. in Biology from Harvard, and a M.S. and Ph.D. in Computer Science from UNC Chapel Hill.

Implicit Surfaces

Jules Bloomenthal
Unchained Geometry, Seattle

I. Introduction

Implicit surfaces are two-dimensional, geometric shapes that exist in three dimensional space. They are defined according to a particular mathematical form. This article examines their definition, representation, and geometric properties. Related fields are discussed and practical methods are reviewed.

As a two-dimensional analogy, imagine a drop of die spreading on a flat surface, changing color as it spreads. Tracing an infinitesimally thin range of color produces a contour. Extending to three dimensions, imagine a drop of dye, released under water, changing shape and color as it radiates outwards. In this case, an infinitesimally thin range of color produces a surface.

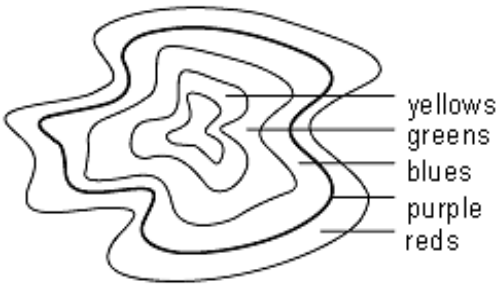
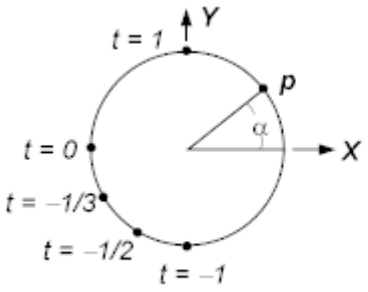


Figure 1. A contour within oil and water.

An implicit surface may be imagined as an infinitesimally thin band of some measurable quantity such as color, density, temperature, pressure, etc. The quantity varies within the volume but is constant along the surface. Thus, an implicit surface consists of those points in three-space that satisfy some particular requirement. Mathematically, the requirement is represented by a function *f*, whose argument is a three-dimensional point **p** (*i.e.*, (*x*, *y*, *z*)).

By definition, if f(***p***) = 0 then ***p*** is on the surface. f inherently characterizes a volume: those points for which $f < 0$ are on one side (nominally the 'inside') of the surface, those points for which $f > 0$ are on the other side of the same surface. *f* does not explicitly describe the surface, but implies its existence. For many functions, *f* is proportional to the distance between ***p*** and the surface. This and other attributes encourage particular forms of geometric design.

Implicit surfaces may differ in appearance, and always differ in expression, from the parametric surfaces more typical of computer aided design and computer graphics. For example, the parametric and implicit expressions for the unit circle, although describing identical shapes, greatly differ in their form and properties. In the equiangular parametric case, it is simple to compute a point on the circle at a given angle; this is not possible for the implicit representation, but it, unlike the parametric, inherently determines whether a point is inside, outside, or on the circle.



equiangular parametric
(transcendental trigonometric)

$\boldsymbol{p} = (\cos(\alpha), \sin(\alpha)), \alpha \in [0, 2\pi]$

non-equiangular parametric (rational)

$\boldsymbol{p} = (\pm(1-t^2)/(1+t^2), 2t/(1+t^2)), t \in [-1, 1]$

implicit

$\boldsymbol{p}_x^2+\boldsymbol{p}_y^2-1 = 0$

Figure 2. Expressions for the unit circle.

II. Mathematical Foundations

Analytic geometry is the branch of mathematics devoted to the relationship between geometry and the mathematical expression of the coordinates of points in space. When applied in three dimensions, it is called solid analytic geometry. If geometric relationships between points in three-space are compared to corresponding mathematical (*i.e.*, algebraic) relationships between the coordinates (*x*, *y*, and *z*) of the points, it is possible by algebraic proof to establish a geometric property. For example, the distance between the centers of two spheres can be compared algebraically with the sum of their radii, thereby predicting whether the spheres intersect geometrically.

Analytic geometry has been applied to a wide variety of mathematical functions to establish their properties (especially tangency) and to enable their graphical display. The relationship between the coordinates of points on a geometric object is fundamental to geometric design.

An explicit equation might express the *z* coordinate in terms of the x and *y* coordinates: that is, $z = f(x, y)$. Such a surface is called a height field. The different treatment of *z* from that of *x* and *y* inherently limits shape. For example, a height field cannot contain an overhang or a vertical slope (similarly, a planar curve produced by an explicit equation $y = f(x)$ cannot double-back or be closed, nor can it parallel the *y* axis).

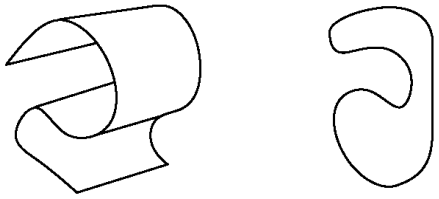


Figure 3. Surface and curve inexpressible by an explicit equation.

There are at least two approaches that treat coordinates symmetrically, thereby resolving the difficulty with vertical slopes. One approach is parametric: each of the coordinates is expressed according to the geometric dimension of the object. That is, for a one-dimensional curve embedded in two-space, $x = f_x(t)$ and $y = f_y(t)$. For a two-dimensional surface embedded in three-space, $x = f_x(s, t)$, $y = f_y(s, t)$, and $z = f_z(s, t)$. Parametric curves and surfaces provide a convenient mapping from the object to the space within which it is embedded. For example, any three-dimensional point on the surface may be specified by an (*s*, *t*) ordered pair. This forward mapping (or parameterization) is useful for display, surface texture, and other applications.

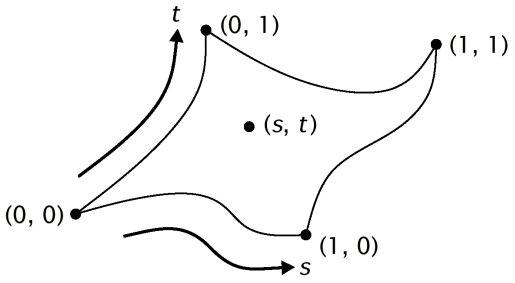


Figure 4. A parametric surface

The other symmetric approach is implicit: the coordinates are treated as functional arguments rather than functional values. In general for surfaces, $F(x, y, z) = c$, where c is a point in $\Re^n$ and F maps $\Re^3 \rightarrow \Re^n$. For most applications, n is 1 and c is a scalar constant. When c is zero, f implicitly defines a locus called an implicit surface; that is, the set of points $\{\boldsymbol{p} \in \Re^3: f(\boldsymbol{p}) = 0\}$ is the implicit surface defined by *f*. *f* is called the *implicit surface function* (also known as a `scalar field,' `field function,' or `potential function'). The implicit surface is sometimes called the zero set (or zero surface) of *f* and may be written $f^{-1}(0)$ or $Z(f)$.

f is typically specified either by 1) discrete samples, usually uniformly spaced within a finite volume, 2) mathematical functions, in which one or more equations evaluate the coordinates of ***p***, or 3) procedural methods, in which an algorithmic process evaluates ***p***.

Discrete samples are usually physical measurements such as opacity, density, etc. Related to $f^{-1}(0)$ is the isosurface (also called a level set or level surface), which is $\{\boldsymbol{p} \in \Re^3: f(\boldsymbol{p}) = c\}$, where c is the isocontour value of the surface. Isosurfaces are popular for scientific visualization (of medical, material, or atmospheric data, for example), especially when varying c is of interest.

If *f* is a mathematical function, it may contain any mathematical expression. If *f* is polynomial only, it is called algebraic (*i.e.*, it contains a finite number of terms). The resulting surface is called an algebraic surface (algebraic surfaces belong to the domain of algebraic geometry, which is the study of zeros of polynomial equations, the algebraic representation of figures, and, frequently, those properties that remain invariant when the equations undergo transformation). Non-polynomials are called transcendental; they arise frequently in scientific disciplines and include the trigonometric, exponential, logarithmic, and hyperbolic functions.

If *f* is an arbitrary procedural method (*i.e.*, a black box function that evaluates ***p***), the geometric properties of the surface can be deduced only through numerical evaluation of the function.

The implicitly defined surface can be bounded (*i.e.*, finite in size), such as a sphere, or unbounded, such as a plane. The value of *f* at a point ***p*** is often a measure of proximity between ***p*** and the surface. The measure is Euclidean if it is ordinary (*i.e.*, physical) distance. For an algebraic surface, *f* measures algebraic distance.

Those geometric and topological aspects of implicit surfaces that affect practical issues such as surface representation and display are discussed in the following section.

II A. Continuity, Differentiability, and Manifoldness

In order that normals be defined along an implicit surface, the function *f* must be continuous and differentiable. That is, the first partial derivatives $\delta f/\delta x$, $\delta f/\delta y$, $\delta f/\delta z$ must be continuous and not all zero, everywhere on the surface. Such a function is known as analytic (or is considered analytic

in a region that is differentiable). When given as an ordered triplet, the partials define the gradient ∇f of the function. The unit-length gradient is usually taken as the surface normal.

For example, the gradient of the unit sphere, $f(x, y, z) = x^2+y^2+z^2-1$, is $(2x, 2y, 2z)$. Thus, a point on the sphere at (1, 0, 0) has a (unit-length) normal of (1, 0, 0), which points outwards (complying with display convention). Negating the implicit function will invert the surface (*i.e.*, its sense of inside and outside), with a corresponding reversal of surface normals.

For a 'black-box' or other non-differentiable function, the gradient may be approximated numerically using forward differences and some discrete step-size Δ:

$$\nabla f(\boldsymbol{p}) \cong (f(\boldsymbol{p}+\Delta x)-f(\boldsymbol{p}),\ f(\boldsymbol{p}+\Delta y)-f(\boldsymbol{p}),\ f(\boldsymbol{p}+\Delta z)-f(\boldsymbol{p}))/\Delta,$$

where Δx, Δy, and Δz are displacements by Δ along the respective axes. For small Δ, the error is proportional to Δ. If ∇f is computed by central differences:

$$\nabla f(\boldsymbol{p}) \cong (f(\boldsymbol{p}+\Delta x)-f(\boldsymbol{p}-\Delta x),\ f(\boldsymbol{p}+\Delta y)-f(\boldsymbol{p}-\Delta y),\ f(\boldsymbol{p}+\Delta z)-f(\boldsymbol{p}-\Delta z))/2\Delta,$$

the error is proportional to Δ^2.

If the gradient is non-null at a point $\boldsymbol{p}$, then $\boldsymbol{p}$ is said to be regular (or simple) and $\nabla f(\boldsymbol{p})$ is normal (*i.e.*, perpendicular) to the surface at $\boldsymbol{p}$. If, however, the gradient (or, equivalently, the tangent vector) is indeterminate, the point is singular (also called critical or non-regular). The normal at a singular point is sometimes given as the average of the normals of surrounding vertices.

A regular value c of f exists if, for every $\boldsymbol{p} \in f^{-1}(c)$, $\boldsymbol{p}$ is a regular point. For example, the cone $f = -x^2+y^2+z^2$ is regular with the exception of a singularity at the origin. Thus, 0 is not a regular value of f, but all others are. The detection of singular points for low degree algebraic curves and surfaces is described in (1).

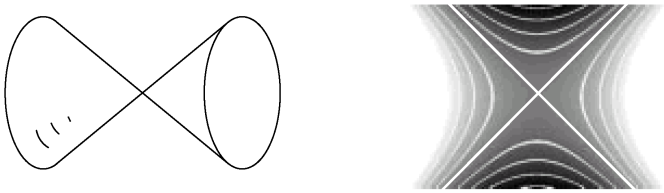


Figure 5. The apex of a cone is a singular point.
left: zero set, right: cross-section (contours added) reveals non-zero values are regular

If the surface is regular and the second partial derivatives are continuous, then the surface will have continuous curvature (*i.e.*, the surface is G^2 continuous). Also, if the surface is regular, it is manifold.

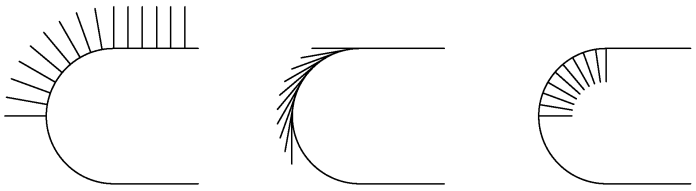


Figure 6. Normal, tangent, and curvature vectors.
normal vectors (left) and tangent vectors (middle) are continuous
curvature (right) is directed inwards along the semi-circle, then instantly becomes null

The 2-manifold is a fundamental concept from algebraic topology (2) and differential topology (3). It is a surface embedded in $\Re^3$ such that the infinitesimal neighborhood around any point on the

surface is topologically equivalent (`locally diffeomorphic') to a disk. Intuitively, the surface is `watertight' and contains no holes or dangling edges. Typically, the manifold is bounded (or closed). For example, a plane is a manifold but is unbounded and thus not watertight in any physical sense. A manifold-with-boundary is a surface locally approximated by either a disk or a half-disk. All other surfaces are non-manifold.

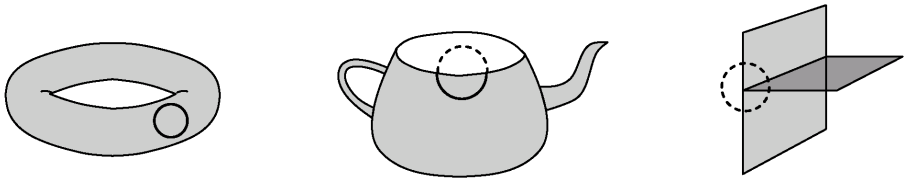


Figure 7. Manifold, manifold-with-boundary, and non-manifold.

Although f is sometimes called an 'implicit function,' that term formally refers to the implicit definition of one variable in terms of one or more other variables. For example, $f(x, y, z) = 0$ may be rewritten as $f(x, y, g(x,y)) = 0$. The Implicit Function Theorem gives those conditions under which a unique g exists and is C^1 continuous (4).

From the implicit function theorem it may be shown that for $f(\boldsymbol{p}) = 0$, where 0 a regular value of f and f is continuous, the implicit surface is a two-dimensional manifold (5, prop. 4.16). The Jordan-Brouwer Separation Theorem states that such a manifold separates space into the surface itself and two connected open sets: an infinite `outside' and a finite 'inside' (3).

Consider two examples for which no manifold exists. The first is simply $f(\boldsymbol{p}) = 0$. Here, ∇f is everywhere 0, there is no `inside' nor `outside' and no boundary between the two. The second is a degenerate sphere $f(x, y, z) = x^2+y^2+z^2$. Here, $\nabla f = (2x, 2y, 2z)$, which is null at the origin, the only point satisfying f; intuitively, the `inside' is degenerate. Whether or not a surface is manifold concerns its polygonal representation.

III. Polygonal Representation

For many applications it is useful to approximate an implicit surface with a mesh of triangles or polygons (formally, a discrete set of piecewise-linear, semi-disjoint elements). For differentiable f this is always possible because all manifold surfaces may be triangulated (6).

Although approximate, the mesh is a practical representation for $Z(f)$. (7) notes the difficulty in obtaining exact mathematical representations (parametric or implicit) for conceptually simple surfaces such as the offset surface (a surface a fixed distance from a base surface) and the equi-distance surface (a surface lying between two surfaces). There are simple procedural descriptions for both of these surfaces, each readily converted to a polygonal mesh.

Mesh conversion, popularly known as polygonization, usually involves partitioning space into convex cells (typically cubes or tetrahedra). A cell is transverse if any of its edges intersects the implicit surface (*i.e.*, one edge endpoint evaluates negatively, the other positively). For each transverse edge, a surface vertex is computed (by the Intermediate Value Theorem, a point $\boldsymbol{p}$: $f(\boldsymbol{p}) = 0$ must exist along a transverse edge if f is continuous). The surface vertices belonging to the transverse edges of a cell are connected to form one or more polygons (alternatively, patches may be produced). The edges of the polygons lie within the faces of the cell.

The order of vertex connectivity is often stored in a table of polarity configurations of the cell corners. For the cube (8 corners) and the tetrahedron (4 corners, *i.e.*, a three-dimensional simplex) there are 256 and 16 possibilities, respectively. Any convex cell may be decomposed

into tetrahedra, thereby simplifying the case analysis. Any (possibly non-planar) *n*-sided polygon produced may be decomposed into *n*+2 triangles; alternatively, *n* triangles can radiate from a polygon centroid.

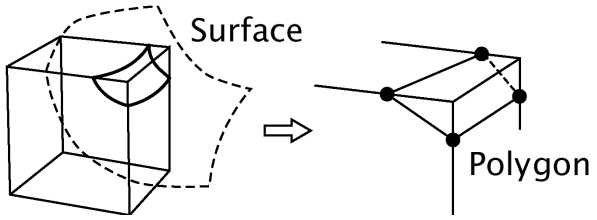


Figure 8. Polygonization.

A review of discrete and continuous polygonization methods is given in (8). An analysis of implementation complexity, polygon count, and topological and geometric accuracy is given in (9). Other criteria, such as the number of function evaluations, the adaptive distribution of polygons, and visual appearance are discussed in (10). A review of discrete data methods is given in (11).

Software implementations typically utilize exhaustive enumeration (12), subdivision (13), or numerical continuation (14).

III A. Exhaustive Enumeration

Exhaustive enumeration operates on a set of samples of *f* arranged as a regular, typically rectilinear lattice known as a *scalar grid* or *voxel array*. The samples may be experimental, such as CAT and MRI scans, or computed, as in simulations of fluid flow. The lattice is readily represented by a three-dimensional memory array, which can be filled by a hardware scanner in constant time.

Once the samples are obtained, each transverse cell is polygonized. Given $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, lattice neighbors of opposite sign, a surface vertex $\boldsymbol{v}$ is usually is computed using linear interpolation:
$\boldsymbol{v} = \alpha\boldsymbol{c}_1+(1-\alpha)\boldsymbol{c}_2$, where $\alpha = f(\boldsymbol{c}_2)/(f(\boldsymbol{c}_2)-f(\boldsymbol{c}_1))$

This method is popularly known as `marching cubes' and may be optimized for one plane of cells at a time (15). Cells may be pre-sorted according to minimum and maximum *f*; should an offset (*i.e.*, isovalue) be applied to *f*, transverse cells can be quickly identified from the sort (16, 17).

The application of marching cubes algorithms includes electron motion (18), computational electromagnetics (19), polypeptide visualization (20), biomedical visualization (21), and molecular modeling (22, 23). Rendering and polygonization schemes for irregular lattices, such as produced by finite element methods, are discussed in (24).

Unlike exhaustive evaluation, subdivision and continuation operate on synthetic functions (*i.e.*, *f* may be algebraic or procedural), typically for the purpose of design (25). *f* may be evaluated at arbitrary locations, which allows methods such as binary sectioning to compute surface vertex locations with arbitrary precision, unlike linear interpolation. These algorithms seek to minimize the number of evaluations of *f*, which may be arbitrarily demanding to evaluate.

III B. Subdivision

Subdivision is the recursive division of space into sub-volumes. With the exception of the root cell (which completely encloses the object), subdivision is applied only to transverse cells. Surface

vertices and polygons are produced from the 'terminal' cells, which collectively enclose the implicit surface. Forms of subdivision include the octree (26), *KD*-tree and bintree (27).

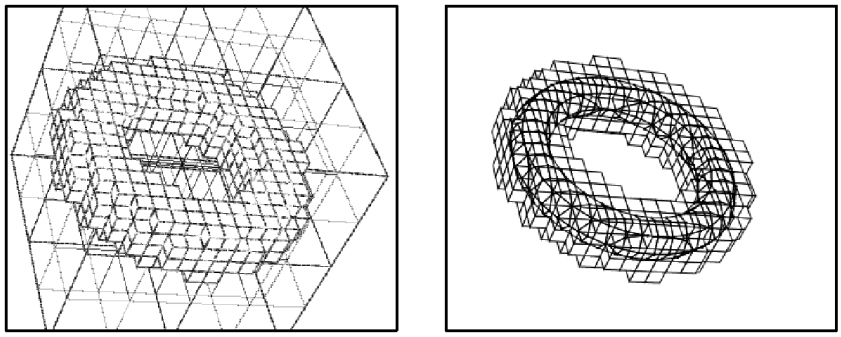


Figure 9. Subdivision enclosing a torus.
left: octree after four recusions, right: polygons in terminal nodes

The cube is the only polyhedron that may be subdivided into similarly shaped and oriented sub-polyhedra (one type of tetrahedron, the Kuhn simplex, may be subdivided into similar sub-tetrahedra (28)). Without similarity, the sub-volumes become thinner, yielding poorly shaped polygons.

Subdivision requires a priori knowledge of the extent of a surface; this can be computed if *f* consists of primitives, each with an associated range. Subdivision must also determine whether a cell is transverse; typically this is achieved by examining *f* at cell corners. If the corners are of the same sign, the surface may nonetheless penetrate the cell; robust and accurate determination of transversality is possible using interval analysis (29; 30, 31), Lipschitz constants (32, 33), or derivative bounds (34).

III C. Continuation Methods

Rather than subdivide a large cell, it is possible to propagate from one small cell to another. This is a form of numerical continuation, a class of techniques usually divided into piecewise-linear and predictor-corrector (14).

III C 1. Piecewise-Linear Continuation

Piecewise-linear principles (see (35), (36)) have been applied to implicit surfaces using a tetrahedral cell (37) and a cubic cell (38). Beginning with a single transverse `seed' cell, new cells are propagated across transverse faces until the entire surface is enclosed.

Because only transverse cells are generated, piecewise-linear continuation requires $O(n^{-2})$ function evaluations, where n is cell size. In comparison, exhaustive enumeration requires $O(n^{-3})$ samples. Compared with subdivision, continuation appears less prone to under-sampling.

Exhaustive enumeration yields all disjoint (and detectable) surface components. Continuation, however, produces a single component for each seed cell; to polygonize all disjoint surface components, continuation must be performed for each, using an appropriate seed cell.

III C 2. Predictor-Corrector Continuation

Predictor-corrector methods (similar to `meshing' used in numerical grid generation) apply directly to the surface, creating elements (usually triangles or polygons) by joining an initial surface point with additional points. New points are computed by displacement from a known point along the

tangent plane and then corrected (*e.g.*, using Newton iteration) onto the surface (39). These methods are problematic for surfaces because surface vertices are not intrinsically ordered (unlike a one-dimensional contour), which complicates detection of global overlap.

III D. Adaptive Polygonization

Polygonization is a sampling process; if the spacing between samples is large with respect to surface curvature, detail is lost. Resolution requirements may also change with viewpoint. Any fixed sampling rate may be excessive for relatively flat regions of the surface and insufficient for relatively curved regions. If the cell size is inversely proportional to local curvature, the resulting adaptive polygonization minimizes polygon count while maintaining geometric accuracy (40). Both subdivision and continuation may be performed adaptively (13). Accurate representation of non-differentiable f, however, may require explicit computation of its singular points.

Surface refinement is an adaptive method in which a coarsely polygonized surface is followed by subdivision of insufficiently accurate polygons. For example, if the center of a triangle is too distant from the surface, the triangle may be split at its center, which is moved to the surface (41). Similarly, a triangle may be divided along its edges if the divergence between surface normals at the triangle vertices is too great (42).

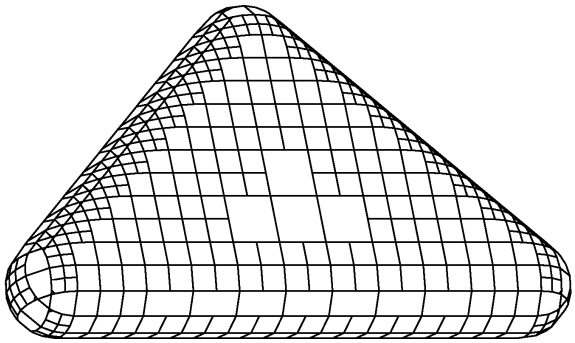


Figure 10. Adaptively polygonized object.

Adaptive polygonization is also possible through the use of 'physically-based particles' distributed along the implicit surface, with particle density locally proportional to surface complexity (43). Particle location is determined by various heuristics, including the use of the gradient of f (44; 45). During animation, the topology of a particle-based polygonization may be maintained by operating on those critical points at which topological change occurs (46).

III E. Ad Hoc Polygonization

III E 1. Non-Manifold Polygonization

Although a manifold-with-boundary may be specified by a continuous function, all points off the zero set are of the same sign. Consequently, conventional polygonization fails.

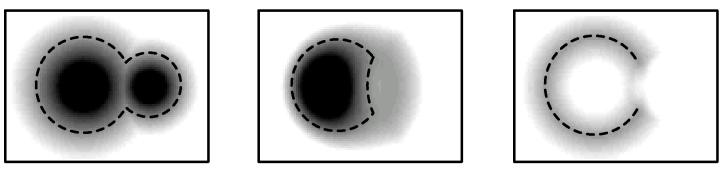


Figure 11. Union, difference, and manifold-with-boundary (zero contours are dashed)**.**
left to right: $\min(f_1, f_2)$, $f_1 max(f_1,f_2)$, $abs(f_1) - min(0,f_2)$
where $f_1 = ||\boldsymbol{p}-\boldsymbol{c}_1||/r_1 - 1$ and $f_2 = ||\boldsymbol{p}-\boldsymbol{c}_2||/r_2 - 1$ are two circles

As suggested in (47), a non-manifold can be implicitly represented by extending the definition of *f* to be the separation between arbitrary regions of space. A continuation method using this scheme is given in (48).

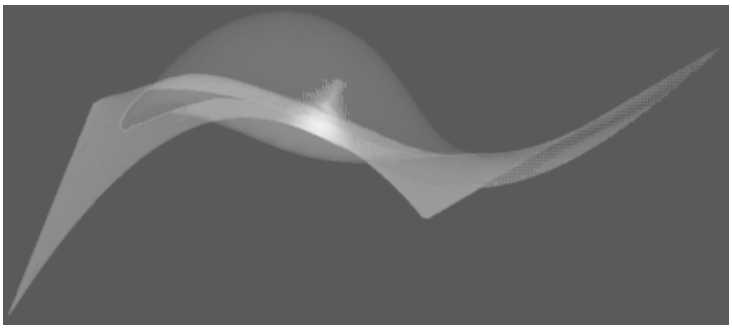


Figure 12. Polygonized non-manifold.

III E 2. CSG Polygonization

The primitives in constructive solid geometry (CSG) may be represented implicitly and combined by set-theoretic Boolean operations. These operations may create hard-edged junctions that conventional polygonizers cannot accurately approximate. A method presented in (49) computes surface vertices on cell edges in the usual manner, but the CSG state at the cell corners determines whether a crease is applied to the resulting polygon.

Figure 13: Polygonized CSG objects.
courtesy Brian Wyvill and Kees van Overveld

IV. Relation to Other Surfaces

IV A. Relation to Parametric Surfaces

Both parametric and implicit methods are well developed in computer graphics. A modern treatment of parametric surfaces is in (50). Traditionally, computer graphics has favored polynomial parametric over implicit surfaces because they are simpler to render and more convenient for geometric operations such as computing curvature and controlling position and tangency. Parametric surfaces are generally easier to draw, tessellate, subdivide, bound, and navigate along (51).

An implicit surface naturally describes an object's interior, whereas a comparable parametric description is usually piecewise. The ability to enclose volume and to represent blends of volumes provides a straightforward (although less precise) implicit alternative to fillets, rounds, and other 'free-form' parametric surfaces that require care in joining so that geometric continuity is established along the seams (52). Consequently, animations of organic shapes commonly employ implicit surfaces.

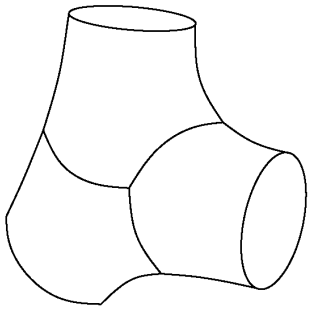


Figure 14. A free-form parametric surface.

Point classification (determining whether a point is inside, outside, or on a surface) is simpler with implicit surfaces, depending only on the sign of f. This facilitates the construction of complex objects from primitive ones (25), and simplifies collision detection (53).

Certain shapes may be described exactly in both parametric and implicit form, as demonstrated for the unit circle. The three-dimensional case is:

trigonometric $x = (\cos(\alpha)\cos(\beta), y = \sin(\alpha), z = \cos(\alpha)\sin(\beta), \alpha \in [0, \pi], \beta \in [0, 2\pi)$

rational $x = 4st/w, y = 2t(1-s^2)/w, z = (1-t^2)(1+s^2)/w$, for $w = (1+s^2)(1+t^2), s,t \in [0, 1]$

implicit $f(x, y, z) = x^2+y^2+z^2-1$

(25) observes that the implicit representation is often more compact.

Points on the parametrically defined sphere are readily found by substitution of α and β into the equations for x, y, and z (similarly for s and t). By sweeping (α, β) through its domain in $\Re^2$, points along the entire surface are conveniently generated for display, piecewise approximation, etc.. This natural conversion from the parametric (two-dimensional) space of a surface to the geometric (three-dimensional) space of an object is a fundamental convenience. There is no comparable mechanism for implicit surfaces (unless the implicit equation is reduced to two explicit equations, as is possible for some low degree algebraic surfaces).

The surface normal for a regular point on an implicit surface is computed as the unit-length gradient; the normal to a parametric surface is usually computed as the cross-product of the surface tangents in the two parametric directions.

The class of algebraic surfaces subsumes that of rational parametric surfaces. Thus, implicit surfaces are more likely to be closed under certain operations than their parametric counterparts. For example, the offset surface from an implicit surface remains an implicit surface, whereas the offset from a parametric surface is, in general, not parametric (54).

Because parametric and implicit forms have complementary advantages, it is useful to convert from one form to the other. To calculate the intersection of two parametric surfaces, for example, the parametric equation for one surface may be substituted into the implicit form for the other (7).

Conversion from parametric to the implicit form is known as implicitization, and may be performed on any rational parametric surface (or curve) (55, 56). This is accomplished by elimination of the parameters in the parametric form. For example, elimination of s and t from the rational equations yields the implicit form in x, y, and z (7, 57, 58, 55, 1).

Implicitization is not always tractable; although the implicit and parametric representations of a curve are of the same degree, the implicit representation of a parametric triangular patch of degree n is degree n^2 and the implicit representation of a tensor product surface of degree m by n is degree $2mn$ (54). The number of terms is $O(n^2)$ (56), so that the implicitization of a bicubic patch is degree 18, with 1330 terms (54). In some common cases the degree and number of terms are significantly reduced (59).

The conversion from implicit to parametric form is known as parameterization. Associating a point (x, y, z) with its equivalent parametric position (s, t) is known as inversion (60). Parameterization is not always possible because implicit surfaces defined by certain polynomials of fourth and higher degree cannot be parameterized by rational functions (61). Conversion is always possible for non-degenerate quadrics and for cubics that have a singular point.

IV B. Relation to Solid Modeling and CSG

Point classification, which is inherently implicit, is fundamental to solid modeling, a geometric method that emphasizes the unambiguous calculation of well defined geometric properties (such as volume, center of mass, etc.). A solid model consists of a surface and its interior; it may be specified or modified by several robust methods.

The theoretical underpinnings of solid modeling are found in point-set topology: a ‘reasonable’ solid is ‘all material,’ *i.e.*, a bounded, closed set of points in $\Re^3$ that is regular (free from any dangling points, edges, or faces). Regularity is ``widely used as a characterization of reasonable solids'' (12). A finite, regular point-set is called an R-set (62), and a solid model is usually confined to an R-set whose surface is analytic.

An initial means to specify solids was introduced in (25), which developed a `constructive geometry' for the purpose of defining complex shapes derived from operations, including blend, upon simple implicit primitives (the operations, known as Comba-Ricci sums, are reviewed in (63)). A primitive solid is described by $P(\boldsymbol{p}) < 0$ (convention of sign is not generally observed for implicit surfaces).

The closed-form definition given in (25) was superceded by constructive solid geometry (CSG), which is characterized by a `bottom-up' binary tree evaluation. The leaf nodes are usually arbitrarily placed and oriented low degree polynomial primitives (*viz.*, the parallelepiped, sphere, ellipsoid, cylinder, cone, and torus). The internal nodes represent regularized Boolean set-theoretic operations (union, intersection, and difference), which are common in computer aided design and manufacture.

Union is given as $\min(P_1, P_2)$; intuitively, if a point is within any sphere it evaluates negatively, regardless of the number of surrounding spheres. Intersection is given as $\max(P_1, P_2)$; *i.e.*, if a

point is outside any sphere, it evaluates positively. Difference is given by $\max(P_1, -P_2)$. Because the solid model unambiguously separates inside from outside, it defines a realizable manifold and polygonization methods typically succeed.

min and max are semi-analytic, however, as they are not everywhere differentiable. Analytic expressions approximating union and intersection (for n functions) are given in (25) as:

$$union_\alpha\,(f_1, \ldots, f_n) = (f_1^{-\alpha} + \ldots + f_n^{-\alpha})^{-1/\alpha}$$

and

$$intersect_\alpha\,(f_1, \ldots, f_n) = (f_1^{\alpha} + \ldots + f_n^{\alpha})^{1/\alpha}$$

where $\alpha > 0$. The limits of these functions (as α approaches zero) are min and max, respectively. Forms of union and intersection are also given by *R*-functions, which are sets of semi-analytic functions that partition the real line (64, 65).

Figure 15. Set-theoretic difference and intersection of two circles.

The solid model is usually represented by a plane model and is called a boundary representation, or BRep. The plane model, developed by Möbius, is a planar directed graph containing a finite number of *n*-dimensional facets (*i.e.*, faces, edges, and vertices) that represent the boundary of an object (66). A computer implementation is the 'winged-edge' data structure (67).

According to planar graph theory (from algebraic topology), a plane model is a realization of a 2-manifold if it 1) divides a surface such that every edge of a face is identified with one and only one other oppositely directed edge of an adjacent face, 2) at each vertex a cycle of faces exist such that two consecutive faces share an edge emanating from the vertex, and 3) the division is orientable, meaning each face is bounded by consistently oriented edges with all edges used once (12).

To compute the BRep of a CSG model, typically each leaf node of the CSG tree is converted to a BRep. Then, in bottom-up order for each internal node, appropriate Euler operators are applied to the two child nodes, producing at each internal node an intermediate regularized BRep (*i.e.*, the BRep must be closed under all Boolean operations (12, 68)). Regularity typically disallows non-manifold objects, although support does exist for non-regular and other free-form methods (69). To maintain a regularized BRep at each node, each set operation must support a multitude of geometric intersections, many of which are difficult to implement robustly in the presence of numerical errors (12).

Either an ordered sequence of edges around each vertex or an ordered list of edges around each face are sufficient to reproduce an adjacency graph embedded in a two-dimensional manifold (70). In other words, the topologically more complete BRep can be obtained from the representationally simpler mesh produced by polygonization.

IV C. Relation to Algebraic Surfaces

When *f* is polynomial the corresponding implicit surface is an algebraic surface (also called algebraic set). The basis of the polynomial is usually the power basis (i.e, x, x^2, x^3 ...) but could be another, such as the Bernstein (used by Bézier curves and surfaces). For the purpose of geometric modeling, coefficients are limited to the reals. The degree of an algebraic expression is the maximum degree of its terms. When *f* is linear (degree 1), it describes a plane. When *f* is

quadratic (degree 2), it describes a quadric surface, which is an ellipsoid, sphere, cylinder, cone, paraboloid, hyperboloid, or hyperbolic paraboloid, or is degenerate (a plane, line, or point). The quadrics may also be expressed in trigonometric form; when exponentiated, the trigonometric terms yield a superquadric (71).

It may be difficult to perform geometric operations, such as surface/surface intersection, on algebraic surfaces of degree greater than three because the degree of the resulting surface is often very high.

The animation of algebraic surfaces is discussed in (72). A general development of algebraic surfaces is given in (54, 73, 74). Other salient properties of algebraic surfaces are discussed in (1, 75).

Algebraic surfaces may also be defined parametrically by three independent equations, one each for x, y, and z. Further, the equations may be rational; that is, $x = f(s, t) / g(s, t)$, where f and g are polynomials (similarly for y and z). Rational polynomial parametric surfaces are a subset of implicit algebraic surfaces; each rational parametric surface may be expressed in implicit form, but the converse does not hold (56). A theorem by Noether states that a planar algebraic curve $f(x, y) = 0$ has an equivalent rational parametric form if and only if f has a genus of 0. A similar theorem for surfaces is provided by Castelnuovo (75). All non-degenerate quadric surfaces in implicit form may be converted to parametric form (61).

Algebraic surfaces may define non-manifolds. For example, the Steiner surface $(x^2y^2+y^2z^2+z^2x^2+xyz = 0)$ contains the coordinate axes, where it is singular.

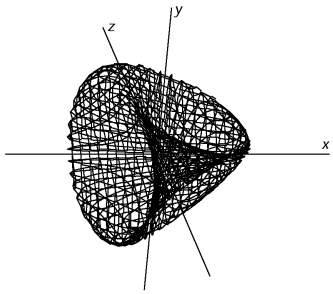


Figure 16. The Steiner patch.

An algebraic surface must consist of a finite number of components, which is not the case for transcendental functions. For example, $f(x, y) = cos(x)sin(y)-1 = 0$ yields zero-contours throughout the plane.

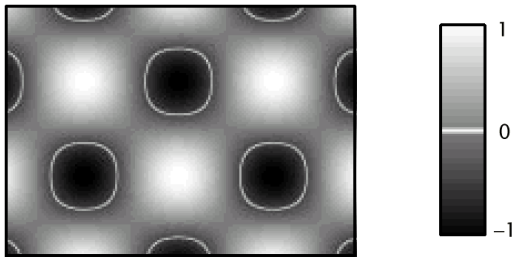


Figure 17. A transcendental function (zero-contours highlighted in white)**.**

An algebraic representation for a particular surface is not unique. A plane passing through the origin is specified by $ax+by+cz$, for example, and the same plane is described by $a = b = c = 1$ and by $a = b = c = 1/3$, but only the latter yields a plane normal (a, b, c) of unit length.

Algebraic surfaces may be interpolated by interpolating the corresponding algebraic equations (76). For example, a torus $((x^2+y^2+z^2+r_{major}^2-r_{minor}^2)^2-4r_{major}^2(x^2+z^2))$ may be interpolated to a sphere $(x^2+y^2+z^2-r^2)$.

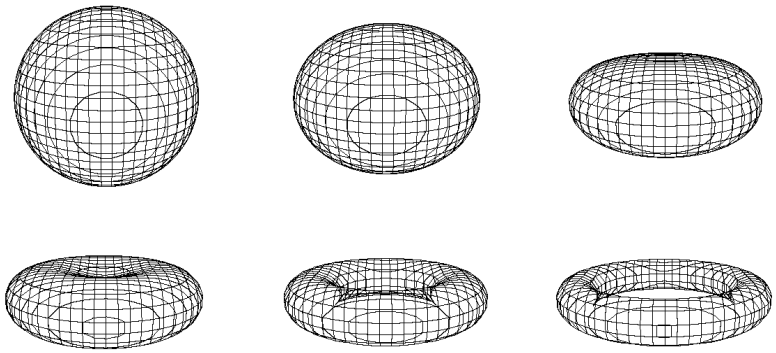


Figure 18. Interpolation of algebraic functions.

Related to the quadric surface is the `blobby molecule,' a blend of primitive 'atoms' (usually spheres or ellipsoids) (77). Each primitive P_i computes a normalized distance r_i (usually to the center of a sphere or to the foci of an ellipsoid). The molecule is given by $\Sigma\, h(r_i)-t$, where h is a blend function and t is some threshold.

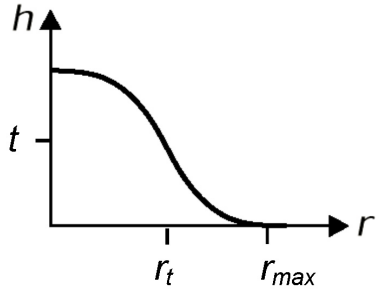


Figure 19. Contribution of a primitive P as a function of distance.

h is usually monotonic and sigmoidal. The domain r_{max} of h affects the primitive's range of influence and the shape of h determines the primitive's radius in isolation and blending characteristics. In (77) h is an exponential; in (78) piecewise quadratics produce what are popularly called 'metaballs' and in (38) a sixth degree polynomial produces 'soft objects.'

The geometric continuity of h at r_{max} determines the effect of one atom on its neighbor. For example, (38) gives:

$h(r) = 1-(4/9)r^6+(17/)9r^4-(22/9)r^2$ for $r \in [0, 1]$, 1 for $r < 0$, 0 for $r > 1$,

where $r = d/R$, d is distance to the primitive, and R is the range of influence of the primitive. This may be factored into:

$(r^2-1)^2\ (9-4r^2)/9$

The two roots at $r = 1 = r_{max}$ imply an order of continuity of 1. That is, if the influence of two atoms overlap, the seam between the area of mutual influence and the areas influenced only by one atom is G^1 continuous. Another function due G.Wyvill is $(1-r^2)^3$, which, having three multiple roots at $r = 1$, implies a G^2 continuous blend (79, chapter 5).

The union of two algebraic surfaces is usually given by the product of the corresponding algebraic functions. Intuitively, if f is zero, then any multiple of f, including multiplication by another function, is zero. For example, consider two spheres given by $f_1 = ||\boldsymbol{p}-\boldsymbol{c}_1||-r_1$ and $f_2 = ||\boldsymbol{p}-\mathbf{c}_2||-r_2$, where $\boldsymbol{c}_i$ and r_i are the centers and radii. Each function is negative inside the sphere's radius and positive outside.

The multiplication f_1f_2 confuses the sense of inside and outside, however. That is, points that are within both spheres as well as points beyond both spheres evaluate positively; only points within one and only one sphere evaluate negatively. This produces internal boundaries that are difficult to polygonize and typically undesirable in geometric design. In contrast, solid modeling operates on volumes, rather than surfaces, and does not produce internal boundaries.

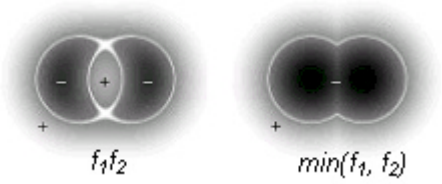


Figure 20. Algebraic (left) **and set-theoretic** (right) **unions** (zero set is highlighted in white)**.**

The complexity of an algebraic surface can be partly understood in terms of intersections with curves or surfaces. A generalization of Bezout's Theorem states that an algebraic curve of degree m intersects an algebraic surface of degree n in at most mn points (assuming no part of the curve is common with the surface), and that the intersection of a surface of degree m with a surface of degree n is an algebraic curve of degree mn or less (75). For example, a line may intersect an algebraic surface of degree m no more than m times; two ellipses (each of degree 2) may intersect at no more than four points.

V. Geometric Operations

V A. Deformations

An implicit surface may be defined by a deformation. A deformation D maps each point in three-space to some new location; that is, $\boldsymbol{p}' = D(\boldsymbol{p})$. If D is to apply to an implicit surface, the inverse of D must be applied to the space within which the implicit surface is embedded; in other words, $f_D(\boldsymbol{p}) = f(D^{-1}(\boldsymbol{p}))$, where f_D is the deforming implicit function. For example, to scale the unit circle by 2, the coordinate system is scaled by ½ so that points that satisfy f become twice as far from the origin.

A tangent vector $\boldsymbol{v}$ and normal vector $\boldsymbol{n}$ of the undeformed surface may be transformed to yield the tangent and normal vectors of the deformed surface (80). Specifically, $\boldsymbol{v}D = J\boldsymbol{v}$ and $\boldsymbol{n}D \propto J^{-1}\mathbf{n}$, where J is the Jacobian of D, given by $\delta D(\boldsymbol{p})/\delta \boldsymbol{p}$.

Deformation includes the twist, bend, and taper operations introduced by (80). In (81) twist is applied to implicitly defined swept surfaces. Other deformations include offset of a surface in the direction of its normal or in arbitrary directions (82); the offset may be a fixed amount or may be governed by a displacement map (53).

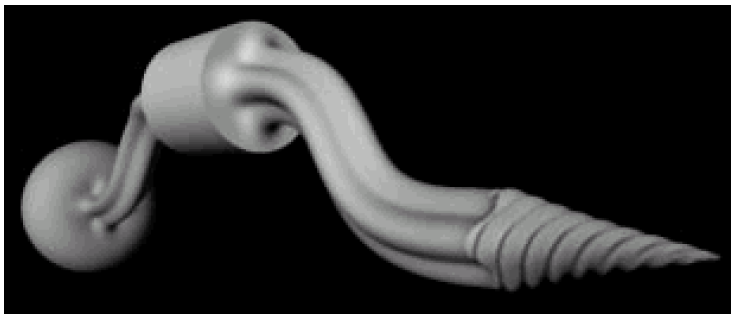


Figure 21. Deformed swept surface.
courtesy Benoit Crespin, Carole Blanc, and Christophe Schlick

V B. Patches

There are two principal means to define algebraic objects more complex than low-order surfaces. One is the use of higher order algebraic surfaces, which are difficult to design because the relation between shape and polynomial coefficients is not readily perceived. This has prompted the use of piecewise algebraic surfaces, also known as semi-algebraic sets or implicit patches. Each surface piece is low order and spans a single (usually tetrahedral) cell within a spatial partitioning.

Patch shape is typically specified by a grid of control points that deforms the enclosed space according to a Bernstein polynomial (although other bases, such as the Bspline, can be used) (73). This method is central to free-form deformation, a technique to manipulate solid models and algebraic surfaces (and widely applied to other geometric objects, such as curves, patches, and polygonal meshes) (83).

As with parametric patches, the algebraic patches must be carefully joined to maintain geometric continuity along their boundaries. Depending on the application, the cells can be recursively subdivided to adapt to local curvature while maintaining C^1 or C^2 continuity across the patch boundaries. (84) provides a method to generate implicit patches from a polygonal mesh. At each mesh location the derivatives of the patches are averaged to improve continuity along the seams.

Algebraic patches provide a compact and highly continuous surface representation; they are reviewed in (56).

V C. Procedural Methods

As observed in (25), f may be procedural, *i.e.*, any arbitrary process that computes a real value given a point in space. The process may use mathematical functions, conditionals, tables, randomness, etc. The procedurally defined `hypertexture' (85) is an implicit surface derived from stochastically varied densities within a volume. Iterative and fractal surfaces may also be procedurally implicit. Procedural methods are not generally expressible in closed form, however, and so cannot be understood with analytic geometry. Further, their interactive specification is not well understood, with few examples (86, 87).

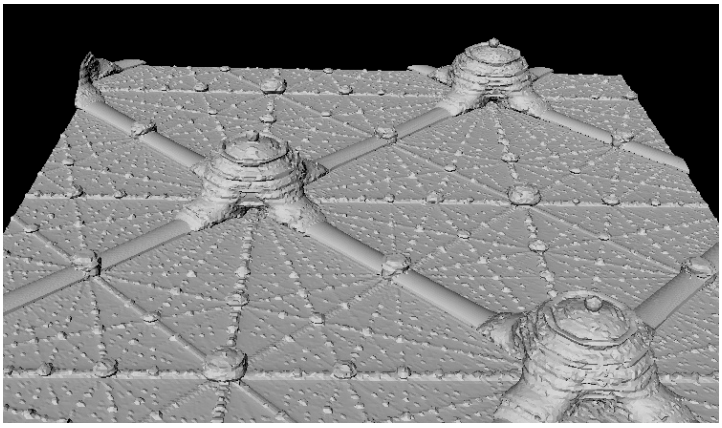


Figure 22. Iteratively computed slice through quadratic Julia sets.
courtesy John Hart and Greg Turk

Voxels may be employed as a procedural design method. For example, accumulation modeling disperses values along some path, iteratively to neighboring array elements (88, 89, 90). This can be computationally demanding, but allows the simulation of developmental processes. Other uses include smoothing (91; 79, chapter 7; 92) and volume-based metamorphosis (93; 94).

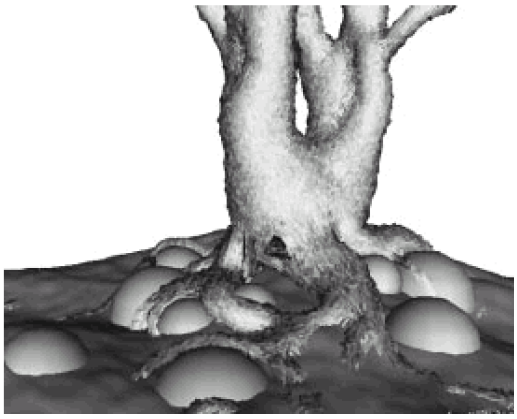


Figure 23. Voxel-based model of tree roots and obstacles.
courtesy Ned Greene

As with any sampling process, the Sampling Theorem (see 95) requires that each voxel represent a filtered volume (*i.e.*, a weighted average of the neighborhood surrounding the sample point); otherwise, aliasing may result (96).

V D. Skeletal Methods

The skeleton, a standard representation, may be used as a procedural, implicit design element. It typically consists of a hierarchical set of 'limbs' that each generate an implicit primitive. Each limb may support one or more descendent limbs. The relationship between child and parent limbs is usually given as an affine transformation that specifies size and orientation (97; 98).

With modern input and display devices, it is feasible to manipulate and display in real-time complex skeletons and associated implicit primitives (44; 99). Shapes intermediate to key poses typically rely on rigid body rotation at the skeletal joints, as other schemes appear unnatural.

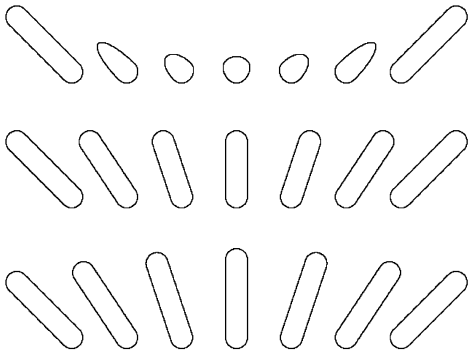


Figure 24. Interpolations of implicit contours generated by skeletons S_1 and S_2.
top to bottom: algebraic interpolation, interpolated segment endpoints, segment angle

f may be given as the union of primitives or, for a more smooth result, as a blend. For example, given two primitives $P_1(\boldsymbol{p})$ and $P_2(\boldsymbol{p})$, $f = B(P_1, P_2) = 0$, where *B* is some blend function, such as $1-(1-P_1)^2-(1-P_2)^2$. Various blends are examined in (51; 100; 101); (102) provides a survey.

The sum of the convolution of each skeletal limb produces rounds along convex portions of the skeleton and fillets along concave portions, and supports complex branching (103) .

Figure 25. A surface defined by convolution.

VI. Visualization

Implicit surface definitions may be converted to polygonal meshes and visualized ('rendered') with general-purpose polygon renderers. Methods not requiring an intermediate surface representation include rasterization, ray-tracing, line-drawing, and particle display.

Incremental scan-line methods ('rasterization') may be applied to quadric surfaces (104; 105; 77; 106; 107; 108; 109). Rasterization of implicitly defined curves is discussed in (110).

Implicit surfaces may be ray-traced directly from *f*, assuming the intersection of each ray with the implicit surface is computable. General methods include a spatial partitioning (such as an octree)

that reduces the problem to the intersection of a ray with terminal cells of the partitioning (111). For each terminal cell, assuming there is a positive intersection and a negative intersection with the ray, the ray-surface intersection may be computed by binary sectioning, *regula falsi*, or some other technique; for low-degree algebraic surfaces, analytic methods may be used (111). Performance may be improved by application of interval analysis (112), with specific optimizations reported for metaballs (113). Algebraic surfaces may be ray-traced by symbolic methods that are particularly efficient and accurate (114). Implicit surfaces generally appear simpler to ray-trace than parametric patches (115).

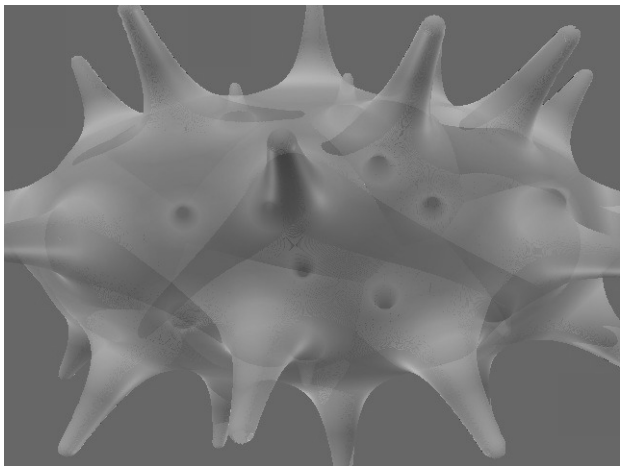


Figure 26. A ray-traced implicit surface.

An alternative to shaded imagery is the contour-line drawing, accomplished by intersecting an implicit surface with a series of planes, each perpendicular to the line of sight and receding from the viewpoint (25). For each plane the zero-contour is drawn, excepting those parts obscured by previously drawn contours. Contour-line drawings are particularly useful for engineering applications (116). Like ray-tracing, the technique may be optimized using a spatial partitioning.

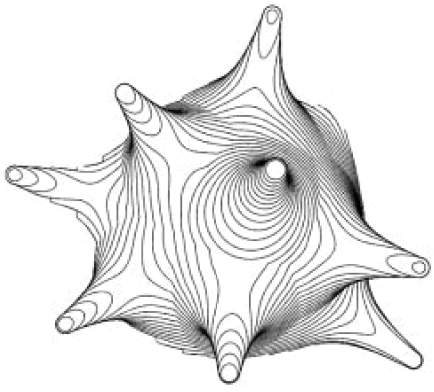


Figure 27. A contour line drawing.

Particle display is a method for rapid visualization of representative points along the implicit surface (44). Particles also serve as control points for design and modification (99).

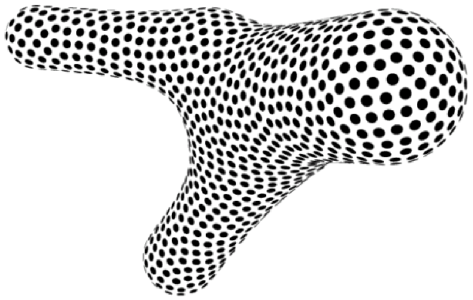


Figure 28. Particles distributed by mutual repulsion.
courtesy Paul Heckbert and Andrew Witkin

Other ad hoc methods, such as the display of a planar 'slice' of *f*, are reviewed in (117; 44).

When solid material is of interest, an entire volume may be rendered using volume visualization, which models light attenuation through an optical medium and is generally applied to an array of samples of f (118; 119; 120; 121), with variations concerned with performance (122; 123; 124) and ray-tracing (125). Some methods employ partial shading of estimated surfaces within the volume (118; 126). Volume visualization is normally an orthographic projection of a regular, affinely transformed grid; alternative methods are considered in (127; 128).

Photorealistic image generation requires considerable computation, encouraging the use of efficient structures such as hierarchical detail. Hierarchical detail may be added to implicit surfaces by applying minute geometric features to simpler, underlying shapes (53); this may be implemented as a displacement (129; 82), as an operation within a procedural definition, or as functional composition from $\Re^3$ to $\Re^3$.

Figure 29. Detail as implicit composition (color-mapped to emphasize contours)**.**
left: $a(\boldsymbol{p})$ is a vertical gradient; middle: $b(a(\boldsymbol{p}))$ is a smooth modification to a
right: $f(\boldsymbol{p}) = c(b(a(\boldsymbol{p})))$ provides fine detail

Alternatively, detail may be added to a polygonized implicit surface via texture synthesis (130; 131) or via general texture parameterization, *i.e.*, *uv*-coordinate assignment (132; 133; 134). A parameterization free of singularities does not necessarily exist for a given surface, and this can complicate the creation of a realistic texture mapping. A general approach to provide *uv*-parameterizations for implicit surfaces is presented in (135), which observes that the implicit representation overcomes several limitations of patch-based interactive texture painting.

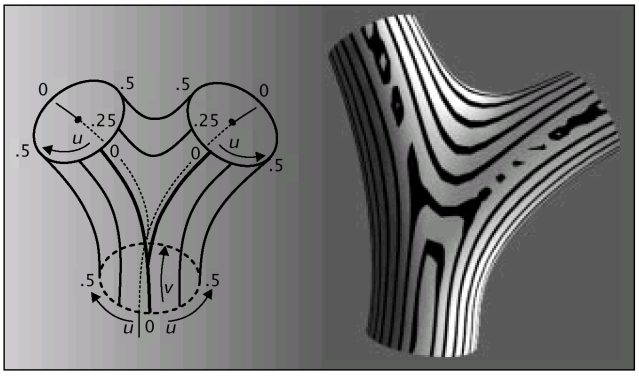


Figure 30. UV-coordinate assignment governed by underlying skeleton.
left: texture coordinates, right: rendered surface

Solid texture (see (136; 137)) may also be applied to implicit surfaces (138). Such texture relies on surface position not parameterization, and creates the appearance of an object carved from, rather than covered by, a material.

VII. Other Applications

The most common techniques for implicit modeling presently include algebraic surfaces, implicit patches, CSG, and sums of skeletal primitives.

Implicit methods are also useful in surface reconstruction from unorganized surface points through the use of algebraic sums (139; 140), the use of implicitly defined distance to planes tangential to the surface points (141), or the use of algebraic patches (142; 74; 143). Techniques to extract an implicit surface from laser range data are given in (144; 145). Other applications include the construction of the medial axis (146).

Outstanding issues related to implicit surface design and visualization include improved parameterization and texturing, hardware support for visualization, and improved control of shape.

Implicit surfaces constitute an evolving subject of considerable breadth and depth. This brief review cannot fully cover the subject but has, hopefully, provided insight into the properties and applications of implicit surfaces.

VIII. References

(1) C.Hoffmann, Geometric and Solid Modeling: an Introduction, San Francisco, Morgan Kaufmann, 1989.

(2) J.Mayer, Algebraic Topology, Prentice-Hall, 1972.

(3) V.Guillemin, A.Pollack, Differential Topology, Prentice-Hall, 1974.

(4) M.Spivak, Calculus on Manifolds, Addison-Wesley, 1965.

(5) J.Bruce, P.Giblin, Curves and Singularities, Cambridge University Press, 1992.

(6) H.Whitney, Elementary Structure of Real Algebraic Varieties, Annals of Mathematics 66:3, pp. 545-556, 1957.

(7) C.Hoffmann, Implicit Curves and Surfaces in Computer Aided Geometric Design, IEEE Computer Graphics and Applications, 13:1, Jan. 1993, pp. 79-88.

(8) A.Kalvin, A Survey of Algorithms for Constructing Surfaces from 3D Volume Data, IBM Research Report RC 17600, Jan. 1992.

(9) P.Ning, J.Bloomenthal, An Evaluation of Implicit Surface Tilers, IEEE Computer Graphics and Applications, 13:6, Nov. 1993, pp. 33-41.

(10) M.Schmidt, Cutting Cubes: Visualizing Implicit Surfaces by Adaptive Polygonization, The Visual Computer, 10:2, 1993, pp. 101-115.

(11) W.Schroeder, K.Martin, W.Lorensen, The Visualization Toolkit, An Object-Oriented Approach To 3D Graphics, Prentice Hall, 1996.

(12) M.Mäntylä, An Introduction to Solid Modeling, Computer Science Press, 1988.

(13) J.Bloomenthal, Polygonization of Implicit Surfaces, Computer Aided Geometric Design, 5:4, Nov. 1988, pp. 341-355.

(14) E.Allgower, K.Georg, Numerical Continuation Methods, an Introduction, Springer-Verlag, 1990 (series in Computational Mathematics, v. 13).

(15) W.Lorensen, H.Cline, Marching Cubes: a High Resolution 3D Surface Construction Algorithm, Computer Graphics, 21:4, 1987, pp. 163-169 (Proc. SIGGRAPH 87).

(16) J.Wilhelms, A.van Gelder, Octrees for Faster Isosurface Generation, ACM Trans. on Graphics, 11:3, pp. 210-227, Jul. 1992.

(17) M.Laszlo, Fast Generation and Display of Iso-Surface Wireframes, Computer Vision Graphics and Image Processing, 54:6, pp. 473-483, 1992.

(18) G.Tindle, Fermi Surface Display, Computers and Graphics, 10:1, 1986, pp 77-79.

(19) J.Ambrosiano, S.Brandon, R.Löhner, C.DeVore, Electromagnetics via the Taylor-Galerkin Finite Element Method on Unstructured Grids, J. Computational Physics, v. 110, pp. 310-319, 1994.

(20) I.Fujii, Y.Morimoto Y.Higuchi, N.Yasuoka, A Polypeptide Model-building Program for a Graphics Workstation, J. Molecular Graphics, 10:3, Sept. 1992, pp 185-189.

(21) A.Kalvin, Segmentation and Surface-based Modeling of Objects in 3D Biomedical Images, Courant Inst. of Math. Sciences, Ph.D. dissertation, 1991.

(22) A.Koide, A.Doi, K.Kajioka, Polyhedral Approximation Approach to Molecular Orbital Graphics, J. Molecular Graphics, 4:3, 1986.

(23) G.Purvis and C.Culberson, On the Graphical Display of Molecular Electrostatic Force-Fields and Gradients of the Electron Density, 1985, pp. 317-332 (North American Treaty Organization Conference).

(24) T.Itoh, K.Koyamada, Automatic Isosurface Propagation Using an Extrema Graph and Sorted Boundary Cell Lists, IEEE Trans. Visualization and Computer Graphics, 1:4, 1995, pp. 319-327.

(25) A.Ricci, A Constructive Geometry for Computer Graphics, The Computer J., 16:2, May 1973, pp. 157-160.

(26) D.Meagher, Geometric Modeling Using Octree Encoding, Computer Graphics and Image Processing 19:2, June 1982.

(27) H.Samet, Design and Analysis of Spatia Data structures, Addison-Wesley, 1990.

(28) D.Moore, Subdividing Simplices, in D.Kirk (ed.), Graphics Gems III, Academic Press, 1992.

(29) K.Suffern, Recursive Space Subdivision Techniques for Rendering Implicit Surfaces, Proc. Australian Conf. on Computer Graphics, 1989, pp 239-250.

(30) J.Snyder, Interval Analysis for Computer Graphics, Computer Graphics, 26:2, 1992, pp. 121-130 (Proc. SIGGRAPH 92).

(31) T.Duff, Interval Arithmetic and recursive Subdivision for Implicit functions and Constructive Solid Geometry, Computer Graphics, 26:2, 1992, pp. 131-138 (Proc. SIGGRAPH 1992).

(32) B.Von Herzen, A.Barr, Accurate Triangulations of Deformed, Intersecting Surfaces, Computer Graphics 21:4, 1987 (Proc. SIGGRAPH 87).

(33) D.Kalra, A.Barr, Guaranteed Ray Intersections with Implicit Surfaces, Computer Graphics, 23:4, 1989, pp. 297-306 (Proc. SIGGRAPH 89).

(34) J.Hart, Sphere-Tracing: a Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces, The Visual Computer, 12:10, 1997, pp. 527-545.

(35) H.Coxeter, Regular Polytopes. Macmillan, 1963.

(36) H.Freudenthal, Simplizialzerlegungen von Beschrankter Flachheit, Annals of Mathematics, v. 43, 1942, pp. 580-582.

(37) E.Allgower, P.Schmidt, An Algorithm for Piecewise-Linear Approximation of an Implicitly Defined Manifold, SIAM J. Numerical Analysis, 22:2, pp. 322-346, 1985.

(38) G.Wyvill, C.McPheeters, B.Wyvill, Data Structure for Soft Objects, The Visual Computer, 2:4, Aug. 1986, pp. 227-234.

(39) W.Rheinboldt, On the Computation of Multi-Dimensional Solution Manifolds of Parameterized Equations, Numerische Mathematik, v. 53, pp. 165-182, 1988.

(40) M.Hall, J.Warren, Adaptive Polygonalization of Implicitly Defined Surfaces, IEEE Computer Graphics and Applications, 10:6, Nov. 1990 pp. 33-42.

(41) E.Allgower, S.Gnutzmann, Simplicial Pivoting for Mesh Generation of Implicitly Defined Surfaces, Computer-Aided Geometric Design, 8:4, Oct. 1991, pp. 305-25.

(42) L.Velho, Simple and Efficient Polygonization of Implicit Surfaces, J. Graphics Tools, 1:1, 1996, pp. 5-24.

(43) L.deFigueiredo, J.deMiranda Gomes, D.Terzopoulos, L.Velho, Physically-Based Methods for Polygonization of Implicit Surfaces, Proc.. Graphics Interface 92, pp. 250-257.

(44) J.Bloomenthal, B.Wyvill, Interactive Techniques for Implicit Modeling, Computer Graphics, 24:2, Mar. 1990, pp. 109-116.

(45) L.deFigueiredo, J.deMiranda Gomes, Sampling Implicit Objects with Physically-based Particle Systems, Computers and Graphics, 20:3, 1996, pp. 363-375.

(46) B.Stander, J.Hart. Guaranteeing the Topology of an Implicit Surface Polygonization. Computer Graphics, 31:4, 1997, pp. 279-286 (Proc. SIGGRAPH 97).

(47) J.Rossignac, M.O'Connor, SGC: a Dimension-Independent Model for Pointsets with Internal Structures and Incomplete Boundaries, Geometric Modeling for Product Engineering, Elsevier Science, 1990.

(48) J.Bloomenthal, K.Ferguson, Polygonization of Non-Manifold Surfaces, Computer Graphics, pp. 309-316 (Proc. SIGGRAPH 95).

(49) B.Wyvill, K.van Overveld, Polygonization of Implicit Surfaces with Constructive Solid Geometry, J. Shape Modeling, 2:4, World Scientific Publishing, 1997, pp. 257-273.

(50) G.Farin, Curves and Surfaces for Computer-Aided Geometric Design: A Practical Guide, Academic Press, 1993.

(51) A.Rockwood, The Displacement Method for Implicit Blending Surfaces in Solid Models. ACM ACM Trans. on Graphics, 8:4, Oct. 1989, pp. 279-297.

(52) P.Charrot, J.Gregory, A Pentagonal Surface Patch for Computer-Aided Geometric Design, Computer-Aided Geometric Design, 1:1, 1984, pp. 87-94.

(53) S.Sclaroff, A.Pentland, Generalized Implicit Functions for Computer Graphics, Computer Graphics, 25:4, 1991, pp. 247-250 (Proc. SIGGRAPH 91).

(54) T.Sederberg, Algebraic Geometry for Surface and Solid Modeling, in Geometric Modeling: Algorithms and Trends, G. Farin (ed.), SIAM Press, 1987.

(55) T.Sederberg, Implicit and Parametric Curves and Surface for Computer-Aided Geometric Design, Ph.D. dissertation, Purdue University, 1983.

(56) C.Bajaj, The Emergence of Algebraic Curves and Surfaces in Geometric Design, in Directions in Geometric Computing, R. Martin (ed.), pp. 1-29, Information Geometers Press, UK, 1993.

(57) B.van der Waerden, Modern Algebra, v. 1 and 2, Frederick Publishing, 1950.

(58) D.Kapur, Y.Lakshman, Elimination Methods: An Introduction, in Donald, Kapur, Mundy (eds.), Symbolic and Numerical Computation: An Integration, Academic Press, 1992.

(59) T.Sederberg, F.Chen, Implicitization using Moving Curves and Surfaces, Computer Graphics Proc., 1995, pp. 301-308 (Proc. SIGGRAPH 95).

(60) T.Sederberg, J.Snively, Parameterizing Cubic Algebraic Surfaces, in The Mathematics of Surfaces II, R.Martin (ed.), Oxford University Press, 1987, pp. 299--320.

(61) G.Salmon, A Treatise on the Algebraic Geometry of Three Dimensions, 1:2, R.Rogers (ed.), Chelsea Publishing, 1914.

(62) A.Requicha, Representations for Rigid Solids: Theory, Methods, and Systems, Computing Surveys 12:4, Dec. 1980, pp. 437-464.

(63) G.Tavares, J.deGomes, Concordance Operations for Implicitly-Defined Manifolds, Proc. SIAM Conference on Geometric Design, 1989, SIAM Press.

(64) V.Shapiro, Real Functions for Representation of Rigid Solids, Cornell University technical report TR 91-1245, Nov. 1991.

(65) A.Pasko, V.Adzhiev, A.Sourin, V.Savchenko, Function Representation in Geometric Modeling: Concepts, Implementation and Applications, The Visual Computer, 11: 8, 1995, pp. 429-446.

(66) M.Henle, A Combinatorial Introduction to Topology, W.H. Freeman, 1979.

(67) B.Baumgart, Geometric Modeling for Computer Vision, Ph.D. dissertation, Stanford University, 1974.

(68) H.Chiyokura, Solid Modeling with Designbase, Addison-Wesley, 1988.

(69) J.Rossignac, A.Requicha, Constructive Non-Regularized Geometry, in Beyond Solid Modeling, special edition of Computer Aided Design, 1991.

(70) K.Weiler, Topologoical Structures for Geometric Modeling, Ph.D. dissertation, Rensselaer Polytechnic Institute, 1986.

(71) A.Barr, Superquadrics and Angle-Preserving Transformations, IEEE Computer Graphics and Applications, 1:1, Jan. 1981.

(72) D.Saupe, M.Ruhl, Animation of Algebraic Surfaces, Proc. Visual Mathematics, Berlin, June 1995 (Int'l Workshop on Visualization and Mathematics).

(73) T.Sederberg, Piecewise Algebraic Surface Patches, Computer Aided Geometric Design, v. 2, 1985, pp. 53-59.

(74) C.Bajaj, Surface Fitting with Implicit Algebraic Surface Patches, in Topics in Surface Modeling, H. Hagen (ed.), pp. 23-52, SIAM Publications, 1992.

(75) O.Zariski, Algebraic Surfaces, v. 4, Ergebnisse der Mathematik und ihre Grenzgebrete, 1935.

(76) C.Bajaj, I.Ihm, Algebraic Surface Design with Hermite Interpolation, ACM Proc. on Graphics, 11:1, pp. 61-91, Jan. 1992.

(77) J.Blinn, A Generalization of Algebraic Surface Drawing, ACM Trans. Graphics, 1:3, Jul. 1982, pp. 135-256.

(78) H.Nishimura, M.Hirai, T.Kawai, T.Kawata, I.Shirakawa, K.Omura, Object Modeling by Distribution Function and a Method of Image Generation, Trans. Inst. Electronics and Communication Engineers of Japan, 1985, J68-D:4, pp. 718-725 (in Japanese).

(79) J.Bloomenthal (ed.), Introduction to Implicit Surfaces, Morgan Kaufmann, 1997.

(80) A.Barr, Global and Local Deformations of Solid Primitives, Computer Graphics, 18:3, pp. 21-30 (Proc. SIGGRAPH 84).

(81) B.Crespin, C.Blanc, C.Schlick, Implicit Sweep Objects, Computer Graphics Forum, 15:3, 1996, pp. 165-174.

(82) H.Pedersen, Displacement Mapping Using Flow Fields, Computer Graphics, 1994, pp. 279-286 (Proc. SIGGRAPH 94).

(83) T.Sederberg, S.Parry, Free-Form Deformation of Solid Geometric Models, Computer Graphics, 20:4, 1986, pp. 151-160 (Proc. SIGGRAPH 86).

(84) J.Warren, Free-Form Blending: a Technique for Creating Piecewise Implicit Surfaces, Topics in Surface Modeling, H.Hagen (ed.), SIAM Press, 1992, pp. 3-21.

(85) K.Perlin, E.Hoffert, Hypertexture, Computer Graphics, 23:4, 1989, pp. 253-262 (Proc. SIGGRAPH 89).

(86) G.Nelson, Juno, a Constraint-Based Graphics System, Computer Graphics, 19:3, 1985, pp. 235-243 (Proc. SIGGRAPH 85).

(87) D.Fowler, P.Prusinkiewicz, J.Battjes, A Collision-Based Model of Spiral Phyllotaxis, Computer Graphics, 26:2, 1992, pp. 361-368 (Proc. SIGGRAPH 92).

(88) A.Smith, Plants, Fractals, and Formal Languages, Computer Graphics, 18:3, 1984 (Proc. SIGGRAPH 84).

(89) L.Williams, 3D Paint, Computer Graphics, 24:2, 1990, pp. 225-233 (Symp. Interactive 3D Computer Graphics).

(90) N.Greene, Voxel Space Automata: Modeling with Stochastic Growth Processes in Voxel Space, Computer Graphics, 23:4, 1989, pp. 175-184 (Proc. SIGGRAPH 89).

(91) T.Galyean, J.Hughes, Sculpting: An Interactive Volumetric Modeling Technique, Computer Graphics, 25:4 1991, pp. 267-274 (Proc. SIGGRAPH 91).

(92) J.Wilhelms, Animals with Anatomy, IEEE Computer Graphics and Applications, 17:3, May 1997.

(93) J.Hughes, Scheduled Fourier Volume Morphing, Computer Graphics, 26:2, 1992, pp. 43-46 (Proc. SIGGRAPH 92).

(94) A.Lerios, C.Garfinkle, M.Levoy, Feature-Based Volume Metamorphosis, Computer Graphics, 1995, pp. 449-456 (Proc. SIGGRAPH 95).

(95) A.Oppenheim, R.Schafer, Digital Signal Processing, Prentice Hall, 1975.

(96) S.Wang, A.Kaufman, Volume-Sampled 3D Modeling, IEEE Computer Graphics and Applications, Sept. 1994, 14:5, pp. 26-32.

(97) T.O'Donnell, A.Olson, GRAMPS: A Grapics Language Interpreter for Real-Time, Interactive, Three-Dimensional Picture Editing and Animation, Computer Graphics, 15:3, 1981, pp. 133-142 (Proc. SIGGRAPH 81).

(98) W.Reeves, E.Osby, The MENV Modeling and Animation Environment, J. Visualization and Computer Animation, 1:1, 1990.

(99) A.Witkin, P.Heckbert, Using Particles to Sample and Control Implicit Surfaces, Computer Graphics Proc., 1994, pp. 269-278 (Proc. SIGGRAPH 94).

(100) J.Warren, Blending Algebraic Surfaces, ACM Transactions on Graphics 8:4, Oct. 1989, pp. 263-278.

(101) C.Hoffmann, J.Hopcroft, The Potential Method for Blending Surfaces and Corners, in Geometric Modeling, G.Farin (ed.), SIAM, 1987.

(102) John Woodwark, Blends in Geometric Modeling, in The Mathematics of Surfaces, 1987, R.Martin (ed.), Clarendon Press, pp. 255-297.

(103) J.Bloomenthal, K.Shoemake, Convolution Surfaces, Computer Graphics 25:4, 1991 (Proc. SIGGRAPH), pp. 251-256.

(104) R.Goldstein, R.Nagel, 3D Visual Simulation, Simulation, 16:1, 1971, pp. 25-31.

(105) J.Levin, A Parametric Algorithm for Drawing Pictures of Solid Objects Bounded by Quadric Surfaces, C. ACM, 19:11, pp. 555-563, Nov. 1976.

(106) P.Mittelman, Computer Graphics at MAGI, Computer Graphics 83, Online Publications, UK, pp. 291-301.

(107) J.Davis, R.Nagel, W.Guber, A Model Making and Display Technique for 3-D Pictures, Proc. 7th Annual Meeting of UAIDE, 1968, pp. 47-72.

(108) W.Bronsvoort, Direct Display Algorithms for Solid Modeling, Jun. 1990, Ph.D. dissertation, Delft University of Technology.

(109) R.van Kleij, Display of Solid Models with Quadratic Surfaces, Ph.D. dissertation, Delft University of Technology, 1993.

(110) G.Taubin, Distance Approximations for Rasterizing Implicit Curves, ACM Trans. on Graphics, 13:1, Jan. 1994, pp. 3-42.

(111) S.Roth, Ray Casting as a Method for Solid Modeling. Computer Graphics and Image Processing, 18:2, Feb. 1982, pp. 109-144.

(112) D.Mitchell, Robust Ray Intersection with Interval Arithmetic, Proc. Graphics Interface 90, pp. 68-74, May 1990.

(113) T.Nishita, E.Nakamae, A Method for Displaying Metaballs by Using Bezier Clipping, Computer Graphics Forum, v. 13 (Proc. Eurographics 94), Sept. 1994, pp. C271-C280.

(114) P.Hanrahan, Ray Tracing Algebraic Surfaces, Computer Graphics, 17:3, 1983, pp. 83-90 (Proc. SIGGRAPH 83).

(115) J.Kajiya, Ray Tracing Parametric Patches, Computer Graphics, 16:3, 1982, pp. 245-254 (Proc. SIGGRAPH 82).

(116) A.Forrest, On the Rendering of Surfaces, Computer Graphics, 13:2, 1979, pp. 253-259 (Proc. SIGGRAPH 79).

(117) G.Nielson, B.Hamann, The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes, Proc. Visualization 91, Oct. 1991, pp. 83-91, G.Nielson, L.Rosenblum (eds.).

(118) R.Drebin, L.Carpenter, P.Hanrahan, Volume Rendering, Computer Graphics, 1988, pp. 65-74 (Proc. SIGGRAPH 88).

(119) C.Upson, M.Keeler, V-Buffer: Visible Volume Rendering, Computer Graphics, 22:4, 1988, pp. 59-64 (Proc. SIGGRAPH 88).

(120) P.Sabella, A Rendering Algorithm for Visualizing 3D Scalar Fields, Computer Graphics, 22:4, 1988, pp. 51-58 (Proc. SIGGRAPH 88).

(121) M.Levoy, Display of Surfaces from Volume Data, IEEE Computer Graphics and Applications, 8:3, 1988, pp. 29-37.

(122) N.Max, P.Hanrahan, R.Crawfis, Area and Volume Coherence for Efficient Visualization of 3D Scalar Functions, Computer Graphics, 24:5, Nov. 1990, pp. 27-33, (Proc. Workshop on Volume Visualization, Dec. 1990).

(123) P.Hanrahan, Three-Pass Affine Transforms for Volume Rendering), Computer Graphics, 24:5, pp. 71-78, 1990 (Proc. Workshop on Volume Visualization 1990).

(124) L.Westover, Interactive Volume Rendering, Proc. Chapel Hill Workshop on Volume Visualization, May 1989, pp. 9-16.

(125) J.Kajiya, B.Von Herzen, Ray Tracing Volume Densities, Computer Graphics, 18:3, 1984, pp. 165-174 (Proc. SIGGRAPH 84).

(126) R.Gallagher, J.Nagtegaal, An Efficient 3D Visualization Technique for Finite Element Models and Other Coarse Volumes, Computer Graphics, 23:4, 1989, pp. 185-194 (Proc. SIGGRAPH 89).

(127) M.Garrity, Raytracing Irregular Volume Data, Computer Graphics, 24:5, pp. 35-40, Nov. 1990.

(128) K.Novins, F.Sillion, D.Greenberg, An Efficient Method for Volume Rendering using Perspective Projection, Computer Graphics, 24:5, Nov. 1990, pp. 95-102 (Proc. Workshop on Volume Visualization, Dec. 1990).

(129) R.Cook, Shade Trees, Computer Graphics, 18:3, 1984, pp. 223-230 (Proc. SIGGRAPH 84).

(130) G.Turk, Generating Textures on Arbitrary Surfaces using Reaction Diffusion, Computer Graphics, 25:4, 1991, pp. 289-298 (Proc. SIGGRAPH 91).

(131) A.Witkin, M.Kass, Reaction-Diffusion Textures, Computer Graphics, 25:4, 1991, pp. 299-308 (Proc. SIGGRAPH 91).

(132) A.Gagalowicz, S.deMa, Model Driven Synthesis of Natural Textures for 3D Scenes, Eurographics 85, C.Vandoni (ed.), Elsevier Science, 1985.

(133) G.Turk, Re-Tiling Polygonal Surfaces, Computer Graphics, 26:2, p. 55-64 (Proc. SIGGRAPH 92).

(134) K.Opitz, H.Pottmann, Computing Shortest Paths on Polyhedra: Applications in Geometric Modeling and Scientific Visualization, Int'l J. Computational Geometry and Applications, 4:2, 1994, pp. 165-178.

(135) H.Pedersen, Decorating Implicit Surfaces, Computer Graphics, 1995, pp. 291-300 (Proc. SIGGRAPH 95).

(136) K.Perlin, An Image Synthesizer, Computer Graphics, 19:3, 1985, pp. 287-296 (Proc. SIGGRAPH 85).

(137) D.Peachey, Solid Texturing of Complex Surfaces, Computer Graphics, 19:3, 1985, pp. 276-286 (Proc. SIGGRAPH 85).

(138) G.Wyvill, C.McPheeters, B.Wyvill, Solid Texturing of Soft Objects, IEEE Computer Graphics and Applications, 7:12, Dec. 1987, pp. 20-26.

(139) S.Muraki, Volumetric Shape Description of Range Data Using Blobby Model, Computer Graphics, 25:4, 1991, pp. 227-235 (Proc. SIGGRAPH 91).

(140) E.Bittar, N.Tsingos, M-P.Gascuel, Automatic Reconstruction of Unstructured 3D Data: Combining Medial Axis and Implicit Surfaces, Computer Graphics Forum, 14:3, Netherlands, 1995.

(141) H.Hoppe, T.DeRose, T.DuChamp, J.McDonald, W.Stuetzle, Surface Reconstruction from Unorganized Points, Computer Graphics, 26:2, 1992, pp. 71-78 (Proc. SIGGRAPH 92).

(142) D.Moore, J.Warren, Adaptive Mesh Generation II: Packing Solids, Rice University technical report TR90-139, Mar. 1991.

(143) B.Bailey, C.Bajaj, M.Fields, The Vaidak Medical Imaging and Model Reconstruction Toolkit, tech. report CSD-TR-91-066, Purdue University, 1991.

(144) C.Bajaj, J.Chen, G.Xu, Free-Form Modeling with C^2 Quintic A-Patches, 4 th SIAM Conf. on Geometric Design, Sept. 1995.

(145) B.Curless, M.Levoy, A Volumetric Method for Building Complex Models from Range Images, Computer Graphics, 1996, pp. 303-312 (Proc. SIGGRAPH 96).

(146) J.Bloomenthal, C.Lim, Skeletal Methods of Shape Manipulation, Shape Modeling Int'l, Aizu-Wakamatsu, Japan, 1999.

(147) J.deGomes, L.Velho, Implicit Objects in Computer Graphics, Monografias de Matematica No. 53, Instituto de Matematica Pura e Aplicado, Rio de Janeiro, 1992.

(148) J.Foley, A.vanDam, S.Feiner, J.Hughes, Computer Graphics, Principles and Practice, Addison-Wesley, 1992.

Hand Crafted

Jules Bloomenthal
University of Calgary

Extended Abstract

In [Bloomenthal and Shoemake] *convolution surfaces*, an implicit modeling technique based upon three dimensional convolution, was applied to *skeletal primitives* (see [Bloomenthal and Wyvill]) to simulate a human arm. We present here the use of skeletal primitives and convolution surfaces for the modeling of a human hand. We begin with a review of the arm model, which consists of five 'muscles,' blended together and each represented by a planar polygon, diagrammed in Figure 1.

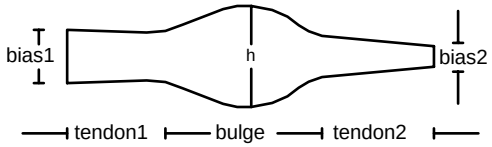


Figure 1: Individual Muscle and Parameters.

The five polygons, arranged as in Figure 2, were individually convolved with a windowed Gaussian kernel, and then summed. The surface was taken as the set {**P**} such that $F(\mathbf{P}) = \Sigma CS_i(\mathbf{P}) - c = 0$, where c is a positive constant and $CS_i(\mathbf{P})$ is the value at point **P** of the i-th convolution primitive.

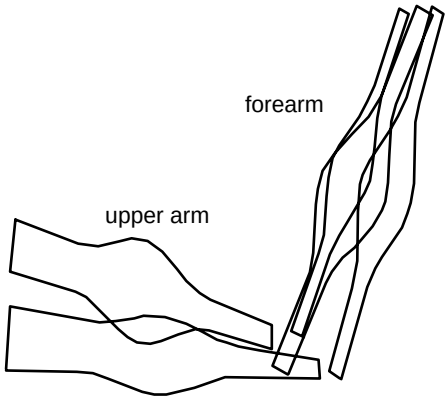


Figure 2: Arm Muscles.

Each convolution surface is evaluated independently, and, because of the superposition property of convolution, the sum of individual convolution surfaces does not produce unwanted bulging. Consider the convolution of a primitive with a filter kernel; this yields an iso-surface valued from (a normalized) 1 on the primitive to 0 when the distance to the primitive is greater than the kernel size. When the primitives are contiguous, the resulting implicit surface contains no bulge. The problem of bulging in implicit modeling is discussed in [Middleditch and Sears]; [Rockwood] presents an elegant solution that, however, requires interdependent evaluation of the primitives.

The arm is a collection of relatively amorphous muscles. We achieve greater definition in the hand by representing the bones of the palm and fingers with contiguous primitives. An additional primitive represents the muscle near the thumb, two sets of contiguous primitives represent the veins in the back of the hand, and five sets of contiguous primitives represent the tendons.

Modeling the human hand has interested numerous researchers. [Thompson *et al*] simulates the mechanical aspects of the hand by manipulating the position and orientation of bones; an overlying skin is not, however, produced. [Gourret *et al*] models the hand as a deformable surface overlying a bone structure; the surface is parametrically defined and deformed according to forces acting upon the hand. [Forsey and Bartels] introduces a sophisticated method for controlling parametric surfaces that allows the automatic creation of surfaces from articulated, contiguous skeletons; inclusion of bulges from disjoint primitives is considered in [Forsey].

Unlike implicit techniques, parametric techniques must explicitly account for surface topology. For example, as the thumb muscle contracts, the topology of the vertex network of the surface may change. Such changes are easily accomodated within an implicit modeling system. This is an issue not only for models whose shape changes, but also for the design of new models: parametric systems must carefully attend to the topology of the resulting surface.

As shown in Figure 3, the skeletal primitives of the hand include:

palm bones: a contiguous set of 15 triangles,
finger bones: a contiguous set of 48 triangles,
tendons: five sets of connected line segments, 10 in total,
veins: two sets of connected line segments, 11 in total,
muscle (adductor pollicis): one line segment,

resulting in a total of 85 convolution surface primitives.

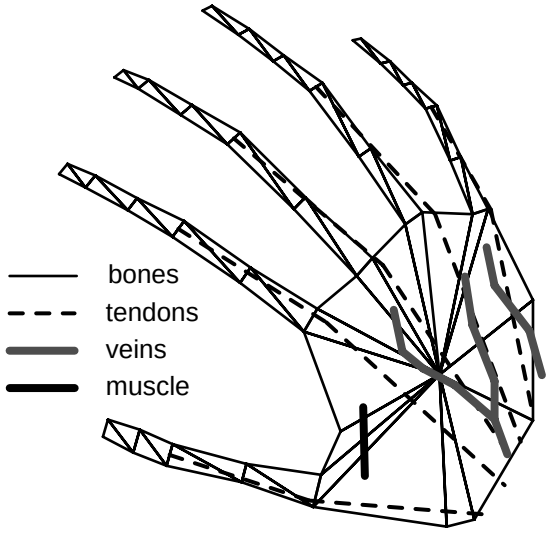


Figure 3: Hand Skeleton.

The triangles that constitute the skeleton beome oblong upon convolution. This oblong quality is only an approximation to the bone and muscle within a finger, and the smooth encasing by skin.

The user does not directly define these primitives, however; rather, each finger is specified by two Euler angles and the amount of grip at each of its two (one for the thumb) joints. Even this level of control may be unnecessarily complex; [Wilhelms] has proposed methods whereby joint angles may be derived from goal oriented specifications. Alternatively, finger parameters may be read directly from special purpose input devices.

The following page displays individual and summed convolution surfaces that constitute the hand. At the upper left are the fingers in isolation; at the upper right are the tendons in isolation. The middle left shows the palm and the middle right shows the veins and the thumb muscle. At the lower left is the combination of palm and fingers, and at the lower right is the complete hand, consisting of fingers, palm, muscle, tendons, and veins.

We hope these images demonstrate that a design environment utilizing convolution surfaces can create complex, well-behaved shapes by simply adding/deleting, positioning/orienting, and scaling/proportioning individual skeletal primitives.

We have yet to animate the hand to observe its articulated behavior. A number of issues are expected to arise, including the prevention of unwanted blending between individual convolution surfaces; for example, when two fingers approach each other, they should not blend.

Additional aspects of the model concern realism. The easily controlled blending of individual convolution surfaces facilitates the design of the macro structure of the hand. Highly realistic models also require micro details such as nails, wrinkles, and creases. We hope to investigate these details in the future.

References

Bloomenthal, J. and Shoemake, K. Convolution Surfaces. In *Computer Graphics* **25**, 4, July, 1991.

Bloomenthal, J. and Wyvill, B. Interactive Techniques for Implicit Modeling. In *Computer Graphics* **24**, 2, March, 1990.

Forsey, D. and Bartels, R. Hierarchical B-Spline Refinement. In *Computer Graphics* **22**, 4, August, 1988.

Forsey, D. A Surface Model for Skeleton-based Character Animation. Eurographics Workshop on Animation and Simulation, Vienna, Austria, September, 1991.

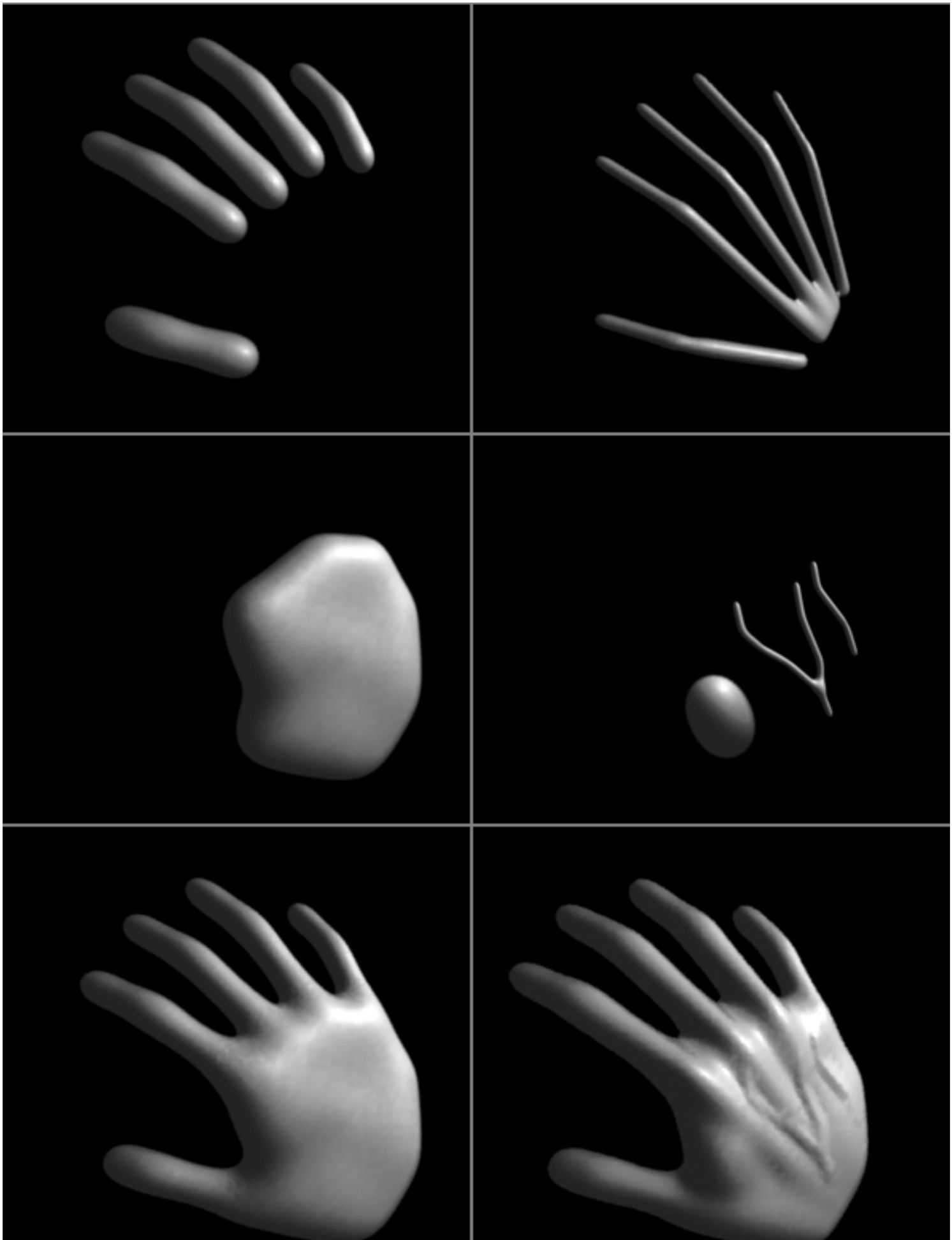
Gourret, J., Thalmann, N., and Thalmann, D. Simulation of Object and Human Skin Deformations in a Grasping Task. In *Computer Graphics*, **23**, 3, July, 1989.

Middleditch, A. and Sears, K. Blend Surfaces for Set Theoretic Volume Modeling Systems. In *Computer Graphics* **19**, 3 July 1985.

Rockwood, A. The Displacement Method for Implicit Blending Surfaces in Solid Models. *ACM Transactions on Graphics* **8**, 4, October, 1989.

Thompson, E., Buford, W., Myers, L., Giurintano, D., and Brewer J. A Hand Biomechanics Workstation. In *Computer Graphics* **22**, 4, August, 1988.

Wilhelms, J. Toward Automatic Motion Control. *IEEE Computer Graphics and Applications*, **7**, 4, 1987.



Bulge Elimination in Implicit Surface Blends

Jules Bloomenthal
George Mason University
Fairfax, Virginia USA
jbloom@vit.gmu.edu

Abstract

The relationship between surface bulges and several implicit blend techniques, particularly those based on convolution of a skeleton, is discussed. An examination of branching skeletons reveals that for two and three-dimensional skeletons, the surface will be bulge-free if elements are sufficiently large with respect to the convolution kernel.

Keywords and Phrases: implicit surface, blend, bulge, geometric modeling, skeleton, convolution.

Introduction

Implicit surface blends can be obtained by combining *distance surfaces*. These are surfaces defined by distance to 'skeletal' elements such as points, line segments, polygons, or any free-form curve, surface, or volume. To illustrate, we consider distance to the planar curve shown below. Computing this distance is demanding [Bloomenthal 1989], [Schneider 1990], and often a piecewise linear approximation is substituted for the curve.

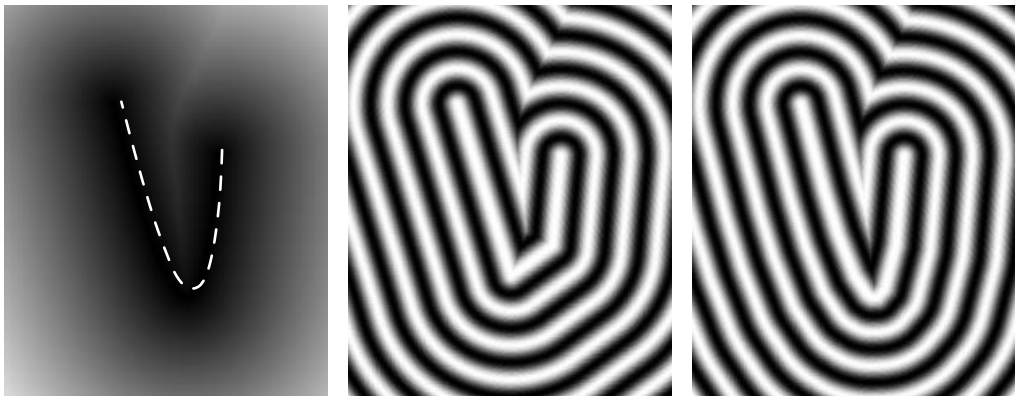


Figure 1. Distance to a Curve
left: curve shown as dashed and distance shown as greyscale intensity
middle and right: curve approximated by three and nine segments (contours emphasized)

When distance to a curve is defined for three-dimensional points, the resulting implicit surface is a generalized cylinder. For a skeleton consisting of n segments, the implicit definition for a generalized cylinder with radius r is given by:

$$(1) \qquad f(\mathbf{p}) = \min_{i}^{n} (d(\mathbf{p}, segment_i)^2)/r^2 - 1 = 0.$$

This computation requires a single comparison for each skeletal element and a single record in memory to store the smallest distance. The price for such simplicity is that the resulting surface encloses the simple union of the component primitive volumes. That is, if $\boldsymbol{p}$ is within any individual primitive $volume_i$ (defined by $segment_i$), then $d(\boldsymbol{p}, segment_i)^2/r^2 < 1$, $f(\boldsymbol{p}) < 0$, and $\boldsymbol{p}$ is interior to the solid model. Thus, *min* in equation (1) corresponds to the *union* of individual volumes.

In general, distance surfaces are rounded wherever the skeleton is convex. Where the skeleton is concave, as along the upper part of the above curve, the surface (or contour) is tangent discontinuous and exhibits a crease, which we regard as undesirable. To eliminate creases, the primitive volumes must form a *blend*, rather than a union.

The blending of primitives in the context of solid modeling has received considerable study, as seen in a variety of recent work and the survey in [Woodwark 1986]. From these sources we learn that blended surfaces (or *blends*) are used in mechanical design to reduce stress, improve air or water flow, simplify casting, and improve aesthetics. Implicit blends may be categorized as rolling-ball, volume-bounded, range-controlled, and global [*ibid*.]. The first three produce surfaces that 'heel' to parts of individual primitive surfaces when those parts are sufficiently distant from other primitives.

We briefly review some aspects of algebraic blends, illustrating with the range-controlled, super-elliptic blend introduced in [Rockwood and Owen 1985]. Two implicit primitives, P_1 and P_2, are combined according to a two-dimensional blending function, B:

$$(2) \qquad f(\boldsymbol{p}) = B(P_1, P_2) = 1 - \left[1 - \frac{P_1(\boldsymbol{p})}{r_1}\right]_+^t - \left[1 - \frac{P_2(\boldsymbol{p})}{r_2}\right]_+^t, \text{ where}$$

P_1, P_2 are algebraic distances to skeletal elements 1 and 2, usually C^1 continuous.
r_1 and r_2 are the ranges of influence for primitives P_1 and P_2,
$[x]_+$ is $max\ (0, x)$, and t is the 'thumbweight.'

The blend is called super-elliptic because the graph of $B(x, y)$ is super-elliptical (elliptical for $t = 2$). Consider an example from [Rockwood 1989] that blends the unit sphere with a cylinder (centered on the x-axis with $x \in [0, 2]$ and radius .4):

$$\text{sphere: } P_1(\boldsymbol{p}) = (\boldsymbol{p}_x^2+\boldsymbol{p}_y^2+\boldsymbol{p}_z^2)^{1/2}-1$$

$$\text{cylinder: } P_2(\boldsymbol{p}) = \begin{cases} (\boldsymbol{p}_x^2+\boldsymbol{p}_y^2+\boldsymbol{p}_z^2)^{1/2}-.4, & \boldsymbol{p}_x < 0 \\ (\boldsymbol{p}_y^2+\boldsymbol{p}_z^2)^{1/2}-.4, & 0 < \boldsymbol{p}_x < 2 \\ ((\boldsymbol{p}_x-2)^2+\boldsymbol{p}_y^2+\boldsymbol{p}_z^2)^{1/2}-.4, & \boldsymbol{p}_x > 2 \end{cases}$$

The blend $f(\boldsymbol{p}) = B(P_1(\boldsymbol{p}), P_2(\boldsymbol{p}))$, with $t = 3$, is displayed below, left, as a cross-section in the xy-plane. Although P_1 and P_2 are linear with respect to distance, B is not, which accounts for the unequal contour spacing. The rendered surface is shown below, right.

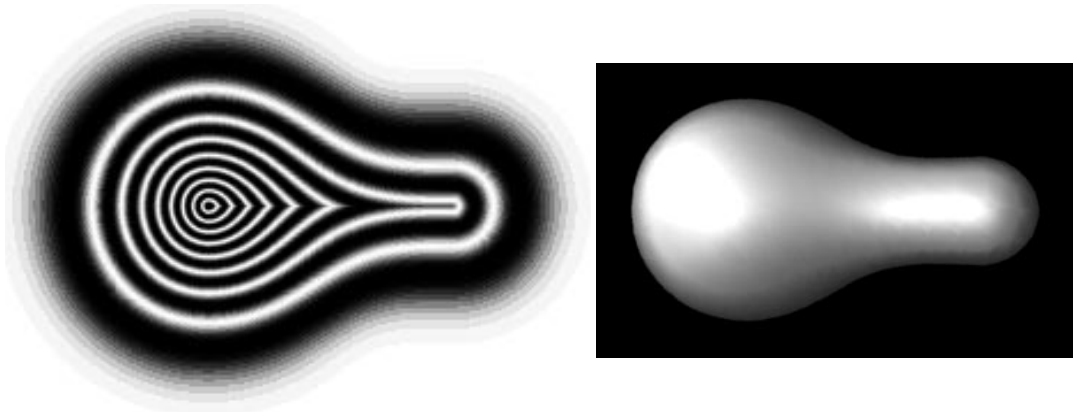


Figure 2. Super-Elliptic Blend of Sphere and Cylinder

Now consider two cylinders, one along the *x*-axis and one along the *z*-axis, as shown below, left. Contours of the blend, in the $z = 0$ plane, are shown below, right, and exhibit a bulge where the cylinders intersect. Similar artifacts are visible in [Hoffmann and Hopcroft 1985] and [Middleditch and Sears 1985].

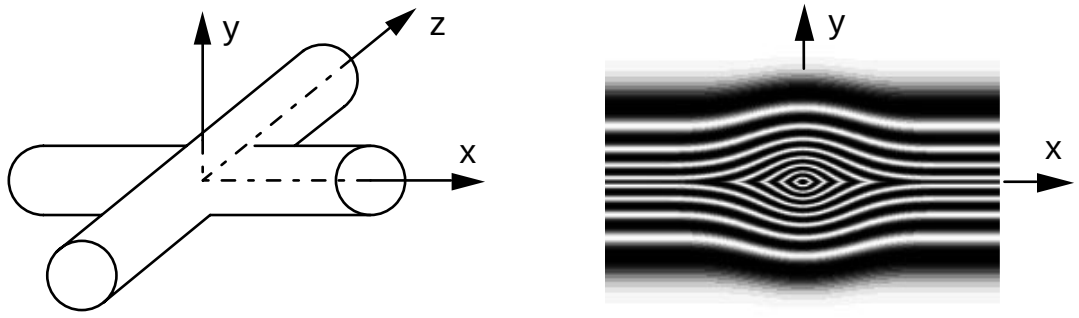


Figure 3. Super-Elliptic Blend of Two Cylinders

We have not encountered a formal definition of 'bulge,' and, so, we propose the following. A 'surface bulge' has a cross-section that exhibits negative, then positive, then negative curvature with respect to an underlying skeleton. For the super-elliptic blend, a bulge is to be expected, as those primitive values P_1 and P_2 that satisfy $B(P_1, P_2) = 0$ do not sum to a constant. To compensate, a modification to the blend is discussed in [Rockwood 1989] whereby the range of a primitive is diminished according to the angle θ between the gradients of the two primitives at a point $\boldsymbol{p}$:

$$(3) \qquad B(P_1, P_2) = 1 - \left[1 - \frac{P_1(\boldsymbol{p})}{r_1(1 - \cos\theta)}\right]_+^t - \left[1 - \frac{P_2(\boldsymbol{p})}{r_2(1 - \cos\theta)}\right]_+^t$$

For $\theta = 0$ the range is fully diminished and the simple union of the primitives results; for the concave condition $\theta = 90^o$, the range is undiminished, and a blend occurs. This

agrees with our previous observation that distance surfaces produce rounds along convex regions of a skeleton, but require blends along concave regions. $cos(\theta)$ must be non-negative, however, to avoid enlarging the primitive ranges. The following illustration depicts possible combinations of two primitives.

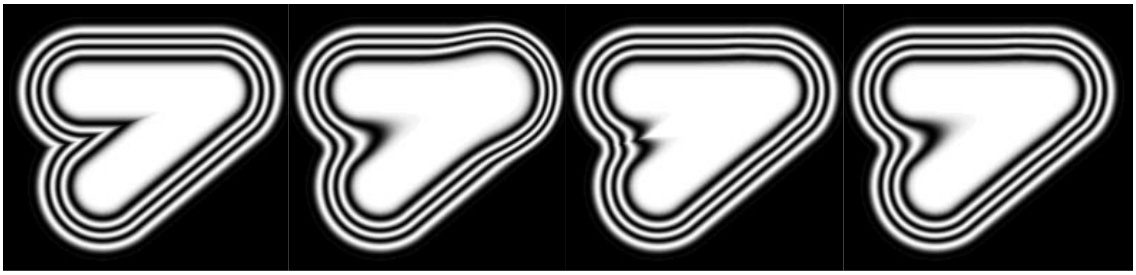


Figure 4. Blend Based on Two Segments
far left: simple union, left: bulging blend, right: use of cos, far right: use of nonnegative cos

In [Rockwood 1989] the blend is extended to k primitives:

$$(4) \qquad B_k = 1 - \sum_{i=1}^{k} \left[1 - \frac{P_i(\boldsymbol{p})}{r_i}\right]_+^t - c$$

The application of equation (3) to equation (4) is problematic, as θ can be applied to two primitives only. Primitives could be functionally composed pairwise, but we prefer an independent evaluation of primitives, in arbitrary order. Such independence simplifies implementation and promotes extensibility, without influencing the design of an object.

Convolution Surfaces

An independent evaluation of primitives is a feature of *convolution surfaces*, proposed as a bulge-free implicit blend technique in [Bloomenthal and Shoemake 1991]. Three-dimensional convolution treats a skeleton S as a set of points, each of which contributes to the implicit surface function according to its distance to $\boldsymbol{p}$. This is reminiscent of [Blinn 1982], in which an implicit surface is defined as the summation of terms, each based on the exponential (*i.e.*, *Gaussian*) decay of distance to a point:

$$(5) \qquad f(\boldsymbol{p}) = c - \sum_i e^{-\|\boldsymbol{p} - \boldsymbol{s}_i\|^2/2}, \text{ where a point on the skeleton is denoted by } \boldsymbol{s}_i.$$

If $S = \{\boldsymbol{s}_i\}$ is a set of infinitesimally spaced points, f can be expressed as an integral:

$$(6) \qquad f(\boldsymbol{p}) = c - \int_S e^{-\|\boldsymbol{p} - \boldsymbol{u}\|^2/2}\, d\boldsymbol{u}, \text{ where } \boldsymbol{u} \text{ ranges over all points on the skeleton.}$$

Convolution is a process whereby a *signal* is modified by a *filter*. Here, the signal is a skeleton and the filter is a three-dimensional Gaussian *kernel*. Unlike algebraic surfaces, a blend is achieved by integration. The evaluation of the convolution integral is discussed

in [Bloomenthal and Shoemake 1991] and examined further in [Bloomenthal 1995].

In general, the frequency components of the signal are scaled by those of the filter. This can produce various results, but low-pass filtering, *i.e.*, the removal of high frequency components in the signal, is the most relevant to blending. When a signal is low-pass filtered, it loses detail and is said to be *smoothed*. For example, a two-dimensional image becomes blurred and objectionable creases, such as those in figure 1, are reduced or eliminated. The affect of a filter can be shown by the *Fourier transform*, which converts a signal (such as the kernel itself) into its frequency components. Interestingly, the Gaussian, shown below, is its own Fourier transform; thus, the upper frequencies of the original shape are attenuated to a gently increasing degree. We call the zero set of the smoothed function a *convolution surface*.

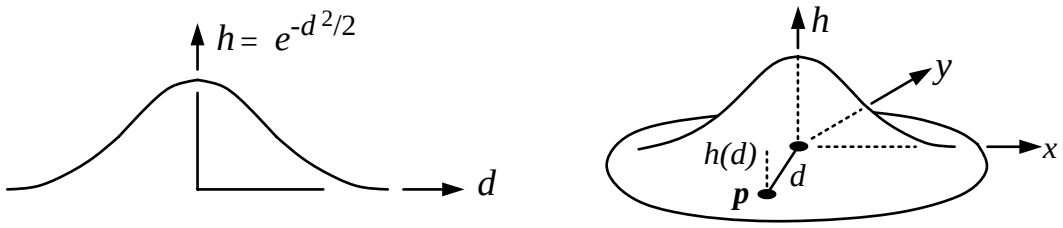


Figure 5. Gaussian Kernels
left: one-dimensional, right: two-dimensional

Convolution is well-known as an elegant solution to a wide range of application problems; here its elegance lies in its ability to smooth a shape without introducing bulges. This is due to its property of *superposition*. Because convolution is a linear operator, the sum of the convolutions of any division of a skeleton is identically equal to the single convolution of the entire skeleton. That is, using $\otimes$ to represent convolution, $h \otimes (s_1 + s_2) = (h \otimes s_1) + (h \otimes s_2)$. This guarantees, for example, that two abutting, collinear segments produce the same convolution as does the single segment that is their union. Because of this property, we may convolve each skeletal element individually and sum the results. The division of a skeleton into small elements will not introduce any seam or bulge in the surface near the joins of the elements. This contrasts with algebraic blends, which can produce bulges near the joins of the skeletal primitives.

For example, the contours below illustrate the sum of two convolutions; the result is smooth regardless of the angle between segments. Along convex portions of the skeleton the surface mimics the union operator; along concave portions, the surface yields a blend. The intermediate contours smoothly interpolate the extrema. For isolated convex skeletons, such as triangles or segments, convolution produces surfaces of similar shape to distance surfaces. For complex skeletons, however, convolution yields crease-free surfaces with adjacent primitives blending without seam or bulge.

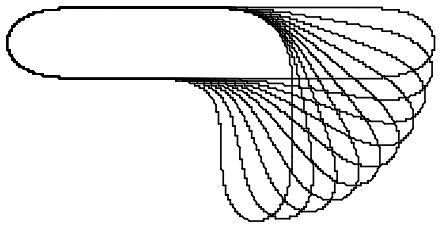


Figure 6. Two Segments Convolved with the Gaussian Kernel

Other Kernels

Although the Gaussian is a satisfactory kernel, we also consider the B-spline and Wyvill kernels, both prominent in the literature [Bartels *et al.*], [Wyvill *et al.*]. The Gaussian is separable, but its integral must be numerically approximated. The B-spline and Wyvill kernels can be analytically integrated, but are not separable. Experiments with negatively-lobed filters (such as $sin(\pi x)/\pi x$) suggest monotonicity of the kernel is necessary for a satisfactory convolution surface; otherwise, the filter may introduce spurious contours, [Bloomenthal 1995].

The B-spline is given by a set of 4^{th} order basis functions: $(x+2)^3/6$ for $x \in [-2, -1]$, $(-3x^3-6x^2+4)/6$ for $x \in [-1, 0]$, $(3x^3-6x^2+4)/6$ for $x \in [0, 1]$, and $(2-x)^3/6$ for $x \in [1, 2]$. The Wyvill is given by $(9-4x^6+17x^4-22x^2)/9$. The Gaussian has infinite support (although, in practice, there is negligible energy above 3), the B-Spline has a support of 2, and the Wyvill has a support of 1. All three kernels, when scaled so that $h(0) = 1$ and $h(1) = ½$, are similarly shaped, as shown below.

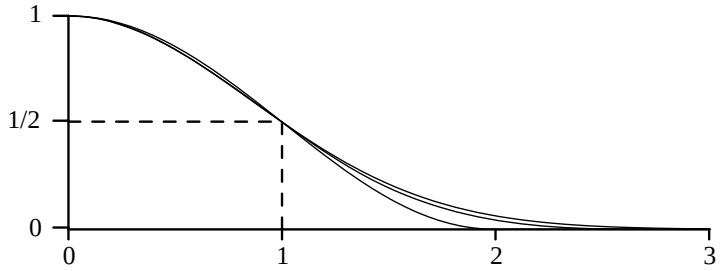


Figure 7. Comparison of Filter Kernels
upper: Gaussian, middle: B-spline, lower: Wyvill

Normalizations

If the exponent in equations (5) and (6) is scaled by π, the kernel will have unit integral. That is, the area under the curve (figure 5, left) and the volume under the surface (figure 5 right) will both be one. This implies that a signal modified by such a kernel will maintain its original energy. For example, if we convolve a box function (whose width exceeds the full filter support) with a unit integral kernel, the peak amplitude will equal the amplitude of the original box function. Or, if we convolve a two-dimensional image, the overall

energy in the image will be preserved. A kernel with integral less than one, however, would attenuate a signal; a kernel with integral greater than one would amplify the signal.

Figure 8. Convolution of a Box Function

Because the Gaussian is symmetric, the convolution equals ½ where the box function undergoes transitions. Thus, for an iso-surface contour level of ½, the convolution surface will pass through the endpoints of skeletal segments and through the edges of skeletal polygons. Accordingly, we express the convolution surface as:

(7) $f(\mathbf{p}) = ½ - (h \otimes s)(\mathbf{p})$.

Ramification

Consider a distance surface given by a branching skeleton that is organized hierarchically into parent and child segments. The implicit surface function can be given as the parent segment function or the summation of the child segment functions, whichever is greater:

(8) $$f(\mathbf{p}) = max(f_{segment}(\mathbf{p}, parent), \sum_{i}^{n} f_{segment}(\mathbf{p}, child_i),$$

where $f_{segment}$ is some function of distance from $\mathbf{p}$ to a segment and n is the number of child segments. To maintain continuity of radii, the radius of *parent* is scaled such that in the plane perpendicular to *parent*, at its endpoint, $f_{segment}(parent) = \Sigma f_{segment}(child_i)$. As a consequence of the summation in equation (8), the parent branch will be thicker than the individual child branches. This is another instance of bulging in implicit modeling. It is reasonable to ask whether convolution can solve the problem of ramiform bulge.

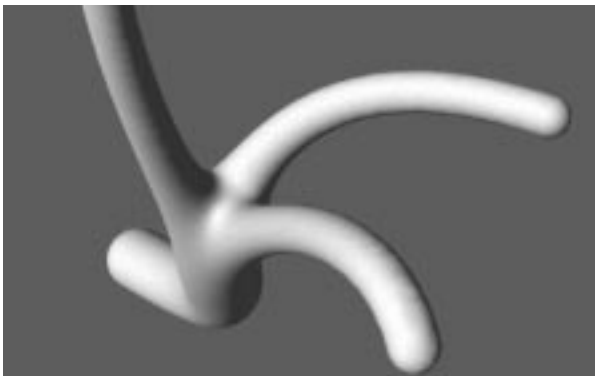


Figure 9. A Trifurcated Ramiform

We first examine a simple 'tee' skeleton, below, which consists of two segments. Each cylinder has radius r. Application of a Gaussian filter yields the following contours; the

dashed lines in the shaded images demarcate the isolated cylindrical primitives. Slight bulges at the segment junction are apparent in both planes.

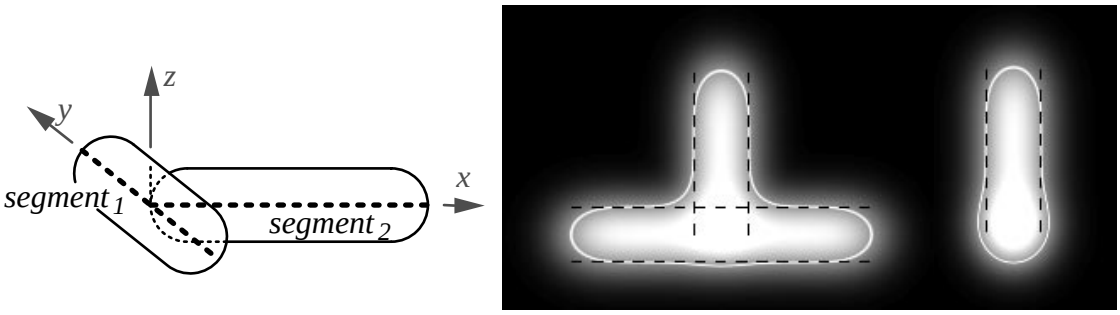


Figure 10. A 'Tee'
left: skeletal geometry, middle: contour in xy-plane, right: contour in xz-plane

To understand this phenomenon better, consider the behavior of $f(\boldsymbol{p})$ for points $\boldsymbol{p} = (x, 0, r)$, shown below, left. As x increases and $\boldsymbol{p}$ moves along the $z = r$ line, the convolution of the two segments, measured at $\boldsymbol{p}$, changes. This is graphed in the figure below, right, and predicts the bulge at the junction of the two segments. Details of the computation are provided in [Bloomenthal 1995].

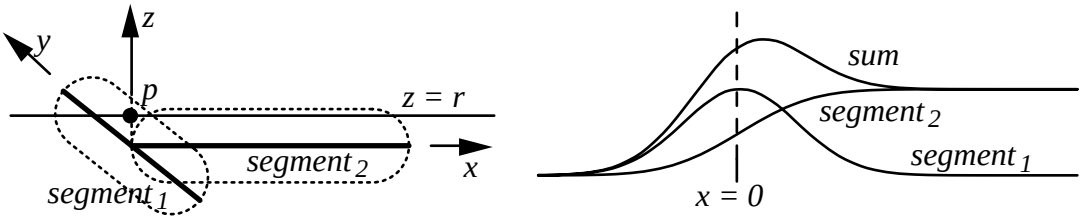


Figure 11. 'Tee' Junction with Segment$_1$ and Segment$_2$ Touching

Bulge Reduction

It is possible to reduce the bulge by slightly separating the segments. If the left endpoint of *segment*$_2$ is moved to the right by r, as shown below, the summation, although still not constant, shows less variation.

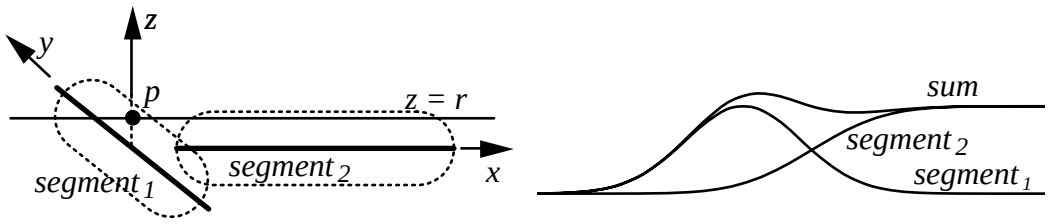


Figure 12. 'Tee' Junction with Shortened Segment$_2$

Shortening $segment_2$ does, however, improve the appearance of the contours shown below. The bulge and dip predicted by the mathematical model, although subtle, are visible in the shaded surface.

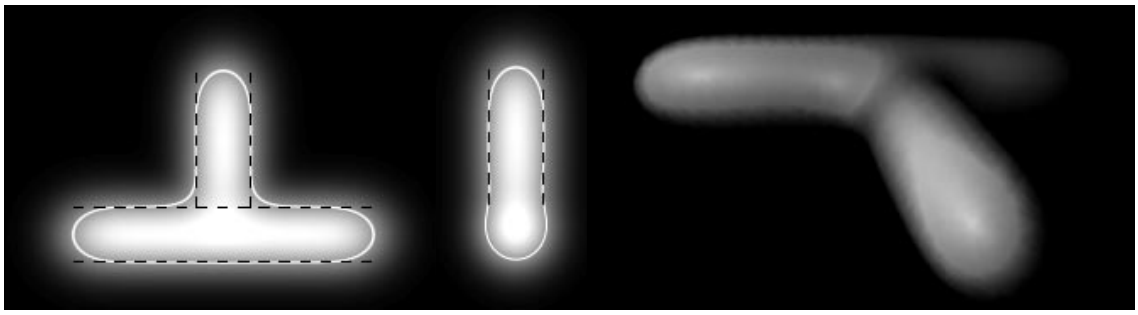


Figure 13. Contours and Surface for the 'Tee' with Shortened Segment$_2$

The Combination Surface

It would appear that we must blend when ***p*** is within the plane of the junction and avoid a bulge by not blending when ***p*** is out of the plane of the 'tee.' This 'combination surface' is reminiscent of equation (3) and is readily implemented for a skeleton by associating an approximating plane with each skeletal joint. The contribution of the 'union surface' can be derived from the angle between the plane normal and the vector from the joint to the ***p***. In the figure below we compare the three types of surfaces.

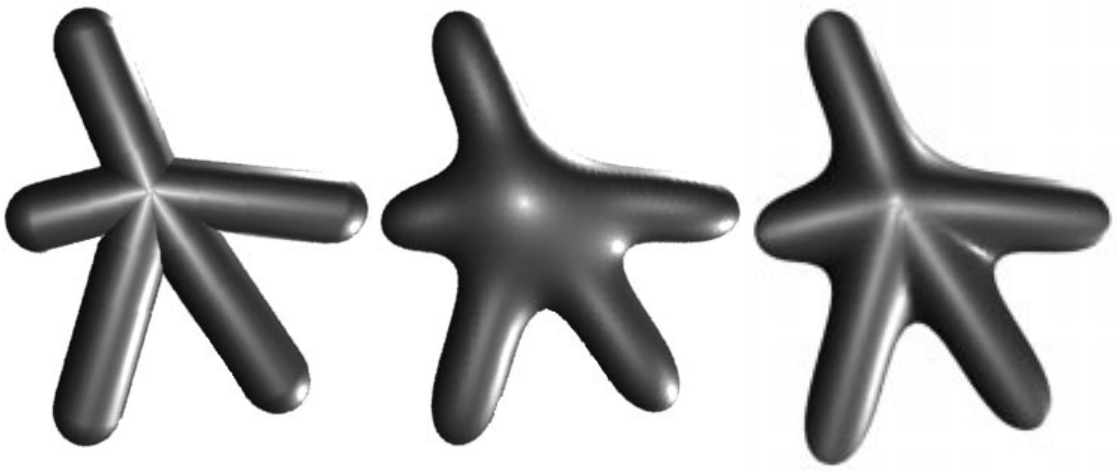


Figure 14. Union Surface (left), Convolution Surface (middle), and Combination (right)

The combination surface can be modified in several ways; we may, for example, experiment with the function that interpolates the union and convolution surfaces. Or, rather than fit a plane to the vertices of a joint, a free-form surface could be fit. As described in [Bloomenthal 1995], the webbing that blends together pairs of limbs in the combination surface will, necessarily, contain a small crease. This does not appear

objectionable in the above image, but prompts further investigation.

Two Dimensional Skeletal Elements and Bulge Elimination

To accommodate objects with non-circular cross-sections, one-dimensional skeletal curves were extended to two-dimensional skeletal polygons [Bloomenthal and Shoemake 1991]. As with one-dimensional skeletal elements, the convolution surface for two-dimensional skeletal elements is evaluated as the sum of independent primitives. For example, the two polygons, below, left, yield the surface shown below, right.

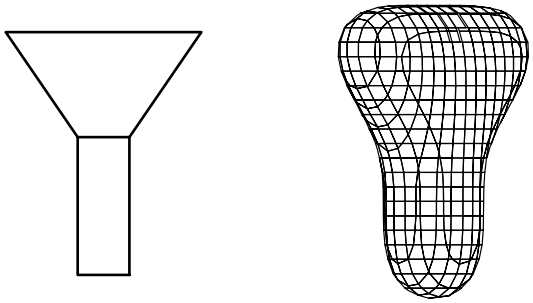


Figure 15. Polygonal Skeleton (front view) and Convolution Surface (oblique view)

A more sophisticated application of both one and two-dimensional skeletal elements, below, left, yields the surface shown below, right.

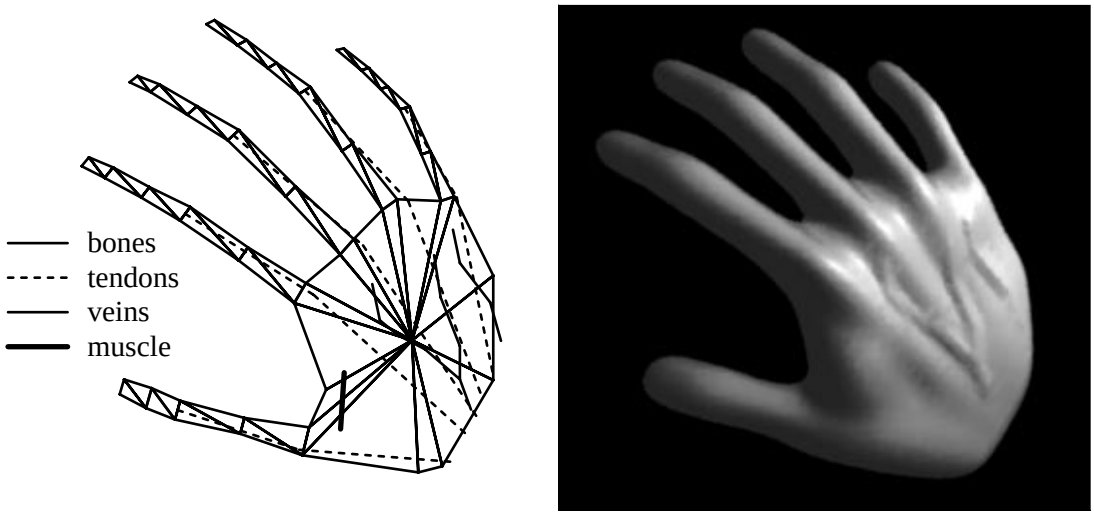


Figure 16. Skeleton and Hand

In the previous section we expressed dissatisfaction concerning the geometric quality of the combination surface developed to prevent bulges resulting from ramified skeletons. As seen in figure 11, a bulge is a consequence of increased skeletal density at the junction of the ramiform. In particular, an endpoint of $segment_2$ touches $segment_1$, and the skeletal

density increases at this point. For polygons, a comparable arrangement might be two overlapping polygons, as shown below, left. We speculate that a contiguous (*i.e.*, abutting and non-overlapping) skeletal arrangement, such as below, right, alleviates the bulge.

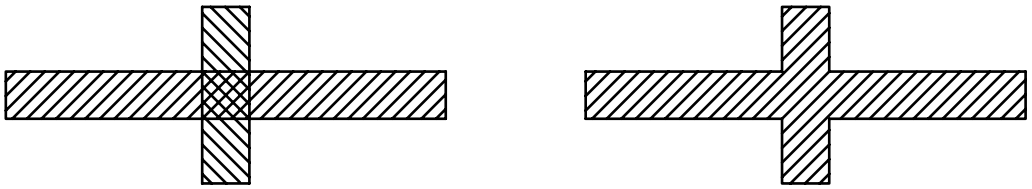


Figure 17. Overlapped (left) and Contiguous (right) Skeletons

To demonstrate this, and to explain the conditions under which it is true, we must examine the cross-section of a convolution surface derived from a polygon. As we observed in developing equation (7), the convolution surface of a polygon passes through the polygon edges. As shown below, different filter kernels require different polygon widths to produce equally thick surfaces.

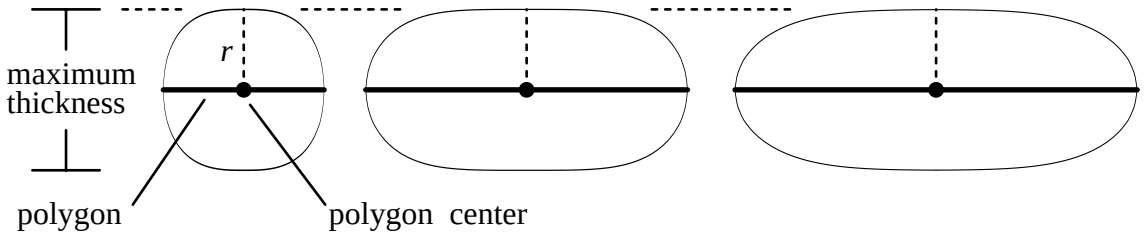


Figure 18. Surface Cross-Sections for Different Kernels
left: Wyvill (width = 2r), middle: B-Spline (4r), right: Gaussian (5r)

For each of the filters shown above, the polygon width equals the effective support of the filter. This means the surface just reaches its maximal thickness at a point above the center of the polygon. Widening the polygon will not change the function value for points above the polygon center, but it will create a 'plateau' along the top and bottom of the surface, as shown below, left. If the polygon becomes narrower than the filter support, however, there is no point above the polygon for which the kernel, integrated over the polygon, can yield unity. This reduces the thickness of the surface, as shown below, right.

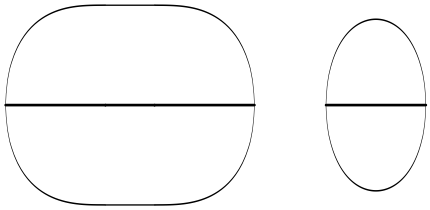


Figure 19. Varying Polygon Width (Wyvill Kernel)
left: a wider polygon widens surface, right: a narrower polygon reduces object thickness

Now, consider the contiguous skeletons shown below. For polygons whose width equals the support of the filter kernel, the entire filter (represented as a circle) projects onto the polygon, and the very centers of the polygons will yield a surface of maximal thickness. For the wider polygon, the region of maximal thickness will widen, creating a plateau. But for the narrower polygon, only at the junction center is there sufficient room for an entire kernel; elsewhere the kernel is clipped. A bulge will occur at the center of the junction of the narrow polygons, but not with the wider polygons.

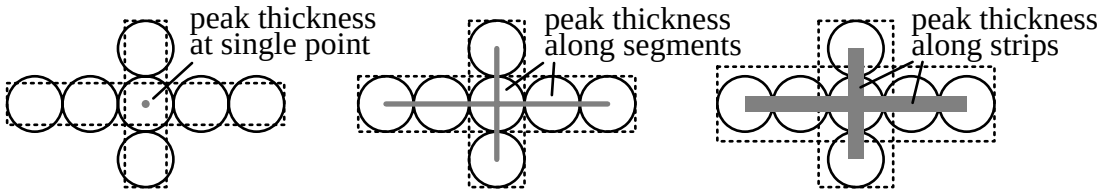


Figure 20. Affect of Varying Polygon Width on Integration Filter
left: polygon width < 2r, middle: polygon width = 2r, right: polygon width > 2r

Thus, for an appropriate choice of polygon width, a contiguous polygonal skeleton yields ramiforms without bulge. The cross-sections of surfaces are not perfectly circular, however, unlike those for segments. The above skeleton is articulated, below.

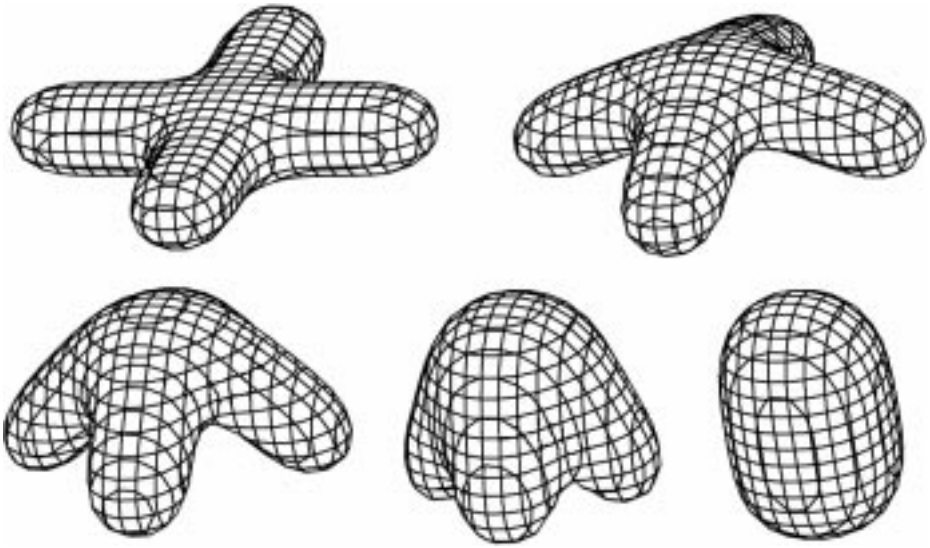


Figure 21. A Smoothly Folding, Bulge-Free Form

In the previous section we discussed the 'plateau' that develops over wide polygons. Because of the superposition property of convolution, this plateau may develop from an arbitrary configuration of coplanar, abutting polygons, and will be seamless. This seamlessness provides flexibility in the definition of polygonal skeletons, which we attempt to demonstrate in the following illustration.

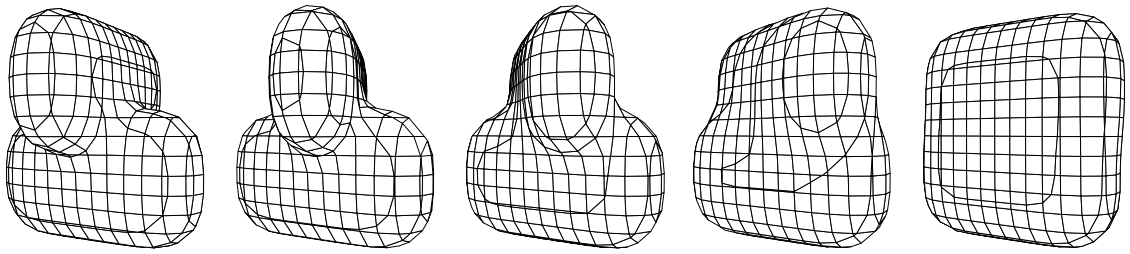


Figure 22. Seamlessness

Conclusions

Because convolution is an integration, it produces a material blend of primitive volumes; the resulting object appears pliable. The object is not amorphous, however, because blending occurs only where skeletal elements are in proximity; the remainder of the surface closely follows the skeletal structure.

We dislike the popular term 'blobby' because it suggests something amorphous and without structure. The skeleton, however, is meant to provide structure, and, therefore, we prefer the term 'blend.'

We speculate that one-dimensional segments and two-dimensional polygons provide great flexibility for the design of convolution surfaces, although the use of point and volumetric skeletons may also be appropriate for certain forms. As described in [Bloomenthal 1995], the convolution of a single point source may be performed analytically, the convolution of a segment requires analytical computation as well as a table for the Gaussian integral, and the convolution of a polygon requires a raster representation for the polygon. A practical implementation for the convolution of a volume would require a discrete, voxel representation.

Just as the use of polygons eliminates the bulge introduced by a corresponding skeletal structure based on segments, the use of volumes eliminates the bulge introduced by a corresponding skeletal structure based on polygons. In other words, to avoid bulges in implicit blends, a skeleton should be locally manifold. For a one-dimensional skeleton, this means there should be no points incident to a segment; for a two-dimensional skeleton, there should be no edge incident to a polygon. A comparison is given below.

Figure 23. One, Two, and Three-Dimensional Skeletons
left: those that produce bulges, right: those that do not

A design environment utilizing convolution surfaces enables the creation of complex, well-behaved shapes through the specification of skeletal elements. In using convolution, however, the designer loses explicit control over fillets and chamfers. This is more than compensated by the generality of convolution blending. Resulting surfaces are smooth and are bulge-free provided the skeletal elements are contiguous and, collectively, at least as wide as the full filter support.

Acknowledgments

My thanks to Paul Heckbert, Przemek Prusinkiewicz, and Brian Wyvill. Their help has been greatly appreciated.

References

R. Bartels, J. Beatty, and B. Barsky, *An Introduction to Splines for use in Computer Graphics and Geometric Modeling*, Morgan Kaufmann Publishers, Los Altos CA, 1987.

James Blinn, *A Generalization of Algebraic Surface Drawing*, ACM Transactions on Graphics, July 1982).

Jules Bloomenthal, *Techniques for Implicit Modeling*, Xerox PARC Technical Report P89-00106, 1989.

Jules Bloomenthal, *Skeletal Design of Natural Forms*, Ph.D. dissertation, Department of Computer Science, University of Calgary, 1995.

Jules Bloomenthal and Ken Shoemake, *Convolution Surfaces*, Proceedings of SIGGRAPH'91, Las Vegas, NV, in Computer Graphics 25, 4, July 1991.

Christoph Hoffmann and John Hopcroft, *The Potential Method for Blending Surfaces and Corners*, Technical Report TR 85-674 Computer Science Dept., Cornell University, 1985.

A. E. Middleditch and K. H. Sears, *Blend Surfaces for Set Theoretic Volume Modeling Systems*, Proceedings of SIGGRAPH'85, San Francisco, CA, in Computer Graphics 19, 3, July 1985.

Alyn Rockwood, *The Displacement Method for Implicit Blending Surfaces in Solid Models*. ACM Transactions on Graphics 8, 4, October 1989.

A. Rockwood and J. C. Owen, *Blending Surfaces in Solid Modeling*, Proceedings of SIAM Conference on Geometric Modelling and Robotics, G. Farin, editor, Albany, NY, 1985.

Philip Schneider, *Solving the Nearest-Point-On-Curve Problem*, in Graphics Gems, Andrew Glassner, editor, Academic Press, New York, 1990.

J. R. Woodwark, *Blends in Geometric Modelling*, Proceedings of the 2nd IMA Conference on the Mathematics of Surfaces, Cardiff, September 1986.

G. Wyvill, C. McPheeters, and B. Wyvill, *Data Structure for Soft Objects*. Visual Computer 2, 4, August 1986.

Skeletal Methods of Shape Manipulation

Jules Bloomenthal and Chek Lim
Unchained Geometry Inc.

Abstract

The geometric skeleton is derived from a static object using an implicit 'directions' method; an IK skeleton is derived from and used to manipulate the geometric skeleton. The model may be reconstructed from the modified skeleton using implicit distance and convolution methods.

Introduction

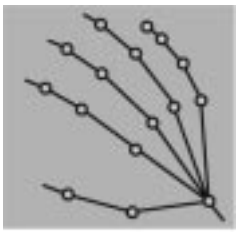
We examine the use of the skeleton to manipulate erstwhile static 3D models. These models may be generated by a design system or a hardware scanner, and may be disseminated through the public domain, catalog services, or scanning services. Originally static, the models may be positioned, oriented, and scaled, but not articulated. They may contain separate parts (such as a car and four wheels), permitting only limited animation.

In order to animate such a model, we first extract its skeleton. Then we articulate the skeleton, from which we reconstruct the animated surface.

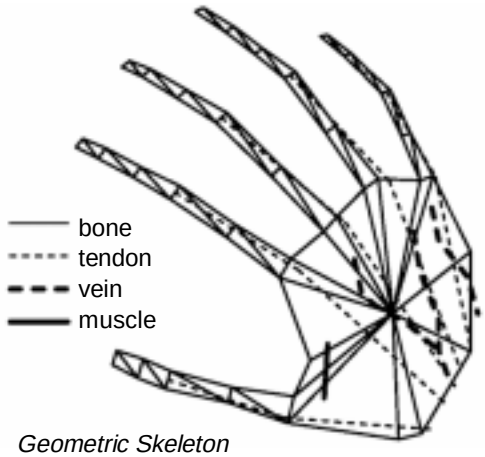
The 'IK skeleton' is traditionally used to control an articulable (*i.e.*, nonstatic) model. More recently it has been applied to static models, but this approach has several problems. The skeleton must be created, usually manually; and a correspondence between skeleton and surface must be established. Articulation is limited and arbitrary metamorphosis is not possible.

Geometrically, the 'skeleton' has the precise meaning of medial axis (or surface) of the object. The medial axis is similar to the IK (inverse-kinematic) skeleton, but is (typically) two-dimensional, whereas the IK skeleton is one-dimensional.

The following figure depicts an object, its geometric skeleton, and its IK skeleton. The geometric skeleton consists of 2D surfaces and 1D curves, and is completely surrounded by the object. Every point on the



IK Skeleton



Geometric Skeleton

Figure 1. Skeletons and Surface

geometric skeleton is a center of a sphere fully inscribed within the model and touching it at two or more distinct points.

Skeletonization

The geometric skeleton may be derived automatically from the object, and the IK skeleton derived automatically from the geometric skeleton; in the process, the skeleton/surface correspondence is automatically established There appear to be three established methods to extract the geometric skeleton.

Delaunay Triangulation approximates the skeleton as the collection of centers of an optimal triangulation (tetrahedralization) of points on the object's surface; additional points may be needed to provide a sufficiently dense triangulation.

Angle Bisection defines the skeleton as the collection of surfaces internally bisecting the dihedral angles of an original polygonal model; these bisecting surfaces must be carefully trimmed against each other.

Volumetric Thinning is an iterative attrition of voxels originally representing the object's volume; errors accumulate during the iteration.

Because each of these methods has limitations, we developed *Direction Testing*. This is an implicit method that defines the skeleton as the set of points at which the direction to the nearest point on the object undergoes a sudden transition. For example, the central skeletal component for the rectangle below contains a direction change of 180°; for the sub-components, the change is 90°.

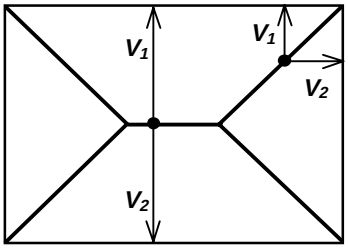


Figure 2. Rectangle, Skeleton, and Directions

Locating and connecting the skeletal points is performed by a piecewise-linear implicit surface polygonizer (see [1] for a survey of polygonization methods). At each lattice point the direction to the nearest point on the object is computed. For those edges connecting points of sufficiently divergent directions, a skeletal point is computed using binary subdivision.

Our polygonizer utilizes adaptive subdivision to improve precision in regions of high curvature, and coalesces coplanar neighboring polygons to reduce storage in regions of low curvature. The lattice is deformed according to [4], eliminating thin or small triangles.

The polygonizer must support the non-manifold and manifold-with-boundary edges that occur in skeletons. An example surface produced by a non-manifold polygonizer is shown below, from [2].

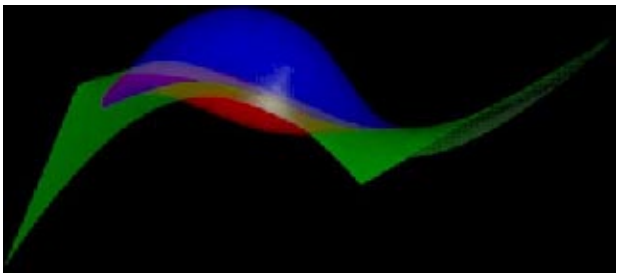


Figure 3. Non-Manifold Polygonization

We optimize the direction computation for a model described by a triangle mesh by assigning to the terminal nodes of an octree the n nearest triangles, such that d_n-$d_1 > s$, where s is the length of the node's major diagonal and $\boldsymbol{c}$ is the center of the node. For any point within the octree node, the nearest point on the object must belong to one of the n triangles.

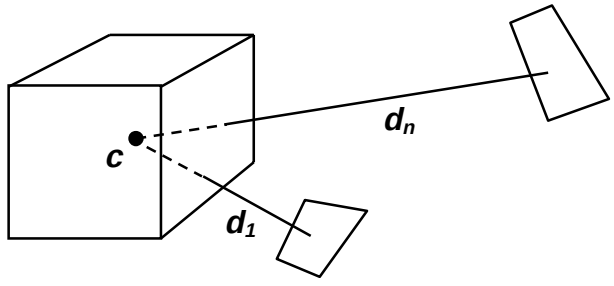


Figure 4. Octree Node and Triangle List

This method appears to offer advantages in speed and precision. The octree is constructed before skeletonization. Assuming its depth is adjusted so that a near constant number of triangles per terminal node is produced, the skeletonization running time is near linear with respect to the geometric extent of the skeleton.

Because a minimum direction divergence is required, small irregularities in the surface do not produce unwanted noise (*i.e.*, branching) in the skeleton (alternatively; this may be considered a disadvantageous loss of surface detail).

Reconstruction

Once the geometric and IK skeletons are extracted from an object, the user manipulates the IK skeleton, which modifies the geometric skeleton. For every point of the geometric skeleton, we associate a distance and angular orientation to a point on the IK skeleton. When the IK skeleton is rotated, twisted, bent, or otherwise manipulated, the geometric skeleton is modified accordingly.

To reconstruct the modified object from the modified geometric skeleton, we implicitly define a distance surface to the skeleton, and polygonize. This rounds those regions of the object that correspond with convex portions of the skeleton. Optionally, we employ convolution surfaces [3] to fillet those regions of the object that correspond with concave portions of the skeleton.

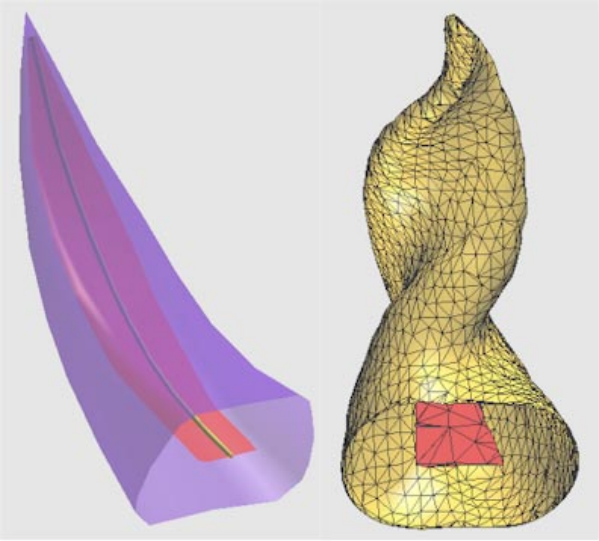


Figure 5. Skeleton and Reconstruction

It is possible to associate points on the object directly with the IK skeleton; some commercial systems, for example, associate an object point with the nearest skeletal joint. This simpler approach can, however, produce self-penetration, shearing, and creasing of the modified object, especially if the modified object is limited to the polygonal mesh of the original object. Some of these artifacts are apparent in the following figure.

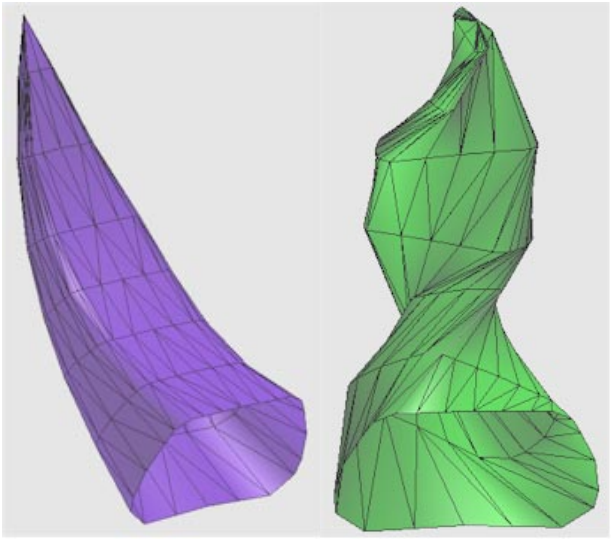


Figure 6. Reconstruction from IK Skeleton

Conclusion and Future Work

Stephen Wainwright once wrote, “structure without function is a corpse, and function without structure is a ghost” [5]. It is the skeleton that underlies the structure and function of an object.

The skeleton also offers an embedding for transformation hierarchies, maps well with motion-capture data, and is easily represented, manipulated, and rendered.

The geometric skeleton (*i.e.*, the medial axis) can be produced automatically from computer models. In conjunction with scanned objects, it permits the animation of otherwise static objects. In conjunction with motion-capture data, it permits the application of real-world motions to sampled or synthesized objects. The geometric skeleton can produce the IK skeleton, obviating a tedious and imprecise manual process.

The methods discussed above are applicable not only to the animation of existing objects, but also to the design of new objects. A skeleton may be constructed interactively or specified procedurally. Skeletons facilitate a constructive and hierarchical approach to modeling and to levels of detail. Two skeletons can be interpolated, permitting the interpolation of two different objects more convincingly than current ‘morph’ methods.

The skeleton is more compact than its corresponding object, requiring less storage and transmission bandwidth. The skeleton supports ancillary properties, such as dynamics.

The skeleton facilitates the design of rounded and filleted surfaces, which are often difficult and tedious to achieve using conventional design tools. CAD systems specify these details with greater precision, however, and have greater application in engineering disciplines. Skeletal design will likely find more application in related disciplines such as industrial design, in which appearance can be more important than geometric precision.

The large design tools provided commercially are unwieldy for the lay person and, even, for many graphic designers. This is due, we believe, to the tools' emphasis on surface manipulation. The use of the skeleton may signal the development of more intuitive design tools.

It would be useful to identify those canonical skeletal operations from which more complex operations can be composed. These could form the basis for clip-behavior, offering a large repertoire of design and animation to the user. Existing shape taxonomies may provide some guidance in the development of these operations.

Is there an optimal language of geometric modeling and, if so, what are its semantics and syntax? What is an optimal form of user interaction? We may ponder these questions for many years, but it may not be long until the skeleton is the established key to desktop 3D.

References

1. Jules Bloomenthal, editor, *Introduction to Implicit Surfaces*, Morgan Kaufmann Publishers, San Francisco 1997.

2. Jules Bloomenthal and Keith Ferguson, *Polygonization of Non-Manifold Surfaces*, Proceedings of SIGGRAPH 95 (Los Angeles, CA, Aug. 6-11, 1995). In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH, pp. 309-316.

3. Jules Bloomenthal and Ken Shoemake, *Convolution Surfaces*, Proceedings of SIGGRAPH 91 (Las Vegas, Nevada, Jul. 28-Aug. 2, 1991). In *Computer Graphics*, 25, 4 (Jul. 1991) ACM SIGGRAPH, New York, 1991, pp. 251-256.

4. Douglas Moore and Joe Warren, *Compact Isocontours from Sampled Data*, Graphics Gems III, David Kirk, editor, Academic Press, New York, 1992].

5. Stephen Wainwright, *Axis and Circumference: the Cylindrical Shape of Plants and Animals*, Harvard University Press, Cambridge, 1988.

Medial-Based Vertex Deformation

Jules Bloomenthal
Unchained Geometry, Inc.

Abstract

Vertex deformation is a popular technique to animate an erstwhile static object. It is difficult, however, to deform those vertices near multiple limbs of the controlling stick-figure skeleton while maintaining a natural-appearing surface.

By applying convolution to the medial axis/surface of the object, the weights associated with vertex deformation can be computed automatically. Fewer undesired artifacts are evidenced in the animated surface.

1. Introduction

Vertex deformation is a method of character animation in which the vertices of an object's surface are moved interactively in response to some control mechanism. The number of vertices and their inter-connectivity do not change.

The control mechanism may be a surrounding lattice or an internal stick-figure 'skeleton'. We consider the skeleton approach, which is called *axial deformation* [Lazarus *et al*. 1994] or *skeleton subspace deformation* [Lewis *et al*. 2000].

The stick-figure skeleton consists of individual *limbs*; two or more limbs meet at an articulable *joint*. Either the skeleton is posed via end effectors, in which case inverse kinematics is used to compute the joint angles, or the joint angles are specified directly by the animator.

It is straightforward to produce a 'rigid-body' rotation of a vertex associated with a single limb. It is considerably more challenging to move a vertex situated near two or more limbs such that the animated surface is smooth and natural in appearance.

Typically, the vertex location is given as a *vertex blend* (*i.e*., a weighted sum) [Akenine-Möller and Haines 2002]. Each term of the summation is computed using a reference frame associated with a corresponding, animated limb.

Vertex deformation appears increasingly popular, in part due to increased hardware support for vertex blends [Akenine-Möller and Haines 2002], [Lander 2001].

As described in [Lewis *et al*. 2000], the default coefficients of the weighted sum often require tedious, manual adjustment and the results often evidence numerous artifacts. Indeed, these difficulties are considered 'notorious'. They are also regarded as inevitable as a consideration of the common case involving two limbs straddling a joint reveals the 'collapsing joint defect' in which the surface is pinched around a joint that is bent or twisted [Lewis *et al*. 2000].

The situation improves when intermediate reference frames are introduced. For example, a 180° twist between successive frames yields a zero radius (see figure below). But with a single intermediate frame the radius shrinks less than 30% and with three intermediate frames (*i.e*., a 45° rotation between frames), the shrinkage is less than 8%.

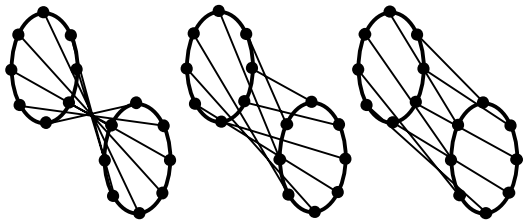


Figure 1. Pinch due to twist between frames (left to right: 180°, 90°, and 45°).

This paper addresses the collapsing joint defect with a method that, unlike conventional 'deformers', computes multiple frames (and associated weights) for each limb influencing an object vertex. The technique reduces artifacts such as pinching and bulging, and obviates the manual adjustment typical of vertex deformation.

The following section provides background on vertex deformation. Section 3 describes our method, which involves convolution of the medial axis/surface of the object. Section 4 presents details concerning the medial and section 5 provides details concerning convolution. Section 6 demonstrates results, and section 7 summarizes the paper and considers future work.

2. Vertex Deformation

This section summarizes the basic methodology of vertex deformation (an early description is given in [Magnenat-Thalmann *et al*. 1988, sec. 5]).

A reference frame (*i.e*., coordinate system) F is established for each limb of the stick-figure skeleton. The frame consists of a location F_p and three orthonormal axes, F_t, F_n, and F_b (*i.e*., the tangent, normal and bi-normal), with F_t typically directed from base towards tip of the limb. F_p may be positioned arbitrarily along the limb; in Figure 2 it is at the base.

With the object at rest (*i.e*., no joint articulation), each surface vertex $\boldsymbol{q}$ is assigned a set of transformed $\boldsymbol{q_i'}$ that correspond with those $limb_i$ influencing $\boldsymbol{q}$. For a given limb L, $\boldsymbol{q'}$ is simply $\boldsymbol{q}$ with respect to the at-rest frame F. Referring to figure 2, $\boldsymbol{q'}$ is ($\boldsymbol{r}\bullet\boldsymbol{t}$, $\boldsymbol{r}\bullet\boldsymbol{n}$, $\boldsymbol{r}\bullet\boldsymbol{b}$), *i.e.,* the projection of $\boldsymbol{r}$ onto the axes of frame F.

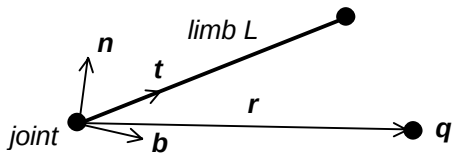


Figure 2. Frame F for *limb* $\boldsymbol{L}$.

In some systems [Alias/Wavefront 1999], the joints can be interpolated by curves, in which case the frame orientation changes along a limb and the at-rest frame F is typically positioned at $\boldsymbol{n}$, the point on the curve nearest to $\boldsymbol{q}$. F is augmented with t, the parametric location of $\boldsymbol{n}$.

Upon articulation, limbs move and rotate, which necessitates that frames F_i be recomputed. The new location of vertex $\boldsymbol{q}$ is given by:

$$\Sigma w_i \, (F_p + \boldsymbol{q'}_x F_t + \boldsymbol{q'}_y F_n + \boldsymbol{q'}_z F_b))$$

where F is the new frame for articulated $limb_i$ and $\boldsymbol{q}$' is $\boldsymbol{q}$ transformed by the at-rest frame F.

The initial w_i assigned to $\boldsymbol{q}_i$ are usually defaulted to some inverse function, such as $1/d_i^2$ where d_i is the distance between $\boldsymbol{q}$ and the nearest point on $limb_i$. Once computed, the weights are normalized, *i.e*., $\Sigma w_i = 1$.

As detailed in [Lewis *et al*. 2000], these default w_i do not yield natural motions of the surface. Significant manual adjustment is usually required and good results difficult to achieve.

3. A New Method to Calculate Vertex Weights

To produce the reference frames and associated weights needed for skeletal vertex deformation, we utilize the *medial* of the three-dimensional object. The medial consists of two-dimensional surfaces and one-dimensional arcs, which connect along non-manifold seams and points. We represent the surfaces and arcs with triangles and segments, respectively, and refer to them as *primitives* P_i.

The original object may be reconstructed by defining a distance field around P_i. A similar but *differentiable* field is obtained by applying a *convolution filter* to P_i.

For a given vertex $\boldsymbol{q}$ of the object, a reference frame F_i is computed for each P_i that has at $\boldsymbol{q}$ a non-zero convolution value; this value is assigned to w_i. The actual method to align F_i within Pi is not significant but must be consistent; for triangle P, the normal and first edge, for example, may be used to construct the frame.

The stick-figure skeleton controls the medial. As the skeleton articulates, vertices of the medial are moved and new medial frames computed. Object vertices are recomputed using these new medial frames. Lists of affected vertices are maintained per stick-figure limb so that the locations of unaffected vertices are not recomputed.

This method is akin to that of [Stalpers and van Overveld 1997], in which rectangles control vertex deformation through reference frames associated with the rectangles. The rectangles are user-specified and the influence of a rectangle on an object vertex is determined by a network-based distance between rectangle and vertex. In contrast, the present work utilizes the medial, which may be derived from the object, and utilizes convolution to produce a differentiable field.

4. The Medial

The medial axis/surface (or simply 'medial') of an object is defined in [Blum 1967]. It is the locus of centers of all maximally sized spheres contained within the object. As such it is the 'center of support' for the object and provides a means to compute local object thickness.

One measure of the integrity of an animated shape is its retention of local thickness. This suggests that when computing w_i for vertex $\boldsymbol{q}$, the distance to the local 'center' of the object should be a factor. As shown

below, the stick-figure skeleton is less accurate than the medial.

Aside from using object thickness to determine w_i, our method does not attempt to maintain a constant volume of the object.

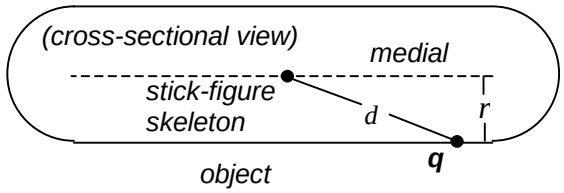


Figure 3. Distance *d* to the stick-figure skeleton (viewed head-on and represented as a single point) is a poor estimate of thickness *r*, as measured to the medial.

The medial exists unambiguously for any finite 3D object, but for even simple objects it can be quite complex and, in general, difficult to compute [Amenta *et al.* 1998]. In [Turkiyyah *et al.* 1997] methods of medial extraction are categorized.

More than one skeletal limb will influence those vertices of the medial that are near skeletal joints. This is accommodated by associating a frame *F* per influencing skeletal limb, and computing a weight via convolution of the limb. That is, the stick-figure skeleton controls the medial in a manner similar to the control of the object by the medial. This is illustrated in figure 4. An example object, its medial, and an articulated medial are shown in figure 5.

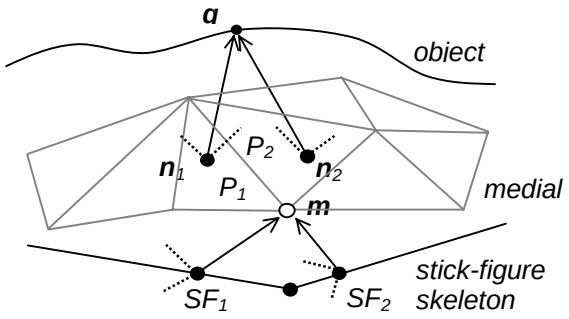


Figure 4. Reference frames (dashed) at $\boldsymbol{n}_1$ and $\boldsymbol{n}_2$, nearest points on triangles P_1 and P_2 to vertex $\boldsymbol{q}$; also shown are SF_1 and SF_2, stick-figure reference frames that control the medial vertex $\boldsymbol{m}$.

5. Convolution

A *convolution field* is produced around the medial by convolving a three-dimensional cubic filter (for example) with the medial. The width of the filter is proportional to and varies as the radius of the medial. Typically the filter is finite in extent, or windowed.

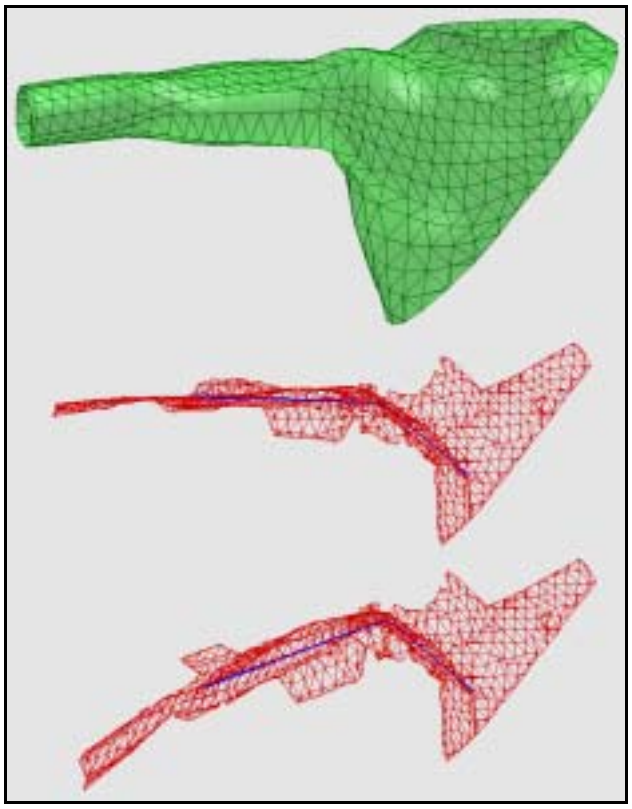


Figure 5. Top: object, middle: at-rest stick-figure skeleton and medial, bottom: articulated skeleton and medial.

The medial may be divided into manageably sized curve and surface primitives P_i. Let $Conv_i(\boldsymbol{q})$ be the value of the convolution field due to P_i, evaluated at $\boldsymbol{q}$. $Conv_i(\boldsymbol{q})$ is the 'influence' that the whole P_i (not simply a single point on P_i) exerts on $\boldsymbol{q}$, and is used for w_i. Because of the super-position property of convolution, the partitioning of the medial into P_i does not affect the overall convolution field at a vertex $\boldsymbol{q}$.

A discrete method to convolve curves and surfaces is developed in [Bloomenthal and Shoemake 1991]. An analytical method is given in [Sherstyuk 1999]. Convolution methods involve some effort to implement, but the computation itself is not unduly demanding. Once the vertex weights are computed, convolution is not used during the animation.

The following several figures employ a one-dimensional contour in two dimensions to compare three forms of deformation: unweighted, weighted by inverse-square, and weighted by convolution. The contour includes 52 points evenly distributed around the at-rest limbs.

The unweighted scheme, in which each contour point is associated with but a single limb, yields self-intersection when the joint is articulated.

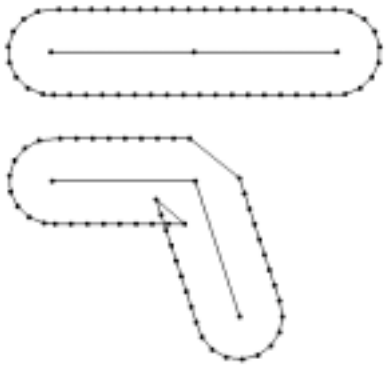


Figure 6. At-rest and articulated contours, unweighted

If, however, each contour point is computed as the weighted sum of two $\boldsymbol{q'}$, one for each of the two limbs, the results are much improved. In the following figure at left is a deformation in which $\boldsymbol{w_i} \propto 1/d_i^2$, and at right a cubic filter is applied to $limb_i$ and $\boldsymbol{w_i} \propto$ to the convolution value at $\boldsymbol{q}$.

With inverse-square weighting, the articulation distorts and shifts the shape; with convolution, the shape is pinched. Both functions are differentiable, which is desirable in that two surface vertices close to each other will have similarly close weights; otherwise, small joint rotations can induce abrupt surface motion

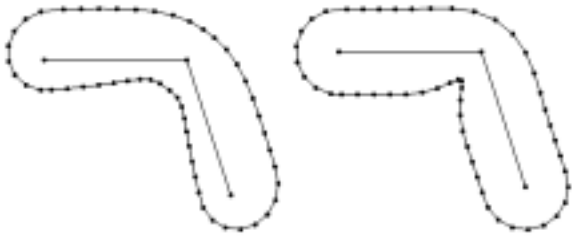


Figure 7. Left: weight based on $1/d^2$, where d is distance from $\boldsymbol{q}$ to limb; right: weight based on the convolution of a limb, measured at $\boldsymbol{q}$

7. Results

The setup time for computing the medial, convolving it, and producing the vertex weights for the torso shown in figure 5, is less than one minute on a 700 MHz processor. After setup, the following shapes are produced in real-time.

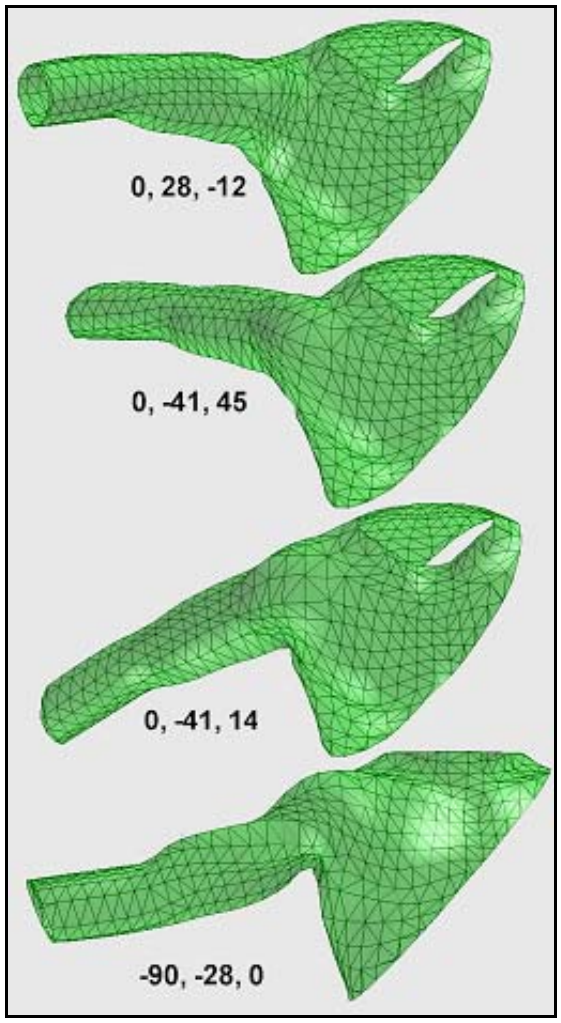


Figure 8. Articulation of the shoulder joint, with twist, pitch, and yaw given in degrees.

8. Conclusion

Conventional vertex deformation schemes typically employ a weighted blend of two points: one associated with the reference frame belonging to the articulating limb, the other associated with the reference frame of the parent limb. We have suggested that additional reference frames ameliorate the collapsing joint artifact, and that these frames may be obtained from the medial of the object.

The method automatically produces reasonable vertex weights, obviating the 'tweaking' otherwise needed. It shows good motion of the shoulder, a body part considered difficult to animate. Side by side comparisons with other methods would be a useful means to evaluate the method but, regrettably, such comparisons are beyond the scope of this paper.

Our deformation scheme is not a pose interpolation

method such as described in [Lewis *et al*.] or [Sloan *et al*. 2001]. Nor is it a reconstruction scheme based on the medial, such as described in [Bloomenthal and Lim 1999], [Gagvani *et al*. 1998] or [Bittar *et al*. 1995]. It may, however, be used in conjunction with these methods.

There are many unanswered questions concerning the present technique, such as its robustness given a 'noisy' medial, its dependence on the resolution of the medial, its ability to use current hardware support, and so on. Especially difficult are numerous questions concerning the control of the medial by the stick-figure skeleton; the concept of 'skeletal distance' as developed in [Stalpers and van Overveld 1997] might ensure that object parts that are physically but not geodesically close (such as, *e.g*., two fingertips) do not influence each other.

The method holds promise for the animation of objects modeled with conventional software or physically scanned. It also holds promise for interactive games, which require the speed of vertex deformation but often suffer from poor surface geometry near joints.

Acknowledgements

The author is indebted to the reviewers for their comments and suggestions.

The method reported here is patent-pending.

References

T.Akenine-Möller and E.Haines, *Real-Time Rendering*, 2nd ed., A.K. Peters Ltd., 2002.

Alias/Wavefront, *Learning Maya 2*, Toronto, 1999.

N.Amenta, M.Bern, M.Kamvysselis, *A New Voronoi-Based Surface Reconstruction Algorithm*, Proc. SIGGRAPH 1998.

E.Bittar, N.Tsingos, and M.-P. Cani, *Automatic Reconstruction of Unstructured 3D Data: Combining a Medial Axis and Implicit Surface*, Eurographics '95.

J.Bloomenthal and C.Lim, *Skeletal Methods of Shape Manipulation*., Proc. Shape Modeling Int'l, Aizu, Japan 1999.

J.Bloomenthal and K.Shoemake, *Convolution Surfaces*, Proc. SIGGRAPH 1991.

H.Blum, *A Transformation for Extracting New Descriptors of Shape*, Models for the Properties of Speech and Visual Form, Walter-Dunn, ed, MIT Press, 1967.

N.Gagvani, D.Kenchammana-Hosekote, and D.Silver, *Volume Animation Using The Skeleton Tree*, IEEE Symposium on Volume Visualization, Research Triangle Park, NC, October, 1998, pp 47-53.

J.Lander, *A Heaping Pile of Pirate Booty*, Game Developer, v. 8, n. 4, pp. 22-30, April 2001.

F.Lazarus, S.Coquillart and P.Jancene, *Axial Deformations: an Intuitive Deformation Technique*, Computer-Aided Design v. 26, n. 8, 1994.

J.Lewis, M.Cordner and N.Fong, *Pose Space Deformation: a Unified Approach to Shape Interpolation and Skeleton-Driven Deformation*, Proc. SIGGRAPH 2000.

N.Magnenat-Thalmann and D.Thalmann, *Joint-Dependent Local Deformations for Hand Animation and Object Grasping*, Proc. Graphics Interface 1988.

A.Sherstyuk, Kernel Functions in Convolution Surfaces: a Comparative Analysis, The Visual Computer, v. 15, n.4, 1999.

P.Sloan, C.Rose III, and M.Cohen, *Shape by Example*, ACM Symp. Interactive 3D Graphics, 2001.

M.Stalpers and C.van Overveld. *Deforming Geometric Models based on a Polygonal Skeleton Mesh.* Journal of Graphics Tools, v. 2, n. 3, pp 1-14, 1997.

G.Turkiyyah, D.Storti, M.Ganter, H.Chen, and M.Vimawala, *An Accelerated Triangulation Method for Computing the Skeletons of Free-form Solid Models*, Computer-Aided Design, v. 29, n. 1, 1997.

Implicit Surfaces that Interpolate

Greg Turk

College of Computing and GVU Center
Georgia Institute of Technology

What are Interpolating Implicits?

Given: Set of 3D points

Create: Implicit surface passing through these points

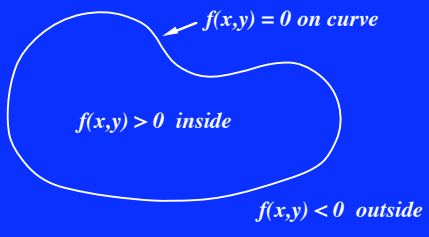
Strengths of Implicit Surfaces

- Well-defined interior and exterior
- Topology changes easy
- Good for
  - Constructive Solid Geometry
  - Ray Tracing
  - Collision Detection
  - Shape Transformation (Morphing)

Implicit Shapes

- Function $f(\boldsymbol{x})$ describes shape
- Function takes on value zero on boundary of shape
- Negative values outside: $f(\boldsymbol{x}) < 0$
- Positive values inside boundary: $f(\boldsymbol{x}) > 0$

Implicit 2D Contour



Implicit Surfaces - Early Work

- James Blinn, 1982 (Blobby molecules)
- Nishimura et al., 1985 (Metaballs)

Blobby Implicit Formulation

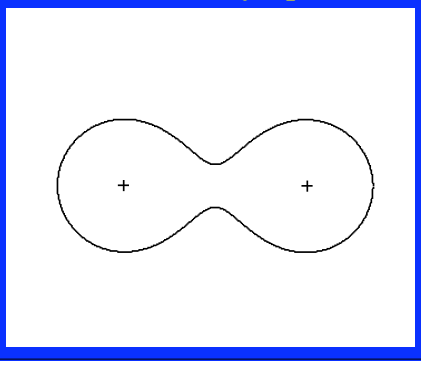
Sum of Gaussian radial basis functions:

$$f(\boldsymbol{x}) = t + \sum_i g_i\left(\left|\boldsymbol{c}_i - \boldsymbol{x}\right|\right)$$

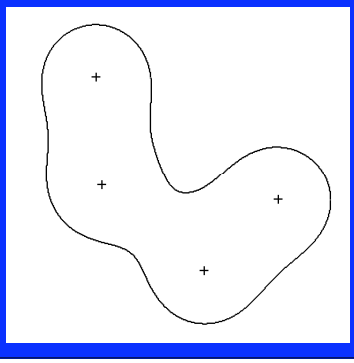
$$g_i(r) = a_i e^{-\frac{r^2}{b_i^2}}$$

$\boldsymbol{c}_i$ is center of Gaussian
a_i and b_i control size of Gaussian
t is global offset

Sum of Blobby Spheres



Sum of Four Blobs

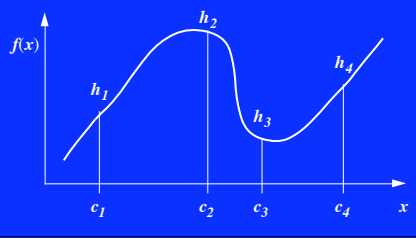


Problems with Blobby Approach

- Indirect specification of surfaces
  - Hard to interpolate points
  - Must decompose object into centers & radii
- Sharp edges and corners hard to capture
- Surfaces have blobby look
- New approach?

Scattered Data Interpolation

Given: Collection of positions $\{\boldsymbol{c}_1, \boldsymbol{c}_2, \ldots \boldsymbol{c}_n\}$
and values $\{h_1, h_2, \ldots, h_n\}$
Create: Smooth function that interpolates all constraints



Thin-Plate Interpolation

- Creates function that exactly passes through constraints
- Works in 2D and 3D
- In 2D, minimizes this energy functional:

$$E(f) = \int_{s\in\Omega} f_{xx}^2(s) + 2f_{xy}^2(s) + f_{yy}^2(s)\,ds$$

Energy functional favors smooth surfaces

Thin-Plate Interpolation

- Exact thin-plate solution is sum of radial basis functions
- Basis functions naturally minimize the energy functional
- In 2D, basis function is $\phi(r) = r^2 \log(r)$
- In 3D, $\phi(r) = r^3$ gives good results

Form of Solution

Solution for 2D problem ($\boldsymbol{x} \in \mathbb{R}^2$):

$$f(\boldsymbol{x}) = \sum w_i \, \phi(\| \boldsymbol{c}_i - \boldsymbol{x} \|) + P(\boldsymbol{x})$$

Where $\phi(r) = r^2 \log(r)$

Weights w_i from solving matrix equation

$P(\boldsymbol{x})$ is a linear polynomial

Simplified Form

Consider: $f(\boldsymbol{x}) = \sum w_i \, \phi(\| \boldsymbol{c}_i - \boldsymbol{x} \|)$

Want $f(\boldsymbol{x})$ to give values h_j at the positions $\boldsymbol{c}_j$:

$$f(\boldsymbol{c}_j) = h_j = \sum w_i \, \phi(\| \boldsymbol{c}_i - \boldsymbol{c}_j \|)$$

Let $\phi_{ij} = \phi_{ji} = \phi(\| \boldsymbol{c}_i - \boldsymbol{c}_j \|)$, then for $i=1$ we have

$$h_1 = w_1 \phi_{11} + w_2 \phi_{12} + w_3 \phi_{13} + \ldots w_n \phi_{1n}$$

Similar for $i = 2, 3, \ldots n$, giving n equations

Matrix Equation

$$\phi_{ij} = \phi\left(\| \boldsymbol{c}_i - \boldsymbol{c}_j \|\right)$$

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \cdots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \cdots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \cdots & \phi_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix}$$

Solve for unknowns w_i

Solving Matrix Equation

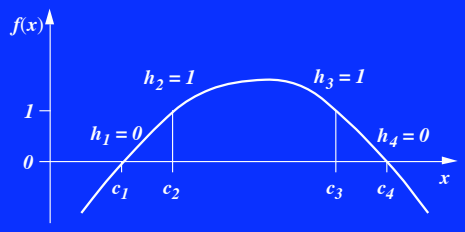
- Can use many solution techniques:
  - LU decomposition
  - SVD
  - Conjugate gradient
- Small systems solved in fraction of second
- Have solved systems with 3,000 constraints
- Large systems require 5-10 minutes

Creating Implicits

- Use thin-plate interpolation!
- Force function to zero on boundary of shape
- Give “hints” about interior/exterior locations

Creating Implicits

Place zero-value constraints on boundary and positive constraints inside.



Boundary Constraints

- Boundary constraints: points through which we want shape to pass
- Each boundary constraint $\boldsymbol{c}_i$ takes on value zero: $f(\boldsymbol{c}_i) = 0$

Interior Constraints

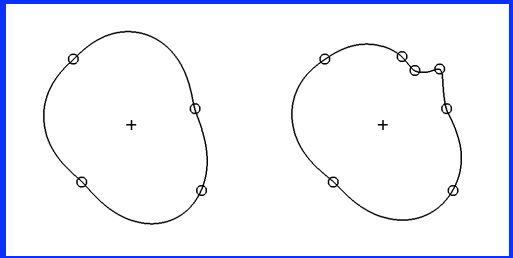
- Interior constraints: hints about what locations should be inside shape
- Interior constraints $\boldsymbol{c}_i$ takes on positive value
- Typically use $f(\boldsymbol{c}_i) = 1$

Creating Implicit Function

- Collect boundary and interior constraints together (locations and values)
- Give to thin-plate solver
- Result is function $f(\boldsymbol{x})$ that is zero exactly on boundary constraints
- Creates implicit function that exactly interpolates our points!

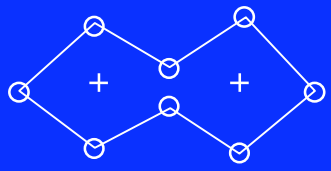
2D Examples

Circles = boundary constraints
Plus = interior constraints

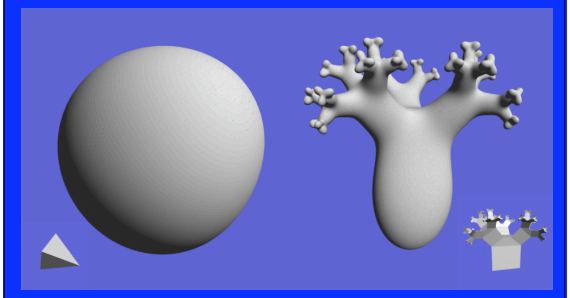


3D Surfaces

- Constriants are 3D points
- Vertices of polyhedron for boundary constraints
- Interior constraints are placed by human



3D Surfaces



History of this Work

First paper on interpolating implicits:

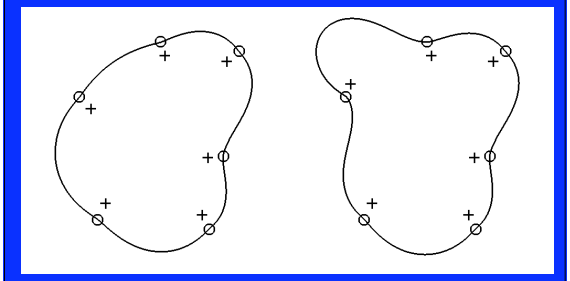
"Function Representation of Solids Reconstructed from Scattered Surface Points and Contours"
V. Savchenko, A. Pasko, O. Okunev, and T. L. Kunni
Computer Graphics Forum, October 1995

Re-discovered: Turk & O'Brien shape interpolation paper, SIGGRAPH 99

Normal Constraints

- Can place interior constraints near each boundary constraint
- Each interior constraint is "hint" about surface normal
- Call these "normal constraints"

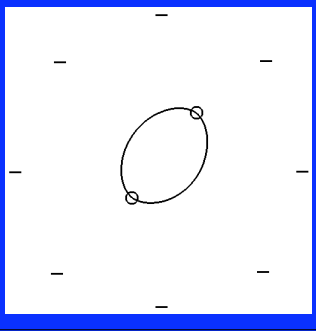
Normal Constraints



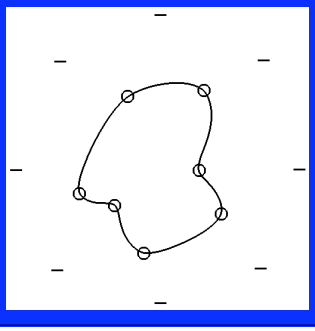
Exterior Constraints

- Can specify exterior constraints instead
- Function takes on negative values at exterior constraints, typically $f(\boldsymbol{c}_i) = -1$
- Useful for free-form modelling

Exterior Constraints



Exterior Constraints

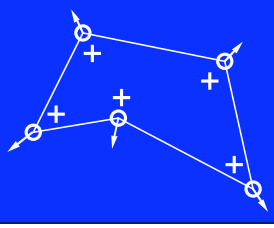


Kinds of Constraints

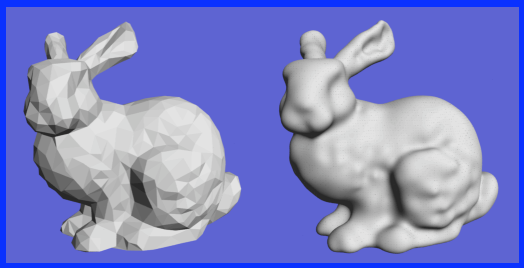
- Boundary constraints are exactly interpolated by the surface
- At least one other kind of constraint needed:
  - Interior constraint
  - Exterior constraint
  - Normal constraint
- Different applications use different constraint types

Implicits from Polygons

- Can create implicit surface directly from polygonal data
- Each vertex is boundary constraint
- Normal constraint from surface normals at vertices



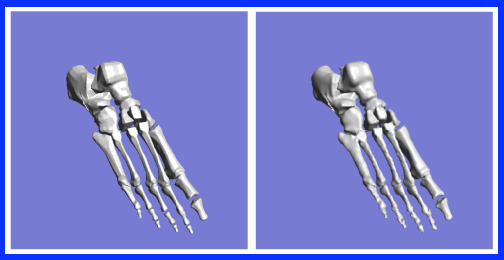
Implicit from Polygons



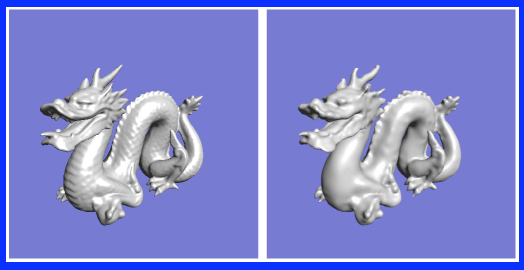
Complex Surfaces from Polygons

- Basic method good for uniformly tiled objects
- More complex surfaces (Gary Yngve):
  - Pick a few constraints, create surface
  - Measure deviation to polygonal surface
  - Iteratively add constraints until close enough
- TVCG paper has details (in course notes)

Complex Implicit Surfaces



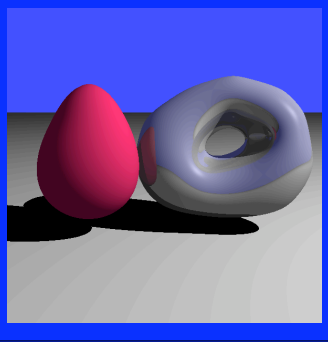
Complex Implicit Surfaces



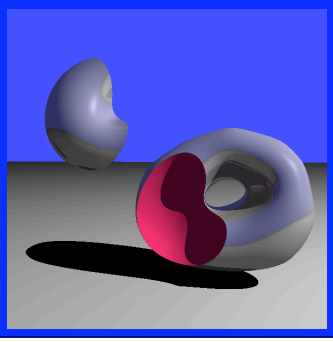
Rendering

- Standard methods of rendering implicits will work
- Two obvious choices:
  - Iso-surface extraction (Jules Bloomenthal's continuation method)
  - Ray tracing (John Hart's sphere tracing)

Raytraced Implicits



Constructive Solid Geometry



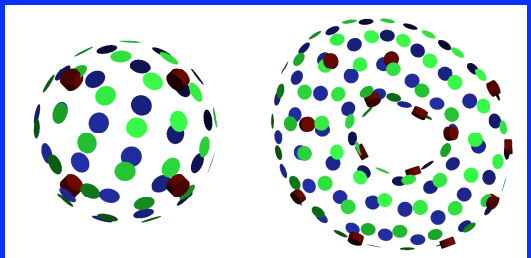
3D Modelling

- Specify sphere of exterior constraints
- Begin with four boundary constraints (tetrahedral arrangement)
- User options
  - Move boundary constraint
  - Add new boundary constraint
  - Create and move normal constraint

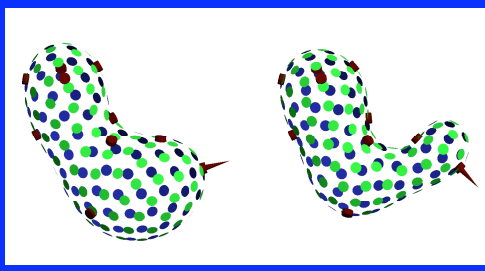
Witkin & Heckbert Floaters

- We use Witkin & Heckbert's interactive method of display
- Surface represented as oriented particles, or "floaters"
- Do NOT use their control particles
  - They have to specify which basis functions are fixed and which move
  - Result in "slippery" implicit surfaces

Interactive Modelling



Changing Surface Normals



Adding New Constraints

- When user clicks on surface, new boundary constraint is added
- Does not affect surface at all
- Can then move new constraint
- Similar to knot insertion for splines
- Easier to use than Witkin & Heckbert control particles

Strengths of Interpolating Implicits

- Direct control of surface position and normals
- Addition of new points like knot insertion
- Implicit surfaces from polygonal data
- Can handle noisy or sparse data
- Variable smoothness parameters

Future Challenges

- Speed, Large numbers of constraints (already work on this)
- Exact normal constraints
- Creating open surfaces (surfaces with boundaries)
- Higher-level modelling tools
- Real-time surface display for modelling

Acknowledgements

- Funded by ONR and NSF
- Thanks to colleagues at Georgia Institute of Technology

Shape Transformation Using Variational Implicit Functions

Greg Turk James F. O'Brien

Georgia Institute of Technology

Abstract

Traditionally, shape transformation using implicit functions is performed in two distinct steps: 1) creating two implicit functions, and 2) interpolating between these two functions. We present a new shape transformation method that combines these two tasks into a single step. We create a transformation between two N-dimensional objects by casting this as a scattered data interpolation problem in $N+1$ dimensions. For the case of 2D shapes, we place all of our data constraints within two planes, one for each shape. These planes are placed parallel to one another in 3D. Zero-valued constraints specify the locations of shape boundaries and positive-valued constraints are placed along the normal direction in towards the center of the shape. We then invoke a variational interpolation technique (the 3D generalization of thin-plate interpolation), and this yields a single implicit function in 3D. Intermediate shapes are simply the zero-valued contours of 2D slices through this 3D function. Shape transformation between 3D shapes can be performed similarly by solving a 4D interpolation problem. To our knowledge, ours is the first shape transformation method to unify the tasks of implicit function creation and interpolation. The transformations produced by this method appear smooth and natural, even between objects of differing topologies. If desired, one or more additional shapes may be introduced that influence the intermediate shapes in a sequence. Our method can also reconstruct surfaces from multiple slices that are not restricted to being parallel to one another.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—surfaces and object representations

Keywords: Shape transformation, shape morphing, contour interpolation, implicit surfaces, thin-plate techniques.

1 Introduction

The shape transformation problem can be stated as follows: Given two shapes A and B, construct a sequence of intermediate shapes so that adjacent pairs in the sequence are geometrically close to one another. Playing the resulting sequence of shapes as an animation would show object A deforming into object B. Sequences of 2D shapes can be thought of as slices through a 3D surface, as shown in Figure 1. Shape transformation can be performed between objects of any dimension, although 2D and 3D shapes are by far the most common cases. Shape transformation has applications in medicine, computer aided design, and special effects creation. We give an overview of these three applications below.

One important application of shape transformation in medicine is contour interpolation. Non-invasive imaging techniques often collect data about a patient's internal anatomy in "slices" of a particular size such as 512×512 samples. Usually many fewer slices are taken along the third dimension so that a resulting volume might, for example, be sampled at $512 \times 512 \times 30$ resolution. To reconstruct a 3D model of a particular organ, the samples are segmented to create shapes (contours) within the slices. Intermediate shapes are then created between slices in the sparsely sampled dimension. The reconstructed 3D object is formed by stacking together the original and the interpolated contours. This is an example of 2D shape transformation.

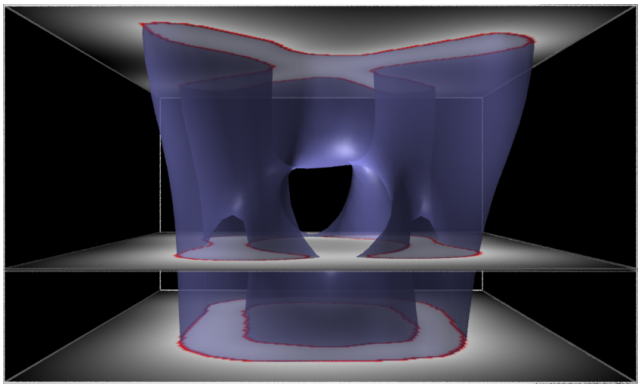


Figure 1: Visualization of transformation between X and O shapes. Top and bottom planes contain constraints for the two shapes. Translucent surface is the isosurface of a 3D variational implicit function, and slices through it give intermediate shapes.

Shape transformation can also be a useful tool in computer aided geometric design. Consider the problem of creating a join between two metal parts with different cross-sections. It is important for the connecting surface to be smooth because those places with sharp ridges or creases are the locations that are most likely to form cracks. The intermediate surface joining the two parts can be created using shape transformation, much in the same way as with contour interpolation for medical imaging. Because of the smoothness properties of variational interpolation methods, we consider them a natural tool to explore for shape transformation in CAD.

Finally, animated shape transformations have been used to create dramatic special effects for feature films and commercials. One of the best-known examples of shape transformation is in the film *Terminator 2*. In this film, a cyborg policeman undergoes a number of transformations from an amorphous and highly reflective surface to various destination shapes. 2D image morphing would not have accurately modeled the reflection of the environment off the surface of the deforming cyborg, hence tailor-made 3D shape transformation programs were used for these effects [9].

In this paper we use variational interpolation in a new way to produce high-quality shape transformations that may be used for any of the previously mentioned applications. Our method allows a user to control the transformation in several ways, and it is general enough to produce transformations between shapes of any topology.

2 Previous Work

Most shape transformation techniques can be placed into one of two categories: parametric correspondence methods and implicit

turk@cc.gatech.edu, job@acm.org.

Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1999), August 1999, pp. 335-342. © 1999 ACM reprinted with permission.

function interpolation. Parametric methods are typically faster to compute and require less memory because they operate on a lower-dimensional representation of an object than do implicit function methods. Unfortunately, transforming between objects of different topologies is considerably more difficult with parametric methods. Parametric approaches also can suffer from problems with self-intersecting surfaces, but this is never a problem with implicit function methods. Techniques that use implicit function interpolation gracefully handle changes in topology between objects and do not create self-intersecting surfaces.

A parametric correspondence approach to shape transformation attempts to find a "reasonable" correspondence between pairs of locations on the boundaries of the two shapes. Intermediate shapes are then created by computing interpolated positions between the corresponding pairs of points. Many shape transformation techniques have been created that follow the parametric correspondence approach. One early application of this approach is the method of contour interpolation described by Fuchs, Kedem and Uselton [10]. Their method attempts to find an "optimal" (minimum-area) triangular tiling that connects two contours using dynamic programming. Many subsequent techniques followed this approach of defining a quality measure for a particular correspondence between contours and then invoking an optimization procedure [22, 25]. There have been fewer examples of using parametric correspondence for 3D shape transformation. One quite successful 3D parametric method is the work of Kent et al. [17]. The key to their approach is to subdivide the polygons of the two models in a manner that creates a correspondence between the vertices of the two models. More recently, Gregory and co-workers created a similar method that also allows a user to specify region correspondences between meshes to better control a transformation [12].

An entirely different approach to shape transformation is to create an implicit function for each shape and then to smoothly interpolate between these two functions. A shape is defined by an implicit function, $f(\mathbf{x})$, as the set of all points $\mathbf{x}$ such that $f(\mathbf{x}) = 0$. For contour interpolation in 2D, the implicit function can be thought of as a height field over a two-dimensional domain, and the boundary of a shape is the one-dimensional curve defined by all the points that have the same elevation value of zero. An implicit function in 3D is a function that yields a scalar value at every point in 3D. The shape described by such a function is given by those places in 3D whose function value is zero (the isosurface).

One commonly used implicit function is the *inside/outside function* or *characteristic function*. This function takes on only two values over the entire domain. The two values that are typically used are zero to represent locations that are outside and one to signify positions that are inside the given shape. Given a powerful enough interpolation technique, the characteristic function can be used for creating shape transformations. Hughes presented a successful example of this approach by transforming characteristic functions into the frequency domain and performing interpolation on the frequency representations of the shapes [15]. Kaul and Rossignac found that smooth intermediate shapes can be generated by using weighted Minkowski sums to interpolate between characteristic functions [16]. They later created a generalization of this technique that can use several intermediate shapes to control the interpolation between objects [24]. Using a wavelet decomposition of a characteristic function allowed He and colleagues to create intermediates between quite complex 3D objects [13].

A more informative implicit function can provide excellent intermediate shapes even if a simple interpolation technique is used. In particular, the *signed distance function* (sometimes called the *distance transform*) is an implicit function that gives very plausible intermediate shapes even when used with simple linear interpolation of the function values of the two shapes. The value of the signed distance function at a point $\mathbf{x}$ inside a given shape is just the Euclidean distance between $\mathbf{x}$ and the nearest point on the boundary of the shape. For a point $\mathbf{x}$ that is outside the shape, the signed distance function takes on the negative of the distance from $\mathbf{x}$ to the closest point on the boundary.

Several researchers have used the signed distance function to interpolate between 2D contours [19, 14]. The distance function for each given shape is represented as a regular 2D grid of values, and an intermediate implicit function is created by linear interpolation between corresponding grid values of the two implicit functions. Each intermediate shape is given by the zero iso-contour of this interpolated implicit function. In contrast to the global interpolation methods described above (frequency domain, wavelets, Minkowski sum), this interpolation is entirely local in nature. Nevertheless, the shape transformations that are created by this method are quite good. In essence, the information that the signed distance function encodes (distance to nearest boundary) is enough to make up for the purely local method of interpolation. Payne and Toga were the first to transform three dimensional shapes using this approach [23]. Cohen-Or and colleagues gave additional control to this same approach by combining it with a warping technique in order to produce shape transformations of 3D objects [7].

Our approach to shape transformation combines the two steps of building implicit functions and interpolating between them. To our knowledge, it is the only method to do so. The remainder of this paper describes how variational interpolation can be used to simultaneously solve these two tasks.

3 Variational Interpolation

Our approach relies on *scattered data interpolation* to solve the shape transformation problem. The problem of scattered interpolation is to create a smooth function that passes through a given set of data points. The two-dimensional version of this problem can be stated as follows: Given a collection of k constraint points $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ that are scattered in the plane, together with scalar height values at each of these points $\{h_1, h_2, \ldots, h_k\}$, construct a smooth surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function $f(\mathbf{x})$ so that $f(\mathbf{c}_i) = h_i$, for $1 \leq i \leq k$.

One common approach to solving scattered data problems is to use variational techniques (from the calculus of variations). This approach begins with an energy that measures the quality of an interpolating function and then finds the single function that matches the given data points and that minimizes this energy measure. For two-dimensional problems, thin-plate interpolation is the variational solution when using the following energy function E:

$$E = \int_{\Omega} f_{xx}^2(\mathbf{x}) + 2 f_{xy}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x}) \tag{1}$$

The notation f_{xx} means the second partial derivative in the x direction, and the other two terms are similar partial derivatives, one of them mixed. The above energy function is basically a measure of the aggregate squared curvature of $f(\mathbf{x})$ over the region of interest Ω. Any creases or pinches in a surface will result in a larger value of E. A smooth surface that has no such regions of high curvature will have a lower value of E. The thin-plate solution to an interpolation problem is the function $f(\mathbf{x})$ that satisfies all of the constraints and that has the smallest possible value of E.

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points $\mathbf{c}_i$ are positions in N-dimensions rather than in 2D, this is called the N-dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for other dimensions. In this paper we will perform interpolation in two, three, four and five dimensions. Because the term *thin-plate* is only meaningful for 2D problems, we will use *variational interpolation* to mean the generalization of thin-plate techniques to any number of dimensions.

The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function, $f(\mathbf{x})$, that will minimize equation 1 subject to the interpolation constraints $f(\mathbf{c}_i) = h_i$. Equation 1 can be solved using weighted sums of the

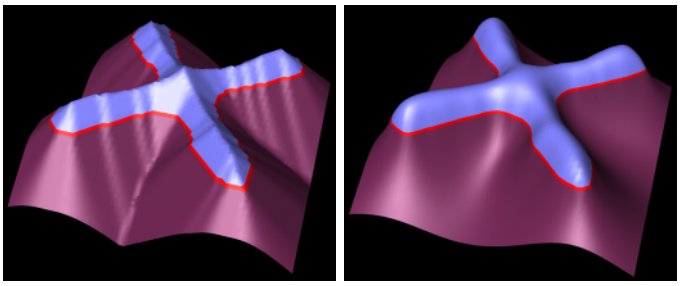


Figure 2: Implicit functions for an X shape. Left shows the signed distance function, and right shows the smoother variational implicit function.

radial basis function $\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|)$. The family of variational problems that includes equation 1 was studied by Duchon [8].

Using the appropriate radial basis function, we can then express the interpolation function as

$$f(\mathbf{x}) = \sum_{j=1}^{n} d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \tag{2}$$

In the above equation, $\mathbf{c}_j$ are the locations of the constraints, the d_j are the weights, and $P(\mathbf{x})$ is a degree one polynomial that accounts for the linear and constant portions of f. Because the thin-plate radial basis function naturally minimizes equation 1, determining the weights, d_j, and the coefficients of $P(\mathbf{x})$ so that the interpolation constraints are satisfied will yield the desired solution that minimizes equation 1 subject to the constraints. Furthermore, the solution will be an exact analytic solution, and is not subject to approximation and discretization errors that may occur when using finite element or finite difference methods.

To solve for the set of d_j that will satisfy the interpolation constraints $h_i = f(\mathbf{c}_i)$, we can substitute the right side of equation 2 for $f(\mathbf{c}_i)$, which gives:

$$h_i \quad = \quad \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}) \tag{3}$$

Since this equation is linear with respect to the unknowns, d_j and the coefficients of $P(\mathbf{x})$, it can be formulated as a linear system. For interpolation in 3D, let $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \dots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The above system is symmetric and positive semi-definite, so there will always be a unique solution for the d_j and p_j [11]. For systems with up to a few thousand constraints, the system can be solved directly with a technique such as symmetric LU decomposition. We used symmetric LU decomposition to solve this system for all of the examples shown in this paper.

Using the tools of variational interpolation we can now turn our attention to creating implicit functions for shape transformation.

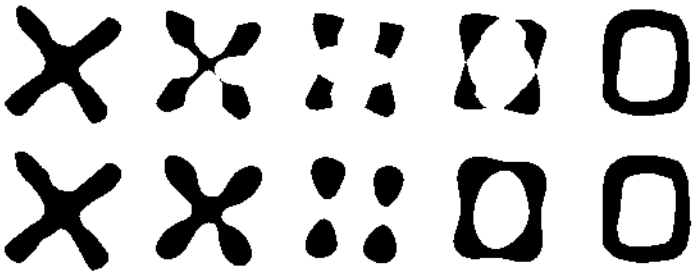


Figure 3: Upper row is a shape transformation created using the signed distance transform. Lower row is the sequence generated using a single variational implicit function.

4 Smooth Implicit Function Creation

In this section we will lay down the groundwork for shape transformation by discussing the creation of smooth implicit functions for a single shape. In particular, we will use variational interpolation of scattered constraints to construct implicit functions. Later we will generalize this to create functions that perform shape transformation.

Let us first examine the signed distance transformation because it is commonly used for shape transformation. The left half of Figure 2 shows a height field representation of the signed distance function of an X shape. The figure shows sharp ridges (the *medial axis*) that run down the middle of the height field. Ridges appear in the middle of shapes where the points are equally distant from two or more boundary points of the original shape. The values of a signed distance function decrease as one moves away from the ridge towards the boundaries. Figure 3, top row, shows a shape interpolation sequence between an X and an O shape that was created by linear interpolation between two signed distance functions. Note the pinched portions of some of the intermediate shapes. These sharp features are not isolated problems, but instead persist over many intermediate shapes. The cause of these pinches are the sharp ridges of signed distance functions. In many applications such artifacts are undesirable. In medical reconstruction, for example, these pinches are a poor estimate of shape because most biological structures have smooth surfaces. Because of this, we seek implicit functions that are continuous and that have a continuous first derivative.

4.1 Variational Implicit Functions in 2D

We can create smooth implicit functions for a given shape using variational interpolation. This can be done both for 2D and 3D shapes, although we will begin by discussing the 2D case. In this approach, we create a closed 2D curve by describing a number of locations through which the curve will pass and also specifying a number of points that should be interior to the curve. We call the given points on the curve the *boundary constraints*. The boundary constraints are locations at which we require our implicit function to take on the value of zero. Paired with each boundary constraint is a *normal constraint*, which is a location at which the implicit function is required to take on the value one. (Actually, any positive value could be used.) The locations of the normal constraints should be towards the interior of the desired curve, and also the line passing through the normal constraint and its paired boundary constraint should be parallel to the desired normal to the curve. The collection of boundary and normal constraints are passed along to a variational interpolation routine as the scattered constraints to be interpolated. The function that is returned is an implicit function that describes our curve. The curve will exactly pass through our boundary constraints.

Figure 4 (left) illustrates eight such pairs of constraints in the plane, with the boundary constraints shown as circles and the normal constraints as plus signs. When we invoke variational interpo-

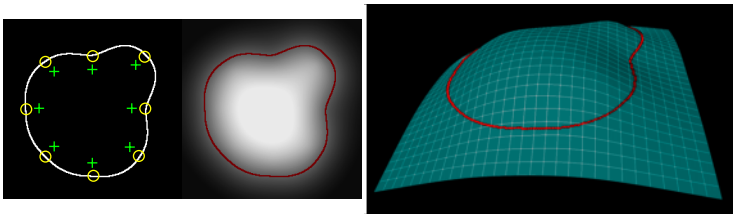


Figure 4: At left are pairs of boundary and normal constraints (circles and pluses). The middle image uses intensity to show the resulting variational implicit function, and the right image shows the function as a height field.

lation with such constraints, the result is a function that takes on the value of zero exactly at our zero-value constraints and that is positive in the direction of our normal constraints (towards the interior of the shape). The closed curve passing through the zero-value constraints in Figure 4 (middle) is the iso-contour of the implicit function created by this method. Figure 4 (right) shows the resulting implicit function rendered as a height field. Given enough suitably-placed boundary constraints we can define any closed shape. We call an implicit function that is created in this manner a *variational implicit function*. This new technique for creating implicit functions also show promise for surface modeling, a topic that is explored in [27].

We now turn our attention to defining boundary and normal constraints for a given 2D shape. Assume that a given shape is represented as a gray-scale image. White pixels represent the interior of a shape, black pixels will be outside the shape, and pixels with intermediate gray values lie on the boundary of the shape. Let m be the middle gray value of our image's gray scale range. Our goal is to create constraints between any two adjacent pixels where one pixel's value is less than m and the other's value is greater. Identifying these locations is the 2D analog of finding the vertex locations in a 3D marching cubes algorithm [21].

We traverse the entire gray-scale image and examine the east and south neighbor of each pixel $I(x,y)$. If $I(x,y) < m$ and either neighbor has a value greater than m, we create a boundary constraint at a point along the line segment joining the pixel centers. A boundary constraint is also created if $I(x,y) > m$ and either neighbor takes on a value less than m. The value of the constraint is zero, and we set the position of the constraint at the location between the two pixels where the image would take on the value of m if we assume linear interpolation of pixel values. Next, we estimate the gradient of the gray scale image using linear interpolation of pixel values and central differencing. We then create a normal constraint a short distance away from the zero crossing in the direction of the gradient. We have found that a distance of a pixel's width between the boundary and normal constraints works well in practice. Figure 2 (right) shows the implicit function for an X shape that was created using variational interpolation from such constraints. It is smooth and free of sharp ridges.

4.2 Variational Implicit Functions in 3D

We can create implicit functions for 3D surfaces using variational interpolation in much the same way as for 2D shapes. Specifically, we can derive 3D constraints from the vertex positions and surface normals of a polygonal representation of an object. Let (x,y,z) and (n_x,n_y,n_z) be the position and the surface normal at a vertex, respectively. Then a boundary constraint is placed at (x,y,z) and a normal constraint is placed at $(x-kn_x, y-kn_y, z-kn_z)$, where k is some small value. We use a value of $k = 0.01$ for models that fit within a unit cube for the results shown in this paper. All of the 3D models that we transform in this paper were constructed by building an implicit function in this manner. Note that we can use this method to build an implicit function whenever we have a collection of points and normals— polygon connectivity is not necessary.

Now that we can construct smooth implicit functions for both two- and three-dimensional shapes, we turn our attention to shape transformation. It would be possible to create variational implicit functions for each of two given shapes and then linearly interpolate between these functions to create a shape transformation sequence. Instead, however, we will examine an even better way of performing shape transformation by generalizing the implicit function building methods of this section.

5 Unifying Function Creation and Interpolation

The key to our shape transformation approach is to represent the entire sequence of shapes with a single implicit function. To do so, we need to work in one higher dimension than the given shapes. For 2D shapes, we will construct an implicit function in 3D that represents our two given shapes in two distinct parallel planes. This is actually simple to achieve now that we know how to use scattered data interpolation to create an implicit function.

5.1 Two-Dimensional Shape Transformation

Given two shapes in the plane, assume that we have created a set of boundary and normal constraints for each shape, as described in Section 4. Instead of using each set of constraints separately to create two different 2D implicit functions, we will embed all of the constraints in 3D. We do this by adding a third coordinate value to the location of each boundary and normal constraint. For those constraints for the first shape, we set the new coordinate t for all constraints to $t = 0$. For the second shape, all of the new coordinate values are set to $t = t_{max}$ (some non-zero value). Although we have added a third dimension to the locations of the constraints, the values that are to be interpolated remain unchanged for all constraints.

Once we have placed the constraints of both shapes into 3D, we invoke 3D variational interpolation to create a single scalar-valued function over $\mathbf{R}^3$. If we take a slice of this function in the plane $t = 0$, we find an implicit function that takes on the value zero exactly at the boundary constraints for our first shape. In this plane, our function describes the first shape. Similarly, in the plane $t = t_{max}$ this function gives the second shape. Parallel slices at locations between these two planes ($0 < t < t_{max}$) represent the shapes of our shape transformation sequence. Figure 1 illustrates that the collection of intermediate shapes are all just slices through a surface in 3D that is created by variational interpolation.

Figure 3 (bottom) shows the sequence of shapes created using this variational approach to shape transformation. Topology changes (e.g. the addition or removal of holes) come "for free", without any human guidance or algorithmic complications. Notice that all of the intermediate shapes have smooth boundaries, without pinches. Sharp features can arise only momentarily when there is a change in topology such as when two parts join. Figure 5 shows two more shape transformations that use this approach and that also incorporate warping. Warping is an another degree of control that may be added to any shape transformation technique, and is in fact

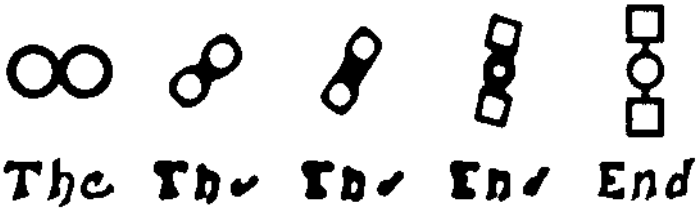


Figure 5: Two shape transformation sequences (using the variational implicit technique) that incorporate warping.

an orthogonal issue to those of implicit function creation and interpolation. Although it is not a focus of our research, for completeness we briefly describe warping in the appendix.

Why has this implicit function building method not been tried using other ways of creating implicit functions? Why not, for example, build a signed distance function in one higher dimension? Because a *complete* description of an object's boundary is required in order to build a signed distance function. When we embed our two shapes into a higher dimension, we only know a *piece* of the boundary of our desired higher-dimensional shape, namely the cross-sections that match the two given objects. In contrast, a complete boundary representation is *not* required when using variational interpolation to create an implicit function. Variational interpolation creates plausible function values in regions where we have no information, and especially in the "unknown" region between the two planes that contain all of our constraints. This plausibility of values comes from the smooth nature of the functions that are created by the variational approach.

5.2 Three-Dimensional Shape Transformation

Just as we create a 3D function to create a transformation between 2D shapes, we can move to 4D in order to create a sequence between 3D shapes. We perform shape interpolation between two 3D objects using boundary and normal constraints for each shape. We place the constraints from two 3D objects into four dimensional space, just as we placed constraints from 2D contours into 3D. Similar to contour interpolation, the constraints are separated from one another in the fourth dimension by some specified distance. We place all the constraints from the first object at $t = 0$, and the constraints from the second object are placed at $t = t_{max}$, where t_{max} is the given separation distance. We then create a 4D implicit function using variational interpolation. An intermediate shape between the two given shapes is found by extracting the isosurface of a 3D "slice" (actually a volume) of the resulting 4D function.

Figure 6 shows two 3D shape transformation sequence that were constructed using this method. To extract these surfaces we use code published by Bloomenthal that begins at a seed location on the surface of a model and only evaluates the implicit function at points near previously visited locations [4]. This is far more efficient than sampling an entire volume of the implicit function and then extracting an isosurface from the volume. The matrix solution for the transformation sequence of Figure 6 (left) required 13.5 minutes, and each isosurface in the sequence took approximately 2.3 minutes to generate on an SGI Indigo2 with a 195 MHz R10000 processor. Figure 6 (right) shows a transformation between 3D shapes that used warping to align features.

6 Surface Reconstruction from Contours

So far we have only considered shape transformation between pairs of objects. In medical reconstruction, however, it is often necessary to create a surface from a large number of parallel 2D slices. Can't we just perform shape interpolation between pairs of slices and stack the results together to create one surface in 3D? Although this method will create a continuous surface, it is almost certain to produce a shape that has surface normal discontinuities at the planes of the original slices. In the plane of slice i, the surface created between slice pairs $i-1$ and i will usually not agree in surface normal with the surface created between slices i and $i+1$. Nearly all contour interpolation methods consider only pairs of contours at any one time, and thus suffer from such discontinuities. (A notable exception is [1]).

To avoid discontinuities in surface normal, we must use information about more than just two slices at a given time. We can accomplish this using a generalization of the variational approach to shape transformation. Assume that we begin with k sets of constraints, one set for each 2D data slice. Instead of considering the contours in pairs, we place the constraints for all of the k slices into 3D simultaneously. Specifically, the constraints of slice i are placed in the plane $z = si$, where s is the spacing between planes. Once the constraints from *all* slices have been placed in 3D, we invoke

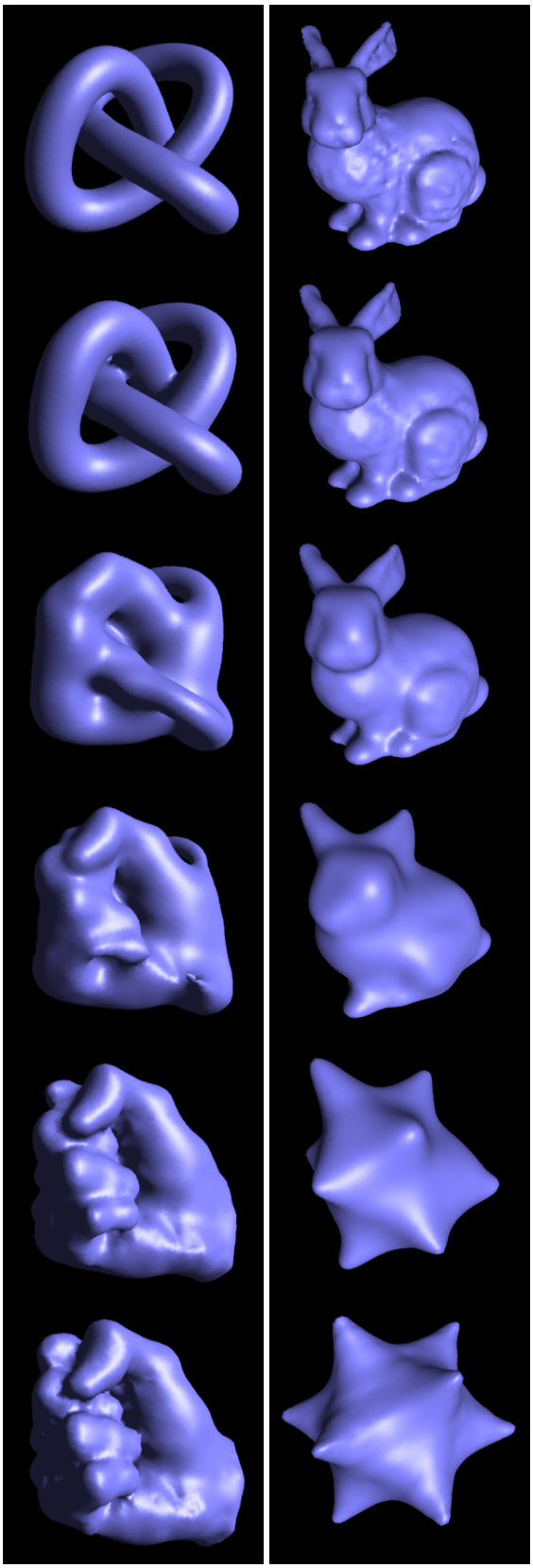


Figure 6: 3D shape transformation sequences.

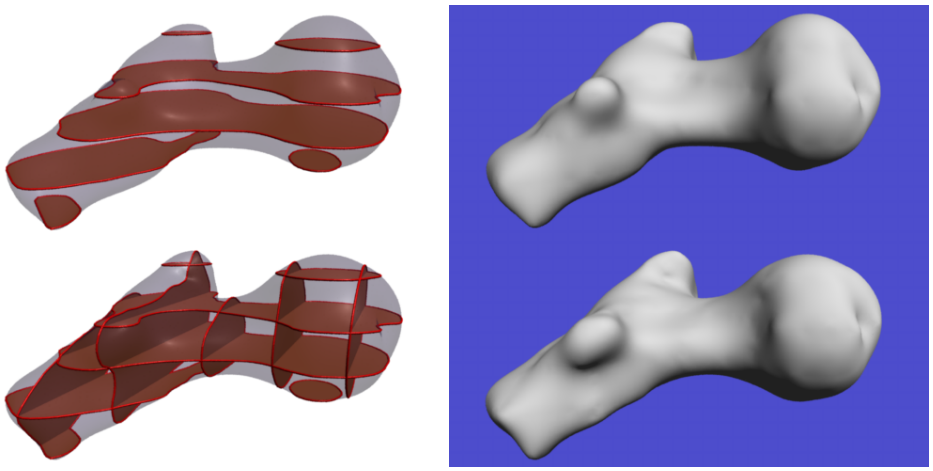


Figure 7: Reconstruction of hip joint from contours. Top row shows the five parallel slices used and the final surface. Bottom row shows intersecting contours and the more detailed surface that is created.

variational interpolation *once* to create a single implicit function in 3D. The zero-valued isosurface exactly passes through each contour of the data. Due to the smooth nature of variational interpolation, the gradient of the implicit function is everywhere continuous. This means that surface normal discontinuities are rare, appearing in pathological situations when the gradient vanishes such as when two features just barely touch. Figure 7 (top row) illustrates the result of this contour interpolation approach. The hip joint reconstruction in the upper right was created from the five slices shown at the upper left.

A side benefit of using the variational implicit function method is that it produces smoothly rounded caps on the ends of surfaces. Notice that in Figure 7 (top left) that the reconstructed surface extends beyond the constraints in the positive and negative z direction (the direction of slice stacking). This "rounding off" of the ends is a natural side effect of variational interpolation, and need not be explicitly specified.

6.1 Non-Parallel Contours

In the previous section, we only considered placing constraints within planes that are all parallel to one another. There is nothing special about any particular set of planes, however, once we are specifying constraints in 3D. We can mix together constraints that are taken from planes at any angle whatsoever, so long as we know the relative positions of the planes (and thus the constraints). Most contour interpolation procedures cannot integrate data taken from slices in several directions, but the variational approach allows complete freedom in this regard. Figure 7 (lower row) shows several contours that are placed perpendicular to one another, and the result of using variational interpolation on the group of constraints from these contours.

6.2 Between-Contour Spacing

Up to this point we have not discussed the separating distance s between the slices that contain the contour data. This separating distance has a concrete meaning in medical shape reconstruction from 2D contours. Here we know the actual 3D separation between the contours from the data capture process. This "natural" distance is the separating distance s that should be used when reconstructing the surface using variational interpolation. Upon reflection, it is odd that some contour interpolation methods do not make use of the data capture distance between slices. In some cases a medical technician will deliberately vary the spacing between data slices in order to capture more data in a particular region of interest. Using variational interpolation, we may incorporate this information about varying separation distances into the surface reconstruction process.

For both special effects production and for computer aided design, the distance between the separating planes can be thought of as a control knob for the artist or designer. If the distance is small, only pairs of features from the two shapes that are very close to one another will be preserved through all the intermediate shapes. If the separation distance is large, the intermediate shape is guided by more global properties of the two shapes. In some sense, the separating distance specifies whether the shape transformation is local or global in nature. The separation distance is just one control knob for the user, and in the next section we will explore another user control.

7 Influence Shapes

In this section we present a method of controlling shape transformation by introducing an *influence shape*. The idea to use additional objects as controls for shape transformation was introduced by Rossignac and Kaul [24]. Such intermediate shape control can be performed in a natural way using variational interpolation. The key is to step into a still higher dimension when performing shape transformation.

Recall that to create a transformation sequence between two given shapes we added one new dimension, called t earlier. We can think of the two shapes as being two points that are separated along the t dimension, and these two points are connected by a line segment that joins the two points along this dimension. If we begin with three shapes, however, we can in effect place them at the three points of a triangle. In order to do so we need not just one additional dimension but two, call them s and t.

As an example, we may begin with three different 3D shapes A, B and C. To each constraint that describes one of the shapes, we can add two new coordinates, s and t. Constraints from shape A at (x,y,z) are placed at $(x,y,z,0,0)$, constraints from shape B are placed at $(x,y,z,1,0)$ and shape C constraints are placed at $(x,y,z,1/2,1/2)$. Variational interpolation based on these 5-dimensional constraints results in a 5D implicit function. Three-dimensional slices of this function along the s-dimension between 0 and 1 are simply shape sequences between shapes A and B when the t-dimension value is fixed at zero. If, however, the t-dimension value is allowed to become positive as s varies from 0 to 1, then the intermediate shapes will take on some of the characteristics of shape C. In fact, the 5D implicit function actually captures an entire family of shapes that are various blends between the three shapes. Figure 8 illustrates some members of such a family of shapes.

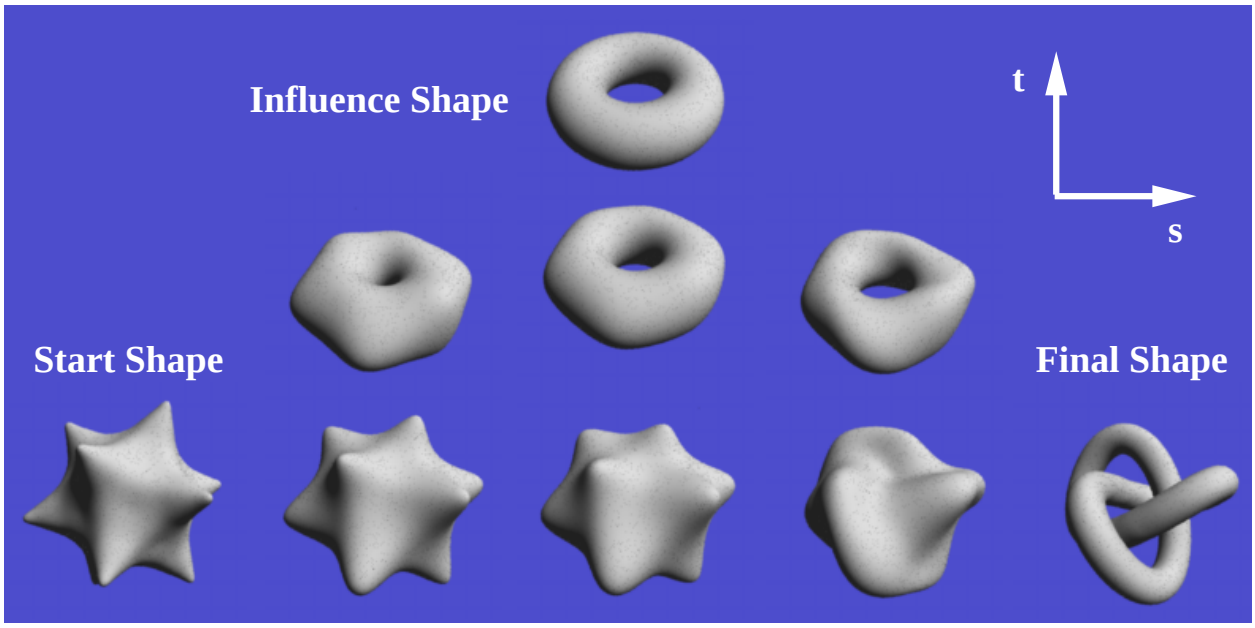


Figure 8: Sequence between star and knot can be influenced by a torus (the influence shape) if the path passes near the torus in the five-dimensional space.

There is no reason to stop at three shapes. It is possible to place four shapes at the corners of a quadrilateral, five shapes around a pentagon, and so on. If we wish to use four shapes, then placing the constraints at the corners of a quadrilateral using two additional dimensions would not allow us to produce a shape that was arbitrary mixtures between the shapes. In order to do so, we can place the constraints in yet a higher dimension, in effect placing the four shapes at the corners of a tetrahedron in $N+3$ dimensions, where N is the dimension of the given shapes.

There are two related themes that guide our technique for shape transformation. The first is that shape transformation should be thought of as a shape-creation problem in a higher dimension. The second theme is that better shape transformation sequences are produced when all of the problem constraints are solved simultaneously— in our case by using variational interpolation. Influence shapes are the result of taking these ideas to an extreme.

8 Conclusions and Future Work

Our new approach uses variational interpolation to produce one implicit function that describes an entire sequence of shapes. Specific characteristics of this approach include:

- Smooth intermediate shapes
- Shape transformation in any number of dimensions
- Analytic solutions that are free of polygon and voxel artifacts
- Continuous surface normals for contour interpolation
- Contour slices may be at any orientation, even intersecting

This approach provides two new controls for creating shape transformation sequences:

- Separation distance gives local/global interpolation tradeoff
- May use influence shapes to control a transformation

The approach we have presented in this paper re-formulates the shape interpolation problem as an interpolation problem in one higher dimension. In essence, we treat the "time" dimension just like another spatial dimension. We have found that using the variational interpolation method produces excellent results, but the mathematical literature abounds with other interpolation methods. An exciting avenue for future work is to investigate what other interpolation techniques can also be used to create implicit functions for shape transformation. Another issue is whether shape transformation methods can be made fast enough to allow a user interactive control. Finally, how might surface properties such as color and texture be carried through intermediate objects?

9 Acknowledgements

This work owes a good deal to Andrew Glassner for getting us interested in the shape transformation problem. We thank our colleagues and the anonymous reviewers for their helpful suggestions. This work was funded by ONR grant N00014-97-1-0223.

References

[1] Barequet, Gill, Daniel Shapiro and Ayellet Tal, "History Consideration in Reconstructing Polyhedral Surfaces from Parallel Slices," *Proceedings of Visualization '96*, San Francisco, California, Oct. 27 – Nov. 1, 1996, pp. 149–156.

[2] Barr, Alan H., "Global and Local Deformations of Solid Primitives," *Computer Graphics*, Vol. 18, No. 3 (SIGGRAPH 84), pp. 21–30.

[3] Beier, Thaddeus and Shawn Neely, "Feature-Based Image Metamorphosis," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 35–42.

[4] Bloomenthal, Jules, "An Implicit Surface Polygonizer," in *Graphics Gems IV*, edited by Paul S. Heckbert, Academic Press, 1994, pp. 324–349.

[5] Bookstein, Fred L., "Principal Warps: Thin Plate Splines and the Decomposition of Deformations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 11, No. 6, June 1989, pp. 567–585.

[6] Celniker, George and Dave Gossard, "Deformable Curve and Surface Finite-Elements for Free-Form Shape Design," *Computer Graphics*, Vol. 25, No. 4 (SIGGRAPH 91), July 1991, pp. 257–266.

[7] Cohen-Or, Daniel, David Levin and Amira Solomovici, "Three Dimensional Distance Field Metamorphosis," *ACM Transactions on Graphics*, 1997.
[8] Duchon, Jean, "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," in *Constructive Theory of Functions of Several Variables*, Lecture Notes in Mathematics, edited by A. Dolb and B. Eckmann, Springer-Verlag, 1977, pp. 85–100.
[9] Duncan, Jody, "A Once and Future War," *Cinefex*, No. 47 (entire issue devoted to the film Terminator 2), August 1991, pp. 4–59.
[10] Fuchs, H., Z. M. Kedem and S. P. Uselton, "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, Vol. 20, No. 10, October 1977, pp. 693–702.
[11] Golub, Gene H. and Charles F. Ban Loan, *Matrix Computations*, John Hopkins University Press, 1996.
[12] Gregory, Arthur, Andrei State, Ming C. Lin, Dinesh Manocha, Mark A. Livingston, "Feature-based Surface Decomposition for Correspondence and Morphing between Polyhedra", *Proceedings of Computer Animation*, Philadelphia, PA., 1998.
[13] He, Taosong, Sidney Wang and Arie Kaufman, "Wavelet- Based Volume Morphing," *Proceedings of Visualization '94*, Washington, D. C., edited by Daniel Bergeron and Arie Kaufman, October 17-21, 1994, pp. 85–92.
[14] Herman, Gabor T., Jingsheng Zheng and Carolyn A. Bucholtz, "Shape-Based Interpolation," *IEEE Computer Graphics and Applications*, Vol. 12, No. 3 (May 1992), pp. 69–79.
[15] Hugues, John F., "Scheduled Fourier Volume Morphing," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 43–46.
[16] Kaul, Anil and Jarek Rossignac, "Solid- Interpolating Deformations: Construction and animation of PIPs," *Proceedings of Eurographics '91*, Vienna, Austria, 2-6 Sept. 1991, pp. 493–505.
[17] Kent, James R., Wayne E. Carlson and Richard E. Parent, "Shape Transformation for Polyhedral Objects," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 47–54.
[18] Lerios, Apostolos, Chase Garfinkle and Marc Levoy, "Feature-Based Volume Metamorphosis," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 95), pp. 449–456.
[19] Levin, David, "Multidimensional Reconstruction by Set-valued Approximation," *IMA J. Numerical Analysis*, Vol. 6, 1986, pp. 173–184.
[20] Litwinowicz, Peter and Lance Williams, "Animating Images with Drawings," *Computer Graphics Proceedings*, Annual Conference Series (SIGGRAPH 94), pp. 409–412.
[21] Lorenson, William and Harvey E. Cline, "Marching Cubes: A High Resolution 3-D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4 (SIGGRAPH 87), July 1987, pp. 163–169.
[22] Meyers, David and Shelley Skinner, "Surfaces From Contours: The Correspondence and Branching Problems," *Proceedings of Graphics Interface '91*, Calgary, Alberta, 3-7 June 1991, pp. 246–254.
[23] Payne, Bradley A. and Arthur W. Toga, "Distance Field Manipulation of Surface Models," *IEEE Computer Graphics and Applications*, Vol. 12, No. 1, January 1992, pp. 65–71.
[24] Rossignac, Jarek and Anil Kaul, "AGRELs and BIPs: Metamorphosis as a Bezier Curve in the Space of Polyhedra," *Proceedings of Eurographics '94*, Oslo, Norway, Sept. 12–16, 1994, pp. 179–184.
[25] Sederberg, Thomas W. and Eugene Greenwood, "A Physically Based Approach to 2-D Shape Blending," *Computer Graphics*, Vol. 26, No. 2 (SIGGRAPH 92), July 1992, pp. 25–34.
[26] Sederberg, Thomas W. and Scott R. Parry, "Free-Form Deformations of Solid Geometric Models," *Computer Graphics*, Vol. 20, No. 4 (SIGGRAPH 86), pp. 151–160.
[27] Turk, Greg and James F. O'Brien, "Variational Implicit Surfaces," Tech Report GIT-GVU-99-15, Georgia Institute of Technology, May 1999, 9 pages.
[28] Wolberg, George, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, California 1990.

10 Appendix: Warping

Warping is a commonly used method of providing user control of shape interpolation. Although warping is not a focus of our research, for the sake of completeness we describe below how warping may be used together with our shape transformation technique. Research on warping (sometimes called deformation) include [2, 26, 28, 3, 18, 7].

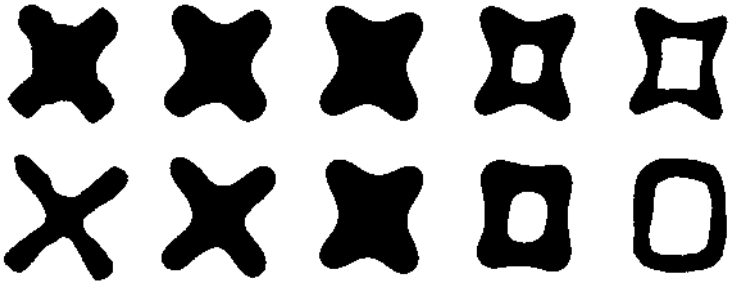


Figure 9: The extreme left and right shapes in the top row have been warped before creating the upper shape transformation sequence. The lower row is an un-warped version of this sequence that gives the final transformation from an X to O.

For symmetry, we choose to warp each shape "half-way" to the other shape. Given a set of user-supplied corresponding points between two shapes A and B, we construct two displacement warp functions $\mathbf{w}_A$ and $\mathbf{w}_B$. The function $\mathbf{w}_A$ specifies what values to add to locations on shape A in order to warp it half-way to shape B, and the warping function $\mathbf{w}_B$ warps B half of the way to A.

In what follows, we will describe the warping process for two-dimensional shapes. Let $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_k\}$ be a set of points on shape A, and let $\{\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k\}$ be the corresponding points on B. We construct the two functions w_A and w_B such that $w_A(\mathbf{a}_i) = (\mathbf{b}_i - \mathbf{a}_i)/2$ and $w_B(\mathbf{b}_i) = (\mathbf{a}_i - \mathbf{b}_i)/2$ hold for all i. Constructing these functions is another example of scattered data interpolation which we can solve using variational techniques. For 2D shapes, if $\mathbf{a}_i = (a_i^x, a_i^y)$ and $\mathbf{b}_i = (b_i^x, b_i^y)$, then the x component w_A^x of the displacement warp $\mathbf{w}_A$ has k constraints at the positions $\mathbf{a}_i$ with values $(b_i^x - a_i^x)/2$. We invoke variational interpolation to satisfy these constraints, and do the same to construct the y component of the warp. The function $\mathbf{w}_B$ is constructed similarly. This is not a new technique, and researchers who use thin-plate techniques to perform shape warping include [5, 20] and others.

In order to combine warping with shape transformation, we use these functions to displace all of the boundary constraints of the given shapes. These displaced boundary constraints are embedded in 3D (as described in Section 5) and then variational interpolation is used to create the implicit function that describes the entire shape transformation sequence. The result of this process is a three-dimensional implicit function, each slice of which is an intermediate shape between two warped shapes. The top row of Figure 9 shows such warped intermediate shapes. We can think of the two "ends" of this implicit function (at $t = 0$ and $t = t_{max}$) as being warped versions of our original shapes. In order to match the two original shapes, the surface of this 3D implicit function needs to be unwarped. To simplify the equations, assume that the value of t_{max} is 2. If $t \leq 1$ the unwarping function $u(x,y,t)$ is:

$$u(x,y,t) = (x + (1-t)w_A^x(x,y), y + (1-t)w_A^y(x,y), t) \qquad (4)$$

If $t > 1$ then the unwarping function is:

$$u(x,y,t) = (x + (t-1)w_B^x(x,y), y + (t-1)w_B^y(x,y), t) \qquad (5)$$

At the extreme of $t = 0$, the warp $u(x,y,t)$ un-does the warping we used for the first shape. At $t = 2$, the function $u(x,y,t)$ reverses the warping used for the second shape. When $t = 1$ (the middle shape in the sequence), no warp is performed. The bottom sequence of shapes in Figure 9 shows the result of the entire shape transformation process that includes warping. Both sequences in Figure 5 were created using warping in addition to shape transformation.

Although we have described the warping process for 2D shapes, the same method may be used for shape transformation between 3D shapes. For Figure 6 (right), warping was used to align the bunny ears to the points of the star.

Modelling with Implicit Surfaces that Interpolate

GREG TURK
Georgia Institute of Technology
JAMES F. O'BRIEN
University of California, Berkeley

We introduce new techniques for modelling with *interpolating implicit surfaces*. This form of implicit surface was first used for problems of surface reconstruction and shape transformation, but the emphasis of our work is on model creation. These implicit surfaces are described by specifying locations in 3D through which the surface should pass, and also identifying locations that are interior or exterior to the surface. A 3D implicit function is created from these constraints using a variational scattered data interpolation approach, and the iso-surface of this function describes a surface. Like other implicit surface descriptions, these surfaces can be used for CSG and interference detection, may be interactively manipulated, are readily approximated by polygonal tilings, and are easy to ray trace. A key strength for model creation is that interpolating implicit surfaces allow the direct specification of both the location of points on the surface and the surface normals. These are two important manipulation techniques that are difficult to achieve using other implicit surface representations such as sums of spherical or ellipsoidal Gaussian functions ("blobbies"). We show that these properties make this form of implicit surface particularly attractive for interactive sculpting using the particle sampling technique introduced by Witkin and Heckbert. Our formulation also yields a simple method for converting a polygonal model to a smooth implicit model, as well as a new way to form blends between objects.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid, and object representations*

General Terms: Algorithms

Additional Key Words and Phrases: Implicit surfaces, thin-plate techniques, function interpolation, modeling

1. INTRODUCTION

The computer graphics, computer-aided design and computer vision literatures are filled with an amazingly diverse array of approaches to surface description. The reason for this variety is that there is no single representation of surfaces that satisfies the needs of every problem in every application area. This paper is about modelling with *interpolating implicit surfaces*, a surface representation that we believe will be useful in several areas in 3D modeling. These implicit surfaces are smooth, exactly pass through a set of given constraint points, and can describe closed surfaces of arbitrary topology.

In order to illustrate our basic approach, Figure 1 (left) shows an interpolating implicit curve, the 2D analog of an interpolating implicit surface. The small open circles in this figure indicate the location of constraints where the 2D implicit function must take on the value of zero. The single plus sign corresponds to an additional constraint where the implicit function must take on the value of some

This work was funded under ONR grant N00014-97-0223.
Authors' addresses: G. Turk, College of Computing, Room 257, Georgia Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280; email: turk@cc.gatech.edu; J. F. O'Brien, Soda Hall, Mail Code 1776, EECS Computer Science Division, University of California, Berkeley, Berkeley, CA 94720-1776; email: job@eecs.berkeley.edu.

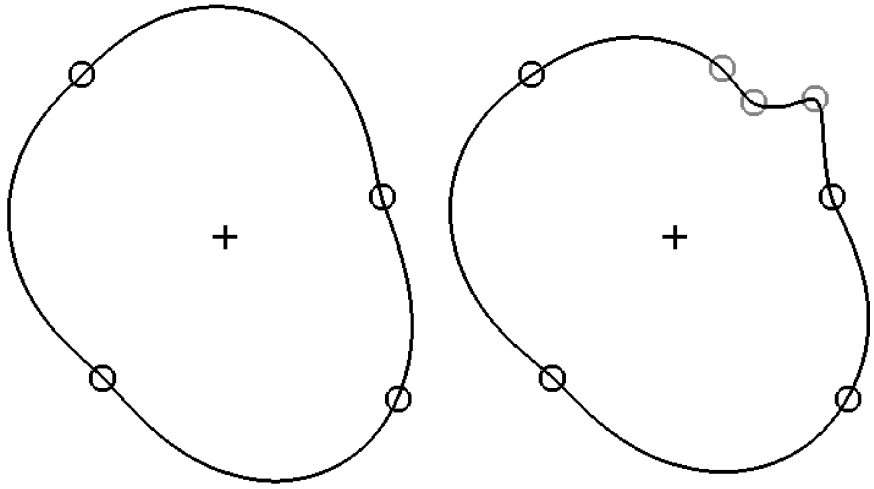


Fig. 1. Curves defined using interpolating implicit functions. The curve on the left is defined by four zero-valued and one positive, constraint. This curve is refined by adding three new zero-valued constraints (shown in red at right).

arbitrary positive constant, which for this example is one. These constraints are passed along to a scattered data interpolation routine that generates a smooth 2D function meeting the given constraints. The desired curve is defined to be the locus of points at which the function takes on the value of zero. The curve exactly passes through each of the zero-value constraints, and its defining function is positive inside this curve and negative outside. For this 2D example, we use a variational technique that minimizes the aggregate curvature of the function that it creates, and this technique for creating a function is often referred to as thin-plate interpolation.

We can create surfaces in 3D in exactly the same way as the 2D curves in Figure 1. Zero-valued constraints are defined by the modeler at 3D locations, and positive values are specified at one or more places that are to be interior to the surface. A variational interpolation technique is then invoked that creates a scalar-valued function over a 3D domain. The desired surface is simply the set of all points at which this scalar function takes on the value zero. Figure 2 (left) shows a surface that was created in this fashion by placing four zero-valued constraints at the vertices of a regular tetrahedron and placing a single positive constraint in the center of the tetrahedron. The result is a nearly spherical surface. More complex surfaces such as the branching shape in Figure 2 (right) can be defined simply by specifying more constraints. Figure 3 shows an example of an interpolating implicit surface created from polygonal data.

The remainder of this paper is organized as follows. In Section 2 we examine related work, including implicit surfaces and thin-plate interpolation techniques. We describe in Section 3 the mathematical framework for solving variational problems using radial basis functions. Section 4 presents three strategies that may be used together with variational methods to create implicit surfaces. These strategies differ in where they place the non-zero constraints. In Section 5 we show that interpolating implicit surfaces are well suited for interactive sculpting. In Section 6 we present a new method of creating soft blends between objects, based on interpolating implicits. Section 7 describes two rendering techniques, one that relies on polygonal tiling and another based on ray tracing. In Section 8 we compare interpolating implicit surfaces with traditional thin-plate surface modeling and with implicit functions that are created using ellipsoidal Gaussian functions. Finally, Section 9 indicates potential applications and directions for future research.

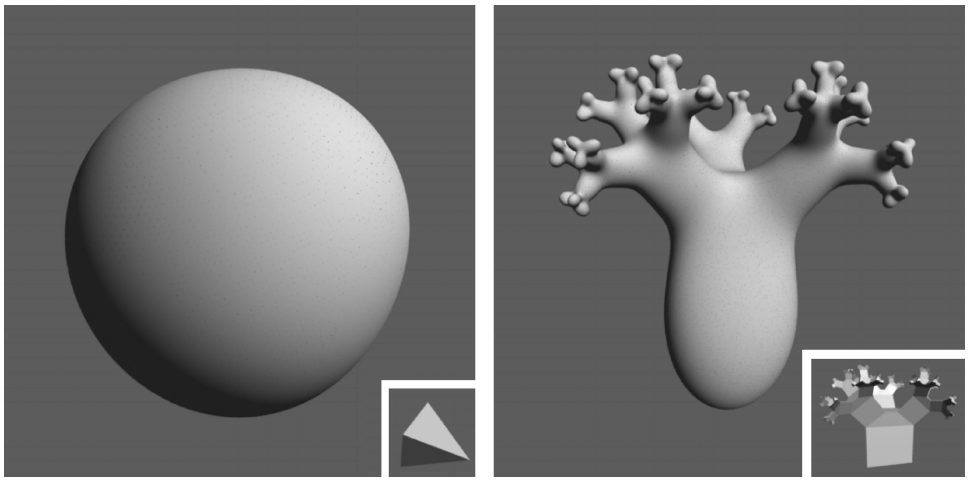


Fig. 2. Surfaces defined by interpolating implicit functions. The left surface is defined by zero-valued constraints at the corners of a tetrahedron and one positive constraint in its center. The branching surface at the right was created using constraints from the vertices of the inset polygonal object.

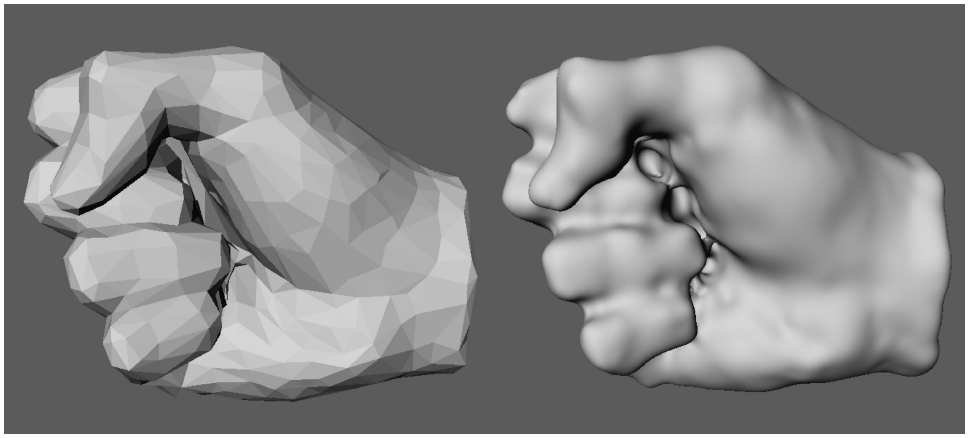


Fig. 3. Polygonal surface of a human fist with 750 vertices (left) and an interpolating implicit surface created from the polygons (right).

2. BACKGROUND AND RELATED WORK

Interpolating implicit surfaces draw upon two areas of modeling: implicit surfaces and thin-plate interpolation. In this section we briefly review work in these two sub-areas. Interpolating implicit surfaces are not new to graphics, and at the close of this section we describe earlier published methods of creating interpolating implicit surfaces.

2.1 Implicit Surfaces

An implicit surface is defined by an implicit function, a continuous scalar-valued function over the domain $\mathbf{R}^3$. The implicit surface of such a function is the locus of points at which the function takes on the value zero. For example, a unit sphere may be defined using the implicit function $f(\mathbf{x}) = 1 - |\mathbf{x}|$, for

points $\mathbf{x} \in \mathbf{R}^3$. Points on the sphere are those locations at which $f(\mathbf{x}) = 0$. This implicit function takes on positive values inside the sphere and is negative outside the surface, as will be the convention in this paper.

An important class of implicit surfaces are the *blobby* or *meta-ball surfaces* [Blinn 1982; Nishimura et al. 1985]. The implicit functions of these surfaces are the sum of radially symmetric functions that have a Gaussian profile. Here is the general form of such an implicit function:

$$f(\mathbf{x}) = -t + \sum_{i=1}^{n} g_i(\mathbf{x}) \tag{1}$$

In the above equation, a single function g_i describes the profile of a "blobby sphere" (a Gaussian function) that has a particular center and standard deviation. The bold letter $\mathbf{x}$ represents a point in the domain of our implicit function, and in this paper we will use bold letters to represent such points, both in 2D and 3D. The value t is the iso-surface threshold, and it specifies one particular surface from a family of nested surfaces that are defined by the sum of Gaussians. When the centers of two blobby spheres are close enough to one another, the implicit surface appears as though the two spheres have melted together. A typical form for a blobby sphere function g_i is the following:

$$g_i(\mathbf{x}) = e^{|\mathbf{x}-\mathbf{c}_i|^2/\sigma_i^2} \tag{2}$$

In this equation, the constant σ_i specifies the standard deviation of the Gaussian function, and thus is the control over the radius of a blobby sphere. The center of a blobby sphere is given by $\mathbf{c}_i$. Evaluating an exponential function is computationally expensive, so some authors have used piecewise polynomial expressions instead of exponentials to define these blobby sphere functions [Nishimura et al. 1985; Wyvill et al. 1986]. A greater variety of shapes can be created with the blobby approach by using ellipsoidal rather than spherical functions.

Another important class of implicit surfaces are the algebraic surfaces. These are surfaces that are described by polynomial expressions in x, y and z. If a surface is simple enough, it may be described by a single polynomial expression. A good deal of attention has been devoted to this approach, and we recommend Taubin [1993] and Keren and Gotsman [1998] as starting points in this area. Much of the work on this method has been devoted to fitting an algebraic surface to a given collection of points. Usually it is not possible to interpolate all of the data points, so error minimizing techniques are sought. Surfaces may also be described by piecing together many separate algebraic surface patches, and here again there is a large literature on the subject. Good introductions to these surfaces may be found in the chapters by Bajaj and Rockwood in Bloomenthal [1997]. It is easier to create complex surfaces using a collection of algebraic patches rather than using a single algebraic surface. The tradeoff, however, is that a good deal of machinery is required to create smooth joins across patch boundaries.

We have only described some of the implicit surface representations that are most closely related to our own work. There are many other topics within the broad area of implicit surfaces, and we refer the interested reader to the excellent book by Bloomenthal and his co-authors, Bloomenthal [1997].

2.2 Thin-Plate Interpolation

Thin-plate spline surfaces are a class of height fields that are closely related to the interpolating implicit surfaces of this paper. Thin-plate interpolation is one approach to solving the *scattered data interpolation* problem. The two-dimensional version of this problem can be stated as follows: Given a collection of k constraint points $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$ that are scattered in the xy-plane, together with scalar height values at each of these points $\{h_1, h_2, \ldots, h_k\}$, construct a "smooth" surface that matches each of these heights at the given locations. We can think of this solution surface as a scalar-valued function $f(\mathbf{x})$ so that

$f(\mathbf{c}_i) = h_i$, for $1 \leq i \leq k$. If we define the word *smooth* in a particular way, there is a unique solution to such a problem, and this solution is the thin-plate interpolation of the points. Consider the energy function $E(f)$ that measures the smoothness of a function f:

$$E(f) = \int_{\Omega} f_{xx}^2(\mathbf{x}) + 2f_{xy}^2(\mathbf{x}) + f_{yy}^2(\mathbf{x})\, d\mathbf{x} \tag{3}$$

The notation f_{xx} means the second partial derivative in the x direction, and the other two terms are similar partial derivatives, one of them mixed. This energy function is basically a measure of the aggregate curvature of $f(\mathbf{x})$ over the region of interest Ω (a portion of the plane). Any creases or pinches in a surface will result in a larger value of E. A smooth function that has no such regions of high curvature will have a lower value of E. Note that because there are only squared terms in the integral, the value for E can never be negative. The thin-plate solution to an interpolation problem is the function $f(\mathbf{x})$ that satisfies all of the constraints and that has the smallest possible value of E. Note that thin-plate surfaces are height fields, and thus they are in fact *parametric* surfaces.

This interpolation method gets its name because it is much like taking a thin sheet of metal, laying it horizontally and bending it so that it just touches the tips of a collection of vertical poles that are set at the positions and heights given by the constraints of the interpolation problem. The metal plate resists bending so that it smoothly changes its height in the positions between the poles. This springy resistance is mimicked by the energy function E. Thin-plate interpolation is often used in the computer vision domain, where there are often sparse surface constraints [Grimson 1983; Terzopoulos 1988]. The above curvature minimization process is sometimes referred to as regularization, and can be thought of as an additional constraint that selects a unique surface out of an infinite number of surfaces that match a set of given height constraints. Solving such constrained problems draws from a branch of mathematics called the variational calculus, thus thin-plate techniques are sometimes referred to as variational methods.

The scattered data interpolation problem can be formulated in any number of dimensions. When the given points $\mathbf{c}_i$ are positions in n-dimensions rather than in 2D, this is called the n-dimensional scattered data interpolation problem. There are appropriate generalizations to the energy function and to thin-plate interpolation for any dimension. In this paper we will make use of variational interpolation in two and three dimensions.

2.3 Related Work on Implicit Surfaces

The first publication on interpolating implicits of which we are aware is that of Savchenko et al. [1995]. We consider this to be a pioneering paper in implicit surfaces, and feel it deserves to be known more widely than it is at present. Their research was on the creation of implicit surfaces from measured data such as range data or contours. Their work did not, however, describe techniques for modelling. Their approach to implicit function creation is similar to our method in the present paper in that both solve a linear system to get the weights for radial basis functions. The work of Savchenko et al. differs from our own in that they use a *carrier solid* to suggest what part of space should be interior to the surface that is being created. We believe that the three methods that we describe for defining the interior of a surface in Section 4 of this paper give more user control than a carrier solid and are thus more appropriate for modelling.

The implicit surface creation methods described in this paper are an outgrowth of earlier work in shape transformation by Turk and Turk and O'Brien [1999]. They created implicit functions in $n + 1$ dimensions to interpolate between pairs of n-dimensional shapes. These implicit functions were created using the *normal constraint* formulation of interpolating implicit surfaces, as described in Section 4.3 of

this paper. The present paper differs from that of Turk and O'Brien [1999] in its introduction of several techniques for defining interpolating implicit surfaces that are especially useful for model creation.

Recently, techniques have developed that allow the methods discussed above to be applied to systems with large numbers of constraints [Morse et al. 2001; Carr et al. 2001]. The work of Morse et al. uses Gaussian-like compactly supported radial basis functions to accelerate the surface building process; they are able to create surfaces that have tens of thousands of constraints. Carr et al. use fast evaluation methods to reconstruct surfaces using up to a half millions basis functions. They use the radial basis function $\phi(\mathbf{x}) = |\mathbf{x}|$, the biharmonic basis function. Both of these improvements for creating surfaces with many constraints are complementary to the work of the present paper, and the new techniques that we describe in Sections 4, 5 and 6 should work gracefully with the methods in both of these papers.

3. VARIATIONAL METHODS AND RADIAL BASIS FUNCTIONS

In this section we review the necessary mathematical background for thin-plate interpolation. This will provide the tools that we will then use in Section 4 to create interpolating implicit surfaces.

The scattered data interpolation task as formulated above is a variational problem where the desired solution is a function, $f(\mathbf{x})$, that will minimize Equation 3 subject to the interpolation constraints $f(\mathbf{c}_i) = h_i$. There are several numerical methods that can be used to solve this type of problem. Two commonly used methods, finite elements and finite differencing techniques, discretize the region of interest, Ω, into a set of cells or elements and define local basis functions over the elements. The function $f(\mathbf{x})$ can then be expressed as a linear combination of the basis functions so that a solution can be found, or approximated, by determining suitable weights for each of the basis functions. This approach has been widely used for height-field interpolation and deformable models, and examples of its use can be found in [Terzopoulos 1988; Szeliski 1990; Celniker and Gossard 1991; Welch and Witkin 1994]. While finite elements and finite differencing techniques have proven useful for many problems, the fact that they rely on discretization of the function's domain is not always ideal. Problems that can arise due to discretization include visibly stair-stepped surfaces and the inability to represent fine details. In addition, the cost of using such methods grows cubically with the desired resolution.

An alternate approach is to express the solution in terms of radial basis functions centered at the constraint locations. Radial basis functions are radially symmetric about a single point, or center, and they have been widely used for function approximation. Remarkably, it is possible to choose these radial functions in such a way that they will automatically solve differential equations, such as the one required to solve Equation 3, subject to constraints located at their centers. For the 2D interplation problem, Equation 3 can be solved using the biharmonic radial basis function:

$$\phi(\mathbf{x}) = |\mathbf{x}|^2 \log(|\mathbf{x}|) \tag{4}$$

This is commonly know as the thin-plate radial basis function. For 3D interpolation, one commonly used radial basis function is $\phi(\mathbf{x}) = |\mathbf{x}|^3$, and this is the basis function that we use. We note that Carr et al. [2001] used the basis function $\phi(\mathbf{x}) = |\mathbf{x}|$. Duchon [1977] did much of the early work on variational interpolation, and the report by Girosi, Jones and Poggio is a good entry point into the mathematics of variational interpolation [Girosi et al. 1993].

Using the appropriate radial basis functions, we can write the interpolation function in this form:

$$f(\mathbf{x}) = \sum_{j=1}^{k} w_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}) \tag{5}$$

In the above equation, $\mathbf{c}_j$ are the locations of the constraints, w_j are the weights, and $P(\mathbf{x})$ is a degree one polynomial that accounts for the linear and constant portions of f. Solving for the weights w_j and the coefficients of $P(\mathbf{x})$ subject to the given constraints yields a function that both interpolates the constraints and minimizes Equation 3. The resulting function exactly interpolates the constraints (if we ignore numerical precision issues), and is not subject to approximation or discretization errors. Also, the number of weights to be determined does not grow with the size of the region of interest Ω. Rather, it is only dependent on the number of constraints.

To solve for the set of w_j that will satisfy the interpolation constraints, we begin with the criteria that the surface must interpolate our constraints:

$$h_i = f(\mathbf{c}_i) \tag{6}$$

We now substitute the right side of Equation 5 for $f(\mathbf{c}_i)$ to give us:

$$h_i = \sum_{j=1}^{k} w_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}) \tag{7}$$

Since the above equation is linear with respect to the unknowns, w_j, and the coefficients of $P(\mathbf{x})$, it can be formulated as a linear system. For interpolation in 3D, let $\mathbf{c}_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then this linear system can be written as follows:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \dots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \dots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \dots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

The sub-matrix in Equation 8 consisting of the ϕ_{ij}'s is conditionally positive-definite on the subspace of vectors that are orthogonal to the last four rows of the full matrix, so Equation 8 is guaranteed to have a solution. We used symmetric LU decomposition to solve this system of equations for all of the examples shown in this paper. Our implementation to set up the system, call the LU decomposition routine and evaluate the interpolating function of Equation 5 for a given $\mathbf{x}$ consists of about 100 lines of commented C++ code. This code plus the public-domain polygonalization routine described in Section 7.1 is all that is needed to create interpolating implicit surfaces.

Two concerns that arise with such matrix systems are computation times and ill-conditioned systems. For systems with up to a few thousand centers, including all of the examples in this paper, direct solution techniques such as LU decomposition and SVD are practical. However as the system becomes larger, the amount of work required to solve the system grows as $O(k^3)$. We have used direct solution methods for systems with up to roughly 3,000 constraints. LU decomposition becomes impractical for more constraints than this. We are pleased that other researchers, notably the authors of [Morse et al. 2001; Carr et al. 2001], have begun to address this issue of computational complexity.

As the number of constraints grows, the condition number of the matrix in equation 8 is also likely to grow, leading to instability for some solution methods. For the systems we have worked with,

ill-conditioning has not been a problem. If problems arise for larger systems, variational interpolation is such a well-studied problem that methods exist for improving the conditioning of the system of equations (see Dyn [1987]).

4. CREATING INTERPOLATING IMPLICIT SURFACES

With tools for solving the scattered data interpolation problem in hand, we now turn our attention to creating implicit functions. In this section we will examine three ways in which to define a interpolating implicit surface. Common to all three approaches, is the specification of zero-valued constraints through which the surface must pass. The three methods differ in specifying where the implicit function takes on positive and negative values. These methods are based on using three different kinds of constraints: *interior, exterior*, and *normal*. We will look at creating both 2D interpolating implicit curves and 3D interpolating implicit surfaces. The 2D curve examples are for illustrative purposes, and our actual goal is the creation of 3D surfaces.

4.1 Interior Constraints

The left portion of Figure 1 (earlier in this paper) shows the first method of describing a interpolating implicit curve. Four zero-valued constraints have been placed in the plane. We call such zero-value constraints *boundary constraints* because these points will be on the boundary between the interior and exterior of the shape that is being defined. In addition to the four boundary constraints, a single constraint with a value of one is placed at the location marked with a plus sign. We use the term *interior constraint* when referring to such a positive-valued constraint that helps to determine the interior of the surface. We construct an implicit function from these five constraints simply by invoking the 2D variational interpolation technique described in earlier sections. The interpolation method returns a set of scalar coefficients w_i that weight a collection of radially symmetric functions ϕ that are centered at the constraint positions. The implicit curve shown in the figure is given by those locations at which the variationally-defined function takes on the value zero. The function takes on positive values inside the curve and is negative at locations outside the curve. Figure 1 (right) shows a refinement of the curve that is made by adding three more boundary constraints to the original set of constraints in the left portion of the figure.

Why does an interior constraint surrounded by zero-valued constraints yield a function that is negative beyond the boundary constraints? The key is that the energy function is larger for functions that take on positive values on both sides of a zero-valued constraint. Each boundary constraint acts much like a see-saw. If we pull the surface up on one side of a boundary constraint (using an interior constraint), then the other side tends to move down.

Creating surfaces in 3D is accomplished in exactly the same way as the 2D case. Zero-valued constraints are specified by the modeler as those 3D points through which the surfaces should pass, and positive values are specified at one or more places that are to be interior to the surface. Variational interpolation is then invoked to create a scalar-valued function over $\mathbf{R}^3$. The desired surface is simply the set of all points at which this scalar function takes on the value zero. Figure 2 (left) shows a surface that was created in this fashion by placing four zero-valued constraints at the vertices of a regular tetrahedron and placing a single interior constraint in the center of the tetrahedron. The resulting implicit surface is nearly spherical.

Figure 2 (right) shows a recursive branching object that is an interpolating implicit surface. The basic building block of this object is a triangular prism. Each of the six vertices of a large prism specified the location of a zero-valued constraint, and a single interior constraint was placed in the center of this prism. Next, three smaller and slightly tilted prisms were placed atop the first large prism. Each of these smaller prisms, like the large one, contributes boundary constraints at its vertices and has a

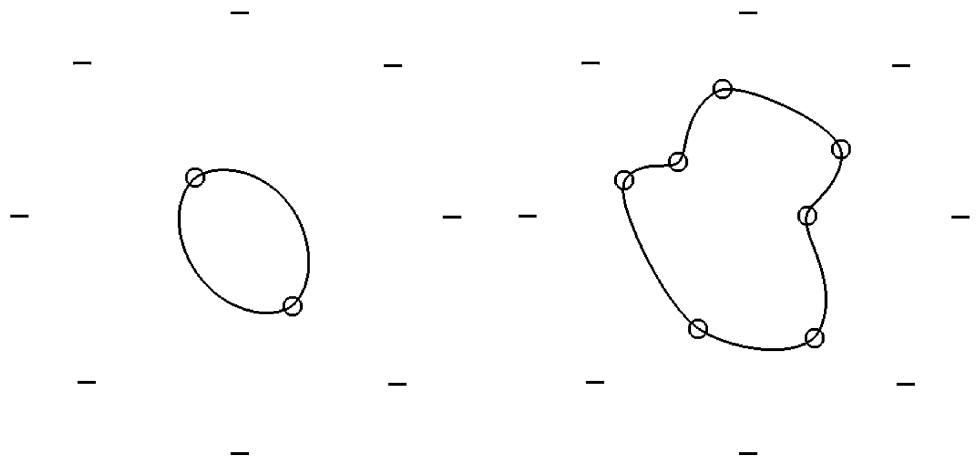


Fig. 4. Curves defined using surrounding exterior constraints. Just two zero-valued constraints yield an ellipse-like curve (on the left). More constraints create a more complex curve (at right).

single interior constraint placed at its center. Each of the three smaller prisms had even smaller prisms placed on top of them, and so on.

Why does this method of creating an implicit function create a smooth surface? We are creating the scalar-valued function in 3D that matches our constraints, and that minimizes a 3D energy functional similar to Equation 3. This energy functional selects a smoothly changing implicit function that matches the constraints. The iso-surface that we extract from such a smoothly changing function will almost always be smooth as well. It is *not* the case in general, however, that this iso-surface is also the minimum of a curvature-based functional over surfaces. Satisfying the 3D energy functional does not give any guarantee about the smoothness of the resulting 2D surface.

Placing one or more positive-valued constraints on the interior of a shape is an effective method of defining interpolating implicit surfaces when the shape one wishes to create is well-defined. We have found, however, that there is another approach that is even more flexible for interactive free-form surface sculpting.

4.2 Exterior Constraints

Figure 4 illustrates a second approach to creating interpolating implicit functions. Instead of placing positive-valued constraints inside a shape, negative-valued constraints can be placed on the exterior of the shape that is being created. We call each such negative-valued constraint an *exterior constraint*. As before, zero-valued constraints specify locations through which the implicit curve will pass. In Figure 4 (left), eight exterior constraints surround the region at which a curve is being created. As with positive-valued constraints, the magnitude of the values is unimportant, and we use the value-negative one. These exterior constraints, coupled with the curvature-minimizing nature of variational method, induce the interpolation function to take on positive values interior to the shape outlined by the zero-valued constraints. Even specifying just two boundary constraints defines a reasonable closed curve, as shown by the ellipse-like curve at the left in Figure 4. More boundary constraints result in a more complex curve, as shown on the right in Figure 4.

We have found that creating a circle or sphere of negative-valued constraints is the approach that is best suited to interactive free-form design of curves and surfaces. Once these exterior constraints are defined, the user is free to place boundary constraints in any location interior to this cage of exterior constraints. Section 5 describes the use of exterior constraints for interactive sculpting.

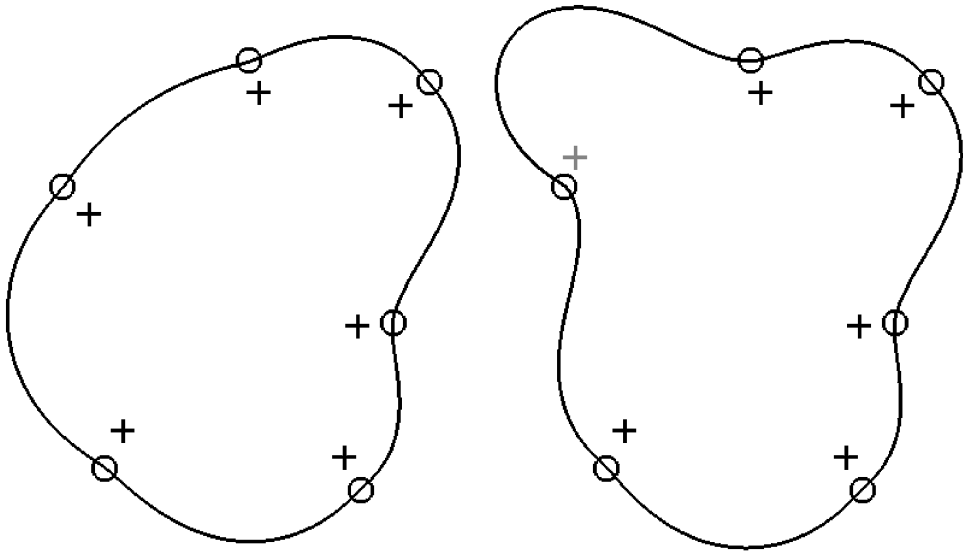


Fig. 5. Two curves defined using nearly identical boundary and normal constraints. By moving just a single normal constraint (the north-west one, shown in red), the curve on the left is changed to that shown on the right.

4.3 Normal Constraints

For some applications we may have detailed knowledge about the shape that is to be modeled. In particular, we may know approximate surface normals at many locations on the surface to be created. In this case there is a third method of defining an interpolating implicit function that may be preferred over the two methods described above, and this method was originally described in Turk and O'Brien [1999]. Rather than placing positive or negative values far from the boundary constraints, we can create constraints very close to the boundary constraints. Figure 5 shows this method in the plane. In the left portion of this figure, there are six boundary constraints and in addition there are six *normal constraints*. These normal constraints are positive-valued constraints that are placed very near the boundary constraints, and they are positioned towards the center of the shape that is being created. A normal constraint is created by placing a positive constraint a small distance in the direction $-\mathbf{n}$, where $\mathbf{n}$ is an approximate normal to the shape that we are creating. (Alternatively, we could choose to place negative-valued constraints in the outward-pointing direction.) A normal constraint is always paired with a boundary constraint, although not every boundary constraint requires a normal constraint. The right part of Figure 5 shows that a normal constraint can be used to bend a curve at a given point.

There are at least two ways in which a normal constraint might be defined. One is to allow a user to hand-specify the surface normals of a shape that is being created. A second allows us to create smooth surfaces based on polyhedral models. If we wish to create an interpolating implicit surface from a polyhedral model, we simply need to create one boundary constraint and one normal constraint for each vertex in the polyhedron. The location of a boundary constraint is given by the position of the vertex, and the location of a normal constraint is given by moving a short distance in a direction opposite to the surface normal at the vertex. We place normal constraints 0.01 units from the corresponding boundary constraints for objects that fit within a unit cube. Figure 6 (right) shows an interpolating implicit surface created in the manner just described from the polyhedral model in Figure 6 (left). This is a simple yet effective way to create an everywhere smooth analytically defined surface. This stands in contrast to

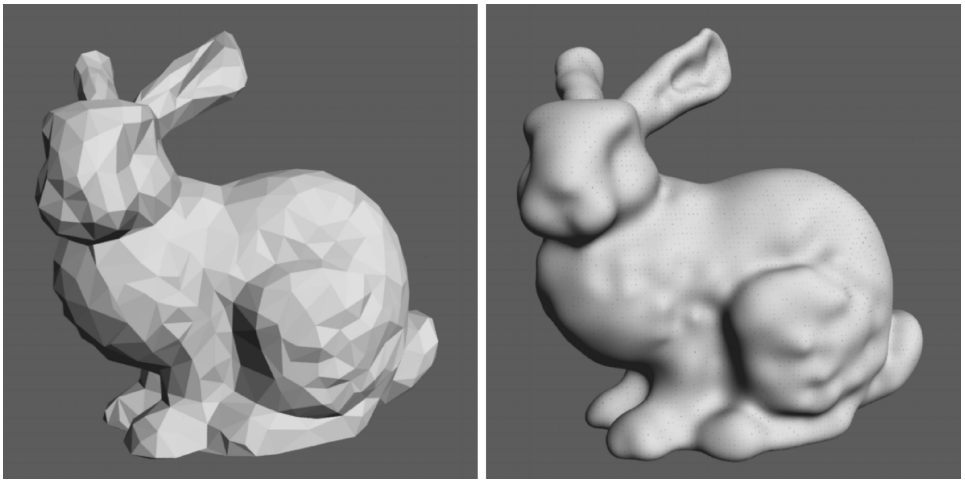


Fig. 6. A polygonal surface (left) and the interpolating implicit surface defined by the 800 vertices and their normals (right).

Table I. Constraint Types

Constraint Types	**When to Use**	**2D Figure**	**3D Figure**
Interior constraints	Planned model construction	Figure 1	Figure 2
Exterior constraints	Interactive modelling	Figure 4	Figures 7, 8, 10
Normal constraints	Conversion from polygons	Figure 5	Figures 3, 6, 9

the complications of patch stitching inherent in most parametric surface modeling approaches. Figure 3 is another example of converting polygons (a fist) to an implicit surface.

4.4 Review of Constraint Types

In this section we have seen three methods of creating interpolating implicit functions. These methods are in no way mutually exclusive, and a user of an interactive sculpting program could well use a mixture of these three techniques to define a single surface. Table I lists each of the three kinds of constraints, when we believe each is appropriate to use, and which figures in this paper were created using each of the methods.

5. INTERACTIVE MODEL BUILDING

Interpolating implicit surfaces seem ready-made for interactive 3D sculpting. In this section we will describe how they can be gracefully incorporated into an interactive modeling program.

In 1994, Andrew Witkin and Paul Heckbert presented an elegant method for interactive manipulation of implicit surfaces [Witkin and Heckbert 1994]. Their method uses two types of oriented particles that lie on the surface of an implicitly defined object. One class of particles, the floaters, are passive elements that are attracted to the surface of the shape that is being sculpted. Floaters repel one another in order to evenly cover the surface. Even during large changes to the surface, a nearly constant density of floaters is maintained by particle fissioning and particle death. A second type of particle, the control point, is the method by which a user interactively shapes an implicit surface. Control points provide the user with direct control of the surface that is being created. A control point tracks a 3D cursor position that is moved by the user, and the free parameters of the implicit function are adjusted so that the surface always passes exactly through the control point. The mathematical machinery needed

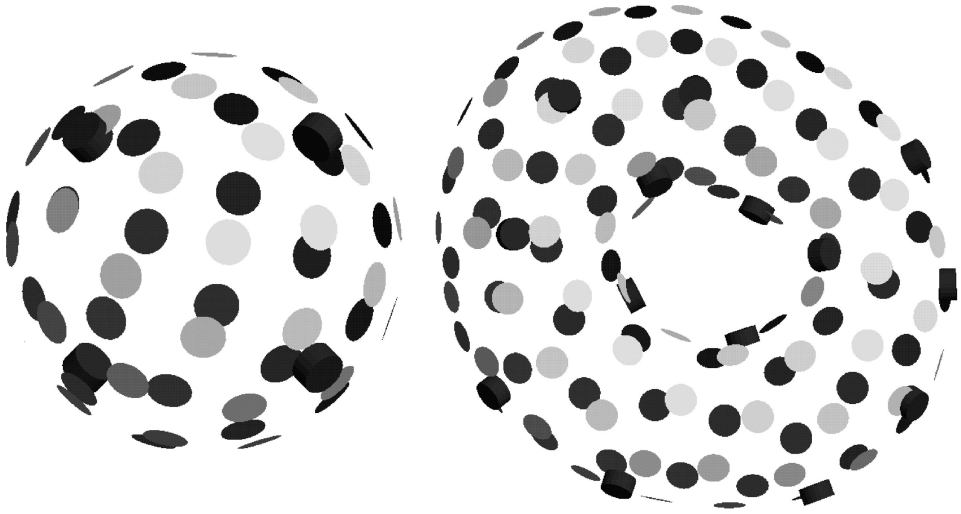


Fig. 7. Interactive sculpting of interpolating implicit surfaces. The left image shows an initial configuration with four boundary constraints (the red markers). The right surface is a sculpted torus.

to implement floaters and control points is presented clearly in Witkin and Heckbert's paper, and the interested reader should consult it for details.

The implicit surfaces used in Witkin and Heckbert's modeling program are blobby spheres and blobby cylinders. We have created an interactive sculpting program based on their particle sampling techniques, but we use interpolating implicit surfaces instead of blobbies as the underlying shape description. Our implementation of floaters is an almost verbatim transcription of their equations into code. The only change needed was to represent the implicit function as a sum of $\phi(\mathbf{x}) = |\mathbf{x}|^3$ radial basis functions and to provide an evaluation routine for this function and its gradient. Floater repulsion, fissioning, and death, work for interpolating implicits just as well as when using blobby implicit functions. As in the original system, the floaters provide a means of interactively viewing an object during editing that may even change the topology of the surface.

The main difference between our sculpting system and Witkin and Heckbert's is that we use an entirely different mechanism for direct interaction with a surface. Witkin/Heckbert control points provide an *indirect* link between a 3D cursor and the free parameters of a blobby implicit function. We do *not* make use of Witkin and Heckbert's control particles in our interactive modelling program. Instead, we simply allow users to create and move the boundary constraints of an interpolating implicit surface. This provides a direct way to manipulate the surface.

We initialize a sculpting session with a simple interpolating implicit surface that is nearly spherical; this is shown at the left in Figure 7. It is described by four boundary constraints at the vertices of a unit tetrahedron (the thick red disks) and with eight exterior (negative) constraints surrounding these at the corners of a cube with a side width of six. (The exterior constraints are not drawn.) A user is free to drag any of the boundary constraint locations using a 3D cursor, and the surface follows. The user may also create any number of new boundary constraints on the surface. The location of a new boundary constraint is found by intersecting the surface with a ray that passes through the camera position and the cursor. After a user creates or moves a boundary constraint, the matrix equation from Section 3 is solved anew. The floaters are then moved and displayed. The right portion of Figure 7 shows a toroidal surface that was created using this interactive sculpting paradigm. The interactive program repeatedly

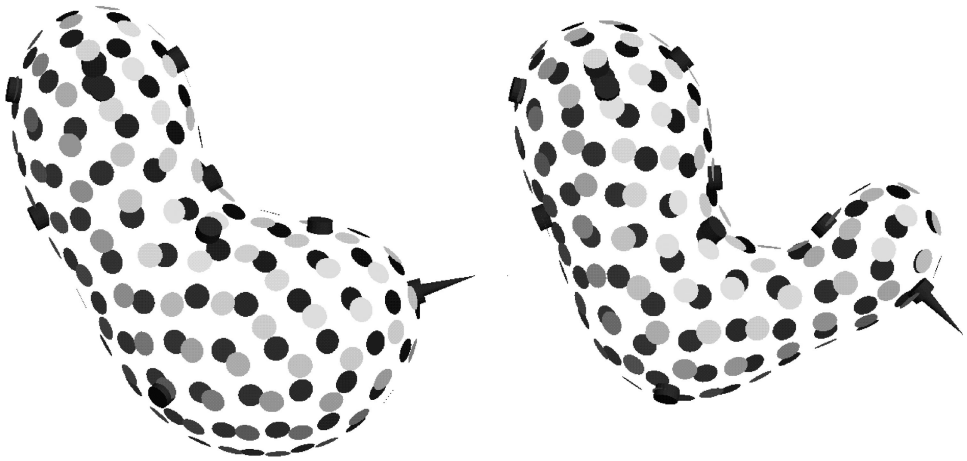


Fig. 8. Changing a normal constraint. Left image shows the original surface, and right image shows the same surface after changing a normal constraint (shown as a red spike).

executes the following steps:

1. Create or move constraints based on user interaction.
2. Solve new variational matrix equation.
3. Adjust floater positions (with floater birth and death).
4. Render floaters.

An important consequence of the matrix formulation given by Equation 8 is that adding a new boundary constraint on the existing surface does not affect the surface shape at all. This is because the implicit function already takes on the value of zero at the surface, so adding a new zero-valued constraint on the surface will not alter the surface. Only when such a new boundary constraint is moved, does it begin to affect the shape of the surface. This ability to retain the exact shape of a surface while adding new boundary constraints is similar in spirit to knot insertion for polynomial spline curves and surfaces. We do not know of any similar capability for blobby implicit surfaces.

In addition to control of boundary constraints, we also allow a user to create and move normal constraints. By default, no normal constraint is provided for a newly created boundary constraint. At the user's request, a normal constraint can be created at any specified boundary constraint. The initial direction of the normal constraint is given by the gradient of the current implicit function. The value for such a constraint is given by the implicit function's value at the constraint location. A normal constraint is drawn as a spike that is fixed at one end to the disk of its corresponding boundary point. The user may drag the free end of this spike to adjust the normal to the surface, and the surface follows this new constraint. Figure 8 shows an example of changing a normal constraint during an interactive modelling session.

What has been gained by using interpolating implicit functions instead of blobby spheres and cylinders? First, the interpolating implicit approach is easier to implement because the optimization machinery for control points of blobby implicits is not needed. Second, the user has control over the surface normal as well as the surface position. Finally, the user does not need to specify which implicit parameters are to be fixed and which are to be free at different times during the editing session. Using the blobby formulation, the user must choose, at any given time, which parameters such as sphere centers, radii of influence, and cylinder endpoints may be altered by moving a control point. With the variational

formulation, the user is always changing the position of just a single boundary or normal constraint. We believe that this direct control of the parameters of the implicit function is more natural and intuitive. Witkin and Heckbert [1994] state the following:

> Another result of this work is that we have discovered that implicit surfaces are slippery: *when you attempt to move them using control points they often slip out of your grasp*. (emphasis from the original paper)

In contrast to blobby implicits, we have found that *interpolating* implicit surfaces are not at all slippery. Users easily grasp and re-shape these surfaces with no thought to the underlying parameters of the model.

6. OBJECT BLENDING

A *blend* is a portion of a surface that smoothly joins two sub-parts of an object. One of the more useful attributes of implicit surfaces is the ease with which they allow two objects to be blended together. Simply summing together the implicit functions for two objects often gives quite reasonable results for some applications. In some instances, however, traditional implicit surface methods have been found to be problematic when creating certain kinds of blends. For example, it is difficult to get satisfactory results when summing together the implicit functions for two branches and a trunk of a tree. The problem is that the surface will bulge at the location where the trunk and the two branches join. Bulges occur because the contribution of multiple implicit functions causes their sum to take on large values in the blend region, and this results in the new function reaching the iso-surface threshold in locations further away from the blend than is desirable. Several solutions have been proposed for this problem of bulges in blends, but these methods are either computationally expensive or are fairly limited in the geometry for which they can be used. For an excellent description of various blending methods, see Chapter 7 of Bloomenthal [1997].

Interpolating implicit surfaces provide a new way in which to create blends between objects. Objects that are blended using this new approach are free of the bulging problems found using some other methods. Our approach to blending together surfaces is to form one large collection of constraints by collecting together the constraints that define of all the surfaces to be blended. The new blended surface is the surface defined by this new collection of constraints. It is important to note that simply using *all* of the constraints from the original surfaces will usually produce poor results. The key to the success of this approach is to throw out those constraints that would cause problems.

Consider the task of blending together two shapes A and B. If we used all of the constraints from both shapes, the resulting surface is not likely to be what we wish. The task of selecting which constraints to keep is simple. Let $f_A(\mathbf{x})$ and $f_B(\mathbf{x})$ be the implicit functions for shapes A and B respectively. We will retain those constraints from object A that are outside of B. That is, a constraint from A with position $\mathbf{c_i}$ will be kept if $f_B(\mathbf{c_i}) < 0$. All other constraints from A will be discarded. Likewise, we will keep only those constraints from object B that are outside of object A. To create a blended shape, we collect together all of the constraints that pass these two tests and form a new surface based on these constraints.

This approach can used to blend together any number of objects. Figure 9 (left) shows three polygonal tori that overlap one another in 3D. To blend these objects together, we first create a set of boundary and normal constraints for each object, using the approach described in Section 4.3. We then keep only those constraints from each object that are outside of each of the other two objects, as determined by their implicit functions. Finally, we create a single implicit function using all of the constraints from the three objects that were retained. Figure 9 (right) shows the result of this procedure. Notice that there are no bulges in the locations where the tori meet.

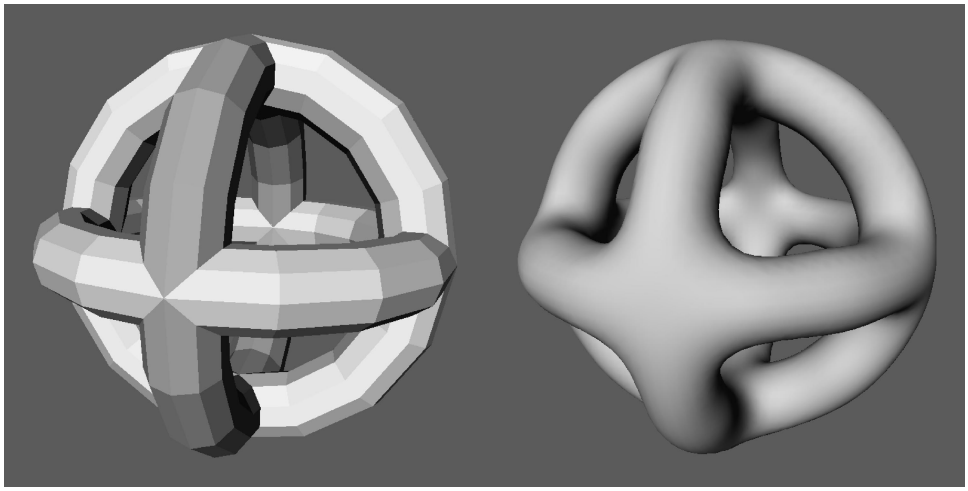


Fig. 9. Three polygonal tori (left), and the soft union created with interpolating implicits (right).

7. RENDERING

In this section we examine two traditional approaches for rendering implicit surfaces that both perform well for interpolating implicits.

7.1 Conversion to Polygons

One way to render an implicit surface is to create a set of polygons that approximate the surface and then render these polygons. The topic of iso-surface extraction is well-studied, especially for regularly sampled volumetric data. Perhaps the best known approach of this type is the Marching Cubes algorithm [Lorensen and Cline 1987], but a number of variants of this method have been described since the time of its publication.

We use a method of iso-surface extraction known as a *continuation* approach [Bloomenthal 1988] for many of the figures in this paper. The models in Figure 2 and in the right images of Figures 6 and 9 are collections of polygons that were created using the continuation method. This method first locates any position that is on the surface to be tiled. This first point can be thought of as a single corner of a cube that is one of an infinite number of cubes in a regular lattice. The continuation method then examines the values of the implicit function at neighboring points on the cubic lattice and creates polygons within each cube that the surface must pass through. The neighboring vertices of these cubes are examined in turn, and the process eventually crawls over the entire surface defined by the implicit function. We use the implementation of this method from Bloomenthal [1994] that is described in detail in Bloomenthal [1988].

7.2 Ray Tracing

There are a number of techniques that may be used to ray trace implicit surfaces, and a review of these techniques can be found in Hart [1993]. We have produced ray traced images of interpolating implicit surfaces using a particular technique introduced by Hart [1997] that is known as *sphere tracing*. Sphere tracing is based on the idea that we can find the intersection of a ray with a surface by traveling along the ray in steps that are small enough to avoid passing through the surface. At each step along the ray the method conservatively estimates the radius of a sphere that will not intersect the surface. We declare that we are near enough to the surface when the value of $f(\mathbf{x})$ falls below some tolerance ε. We

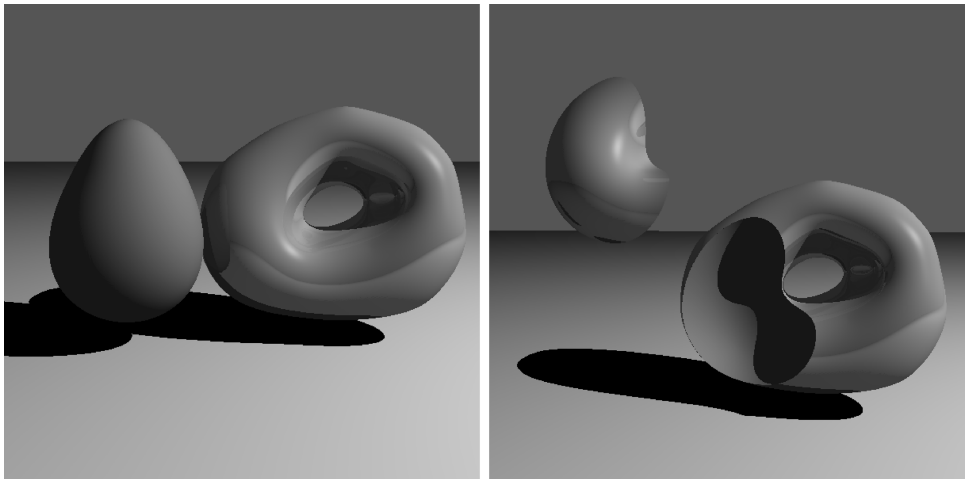


Fig. 10. Ray tracing of interpolating implicit surfaces. The left image shows reflection and shadows of two implicit surfaces, and the right image illustrates constructive solid geometry.

currently use a heuristic to determine the radius of the spheres during ray tracing. We sample the space in and around our implicit surface at 2000 positions, and we use the maximum gradient magnitude over all of these locations as the Lipschitz constant for sphere tracing. For extremely pathological surfaces, this heuristic may fail, although it has worked well for all of our images. Coming up with a sphere radius that is guaranteed not to intersect the surface is a good area for future research. We think it is likely that other ray tracing techniques can also be successfully applied to ray tracing of interpolating implicits, such as the LG-surfaces approach of Kalra and Barr [1989].

Figure 10 (left) is an image of two interpolating implicit surfaces that were ray traced using sphere tracing. Note that this figure includes shadows and reflections. Figure 10 (right) illustrates constructive solid geometry with interpolating implicit surfaces. The figure shows (from left to right) intersection and subtraction of two implicit surfaces. This figure was created using standard ray tracing CSG techniques as described in Roth [1982].

The rendering techniques of this section highlight a key point—interpolating implicit surfaces may be used in almost all of the contexts in which other implicit formulations have been used. This new representation may provide fruitful alternatives for a number of problems that use implicit surfaces.

8. COMPARISON TO RELATED METHODS

At this point it is useful to compare interpolating implicit surfaces to other representations of surface geometry. Although they share similarities with existing techniques, interpolating implicits are distinct from other forms of surface modeling. Because interpolating implicits are not yet well known, we provide a comparison of them to two more well-known modelling techniques.

8.1 Thin-Plate Surface Reconstruction

The scientific and engineering literature abound with surface reconstruction based on thin-plate interpolation. Aren't interpolating implicits just a slight variant on thin-plate techniques? The most important difference is that traditional thin-plate reconstruction creates a *height field* in order to fit a given set of data points. The use of a height field is a barrier towards creating closed surfaces and surfaces of arbitrary topology. For example, a height field cannot even represent a simple sphere-like

object such as the surface shown in Figure 2 (left). Complex surfaces can be constructed using thin-plate techniques only if a number of height fields are stitched together to form a parametric quilt over the surface. This also pre-supposes that the topology of the shape to be modelled is already known. Interpolating implicit surfaces, on the other hand, do not require multiple patches in order to represent a complex model. Both methods create a function based on variational methods, but they differ in the dimension of the scalar function that they create. Traditional thin-plate surfaces use a function with a 2D domain to create a *parametric* surface, whereas the interpolating implicit method uses a function with a 3D domain to specify the location of an *implicit* surface.

8.2 Sums of Implicit Primitives

Section 3 shows that an interpolating implicit function is in fact a sum of a number of functions that have radial symmetry (based on the $|\mathbf{x}|^3$ function). Isn't this similar to constructing an implicit function by summing a number of spherical Gaussian functions (blobby spheres or meta-balls)? Let us consider the process of modeling a particular shape using blobby spheres. The unit of construction is the single sphere, and two decisions must be made when we add new sphere to a model: the sphere's center and its radius. We cannot place the center of the sphere where we want the surface to be—we must displace it towards the object's center and adjust its radius to compensate for this displacement. What we are doing is much like guessing the location of the medial axis of the object that we are modeling. (The medial axis is the locus of points that are equally distant from two or more places on an object's boundary.) In fact, the task is more difficult than this because summing multiple blobby spheres is not the same as calculating the union of the spheres. The interactive method of Witkin and Heckbert [1994] relieves the user from some of this complexity, but still requires the user to select which blobby primitives are being moved and which are fixed. These issues never come up when modeling using interpolating implicit surfaces because we can directly specify locations that the surface must pass through.

Fitting blobby spheres to a surface is an art, and indeed many beautiful objects have been sculpted in this manner. Can this process be entirely automated? Muraki [1991] demonstrated a way in which a given range image may be approximated by blobby spheres. The method begins with a single blobby sphere that is positioned to match the data. Then the method repeatedly selects one blobby sphere and splits it into two new spheres, invoking an optimization procedure to determine the position and radii of the two spheres that best approximates the given surface. Calculating a model composed of 243 blobby spheres "took a few days on a UNIX workstation (Stardent TITAN3000 2 CPU)." Similar blobby sphere data approximation by Bittar et al. [1999] was limited to roughly 50 blobby spheres. In contrast to these methods, the bunny in Figure 6 (right) is an interpolating implicit surface with 800 boundary and 800 normal constraints. It required 1 minute 43 seconds to solve the matrix equation for this surface, and the iso-surface extraction required 7 minutes 43 seconds. Calculations were performed on an SGI O2 with a 195 MHz R10000 processor.

9. CONCLUSION AND FUTURE WORK

In this paper we have introduced new approaches for model creation using interpolating implicit surfaces. Specific advantages of this method include:

- Direct specification of points on the implicit surface
- Specification of surface normals
- Conversion of polygon models to smooth implicit forms
- Intuitive controls for interactive sculpting
- Addition of new control points that leave the surface unchanged (like knot insertion)
- A new approach to blending objects

A number of techniques have been developed for working with implicit surfaces. Many of these techniques could be directly applied to interpolating implicits, indicating several directions for future work. The critical point analysis of Stander and Hart [1997] could be used to guarantee topologically correct tessellation of such surfaces. Interval techniques, explored by Duff, Snyder and others, might be applied to tiling and ray tracing of interpolating implicits [Duff 1992; Snyder 1992]. The interactive texture placement methods of Pedersen [1995; 1996] should be directly applicable to interpolating implicit surfaces. Finally, many marvelous animations have been produced using blobby implicit surfaces [Blinn 1982; Wyvill et al. 1986]. We anticipate that the interpolating properties of these implicit surfaces may provide animators with an even greater degree of control over implicit surfaces.

Beyond extending existing techniques for this new form of implicit surface, there are also research directions that are suggested by issues that are specific to our technique. Like blobby sphere implicits, interpolating implicit surfaces are everywhere smooth. Perhaps there are ways in which sharp features such as edges and corners can be incorporated into an interpolating implicit model. We have shown how gradients of the implicit function may be specified indirectly, using positive constraints that are near zero constraints, but it may be possible to modify the approach to allow the exact specification of the gradient.

Another direction for future research is to find higher-level interactive modelling techniques for creating these implicit surfaces. Perhaps several new constraints could be created simultaneously, maybe arranged in a line or in a circle for greater surface control. It might also make sense to be able to move the positions of more than one constraint at a time. Another modelling issue is the creation of surfaces with boundaries. Perhaps a second implicit function could specify the presence or absence of a surface. Another issue related to interactivity is the possibility of displaying the surface with polygons rather than with floaters. With sufficient processor power, creating and displaying a polygonal isosurface of the implicit function could be done at interactive rates.

ACKNOWLEDGMENTS

We thank the members of the Georgia Tech Geometry Group for their ideas and enthusiasm. Also thanks to Victor Zordan for help with video. Finally, we are grateful to the reviewers for their suggestions to improve this paper.

REFERENCES

BITTAR, E., TSINGOS, N., AND GASCUEL, M.-P. 1995. Automatic reconstruction of unstructured 3D data: Combining a medial axis and implicit surfaces. *Computer Graphics Forum (Proceedings of Eurographics '95) 14,* 3, 457–468.

BLINN, J. F. 1982. A generalization of algebraic surface drawing. *ACM Trans. Graph. 1,* 3, 235–256.

BLOOMENTHAL, J. 1988. Polygonization of implicit surfaces. *Computer-Aided Geometric Design 5,* 4, 341–355.

BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press, Cambridge, 324–349.

BLOOMENTHAL, J. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers, Inc., San Francisco, CA.

CARR, J. C., MITCHELL, T. J., BEATSON, R. K., CHERRIE, J. B., FRIGHT, W. R., MCCALLUM, B. C., AND EVANS, T. R. 2001. Reconstruction and representation of 3d objects with radial basis functions. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2001)*, 67–76.

CELNIKER, G. AND GOSSARD, D. 1991. Deformable curve and surface finite-elements for free-form shape design. *Computer Graphics (SIGGRAPH 91) 25,* 4 (July), 257–266.

DUCHON, J. 1977. Spline minimizing rotation-invariant semi-norms in Sobolev spaces. In *Constructive Theory of Functions on Several Variables, Lecture Notes in Mathematics 571*, W. Schempp and K. Zeller, Eds. Springer-Verlag, Berlin.

DUFF, T. 1992. Interval arithmetic and recursive subdivision for implicit functions and constructive solid geometry. *Computer Graphics (SIGGRAPH 92) 26,* 2 (July), 154–168.

DYN, N. 1987. Interpolation of scattered data by radial basis functions. In *Topics in Multivariate Approximation*, L. L. S. C. K. Chui and F. I. Utreras, Eds. Academic Press, Cambridge, 47–61.

GIROSI, F., JONES, M., AND POGGIO, T. 1993. Priors, stabilizers and basis functions: from regularization to radial, tensor and additive splines. Tech. rep., MIT Artificial Intelligence Laboratory. June. A.I. Memo No. 1430.

GRIMSON, W. E. L. 1983. Surface consistancy constraints in vision. *Computer Vision, Graphics, and Image Processing 24,* 1 (Oct.), 28–51.

HART, J. 1993. Ray tracing implicit surfaces. *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, 1–16.

HART, J. 1997. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12,* 10, 527–545.

KALRA, D. AND BARR, A. 1989. Guarenteed ray intersection with implicit surfaces. *Computer Graphics (SIGGRAPH 89) 23,* 4, 297–306.

KEREN, D. AND GOTSMAN, C. 1998. Tight fitting of convex polyhedral shapes. *Int. J. Shape Modeling*, 111–126.

LORENSEN, W. AND CLINE, H. E. 1987. Marching cubes: A high resolution 3-D surface construction algorithm. *Computer Graphics (SIGGRAPH 87) 21,* 4 (July), 163–169.

MIRAKI, S. 1991. Volumetric shape description of range data using 'blobby model'. *Computer Graphics (SIGGRAPH 91) 25,* 4 (July), 227–235.

MORSE, B., YOO, T. S., RHEINGANS, P., CHEN, D. T., AND SUBRAMANIAN, K. 2001. Interpolating implicit surfaces from scattered surface data using compactly supported radial basis functions. *Shape Modelling International*, 89–98.

NISHIMURA, H., HIRAI, M., KAWAI, T., KAWATA, T., SHIRKAWA, I., AND OMURA, K. 1985. Object modeling by distribution function and a method of image generation. *Trans. Inst. Elect. Commun. Eng. Japan J68-D,* 4, 718–725.

PEDERSEN, H. 1995. Decorating implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 95)*, 291–300.

PEDERSEN, H. 1996. A framework for interactive texturing on curved surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 96)*, 295–302.

ROTH, S. 1982. Ray casting as a method for solid modeling. *Computer Graphics and Image Processing 18,* 2, 109–144.

SAVCHENKO, V. V., PASKO, A. A., OKUNEV, O. G., AND KUNNI, T. L. 1995. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum 14,* 4 (Oct.), 181–188.

SNYDER, J. 1992. Interval analysis for computer graphics. *Computer Graphics (SIGGRAPH 92) 26,* 2 (July), 121–130.

STANDER, B. T. AND HART, J. C. 1997. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 97)*, 279–286.

SZELISKI, R. 1990. Fast surface interpolation using hierarchical basis functions. *IEEE Trans. Pattern Anal. Mach. Intell. 12,* 6 (June), 513–528.

TAUBIN, G. 1993. An improved algorithm for algebraic curve and surface fitting. In *Fourth International Conference on Computer Vision (ICCV '93)*. IEEE, Berlin, Germany, 658–665.

TERZOPOULOS, D. 1988. The computation of visible-surface representations. *IEEE Trans. Pattern Anal. Mach. Intell. 10,* 4 (July), 417–438.

TURK, G. AND O'BRIEN, J. 1999. Shape transformation using variational implicit functions. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1999)*, 335–342.

WELCH, W. AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, 247–256.

WITKIN, A. P. AND HECKBERT, P. S. 1994. Using particles to sample and control implicit surfaces. *Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 94)*, 269–278.

WYVILL, G., MCPHEETERS, C., AND WYVILL, B. 1986. Data structures for soft objects. *The Visual Computer 2,* 4, 227–234.

Received July 2001; revised April 2002; accepted May 2002

Robust Creation of Implicit Surfaces from Polygonal Meshes

Gary Yngve and Greg Turk, *Member*, *IEEE*

Abstract—Implicit surfaces are used for a number of tasks in computer graphics, including modeling soft or organic objects, morphing, collision detection, and constructive solid geometry. Although operating on implicit surfaces is usually straightforward, creating them is not. We introduce a practical method for creating implicit surfaces from polygonal models that produces high-quality results for complex surfaces. Whereas much previous work in implicit surfaces has been done with primitives such as "blobbies," we use implicit surfaces based on a variational interpolation technique (the three-dimensional generalization of thin-plate interpolation). Given a polygonal mesh, we convert the data to a volumetric representation to use as a guide for creating the implicit surface iteratively. We begin by seeding the surface with a number of constraint points through which the surface must pass. Iteratively, additional constraints are added; the resulting surfaces are evaluated, and the errors guide the placement of subsequent constraints. We have applied our method successfully to a variety of polygonal meshes and consider it to be robust.

Index Terms—Geometric modeling, surface representations, implicit surfaces.

1 Introduction

The task of constructing smooth surfaces is ubiquitous throughout computer graphics. Often, parametric surfaces are the choice representation because of the capabilities of many commercial modeling packages. Once constructed, the parametric surfaces are then used in a variety of graphics algorithms, ranging from ray-tracing to morphing. However, many of these graphics algorithms have more elegant solutions when used with implicit surfaces. For implicit surfaces to become more widely used, however, they must become easier to create. We approach this issue by introducing a new method to convert polygonal surfaces to smooth implicit surfaces automatically.

Because points can be evaluated easily as being inside or outside an implicit surface, many applications that are challenging for parametric surfaces (including polygonal meshes) become simple when implicit surfaces are used. Boolean CSG operations (union, intersection, subtraction) reduce to simply examining the signs of the implicit functions. Operations on implicit surfaces that may cause the genus of the surface to change have simple implementations because the operations affect every point in space—on the isosurface, inside, and outside. Shape morphing can be performed simply by interpolating between two implicit functions, and the two shapes can be of arbitrary manifold topology [1], [2], [3]. Implicit surfaces can collide and deform [4], [5]; the resulting fusions and separations are handled automatically. Often, in graphics, implicit functions are created by summing many infinitely differentiable functions, yielding surfaces that are smooth and seamless. The forms that they can represent are useful for modeling organic shapes and some classes of machine parts that require blends and fillets.

Although implicit surfaces have many benefits, they can be difficult to model (as well as to texture and render). Most parametric surface representations use basis functions with finite support and, thus, give the user an easy way to perform local control of the surface shape. In contrast, the bases that are used as primitives for implicit surfaces can often have nonobvious influences on surface position. Modeling with "blobbies" [6] suffers from this problem because each blobby primitive only indirectly influences the position of the isosurface. We note that the work of Witkin and Heckbert is aimed at overcoming this difficulty [7].

In our approach, rather than using the more traditional blobby primitives approach to implicit surface creation, we instead use variational implicit surfaces. This form of implicit surface allows a user to specify locations that the surface will exactly interpolate; this property allows more direct control over surface creation. As we will describe more fully later, solving a set of linear equations will guarantee that the surface interpolates a given set of constraint points. In addition to this interpolating property, variational implicit functions are smooth when their basis functions are chosen to satisfy an energy functional related to the desired degree of smoothness. Our approach to creating these surfaces is to add new constraints iteratively until the model is a close approximation to the input polygonal mesh. Fig. 1 shows 23 iterations of our algorithm while creating a frog model.

We focus on the creation of implicit models from polygonal meshes because of the large number of existing high-quality polygonal meshes. Having a robust automatic conversion procedure from meshes to implicits should provide a pathway towards creating a large library of implicit surfaces. With such a technique, all of the interactive modeling tools for creating polygonal meshes can then be used to create implicit surfaces. This ability

- *G. Yngve is with the Department of Computer Science and Engineering, University of Washington, 114 Sieg Hall, Box 352350, Seattle, WA 98195. E-mail: gyngve@cs.washington.edu.*
- *G. Turk is with the GVU Center, College of Computing, Georgie Institute of Technology, 801 Atlantic Drive, Atlanta, GA 30332-0280. E-mail: turk@cc.gatech.edu.*

Manuscript received 17 Nov. 2000; revised 6 July 2001; accepted 30 July 2001. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number 113171.

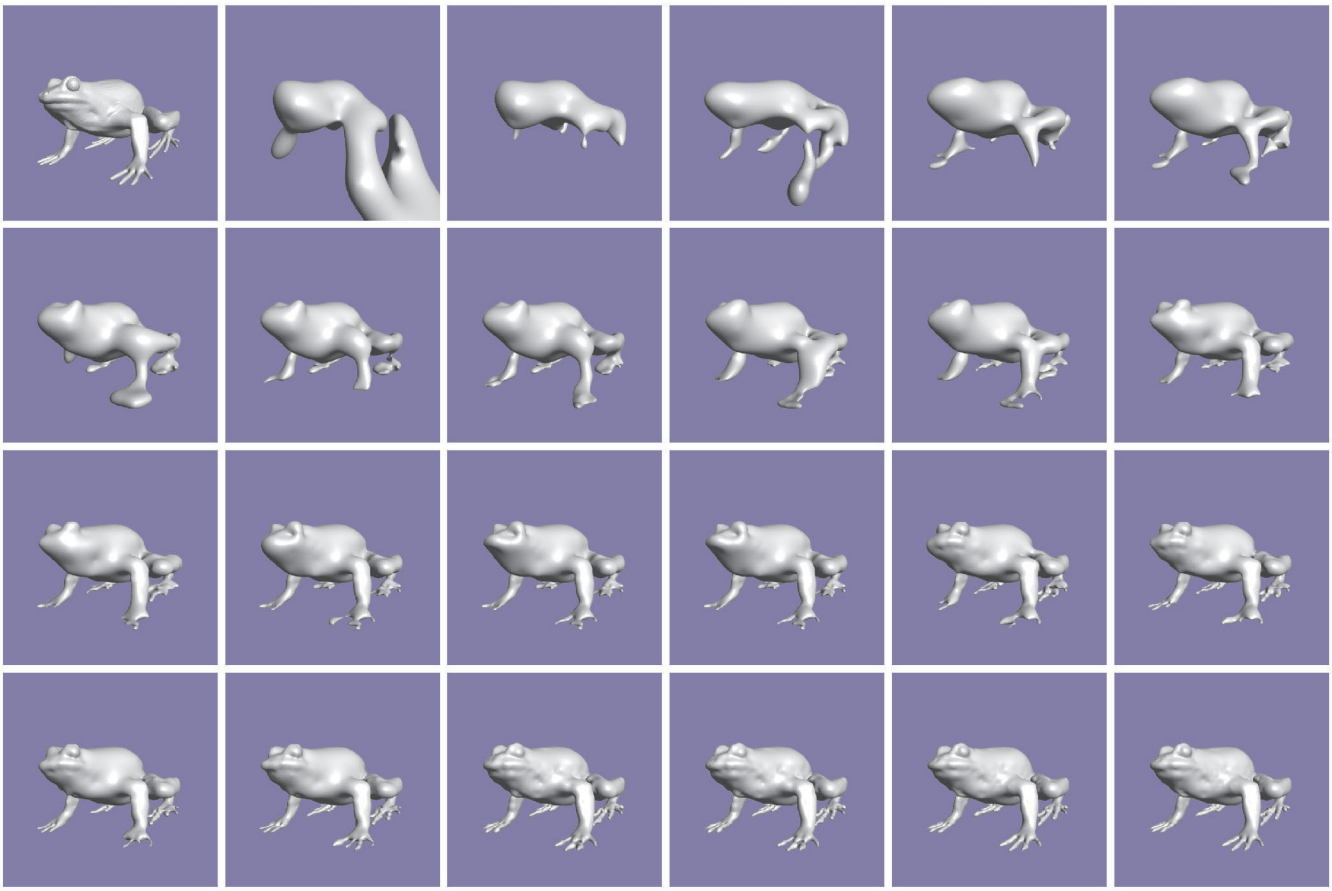


Fig. 1. A polygonal frog (top left) is converted to an implicit surface via our incremental improvement algorithm. The algorithm took 23 iterations to reach the final result. The results from each iteration are shown in successive images. The frog is a near fit after the first three rows; in the last row the toes get refined.

would mean that we can avoid having to create special purpose modeling programs for implicit surfaces.

The rest of the paper will proceed as follows: We briefly discuss previous work in implicit surface modeling in Section 2. In Section 3, we explain the variational implicit surface representation. Then, in Section 4, we introduce a set of tools that will be used by the algorithm. We present the algorithm in Section 5, analyze the algorithm's parameters in Section 6, and show results in Section 7. Finally we conclude and discuss future work in Section 8.

2 Previous Work

The very first implicit surfaces used in computer graphics were quadrics (degree-two polynomials of x, y, and z), such as spheres, ellipsoids, and cylinders [8]. Blinn generalized these implicit surfaces for the purpose of modeling molecules [6]. Basing his model on electron densities, he developed the blobby molecule model, which consists of Gaussian-like primitives blended together:

$$f_i(x) = A_i e^{b_i \, ||x - c_i||^2}. \quad (1)$$

Each primitive is a radial basis function that can be tuned to control its size and blobbiness (its tendency to blend). This method and its variants [9] are widely used in the computer graphics community. As mentioned earlier, however, this form of implicit surface does not allow a user to directly specify points that the surface interpolates. The user must somehow estimate the location of the middle of the shape because this is where the centers of the primitives must be placed.

Another genre of implicit surfaces is the convolution surface [10]. These surfaces are created by convolving a skeleton shape (e.g., a collection of polygons) with a kernel such as a Gaussian. The skeletons for the convolutions can actually be any form of geometry, including both 2D surfaces and solid objects. The resulting convolution surface is smooth. As with the blobby implicits, convolution surfaces do not allow a user to give specific points to be interpolated. Recent work by Sherstyuk [11] shows promising results for creating convolution surfaces interactively. Offset surfaces are used to approximate the convolution surfaces during interactive editing.

Interactive modeling techniques can be used to create implicit surfaces of modest complexity. One elegant method for interactive modeling was described by Witkin and Heckbert, in which they use particles to sample and control implicit surfaces [7]. Particles diffuse across the surface and are created and destroyed as necessary. They implemented their technique with blobby spheres and cylinders, and their technique is adaptable to variational implicit surfaces as well. Ferley et al. [12] maintain a cube list representing the isosurface of an implicit function. By updating the cube list, they can achieve interactive editing rates. Their technique is limited to the fixed resolution of the volume,

but wavelet techniques may grant them more flexibility. Still, however, for creating complicated models, more automatic methods are needed.

Muraki developed a method to approximate range data by a blobby implicit surface [13]. Muraki's method incrementally adds primitives one at a time. At each iteration, his algorithm picks a primitive, duplicates it, and then solves an optimization problem to minimize an energy functional. Because this requires solving an optimization problem every iteration, the method is exceedingly slow—a model with 243 primitives took a few days to create on a Stardent Titan3000 2CPU.

Bittar et al. addressed the modeling of an implicit surface from volume data [14]. They calculate a medial axis of the volume data as an aid to implicit function creation. They then use an optimization scheme based on Muraki's work to add positive primitives along the medial axis in substantially less time than Muraki's approach. They also developed a similar method in which the user dissects an object into intuitive reconstruction windows [15]. The optimization problems run inside the individual reconstruction windows for a significant gain in speed. However, the implicit surfaces that they generated with their method were small (the largest had only about 50 primitives). These three methods only calculate a one-way error, namely for the points on the goal surface, seeing how far the isosurface deviates away. To prevent stray components of the implicit surface that the error metric would not be able to catch, they use an E_{shrink} term that minimizes the field of influence for each primitive. Our method, on the other hand, uses a two-way error metric. Another difference between our method and the above methods is they calculate the deviation of the isosurface by its value at the goal surface, rather than the Euclidean distance of the zero-set from the goal surface. It should also be noted that these methods cannot be applied directly to variational implicit surfaces.

This brief summary barely scratches the surface of work on implicits in computer graphics. For an excellent overview of the area and more details on kinds of implicit surfaces, see the book by Bloomenthal et al. [16].

3 Variational Implicit Surfaces

In this section, we give the equations that describe variational implicit surfaces and outline the algorithm that we use to create such surfaces from polygonal meshes.

3.1 Basic Formulation

Variational implicit surfaces are created by solving a scattered data interpolation problem [17]. The particular solution technique is based on ideas from the calculus of variations (solving an energy minimization problem). To create a variational implicit function, a user specifies a set of k constraint points $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$, along with a set of values $\{h_1, h_2, \ldots, h_k\}$ at the given constraint positions. The surfaces are controlled directly using three types of constraints. *Boundary constraints* are those positions that are specified to take on the value zero, and the created implicit surface will exactly pass through these points. In addition, we can specify that certain points will be interior or exterior to the surface. *Interior constraints* are given positive values, and *exterior constraints* are given negative values. To create the appropriate implicit function, these constraints are handed to a sparse data interpolation routine that creates a function that exactly matches the given constraints.

The form of the function created by this technique is a weighted set of radial basis functions and a polynomial term. The weights of the basis function are found by solving a matrix equation (given below). We use the radial basis function $\phi(\mathbf{x}) = |\mathbf{x}|^3$, which minimizes the curvature functional

$$\int_{\mathbf{x}\in\Omega} \sum_{i,j} \left(\frac{\partial^2 \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}_i \partial \mathbf{x}_j} \right)^2 \mathrm{d}\mathbf{x}. \tag{2}$$

The reason why $\phi(\mathbf{x}) = |\mathbf{x}|^3$ minimizes this functional is nonobvious, and we refer the interested reader to [18] for details. It is important to note that this functional does not represent curvature on the isosurface (the 2-manifold $f(x) = 0$ embedded in bounded region Ω), but rather the aggregate curvature of f over the entire region.

Using this radial basis function, the implicit function that we create has the form

$$f(\mathbf{x}) = \sum_{j=1}^{n} d_j \phi(\mathbf{x} - \mathbf{c}_j) + P(\mathbf{x}). \tag{3}$$

In the above equation, $\mathbf{c}_j$ are the locations of the constraints, the d_j are the weights, and $P(\mathbf{x})$ is a first-degree polynomial that accounts for the linear and constant portions of f.

To solve for the set of d_j that will satisfy the interpolation constraints $h_i = f(\mathbf{c}_i)$, we can substitute the right side of (3) for $f(\mathbf{c}_i)$, which gives:

$$h_i = \sum_{j=1}^{k} d_j \phi(\mathbf{c_i} - \mathbf{c_j}) + P(\mathbf{c_i}). \tag{4}$$

Because (4) is linear with respect to the unknowns, d_j and the coefficients of $P(\mathbf{x})$, it can be formulated as simple matrix equation. For interpolation in three dimensions, let $c_i = (c_i^x, c_i^y, c_i^z)$ and let $\phi_{ij} = \phi(\mathbf{c}_i - \mathbf{c}_j)$. Then, the linear system can be written as the following matrix equation:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \ldots & \phi_{1k} & 1 & c_1^x & c_1^y & c_1^z \\ \phi_{21} & \phi_{22} & \ldots & \phi_{2k} & 1 & c_2^x & c_2^y & c_2^z \\ \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{k1} & \phi_{k2} & \ldots & \phi_{kk} & 1 & c_k^x & c_k^y & c_k^z \\ 1 & 1 & \ldots & 1 & 0 & 0 & 0 & 0 \\ c_1^x & c_2^x & \ldots & c_k^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \ldots & c_k^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \ldots & c_k^z & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_k \\ p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_k \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{5}$$

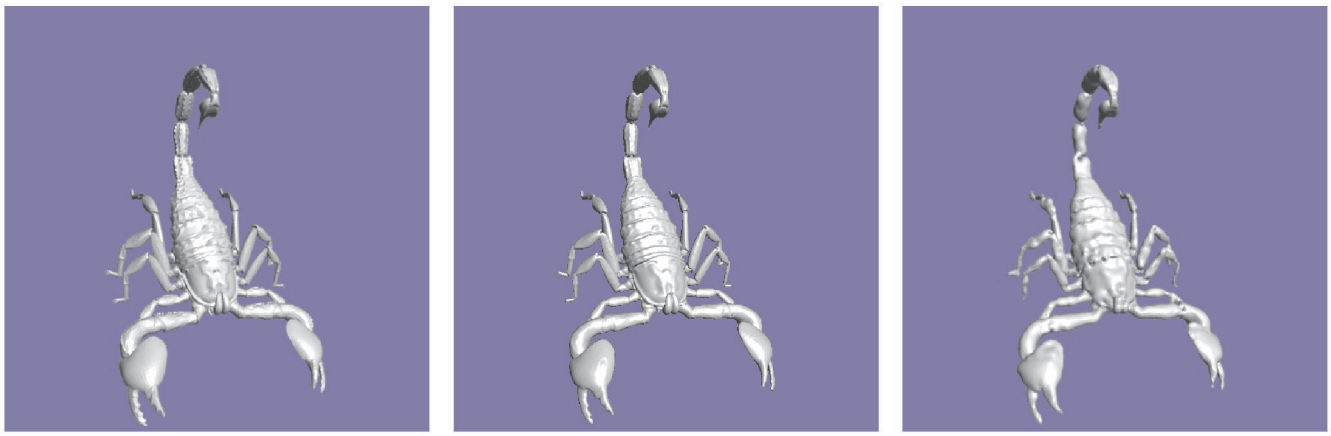


Fig. 2. Original polygonal model of a scorpion (a), intermediate volumetric representation (b), and final implicit surface (c) generated by our algorithm. The volumetric representation captures all but the finest detail, such as the hairs on the tail. Our algorithm refined the implicit surface using the volumetric model as the goal.

The above matrix system is symmetric and is the standard one used to solve this type of interpolation problem [18]. We used LU decomposition to solve this system for all of the examples shown in this paper. With the coefficients from the matrix solution (the ds and ps), evaluating the implicit function from (3) becomes simple. We refer the interested reader to [3], [17], [19] for more mathematical details about this form of implicit function.

3.2 Outline of Our Approach

We will now examine how the constraints for a variational implicit surface can be derived from a polygonal model. This task is easy for models that are composed of polygons that are all nearly the same size. For such a polygonal model, we may use the vertices of the model as the positions of the boundary constraints. Similarly, we can create exterior constraints by moving out from each vertex in the normal direction. This basic technique was originally described in [17]. Unfortunately, most models are made of polygons that are widely varying in size, and for such models it is more difficult to create a variational implicit that faithfully matches a given polygonal model.

To produce high-quality implicit models from polygons, we created an iterative method that repeatedly adds new constraints to a variational implicit representation in a manner that is guided by a volumetric description of the model. To do so, we use a voxelization process to create the volumetric model from the polygons. The volumetric description of the given model acts as an ideal (but storage-intensive) implicit representation of the model that we can use to compare against the current variational implicit surface. Another possibility would have been to use just the mesh vertices to guide the implicit surface, but this would undersample the surface where the original mesh polygons are large. In addition to the volumetric model, we also use a signed distance function to measure errors in the current iteration and to place new boundary, interior and exterior constraints. We repeatedly add new constraints until the implicit model is a near match to the original. In Section 4, we discuss the creation of the volumetric model, the signed distance function and the error metrics for evaluating the model. Later, we describe in detail, how they are used to define new constraints to make a variational implicit surface approximating our original polygonal model.

4 Volumes and Error Metrics

We choose to convert the polygonal model into a volumetric model due to its convenience for rapid evaluation of inside/outside queries. The disadvantages of a volumetric model are storage and computation costs. Using wavelets or subdivision could reduce the complexity; we delegate investigation of such techniques to future work.

4.1 Voxelization of a Polygonal Mesh

To convert a polygonal model into a volumetric representation, we cast a grid of parallel rays through the mesh and regularly sample the points along these rays. Each sample becomes one voxel. A parity count of surface crossings determines interior from exterior. To minimize aliasing artifacts we perform supersampling and filtering so that the final densities vary continuously between zero and one. Further details of the voxelization process, including variations to handle troublesome meshes, can be found in [20]. For example, Fig. 2 shows the original polygonal scorpion model (Fig. 2a), the intermediate volumetric model (Fig. 2b), and the final implicit representation generated by our algorithm (Fig. 2c). The intermediate volumetric model captures all but the finest detail, such as the hairs on the tail, and serves as the goal surface for the surface evaluation and refinement. Note that, if one has a volumetric representation of a given model, our method can be used directly to produce an implicit surface. Unfortunately most models do not originally come in a volumetric form, hence, our need for conversion from polygons to voxels. A number of other researchers have also produced methods of voxelizing polygons [21], [22], [23], and any of these methods could have been used.

4.2 Signed Distance Transform

We use a signed distance transform to measure the error between our volumetric model and a given implicit representation. We use the voxelization of a given object as an inside-outside function of the object, and for this purpose we clamp all densities to either zero or one. A *distance transform* of an inside-outside function is the distance of a point to the nearest boundary (the transition regions between densities of zero and one). A *signed* distance transform negates the distances of those points that are outside the object. We use a three-dimensional version of Danielsson's method for computing Euclidean distances [24] to compute the signed distance in $O(n^3)$. This method requires making a small fixed number of sweeps through the entire volume, and at each voxel only a few neighbors are examined. The final result is a set of distances at each voxel that is a close approximation to the (signed) Euclidean distance to the nearest boundary. We could also have used one of the published methods for converting polygons directly to a distance field, such as [1], [25]. We chose our two-step approach for speed and to allow subvoxel constraint placement.

4.3 Error Metric

To guide our surface creation method, we use a metric to evaluate how closely our current variational implicit surface matches the original data. Let ∂f_{curr} be the set of boundary voxels in the volumetric representation of the current implicit surface, and let ∂f_{goal} be the boundary voxels of the goal, that is, the volumetric representation of the original data. We want ∂f_{curr} to equal ∂f_{goal}, and we can measure the symmetric difference by the Hausdorff metric

$$H = \max\left[\max_{x\in\partial f_{goal}}\left[\min_{y\in\partial f_{curr}}||x-y||\right], \max_{x\in\partial f_{curr}}\left[\min_{y\in\partial f_{goal}}||x-y||\right]\right]. \tag{6}$$

The Hausdorff metric is zero if and only if ∂f_{curr} and ∂f_{goal} are identical. Furthermore, we can identify the voxels x farthest from the other surface and refine the surface by placing constraints at those locations. Using the signed distance functions $sd_{goal}(x)$ and $sd_{curr}(x)$ for the goal and current surfaces as lookup tables, the new constraint location is defined concisely and calculated efficiently as

$$C_{new} = \arg\max\left[\max_{x\in\partial f_{goal}}|sd_{curr}(x)|, \max_{x\in\partial f_{curr}}|sd_{goal}(x)|\right]. \tag{7}$$

The error metric has a two-fold purpose: to evaluate the attempted fit and to suggest where to refine the implicit representation further. Fig. 3 illustrates the use of the metric as part of our algorithm on a two-dimensional teapot. Note that, for 2D objects, the iso-valued set is one or more closed contours. In the black and white portions of this figure, black denotes interior (positive function values) and white denotes exterior (negative values). Fig. 3a, Fig. 3b, Fig. 3c, Fig. 3d, and Fig. 3e show the refinement at the third iteration. Image (a) shows the signed distance function of the current implicit curve ($sd_{curr}(x)$) with the boundary of the goal (∂f_{goal}) overlaid. Similarly, Fig. 3b shows $sd_{goal}(x)$ with ∂f_{curr} overlaid. Positive values of the signed distance are blue and negative red. As the magnitude increases, the colors go from dark to light. To find the locations to place new constraints, we simply walk around the overlaid boundaries in Fig. 3a and Fig. 3b and choose points with the brightest background color (furthest distance from the other curve). The newly chosen constraints are shown as magenta dots in Fig. 3c (boundary constraints) and Fig. 3d (interior and exterior constraints). The implicit curve with these refinements is in Fig. 3e. Fig. 3f, Fig. 3g, Fig. 3h, Fig. 3i, and Fig. 3j show the respective information for the refinement at the seventh iteration. After later iterations, the implicit curve becomes nearly indistinguishable from the original data.

5 Iterative Improvement of Model

We now describe how the above tools let us model implicit surfaces. Using (7) to find candidate locations for new constraints, we can design an iterative method to refine the surfaces. Below is pseudocode for our algorithm.

```
Algorithm MakeImplicitSurface(Volume f_goal,
          SDFunc sd_goal)
Begin
  Constraints = InitialConstraints(NumInit)
  Repeat
    f_curr = MakeImplicit(Constraints)
    sd_curr = SignedDist(f_curr)
    GenerateCandidateConstraints()
    Repeat
      PruneCandidateList()
      NewCandidate = SelectHighestError()
      Constraints.add(NewCandidate)
    Until NoMoreCandidates
  Until DoneRefining
End
```

First, `NumInit` initial constraints are chosen, discussed in Section 5.1. Then, for each iteration, the following steps are taken: The implicit surface for the current set of constraints is generated by solving (5). The result is evaluated over the volume to obtain f_{curr} (see Section 5.2), and its signed distance sd_{curr} is calculated. New candidate constraints are selected by (7). They may be pruned due to proximity to other constraints or other reasons; see Section 5.3. The remaining candidates become new constraints, and the evaluation-refinement process repeats. The algorithm terminates when the surface fits f_{goal} well enough; termination criteria is discussed in Section 5.4.

5.1 Initialization

The algorithm needs to start with an initial guess for the implicit surface before it can begin refining. We need to decide how many initial constraints to place—we want to create a reasonable first attempt, but we do not want to overwhelm the system with too many constraints. To get a good balance between these two extremes, we sample 50 boundary constraints from the points on the surface ∂f_{goal}. Three-dimensional Poisson-disk sampling bandlimits the spatial frequency of constraints; when constraints are close to each other tend to have more influence on the rest of the system and can cause (5) to be ill-conditioned. We

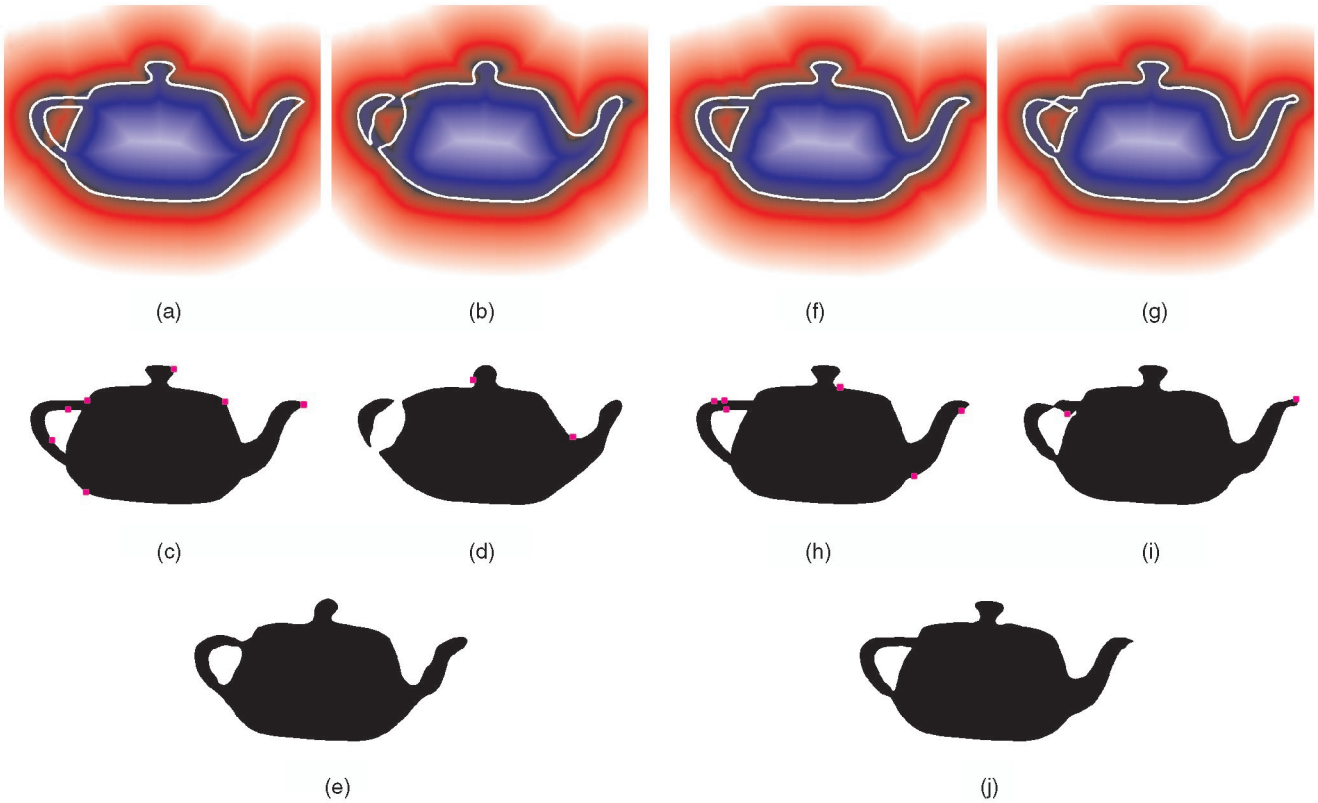


Fig. 3. Creating a two-dimensional implicit curve: To find the locations with greatest errors, we walk around the boundary of the goal curve and see how far it is from the boundary of the current curve. This query is done via a signed distance lookup, seen in (a). (b) Shows the symmetric counterpart, walking around the boundary of the goal curve and using the signed distance of the current curve. Locations of greates errors become new constraints, shown in magenta. (c) Shows the new boundary constraints and (d) shows the new interior and exterior constraints (which appear on the boundary of the incorrect shape). (e) Shows the new curve after these refinements. (f), (g), (h), (i), and (j) show the method applied to a later iteration of the algorithm.

recommend the article by Mitchell for information on Poisson disk sampling [26]. Other methods such as a point-repulsion technique may yield a more uniform sampling, but we feel our simple initialization technique is quite satisfactory.

In addition to choosing boundary constraints, we place nonzero constraints to indicate what portions of space are interior or exterior. It is essential to have these further specifications; otherwise, the iso-surface could fit the goal exactly but the implicit function might be inside-out. These nonzero constraints are weighted by the actual signed distances; constraints more distant from the surface have larger magnitudes. For each of the boundary constraints that we have selected, we follow a path along the gradient of $sd_{goal}(x)$ until we reach a local extremum (or the boundary of the volume) and place a constraint there. The local extrema are the tops of ridges and the bottoms of valleys of the signed distance function—that is, positions on the medial axis of the shape. To decide whether to traverse the gradient uphill or downhill, we try both directions and pick the longer path (hitting the boundary of the volume is by default the longer path). Shorter paths are usually in the direction that is locally concave. By selecting the longer path, our interior and exterior constraints then tend to "fan out" instead of getting clustered in ridges or valleys of the signed distance function. Because the implicit surface is smooth (locally planar), placing the interior or exterior constraints along the gradient of $sd_{goal}(x)$ not only tells the surface what direction is outside but also suggests the surface normal.

5.2 Implicit Function Evaluation

Given a current set of constraints, we solve the variational problem to obtain the basis-function weights for the corresponding implicit surface. Then, the implicit surface is evaluated throughout the volume to find the boundary voxels ∂f_{curr} and the signed-distance function sd_{curr}. Once we classify the boundary voxels and then compute signed distance function, we can evaluate the error metric described in Section 5.

Evaluating the implicit surface throughout the volume can be costly. Although surface-following isosurface-extraction techniques can reduce the complexity by an order of magnitude, they make assumptions about topology, e.g., they may miss a detached portion of the surface. We wish ∂f_{curr} to capture all connected components since they may indicate error in the current implicit surface. Because the radial basis functions we use have infinite support, a refinement could create errors elsewhere. We do not want to overlook any surface components, but we do not want to evaluate over the entire volume. Our solution is to sample the volume finely in a thin shell around the goal boundary voxels and to sample coarsely elsewhere, then sampling more finely if we detect a boundary. First, we sample the

volume at coordinates that are all multiples of four. If any $4 \times 4 \times 4$ cube does not have its eight vertices entirely in the interior or exterior of the surface, we sample that cube voxel by voxel; otherwise it is filled uniformly. Likewise, if a cube is within eight voxels ∂f_{goal}, we sample it finely. Although this coarse sampling could possibly miss very thin components, we have not found this to happen for any of the models that we have tried.

5.3 Refinement

Now that the boundary voxels can be evaluated by the metric, we can add constraints to refine the implicit surface. We want to avoid adding constraints one by one because performing an iteration per constraint would be costly. However, we also want to avoid refinements interfering with each other. Likewise, making fine changes could be useless if coarse changes are made elsewhere on the surface because such changes may have a nonlocal effect.

We scan through ∂f_{curr} and ∂f_{goal} to find the voxel with the maximum error. In a tie, we favor ∂f_{goal}. The error for a voxel x in ∂f_{curr} is $|sd_{goal}(x)|$, and the error for a voxel y in ∂f_{goal} is $|sd_{curr}(y)|$. We will use the notation $|sd(x)|$ to represent both these cases. Searching for the maximum error is equivalent to walking along the overlaid boundaries in Fig. 3a and Fig. 3b, and finding the largest magnitude (lightest background color).

We pick our new constraints from the boundary voxels ∂f_{curr} and ∂f_{goal}. Constraints added from ∂f_{goal} are boundary constraints. To prevent artifacts from the voxelization appearing, these constraints are actually placed at subvoxel precision by trilinearly interpolating the densities of the voxels. Another way to place the constraints more accurately would be to return to the actual polygonal data, but we have not done so. Constraints from ∂f_{curr} are interior or exterior constraints, and take on the values given by the signed distance function of f_{goal}.

Not all candidate constraint locations will improve the surface. To avoid having (5) become ill-conditioned, we discard candidates less than one voxel from any preexisting constraint. We eliminate any candidate that is within $2 \times sd(x)$ of a voxel x where a constraint was added on the current iteration. This distance restriction, along with the greedy approach of adding the constraints with greatest errors first, guarantees that for all i, the spheres of radius $sd(x_i)$ centered at x_i will be disjoint. Often, adding a constraint in one location will greatly improve the surface nearby. Boundary voxels with errors less than half the maximum error at the current iteration are considered too fine an adjustment and are discarded.

5.4 Termination

Finally we discuss how the algorithm terminates. Empirically, we found that the models tend to refine quickly at first and then slow as they converge to the goal (see Fig. 1). We terminate the algorithm under four conditions: if the model has reached a maximum error of one voxel, if a model has not improved in the previous four iterations, if too many iterations have passed (we use 30), or if too many constraints have been placed (we use a maximum of 5,000). When successive models have the same Hausdorff error, we pick the best model based from a similarly derived root-mean-square (RMS) error. For most of our results, termination was from the model not improving over the last few iterations.

6 Program Parameters

Our algorithm has several parameters that govern its behavior. We will discuss each of these parameters and show that the quality of the results are largely insensitive to their values. We note that the parameter settings can have some effect on the running times to generate the results, especially when using extreme numbers of initial constraints. All of the timing data in the tables were from using one 195 MHz R10000 MIPS processor of an SGI Origin.

6.1 Volume Resolution

Because our algorithm attempts to fit an implicit surface to a signed distance function over voxels, the performance is dependent on the voxel resolution. A coarse volume might not capture much detail from a polygonal model, and a fine volume can be computationally expensive. We use one byte per voxel, and by far, the largest factor for memory usage in our algorithm is the size of the voxel grids. For the models shown in Section 7, we use high-resolution volumes, making sure that the volumetric models lose little detail from the polygonal originals. However, our method can also make implicit surfaces out of coarse volumetric data. In this case, much detail cannot be captured in the implicit surfaces because of the absence of detail in the volumetric models, but they can be computed in mere minutes. We also tried running a few models at higher voxel resolutions. There were no noticeable improvements, which is not surprising because the results in Section 7 had errors larger than a voxel. To determine the resolution, a user could view the model with the radius of regularity or the radius of curvature color-coded and textured.

Table 1 shows the results from varying the voxel resolution. For the coarsest models, our algorithm rapidly generates a nearly exact fit because the high-frequency details are smoothed in the volumetric models. Fig. 4 shows the volumetric models and resulting implicit surfaces for models of a horse and the head at two resolutions.

6.2 Number of Initial Constraints

The final implicit surface is dependent on the number of constraints used to construct the initial surface before the refinement passes. (For all the examples shown in Section 7, 50 boundary constraints and a total of 50 interior and exterior constraints were used for initialization.) Too few initial constraints could be inefficient because the algorithm has to spend time up front just trying to get a rough fit of the goal, such as trying to make the surface bounded. While using many initial constraints can cut down the number of iterations needed, using too many initial constraints results in too much time being spent on solving for unnecessary constraints. For example, in Table 2, both models took the longest to calculate when given the most initial constraints.

We experimented with seeding the implicit surface with values ranging from 50 to 1,600 boundary constraints (100 to 3,200 total constraints). For all configurations, the algorithm returned satisfactory results.

TABLE 1
Volume Resolution

Model	Size of Volume	Iterations to Finish	Max Error (in voxels)	RMS Error (in voxels)	Total Constraints	Total Time (h:m)
Bunny	$60 \times 70 \times 69$	12	1	0.1055	742	7
Bunny	$99 \times 120 \times 119$	13	1	0.1399	3166	5:00
Bunny	$138 \times 170 \times 168$	16	2	0.2798	1501	2:27
Horse	$104 \times 70 \times 119$	11	1	0.1416	1358	1:19
Horse	$195 \times 120 \times 228$	18	2	0.1773	3952	4:50
Horse	$286 \times 170 \times 337$	17	2	0.3003	2641	5:21
Head	$69 \times 69 \times 76$	12	1	0.1591	990	15
Head	$118 \times 120 \times 135$	13	2	0.1815	3742	3:45
Head	$168 \times 170 \times 193$	18	2	0.2949	2543	5:57

Furthermore, simply adding more initial constraints did not alone produce a good surface. The results in Table 2 indicate that, although the first iterations of the surfaces get better with more constraints, they are still nowhere near the desired fit. Fig. 5 illustrates using varying amounts of initial constraints on the triceratops model. The implicit surfaces created from the initial constraints alone get consistently better with more constraints, but even with 3,200, key features such as the horns are still lacking, yet the final implicit surfaces captured these key features. Not only do these results indicate the insignificance of the amount of initial constraints, but they also demonstrate the need for iterative refinement.

For all of the results shown in Section 7, we initialize with 50 zero constraints and 50 interior and exterior constraints. Using fewer initial constraints produces results faster, and the results are comparable to those started with more constraints.

6.3 Regularization Parameter

If we wish to approximate rather than exactly interpolate some of the constraints, we can use a slight adaptation of (5) for creating the implicit surfaces. We modify the matrix's diagonal entries of the form ϕ_{ii} to $\phi_{ii} + \lambda_i$. Since $\phi_{ii} = 0$, a constraint with a nonzero λ is not interpolated exactly. Instead, the constraint's position becomes a weighted average of the desire for interpolation and the desire for regularization (smoothness).

We examined the results of our algorithm using nonzero λ values at the boundary constraints. Instead of running our algorithm again with the new λ, we simply took the constraints already produced and solved (5) with the new λ. Although our algorithm refined the set of constraints given the old $\lambda = 0$, the implicit surface with the new λ is still a nice fit. Fig. 6 shows enlarged images of the implicit horse with $\lambda = 0$ and $\lambda = 100$. Using $\lambda = 100$ improved the ears but sacrificed detail around the nostrils. The smoother version of the leg with $\lambda = 100$ is more pleasing than the

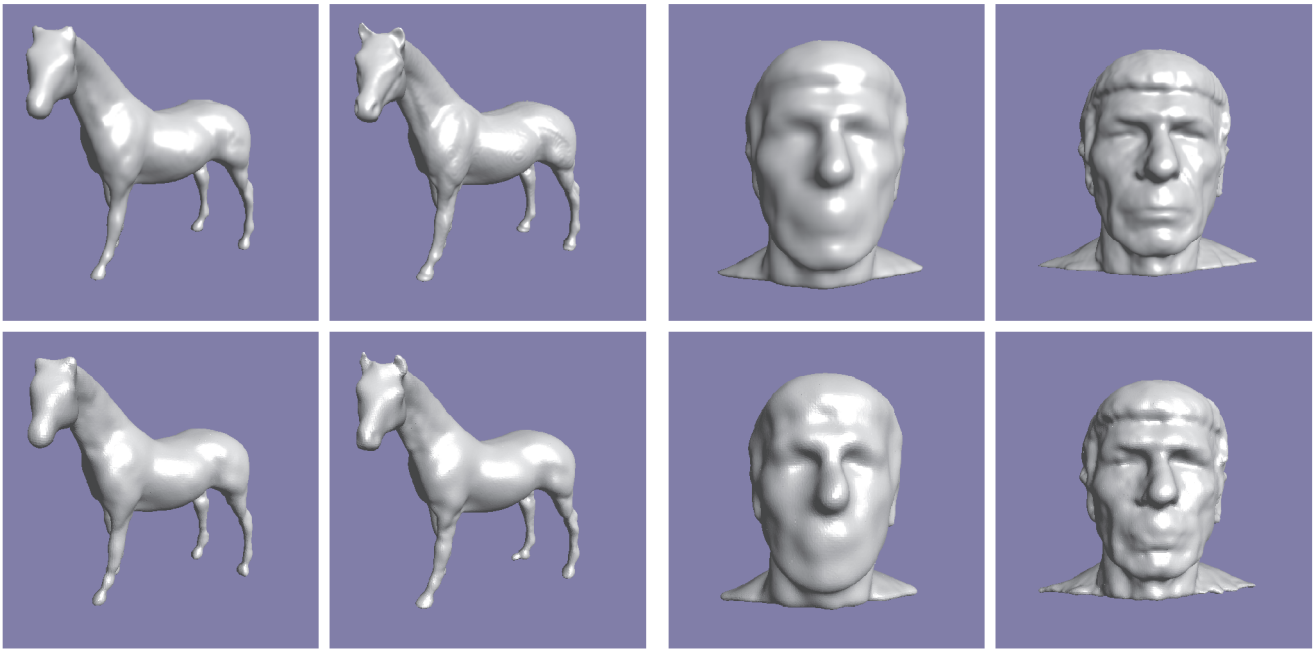


Fig. 4. Top row: low-resolution volumetric representations of the horse ($104 \times 70 \times 119$, $195 \times 120 \times 228$) and the head ($69 \times 69 \times 76$, $118 \times 120 \times 135$). Bottom row: respective implicit surfaces generated by our algorithm.

TABLE 2
Number of Initial Constraints

Model	Initial Constraints	Max Error	RMS Error	Final Constraints	Max Error	RMS Error	Iterations to finish	Total time (h:m)
Bunny	100	53	5.895	2556	2	0.0638	14	7:02
Bunny	200	36	5.335	2515	2	0.2715	16	6:15
Bunny	400	18	2.869	2145	2	0.2655	11	6:08
Bunny	800	31	0.961	2774	2	0.2701	7	5:42
Bunny	1600	5	0.389	2671	2	0.2768	5	7:35
Bunny	3200	6	0.187	4057	2	0.0715	4	11:13
Triceratops	100	25	5.039	1803	2	0.2732	17	3:24
Triceratops	200	27	3.061	1953	2	0.0655	15	4:16
Triceratops	400	17	2.073	1910	2	0.0722	11	3:24
Triceratops	800	24	1.311	1906	3	0.2665	12	6:35
Triceratops	1600	16	0.709	2230	3	0.2658	10	7:13
Triceratops	3200	13	0.4712	3630	2	0.2549	7	12:25

$\lambda = 0$ version, especially the hoof and the dimple on the inner thigh. All of the results in Section 7 use $\lambda = 0$; a user who wishes a smoother model can set λ accordingly.

7 Results

The main goal of our research is to create an entirely automatic algorithm for creating implicit models from polygons, a robust method not requiring human intervention. We tested our algorithm on a variety of polygonal models and the results convinced us that it will behave robustly for all but the most pathological of polygonal models.

Fig. 7 and Fig. 8 show the resulting implicit surfaces from our algorithm. The subimages in these figures are arranged in pairs, with the intermediate volumetric models (the left subimages in a pair), which do not capture all the details from the polygonal models, shown side-by-side with the resulting implicit surfaces created by our method (right subimage of pair). For example, some surfaces that are paper-thin are problematic for our algorithm because the voxelization will alias those regions or not even capture them. While some forms of implicit surfaces can represent such thin features, we do not know of any automatic method for creating thin-featured models from polygons.

Fig. 7 shows our results on six high-resolution models obtained from laser range scanning. In all cases, our

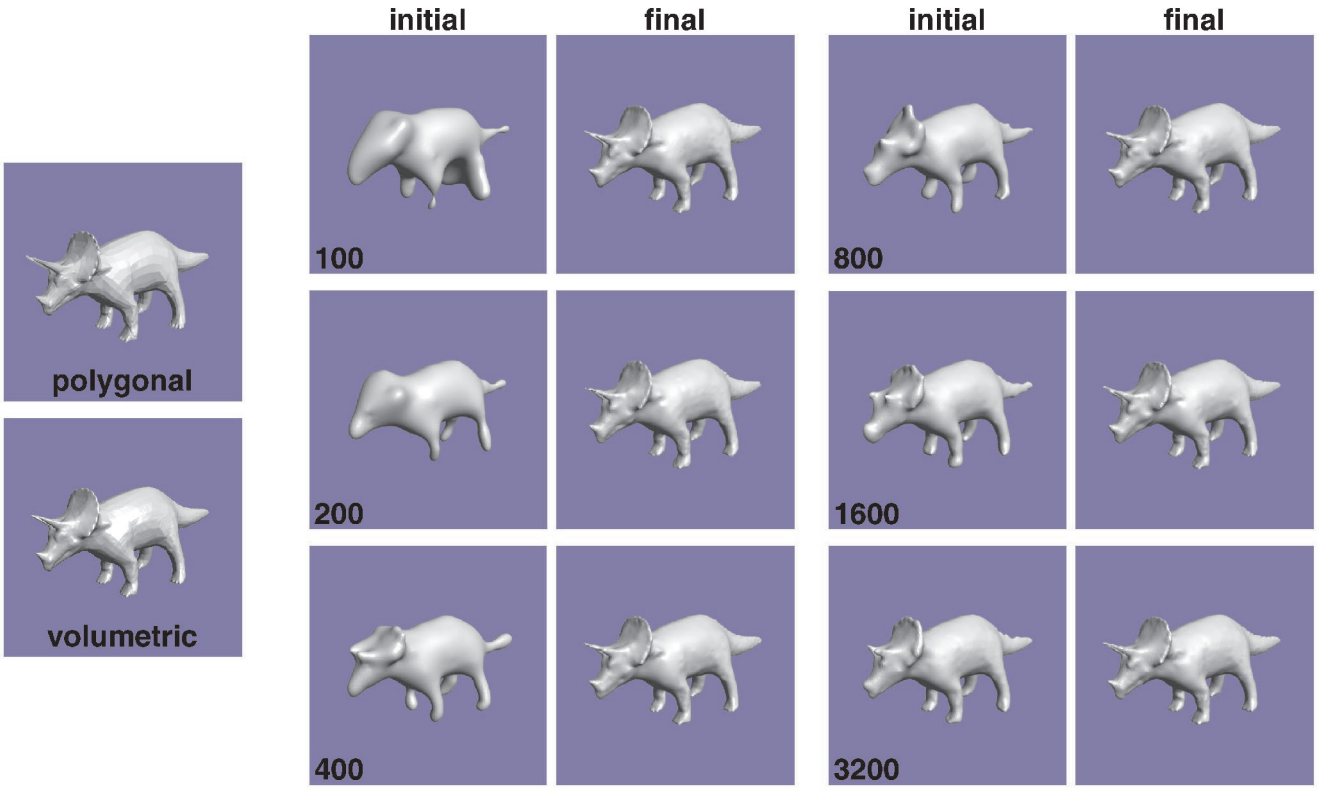


Fig. 5. Triceratops model, showing the effect of the number of initial constraints. At far left is original polygon model and the volumetric mode. Pairs of models show results after just the initial constraints (left image of pairs, with number of constraints in corner), and after subsequent refinement until the algorithm terminates (right image of pairs).

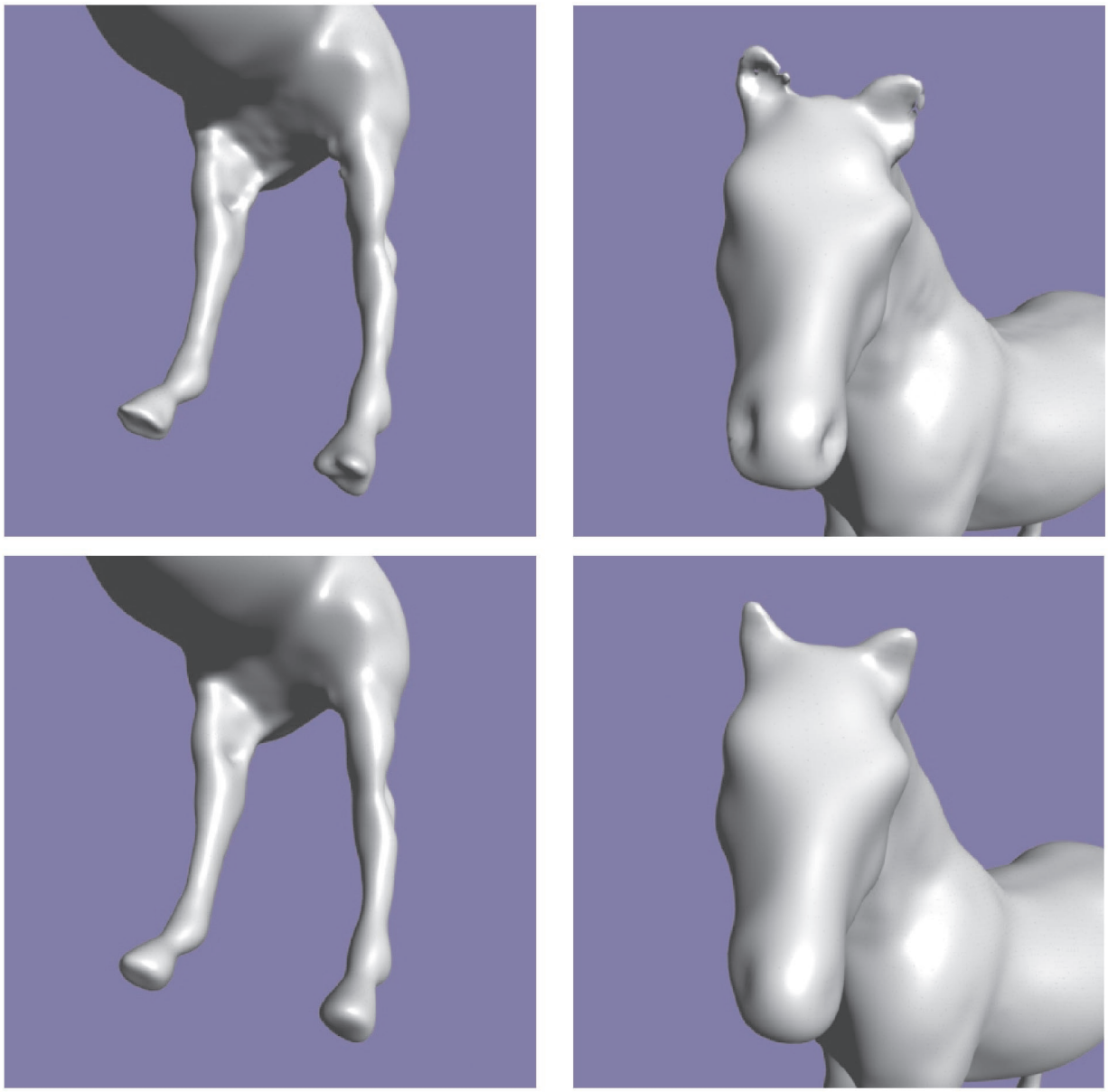


Fig. 6. Implicit horse: head and leg. Top shows $\lambda = 0$ (exact interpolation) and bottom shows $\lambda = 100$ (weighted average of exact interpolation and curvature minimization).

algorithm produced implicit surfaces that capture all of the main features of each model. The bunny and teeth implicit surfaces look nearly identical to their volumetric counterparts. The two head models are also close matches, although small but sharp creases such as wrinkles or eyelids are not always captured. The algorithm performs the least well on the Buddha model, apparent from both the image and Table 3. The Buddha model is difficult because of the fine detail, especially high curvature areas such as the folds on the robe. The implicit surface for the dragon model captures all of the macroscopic features but does not capture the fine scales. Perhaps more constraints could capture these very fine details, but at too great a cost. A more reasonable approach would be to encode the scales as a bump map or displacement map (see Future Work). We do not know of any other method for creating implicit surfaces using radial basis functions that would give results comparable to those shown in this figure.

Fig. 8 shows our results on six polygonal models that contain high-curvature regions or thin surfaces. The triceratops has sharp horns, and the horse has skinny legs and pointy ears. All of these features are captured by the surface created by our method. The foot model contains many slender bones, but the most difficult feature is not the bones, but the thin gaps between the bones. The flamingo not only has wiry legs but also the thin foot webbing and wings. The wings are particularly hard to fit because they are separated from the rest of the body by a very small gap. The model with the ants crawling around the trefoil knot is also a very difficult model because of the high curvature around the many holes. The implicit representation we use has the effect of rounding off the sharp edges of this model. For the other five models, despite their high-curvature features, our results fit the original data well.

For more quantitative comparisons, Table 3 gives the numbers of iterations, the numbers of constraints in the final implicit surfaces, and the errors of the final implicit

Fig. 7. Six large scanned polygonal models: intermediate volumetric representations (left images of pairs) and final implicit surfaces (right images of pairs) from our algorithm.

surfaces. Each final implicit surface was specified using roughly two to three thousand constraints and achieved a root-mean-squared error of less than one voxel. The Igea artifact model (upper left in Fig. 7) took the fewest iterations; most likely the algorithm did not have to refine much because the model did not have many high-curvature regions. The scorpion and flamingo took the most iterations; we conjecture that the variational implicit surfaces had a difficult time fitting these goal models due to their high curvature.

Table 4 shows the running times for the algorithm on all of the models. The running times ranged from a few hours to just over a day for the Buddha model. Most of the compute time for the algorithm was in the implicit function evaluation stage. Each evaluation of the implicit surface at a point requires time linear in the number of constraints. Our evaluation method tests a shell of voxels finely and a volume of voxels coarsely. However, the shell is rather thick, and in the later iterations, evaluating at a point becomes costly from so many constraints. For many of our models, the last few iterations amounted to half the total running time or more. Because we wanted to err conservatively regarding evaluating the implicit function as exactly as possible, there may be room for speeding up the evaluation, such as reducing the thickness of the shell. Solving (5) is also expensive for a large number of constraints. LU decomposition is cubic in the number of constraints, but because the evaluation was still more expensive, we did not focus on improving the matrix solution. Jacobi iteration with the last set of weights as the initial value could accelerate the matrix solution. The table also shows timing information to achieve root-mean-squared errors less than two voxels and less than one voxel; for most of the models, the algorithm reached these accuracies in substantially less time.

8 Conclusions and Future Work

To the best of our knowledge, our method is the first approach that automatically converts an arbitrary polygonal mesh to a smooth radial-basis-function implicit surface. We have tested our method on a variety of complex polygonal meshes, and we are convinced empirically that it behaves robustly. Specifically, we feel that our method makes the following new contributions to computer graphics:

- Automatically converts any manifold polygonal mesh to a smooth implicit surface.
- Generates implicit surfaces from low-resolution data very fast.

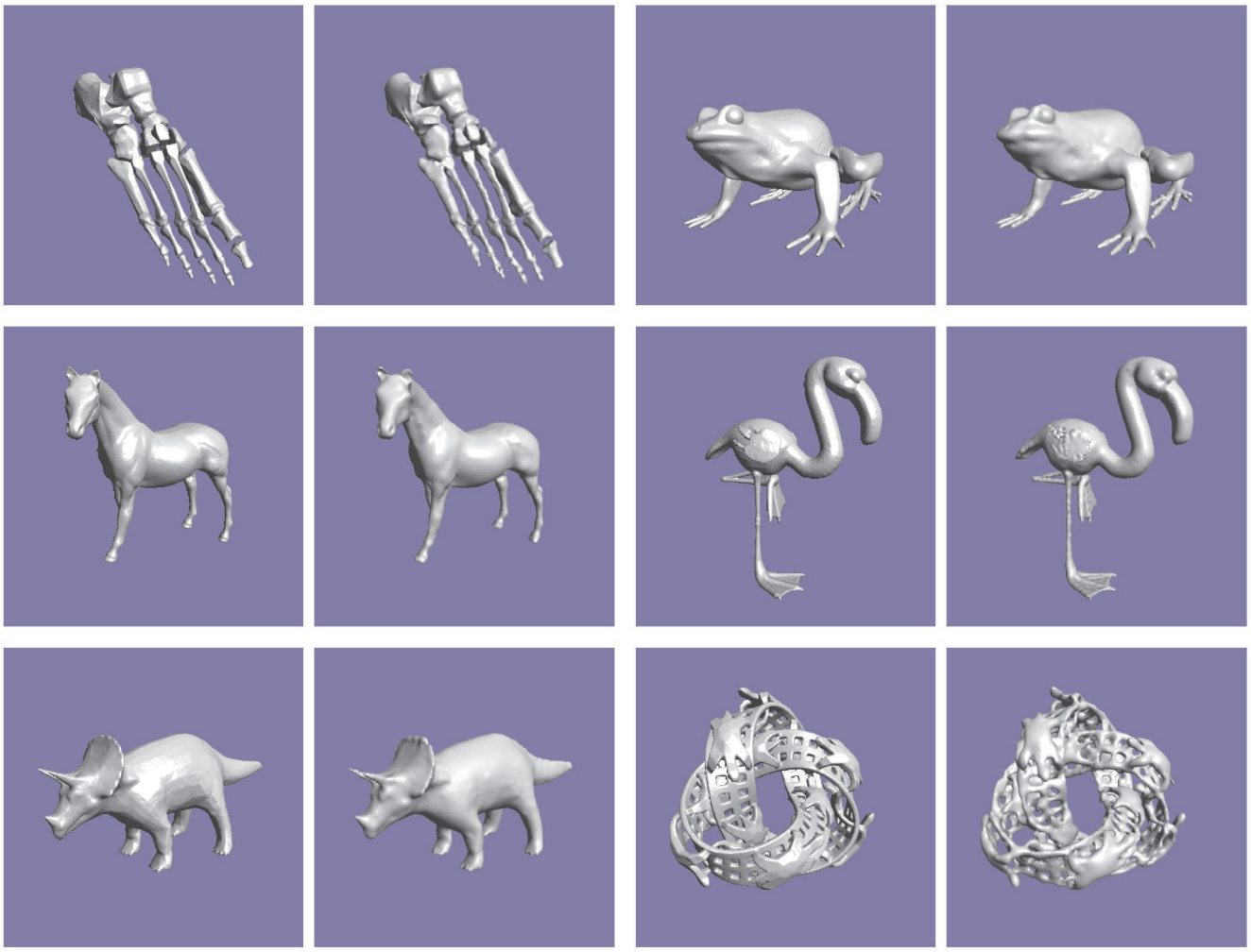


Fig. 8. Six models with thin or high-curvature regions: intermediate volumetric representations (left images of pairs) and final implicit surfaces (right images of pairs) from our algorithm.

- Provides a general framework for other basis functions or other implicit surface representations.

One avenue for future work is looking at classes of basis functions that either have finite support or approach zero asymptotically. We will still be able to specify constraints to be interpolated exactly, but the basis functions will not minimize the aggregate curvature discussed in Section 3. However, using such basis functions should speed up the

TABLE 3
Implicitization of Polygonal Models

Model	Iterations to Finish	Zero Constraints	Interior Constraints	Exterior Constraints	Max. Error (in voxels)	RMS Error (in voxels)
Bunny	16	1718	63	734	2	0.2715
Dragon	22	1627	163	791	2	0.3329
Flamingo	26	1858	143	472	2	0.2726
Foot Bones	22	2307	138	681	2	0.3018
Frog	24	1900	137	791	2	0.2840
Happy Buddha	24	1788	196	648	4	0.6732
Horse	17	1829	88	724	2	0.3003
Igea Artifact	8	1591	54	842	2	0.3149
Scorpion	27	1992	179	483	3	0.7784
Head	18	1444	81	1018	2	0.2949
Teeth Cast	17	2489	81	915	2	0.2655
Trefoil	23	2463	501	89	5	0.3844
Triceratops	17	1333	84	386	2	0.2732

TABLE 4
Running Time of Implicitization (In Hours: Minutes on a 195 MHz R10k)

Model	Size of Volume	Time until RMS Error < 2	Time until RMS Error < 1	Total Time
Bunny	$176 \times 220 \times 218$	1:15	1:45	7:02
Dragon	$113 \times 220 \times 163$	49	1:50	5:41
Flamingo	$237 \times 120 \times 288$	39	1:21	7:40
Foot Bones	$138 \times 320 \times 124$	47	1:30	6:46
Frog	$247 \times 220 \times 151$	1:44	3:46	16:07
Happy Buddha	$170 \times 170 \times 373$	6:15	15:42	24:47
Horse	$286 \times 170 \times 337$	1:58	3:01	5:21
Igea Artifact	$220 \times 220 \times 161$	14	52	2:16
Scorpion	$335 \times 220 \times 180$	1:40	2:57	3:56
Head	$168 \times 170 \times 193$	27	1:18	5:57
Teeth Cast	$223 \times 170 \times 269$	1:09	2:44	11:35
Trefoil	$119 \times 220 \times 208$	2:11	7:30	9:23
Triceratops	$124 \times 320 \times 155$	41	1:03	3:24

algorithm because the matrix will be sparse and implicit function evaluations will only have to use nearby basis functions. Recent work by Morse et al. has demonstrated the promise of using compactly supported basis functions [19].

More work still needs to be done on representing high-frequency features such as thin surfaces and fine detail. Thin surfaces might be better represented by other radial basis functions or by basis functions that could be weighted along principal directions (much like the covariance matrix for a Gaussian). We do not feel that adding more constraints is the right way to capture fine detail. Rather, fine detail could be added by local-influence implicit surfaces. Another logical path to explore is adding fine features (such as scales on the dragon) using normal or displacement maps, such as the work in [27], [28], [29].

A different representation of the volumes could be used to reduce memory usage and computation times. Recently, researchers at MERL used adaptively sampled distance fields to represent surfaces [30] efficiently. Using wavelets to represent the volume data might have similar benefits [22]. Such an adaptive approach might be useful to speed up several of the algorithm tasks, especially in evaluating the metrics and searching for locations where the surface needs to be refined.

Acknowledgments

The authors would like to thank Hughes Hoppe for providing the head dataset. They would also like to thank James O'Brien and the Geometry Group at Georgia Tech for helpful advice and discussions. This research was funded in part by ONR grant N00014-97-1-0223. The first author was funded in part by an US National Science Foundation Graduate Fellowship.

References

[1] B. Payne and A. Toga, "Distance Field Manipulation of Surface Models," *IEEE Computer Graphics and Applications,* vol. 12, pp. 65-71, 1992.

[2] J. Hughes, "Scheduled Fourier Volume Morphing," *Proc. SIGGRAPH '92,* pp. 43-46, 1992.

[3] G. Turk and J.F. O'Brien, "Shape Transformation Using Variational Implicit Functions," *Proc. SIGGRAPH '99,* pp. 335-342, 1999.

[4] M. Desbrun and M.-P. Gascuel, "Animating Soft Substances with Implicit Surfaces," *Proc. SIGGRAPH '95,* pp. 287-290, 1995.

[5] M.-P. Gascuel, "An Implicit Formulation for Precise Contact Modeling Between Flexible Solids," *Proc. SIGGRAPH '93,* pp. 313-320, 1993.

[6] J.F. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. Graphics,* vol. 1, no. 3, pp. 235-256, 1982.

[7] A.P. Witkin and P.S. Heckbert, "Using Particles to Sample and Control Implicit Surfaces," *Proc. SIGGRAPH '94,* pp. 269-278, 1994.

[8] J. Levin, "A Parametric Algorithm for Drawing Pictures of Solid Objects Composed of Quadric Surfaces," *Comm. ACM,* vol. 19, pp. 555-563, 1976.

[9] G. Wyvill, C. McPheeters, and B. Wyvill, "Data Structures for Soft Objects," *The Visual Computer,* vol. 2, no. 4, pp. 227-234, 1986.

[10] J. Bloomenthal and K. Shoemake, "Convolution Surfaces," *Proc. SIGGRAPH '91* pp. 251-256, 1991.

[11] A. Sherstyuk, "Interactive Shape Design with Convolution Surfaces," *Shape Modeling International,* 1999.

[12] E. Ferley, M.-P. Cani, and J.-D. Gascuel, "Practical Volumetric Sculpting," *Implicit Surfaces,* 1999.

[13] S. Muraki, "Volumetric Shape Description of Range Data Using 'Blobby Model'," *Proc. SIGGRAPH '91,* pp. 227-235, 1991.

[14] E. Bittar, N. Tsingos, and M.-P. Gascuel, "Automatic Reconstruction of Unstructured 3D Data: Combining a Medial Axis and Implicit Surfaces," *Computer Graphics Forum (Proc. Eurographics '95),* vol. 14, pp. 457-468, 1995.

[15] N. Tsingos, E. Bittar, and M.-P. Gascuel, "Semi-Automatic Reconstruction of Implicit Surfaces for Medical Applications," *Computer Graphics Int'l,* vol. 14, 1995.

[16] *Introduction to Implicit Surfaces,* J. Bloomenthal, ed., Morgan Kaufmann, 1997.

[17] G. Turk and J.F. O'Brien, "Variational Implicit Surfaces," Technical Report 15, Georgia Inst. of Technology, 1999.

[18] J. Duchon, "Splines Minimizing Rotation-Invariant Semi-Norms in Sobolev Spaces," *Lecture Notes in Math. 571,* pp. 85-100, 1976.

[19] B. Morse, T. Yoo, P. Rheingans, D. Chen, and K.R. Subramanian, "Interpolating Implicit Surfaces from Scattered Surface Data Using Compactly Supported Radial Basis Functions," *Proc. Int'l Conf. Shape Modeling and Applications,* pp. 89-98, May 2001.

[20] F.S. Nooruddin and G. Turk, "Simplification and Repair of Polygonal Models Using Volumetric Techniques," Technical Report GIT-GVU-99-37, Graphics, Visualization and Usability Center, Georgia Inst. of Technology, 1999.

[21] A.T Kaufman, "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *Computer Graphics (Proc. SIGGRAPH '87),* vol. 21, no. 4, pp. 171-179, July 1987.

[22] L. Velho and J. Gomez, "Approximate Conversion of Parametric to Implicit Surfaces," *Computer Graphics Forum,* vol. 15, no. 5, pp. 327-337, Dec. 1996.

[23] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion, "An Accurate Method for Voxelizing Polygon Meshes," *Proc. 1998 IEEE Symp. Volume Visualization,* pp. 119-126, Oct. 1998.
[24] P.-E. Danielsson, "Euclidean Distance Mapping," *Computer Graphics and Image Processing,* vol. 14, pp. 227-248, 1980.
[25] W.J. Schroeder and W.E. Lorensen, "Implicit Modeling of Swept Surfaces and Volumes," *Proc. Visualization '94,* pp. 40-45, Oct. 1994.
[26] D.P. Mitchell, "Generating Antialiased Images at Low Sampling Densities," *Computer Graphics (Proc. SIGGRAPH '87),* vol. 21, no. 4, pp. 65-72, July 1987.
[27] V. Krishnamurthy and M. Levoy, "Fitting Smooth Surfaces to Dense Polygon Meshes," *Proc. SIGGRAPH '96 Conf. Proc.,* H. Rushmeier, ed., pp. 313-324, Aug. 1996.
[28] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini, "Preserving Attribute Values on Simplified Meshes by Re-sampling Detail Textures," *The Visual Computer,* vol. 15, no. 10, pp. 519-539, 1999.
[29] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped Textures," *Computer Graphics Proc., Ann. Conf. Series (Siggraph '00),* pp. 465-470, 2000.
[30] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics," *Proc. SIGGRAPH '00,* pp. 249-254, 2000.

Gary Yngve received the BS (2000) from Georgia Institute of Technology, where he worked with Greg Turk on implicit surfaces. He is pursuing a PhD in computer science at the University of Washington on a US National Science Foundation Graduate Fellowship. His current work focuses on character animation, and his other research interests include modeling and simulation, natural phenomena, photo-realistic rendering, and mathematical techniques for computer graphics.

Greg Turk received the PhD in computer science in 1992 from the University of North Carolina at Chapel Hill. He was a postdoctoral researcher at Stanford University for two years, followed by two years as a research scientist at UNC Chapel Hill. He is currently an associate professor at the Georgia Institute of Technology, where he is a member of the College of Computing and the Graphics, Visualization and Usability Center. His research interests include computer graphics, computer vision, and scientific visualization. He is a member of the IEEE.

Medical Applications of Implicit Surfaces

Terry S. Yoo
Office of High Performance Computing and Communications
National Library of Medicine, NIH

Acknowledgements

Volume Modeling Consortium
- Bryan Morse (BYU)
- K.R. Subramanian (UNCC)
- Penny Rheingans (UMBC)
- Kathleen Hoffman (UMBC)
- David T. Chen (NLM/NIH)
- Terry S. Yoo (NLM/NIH)

Also
- Greg Turk (GA Tech)
- James F. O'Brien (UCB)

Philosophical Problem

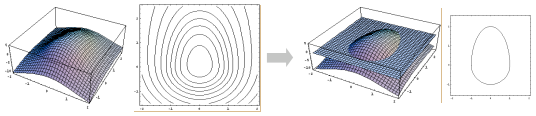
- Modeling and representing anatomical or other medical objects requires a respect for detail and a commitment to the truth.
- Discrete representations lead to sampling error, aliasing.
- Under magnification, approximations become visible.
- What is an ideal modeling/interpolating primitive?

Invariant with respect to rotation, translation, and zoom

General Problem

- Medical imaging produces discretely sampled images or volumes.
- May want/need to have continuous surfaces fit to those discrete points for analysis.
- Could a class of interpolants known as interpolating (or constrained) implicit surfaces be used?
- Lots and lots of points! Too slow!
- Or is it?

Implicit Surfaces



- Build a embedding or characteristic function.
- One connected isosurface is the implicit surface.
- Related to research in level sets.
- Not an parametric or polygonal surface representation.

Related Work

- Hoppe 1992
- Witkin and Heckburt 1994
- Savchenko, *et al.* 1995
- Turk and O'Brien 1999
- Yngve and Turk 1999
- Morse, *et al.* 2001
- Carr, *et al.* 2001

Implicit Surfaces that Interpolate

Savchenko, *et al.*, 1995.

- Interpolate surfaces from scattered points or contours.
- Build their model from Green's function.

Turk and O'Brien, 1999

- "Variational Implicit Surfaces"
- Build their model from thin plate splines.
- Demonstrated their approach for morphing or shape interpolation.

Variational Implicit Surfaces Turk and O'Brien, 1999

Given: a set of oriented points (constraints)

- surface scanners, segmented medical data, etc.

Build a linear combination of radial basis functions (RBF) to create an embedding function.

Choice of RBF: thin plate splines

- Interpolates across relatively large distances.
- Energy min. thin plate deformation. (variational)

Thin-Plate Splines

- Used by Turk and O'Brien (1998, 1999), Carr 2001
- Interpolating functions that minimize smoothness metric ("bending energy")

$$E(f) = \int_{s\in\Omega} f_{xx}^2(s) + 2f_{xy}^2(s) + f_{yy}^2(s)\,ds$$

- Solved using radial basis functions (Duchon, 1978)

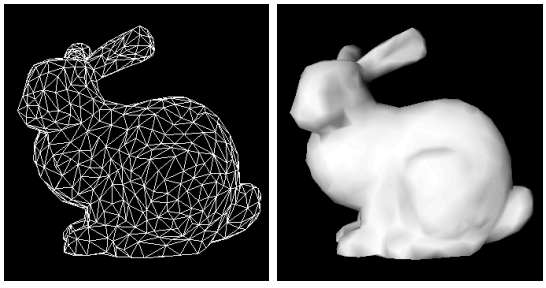
Variational Implicit Surfaces (continued)

$$f(\bar{c}_i) = \sum_{j=1}^{n} d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

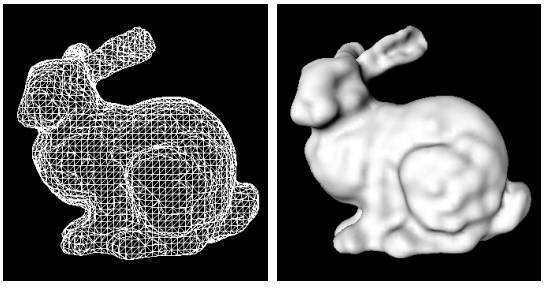
$$\phi(r) = r^3 \qquad \phi(r) = r^2 \log r$$

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

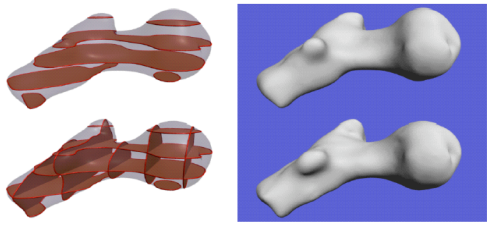
Example: Coarse Model



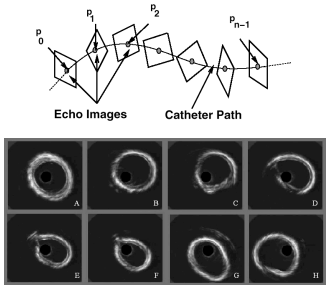
Example: Interpolated Model



Medical Implicit Models
(from Turk and O'Brien 1999)



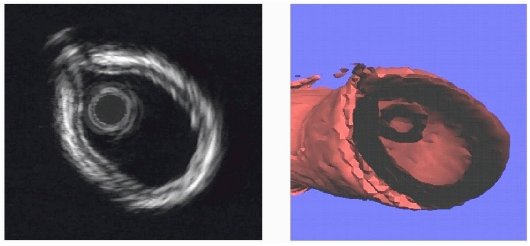
3D Reconstructions from Intravenous Ultrasound



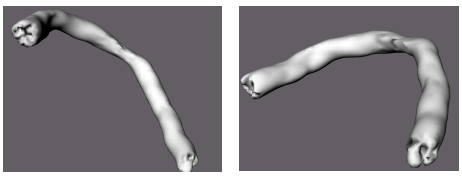
3D Ultrasound Reconstruction

- Implicit surfaces using thin plate spline radial basis functions.
- Reconstructs 3D shapes from arbitrarily oriented segmented 2D contours.
- Thin plate splines are useful for interpolating over large, unpredictable distances.
- Requires segmentation (easy for IVUS)

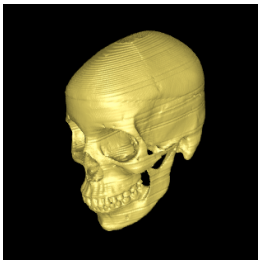
Polygonal Reconstruction



Implicit Reconstruction



How Many Points?



Subject: Dry Skull
Slices: 124, 512x512

Vertices:
584,631
Triangles:
1,169,608

Issues

- **Dense data increases computational load.**
- **Thin plate spline interpolant limited to moderate numbers of (20,000) constraints.**
- **This approach not appropriate for complex models or large data.**
  - Dense modalities (CT, MRI, etc.)
  - Complex surfaces: BRAIN!
- **Carr, *et al.* 2001, limit the number of constraints**

Linear System for RBFs

Linear system of equations:

$$f(\bar{c}_i) = \sum_{j=1}^{n} d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

Unknown Unknown

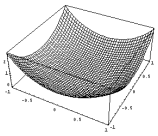
Solve Ax = b

- A is symmetric, positive definite
- Guaranteed to have a solution
- A is $(n + d+1)^2$ for n points in d dimensions
- May not be fast to solve for large n!

Thin-plate Spline Implementation (revisited)

$$f(\bar{c}_i) = \sum_{j=1}^{n} d_j \phi(\|\bar{c}_i - \bar{c}_j\|) + P(\bar{x}) = h_i$$

$$\phi(r) = r^3$$



$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Overview of Algorithm

Build matrix of $\prod(c_i\text{-}c_j)$

- $O(n^2)$ calculation
- $O(n^2)$ storage

Solve matrix

- $O(n^3)$ calculation
- $O(n^2)$ calculation possible

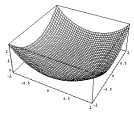
Extract implicit surface

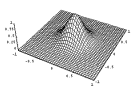
- O(n) per sample

Skull Example

- 584,632 vertices
- 584,632 normals
- 3 Dimensions
- Matrix is: 1,169,268 x 1,169,268 x IEEE floating point word = 5.468 Terabytes

Approach: Change of RBF





Thin-plate spline interpolant has infinite support

- not band limited

Creates dense matrices, difficult to solve.
Shift to compactly supported RBF with high continuity.

Other Radial Basis Functions

Other RBFs minimize other functionals.

Gaussian:

- Radially symmetric
- Smooth result
- Decays to zero rather than increasing
- Effect of farther-away points is *less* rather than *more*
- Matrix solution is better conditioned

Compact Local RBFs

Can go one step further and use *compact, locally-supported radial basis functions*.

Can't just use arbitrary functions

- Smooth
- Differentiability of resulting interpolation
- *Must produce positive definite matrix*

Compact Local RBFs

- Wendland (1995) has solved for minimum-degree compact functions that guarantee that the solution matrix is positive definite.
- All of the form (can be scaled if needed)

$$\phi(r) = \begin{cases} (1-r)^p P(r) & \text{if } |r| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

Compact Local RBFs

Major implication: radius of support ϕ imposes strict locality of solution.

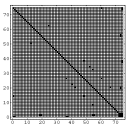
Advantages during all phases of using interpolating implicit surfaces

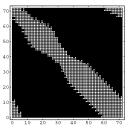
- Building matrix
- Solving matrix
- Evaluating embedding function

Building the Matrix

- Don't need to compute value of RBF between all pairs of points, only those within ϕ of each other.
- Can use spatial data structure such as k-d tree to find "nearby" points in log n time.
- Calculation is O(n log n) vs. $O(n^2)$
- Sparse storage: O(n) vs. $O(n^2)$

Compactly Supported RBF





- Local nature of the interpolants allows for spatial subdivision.
- Encourages the use of advanced data structures (K-D trees for spatial subdivision).
- Faster solvers for sparse matrices.

Solving the Matrix

Sparse matrix solution

- Varies with the number of non-zero elements, not the absolute size of the matrix
- Better conditioned

Calculation is ~ $O(n^{3/2})$ vs. $O(n^3)$ or $O(n^2)$

Evaluating the Function

- Don't need to compute value of RBF for all constraints, only those within □ of the point being evaluated.
- Again use spatial data structure such as k-d tree to find "nearby" points in log n time.
- Calculation is O(log n) vs. O(n) to perform a single evaluation.

Overview of Algorithm

Build matrix of $\sum(c_i-c_j)$ only if $c_i-c_j<$□

- O(n log n) calculation (using k-d tree)
- O(n k) storage
  where k = avg. number of points in support

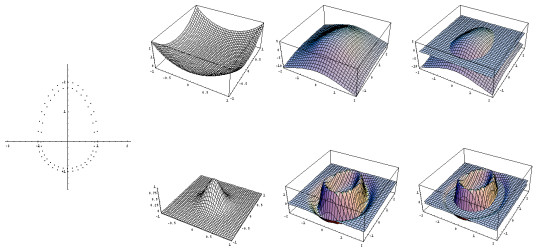
Solve matrix

- O(n k) calculation

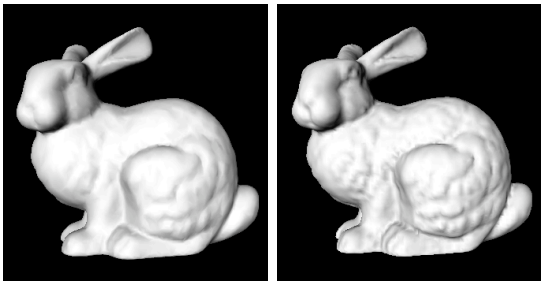
Extract implicit surface

- O(log n) per function eval. (using k-d tree)

Comparing TPS and CSRBFs



8000 Interpolated to 41,864

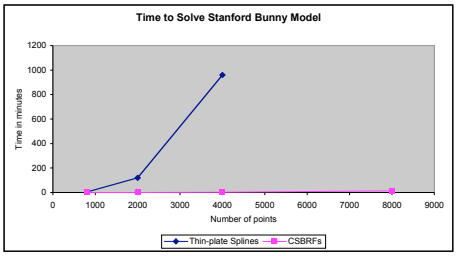


Timing: Computing the Embedding Function

"Bunny" model	TPS	CSRBF
• 800 points	3:22 min	7.7 sec
• 2000	2 hrs	30.6 sec
• 4000	16 hrs*	132.8 sec
• 8000	n/a**	864 sec*

* Significantly affected by swapping
** 2GB storage required

Timing: Computing the Embedding Function

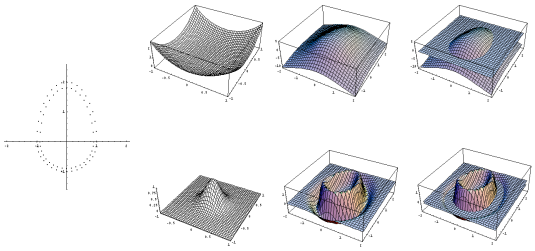


Why So Much Faster?

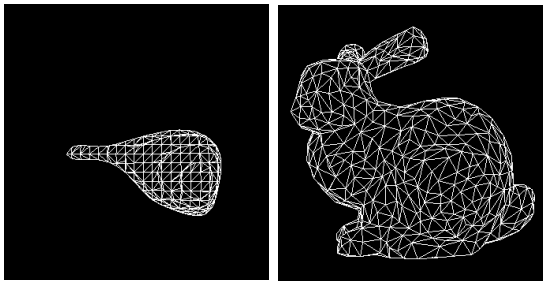
Number of points in matrix

Points	TPS	CSRBF	%
• 800	646,416	69,582	10.8%
• 2000	4,016,016	220,095	5.5%
• 4000	16,032,016	850,675	5.3%
• 8000	64,064,016	3,342,253	5.2%

Comparing TPS and CSRBFs



Example: Inner Hull



Outer hull… interfering with ray tracing?

Ray tracing is possible

- Adjust the level set to suppress the outside isosurface

OR

- Apply a transfer function to reflect only at zero crossings with high gradient magnitude

Other Implications for Future Work

Faster method means interpolating implicit surfaces are viable for medical models

- Resampling models
- More accurate model simplification

Local support means local effect

- Incremental updating
- Interactive modeling

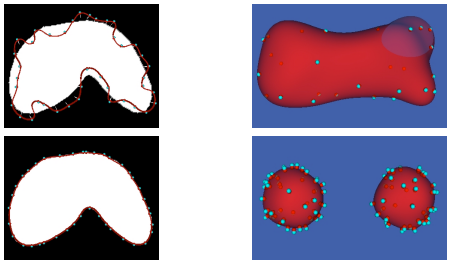
Future Work

- Error analysis vs. thin-plate splines
- More efficient data structures for managing spatial locality
- Other types of sparse solvers
  - Using LU, could use SVD, CG, etc.
- Better ways to deal with extracting desired isosurface and not the inner/outer hulls
- More timing studies/profiling

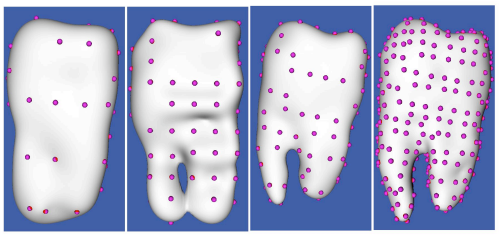
Future Work: Reinventing 2D segmentation

- Implicit surfaces can interpolate 2–D segments (unevenly sampled slices).
- Implicit Interpolating methods might provide better (smoother, more continuous) priors to initialize active contours and level set segmenters.
- Implicit Interpolators can be used to regularize stacks of hand segmented slices (adapted from Yngve &Turk).

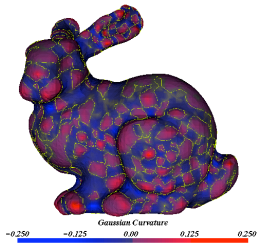
Implicit Snakes
Yoo and Subramanian 2001



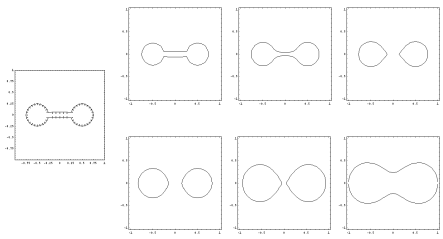
Implicit Snakes (cont.)
Yoo and Subramanian 2001



Future Work: Geometry of Complex Models



Future: Scale Space Study of Implicit Surfaces



Conclusions

- Implicit functions interpolated with radial basis functions are flexible and easily applied to anatomical modeling.
- Choice of the RBF depends on application (number of points).
- A little computer science goes a long way.
- Leads to interesting interdisciplinary work.
  - computer vision, computational geometry
  - computer graphics, visualization, medicine

Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions

Bryan S. Morse[1], Terry S. Yoo[2], Penny Rheingans[3], David T. Chen[2], K. R. Subramanian[4]

[1]Department of Computer Science, Brigham Young University

morse@byu.edu

[2]Office of High Performance Computing and Communications, National Library of Medicine

{yoo | dave}@nlm.nih.gov

[3]Department of Computer Science and Electrical Engineeering, University of Maryland Baltimore County

rheingan@cs.umbc.edu

[4]Department of Computer Science, University of North Carolina at Charlotte

krs@cs.uncc.edu

Abstract

We describe algebraic methods for creating implicit surfaces using linear combinations of radial basis interpolants to form complex models from scattered surface points. Shapes with arbitrary topology are easily represented without the usual interpolation or aliasing errors arising from discrete sampling. These methods were first applied to implicit surfaces by Savchenko, et al. and later developed independently by Turk and O'Brien as a means of performing shape interpolation. Earlier approaches were limited as a modeling mechanism because of the order of the computational complexity involved. We explore and extend these implicit interpolating methods to make them suitable for systems of large numbers of scattered surface points by using compactly supported radial basis interpolants. The use of compactly supported elements generates a sparse solution space, reducing the computational complexity and making the technique practical for large models. The local nature of compactly supported radial basis functions permits the use of computational techniques and data structures such as k-d trees for spatial subdivision, promoting fast solvers and methods to divide and conquer many of the subproblems associated with these methods. Moreover, the representation of complex models permits the exploration of diverse surface geometry. This reduction in computational complexity enables the application of these methods to the study of shape properties of large complex shapes.

1 Introduction

As a research field of growing interest, implicit surface representations have a colorful history, with their foundations established early in the development of 3D curves and surfaces in computer graphics [2]. However, the computation of implicit surfaces has often been hampered by the constraints of available processing power and the limited complexity of the models that can be created. As CPU speeds, available memory, and computing costs have evolved, more complex models and techniques have become possible, spurring general interest in implicit surfaces [3, 6]. However, managing complex models remains difficult. The core research in modeling with implicit surfaces centers around primitives that do not always facilitate the control of the exact surface. While alternatives to surface control through simple primitives exist, they also have drawbacks. In their recent work on adaptive particle sampling and control of implicit surfaces, Heckbert and Witkin [18] report that particle control of such surfaces is slippery and elusive.

Recent growth in level set techniques [9, 12] and variational methods [7] have created new interest in understanding and manipulating complex surface models directly from the surface representation. Whitaker and Breen [17] have shown how level set techniques can be used to model and manipulate computer graphic shapes, effectively morphing from one to another. They characterize the level set as an implicit representation where the primitives are distributed throughout an active volumetric cloud layer near the surface boundary. They utilize Sethian's notion of the active set, to reduce the size of the volume representation to a sparse collar of primitives. However, even with this reduction, the representations are cumbersome, and the bookkeeping to support these methods are intricate and difficult to maintain.

Other recent work has developed techniques for interpolating an implicit surface directly from surface point data [11, 14]. This work provides some insight into how to

manage and employ a collection of implicit primitives while simultaneously directly controlling the surface parameters. This method allows direct specification of a complex surface from sparse, irregular surface samples. The method is quite flexible and has been extended to higher dimensions to support shape interpolation [15]. However, because of computational and storage complexity, the technique as described cannot be used to model surfaces where large numbers of surface points are included, making it unsuitable for applications where range data or tomographic reconstruction often lead to data described by hundreds of thousands of surface points.

This paper primarily addresses the topic of computational complexity. We explore an adaptation of the methods in [11, 14], applying compactly supported radial basis functions [16] to create an efficient algorithm for computing interpolated surfaces. Our technique produces significant improvements in memory utilization and computational efficiency. We discuss both the advantages of our technique as well as the consequences or prerequisite requirements imposed by our methods in later sections.

This more efficient approach is creating opportunities to explore complex shapes through implicit modeling methods. In particular, the thin-plate-spline radial basis function and the Green's function solutions with their order O(n^2) solutions are impractical when the number of constraints exceeds a few thousand points. By shifting to a compactly supported radial basis function, we can create differentiable analytic representations of large complex models. These efficient solutions make possible techniques for studying the deep structure of solid shapes. In the discussion section of this paper, we present early applications of these implicit surfaces to problems such as surface shape analysis.

2 Background / Problem

The key idea in both [11] and [14] is that one may produce an implicit surface from known surface points by interpolating the embedding function within which the surface is implicitly defined. While we primarily follow here the presentation found in [14], we also encourage the interested reader to see an alternative and earlier formulation in [11].

2.1 Interpolating Surfaces by Interpolating Embedding Functions

An *implicit surface* is defined by $\{\overline{x} : f(\overline{x}) = 0\}$ for some embedding function $f : \mathbf{R}^n \rightarrow \mathbf{R}$. The key idea behind interpolated implicit surfaces is to find a smooth embedding function f such that $f(\overline{x}_i) = 0$ for each known surface point $\overline{x}_i$, and $f(\overline{y}_i) = 1$ for one or more points $\overline{y}_i$ known to be inside the shape. (Alternatively, constraints of the form $f(\overline{y}_i) = -1$ may be added for points outside the shape.) Turk and O'Brien select these interior points using normals at the surface points as shown in Figure 1.

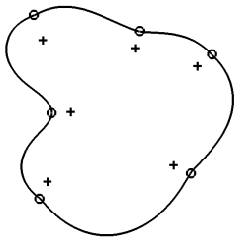


Figure 1. Implicit curve interpolated using zero-constraints along the curve and positive constraints just inside these points in the direction opposite the known (or desired) normals. (Figure courtesy of Greg Turk.)

This may be generalized to a *scattered-data interpolation* problem as follows. Given a set of positions $\overline{c}_i$ and corresponding values h_i, solve for an embedding function f such that $f(\overline{c}_i) = h_i$. Thus, surface interpolation may be turned into higher-dimensional scattered-data interpolation, a well-studied field.

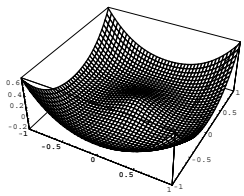
Turk and O'Brien chose to use *thin-plate splines*, which minimize the bending energy of the embedding function: $E = \int_\Omega f_{xx}^2(\overline{x}) + f_{xy}^2(\overline{x}) + f_{yy}^2(\overline{x})\ d\overline{x}$. They called their method *variational implicit surfaces*, because they formulate the problem as one of variational interpolation (minimizing an energy functional subject to interpolative constraints). They did not, however, solve for the embedding function using an iterative minimization approach but instead solve for the known closed-form solution using radial basis functions as described in Section 2.2. (Savchenko, et al. use similar splines based on the use of Green's function.)

2.2 Radial Basis Functions

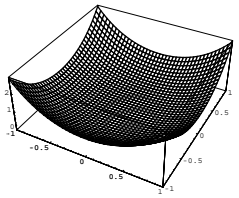
Scattered data interpolation can be achieved using *radial basis functions* centered at the constraints. Radial basis functions are circularly-symmetric functions centered at a particular point.

Duchon [5] has shown that solving for thin-plate splines through known points in two dimensions is equivalent to interpolating these points using the biharmonic radial basis function $\phi(r) = r^2 \log |r|$ (Figure 2a). In three dimensions, the thin-plate solution is equivalent to interpolating these points using the radial basis function $\phi(r) = |r|^3$ (Figure 2b).

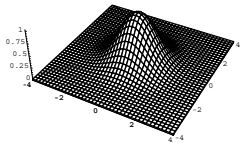
Radial basis functions may be used to interpolate a function with n points by using n radial basis functions centered at these points. The resulting interpolated function thus be-



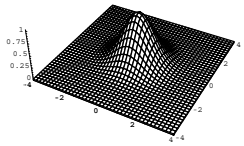
a) Thin-plate (2-d)
$\phi(r) = r^2 \log r$



b) Thin-plate (3-d)
$\phi(r) = r^3$



c) Gaussian
$\phi(r) = e^{r^2\sigma^2}$



d) Compactly Supported
$\phi(r) = (1-r)^4_+(4r+1)$

Figure 2. Comparison of different radial basis functions

comes

$$f(\overline{x}) = \sum_{i=1}^{n} d_i \phi(\|\overline{x} - \overline{c}_i\|) \tag{1}$$

where $\overline{c}_i$ is the position of the known values, d_i is the weight of the radial basis function positioned at that point. In some cases (including the thin-plate spline solution), it is necessary to add a first-degree polynomial P to account for the linear and constant portions of f and ensure positive-definiteness of the solution:

$$f(\overline{x}) = \sum_{i=1}^{n} d_i \phi(\|\overline{x} - \overline{c}_i\|) + P(\overline{x}) \tag{2}$$

To solve for the set of weights d_i that satisfy the known constraints $f(\overline{c}_i) = h_i$, we substitute each $\overline{c}_i$ into Eq. 2:

$$f(\overline{c}_i) = \sum_{j=1}^{n} d_j \phi(\|\overline{c}_i - \overline{c}_j\|) = h_i \tag{3}$$

or, if a polynomial is required:

$$f(\overline{c}_i) = \sum_{j=1}^{n} d_j \phi(\|\overline{c}_i - \overline{c}_j\|) + P(\overline{x}) = h_i \tag{4}$$

Solving for the weights d_j using Eq. 3 and denoting $\phi_{ij} = \phi(\|\overline{c}_i - \overline{c}_j\|)$ produces the following system:

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \end{bmatrix} \tag{5}$$

If a polynomial is required, Eq. 4 similarly becomes

$$\begin{bmatrix} \phi_{11} & \phi_{12} & \dots & \phi_{1n} & c_1^x & c_1^y & c_1^z & 1 \\ \phi_{21} & \phi_{22} & \dots & \phi_{2n} & c_2^x & c_2^y & c_2^z & 1 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{n1} & \phi_{n2} & \dots & \phi_{nn} & c_n^x & c_n^y & c_n^z & 1 \\ c_1^x & c_2^x & \dots & c_n^x & 0 & 0 & 0 & 0 \\ c_1^y & c_2^y & \dots & c_n^y & 0 & 0 & 0 & 0 \\ c_1^z & c_2^z & \dots & c_n^z & 0 & 0 & 0 & 0 \\ 1 & 1 & \dots & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \\ p^x \\ p^y \\ p^z \\ 1 \end{bmatrix} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_n \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{6}$$

In both Eqs. 5 and 6 the matrix is obviously real symmetric, and with proper selection of basis functions it can be made positive-definite. Thus, a solution always exists to these systems.

2.3 Algorithmic Complexity

Calculating and using implicit surfaces that interpolate may be analyzed in three parts:

1. Constructing the system of equations,
2. Solving the system of equations, and
3. Evaluating the interpolating function (as required).

2.3.1 Constructing the System of Equations

A significant portion of the computational cost involved in calculating these implicit surfaces is the cost required to construct the matrix (or submatrix) $\phi_{ij} = \phi(\|\overline{c}_i - \overline{c}_j\|)$. Recall that the thin-plate radial basis function is $\phi(r) = r^2 \log r$ (two dimensions) or $\phi(r) = r^3$ (three dimensions). This means that *the matrix is entirely non-zero except along the diagonal*, requiring the calculation of all inter-point distances within the set $\{\overline{c}_i\}$. Although the symmetry of the matrix cuts the computational cost in half, the computational complexity is still $O(n^2)$. Furthermore, storage of such a matrix requires $O(n^2)$ floating-point values—a potentially more prohibitive factor than the computational complexity.

2.3.2 Solving the System of Equations

Although Turk and O'Brien use LU factorization (an $O(n^3)$ algorithm) to solve Eq. 5, they correctly point out that it is possible to solve this system in $O(n^2)$ by iterative means. Thus, while solution of the system may appear to be the limiting step, it need only be as computationally expensive as constructing the system.

2.3.3 Evaluating the Function

For nearly all applications it is not enough to simply solve for the weights of the respective radial basis functions.

Rather, it is necessary to evaluate this embedding function at potentially many points in order to extract the isosurface, calculate normals or other derivative quantities, etc. Because the terms $\phi(\|\overline{x} - \overline{c}_i\|)$ in Eq. 1 are all non-zero for the thin-plate solution (except for one zero term when $\overline{x} \in \{\overline{c}_i\}$), all of the terms must be used in calculating any one point. Thus, each evaluation of the interpolated function is $O(n)$.

2.4 Problems with the Thin-Plate Solution

While the thin-plate spline embedding function does indeed minimize bending energy, it has the following drawbacks in computation and usefulness for user interaction:

1. $O(n^2)$ computation is required to build the system of equations.
2. $O(n^2)$ storage is required (for the nearly-full matrix) to represent the system.
3. $O(n^2)$ computation is required to solve the system of equations.
4. $O(n)$ computation is required per evaluation
5. Because every known point affects the result, a small change in even one constraint is felt throughout the entire resulting interpolated surface, an undesirable property for shape modelling.

3 Using Compactly-Supported Radial Basis Functions

Wendland [16] has recently solved for the minimum-degree polynomial solution for compact, locally-supported radial basis functions that guarantee positive-definiteness of the matrix (Figure 2d). All of the solutions have the form

$$\phi(r) = \begin{cases} (1-r)^p P(r) & \text{if } r < 1 \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

For various degrees of desired continuity (C^k) and dimensionality (d) of the interpolated function, he has derived the following radial basis functions:

$d = 1$	$(1-r)_+$	C^0
	$(1-r)_+^3(3r+1)$	C^2
	$(1-r)_+^5(8r^2+5r+1)$	C^4
$d = 3$	$(1-r)_+^2$	C^0
	$(1-r)_+^4(4r+1)$	C^2
	$(1-r)_+^6(35r^2+18r+3)$	C^6
	$(1-r)_+^8(32r^{3'}+25r^2+8r+1)$	C^6
$d = 5$	$(1-r)_+^3$	C^0
	$(1-r)_+^5(5r+1)$	C^2
	$(1-r)_+^7(16r^2+7r+1)$	C^4

These functions have radius of support equal to 1. Scaling of the basis functions (i.e., $\phi(r/\alpha)$) allows any desired radius of support α.

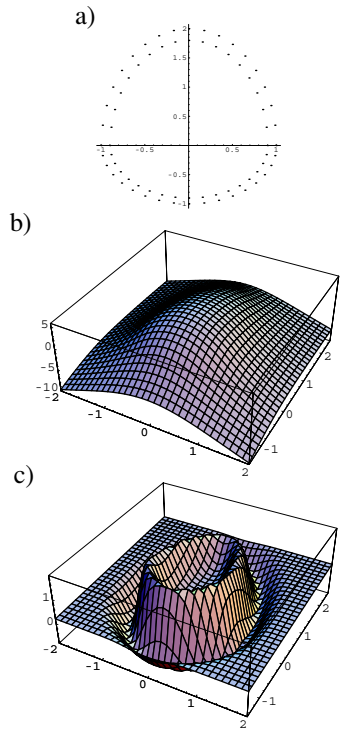


Figure 3. A simple 36-point ovoid (a) interpolated using thin-plate (b) and compactly-supported (c) radial basis functions.

3.1 Example

Figure 3 illustrates using both thin-plate and compactly-supported radial basis functions to compute embedding functions. The constraint points consist of 36 points in an ovoid shape with 36 normal (positive valued) constraints placed just inside (3a). A thin-plate radial basis function produces a globally-smooth embedding function (3b). A compactly-supported radial basis function produces an embedding function that does not have global smoothness but is as smooth as the thin-plate spline interpolation in a narrow band surrounding the shape both inside and out.

3.2 Algorithm

Using compactly-supported radial basis functions provides advantages in all three phases of the implicit-surface interpolation algorithm.

3.2.1 Constructing the system

Because the radial basis functions have finite support, $\phi(\|\overline{c}_i - \overline{c}_j\|) = 0$ for all $(\overline{c}_i, \overline{c}_j)$ farther apart than the radius of support. By using a k-d tree [1], the set of all points

within distance r of a particular point $\overline{c}_i$ can be determined in $O(\log n)$ time.

A k-d tree is a multidimensional binary tree with the following sorting property for a tree with point $\overline{x}$ at the root and subtrees T_{left} and T_{right}.

$$\forall \overline{y} \in T_{\text{left}} : \overline{y}^d \leq \overline{x}^d$$

$$\forall \overline{y} \in T_{\text{right}} : \overline{y}^d > \overline{x}^d$$

where the sorting dimension d changes at each level of the tree.

k-d trees can be used to find all points within distance r of a particular constraint in $O(n \log n)$ time using the following algorithm (C pseudocode):

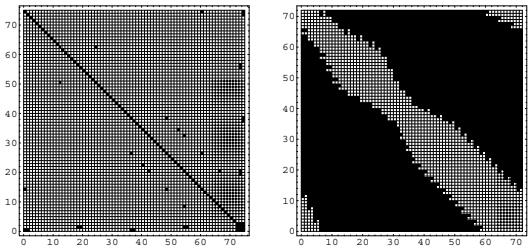
```
void Search (KDtree T, Point P, int radius)
{
  if (T->point[T->dim]) < P[dim] + radius)
    Search(T->right);
  if (T->point[T->dim]) > P[dim] - radius)
    Search(T->left);
  Test(T->point,P,radius);
}
```

While a number of points must still be tested explicitly, the multidimensional sorting nature of the k-d tree allows a large number of points to be rejected at each level of the tree.

The resulting matrix is extremely sparse, as shown in Figure 4. Using a sparse-matrix representation (we use the Hartwell-Boeing format), only $O(n)$ storage is required.

3.2.2 Solving the system

If the average number of points within the radius of support of each constraint $\overline{c}_i$ is less than some constant k, the number of non-zero entries in the matrix is $O(n)$. We use a direct (LU) sparse matrix solver [4] to find the solution to the system of equations. The computational complexity of such a solver depends on the amount of matrix "fill in" that occurs during the solution, but some authors have reported behavior in the range $O(n^{1.2})$ to $O(n^{1.5})$ [10]. Our own experience (Section 4) agrees with this.



Thin-plate Compactly-supported

Figure 4. Structure of the matrices produced by thin-plate and compactly-supported radial basis functions (Figure 3). The compactly-supported basis function produce a matrix that is sparse (black), while the thin-plate basis functions produce a matrix that is nearly full (white).

3.2.3 Evaluating the interpolating function

We can exploit the spatial locality of the compactly-supported radial basis functions during evaluation of the embedding function f by recognizing that only a fraction of the terms of Eq. 1 are non-zero for a given $\overline{x}$: $\phi(\overline{c}_i - \overline{c}_j) \neq 0$ iff $\|\overline{c}_i - \overline{c}_j\| < 1$. By again using a k-d tree to organize the constraints spatially, each evaluation of the interpolating function requires only $O(\log n)$ operations to determine these non-zero terms.

3.3 Thin-Plate vs. Compactly-Supported Radial Basis Functions

Using compactly-supported radial basis functions directly addresses each of the five problems identified previously with the thin-plate spline basis functions (Section 2.4) as follows:

	Thin-plate	Compact
Computation to build	$O(n^2)$	$O(n \log n)$
Computation to solve	$O(n^2)$	$O(n^{1.5})$
Storage to build/solve	$O(n^2)$	$O(n)$
Computation to evaluate	$O(n)$	$O(\log n)$
Effect of a single point	Global	Local

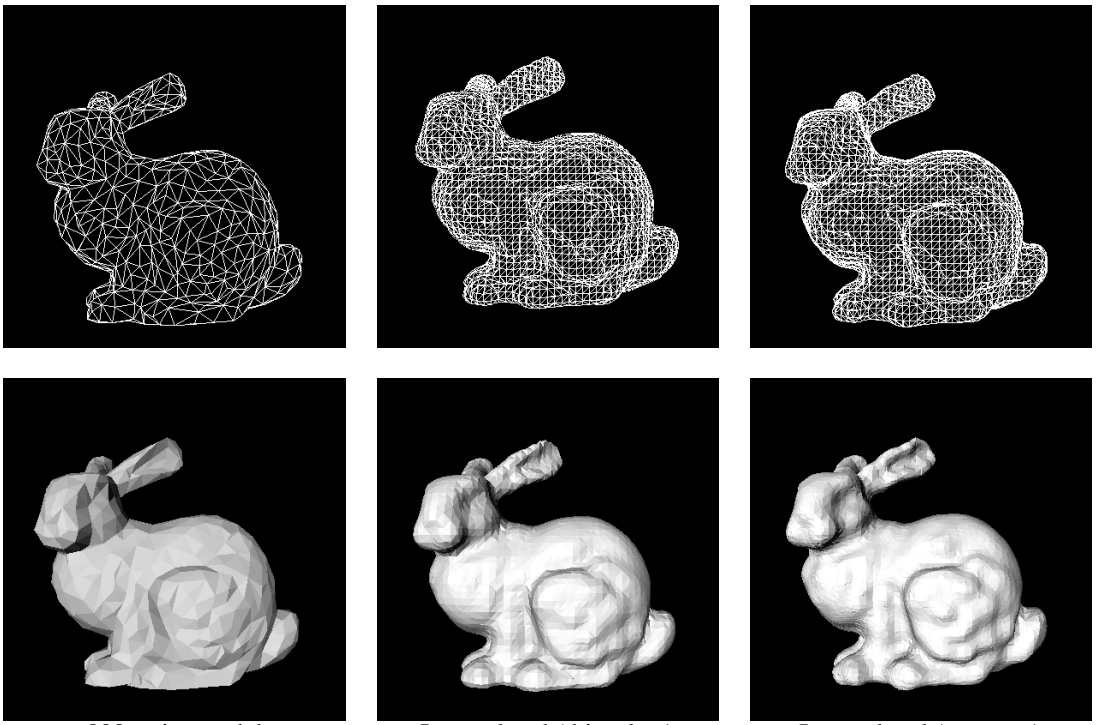
4 Results

Figure 5 compares the results of interpolating an 800-point model of the Stanford bunny using both thin-plate and compactly-supported radial basis functions. The results of of the two methods are qualitatively identical.

Interpolation of larger models is possible using compactly-supported radial basis functions. The 8000-point model in Figure 6 was interpolated using compactly-supported radial basis functions, and the resulting isosurface was tesselated to 41,864 points.[1] The interpolated image shows sharpness and detail hidden by the flat polyhedra of the original model.

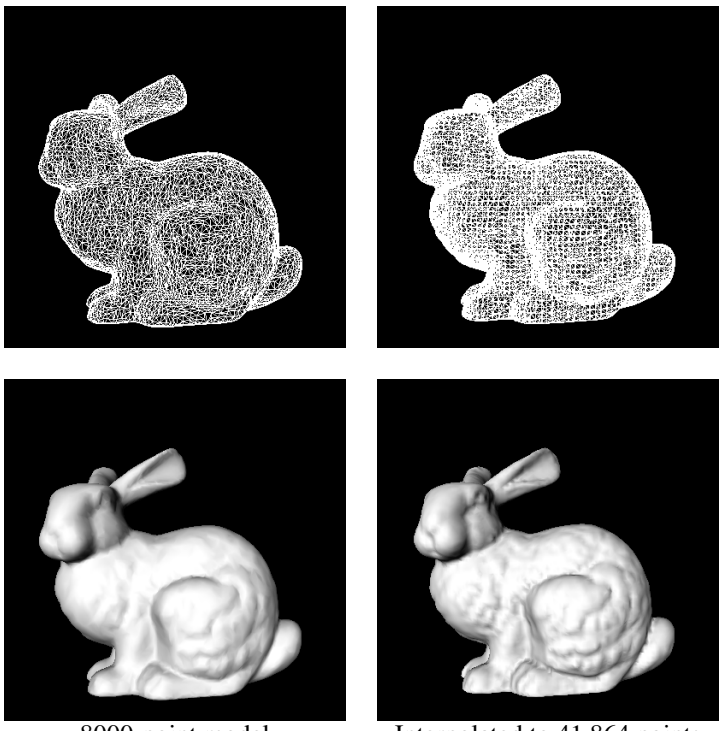
Table 1 summarizes the time required to calculate various differing-resolution models of the same figure (Stanford bunny). The radius of support used for each model was selected so as to keep the number of points in the radius of

[1] A similar thin-plate interpolation would require almost 2 GB of storage for the matrix alone and could not be calculated on our system.



800-point model | Interpolated (thin-plate) | Interpolated (compact)

Figure 5. Comparison of original model to implicit surfaces extracted from embedding functions calculated using both thin-plate and compactly-supported basis functions. The results of the interpolations are qualitatively identical.



8000-point model | Interpolated to 41,864 points

Figure 6. Interpolation of an 8000-point model of the Stanford bunny

Points	Constraints	Radius	Non-zero	Per Row	% Full	Build k-d	Build Matrix	Solve Matrix	Total
2922	5848	0.200	476173	81.4	1.39%	0.02	1.05	5.93	7.00
5839	11682	0.150	973423	83.3	0.71%	0.04	2.09	17.29	19.42
11831	23666	0.100	1663733	70.3	0.30%	0.08	4.00	45.44	49.52
35947	71898	0.004	5061542	70.4	0.10%	0.93	31.51*	284.01*	316.45*

* some virtual memory swapping

Table 1. Execution times in seconds for various phases of computation for interpolated implicit surfaces using compactly-supported radial basis functions.

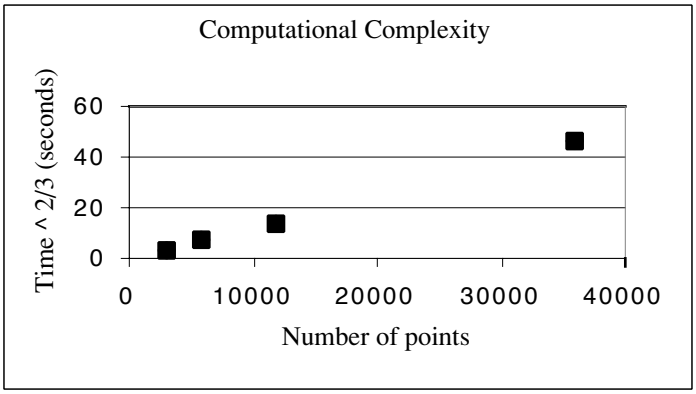


Figure 7. Timing results for four different-resolution models of the same figure (Table 1). The linear graph when plotted on a $time^{\frac{2}{3}}$ vertical axis indicates $O(n^{1.5})$ complexity.

Thin-Plate			Compactly-Supported		
Points	Build	Solve	Points	Build	Solve
800	0.58 sec	2.56 min	800	0.57 sec	1.53 sec
2000	6.3 sec	2:05 hrs	2922	1.07 sec	5.93 sec
4000	25.0 sec	16:11 hrs*	5839	2.13 sec	17.29 sec
8000	n/a (1)	n/a	11831	4.08 sec	97.53 sec
35947	n/a (2)	n/a	35947	31.44 sec	284.01 sec

* significant virtual memory swapping

Table 2. Comparison of execution times required to calculate embedding functions using thin-plate vs. compactly-supported radial basis functions

Thin-Plate		Compactly-Supported	
Points	Memory (MB)	Points	Memory (MB)
800	19.5	800	1.0
2000	122.1	2922	3.6
4000	488.3	5839	7.4
8000	1,953.1	11831	12.7
35947	39,434.4	35947	38.6

Table 3. Comparison of memory requirements (matrix only, double-precision floating-point values) required to calculate embedding functions using thin-plate vs. compactly-supported radial basis functions

support (approximately) constant, thus allowing comparison of the results. The dominant term in the required computation is the time to solve the system of equations, which seems to demonstrate $O(n^{1.5})$ complexity (Figure 7).

Table 2 compares these running times to the time required to calculate comparable thin-plate interpolations (using same-size or smaller models). The $O(n^2)$ time required to compute these models using thin-plate radial basis functions quickly becomes prohibitive.

Similarly, Table 3 compares the memory required to represent the matrix for the models described in Table 2. As with the computational complexity, the $O(n^2)$ storage required to compute these models using thin-plate radial basis functions also quickly becomes prohibitive.

5 Considerations

The finite nature of the compactly-supported radial basis functions introduces two factors that must be considered when using them to interpolate embedding functions.

5.1 Selecting the Radius of Support

The finite radius of support introduces an additional parameter that doesn't exist in the thin-plate implementation. Proper selection of the radius of support is critical to achieving optimal efficiency of computation and results. Too small a radius can produce basis functions that are unable to span the inter-constraint gaps. Too large a radius does not adversely affect the results but reduces the sparseness of the matrix, thus increasing the computation required. It is thus necessary to select a radius of support that is both large enough to produce effective results and not so large that the computation becomes impractical.

5.2 Isosurface Extraction

Because of the finite extent of the compactly-supported radial basis functions, only those points within the radius of support of one of the original positions have non-zero values. For all points outside this band, all of the terms of Eq. 1 are zero. (Figure 3c illustrates this.) In this way, these embedding functions are not the same as those normally used for implicit surfaces—the implicit surface represented is not the only set of zero-valued points in the space. However, the implicit surface does form a unique contiguous locus of zero-valued points passing through the constraints. In this sense, the method presented here is somewhat similar to the narrow-band active set approach of Sethian [12] or Whitaker and Breen [17].

An isosurface extractor may be used to extract this surface by seeding it with any one of the initial constraints. However, care must be taken so that the step size of the extractor does not cause it to jump outside the band of non-zero points. It is, however, rather easy to explicitly recognize when no non-zero terms are found in Eq. 1 (none of the constraint points lie within the radius of support of the point being evaluated).

Because the surface of interest is not the only zero set of the embedding function, the resulting embedding function has limited application in CSG, interpolation [15], or similar applications. However, we are experimenting with a hybrid approach that interpolates a subset of the points using radial basis functions with infinite support and the difference using basis functions with compact support.

6 Discussion: Analytic Approaches Enabled by Efficient Algorithms

An efficient framework for finding embedding functions that define implicit surfaces from scattered data points increases the practicality of studying complex shape models represented by large numbers of such points. Models captured from physical phenomena usually contain large numbers of surface points, polygons, or other surface primitives that are not easily reduced. For instance, polygonal representations of medical data often begin with models containing millions of triangles, which can later be simplified to hundreds of thousands of polygons.

While the polygonal representations of these models can be rendered using current graphics hardware, the discrete sampling introduced by the process of producing the polyons raises barriers to deeper studies of the geometry of the surfaces themselves. Implicit methods solve this difficulty by creating analytic functions that smoothly reconstruct a surface from a constellation of points. In addition, the implicit surface constructed using the method presented here is differentiable. Local surface geometry now becomes approachable since numerically stable solutions can be found to sample higher order derivatives of the implicit surface.

6.1 Differential Geometry of Shape

Differential geometry is the study of multilocal surface behavior, employing the normal vector and tangent plane at each surface point as a reference environment. Curvature and other geometric features relative to the surface tangent are measured using second order derivatives [13]. Volumetric approaches can compute approximations to these measurements based on discretely sampled grids [12]. However, the accuracy and behavior of the sampled derivatives is subject to aliasing and sampling issues, exacerbated by the noise-amplifying effects of higher-order functions.

Implicit surfaces allow us to reconstruct smooth surface representations from a set of oriented points and sample

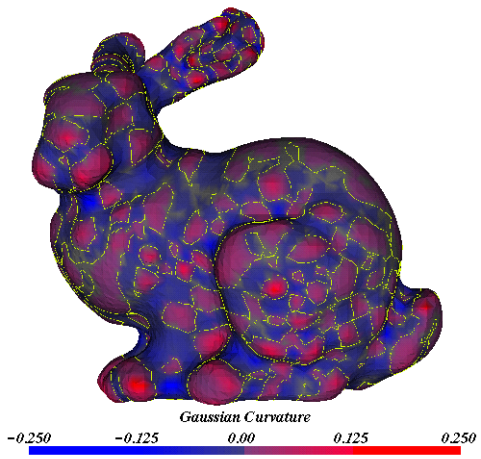


Figure 8. Gaussian curvature computed over an analytically-defined implicit surface calculated from scattered surface points

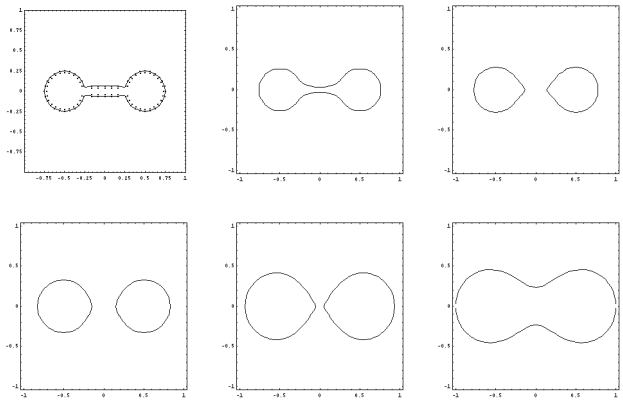


Figure 9. A dumbell figure reconstructed from sample points and represented at successively larger scales by convolving the embedding function with successively larger Gaussian kernels

higher order derivatives with instantaneous precision. Analytic representations of not only surface shape, but also of the embedding function, simplifies the computation of the normal vector, the tangent plane, and the principal curvatures of the implicit surface at any arbitrary location.

Figure 8 shows the Gaussian curvature (the product of the principal curvatures) of an array of implicit surfaces calculated using the method presented here. Blue areas represent positive Gaussian curvature (elliptical regions) and red areas represent negative Gaussian curvature (hyperbolic or saddle-shaped regions). The yellow contour lines indicate the places where the Gaussian curvature is zero, separating the red and blue surface regions. The yellow contours denote parabolic curves where specular and diffuse highlights fuse or reproduce under changing lighting conditions.

Note: the polygonalization of the surface and the interpolation errors are artifacts of the visualization technique. The implicit surface itself can be sampled with arbitrary precision to generate smooth models with correspondingly smooth representations of curvature.

6.2 Future Directions: Scale Space, Ray Tracing, and Other Topics

Linear scale space filtering was introduced by Witkin [19] as a means of measuring the saliency of features within an image. Further work established the general field of scale-space theory in computer vision [8]. The fundamental notion of such analysis is that significant details of an image (or surface representation) persist as the scale or the measurement aperture is increased. In the pursuit of multiscale image descriptions, a Gaussian kernel is usually used as the measurement aperture function.

From this perspective, a scale space representation of the implicit models presented in this paper can be constructed as a convolution of a Gaussian kernel with the embedding function, $f(\bar{x}) \otimes g(\bar{x}, \sigma)$. Since convolution is both commutative and distributive with respect to addition, this is equivalent to convolving each of the radial basis interpolants with a Gaussian. This process can be approximated by solving for the implicit surface with a particular radius of support, and later dilating the compactly supported radial basis function during the evaluation to create a scale space level set representation of the basic function.

Figure 9 shows a 2-D implicit figure represented at a range of scales. The original model has been reconstructed from oriented points as an implicit surface. In the subsequent representations, the embedding function has been convolved with an approximation to a Gaussian kernel, and the implicit surface reconstructed. Fine details such as the corners and discontinuities associated with the cross member in the figure are suppressed at moderate scales. Eventually at the largest scales, the figure is viewed as a single topologically simple object. This approach to representing object shape may have applications in modeling and computer graphics in the representation of objects at multiple levels of detail.

Beyond the application of scale-space image-analysis techniques to implicit surface representations, there remain interesting problems of rendering these models. Implicit surfaces based on signed distance functions and other embedding functions with similar properties are easily rendered through ray tracing. Because of the multiple zero level sets created by the compactly supported radial basis function approach, a basic ray tracing method is insufficient for rendering these models. However, in the visual-

ization of discrete volume data, complex transfer functions that include geometric information such as gradient magnitude and isosurface curvature are able to capture surfaces based on features other than simple isovalues. Future work will include the development of transfer functions for raycast rendering of these models.

7 Conclusion

Given a set of points $C = \{\overline{c}_1, \overline{c}_2, \ldots \overline{c}_n\}$ and associated normals, the method presented here interpolates an implicit surface through those points by finding a scalar embedding function $f(\overline{x})$ whose zero level set, $\{\overline{x} : f(\overline{x}) = 0\}$, passes through all points in C. Following Turk and O'Brien [14] (see also [11]), we employ radial basis interpolating functions in a linear system of $2n$ equations and $2n$ unknowns, expressed as an $2n \times 2n$ matrix. Unlike the original use of $\phi(r) = |r|^3$ as the underlying interpolant, we use a family of radial basis functions with the necessary continuity but also with compact local support. The result is a sparse system whose solution can be accelerated. The result is a single, accelerated, closed form analytic representation of the desired surface.

The shift from a radial basis function of infinite extent to a compactly supported one creates dramatic gains in memory utilization and computational complexity. Previous work described solutions for systems of equations of order $O(n^2)$ complexity. The shift to finite interpolants and sparse matrices has shifted the complexity of the matrix solution to order $O(n^{1.5})$, the loading of the matrix data structures to $O(n \log n)$, and the memory requirements to order $O(n)$. Evaluation of the interpolated embedding function is similarly reduced to $O(\log n)$.

These improvements in efficiency make possible a variety of applications that were previously impractical with an infinite radial basis function. We have briefly surveyed our first probes into the differential geometry of surface shape and explorations in scale space analysis of complex models using implicit surfaces interpolated from scattered surface data points.

Acknowledgments

This work was performed in large part at the National Library of Medicine under a visiting faculty program supporting both Dr. Morse and Dr. Subramanian. Dr. Rheingans was supported in part by NSF CAREER Grant #9996043. We would like to thank Greg Turk for his useful conversations and for making his code available to us, upon which our implementation is based. We would also like to thank Dr. Michael Ackerman and the staff of NLM's Office of High Performance Computing and Communications for their help and support. Finally, we would like to thank the reviewers and program committee for SMI2001 for their many helpful suggestions.

References

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.

[2] J. Blinn. A generalization of algebraic surface drawing. *IEEE Transactions on Graphics*, 1(3):235–246, 1982.

[3] J. Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan-Kaufman, 1997.

[4] J. J. Dongarra, J. D. Croz, S. Hammarling, and I. Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, Mar. 1990.

[5] J. Duchon. Sur l'erruer d'interpolation des fonctions de plusieurs variables par les d^m splines. *R.A.I.R.O Analyse numerique*, 12(4):325–334, 1978.

[6] J. Hart and e. D. Ebert. *New Fontiers in Modeling and Texturing*. Siggraph 97 Course Notes, 1997.

[7] B. Kimia, A. Tannenbaum, and S. Zucker. On optimal control methods in computer vision and image processing. In B. t.H. Romeny, editor, *Geometry Driven Diffusion in Computer Vision*, pages 307–338. Kluwer, 1994.

[8] T. Lindeberg. *Scale-space theory in computer vision*. Kluwer Academic Publishers, 1994.

[9] S. Osher and J. A. Sethian. Fronts propogating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation. *J. Comput. Phys.*, 79:12–49, 1988.

[10] R. A. Saleh, K. A. Gallivan, M. Chang, I. N. Hajj, D. Smart, and T. N. Patrick. Parallel circuit simulation on supercomputers. *Proceedings of the IEEE*, 77(12):1915–1930, 1989.

[11] V. V. Savchenko, A. A. Pasko, O. G. Okunev, and T. L. Kunii. Function representation of solids reconstructed from scattered surface points and contours. *Computer Graphics Forum*, 14(4):181–188, 1995.

[12] J. A. Sethian. *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[13] J. Thirion. New feature points based on geometric invariants for 3d image registration. Technical Report INRIA-RR-1901, INRIA, 1993.

[14] G. Turk and J. F. O'Brien. Variational implicit surfaces. Technical Report GIT-GVU-99-15, Georgia Institute of Technology, 1998.

[15] G. Turk and J. F. O'Brien. Shape transformation using variational implicit surfaces. In *Computer Graphics Proceedings, Annual Conference Series*, 1999.

[16] H. Wendland. Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree. *AICM*, 4:389–396, 1995.

[17] R. Whitaker and D. Breen. Level-set models for the deformation of solid objects. In *The Third International Workshop on Implicit Surfaces*, pages 19–35. Eurographics, 1998.

[18] A. Witkin and P. Heckbert. Using particles to sample and control implicit surfaces. In A. Glassner, editor, *SIGGRAPH '94 Proceedings*, Computer Graphics Proceedings, Annual Conference Series, pages 269–278. ACM SIGGRAPH, ACM Press, July 1994.

[19] A. P. Witkin and J. M. Tenenbaum. On the role of structure in vision. In J. Beck, B. Hope, and A. Rosenfeld, editors, *Human and Machine Vision*, pages 481–543. Academic Press, New York, NY, 1983.

Implicit Shapes: Reconstruction from Image-Based Data

H. Quynh Dinh

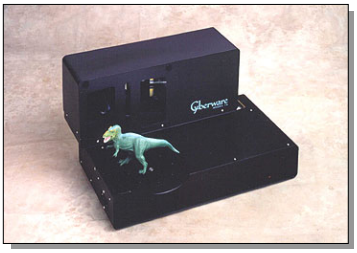
Multimedia Vision and Visualization Group
Department of Computer Science
Stevens Institute of Technology

Goals

- Automated 3D content creation
  - 3D surface scanning
  - Surface reconstruction from point sets
- Preservation of sharp features in Radial Basis Function (RBF) framework

3D Scanning

- Active scanners
  - Desktop scanner
    - Scan vol. 10" x 7" x 3"
    - $23,000



- Passive scanners
  - Vision algorithms
    - Stereo matching
    - Silhouette carving
    - Voxel coloring
  - $500-$2000

Voxel Coloring

[Images courtesy of Steve Seitz]

Problems in 3D Scanning

- View coverage of models
- Scanned data sets are noisy

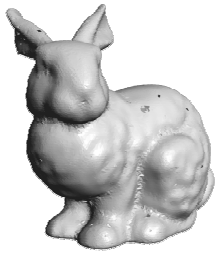
[courtesy of Steve Seitz]

Implicit Surface Reconstruction: Motivation

- Bridge gap between affordable vision systems and graphics applications
- Input data:
  - Noisy, non-uniform, low resolution
- Output model:
  - Smooth, seamless, manifold, detailed

Input: Image-based Range Data

- 3D surface point sets obtained from processing multiple images
- Comparison to active scanners:



Active Range Data

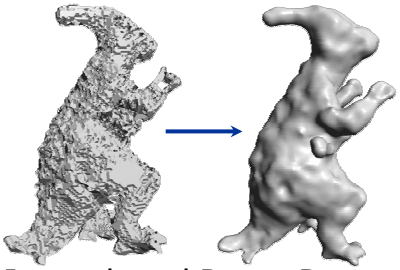
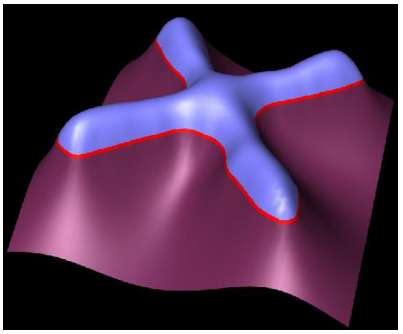


Image-based Range Data

Output: Implicit Surface

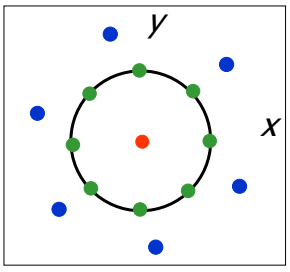
- Single implicit function, $f(\bar{x})$, defines shape:
  - $f(\bar{x}) = 0$ on surface
  - $f(\bar{x}) > 0$ inside
  - $f(\bar{x}) < 0$ outside

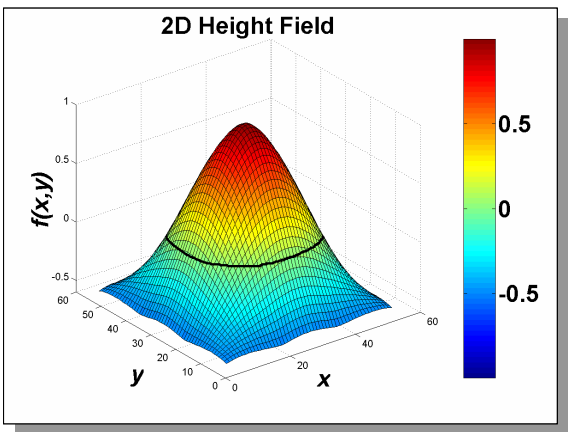


Overview of Concepts

- Interpolation vs. approximation
- Constructing approximating surfaces
- Multi-order radial basis functions
- Preserving sharp features using anisotropic basis functions

2D Implicit Function

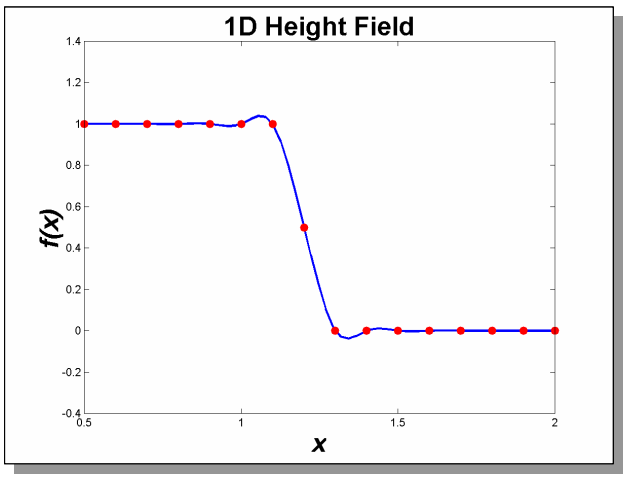




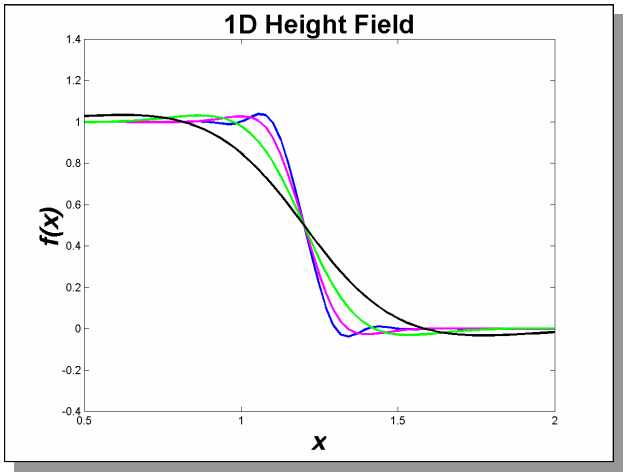
Reasonable Surfaces

- Surface reconstruction = data interpolation
- Infinitely many plausible surfaces that pass through given set of data points
- Goal is to find a *reasonable* surface
  - Continuity
  - Smoothness
  - Data fitness
- Balance factors using energy functional

Data Interpolation



Interpolation vs. Approximation



Interpolation vs. Approximation

- Surface interpolation retains noise

Surface Interpolation

Surface Approximation

Overview of Concepts

- ✓ Interpolation vs. approximation
- Constructing approximating surfaces
- Multi-order radial basis functions
- Preserving sharp features using anisotropic basis functions

Volumetric Regularization

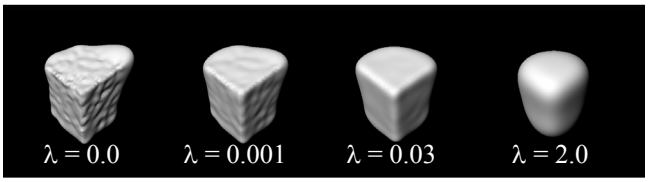
- Method of surface approximation
- Obtain shape function, *f*, by minimizing cost functional

$$H[f] = \sum_{i=1}^{n} \frac{1}{\lambda_i}(y_i - f(\vec{\mathbf{x}}_i))^2 + \beta[f]$$

y_i = observed data values at $\vec{\mathbf{x}}_i$
$\beta[f]$ = smoothness assumption
λ = regularization parameter

Volumetric Regularization

- Regularization parameter, λ, controls amount of fitness versus smoothness
- Large λ values promote smoothness



Overview of Concepts

- ✓ Interpolation vs. approximation
- ✓ Constructing approximating surfaces
- Multi-order radial basis functions
- Preserving sharp features using anisotropic basis functions

Solution: Radial Basis Functions

- Radially symmetric functions (e.g. Gaussians)
  - Duchon ’77
  - Blinn ’82
  - Yuille & Grzywacz ’88
  - Muraki ’91
  - Girosi et al ’93
  - Chen & Suter ’96, ’00
- Basis functions may be energy-minimizing
  - 1st order – minimizes surface area
  - 2nd order – minimizes curvature
  - 3rd order – minimizes change in curvature
  - Higher orders promote surface smoothness

Multi-Order RBF

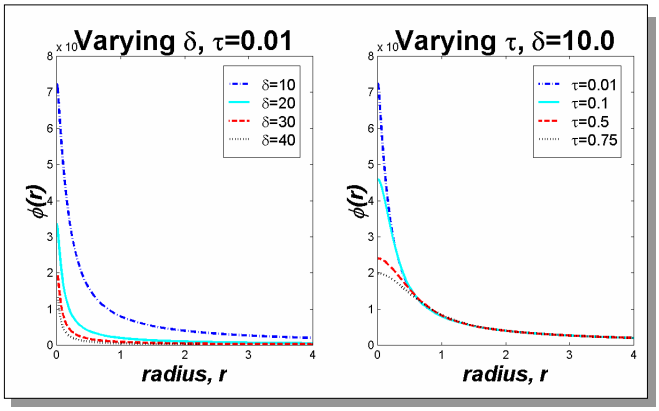
$$\phi(r) = \frac{1}{4\pi\delta^2 r}\left(1 + \frac{we^{-\sqrt{v}r}}{v-w} - \frac{ve^{-\sqrt{w}r}}{v-w}\right)$$

$$v = \frac{1+\sqrt{1-4\tau^2\delta^2}}{2\tau^2}$$

$$w = \frac{1-\sqrt{1-4\tau^2\delta^2}}{2\tau^2}$$

Smoothness assumption: $-\delta\Delta f + \Delta^2 f - \tau\Delta^3 f = 0$

Multi-Order RBF



- Large δ promotes sharpness
- Large τ promotes smoothness

Variational Implicit Surfaces

[Turk, O'Brien]

- Implicit function is a sum of weighted radial basis functions (RBF)

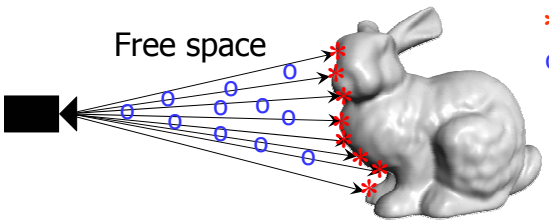
$$f(\vec{\mathbf{x}}) = \sum_{j=1}^{n} w_j \phi(\vec{\mathbf{x}} - \vec{\mathbf{c}}_j) + P(\vec{\mathbf{x}})$$

$\vec{\mathbf{x}}$ = 3D location
$\vec{\mathbf{c}}_j$ = center of basis
w = weights to be solved
$\phi(\vec{\mathbf{x}} - \vec{\mathbf{c}}_j)$ = radial basis function
$P(\vec{\mathbf{x}})$ = null space of $f(\vec{\mathbf{x}})$

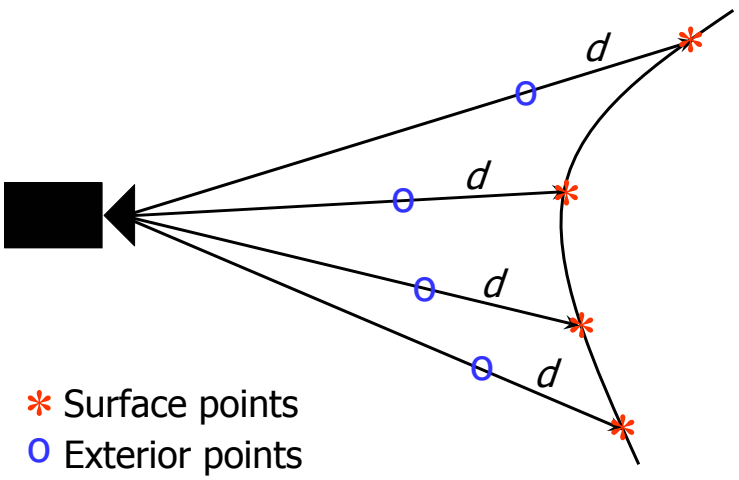
- Placement and weights of RBF?

Placement of RBFs

- RBFs are placed on the surface and at exterior points surrounding the model
- Use camera information to obtain exterior points



Placement of RBFs

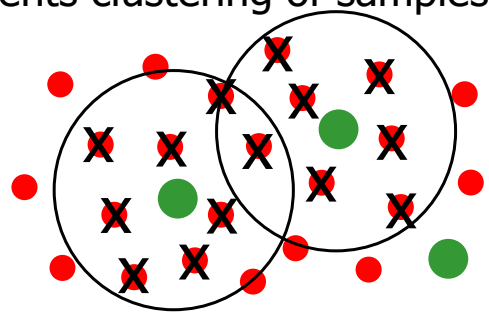


Uniform Sampling

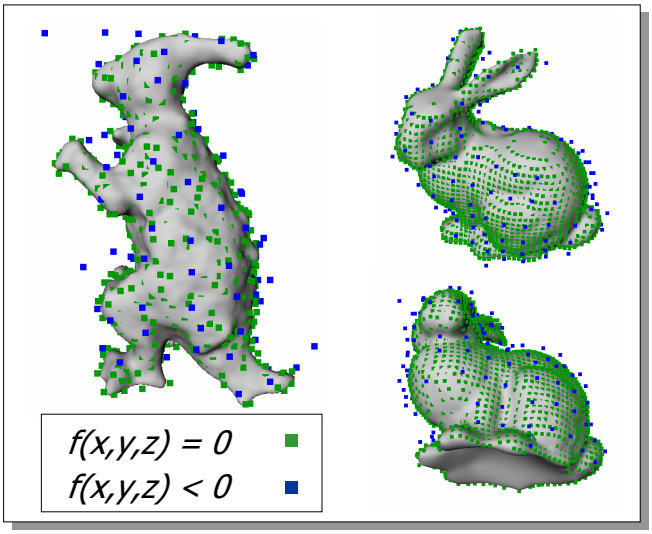
- Using entire dense data set is:
  - Intractible
  - Verbose
  - Unnecessary [Yngve & Turk, Carr et al '01]
- Subsample using Poisson disc sampling
- Select about 3000 surface and 300 exterior samples

Poisson Disc Sampling

- Neighborhood around each selected sample is eliminated
- Prevents clustering of samples



Placement of RBFs



Weights of RBFs

$$f(\vec{\mathbf{x}}) = \sum_{j=1}^{n} w_j \phi(\vec{\mathbf{x}} - \vec{\mathbf{c}}_j) + P(\vec{\mathbf{x}})$$

- Form linear system by applying known values to implicit function
- Solve for weights using a known matrix solver, e.g. conjugate gradient

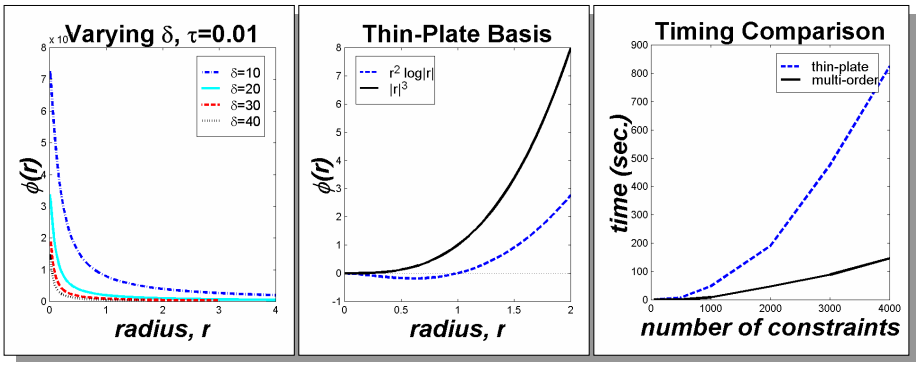
System Matrix

- λ appears on diagonal

$$\begin{bmatrix} \phi(r_{11}) + \lambda_1 & \cdots & \phi(r_{1n}) & \vec{c}_1 \\ \vdots & & \vdots & \vdots \\ \phi(r_{n1}) & \cdots & \phi(r_{nn}) + \lambda_n & \vec{c}_n \\ \vec{c}_1 & \cdots & \vec{c}_n & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ \vec{p} \end{bmatrix} = \begin{bmatrix} f(\vec{c}_1) \\ \vdots \\ f(\vec{c}_n) \\ 0 \end{bmatrix}$$

$$r_{ij} = |\vec{c}_i - \vec{c}_j|$$

Timing Results



Results

Voxel Coloring
Data Set

Reconstructed
Surface

Textured
Reconstruction

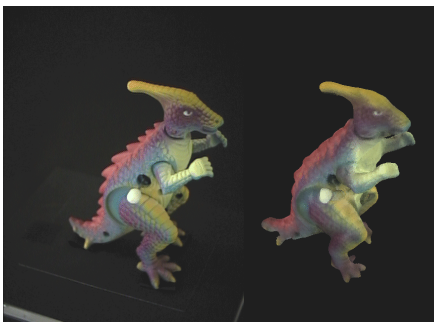
Comparison to Input Images

Input
Images

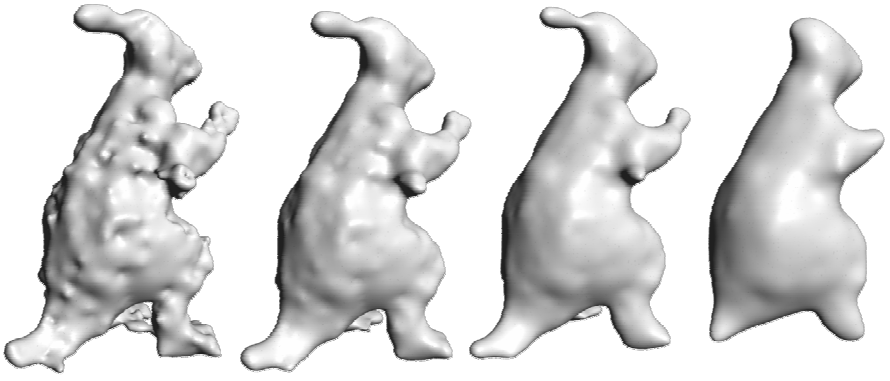
Output
3D Model

Input
Images

Output
3D Model

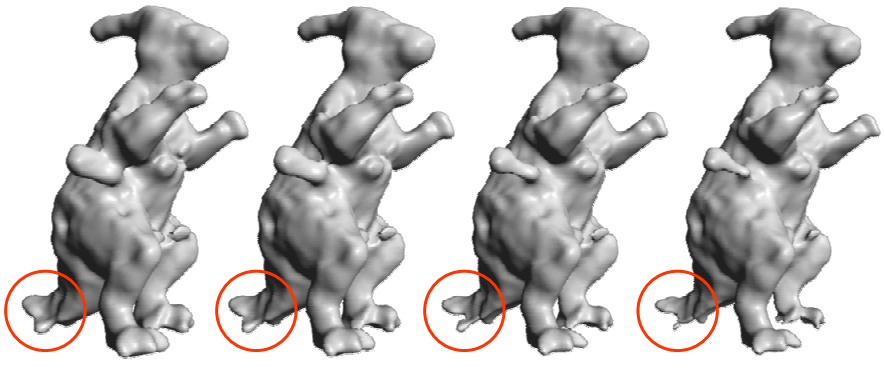


Varying λ



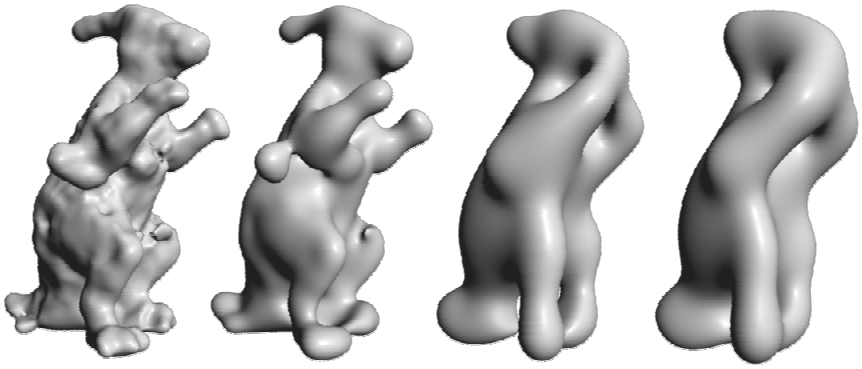
λ = 0.0 λ = 0.001 λ = 0.003 λ = 0.01

Varying δ, τ=0.01



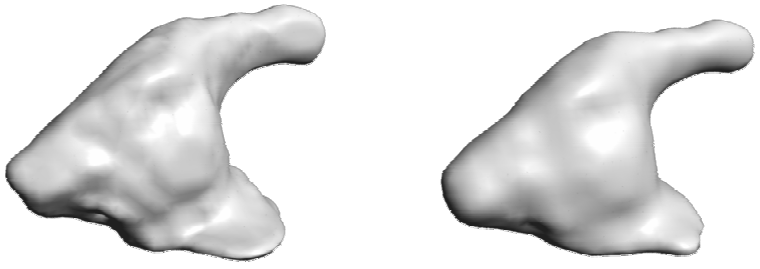
δ = 10 δ = 20 δ = 30 δ = 40

Varying τ, δ=0.25



τ = 0.01 τ = 0.1 τ = 0.5 τ = 0.75

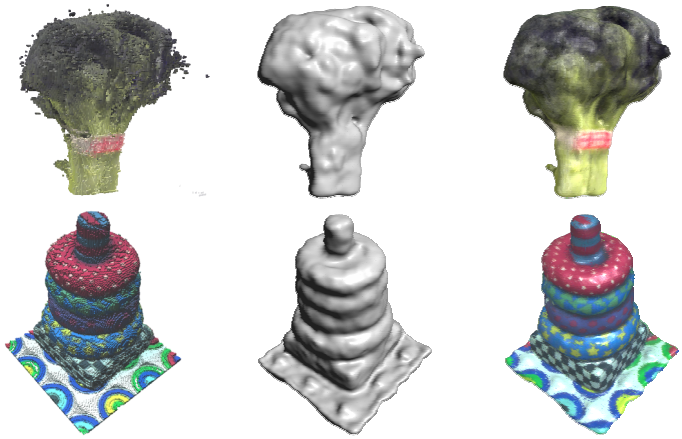
Effects of Subsampling Data Points



3173 Surface Points
(all data points)

477 Surface Points

Results



Voxel Coloring Data Set | Reconstructed Surface | Textured Reconstruction

Related Work in Reconstruction

- Surface representation
- Surface reconstruction
  - Range data merging [Turk & Levoy '94, Soucy & Laurendeau '95, Curless & Levoy '96, Hilton et al '96]
  - Region growing [Hoppe et al '92, Tang & Medioni '98]
  - Computational Geometry [Edelsbrunner & Mucke '94, Amenta et al '98, Bernardini et al '99]
  - Algebraic [Blinn '82, Taubin '93, Bajaj '95, Gotsman & Keren '99, Blane et al '00]
  - Superquadrics [Pentland '90, Terzopoulos & Metaxas '91, Sclaroff '91]
  - Level Sets [Whitaker '98, Breen & Whitaker '01]
- Surface smoothing
  - [Terzopoulos '88, Girosi et al '93, Nielson et al '93, Taubin '95, Desbrun et al '99]
- Regularization
  - [Boult & Kender '86, Fang & Gossard '95, Terzopoulos '86]

Related Work in Reconstruction

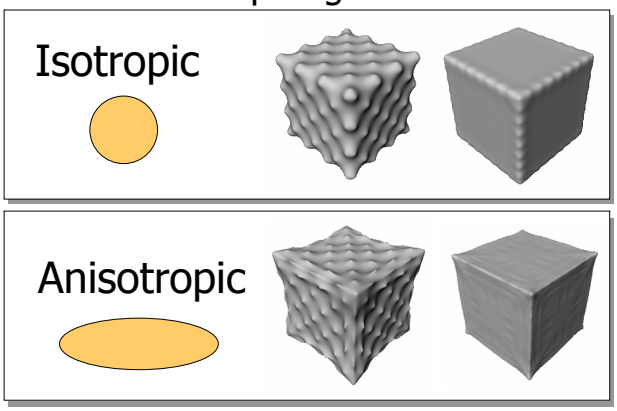
Method	Shape Rep.	Arbitrary Topology	Complex Models	Robust To Noise	Close Gaps
Distance Fields	Discrete	✓	✓		
Region Growing	Piecewise Cont.	✓	✓		
Computational Geometry	Piecewise Cont.	✓	✓		
Algebraic	Analytical	✓		✓	✓
Superquadrics	Analytical				
Level Sets	Piecewise Cont.	✓	✓	✓	✓
Volumetric Regularization	Analytical	✓	✓	✓	✓

Overview of Concepts

- ✓ Interpolation vs. approximation
- ✓ Constructing approximating surfaces
- ✓ Multi-order radial basis functions
- Preserving sharp features using anisotropic basis functions

Preserving Sharp Features

- Anisotropic basis function models asymmetry of surfaces at sharp edges



Anisotropic Basis Function

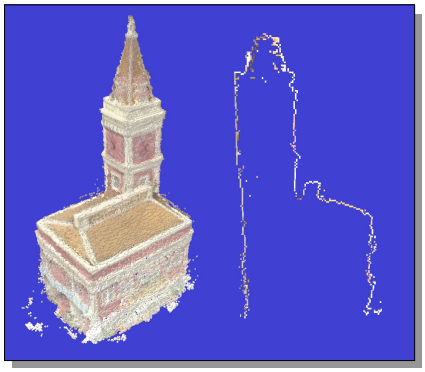
- Anisotropic basis *B* formed by stretching the isotropic basis ϕ

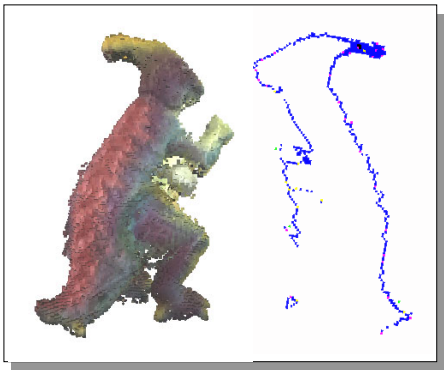
$$B(\vec{\mathbf{x}}) = \phi(|M(\vec{\mathbf{x}} - \vec{\mathbf{c}})|)$$

- *M* non-uniformly scales the distance function
- Placement, direction and magnitude of anisotropy?

Voxel Coloring Data

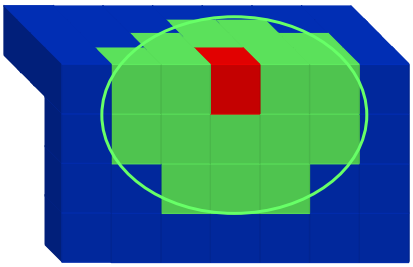
- Thin-shelled voxelized surface





Principle Component Analysis

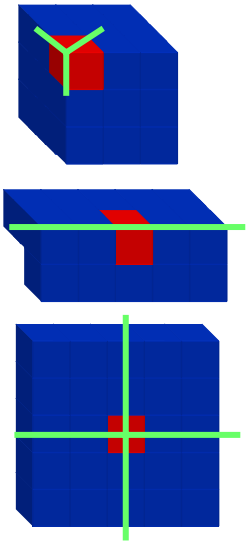
- Collect points in small neighborhood around each surface point



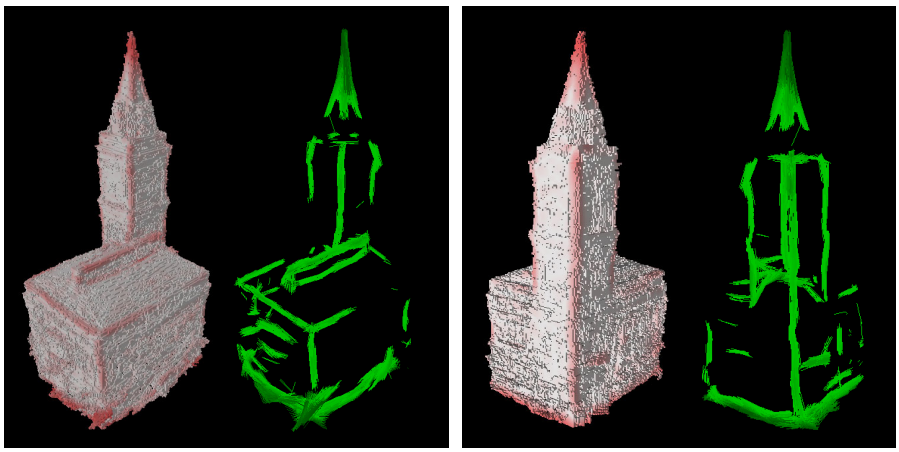
- Perform Eigenvalue decomposition of covariance matrix

Eigenvalues Indicate Features

- 3 small values
  - Corner point
- 1 large, 2 small values
  - edge point
- 2 large, 1 small value
  - planar point

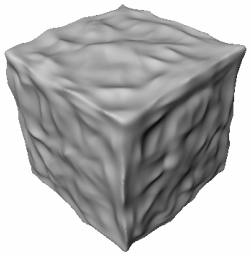


Anisotropic Magnitude & Direction

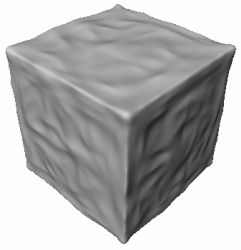


Filtering Anisotropic Directions

- PCA is sensitive to noise
- Low pass filter covariance matrices



Unfiltered



Filtered

Preserving Sharp Features

Voxel Coloring Data Set

Isotropic Reconstruction

Unfiltered Anisotropic

Filtered Anisotropic

Textured Reconstruction

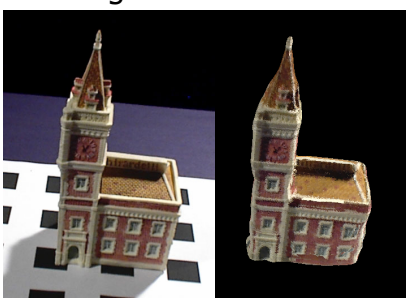
Comparison to Input Images

Input Images

Output 3D Model

Input Images

Output 3D Model



Summary

- Implicit surfaces from image-based data
  - Construct approximating surfaces
  - Multi-order basis function retains local detail
  - Sample *free space* for exterior data points
- Preservation of sharp features in RBF framework
  - Anisotropic basis function models asymmetry
  - Direction and magnitude of anisotropy determined by PCA

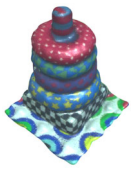
Application

- Automated 3D content creation
  - Affordable
  - Requires little user input
  - Robust to noise
  - Fills gaps in data set
  - Globally smooth, locally detailed surfaces

Acknowledgements

- Image-based range data sets
  - Steve Seitz (toy dinosaur)
  - Tom Malzbender & Bruce Culbertson (tori)
  - Greg Slabaugh (broccoli, Ghiradelli Square)
- Georgia Tech Geometry Group

Publications



Dinh, H.Q., G. Turk, and G. Slabaugh
"Reconstructing Surfaces by Volumetric Regularization Using Radial Basis Functions"
To appear in *IEEE Pattern Analysis and Machine Intelligence (PAMI), October 2002.*

Dinh, H.Q., G. Turk, and G. Slabaugh
"Reconstructing Surfaces Using Anisotropic Basis Functions"
Appears in *International Conference on Computer Vision (ICCV), July 2001.*

Reconstructing Surfaces By Volumetric Regularization Using Radial Basis Functions

Huong Quynh Dinh, Greg Turk, *Member*, *IEEE*, and Greg Slabaugh, *Student Member*, *IEEE*

Abstract—We present a new method of surface reconstruction that generates smooth and seamless models from sparse, noisy, nonuniform, and low resolution range data. Data acquisition techniques from computer vision, such as stereo range images and space carving, produce 3D point sets that are imprecise and nonuniform when compared to laser or optical range scanners. Traditional reconstruction algorithms designed for dense and precise data do not produce smooth reconstructions when applied to vision-based data sets. Our method constructs a 3D implicit surface, formulated as a sum of weighted radial basis functions. We achieve three primary advantages over existing algorithms: 1) the implicit functions we construct estimate the surface well in regions where there is little data, 2) the reconstructed surface is insensitive to noise in data acquisition because we can allow the surface to approximate, rather than exactly interpolate, the data, and 3) the reconstructed surface is locally detailed, yet globally smooth, because we use radial basis functions that achieve multiple orders of smoothness.

Index Terms—Regularization, surface fitting, implicit functions, noisy range data.

1 Introduction

The computer vision community has developed numerous methods of acquiring three-dimensional data from images. Some of these techniques include shape from shading, depth approximation from a pair of stereo images, and volumetric reconstruction from images at multiple viewpoints. The advantage of these techniques is that they use cameras, which are inexpensive resources when compared to laser and optical scanners. Because of the affordability of cameras, these vision-based techniques have the potential to enable the creation of digital models by home computer users who may not have professional CAD training. On the other hand, models in popular use in the entertainment industry (animation and gaming applications), video and image editing, and computer graphics research come from dense laser scans or medical scans, not from vision-based techniques. There are significant differences in terms of quality and accuracy between data sets obtained from active scanning technology (e.g., optical, laser, and time-of-flight range scanners) and passive scanning technology (e.g., shape from shading, voxel coloring) that use only images and camera calibration to obtain 3D point sets. Many of the well-known and often used reconstruction algorithms were designed to generate surfaces from dense and precise data such as those obtained from active scanners. These methods are not robust to the challenges posed by data obtained from passive scanning technology. The aim of our method is to be able to reconstruct smooth and continuous surfaces from the more challenging vision-based data sets.

The new approach presented in this paper constructs a 3D implicit function from vision-based range data. We use an analytical implicit representation that can smoothly interpolate the surface where there is little or no data that is compact when compared to discrete volumetric distance functions and that can either approximate or interpolate the data. The resulting surfaces are inherently manifold, smooth, and seamless. Implicit surfaces are well-suited for operations such as collision detection, morphing, blending, and modeling with constructive solid geometry because they are formulated as a single analytical function, as opposed to a piecewise representation such as a polygonal model or a dense volumetric data set. Implicit surfaces can also accurately model soft and organic objects and can easily be converted to a polygonal model by iso-surface extraction.

We construct an implicit surface using volumetric regularization. This approach is based on the variational implicit surfaces of Turk and O'Brien [48]. Our implicit function consists of a sum of weighted radial basis functions that are placed at surface and exterior constraint points defined by the data set. The weights of the basis functions are determined by solving a linear system of equations. We can approximate the data set by relaxing the linear system through volumetric regularization. The ability to choose whether to approximate or interpolate the data is especially advantageous in the presence of noise. Surface detail and smoothness are obtained by using basis functions that achieve multiple orders of smoothness.

Our main contributions are: 1) introducing the use of variational implicit surfaces for surface reconstruction from vision-based range data, 2) the application of a new radial basis function that achieves multiple orders of smoothness, 3) enhancement of fine detail and sharp features that are

- *H.Q. Dinh and G. Turk are with the Graphics, Visualization, and Usability Center, College of Computing Georgia Institute of Technology, Atlanta, GA 30332-0280. E-mail: {quynh,turk}@cc.gatech.edu.*
- *G. Slabaugh is with the Center for Signal and Image Processing, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30318. E-mail: slabaugh@ece.gatech.edu.*

Manuscript received 31 Jan. 2001; revised 18 Sept. 2001; accepted 28 Nov. 2001. Recommended for acceptance by E. Hancock.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 113557.

TABLE 1
Comparison of Related Works

Methods	Shape Representation	Arbitrary Topology	Complex Models	Robust to Noise	Fills Gaps
Distance Fields	discrete	yes	yes	no	no
Region Growing	piecewise continuous	yes	yes	no	no
Computational Geometry	piecewise continuous	yes	yes	no	no
Algebraic Methods	analytical	yes	no	yes	yes
Deformable Superquadrics	analytical	no	no	yes	yes
Volumetric Regularization	analytical	yes	yes	yes	yes

often smoothed-over by the variational implicit surfaces, and 4) construction of approximating, rather than interpolating surfaces to overcome noisy data.

The remainder of the paper is organized as follows: In Section 2, we review related work in surface representation and reconstruction. We give an overview of our approach in Section 3. In Section 4, we introduce volumetric regularization and describe our approach to constructing approximating surfaces using the variational implicit surface representation. In Section 5, we introduce a radial basis function that achieves multiple orders of smoothness. In Section 6, we discuss sampling issues and the preservation of topology in our framework. Results from synthetic range images and from real space carved data sets are shown in Section 7.

2 Related Work

Our approach to surface reconstruction can be compared to previous works in the areas of shape representation, reconstruction, smoothing, and surface regularization. The large number of published methods in these areas makes it nearly impossible to perform a comprehensive survey. Instead, we describe some of the more well-known approaches, with a bias towards those more closely related to our own approach. Table 1 summarizes the comparison between related reconstruction algorithms and our own.

2.1 Surface Representation

Three general classes of surface representations include discrete, parametric, and implicit approaches. Discrete forms, such as a collection of polygons and point samples, are the most widely used representations. The primary disadvantages associated with them are that they are verbose, that they can only approximate smooth surfaces, and that they have fixed resolution. In contrast, parametric surfaces, such as B-splines and Bezier patches, may be sampled at arbitrary resolution and can be used to represent smooth surfaces. The main drawback of paramteric surfaces is that several parametric patches need to be combined to form a closed surface, resulting in seams between the patches. Implicit representations, on the other hand, do not require seams to represent a closed surface. Implicit representations come in both analytical and discrete sampled forms. Analytical representations, such as our own, are more compact than sampled representations. Examples of sampled implicit functions include gridded volumes and octree representations such as those used by Szeliski and Lavallee [39], Frisken et al. [18], and Curless and Levoy [12].

2.2 Surface Reconstruction

In this section, we discuss the more popular reconstruction algorithms. The shape reconstruction methods we describe include range data merging and mesh reconstruction, region growing, algorithms based on computational geometry, and algebraic fitting.

Although our work does not focus on reconstructing surfaces from dense and precise range data, methods that merge multiple range images and reconstruct smooth meshes address issues similar to our own. Issues that arise in such work include merging multiple range images, closing of gaps in the reconstruction, and handling of outliers. Curless and Levoy [12] and Hilton et al. [20] construct signed distance functions from the range images and obtain a manifold surface by iso-surface extraction. Soucy and Laurendeau [37] and Turk and Levoy [47] merge triangulations of the range points. Note that all of these methods require range data using structured light that is much more accurate than can be measured passively using photographs alone.

Another approach is region growing and examples include Hoppe et al.'s work on surface reconstruction [21] and Lee and Medioni's work [26] and Tang and Medioni's work [40] on tensor voting. Hoppe uses a plane that is fitted to a neighborhood around each data point, providing an estimate of the surface normal for the point. The surface normals are propagated using a minimal spanning tree and then a signed distance function is contructed in small neighborhoods around the data points. Lee and Medioni's tensor voting method is similar in that neighboring points are used to estimate the orientations of data points. The tensor is the covariance matrix of the normal vectors of a neighborhood of points. Each data point votes for the orientation of other points in its neighborhood using its tensor field. In [40], the surface is reconstructed by growing planar, edge, and point features until they encounter neighboring features. Both methods described above are sensitive to noise in the data because they rely on good estimates for the normal vector at each data point.

Several algorithms based on computational geometry construct a collection of simplexes that form the shape or surface from a set of unorganized points. These methods exactly interpolate the data—the vertices of the simplexes consist of the given data points. A consequence of this is that noise and aliasing in the data become embedded in the reconstructed surface. Of such methods, three of the most successful are Alpha Shapes [15], the Crust algorithm [1], and the Ball-Pivoting algorithm [4]. In Alpha shapes, the shape is carved out by removing simplexes of the Delaunay

triangulation of the point set. A simplex is removed if its circumscribing sphere is larger than the alpha ball. In the Crust algorithm, Delaunay triangulation is performed on the original set of points along with Voronoi vertices that approximate the medial axis of the shape. The resulting triangulation distinguishes triangles that are part of the object surface from those that are on the interior because interior triangles have a Voronoi vertex as one of their vertices. Both the Alpha Shapes and Crust algorithms need no other information than the locations of the data points and perform well on dense and precise data sets. The object model that these approaches generate, however, consists of simplexes that occur close to the surface. The collection of simplexes is not a manifold surface and extraction of such a surface is a nontrivial post-processing task. The Ball-Pivoting algorithm is a related method that avoids nonmanifold constructions by growing a mesh from an initial seed triangle that is correctly oriented. Starting with the seed triangle, a ball of specified radius is pivoted across edges of each triangle bounding the growing mesh. If the pivoted ball hits vertices that are not yet part of the mesh, a new triangle is instantiated and added to the growing mesh. In Fig. 1b, the Crust algorithm is applied to real range data obtained from the generalized voxel coloring method of [11]. Although the general shape of the toy dinosaur is recognizable, the surface is rough due to the noisy nature of the real range data.

Many algebraic methods avoid creating noisy surfaces by fitting a smooth function to the data points and by not requiring that the function pass through all data points. The reconstructed surface may consist of a single global function or many functions that are pieced together. Examples of reconstruction by global algebraic fitting are the works of Taubin [41], [42], Keren and Gotsman [22], [23], and Blane et al. [5]. Taubin fits a polynomial implicit function to a point set by minimizing the distance between the point set and the implicit surface. In [41], Taubin develops a first order approximation of the Euclidean distance and improves the approximation in [42]. Gotsman and Keren create parameterized families of polynomials that satisfy desirable properties, such as fitness to the data or continuity preservation. Such a family must be large so that it can include as many functions as possible. This technique leads to an over-representation of the subset, in that the resulting polynomial will often have more coefficients for which to solve than the simpler polynomials included in the subset, thus requiring additional computation. Blane et al. performs polynomial fitting of points on a zero level set and (for stability) fits points on two additional level sets close to the zero level set—one internal and one external level set. The primary limitation of global algebraic methods is their inability to reconstruct complex models. The highest degree polynomials that have been demonstrated are around degree 12 and this is far too small to represent complex shapes.

In [3], Bajaj et al. overcomes the complexity limitation by constructing piecewise polynomial patches (called A-patches) that combine to form one surface. Bajaj et al. use Delaunay triangulation to divide the point set into groups delineated by tetrahedrons. An A-patch is formed by fitting a Bernstein polynomial to the data points within each tetrahedron. By constructing a piecewise surface, Bajaj et al.'s approach loses the compact characteristic of a global representation and operations such as collision detection, morphing, blending, and modeling with constructive solid geometry become more difficult to perform since the representation is no longer a single analytical function.

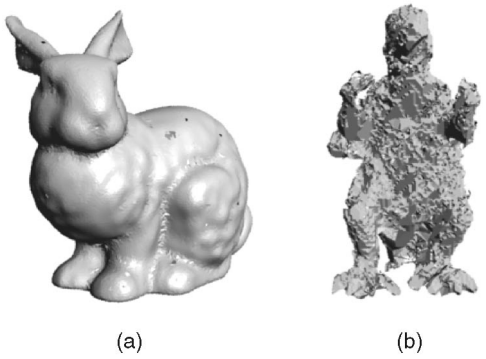


Fig. 1. (a) Stanford Bunny data set from cyberware scanner. (b) The toy dinosaur data set from voxel coloring. Both reconstructions were generated using the Crust algorithm. The dinosaur data set obtained from passive scanning is noisier and lower in resolution.

Examples of algebraic methods developed earlier in the vision community that provide both smooth global fitting and accurate local refinement include the works of Terzopoulos and Metaxas on deformable superquadrics [46] and Pentland [32], and Sclaroff and Pentland [34] on generalized implicit functions. Both methods use superquadric ellipsoids as the global shape and add local deformations to fit the data points. Terzopoulos and Metaxas separate the reconstructed model into global parameters defined by the superquadric coefficients, and local displacements defined as a linear combination of basis functions. The global and local deformation parameters are solved using dynamics. Pentland and Sclaroff define a generalized implicit model that consists of a superquadric ellipsoid and a modal deformation matrix. The modal deformation parameters are found by iteratively finding the minimum RMS error to the data points. The residual error after the deformation parameters have been found are incorporated into a displacement map to better fit the data. As with most algebraic methods, the drawback of these techniques is their inability to handle arbitrary topology.

Our approach is similar to global algebraic fitting in that we construct one global implicit function, although our basis functions are not polynomials. Previous work that is most closely related to our own are methods based on *regularization* which we describe next.

2.3 Surface Regularization

Surface reconstruction is an ill-posed inverse problem because there are infinitely many surfaces which may pass through a given set of points. *Surface regularization* restricts the class of permissible surfaces to those which minimize a given energy functional. Terzopoulos pioneered finite-differencing techniques to compute approximate derivatives used in minimizing the thin-plate energy functional of a height-field. He developed computational molecules from the discrete formulations of the partial derivatives and uses a multiresolution method to solve for the surface. Boult and Kender compare classes of permissible functions and discuss the use of basis functions to minimize the energy functional associated with each class. Using synthetic data, they show examples of overshooting surfaces that are often

encountered in surface regularization. As exemplified by these two methods, many approaches based on surface regularization are restricted to height fields.

In [16], Fang and Gossard reconstruct piecewise continuous parametric curves. The advantage of parametric curves and surfaces over height-fields is the ability to represent closed curves and surfaces. Each curve in their piecewise reconstruction minimizes a combination of first, second, and third order energies. Unlike previous examples, the derivative of the curve in this method is evaluated with respect to the parametric variable. Each curve is formulated as a sum of weighted basis functions. Fang and Gossard show examples using Hermite basis. The approach we present in this paper has similar elements. We also use basis functions to reconstruct a closed surface which minimizes a combination of first, second, and third order energies. We differ from the previous work in that we reconstruct complex 3D objects using a single implicit function; we perform volumetric rather than surface regularization; and we use energy-minimizing basis functions as primitives.

Because our method of reconstruction applies regularization, comparisons can also be made to other classes of stabilizers (or priors) and other energy-minimizing basis functions. We postpone the discussion of other prior assumptions and resulting basis functions to Section 5 where we introduce the multiorder basis function that we use to reconstruct implicit surfaces. The use of radial basis functions for graphical modeling was introduced by Blinn [6]. Since then, methods have been published that use this surface representation for surface reconstruction, including Muraki [29] and Savchenko et al. [33]. Our work differs from these methods in that we use a basis function that minimizes multiple energies in 3D, including thin-plate and membrane. Comparison with reconstructions using Gaussian and thin-plate basis functions will be addressed in Section 5.1.

2.4 Surface Smoothing

A closely related topic is that of mesh smoothing, where a low-pass filter is applied to a mesh to reduce noise. Examples of this method include the works of Taubin [43] and Desbrun et al. [13]. The primary drawback of mesh smoothing methods is that they require an initial mesh. Our approach creates and smoothes a surface in one step.

Regularization and smoothing are closely tied. The relationship between regularization and smoothing has been studied by many, including Girosi et al. [19], Terzopoulos [44], and Nielson et al. [30]. In Section 5.1, we use a volumetric data set to demonstrate the similarity between regularization and spatial smoothing. Our reconstruction of the data set (which uses no information about the gridded structure of the volume) comes very close to a model obtained by spatially smoothing the 3D data set prior to iso-surface extraction. The advantage of our reconstruction algorithm is that it may be applied to data sets that are unstructured and nonuniform. Spatial smoothing cannot easily be applied to such data.

2.5 Active versus Passive Scanning Technology

Many of the methods described above reconstruct surfaces from dense and precise data obtained from active scanning. In this paper, we address the problem of reconstructing smooth and seamless surfaces using data obtained from passive scanning. In passive scanning, only images and camera calibration information are used to obtain 3D point sets. Active scanning technology (e.g., light stripe and time-of-flight range scanners) differ from passive scanning technology (e.g., shape from shading, voxel coloring) in terms of quality, accuracy, and cost. The typical scanning resolution of cyberware scanners is 0.5 mm, while that of the voxel coloring data sets we use as examples in this paper are approximately 1.25 mm. Data from passive scanning is comparatively more noisy, more nonuniform, and more sparse than data from active scanners. In particular, surface reconstruction methods such as [12], [20], [47], [37] are not suited for creating models from data captured using passive scanning techniques.

Fig. 1 is a comparison between data sets obtained from laser scanners and that obtained from voxel coloring. Both data sets were reconstructed using the Crust algorithm of Amenta et al. which exactly interpolates all data points. The toy dinosaur data set obtained from voxel coloring is significantly lower in resolution and accuracy than the Stanford Bunny obtained using a cyberware scanner. The primary advantage of passive scanning methods is the low cost of digital cameras (less than $1,000) that are used to capture the images. Camera calibration is obtained using a calibration grid that is captured in the images. In contrast, the current cost of active range scanners is from $10,000 to over $100,000.

3 Overview of the Approach

Our approach to surface reconstruction is based on creating a single implicit function $f(\mathbf{x})$ by summing together a collection of weighted radial basis functions. We adopt the convention that the implicit function is positive inside the surface, zero on the surface, and negative outside the surface. The nature of the radial basis functions that are used is important to the quality of the reconstructions, and we discuss the basis function selection in detail in Section 5. As input to implicit function creation, our method requires a collection of constraint points $\mathbf{c}_i$ that specify where the function should take on particular values. Most of the constraint points come directly from the input data and these are points where the implicit function should take on the value zero. We call these 3D locations *surface constraints*. In addition, our method requires that some 3D points be explicitly identified as being outside the surface, and we call these *exterior constraints*. Scattered data approximation of the surface and exterior constraints is then used to construct the implicit function. In Section 4.2, we describe the details of the implicit formulation and, in Section 6, we discuss the sampling of surface, and exterior constraints from the measured data of an object.

4 Volumetric Regularization

The surface reconstruction technique that we present in this paper is an extension of the variational implicit surfaces of [48]. This approach is based on the calculus of variation and is similar to surface regularization in that it minimizes an energy functional to obtain the desired surface. Unlike surface regularization, however, the energy functional is defined in R^3 rather than R^2. Hence, the functional does **not** act on the space of surfaces, but rather, on the space of 3D functions. We call this *volumetric regularization*. We use volumetric regularization to obtain a smooth 3D implicit function whose zero level set is our reconstructed surface. By Sard's theorem [8], [17], the set of nonregular values of

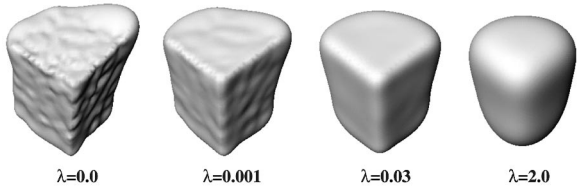


Fig. 2. Reconstructions of a synthetic range image of a cube corner using various values of λ.

such a smooth implicit function is a null set. Hence, the surface described by the zero level set of our implicit function does not contain pathological, or nondifferentiable, points. In this section, we describe how we construct an approximating surface and obtain the implicit function representing the surface using volumetric regularization.

4.1 Approximation versus Interpolation

Scattered data interpolation is the process of estimating previously unknown data values using neighboring data values that are known. In the case of surface reconstruction, the surface passes exactly through the known data points and is interpolated between the data points. Data interpolation is appropriate when the data values are precise. In vision-based data, however, there is some uncertainty in the validity of the data points. Using data interpolation to construct the surface is no longer ideal because the surface may not actually pass exactly through the given data points. This is precisely the problem with algorithms from computational geometry that generate polygonal meshes using data points as the vertices of the mesh. If the uncertainty of the data points is known, a surface that better represents the data would pass close to the data points rather than through them. Constructing such a surface is known as *data approximation*. Many vision-based techniques for capturing 3D surface points have an associated error distribution for the data points. In this section, we discuss how data approximation is achieved in our framework using volumetric regularization.

In regularization, the unknown function is found by minimizing a cost functional, H, of the following form:

$$H[f] = \beta[f] + \frac{1}{\lambda}\sum_{i=1}^{n}(y_i - f(\mathbf{x}_i))^2. \tag{1}$$

In the above equation, f is the unknown implicit surface function; $\beta[f]$ is the smoothness functional, such as thin-plate; n is the number of constraints, or observed data points; y_i are the observed values of the data points at locations $\mathbf{x}_i$; and λ is a parameter (often called the *regularization parameter*) to weigh between fitness to the data points and smoothness of the surface. We can allow the surface to pass close to, but not necessarily through, the known data points by setting $\lambda > 0$. When $\lambda = 0$, the function interpolates the data points. The λ values may be assigned according to the noise distribution of the data acquisition technique. Figs. 2 and 3 show the results of applying different λ values on synthetic and real data sets. As λ approaches zero, the surface becomes rougher because it is constrained to pass closer to the data points. At $\lambda = 0$, the surface interpolates the data, and overshoots are much more evident. At larger values of λ, the reconstructed model is smoother and approaches an amorphous bubble.

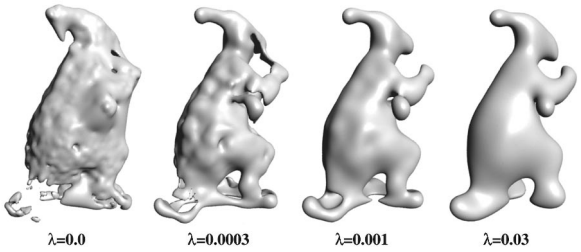


Fig. 3. Reconstructions of the toy dinosaur from real voxel coloring data using various values of λ.

4.2 A Solution to the Regularizing Cost Functional

Derivations presented in [19], [49] show that the cost functional given in (1) is minimized by a sum of weighted radial basis functions as shown below:

$$f(\mathbf{x}) = P(\mathbf{x}) + \sum_{i=1}^{n} w_i \phi(|\mathbf{x} - \mathbf{c}_i|). \tag{2}$$

In the above equation, $f(\mathbf{x})$ is an implicit function that evaluates to zero on the surface, negatively outside, and positively inside; ϕ is the radially symmetric basis function; n is the number of basis; $\mathbf{c}_i$ are the locations of the centers of the basis; and w_i are the weights for the basis. In [48], Turk and O'Brien center a basis function at each constraint point. We do the same in this work. The constraints may be points on the surface of the object to be reconstructed or points external to the object. The polynomial term, $P(\mathbf{x})$, spans the null space of the basis function. For thin-plate energy, the polynomial term consists of linear and constant terms because thin-plate energy consists of second order derivatives. In 3D, where $\mathbf{x} = (x, y, z)$, the polynomial term for thin-plate is $P(\mathbf{x}) = p_0 + p_1 x + p_2 y + p_3 z$. The unique implicit function is found by solving for the weights, w_i, of the radial basis functions and for the coefficients, p_0, p_1, p_2, and p_3, of $P(\mathbf{x})$. The unknowns are solved by constructing the following linear system formed by applying (2) to each constraint, $\mathbf{c}_i$.

$$\begin{bmatrix} \phi(r_{11})+\lambda_1 & \cdots & \phi(r_{1n}) & 1 & \mathbf{c}_1 \\ \vdots & & \vdots & 1 & \vdots \\ \phi(r_{n1}) & \cdots & \phi(r_{nn})+\lambda_n & 1 & \mathbf{c}_n \\ 1 & 1 & 1 & 0 & 0 \\ \mathbf{c}_1 & \cdots & \mathbf{c}_n & 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \\ p_0 \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} f(\mathbf{c}_1) \\ \vdots \\ f(\mathbf{c}_n) \\ 0 \\ 0 \end{bmatrix} \tag{3}$$

$$r_{ij} = |\mathbf{c}_i - \mathbf{c}_j|.$$

In the above equation, p_0 and $\mathbf{p} = (p_1, p_2, p_3)$ are coefficients of $P(\mathbf{x})$. The function value, $f(c_i)$, at each constraint point is known since we have defined the constraint points to be on the surface or external to the object. $f(\mathbf{c}_i) = 0$ for all $\mathbf{c}_i$ on the surface. All exterior constraints are placed at the same distance away from the surface constraints and are assigned a function value of -1.0 (more details will be given in Section 6.1 on selection of exterior constraints). Notice that in the above system matrix, λ appears on the diagonal. By increasing the value of λ, the system matrix becomes better conditioned because it becomes more diagonally dominant. The addition of λ does not invalidate (2) because $\lambda \sum_{i=1}^{n} w_i = 0$ (as seen in row $n+1$ of the matrix). The use of λ for trading off interpolation and approximation is found in numerous other publications, including those of Girosi et al. [19], Yuille and

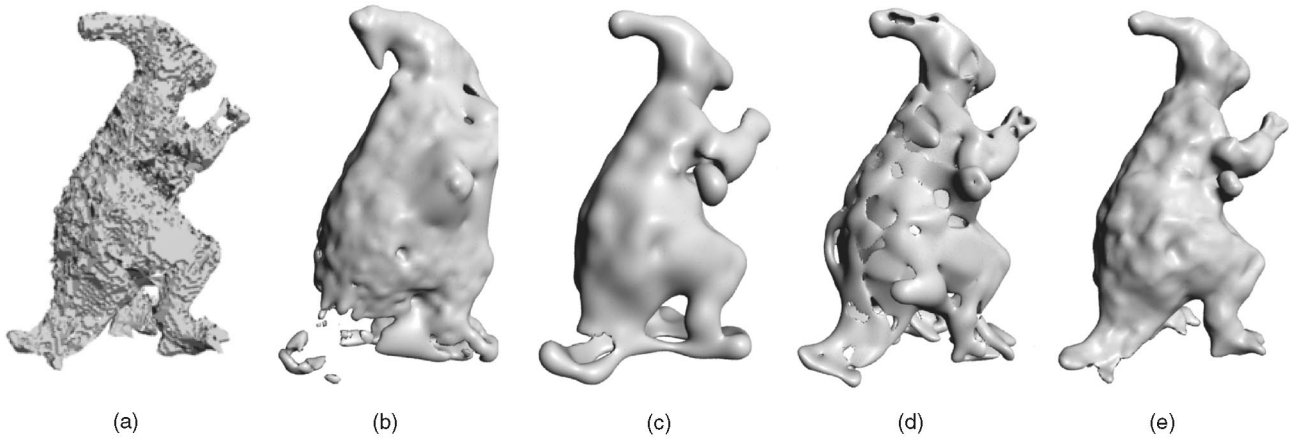


Fig. 4. Reconstructions of the toy dinosaur. (a) Crust reconstruction. (b) Exact interpolation using thin-plate basis function. (c) Surface approximation using thin-plate basis function. (d) Surface approximation using Gaussian basis function. (e) Surface approximation using multiorder basis function.

Grzywacz [51], and Wahba [49] where a detailed derivation can be found.

It is possible to assign distinct λ values to individual constraints. In this case, $\sum_{i=1}^{n} \lambda_i w_i \neq 0$, but instead, becomes part of the constant in the null space term, $P(\mathbf{x})$. This flexibility is especially important when we use exterior constraints because they are added only to provide orientation to the surface but do not represent real data. In practice, we have found that λ works well as a semi-global regularizing parameter, where one λ value is used for all surface constraints, and another for all exterior constraints. Using one λ value for all surface constraints is appropriate when the spatial distribution of noise is isotropic. This is a reasonable model for many vision-based data sets including the voxel-coloring data set that we later use as examples. With other noise models, it may be more appropriate to use λ as a local fitting parameter by assigning a λ value for each surface constraint based on the confidence measurement of the point. A large λ value such as 2.0 is often used for exterior constraints, while small values such as 0.001 is often used for surface constraints. This choice of λ for surface constraints was found through measures of fitness and curvature applied to the voxel coloring data set of a toy dinosaur. We found that a practical upper bound for λ for surface constraints from these types of data sets is 0.003. A detailed description of the fitness measures and results for various values of λ can be found in our technical report [14].

The implicit formulation described by (2) has been used in a number of previous work, including [6], [9], [28], [29], [31], [33], [48], [50], [51]. In [6], [29], [51], the basis function, ϕ, was a Gaussian, while in [9], [31], [33], [48], [50], ϕ inherently minimized thin-plate energy. In [6], [29], the basis functions were not centered at surface data points and regularization was not applied to obtain the weights for the implicit function. Instead, Muraki iteratively added Gaussian basis functions until a sufficiently close fit is obtained. In [28], [48], [50], reconstructions were performed on accurate, dense cyberware scanned data. Hence, regularization was not necessary and simply using basis functions which minimize a desired energy was sufficient. In the next section, we compare the various choices of ϕ and discuss our selection of a basis function that minimizes multiple orders of energy.

Fig. 4 is a comparison of reconstructions of a toy dinosaur. The Crust algorithm was used to reconstruct the surface shown in Fig. 4a which exactly interpolates all 20, 120 data points, thin-plate basis functions were used to construct the interpolating implicit surface shown in Fig. 4b, and in Fig. 4c, thin-plate basis functions were used to construct the approximating implicit surface with λ set to 0.001. Only 3,000 surface and 264 exterior constraints were used to reconstruct the implicit models. The approximating thin-plate surface is much smoother than either of the other two surfaces. The overshoots are less apparent and there are fewer protruding bumps and fewer small pockets embedded in the surface. Unfortunately, the toy dinosaur's features are blobby and amorphous, especially at the feet and hands. Distinct limbs, such as the feet and tail, are fused together. It is apparent from this result that the thin-plate basis function used by Turk and O'Brien generates models which are too blobby.

5 A Radial Basis Function for Multiple Orders of Smoothness

The results in Figs. 4a, 4b, and 4c show that a balance is needed between a tightly fitting, or *shrink-wrapped*, surface, and a smooth surface. A tightly fitting surface separates the features of the model but is prone to jagged artifacts. For example, the Crust reconstruction, shown in Fig. 4a, is an exact fit to the data with no smoothness constraint. On the other hand, a smooth surface may become too blobby as seen in Figs. 4b and 4c, which show that minimizing the thin-plate energy alone is not sufficient to produce a surface that separates features well and is locally detailed.

In [10], Chen and Suter derive radial basis functions for the family of Laplacian splines. The basis functions are comprised of $|r|^k$, $|r|^k log|r|$, exponential, and Bessel function terms, where r is the distance from the center of the radially symmetric basis. The value of k depends on the dimension and order of smoothness. Turk and O'Brien use $\phi(r) = |r|^2 log|r|$ for 2D thin-plate interpolation, and $\phi(r) = |r|^3$ for 3D thin-plate interpolation. Fig. 5a shows that these functions exhibit global influence because the value of the function tends toward infinity as the distance from its center increases. The system matrix, which consists of the evaluation of the basis function at distances between pairs

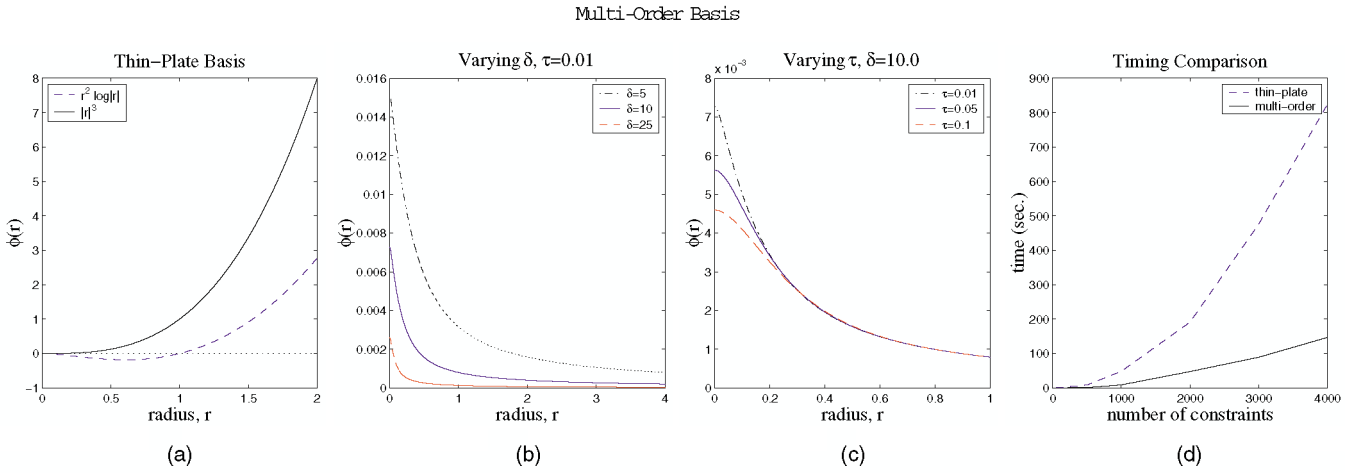


Fig. 5. (a) Cross sections of radially symmetric basis functions for $|r|^2 log|r|$ and $|r|^3$. (b) and (c) Cross sections of basis functions for a combination of first, second, and third order smoothness for various values of δ and τ. (d) Comparison of running times to solve for weights for the thin-plate and and for the multiorder basis functions.

of constraints, is dense because constraint points are uniformly spread across the region of interest.

First, second, and third order energy-minimizing splines are also members of the family of Laplacian splines. Thin-plate energy is equivalent to second order energy and membrane to first order energy. Surprisingly, a radial basis function that minimizes a combination of first, second, and third order energies quickly falls toward zero, yielding a better conditioned system matrix than one that minimizes thin-plate energy alone. In [38], Suter and Chen used basis functions that minimize multiple orders of smoothness (beyond the first and second order) to reconstruct human cardiac motion. They found that a model minimizing third and fourth order energy resulted in the smallest RMS error. They concluded that basis functions that minimize more than just the first and / or second order energy generate more accurate reconstructions. In addition, as the spacital dimension increases, the order of continuity of the thin-plate spline at data points decrease. Suter and Chen show that in 3D, the thin-plate spline basis has discontinuous first order derivatives at the data points. We chose to use a basis that achieves first, second, and third order smoothness because, unlike motion, object surfaces may contain sharp features that are C^1 discontinuous. The resulting implicit function has continuous derivatives due to the additional third order smoothness (although, the isosurface may not have continuous derivatives). The geometric analogy to minimizing third order energy is curvature continuity. It has been shown in previous work by Fang and Gossard [16] that including curvature continuity results in improved curve and surface fitting. Terzopoulos also speculates on the use of curvature continuous stabilizers in [44].

In [10], Chen and Suter derive such a basis, using a smoothness functional comprised of the first, second, and third order Laplacian operator. The associated partial differential equation is similar to Laplace's equation $-\Delta f = 0$, but also has higher order terms:

$$-\delta\Delta f + \Delta^2 f - \tau\Delta^3 f = 0. \tag{4}$$

In the above equation, the Laplacian operator Δ in 3D is:

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2}. \tag{5}$$

In (4), δ controls the amount of first order smoothness, and τ controls the amount of third order smoothness. The balance between δ and τ controls the amount of second order smoothness. The radial basis function that inherently minimizes the above energy functional in 3D as derived in [10] is:

$$\phi(r) = \frac{1}{4\pi\delta^2 r}\left(1 + \frac{we^{-\sqrt{v}r}}{v-w} - \frac{ve^{-\sqrt{w}r}}{v-w}\right)$$
$$v = \frac{1+\sqrt{1-4\tau^2\delta^2}}{2\tau^2} \qquad w = \frac{1-\sqrt{1-4\tau^2\delta^2}}{2\tau^2}. \tag{6}$$

In the above equations, r is the distance from the center of the radial basis function. The polynomial term spanning the null space of the multiorder basis function is simply a constant, $P(\mathbf{x}) = p_0$. Figs. 5b and 5c show plots of the above function for various values of δ and τ. Unlike the plot for $\phi(r) = |r|^3$, these plots show that the value of the basis function quickly falls toward zero as the distance from its center increases.

5.1 Comparison with Gaussian, Thin-Plate Radial Basis Functions, and Spatial Smoothing

The multiorder basis function described by (6) has several advantages over the thin-plate and Gaussian basis functions used by Blinn, Muraki, Yuille, and others [6], [51], [29]. The system matrix formed by the thin-plate basis function is dense, and nonzero values grow larger away from the diagonal. Computation time increases significantly as more constraints are specified. In contrast, the system matrix formed by the multiorder basis function is diagonally dominant and is especially amenable to the biconjugate gradient method of solving linear equations. Even though the matrix formed by the multiorder basis is dense, nonzero values diminish away from the diagonal. Timing results show that the unknown weights of (2) were solved in 1.5 minutes using the multiorder basis function with $\delta = 10$ and $\tau = 0.01$, while the system matrix generated for the same set of 3,264 constraints using the

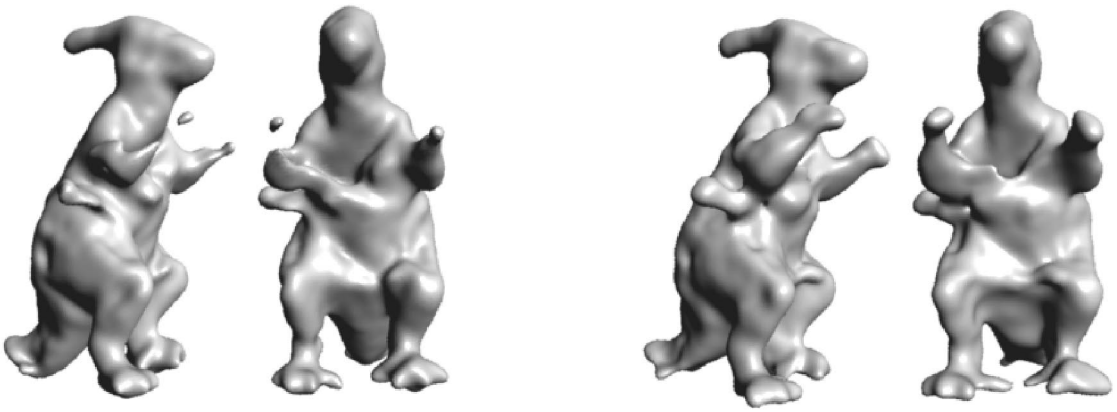


Fig. 6. (a) Iso-surface extraction of volumetric data after spatial smoothing using a Gaussian filter with a radius of four voxels. (b) Reconstruction using the multiorder basis function with 3,000 surface constraints and $\delta = 10.0$ and $\tau = 0.025$.

thin-plate basis function required 7.9 minutes to solve on an SGI Origin with 195 MHz MIPS R10000 processor. Fig. 5d is a comparison of running times versus number of constraints for the thin-plate and multiorder basis functions. The increase in running time as the number of constraints increase is fairly linear for the multiorder basis function as opposed to the thin-plate basis. The system matrix formed using Gaussian basis functions (with $\sigma = 0.01$) is sparse, requiring only 2.6 minutes to solve. The system matrix is solved even more quickly with smaller values of σ, but at the cost of worse reconstructions.

In terms of reconstruction quality, the multiorder basis function is able to reconstruct more locally detailed models while still retaining global smoothness. Both the thin-plate and the Gaussian basis functions result in models with overshooting surfaces. The thin-plate basis creates poorer reconstructions than the multiorder basis because the thin-plate basis forces the surface to be too smooth, resulting in blobby models. As an infinite mixture of Tikhonov stabilizers, the Gaussian basis also generates surfaces that are too smooth. Holes can form in the Gaussian surface when the kernel size is smaller than the interspacing between surface points. Morse et al. also mentions encountering poor reconstructions due to small Gaussian kernels [28]. Using a larger kernel size reduces the embedded holes, but increases smoothness, thereby losing surface detail. At $\sigma = 0.01$, the Gaussian surface is comparable in detail to the thin-plate and multiorder reconstructions, though there are many embedded holes. Larger σ values result in an overly smooth surface with fused limbs. Fig. 4 is a comparison of reconstructions of the toy dinosaur using the thin-plate (c), the Gaussian (d), and the multiorder (e) basis functions. Note that the round protrusion beneath the arm is the wind-up key for the toy and that the bumps on the back are the scales and spines of the actual toy dinosaur (see Fig. 11 for two of the original images).

Another difference between reconstruction using the multiorder and the thin-plate basis is in the use of nonzero interior and exterior constraints. Reconstruction using the thin-plate basis is much more dependent on the dense placement of exterior constraints to prevent the surface from overshooting into regions where the model should not exist and on the placement of interior constraints to define the orientation of the surface. In [48], Turk and O'Brien pair each surface constraint with a *normal constraint* that is interior to the surface and has a function value of 1.0. The multiorder basis does not overshoot as much as the thin-plate basis. Hence, a sparse, uniform spread of exterior constraints are enough to orient the implicit surface. We have found, in practice, that approximately one exterior constraint for every ten surface constraints is sufficient and that interior constraints are unnecessary. More details are provided in Section 6.1 on how exterior constraints are obtained.

The real voxel coloring data sets we use, described in Section 7, are embedded in a global grid structure. In such cases, it is possible to spatially smooth the data in 3D and obtain a smooth reconstruction through iso-surface extraction. Note that this is not true in the general case where the input data set may be unstructured. As it turns out, the multiorder prior we use can give reconstructions that are very similar to spatial smoothing when δ and τ are appropriately set to be smooth. Fig. 6 compares the reconstruction of the toy dinosaur using spatial smoothing and using the multiorder basis. The similarity of these reconstructions show that the multiorder basis is indeed closely related to spatial smoothing. As noted in [43], spatial smoothing tends to shrink features (such as the paws of the dinosaur), while volumentric regularization does not. An added advantage of using energy-minimizing basis functions is that it can create smooth reconstructions of unstructured and nonuniform data, to which spatially smoothing cannot easily be applied. Uniform spatial smoothing of unstructured data would require a resampling step to integrate all data points into a structured grid, as was done in [12]. In addition, the parameters, δ and τ, associated with the multiorder basis allows finer control over how much smoothing is applied. For example, in Fig. 4e, δ and τ were set to preserve the scales and spines on the back of the toy dinosaur which is lost by too much smoothing in Fig. 6.

6 Constraint Specification

As described in Section 4.2, the implicit function we reconstruct evaluates to zero on the surface, positively inside the surface, and negatively outside. The data sets we use to perform the reconstruction is from passive range scanning. Such data sets are noisy, low in resolution, and more sparse than data sets from active range scanning. We describe the data sets in more detail in Section 7. In this section, we describe the method by which we obtain surface and exterior (negative) constraints used in the reconstruction. We also

address the sampling required to guarantee that the topology of the object is correctly reconstructed and how this sampling density is mapped to the selected values for the parameters, δ and τ, controlling the amount of first and third order smoothness respectively.

6.1 Exterior Constraints

The computer vision community has developed many methods to acquire 3D positional information from photographic images taken by cameras. The goal of all these methods is to determine a collection of 3D points that lie on a given object's surface. When such a collection of points is acquired using cameras, the camera position and direction provide additional information that can be used for surface reconstruction. If a surface point is seen from a particular camera, there are no other surfaces between the camera and the point. We call the region between the camera and the surface *free space*. Other approaches to surface reconstruction make use of this information as well [12]. We can use this a priori knowledge about the object surface locations and the free space to define constraints that lie on or outside of the object, as seen in Fig. 7.

Recall, that, the exterior constraints are those locations where we want our implicit function to be negative, and the surface constraints are where the implicit function should evaluate to zero. In practice, we place exterior constraints at the same distance away from the surface constraints toward the camera viewpoints and assign them a function value of -1.0. As mentioned in Section 4.2, exterior constraints do not represent actual data, but rather, are hints to the surface orientation. Hence, a sparse sampling of exterior constraints is sufficient to properly orient the surface, and a large value of λ, such as 2.0, indicates that the negative data point should be highly approximated. We have found that one exterior constraint per ten surface constraints works well in practice. An additional sparse set (about 16 points) of exterior constraints on a bounding sphere around the object helps to constrain the surface, and alone, is often sufficient to define the surface orientation. Next, we discuss how we subsample both the exterior and surface constraints.

6.2 Subsampling Surface Constraints

Because our method of reconstruction requires the solution of a linear system, it is computationally limited in the number of constraints that can be used to construct the surface. Examples shown in this paper have used around 3,000 surface points, sampled from a set of around 20,000 surface points. Using the entire data set would not only be intractable, but would also result in an implicit function that is equal in size to the original discrete data set. In this case, the representation would no longer be compact.

The sampling density of a reduced data set must be such that the features in the data are well sampled. Since this information is not known a priori, our approach is to uniformly sample the data and then map this sampling density to appropriate δ and τ parameters. Surface points from the full data set are randomly selected. Each time a sample is selected, the neighboring samples within a small radius are eliminated from possible selection in the next round. The elimination process prevents clusters of closely placed constraint points, and resembles a 3D version of Poisson disc sampling. We have applied this method to uniformly subsample the voxel coloring data and exterior constraints previously described.

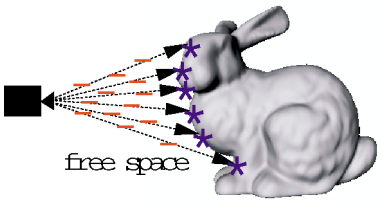


Fig. 7. Free space is carved out by rays projecting from the camera to the object surface. Surface (*) and exterior (-) constraints are defined by the free space.

Experiments show that the reduced data set is sufficient to capture the details present in the noisy data. Fig. 8 is a comparison between reconstructions from the entire data set and from a sampled subset. The full data set consists of 3,173 surface points, while the reduced set consists of 477 points. The total distance, or error, between the original 3,173 surface points and the surface reconstructed from the full data set was 0.008, while the total error between the 3,173 surface points and the surface reconstructed from the reduced set was 0.009. The model itself was constrained to be within a $2 \times 2 \times 2$ box.

Adaptively increasing the sampling in highly detailed regions is not appropriate in many vision-based data sets. Detailed regions are often synonymous with areas of high curvature and small area. In a vision-based system, these small areas map to few pixels in the acquired images, resulting in low confidence for such regions. Increasing the sampling density in these small, detailed regions would taint the reduced data set with many low confidence points.

It is possible to partition the data set, construct a separate implicit surface for each partition, and then combine the surfaces. However, the resulting representation would not be compact. We opted not to take this approach since the difference in the fitness errors between the full and the reduced data sets was minimal. Yngve and Turk [50] and Carr et al. [9] have also shown that it is unnecessary to have a basis function for each surface point. Their approach was to iteratively add basis functions until the fitness error was sufficiently low. We avoid an iterative solution by uniformly sampling the data set. One drawback of the uniform sampling approach is that noise at the scale of features cannot be removed. Some examples of this effect are shown in the toy dinosaur's chest area.

6.3 Mapping Surface Sampling Density to δ and τ Values

Recall from Section 5 that δ controls the amount of first order smoothness, while τ controls the amount of third order smoothness. The values of δ and τ that correspond to the best reconstruction of a surface is dependent on the sampling density of the surface and the desired smoothing. In our work, we maintain consistent average sampling density across all models by constraining the size of the model and by using nearly the same number of surface constraints to cover the data set. We scale all the models to lie within a $2 \times 2 \times 2$ box. By applying this normalization, the feature size, average sampling density, and choice of δ and τ are consistent across all models. This normalization is appropriate because all our input data sets have approximately the same resolution. One measure of this normalization is the average minimal

distance between sample points. We compute this distance by averaging the distances between each sample point and its closest neighboring sample point. We show later in Section 7 where we discuss the data sets in more detail that this average minimal distance is similar across all data sets after normalization and sampling.

We chose appropriate values for δ and τ by comparing models that have been reconstructed at various values of δ and τ. We have two methods of validation and comparison between the reconstructed models. These methods are a measure of fitness error and a measure of average curvature. We define fitness error to be the aggregate distance between the original data points and the reconstructed surface. To measure the average curvature of a surface, we first extract a polygonal model from the implicit function. We measure curvature at each vertex of the polygonal model using an approximation that was developed for the smoothing operator in [13]. The average curvature is obtained by dividing the aggregate curvature by the number of vertices in the polygonal model. High curvature is associated with sharp features in the surface, while low curvature is associated with overshoots and blobby surfaces.

We applied the measures of fitness and curvature to the toy dinosaur data set to guide selection of appropriate values for λ, δ, and τ. For details on the selection of these values, see our technical report [14]. We have found in practice that values of λ between 0.001 to 0.003, δ between 5.0 to 40.0 and τ between 0.005 to 0.025 can be used to produce locally detailed, yet globally smooth, reconstructions with minimal error on a variety of data sets.

6.4 Handling Outliers

Outliers are handled by a preprocessing step that finds the largest connected component in the data set. For the voxel coloring data set, we traverse the volume of surface points and group together voxels that are within the 26-neighborhood of each other. The single, largest connected component is kept and all other surface points are eliminated. If n components exist (where $n > 1$), then we can sort the components in the data set according to their size and keep only the first n largest components.

6.5 Topology Adaptation

One of the main advantages of the variational implicit surface technique is its ability to reconstruct models of arbitrary topology without explicit knowledge of the topology of the model beforehand. The resulting topology is, however, dependent on the data samples used to reconstruct the model. It is necessary to sufficiently specify surface and exterior constraints to define the topology. For example, if a torus is to be reconstructed, then at least one exterior constraint is needed near the torus hole to force the existence of the hole in the middle of the torus. As long as the surface and exterior space are uniformly sampled, the topology is correctly reconstructed. However, since we are using data from vision-based methods, view occlusion or a lack of reference views may prevent correct sampling of the space. For example, if no views of the torus showing the hole in the center are available, then the hole may not be correctly reconstructed. We argue, that in such a case, the topology of the reconstructed model is consistent with the ambiguity of the topology in the data set.

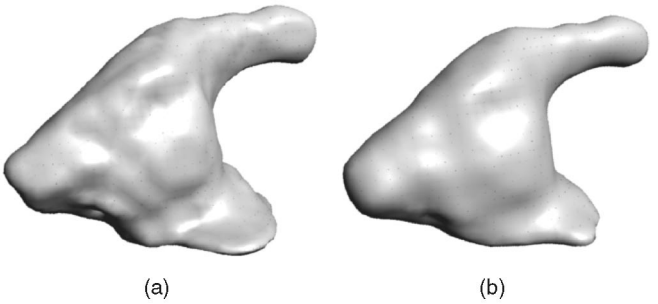


Fig. 8. Reconstructions of the head of the toy dinosaur. (a) Reconstruction using full data set (3,173 surface points). (b) Reconstruction using a subset of available data (477 surface points).

7 Results

We now show that volumetric regularization generates globally smooth, yet detailed, surfaces and discuss the addition of color to the models. We reconstructed surfaces using the multiorder basis on two types of data—synthetic range data and real voxel coloring data. Our method of reconstruction can generate smooth surfaces from data sets that are globally unstructured and noisy. Although neither types of data sets we use have both these features, each is an example of one feature. The synthetic range data is not embedded in a global grid, while the voxel coloring data is quite noisy in comparison to active range scanning data.

7.1 Synthetic Range Data

We use a modified ray-tracer [24] to generate synthetic range images as one test of our approach. We used the Stanford Bunny as our test model, and created three synthetic range images from positions separated by 120 degrees on a circle surrounding the model. For each range image, surface constraints are created by uniformly downsampling the range image to reduce the size of the data set. For every ten surface constraints, one exterior negative constraint is created within the free space described in Section 6. Additional exterior constraints are defined on a sphere surrounding the bounding box of the object at a distance farther away from the object. Fig. 9a shows the original Stanford Bunny model consisting of 69,451 triangles, while (b), (c), and (d) show the implicit surface reconstructed from 2,168 surface and 193 exterior constraints using the multiorder basis function. Figs. 9c and 9d also show the distribution of the constraints overlayed on top of the reconstruction. The average minimum distance between surface samples used in the reconstruction is 0.051. Values of $\lambda = 0.001$, $\delta = 10$, and $\tau = 0.01$ were used to reconstruct the surface. The implicit surface is quite similar to the ground truth. Our method of reconstruction produces plausible surfaces even in locations where the data is sparse. The model is closed on the top and bottom of the Bunny even though few constraint points were placed there. The model is closed at these places due to the inherently manifold nature of implicit surfaces, and it is smooth at these locations by virtue of minimizing the cost functional.

7.2 Real Volume-Carved Data

Synthetic data does not have the noisy characteristic of real data. We now describe the real space carved data that we use and how we define the surface and exterior constraints. We use three data sets of real objects obtained through voxel

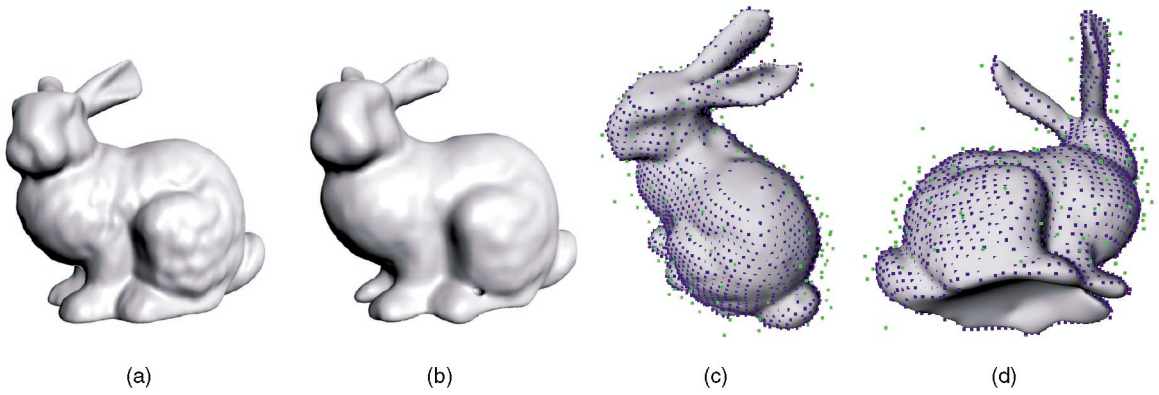


Fig. 9. Panel (a) is the original Stanford Bunny consisting of 69,451 triangles. Panels (b), (c), and (d) show the reconstructed surface using the multiorder basis reconstruction method of this paper. Panels (c) and (d) show the surface constraints (blue squares) and the exterior constraints (green squares) used in the reconstruction overlayed on top of the reconstructed surface. Note that the reconstructed surface is closed on the top and bottom even though few constraints are present.

coloring [35], [11]—a toy dinosaur (from Seitz and Dyer [35]), a broccoli stalk, and a stack of toy tori (from Slabaugh et al. [36] and referred to as the *towers* data set). Both data sets were obtained by taking about 20 images approximately on a circle around each object. Thin-shelled, voxelized surfaces were then constructed using the generalized voxel coloring algorithm [11]. The space is carved by splatting each visible voxel towards each calibrated camera and determining the consistency of the color across the images. If the variance in color intensity is below a specified threshold, the voxel is kept as part of the object surface. Otherwise, it is cast out and assigned a zero opacity value. The data consists of red, green, and blue channels. Nonempty voxels represent the presence of a surface, as deduced by the voxel coloring algorithm. Fig. 10 shows the real voxel coloring data sets.

We apply the technique described in Section 6 to obtain surface and exterior constraints for the voxel coloring data set. Nonempty voxels are surface locations. Exterior constraints are found by projecting each surface voxel in the volume to the image plane of each camera. If the ray from the surface voxel to a camera intersects other surface voxels, then the view of the voxel is blocked. Otherwise, the camera has an unobscured view, and an exterior constraint can be placed at a small distance away from the surface voxel along the ray towards the camera, as depicted in Fig. 7. Note that for each surface voxel, an exterior constraint is created for each camera that has an unobscured view of the surface voxel. Again, only a subset of the surface and exterior constraints are selected by the Poisson disc sampling technique in Section 6.2. Once a specified number of constraints have been collected, they are given to the reconstruction algorithm. In this paper, we have used from 2,000 to 3,000 surface constraints. We have found that 100 to 300 exterior constraints suffice to define the orientation of the surface. Fig. 10 shows examples of our reconstructions from space carved data. The average minimum distance between surface samples used in the reconstruction for the toy dinosaur, broccoli, and towers data sets are 0.035, 0.041, and 0.042, respectively. Note that the bumps on the back of the dinosaur are the scales and spines of the actual toy. The small protrusion near the base of the broccoli stalk is an actual leaf that has been accurately detected by the voxel coloring algorithm and has been correctly sampled and reconstructed by the method we describe in this paper. The running time for Marching Cubes [27] to extract an iso-surface is dependent on the desired resolution of the model and the number of terms (or constraints) in the implicit function. Surface extraction of the toy dinosaur at the resolution shown in Fig. 10 took 14.5 minutes.

7.3 Model Coloring

We texture the polygonal model (obtained through iso-surface extraction [27]) by reprojecting the triangles back to the original input images. Each triangle in the polygonal model is subdivided until its projected footprint in the images is subpixel in size, so that it can simply take on the color of the pixel to which it projects. In most cases, a triangle is visible in several of the original images. We combine the colors from the different images using a weighted average. The weight of each color contribution is calculated by taking the dot product between the triangle normal and the view direction of the camera that captured the particular image. Cameras with viewing directions that are nearly perpendicular to the triangle normal contribute less than those with viewing directions that are nearly parallel to the triangle normal. We use z-buffering to ensure that only cameras with an unobscured view of the triangle can contribute to the triangle color. Fig. 10 shows the final models of the toy dinosaur, broccoli, and towers from novel viewpoints after color has been applied. Fig. 11 is a comparison of two of the original input images of the toy dinosaur with rendered images of the reconstructed implicit surface from the same camera viewpoints.

7.4 Limitations of Volumetric Regularization

Surface reconstruction using volumetric regularization does not generate surfaces with boundaries. Instead, our method closes over gaps in the data set to construct a manifold surface. Open surfaces can be generated by placing limits on the iso-surface extraction.

As noted in Section 6, the features and topology of the reconstructed model is dependent on the density of the input data set. Features that are not inherent in the data will not be reconstructed. Conversely, noise that is the size of features will become embedded in the reconstruction. This limitation is common to most methods of reconstruction and smoothing.

Our method of reconstruction requires the solution of a matrix system. This requirement constrains the size of the data sets that we can reconstruct due to speed and memory

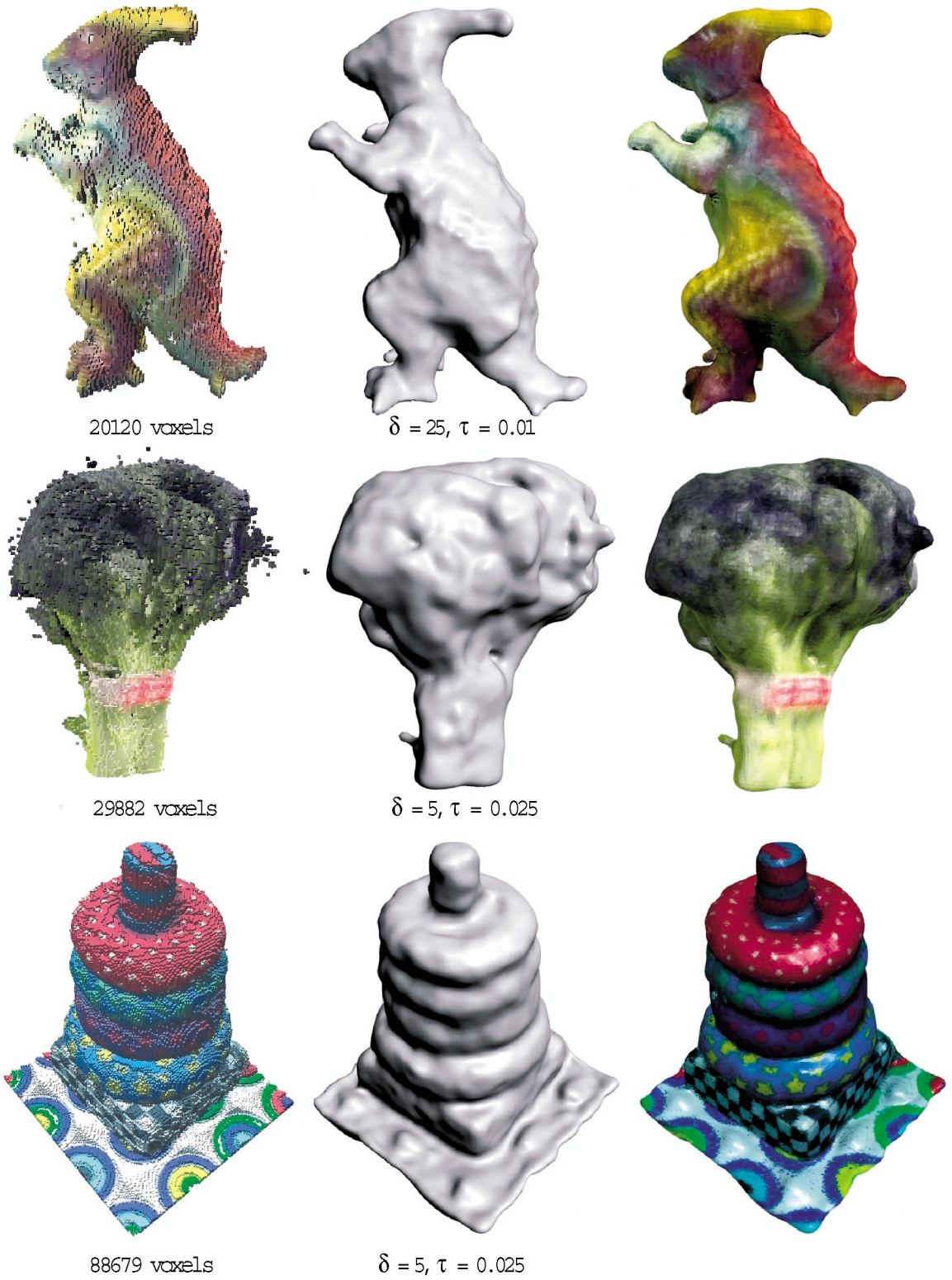
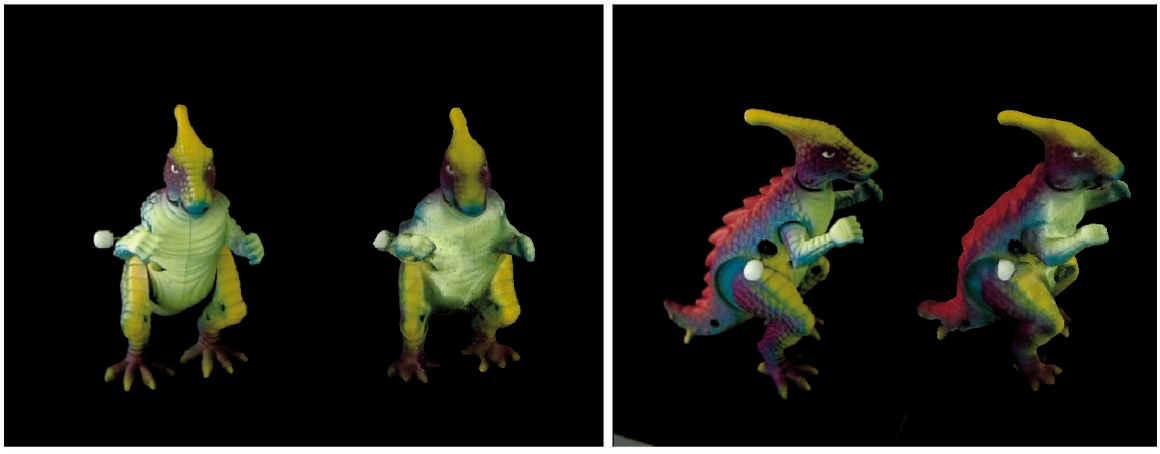


Fig. 10. From left to right: Original voxel data sets from voxel coloring, our new implicit surface reconstructions using the multiorder radial basis function, and textured versions of our reconstructions. From top to bottom: toy dinosaur, broccoli, and towers data sets. Three thousand surface constraints were used to construct the implicit surfaces.

limitations. Recently published work by Carr et al. [9] on reconstructing surfaces from dense, precise data sets using the thin-plate spline offers an efficient solution to the variational implicit method. We believe that their work using the Fast Multipole Method can also be applied here with the multi-order basis.

8 Conclusion and Future Work

The reconstruction algorithm we have presented in this paper generates models that are smooth, seamless, and manifold. Our method is able to address challenges found in real data sets, including noise, nonuniformity, low resolution, and



(a) (b)

Fig. 11. Each pair of images is a comparison of the original input image used to generate the voxel coloring data set (a) and the reconstructed implicit model rendered from the same camera viewpoint (b). A novel viewpoint of the implicit model is shown in Fig. 10.

gaps in the data set. We have compared our technique to an exact interpolation algorithm (Crust), to thin-plate and Gaussian variational implicits, and to the original volumetric reconstruction using the toy dinosaur as a running example. Obvious advantages to the models generated by volumetric regularization are that there are no discretization artifacts as found in volumetric models, and the surface is not jagged as in the Crust reconstruction. Volumetric regularization can generate approximating, rather than interpolating, surfaces and is most closely related to the thin-plate variational implicit surfaces. It compares favorably to the thin-plate variational implicit surfaces in computation time as well as in the surfaces that are generated. Using the multiorder radial basis function, volumetric regularization generates locally detailed, yet globally smooth surfaces that properly separate the features of the model.

We have adapted the variational implicit surfaces approach to real range data by developing methods to define surface and exterior constraints. Although surface points are directly supplied by the range data, we have introduced new methods for creating exterior constraints using information about the camera positions used in capturing the data. We have applied this technique to space carved volumetric data and synthetic range images.

We plan to look at several potential improvements to our approach, including use of confidence measurements and modifying the basis functions locally. For each 3D surface point obtained from the generalized voxel coloring algorithm, the regularization parameter, λ, can be assigned based on the variance of the colors to which the surface voxel projects in the input images. Another alternative is to assign different δ and τ values for the multiorder basis according to the curvature measure at constraint points. These future directions hold promise of further refining the sharp features of reconstructed surfaces of real-world objects.

Acknowledgments

The authors thank James O'Brien for stimulating discussions about alternate basis functions for surface creation. The authors also thank Marc Levoy for discussions on using implicit functions to perform surface reconstruction. The authors sincerely acknowledge Steve Seitz for providing us with the original input images of the toy dinosaur and camera calibration information, and Bruce Culbertson and Tom Malzbender of *Hewlett-Packard* for the input images and camera calibration information of the towers data set. This work was funded by the US Office of Naval Research grant #N00014-97-1-0223 and the US National Science Foundation grant #ACI-0083836.

References

[1] N. Amenta, M. Bern, and M. Kamvysselis, "A New Voronoi-Based Surface Reconstruction Algorithm," *Proc. SIGGRAPH,* pp. 415-420, Aug. 1998.

[2] C.L. Bajaj, J. Chen, and G. Xu, "Free Form Surface Design with A-Patches," *Proc. Graphics Interface,* pp. 174-181, 1994.

[3] C.L. Bajaj, F. Bernardini, and G. Xu, "Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans," *Proc. SIGGRAPH,* pp. 109-118, Aug. 1995.

[4] F. Bernardini, J. Mittleman, H. Rushmeier, and C. Silva, "The Ball-Pivoting Algorithm for Surface Reconstruction," *IEEE Trans. Visualization and Computer Graphics,* vol. 5, no. 4, pp. 349-359, Oct./Dec. 1999.

[5] M.M. Blane, Z. Lei, H. Civi, and D.B. Cooper, "The 3L Algorithm for Fitting Implicit Polynomial Curves and Surfaces to Data," *IEEE Trans. Visualization and Computer Graphics,* Mar. 2000.

[6] J. Blinn, "A Generalization of Algebraic Surface Drawing," *ACM Trans. Graphics,* vol. 1, no. 3, July 1982.

[7] T.E. Boult and J.R. Kender, "Visual Surface Reconstruction Using Sparse Depth Data," *Computer Vision and Pattern Recognition Proc.,* pp. 68-76, 1986.

[8] J.W. Bruce and P.J. Giblin, *Curves and Singularities, second ed.* Cambridge Univ. Press, pp. 59-98, 1992.

[9] J. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, and B.C. McCallum, "Reconstruction and Representation of 3D Objects with Radial Basis Functions," *Proc. SIGGRAPH,* pp. 67-76, Aug. 2001.

[10] F. Chen and D. Suter, "Multiple Order Laplacian Spline - Including Splines with Tension," Technical Report, MECSE 1996-5, Dept. of Electrical and Computer Systems Eng. Monash Univ., July 1996.

[11] W.B. Culbertson, T. Malzbender, and G.G. Slabaugh, "Generalized Voxel Coloring," *Vision Algorithms: Theory and Practice,* vol. 1883, pp. 67-74, Sept. 1999.

[12] B. Curless and M. Levoy, "A Volumetric Method for Building Complex Models from Range Images," *Proc. SIGGRAPH,* pp. 303-312, Aug. 1996.

[13] M. Desbrun, M. Meyer, P. Schroder, and A.H. Barr, "Implicit Fairing of Irregular Meshes Using Diffusion and Curvature Flow," *Proc. SIGGRAPH,* pp. 317-324, Aug. 1999.
[14] H. Dinh and G. Turk, "Reconstructing Surfaces by Volumetric Regularization," Technical Report, GVU-00-26, College of Computing, Georgia Tech., Dec. 2000.
[15] H. Edelsbrunner and E.P. Mucke, "Three-Dimensional Alpha Shapes," *ACM Trans. Graphics,* vol. 13, no. 1, pp. 43-72, Jan. 1994.
[16] L. Fang and D. Gossard, "Multidimensional Curve Fitting to Unorganized Data Points by Nonlinear Minimization," *Computer-Aided Design,* vol. 27, no. 1, pp. 48-58, Jan. 1995.
[17] A. Fomenko and T. Kunii, *Topological Modeling for Visualization.* Springer, 1997.
[18] S.F. Frisken, R.N. Perry, A.P. Rockwood, and T.R. Jones, "Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphids," *Proc. SIGGRAPH,* pp. 249-254, Aug. 2000.
[19] F. Girosi, M. Jones, and T. Poggio, "Priors, Stabilizers and Basis Functions: From Regularization to Radial, Tensor and Additive Splines," A. I. Memo No. 1430, C. B. C. L. Paper No. 75, Massachusetts Inst. of Technology Artificial Intelligence Lab, June 1993.
[20] A. Hilton, A.J. Stoddart, J. Illingworth, and T. Windeatt, "Reliable Surface Reconstruction from Multiple Range Images," *Proc. Fourth European Conf. Computer Vision,* 1996.
[21] H. Hoppe, T. DeRose, and T. Duchamp, "Surface Reconstruction from Unorganized Points," *Proc. SIGGRAPH,* pp. 71-78, July 1992.
[22] D. Keren and C. Gotsman, "Tight Fitting of Convex Polyhedral Shapes," *Int'l J. Shape Modeling,* (Special Issue on Reverse Eng. Techniques), pp. 111-126, 1998.
[23] D. Keren and C. Gotsman, "Constrained Implicit Polynomials," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 1, pp. 21-31, Jan. 1999.
[24] C. Kolb, *The Rayshade Homepage.* http://www-graphics.stanford.edu/cek/rayshade/info.html.
[25] K.N. Kutulakos and S.M. Seitz, "A Theory of Shape by Space Carving," *Int'l J. Computer Vision,* Aug. 1999.
[26] M. Lee and G. Medioni, "Inferring Segmented Surface Description from Stereo Data" *Proc. Computer Vision and Pattern Recognition Conf.,* pp. 346-352, 1998.
[27] W.E. Lorensen and H.E. Cline, "Marching Cubes," *Proc. SIGGRAPH,* pp. 163-169, July 1987.
[28] B.S. Morse, T.S. Yoo, P. Rheingans, D.T. Chen, and K.R. Subramanian, "Interpolating Implicit Surfaces from Scattered Data Using Compactly Supported Radial Basis Functions," *Proc. of Shape Modeling Int'l Conf.,* pp. 89-98, May 2001.
[29] S. Muraki, "Volumetric Shape Description of Range Data Using Blobby Model," *Proc. SIGGRAPH,* pp. 227-235, July 1991.
[30] M. Nielsen, L. Florack, and R. Deriche, "Regularization, Scale-Space, and Edge Detection Filters," *J. Math. Images and Vision,* 1997.
[31] G.M. Nielson, "Scattered Data Modeling," *Computer Graphics and Applications,* pp. 60-70, 1993.
[32] A. Pentland, "Automatic Extraction of Deformable Part Models," *Int'l J. Computer Vision,* vol. 4, pp. 107-126, 1990.
[33] V.V. Savchenko, A.A. Pasko, O.G. Okunev, and T.L. Kunii, "Function Representation of Solids Reconstructed from Scattered Surface Points and Contours," *Computer Graphics Forum,* vol. 14, no. 4, pp. 181-188, 1995.
[34] S. Sclaroff and A. Pentland, "Generalized Implicit Functions for Computer Graphics," *Proc. SIGGRAPH,* pp. 247-250, July 1991.
[35] S.M. Seitz and C.R. Dyer, "Photorealistic Scene Reconstruction by Voxel Coloring," *Int'l J. Computer Vision,* vol. 35, no. 2, pp. 151-173, 1999.
[36] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer, "A Survey of Methods for Volumetric Scene Reconstruction from Photographs," *Int'l Workshop Volume Graphics,* pp. 51-62, June 2001.
[37] M. Soucy and D. Laurendeau, "A General Surface Approach to the Integration of a Set of Range Views," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 17, no. 4, pp. 344-358, Apr. 1995.
[38] D. Suter and F. Chen, "Left Ventricular Motion Reconstruction Based on Elastic Vector Splines," *IEEE Trans. Medical Imaging,* pp. 295-305, 2000.
[39] R. Szeliski and S. Lavallee, "Matching 3D Anatomical Surfaces with NonRigid Deformations Using Octree-Splines," *Int'l J. Computer Vision,* vol. 18, no. 2, pp. 171-186, 1996.
[40] C. Tang and G. Medioni, "Inference of Integrated Surface, Curve, and Junction Descriptions From Sparse 3D Data," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 20, no. 11, Nov. 1998.
[41] G. Taubin, "Estimation of Planar Curves, Surfaces, and Nonplanar Spaces Curves Defined by Implicit Equations with Applications to Edge and Range Image Segmentation," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 13, no. 11, Nov. 1991.
[42] G. Taubin, "An Improved Algorithm for Algebraic Curve and Surface Fitting," *Proc. Fourth Int'l Conf. Computer Vision,* pp. 658-665, May 1993.
[43] G. Taubin, "A Signal Processing Approach to Fair Surface Design," *Proc. SIGGRAPH,* pp. 351-358, 1995.
[44] D. Terzopoulos, "Regularization of Inverse Visual Problems Involving Discontinuities," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 8, no. 4, July 1986.
[45] D. Terzopoulos, "The Computation of Visible Surface Representations," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 10, no. 4, July 1988.
[46] D. Terzopoulos and D. Metaxas, "Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 13, no. 7, July 1991.
[47] G. Turk and M. Levoy, "Zippered Polygon Meshes from Range Images," *Proc. SIGGRAPH,* 1994.
[48] G. Turk and J.F. O'Brien, "Shape Transformation Using Variational Implicit Functions," *Proc. SIGGRAPH,* pp. 335-342, Aug. 1999.
[49] G. Wahba, "Spline Models for Observational Data," *CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM,* pp. 11-14 and 45-65, 1990.
[50] G. Yngve and G. Turk, "Robust Creation of Implicit Surfaces from Polygonal Meshes," *ACM Trans. Visualization and Computer Graphics,* to appear.
[51] A.L. Yuille and N.M. Grzywacz, "The Motion Coherence Theory," *Proc. Int'l Conf. Computer Vision,* pp. 344-353, 1988.

Huong Quynh Dinh received the BS degree in computer engineering from the George Washington University in 1994 and the MS degree in computer science from Georgia Tech in 1999. Currently, she is a PhD candidate in computer science at the College of Computing in the Georgia Institute of Technology. She is a member of the Graphics, Visualization, and Usability Center. She expects to receive her Ph.D in summer 2002. Her research interests include surface reconstruction and representation, shape topology, and shape morphing.

Greg Turk received the PhD degree in computer science in 1992 from the University of North Carolina (UNC) at Chapel Hill. He was a postdoctoral researcher at Stanford University for two years, followed by two years as a research scientist at UNC, Chapel Hill. He is currently an assistant professor at the Georgia Institute of Technology, where he is a member of the College of Computing and the Graphics, Visualization, and Usability Center. His research interests include computer graphics, computer vision, and scientific visualization. He is a member of the IEEE.

Greg Slabaugh received the bachelor's degree in engineering physics from the University of Michigan in 1994 and a master's degree in electrical engineering from Georgia Tech in 1998. He is currently pursing a PhD and is a graduate student at the Center for Signal and Image Processing in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. His research interests include 3D surface reconstruction from multiple images, new view synthesis, and level set methods. He is a student member of the IEEE.

Computational Topology for Computer Graphics

John C. Hart
Dept. of CS
University of Illinois Urbana-Champaign

What is Topology?

The *topology* of a space is the definition of a collection of sets (called the *open* sets) that include

- the space and the empty set
- the union of any of the sets
- the finite intersection of any of the sets

If you open a book on topology, you get very obtuse definitions involving open sets and such. Topology is really much simpler than that conceptually.

No, Really. What *is* Topology?

The study of properties of a shape that do not change under *deformation*

Rules of deformation

- Onto (all of A ↔ all of B)
- 1-1 correspondence (no overlap)
- bicontinuous, (continuous both ways)
- Can't tear, join, poke/seal holes

A is *homeomorphic* to B

Say we want to deform a shape A into a shape B. The deformation is onto, which means every point of A is used and every point of B is used. The deformation is also one-to-one, which means that each point of A maps to a single point on B; that there is no overlap. The deformation is also continuous so that we can't tear A, join sections of A together, poke holes into A, or seal up holes in A. When all these are true, then we say A is homeomorphic to B, which means that they are topologically equivalent. For example, the donut is topologically equivalent to the coffee cup, because we can deform the torus into the cup. We pull out one end of the torus and form it into a cup. The hole in the torus becomes the handle of the cup.

Why Do We Care?

The study of *connectedness*

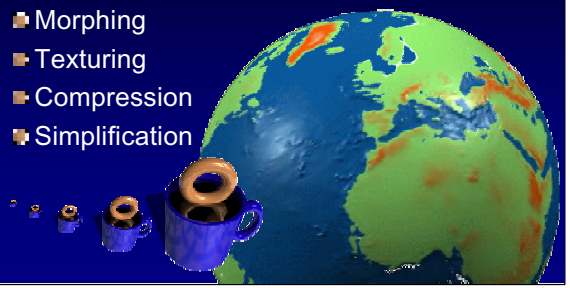
- Understanding
  - *How connectivity happens?*
- Analysis
  - *How to determine connectivity?*
- Articulation
  - *How to describe connectivity?*
- Control
  - *How to enforce connectivity?*

Topology studies the properties of a set that don't change under a homeomorphism (well-behaved deformation). These properties include completeness and compactness. However, in computer graphics, there is really only one property we are interested in and that is connectedness. Specifically how many disjoint components there are, and how many holes do they have. We are also interested in the geometric configuration of these properties, such as which components have how many holes, and where are the holes relative to each other.

For Example

How does connectedness affect…

- Morphing
- Texturing
- Compression
- Simplification



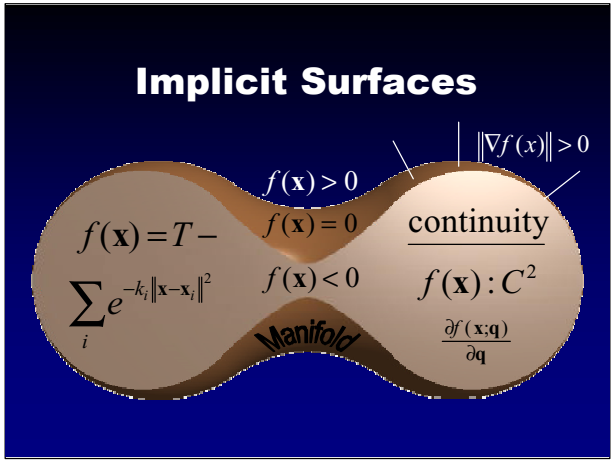
If we morph a donut into a coffee cup, we might want to make sure the donut hole becomes the handle, instead of the hole sealing up and a new hole poked through the handle. We are also interested on how to make a texture consistent when an object changes topological type, or how we can cut up an object to lay it flat and place a texture on it. When compressing the globe, Peter Schroeder at CalTech discovered that his compression technique had only small errors in geometry, but these small errors changed the geography of the European countries! Finally, when we simplify an object to describe it with fewer primitives, we want to know its topology so we can retain important properties such as holes, or remove these properties when they become imperceptible, insignificant or unnecessary.

Geometric Modeling

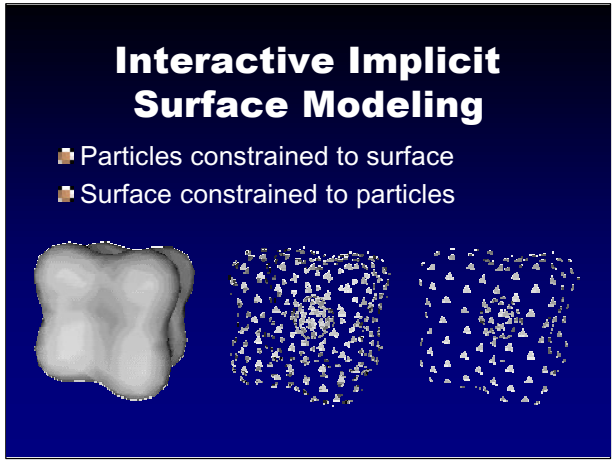
The connectedness of some geometric representations depends on the proximity of their primitives

- Recurrent models
- Implicit surfaces

We first ran into problems with topology when studying fractal models. In fact the Mandelbrot set is a map of the topology of a parameterized family of shapes (called Julia sets). But we focus on implicit surfaces as they are more mainstream.



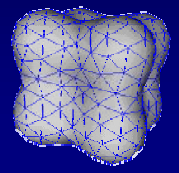
Our implicit surfaces are based on Blinn's Gaussian blobby model, where the function is negative inside. We also require the model to be continuously differentiable with respect to its parameters **q**. We assume a general position and zero is a regular point of *f*, such that the zeroset of *f* is a manifold.



We follow Witkin-Heckbert 94 in constraining particles to a surface. But we can see through the particle system. Removing particles that face away from the viewer only fixes some of the problem. This surface has a hollow region whose particles still face us. What we would like is a solid representation such as the one on the left.

Goal: Meshing Implicit Surfaces

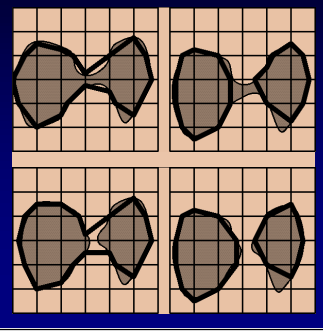
- Constrains vertices to surface
- As surface moves, so do vertices
- Reconnect skinny triangles
- Problems…



So we mesh the surface, placing vertices on the particles and connecting them up into triangles. We reconnect skinny triangles dynamically.

Problem: Mesh Aliasing

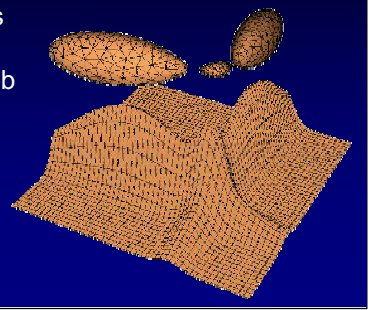
- Determines shape from point samples
- Different coordinates, different shapes



One problem with meshing an implicit surface is that the location of the samples can infer a different topology that that of the underlying surface. Note that if we translate the implicit surface one half unit up or down, the topology of the approximating mesh changes.

Problem: Interloping Components

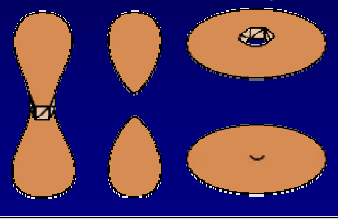
- Blob functions accumulate away from blob centers
- Incremental meshes can miss such components



When we model surfaces in real time, we use polygonal meshes around each component. If we aren't careful, a third component can arise where the tails of the Gaussians overlap and accumulate above the surface isovalue threshold. This might not be discovered until the surfaces are ray traced, such as when a film effect goes into final production!

Problem: Mesh Reconnection

- How do we reconnect polygons when an implicit surface's topology changes?
- How do we even *detect* when an implicit surface's topology changes?



If we are interactively manipulating an implicit surface, we want to be able to maintain a mesh of polygons on the surface. But when the connectivity of the surface changes, we need to reconnect its polygonization.

Solution: Morse Theory

Determines the connectedness of a space from the critical points of a real function

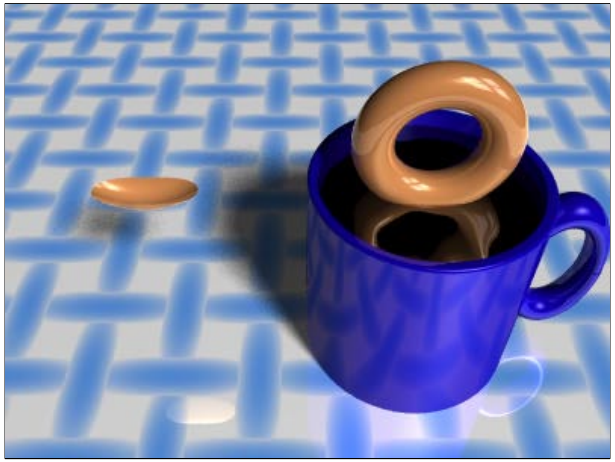
- Critical point occur where the gradient $\nabla f = (\partial f/\partial x, \partial f/\partial y, \ldots)$ vanishes
- *Index* of a critical point is # of principal directions where f decreases
- Perturb to remove degeneracy

Morse theory is many decades old. Our work is based on the book Morse Theory by John Milnor. Morse theory draws a connection between the differential properties of a surface and its topological type. It was coined by Marston Morse. It is based on critical points, where the partial derivatives of a function are zero, such that the gradient no longer has a specific direction.

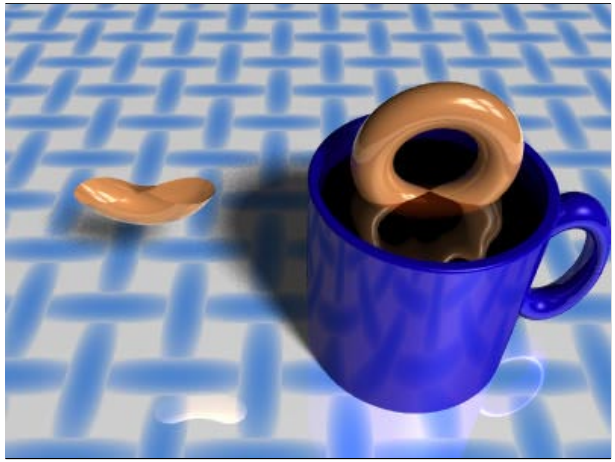
Example: Dunking a Donut

- Space is torus
- Function f is height
- Elevation gradient
- Connectedness of $f \leq h$
- Four critical points
  - Index 0 : minimum
  - Index 1 : saddle
  - Index 1 : saddle
  - Index 2 : maximum

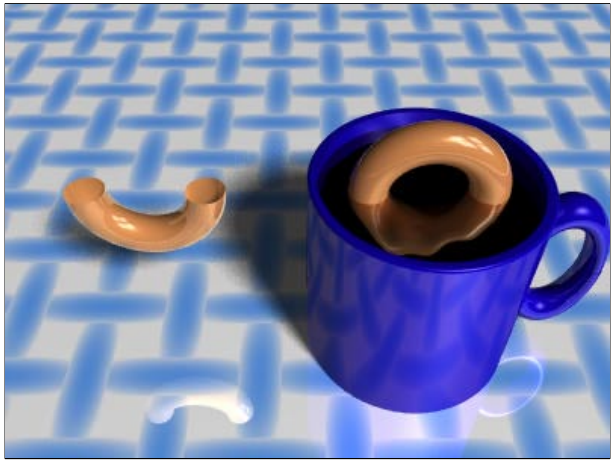
Assume we live on a torus and our function is altitude. Then the derivative of the altitude has the slope for its magnitude and the direction of steepest ascent for its direction. Notice that the four star points denote the four "level" areas on the torus, where there is no single direction of steepest ascent. We will label these points with an index. The index is the number of "principal" (maximal/minimal) directions of descent (including the opposite direction). So the point on the bottom has no direction of descent and its index is zero. The two points on the hole each have one principal direction of descent (and one principal direction of ascent), so their indices are both one. The point at the top has two principal directions of descent so its index is two.



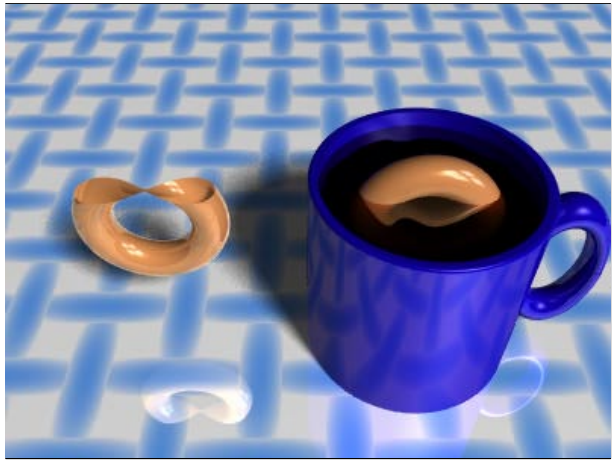
Here we have dunked the donut in the coffee. We are interested in the change in topology of the portion of the donut immersed in the coffee. In this case, we started with no donut and then the bottom critical point was immersed and the topology changed to a surface that is topologically a disk.



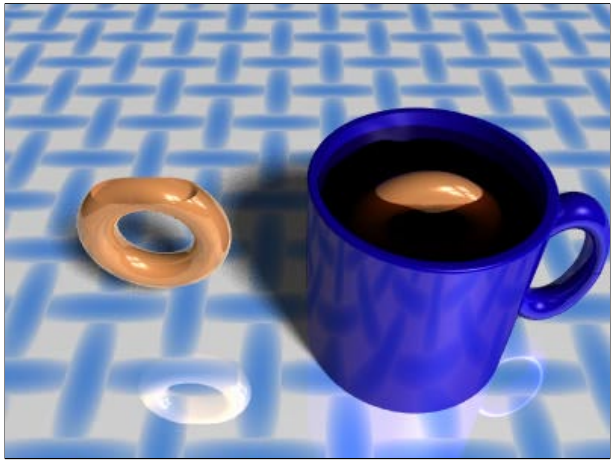
Next we pass the second critical point and the dunked portion changes topology, from a disk…



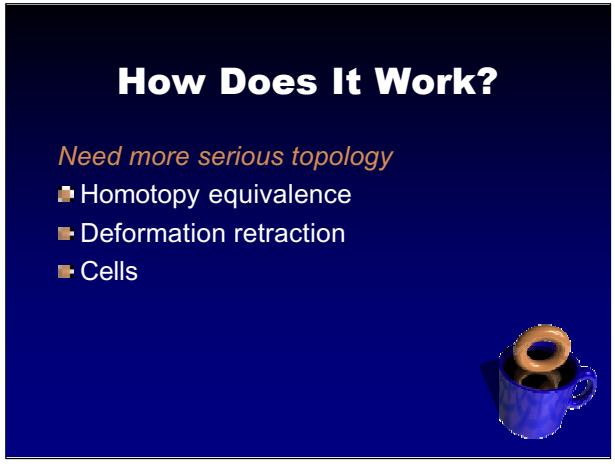
To a cylinder.



Passing the third critical point, we see that the topology is now changing from a cylinder…



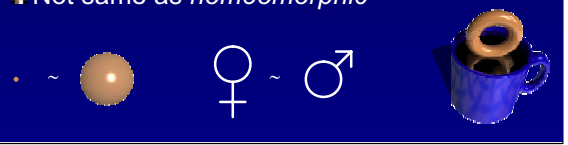
To a punctured torus (a 2-manifold with boundary of genus 2). If we let go of the donut, the entire surface is immersed. We have passed the final critical point, and the punctured torus changes its topology to that of a torus.



In order to understand how this works we need just a few more elements from topology.

Homotopy Equivalence

- A ~ B
- Same # of components
- Same # of holes
- Not necessarily same dimension
- Not same as *homeomorphic*

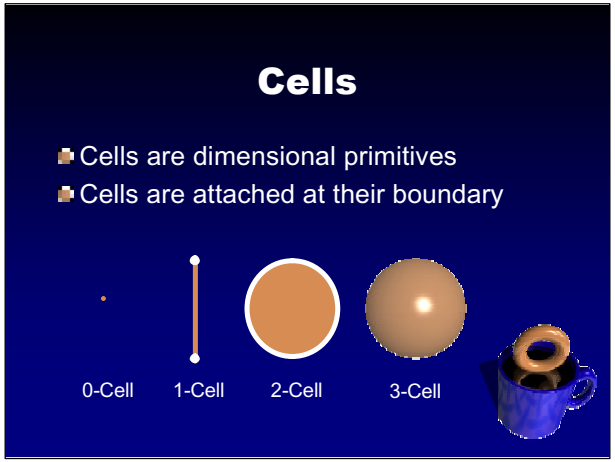


We say two shapes are homotopic when they have the same number of components and the same number of holes. This is not necessarily the same as homeomorphic because the shapes need not be the same dimension. (Dimension is a topological property and so is preserved by homeomorphism.) For example, the point is homotopic to the solid ball (one component, no hole). Also the symbol for female is homotopic to the symbol for male (one component, one hole).

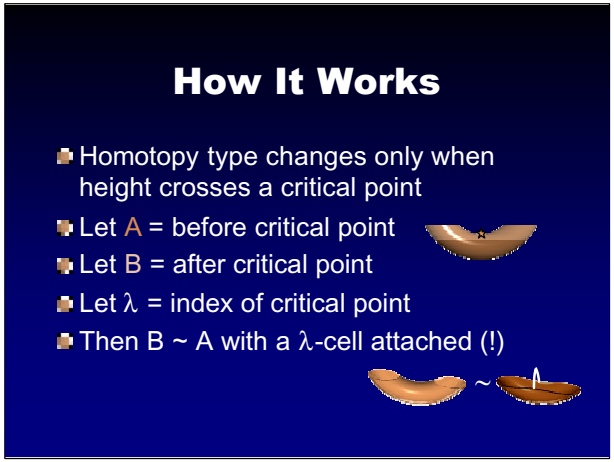
Deformation Retraction

- Function that continuously reduces a set onto a subset
- Any shape is homotopic to any of its deformation retracts
- *Skeleton* is a deformation retract of the solids it defines

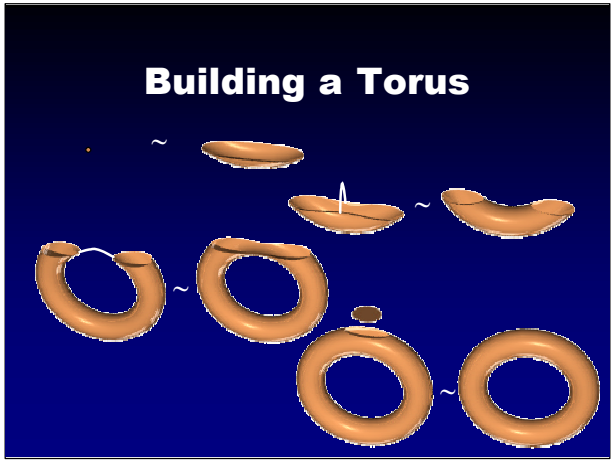
The easiest way to show homotopy equivalence is to use the deformation retract. A deformation retract is a continuous erosion of material, leaving a subset of the original shape. For example, the solid torus can be eroded into a flat anulus, and the anulus can be eroded to a circle. Notice we can't shrink the circle to a point at its center because that point is not a subset of the circle. We also can't erode the circle into a point on the circle because it would require us to break the circle, and a deformation retract is continuous. Notice we can erode the plus in the female sign to its base on the circle and we can likewise erode the plus in the male symbol. Hence all these shapes are homotopic because they can all be deformation retracted to a circle.



Cells are dimensional primitives. A 0-cell is a zero-dimensional primitive, a point. A 1-cell is a line segment, a curve, or a space curve. It has two 0-cells (points) for its boundary. A 2-cell is a disk, a filled square, or a patch. It has a 1-cell closed loop boundary. A 3-cell is a ball, a filled cube, jello. It has a sphere or other closed surface (2-cell) as its boundary.



This is relationship between critical points and homotopy type. Let A be the portion of the torus below a critical value and let B be the portion of the torus below a level just above a critical value. Note that A is a subset of B. Then B is homotopic to A with a cell attached, where the dimension of the cell is precisely the index of the critical point.

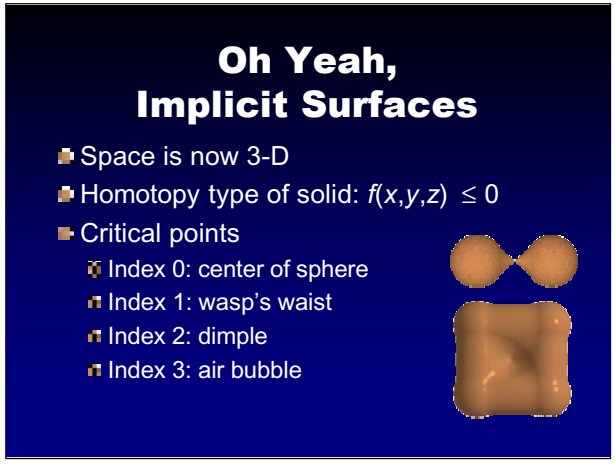


For example, when the minimum is dunked, we have “nothing” becoming the bottom of the torus. This occurs by attaching a 0-cell (point) to “nothing” yielding a point, which is a deformation retract of the bottom of the torus.

When the first saddle is dunked, we have a “disk” becoming a “cylinder.” The cylinder is homotopic to the disk with a 1-cell attached. We can erode the sides of the cylinder down, and the middle of the cylinder to the attached “Easter basket” handle.

When the second saddle is dunked, we have another 1-cell attached. We can erode the edges of the punctured torus down to the edges of the cylinder, except for the attached 1-cell handle.

When the maximum is dunked, we have a 2-cell (disk) attached to the hole in the punctured torus to complete the construction.



What does any of this have to do with implicit surfaces? For implicit surfaces, space is now 3-D and we need to deal with the homotopy type of the implicit solid bounded by the implicit surface. In 3-D we have four kinds of critical points.

An index 0 critical point happens in the center of blobs, where the function increases in all of the principal directions because we are getting closer to the edge of the implicit solid.

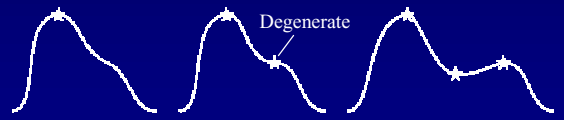
An index 1 critical point occurs when blobs kiss. The function decreases is one principal direction, as we move farther into one of the blobs, and the function decreases in the plane spanned by the other two principal directions where the function increases.

An index 2 critical point occurs at a dimple. The function decreases in the plane spanned by two principal directions whereas the function increases in the direction perpendicular.

An index 3 critical point occurs in a solid where an air bubble might occur. This is where a maxima might dip above the threshhold, creating an air pocket. All directions are decreasing.

Finding Critical Points

- Search all of 3-D space
  - Simple interval search
  - Interval Newton's method
- Tracking
  - Constrain particles to follow critical point
  - Problem: critical points appear out of nowhere

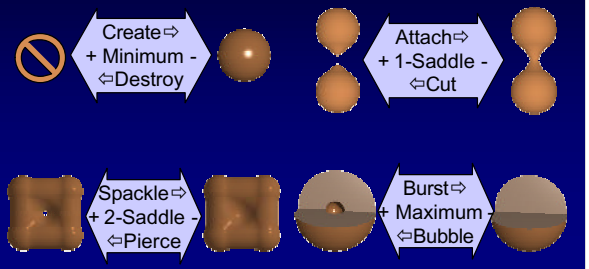


We can find all of the critical points with an interval search. This basically searches interval boxes, eliminating any where any of the partial derivatives can not be zero.

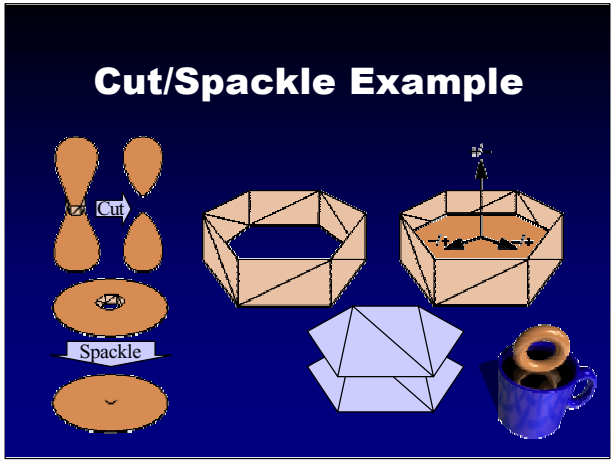
We can also constrain particles to track the critical points, by deriving the derivative of critical point position with respect to the implicit surface parameters. The problem with tracking is that critical points can appear out of this air, at degenerate critical points.

The example at the bottom shows a big blob and a small blob as a 1-D graph. In 1-D we only have two critical points: minima and maxima. On the left we have one minima. As we move the blobs apart, we see a degenerate critical point occur. This point is an inflection, and is degenerate because it is not type 0 nor type 1. Moving the blobs further shows this degenerate critical point bifurcates into a minima-maxima pair.

Implicit Surface Topology Changes



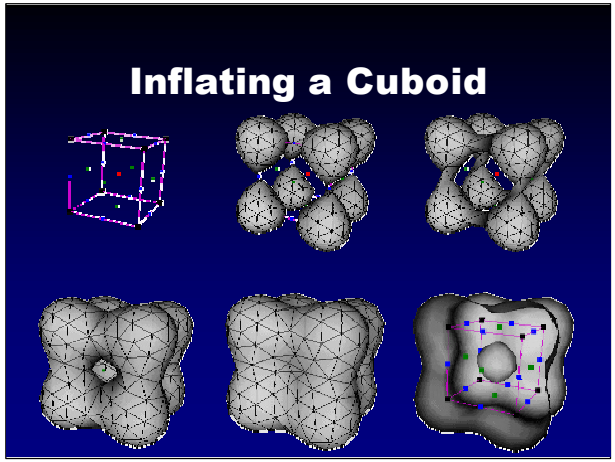
When a critical value changes sign, it signals a change in the homotopy type of the implicit solid. There are four critical points that result in eight different topology changes.



These eight solid topology changes are actually only four surface changes. For example, the cut operation and the spackle operation are identical except for which side of the implicit surface the solid interior lies.

This slide demonstrates the operations necessary to reconnect the mesh during a topology change. A cut occurs when a type one critical value goes positive, and a spackle occurs when a type 2 critical value goes negative. These "saddles" have two principal directions of similar behavior (increasing/decreasing) and a third direction whose behavior is different (decreasing/increasing). We construct a plane spanned by the two similar eigenvectors. This plane intersects a neighborhood of the mesh that forms a ring around the critical point. We delete the faces of this ring, leaving two disjoint edge rings. We sew-up these disjoint edge rings with faces, shown here in lavender.

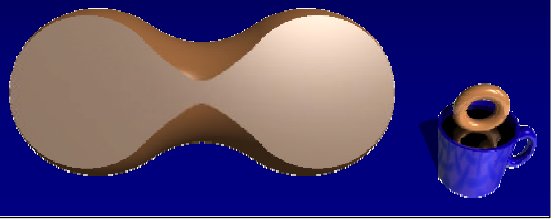
Similar algorithms are implemented for Create/Bubble, Destroy/Burst and Attach/Pierce.



We can demonstrate these operations by inflating a cube. We slowly descend the function. When the minima drop below the zero isovalue, we create eight components. When the twelve type-1 critical values fall below the zero threshold, we have attach events. When the six type-2 critical values go negative, we have spackle events. If we progressed further, the one maxima point would also go negative, resulting in a burst.

Blending

- Automatic, based on proximity
- Uncontrolled, based on proximity



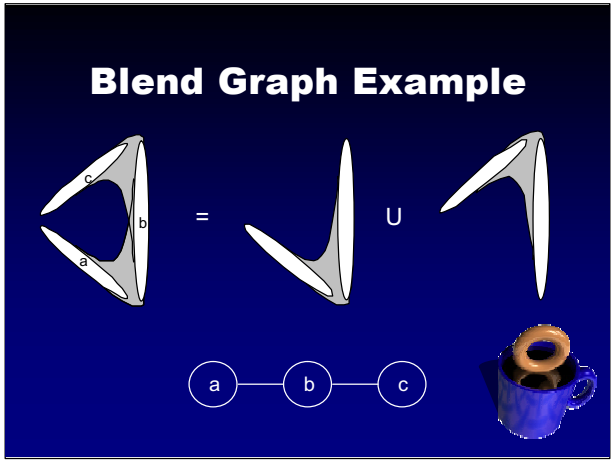
Implicit surfaces are useful because they easily support blending of shapes. All we have to do is place blobs next to each other and they automatically blend. The problem is that blobs always blend together if they are near each other, even if they aren't suppose to.

For example, it is easy to model a hand with blobs, by placing blobby fingers on a blobby palm and letting the blending automatically create a smooth join between them. However when you touch your thumb with your forefinger, the two fingertips blend together.

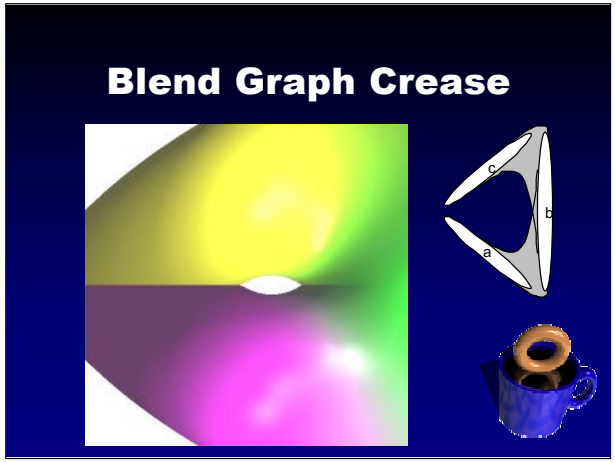
Controlled Blending

- Precise Contact Modeling
  - Cani (Gascuel) SIGGRAPH 93
  - Uses a graph to indicate blending
  - Primitives mutually distorted if not blended
- Blending Graph
  - Guy & Wyvill IS96
  - Uses a graph to indicate blending
  - Primitives unioned (CSG) if not blended
  - Results in C^1 discontinuities

There have been some attempts to prevent the blending of blobs. Precise contact modeling replaces blobs with negative blobs to prevent blending, but this causes problems with connected chains of blobs. The blending graph is a structure to indicate which blobs should blend and which shouldn't.



In the above example, blob a blends with b, and b with c, but a does not blend with c because there is no edge connecting them. This is implemented in the blob tree as the union of a blend b and b blend c.



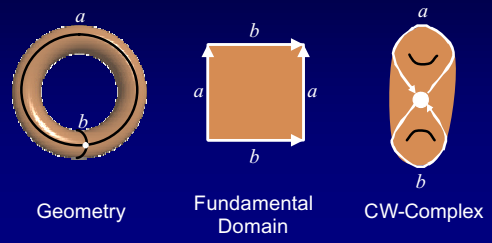
The problem with unions is that they are not C1 continuous. In this case, we have a crease in what should be a smooth blend region between a and b, and b and c.

CW-Complex

- Created by attaching cells
- Generalizes graph to higher dimensions
  - Vertices
  - Edges
  - Faces
  - Volumes
- Represents the homotopy type of the shape

We want to be able to control blending by describing the topology of the resulting shape. We use the CW-complex (or cell-complex) to represent the connectivity of the shape. A cell complex is like a graph, but also includes higher-dimensional primitives like faces and volumes. Note that a ring of edges can either contain a face or be empty, in the same way two vertices can be connected by an edge or not.

Example: Torus



Here we have three versions of the torus. We can make a torus geometrically by sweeping a circle along another circle. We can describe a torus by its fundamental domain by cutting the torus by these two circles and making the appropriate identification along the edges of the resulting square. We can also describe a torus with a cell complex consisting of one vertex, two edges and a face connecting the two edges.

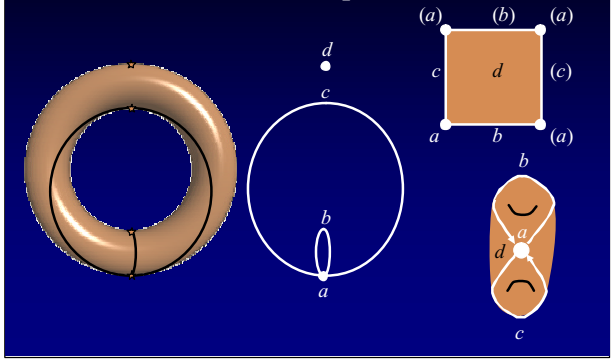
Morse CW-Complex

A surface has the homotopy type of a CW-complex with one λ-cell for each critical point of index λ

But how do we construct this CW-complex for a given shape?

There is a theorem in Morse theory that says that a shape has the homotopy type of a cell complex with one cell for each critical point, and the dimension of each cell corresponds to the index of the critical point. But we don't necessarily know how to construct the cell complex.

Follow the Separatrices



We can create a geometric embedding of the cell complex by using separatrices. Let the index 0 critical point be a zero-cell. We find the 1-cells by tracing separatrix curves from the saddles to the minimum. (Note the torus needs to be slightly tilted forward.) Finally, the index 2 critical point denotes the face connecting the 1-cell edges.

Computing the CW-Complex of a Surface

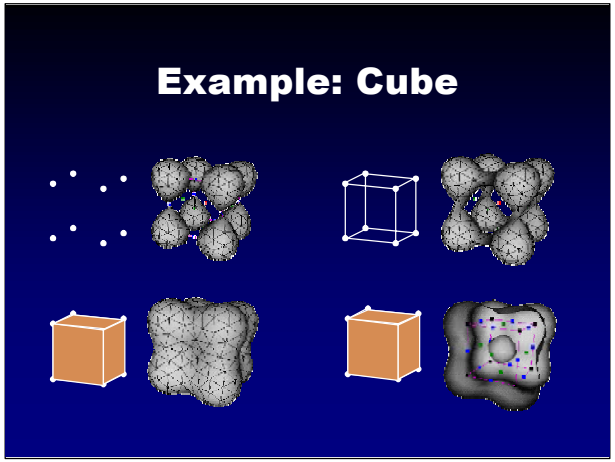
- 0-cells (vertices)
  - Surface search for all minima
- 1-cells (edges)
  - Integrate ODE $x' = \pm v_0(x)$ from saddles to minima
- 2-cells (faces)
  - Seed particle system at 2-saddles
  - Bound domain by 1-cell edges

We can hence develop an algorithm for interrogating the cell complex of an implicit surface.

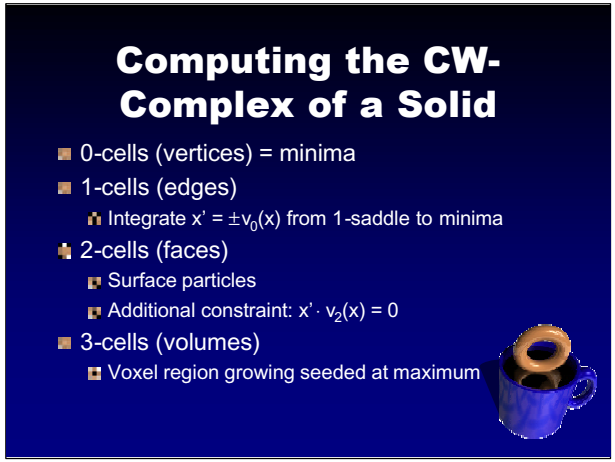
CW-Complex of a Solid

- Use cells to describe shape topology
  - Vertex : component
  - Edge : connection between components
  - Ring of edges : handle
  - Face : no hole
  - Enclosure of faces: hollow shell
  - Volume : solid
- CW-complex ~ shape

We can also use the cell complex to describe the connectedness of a solid. The resulting cell complex has the same homotopy type of the resulting solid it describes.



We can describe eight components with eight zero-cells at their centers. We can connect these components by connecting the vertices with edges. We can fill in the holes with faces in the edge rings. We can also burst the bubble by filling in the region bound by the faces.



We can hence implement algorithms to interrogate the implicit solid cell complex.

Future Work

- Topological skeleton
  - MAT=CW-complex of distance
  - Removes geometric noise
- Crystallography
  - Carroll Johnson – salt molecule
  - Critical net – graph of CW-complex
- Meshing moving interface surfaces
  - Fluid dynamics - droplets
  - Level sets – ODE surfaces

Note that the cell complex for a solid is a topological skeleton of it.

Jim Blinn stole the blobby model from molecular models, and so the Gaussian blobs are actually electon density functions. These are used to model interactions between atoms in molecules. For example, the cuboid used in the previous examples is a salt molecule. Carroll Johnson uses a 3-D graph called the critical net to similarly represent a cell complex for molecules.

We can also apply this work to maintaining stable meshes on moving interface surfaces, especially if the surface changes topology. Such surfaces occur between liquid and solid fuel in rockets, or in the dripping of substances.

The End, For Now

- Birth of a new field
- Morse theory is key
- Solves holy grail problems in implicits
  - Interactive modeling
  - Controlled blending
- Thanks: NSF, Ulrike Axen, Bart Stander, Brian Wyvill, Peter Schroeder

Computational topology is an emerging field with a wealth of new problems, solutions and applications. Morse theory is the key to understanding the relationship between geometry and topology. These techniques have already revealed solutions to nasty problems in implicit surface modeling.

This work was funded by the NSF. Ulrike Axen has applied Morse theory directly to meshes. Bart Stander coded up our implicit surface modeler that used Morse theory to maintain a correct polygonization. Brian Wyvill generated the creased blend picture and Peter Schroeder let me use his compressed atlas of the Earth.

Guaranteeing the Topology of an Implicit Surface Polygonization for Interactive Modeling

Barton T. Stander[1] John C. Hart[2]
School of EECS
Washington State University

Abstract

Morse theory shows how the topology of an implicit surface is affected by its function's critical points, whereas catastrophe theory shows how these critical points behave as the function's parameters change. Interval analysis finds the critical points, and they can also be tracked efficiently during parameter changes. Changes in the function value at these critical points cause changes in the topology. Techniques for modifying the polygonization to accommodate such changes in topology are given. These techniques are robust enough to guarantee the topology of an implicit surface polygonization, and are efficient enough to maintain this guarantee during interactive modeling. The impact of this work is a topologically-guaranteed polygonization technique, and the ability to directly and accurately manipulate polygonized implicit surfaces in real time.

Descriptors: I.3.5 Computational Geometry and Object Modeling — Modeling packages. **General Terms:** Algorithms.
Keywords: catastrophe theory, critical points, implicit surfaces, Morse theory, polygonization, topology, interval analysis, interactive modeling, particle systems.

1 Introduction

Shapes are represented implicitly by a function that classifies points in space as inside the shape, outside the shape or on the shape's surface, called the implicit surface. This representation provides computer graphics with geometric models that can be easily and smoothly joined together, but the incorporation of the implicit representation into graphics systems can be problematic. Polygonization of implicit surfaces allows graphics systems to reap the powerful modeling benefits of the implicit representation while retaining the rendering speed and flexibility of polygonal meshes.

The *topology* (specifically the *homotopy type*) of an implicit surface refers to the connectedness of the shape, including the number of disjoint components and the number of holes in each component (*genus*). This should not be confused with other instances of topology in computer graphics, such as the connection patterns of a mesh of polygons or parametric patches. Thus, for a polygonization to accurately represent the topology of an implicit surface the number of components and the genus of each component of the polygonization need to agree with that of the implicit surface.

Implicit surfaces are commonly polygonized to a given degree of geometric accuracy. This accuracy is in some cases adaptively related to the local curvature of the implicit surface, reducing the number of polygons without affecting the appearance of the surface. This work instead focuses on a polygonization that accurately discerns the topology of an implicit surface. Topologically-accurate polygonization can reduce the number of polygons without affecting the structure of the surface.

Guaranteeing the topology of a polygonization does not necessarily yield an accurate polygonization. A coffee cup is topologically equivalent to a torus, but the torus provides a poor representation for the geometry of a coffee cup. The topological guarantee becomes useful when coupled with a geometrically accurate polygonization scheme.

Guaranteeing the topology of an implicit surface polygonization solves several open problems in computer graphics.

Polygonization topology is coordinate dependent. Polygonization schemes often discern topology from point samples. Different polygonizations of the same implicit surface may differ in topology, and the same polygonization algorithm may return different topological structures depending on the coordinate system of the implicit surface, as demonstrated in Figure 1. For example, the implicit surface might appear connected when polygonized in its modeling coordinate system but could then appear disconnected when polygonized in a scene's coordinate system.

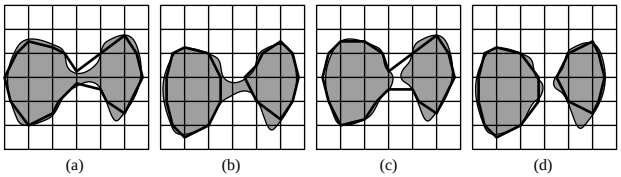


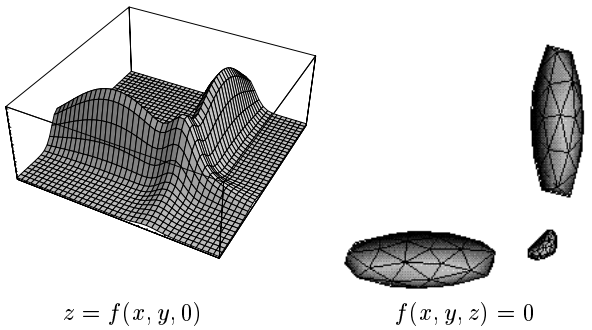
Figure 1: The topology of a connected implicit surface is correctly polygonized (a), but a translated instance is not (b). Two disjoint components are polygonized as a single component (c), but a translated instance is polygonized properly (d).

Interloping components. Some configurations of implicit surfaces can yield unexpected disjoint components. Such isolated components of the implicit surface occur because of an accumulation of neighboring potentials but do not themselves surround any "skele-

[1]Current address: Strata, 1562 El Vista Circle, Saint George, UT 84765, barts@strata3d.com
[2]Address: School of EECS, WSU, Pullman, WA 99164-2752, hart@eecs.wsu.edu

Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 1997), August 1997, pp. 279-286. © 1997 ACM reprinted with permission.

tal" geometry. For example, Figure 2 shows two blobby ellipsoids that produce a third component. Such components would not appear in a continuation-based polygonization[1], as might be used for modeling, but would appear as a surprise in a direct ray tracing of the implicit surface, as might be used for the final rendering.



$z = f(x, y, 0)$ $f(x, y, z) = 0$

Figure 2: An interloping component found at the intersection of the potential of two blobby ellipses (left) and two polygonized blobby ellipsoids (right). The algorithms described in this paper were used to correctly polygonize the interloping configuration on the right.

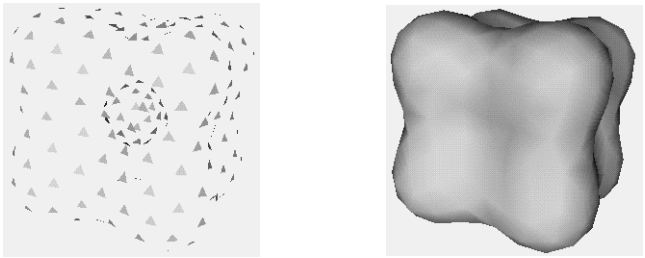


Figure 3: Implicit form displayed using a particle system (left) and polygonized (right).

Real-time implicit surface modeling. Implicit surfaces can be modeled and displayed in real-time using a particle system representation [35]. The viewer must then infer the shape of the implicit surface from the positions and orientations of the particles. Polygonization of the particles provides a better visual representation of the implicit surface, as demonstrated in Figure 3, but adding a polygonization step after every modeling change significantly degrades the otherwise real-time performance of the system. The ability to detect and correct topology changes allows the system to dynamically maintain the polygonization in real time by reconnecting the vertices of a polygonization only when a topology change has occurred, and only in the neighborhood of the topology change.

This work hence has two objectives. The first is to generate a polygonization of an implicit surface that is guaranteed to agree with its topology. The second objective is to maintain this topological guarantee during interactive manipulation of the surface. The topology of implicit surface polygonizations are guaranteed by tracking a few special points, called critical points, that dictate the topology of the implicit surface.

Section 2 examines previous polygonization techniques that consider topology. Section 3 describes critical points and adapts existing interval search methods and constraint techniques to the specific tasks of finding and tracking critical points. Section 4 analyzes the effect of critical points on topology and describes techniques for adjusting the topology of a polygonization to match the topology of the implicit surface. Section 5 describes a new polygonization algorithm based on Morse theory that polygonizes an implicit surface with a guarantee that the topology of the polygonization matches the topology of the implicit surface. Section 6 describes several new algorithms for maintaining this topological guarantee fast enough to support direct manipulation at interactive speeds. Section 7 concludes with a summary, implementation details and directions for further investigation.

[1]The techniques developed in this paper could be used to assist a continuation-based polygonization schemes detect such isolated components.

2 Previous Work

One method for interrogating an implicit surface subdivides space into cells and samples the implicit surface at the corners of these cells [22, 36, 16, 2, 20].

Some cellular polygonizations can guarantee that the implicit surface is contained in the union of a set of arbitrarily small cells, hence yielding a guarantee on surface topology accurate to a given geometric precision (e.g. the diameter of the smallest cells). Both the Lipschitz condition [14] and interval analysis [30, 17] can guarantee that an implicit surface does not pass through some cells. Cells for which this guarantee fails are "ambiguous" and can be subdivided until a given level of precision is reached and the implicit surface is assumed to lie within the union of the ambiguous cells. These ambiguous cells can then be further subdivided into "globally parameterizable" components [28]. The topology of the surface passing through such a component can be determined with a few point samples. Such cellular subdivision schemes yield a geometrically precise guaranteed representation of the implicit surface topology, though at the expense of a polygonization composed of an unnecessarily large number of small polygons.

An alternative method for interrogation constrains a particle system to the implicit surface [3, 33, 31, 9, 8]. When the implicit surface remains static, such as during a pause in user manipulation, it's particle system can be polygonized [6, 35].

A polygonization of the particle system can be maintained during user manipulation in a previous work [24]. Changes in topology were detected by comparing the interpolated polygonization normals to the implicit surface gradients at the vertices of the polygonization. Significant differences in these two vectors implied that the topology of the polygonization might not match the topology of the underlying implicit surface.

Critical points have also been used to determine implicit surface topology during a "shrinkwrap" polygonization [4]. A Lipschitz condition on the derivative of the function was used to guarantee the absence of critical points in a neighborhood, and upon failure, Newton's method was used to find the possible critical point. If Newton's method failed to converge, then the neighborhood was assumed to contain no critical points. The shrinkwrapping work also described a technique for repolygonizing the vertices surrounding a critical point based on the positions and orientations of the nearby polygons. Section 5 further explains and analyzes the shrinkwrap algorithm.

The analysis of topology using critical points is not entirely new to computer graphics. Critical points of vector and tensor fields are used to delineate topologically-distinct regions in the visualization of flow [13, 7]. Catastrophes have been used to understand caustics [18] and to interpret projections [15]. Morse theory has been used to reconstruct surfaces from cross sections [27] and to find surface-surface intersection curves [5].

3 Critical Points

The *implicit surface* defined by a function $f : \mathbb{R}^3 \to \mathbb{R}$ is the set of points $\mathbf{x} \in \mathbb{R}^3$ that satisfy $f(\mathbf{x}) = 0$. We assume the function

returns positive values inside the object, so the *solid* modeled by the implicit surface is the set of points $\{\mathbf{x}|f(\mathbf{x}) \geq 0\}$.

This work requires the function f to be C^2 continuous, with continuous first and second derivatives, and its implicit surface must be a manifold with a well defined, continuously varying surface normal. These restrictions include exponential-based "blobby" models [1], but exclude some of the more efficient C^1 piecewise polynomial approximations [21, 36].

The implicit surface is extended into a family of surfaces defined by $f(\mathbf{x};\mathbf{q})$ continuously parameterized by the vector $\mathbf{q}$ consisting of various model parameters (e.g. the locations of blobby elements). For some values of $\mathbf{q}$, the implicit surface defined by $f(\mathbf{x};\mathbf{q}) = 0$ may contain a cusp, kink or crease, specifically when the implicit surface changes topology. We consider the implicit surfaces in this family before and after but not during such topology changes. An alternative technique exists for interactively manipulating implicit surfaces with cusps, kinks and creases [25].

The *critical points* of a function f occur where its gradient

$$\nabla f(\mathbf{x}) = (f_x(\mathbf{x}), f_y(\mathbf{x}), f_z(\mathbf{x})) \tag{1}$$

vanishes. (The notation $f_x = \partial f/\partial x$.) A *critical value* is the value of the function f at a critical point.

The *Hessian* V (called the *stability matrix* in catastrophe theory) is defined as the Jacobian of the gradient

$$V(\mathbf{x}) = J(\nabla f(\mathbf{x})) = \begin{bmatrix} f_{xx}(\mathbf{x}) & f_{xy}(\mathbf{x}) & f_{xz}(\mathbf{x}) \\ f_{yx}(\mathbf{x}) & f_{yy}(\mathbf{x}) & f_{yz}(\mathbf{x}) \\ f_{zx}(\mathbf{x}) & f_{zy}(\mathbf{x}) & f_{zz}(\mathbf{x}) \end{bmatrix}. \tag{2}$$

Since $f_{xy} = f_{yx}$, etc., the stability matrix is symmetric.

A critical point $\mathbf{x}$ is classified based on the signs of the three eigenvalues $l_1 \leq l_2 \leq l_3$ of $V(\mathbf{x})$ [32]. If all three eigenvalues are non-zero, then the critical point is called *non-degenerate* and is either a maximum, minimum or some kind of saddle point. In three dimensions, saddle points come in two varieties. Table 1 indicates this classification.

l_1	l_2	l_3	Critical Point
-	-	-	Maximum Point
-	-	+	2-Saddle
-	+	+	1-Saddle
+	+	+	Minimum Point

Table 1: Classification of critical points based on the sign of the eigenvalues of the stability matrix.

The critical points are continuously dependent on the parameter vector $\mathbf{q}$. As the parameter vector $\mathbf{q}$ changes, the critical points move in space. They can also appear spontaneously in pairs, or collide in pairs, annihilating each other. If any of the three eigenvalues of the stability matrix of a critical point equal zero then $\mathbf{x}$ is called a *degenerate critical point*. The creation and destruction of critical points occur at degenerate critical points Critical point creation/destruction is demonstrated in Figure 4.

Isolated degenerate critical points are unstable. In the rare event that an isolated degenerate critical point does appear, it can be removed by a small perturbation of the implicit surface parameters without affecting the implicit surface topology.

Some functions can yields non-isolated degenerate critical points (critical sets). For example, the cylinder defined by $f(x,y,z) = x^2+y^2-1$ has a critical line along the z-axis. A small perturbation of the cylinder into an ellipsoid $f(x,y,z) = x^2+y^2+\epsilon z^2-1$ collapses the degenerate critical line into a single non-degenerate critical point at the origin. We assume the family of implicit surfaces is parameterized such that degenerate sets can be removed by such perturbation.

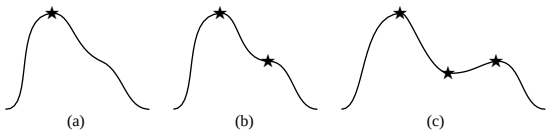


Figure 4: Creation of critical points in 1-D: $y = f(x,0,0)$. (a) Two summed Gaussian bumps, one large and one small, sufficiently close such that there is only a single maximum point in the domain shown. (b) Moving the smaller bump away from the larger creates a degenerate critical point. (c) Moving the smaller bump farther results in the creation of a pair of new critical points: a maximum point and a minimum point. Performing these steps in reverse demonstrates critical point annihilation.

3.1 Finding All Critical Points

Interval analysis searches can be guaranteed to find all points satisfying a given criterion in a given bounded domain to a desired degree of accuracy [19, 23]. Such a search can find all of the critical points of a given function to determine the topology of its implicit surface.

The interval search for critical points starts with an initial box bounding the space of interest. The simple interval search for critical points shown in Figure 5 eliminates large portions of space that cannot contain a critical point.

Given a box (a vector of intervals) $\mathbf{X} = [x_0,x_1] \times [y_0,y_1] \times [z_0,z_1]$ the algorithm checks whether the intervals returned by all of the partial derivatives contain zero. If not, then $\mathbf{X}$ contains no critical points. If so, then the algorithm subdivides $\mathbf{X}$ and tests each component individually. Note that $F_x(\mathbf{X})$ is an interval arithmetic implementation of $\partial f/\partial x$, and likewise for F_y, F_z.

```
Procedure SimpleSearch(X)
  If diam(X) < ε then indicate critical point in X.
  If 0 ∈ F_x(X) and 0 ∈ F_y(X) and 0 ∈ F_z(X) then
    Subdivide X and continue the search recursively.
```

Figure 5: Simple interval divide and conquer search algorithm.

In these algorithms, subdivision means dividing into halves with respect to its widest axis, although any number of subdivision techniques could be used. The diameter of a box $\mathrm{diam}(\mathbf{X})$ is measured using the chessboard metric, and is simply the width of the widest interval-element in the vector $\mathbf{X}$.

Simple subdivision performs remarkably well, discarding large portions of space known not to contain critical points. This technique will eventually finds all critical points to any degree of accuracy within a given bounding box, but with only linear convergence.

When the box diameter reaches a given size, the quadratically-convergent interval Newton's method shown in Figure 6 refines and/or subdivides the box down to the desired numerical precision [28, 11].

Given two points $\mathbf{x},\mathbf{y}$ there exist points $\mathbf{z}$ between[2] $\mathbf{x}$ and $\mathbf{y}$ such that

$$\nabla f(\mathbf{x}) + V(\mathbf{z})(\mathbf{y}-\mathbf{x}) = \nabla f(\mathbf{y}), \tag{3}$$

where the stability matrix V is the Jacobian of ∇f. Let $\mathbf{m}(\mathbf{X})$ return the midpoint of box $\mathbf{X}$. The algorithm seeks $\mathbf{y} \in \mathbf{X}$ such that $\nabla f(\mathbf{y}) = 0$. Since both $\mathbf{x},\mathbf{y} \in \mathbf{X}$, the $\mathbf{z}$ satisfying (3) must be in $\mathbf{X}$ as well. Thus, solving

$$\nabla f(\mathbf{m}(\mathbf{X})) + V(\mathbf{X})(\mathbf{Y}-\mathbf{m}(\mathbf{X})) = 0. \tag{4}$$

yields $\mathbf{Y}$, a box containing all of the critical points in $\mathbf{X}$. Note that

Procedure NewtonSearch($\mathbf{X}$)
  Repeat.
    Solve $V(\mathbf{X})(\mathbf{Y} - \mathbf{m}(\mathbf{X})) = -\nabla f(\mathbf{m}(\mathbf{X}))$ for Y.
    If $\mathbf{Y} \supset \mathbf{X}$ then subdivide $\mathbf{X}$ and search recursively.
    If $\mathbf{Y} \subset \mathbf{X}$ then there is a unique c.p. in $\mathbf{Y}$ (and $\mathbf{X}$).
    If $\mathbf{Y} \cap \mathbf{X} = \emptyset$ then there is no c.p in $\mathbf{X}$. Return.
    Otherwise let $\mathbf{X} = \mathbf{X} \cap \mathbf{Y}$.
  Until $\mathrm{diam}(\mathbf{X}) < \epsilon$.
  Indicate critical point at $\mathbf{m}(\mathbf{X})$.

Figure 6: Interval Newton's method search for critical points.

$V(\mathbf{X})$ returns a matrix of intervals.

An interval version of Gauss-Seidel is recommended for solving (4), but this can lead to two problems. First, the diagonal elements of $V(\mathbf{X})$ might contain zero, requiring the interval arithmetic division operation to correctly perform a division by an interval containing zero. Division by intervals containing zero produce two intervals, leading to additional algorithm recursion [28, 10]. Solving the rows whose diagonal elements do not contain zero first reduces the occurrence of semi-infinite intervals [11].

Solving

$$V_c^{-1} V(\mathbf{X})(\mathbf{Y} - \mathbf{x}) = -V_c^{-1} \nabla f(\mathbf{m}(\mathbf{X})). \tag{5}$$

where $V_c = m(V(\mathbf{X}))$ instead of (4) yields a much tighter bound and hastens the NewtonSearch performance [11]. Note that the expression $m(V(\mathbf{X}))$ returns a scalar matrix consisting of the midpoints of the intervals of $V(\mathbf{X})$.

When a critical point lies on an edge of the box, NewtonSearch's convergence is less quadratic. Extending $\mathbf{X}$ outward by a small percentage each iteration avoids this problem. Time may also be incorporated into the search by crossing $\mathbf{X}$ with the time interval $[t_0, t_1]$.

3.2 Tracking Critical Points

Altering an implicit surface's parameters changes the positions of some or all of the critical points.

The same techniques that constrain particles to adhere to the implicit surface [35], can also cause particles to adhere to any selected critical point.

Let $\mathbf{x} = \mathbf{x}(t)$ be a particle constrained to follow one of the critical points of the function f. Its partial derivatives $f_x(\mathbf{x}), f_y(\mathbf{x})$, and $f_x(\mathbf{x})$ are all constrained to zero. To ensure that they remain zero, their time derivatives $\dot{f}_x(\mathbf{x}) = \frac{d^2 f}{dxdt}(\mathbf{x})$, $\dot{f}_y(\mathbf{x})$ and $\dot{f}_z(\mathbf{x})$ must also be set to zero. Given the parameter vector $\mathbf{q}$ and its velocity $\dot{\mathbf{q}}$, one can solve these equations to determine the critical point velocity $\dot{\mathbf{x}}$. This velocity is then passed to a differential equation solver (such as fourth-order Runge-Kutta) to approximate the new location of the critical point. Newton's method refines the approximation.

4 Detecting and Correcting Topology Changes

The identification of critical points simplifies topologically-guaranteed direct manipulation of implicit surfaces through a polygonal representation. The key to solving the topology problem is that a change in the topology of a surface is always accompanied by a change in the sign of a critical value [12]. Monitoring the critical points greatly simplifies the burden of detecting topological changes, and divides the problem into classifying topological changes, identifying polygons to remove and reconnecting the verticed of the removed polygons.

[2] The notion of "between" for points in space means that $\mathbf{z}$ is in a box with corners at $\mathbf{x}$ and $\mathbf{y}$.

4.1 Classifying Topological Changes

Table 2 enumerates all of the possible critical-point/sign combinations and their corresponding implications on the implicit surface topology. When an implicit surface topology change is detected, the polygonization must be altered to properly represent the new topology.

Critical Point	Sign Changes To	Action
Maximum	-	Destroy
Maximum	+	Create
2-Saddle	-	Cut
2-Saddle	+	Attach
1-Saddle	-	Pierce
1-Saddle	+	Spackle
Minimum	-	Bubble
Minimum	+	Burst

Table 2: The affect of critical point sign on topology.

4.2 Identifying Polygons to Remove

Changes in maximum and minimum critical values cause entire simply-connected components of polygonization to be removed or created. Changes in saddle points require the determination of specific polygons to be removed such that their vertices may be properly reconnected. These polygons intersect a separatrix extending from the saddle point.

The separatrix may be efficiently approximated by a line for 2-saddles, or a plane for 1-saddles. These lines and planes described by the eigenvectors of the stability matrix of a critical point approximate the separatrix.

When separatrixes are linearly approximated by lines and planes, certain errors might occur. For example, a 2-saddle may connect two components, but the line approximating its separatrix might not intersect either component. One must then assume that the parameter vector $\mathbf{q}$ is sufficiently close to the parameter vector at the topology change $\mathbf{q}_*$ that the linear approximation correctly intersect the proper polygonized implicit surface components.

The separatrix extending from a 2-saddle can be treated as an initial value problem, using the positive eigenvector $\mathbf{v}_3(\mathbf{x})$ of the stability matrix to define the ordinary differential equation

$$\dot{\mathbf{x}} = \mathbf{v}_3(\mathbf{x}) \tag{6}$$

and using numerical integration to trace out the path of the separatrix. The midpoint method provided sufficient numerical accuracy for this task in our experiments.

The case where a separatrix intersects a polygonization vertex can be removed with a topology-preserving perturbation.

4.3 Reconnection

The following procedures describe which polygons must be removed, and how their vertices are reconnected to update the topology of the polygonization.

Destroy. When the value at a maximum goes negative, an isolated component in the implicit surface disappears. A ray cast from the maximum point in any direction will first intersect a polygon in this simply-connected component. All polygons connected to this polygon are then removed. Figure 7 pseudocodes this algorithm.

Create. When the value at a maximum goes positive, a new, simply-connected component in the implicit surface appears. A sufficiently small tetrahedron may be placed around the maximum point, letting its vertices adhere to the implicit surface component and adding

```
Procedure Destroy/Burst
  Cast a ray from maximum point in any direction.
  Let p be the first polygon the ray intersects.
  Push p onto stack.
  While stack not empty.
    Let p be the result of popping the stack.
    For all polygons q sharing an edge with p.
      Push q onto stack.
    Remove p from polygonization.
```

Figure 7: The repolygonization algorithm for *Destroy* and *Burst.*

new polygons when necessary. Alternatively, a ray may be cast from the maximum point and intersected with the implicit surface. The first intersection denotes the location where any standard continuation polygonization technique may be applied. Figure 8 pseudocodes this algorithm.

```
Procedure Create/Bubble
  Cast a ray from maximum point in any direction.
  Let x ∈ f^{-1}(0) be the first ray intersection.
  Polygonize the component containing x.
```

Figure 8: The repolygonization algorithm for *Create* and *Bubble.*

Cut. When the value at a 2-saddle goes negative, part of the implicit surface disconnects. The separatrix surface extending from the 2-saddle is found by integrating the two negative eigenvalues of the stability matrix will intersect the polygons in a ring surrounding the 2-saddle. In practical cases, this separatrix is sufficiently approximated by a plane passing through the 2-saddle perpendicular to its positive eigenvector. The ring of polygons intersecting the separatrix surface are removed, yielding two disjoint rings of polygonization vertices. These rings are "sewn up" individually via triangulation. Figure 9 pseudocodes this algorithm, and Figure 10 illustrates the polygon configuration.

```
Procedure Cut/Spackle
  Let P be a plane containing the critical point x perpendicular to the uniquely-signed eigenvector of V(x).
  Cast a ray from x in any direction within P.
  Let p_0 be the first polygon intersected by the ray.
  Initialize i = 0 and repeat.
    Let p_{i+1} be a polygon intersecting P, sharing an edge with p_i, and not equal to any p_j for j ≤ i.
    If no such p_{i+1} exists, break.
    Increment i.
  Let v_0 be any vertex of p_0.
  Call Procedure Ring.
  Triangulate v_i.
  Let v_0 be any vertex of p_0 not currently triangulated.
  Call Procedure Ring.
  Triangulate v_i.

Procedure Ring
  Initialize i = 0 and repeat.
    Let e be an edge connecting vertex v_i to v_{i+1} separating a p_k polygon from a non-p_k polygon, and vertex v_{i+1} not equal to any v_j for j ≤ i.
    If no such v_{i+1} exists, break.
    Increment i.
```

Figure 9: The repolygonization algorithm for *Cut* and *Spackle.*

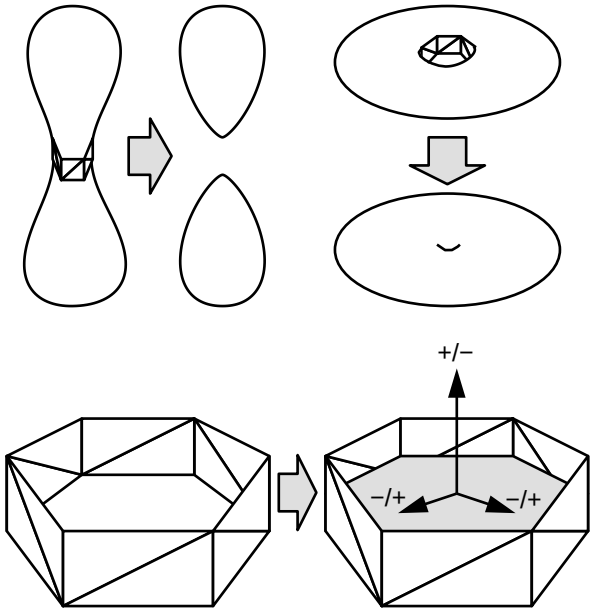


Figure 10: Polygons, eigenvalues and eigenvectors for *Cut* and *Spackle.*

Attach. When the value at a 2-saddle goes positive, two components of the implicit surface connect. The separatrix curve extends from the 2-saddle in the direction of its positive eigenvalue to a maximum point inside each component. The first polygon in each direction the separatrix intersects is removed. This leaves two disjoint rings of vertices that need to be connected. Proper correspondence algorithms between the two polygons can be found (e.g. [26]), but such techniques are not necessary if the polygonization is restricted to triangles. Figure 11 pseudocodes this algorithm.

```
Procedure Attach/Pierce
  Extend separatrix curves from the critical point.
  Let polygons p_0 and p_1 first intersect each separatrix.
  Connect the vertices of p_0 with the vertices of p_1.
  Remove p_0 and p_1.
```

Figure 11: The repolygonization algorithm for *Attach* and *Pierce.*

Pierce. When the value at a 1-saddle goes negative, a hole is pierced in the implicit surface. Similar to the *attach* case (a hole in the implicit surface of f is a connection in the implicit surface of $-f$), the two polygons that intersect the separatrix curve passing through the 1-saddle in the direction of the eigenvector corresponding to the one negative eigenvalue of the stability matrix are identified. These two polygons are removed and now form the ends of the hole. Corresponding and connecting the resulting two rings of vertices form the walls of the hole. The algorithm in Figure 11 also repolygonizes the *pierce* case.

Spackle. When the value at a 1-saddle goes positive, a hole in the implicit surface is filled. Similar to the *cut* case, the separatrix surface is constructed at the 1-saddle perpendicular to the eigenvector corresponding to the one negative eigenvalue of the stability matrix. The local polygons this surface pierces are removed, and the two resulting polygonal holes are "sewn up" by triangulation. The algorithm in Figure 9 also repolygonizes the *spackle* case and Figure 10 illustrates the polygon configuration.

Bubble. When the value at a minimum goes negative, a pocket of

air forms inside the implicit solid. An air pocket in the implicit surface of f is a new component in the implicit surface of $-f$. This pocket of air may therefore be treated as a simply-connected implicit surface component, and polygonized using any of the existing techniques. The algorithm in Figure 8 also repolygonizes the *bubble* case.

Burst. When the value at a minimum goes positive, an air bubble within the implicit solid has burst. As in the Destroy case, a ray is cast from the minimum in any direction. The first polygon this ray intersects, as well as any other polygons with a connection to it, are then removed. The algorithm in Figure 7 also repolygonizes the *burst* case.

5 Polygonization

Morse theory provides the background for a topologically-guaranteed polygonization algorithm. Given a function f(**x**) implicitly defining the surface $f^{-1}(0)$, we consider the family of surfaces $f^{-1}(a)$ for non-negative a. Let a_0 be a value of sufficient magnitude such that the surface $f^{-1}(a_0) = \emptyset$. As a_0 decreases, it will pass critical values for maximum points, 2-saddles, 1-saddles and minimum points. As each critical value is encountered, the topology of the polygonization around its critical point is corrected. This "inflation" algorithm is pseudocoded in Figure 12.

```
Procedure Inflate
  X_c = Search X for {x : ∇f(x) = 0}.
  Let a_0 > max_{x∈X_c} f(x).
  Polygonize f^{-1}(a_0).
  For a = a_0 − ε to 0 step −ε.
    Adjust vertices to f^{-1}(a).
    If ∃x ∈ X_c : a < f(x) < a + ε then
      Correct topology change in polygonization.
  Return polygonization of f^{-1}(0).
```

Figure 12: The "Inflate" polygonization algorithm.

When a maximum point is passed, a new simply-connected component appears via the *create* routine. When a 2-saddle is passed, a connection formed via the *attach* routine. When a 1-saddle is passed, a hole is filled via the *spackle* routine. When a minimum point is passed, a hollow bubble is filled via the *burst* routine. The *Inflation* polygonization of a blobby cube is demonstrated in Figure 13

Shrinkwrapping similarly polygonizes implicit surfaces but from the opposite direction, approaching the isovalue from the negative side [34]. Hence, the polygonization begins with a large simply-connected spheroid, which shrinks and appears to adhere to the final implicit surface. Morse theory can be incorporated to detect changes in topology during the shrinkwrapping process [4].

One problem with shrinkwrapping is that its outside-in processing fails to account for hollow bubbles within an implicit surface whereas *inflate's* inside-out processing correctly detects and polygonizes these regions. While such regions are typically hidden from the viewer, they become visible when the surface is rendered translucently or when the surface's polygonization is later intersected, trimmed, clipped or blended.

6 Interactive Repolygonization

The interaction algorithm consists of an initialization stage followed by an interactive loop of user input, model update, and model display. The system assumes it is initialized with a topologically correct polygonization, such as is described in Section 5.

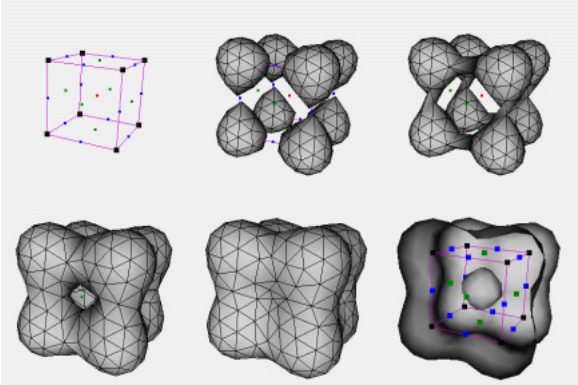


Figure 13: Polygonization via *Inflation* of the blobby cube. The last image illustrates the air bubble by only rendering back-facing polygons. Note that the definition of back facing may be the opposite of one's intuition for an air bubble.

```
Procedure Shrinkwrap
  X_c = Search X for {x : ∇f(x) = 0}.
  Let a_0 < min_{x∈X_c} f(x).
  Polygonize f^{-1}(a_0).
  For a = a_0 + ε to 0 step ε.
    Adjust vertices to f^{-1}(a).
    If ∃x ∈ X_c : a < f(x) < a + ε then
      Correct topology change in polygonization.
  Return polygonization of f^{-1}(0).
```

Figure 14: The "Shrinkwrap" polygonization algorithm [4].

For each time step, the interaction algorithm performs the steps in Figure 15.

```
Procedure InteractionLoop
  Repeat.
    Alter model parameters based on user manipulation.
    Adjust vertex positions, fix mesh.
    Determine critical points.
    Correct polygonization topology.
    Render.
```

Figure 15: The interaction loop for interactive implicit surface modeling.

Figure 16 shows a critical point tracking algorithm. During user interaction, the critical points move and change sign. Furthermore, one or more of the eigenvalues of the stability matrix can change sign at some degenerate critical point $\mathbf{x}$, resulting in the creation or annihilation of a pair of critical points. Critical point annihilation is revealed by the collision of two critical-point tracking particles. Critical point creation occurs spontaneously and is not detected by the tracking particles.

Searching in four-dimensions, as shown in Figure 17, allows us to find the location in space and time of degenerate critical points. Since critical points can be tracked and annihilation can be detected, the interval search would serve to detect the spontaneous creation of critical points. Such occurrences rarely happen, but when they do occur they appear as double zeros (both $\nabla f(\mathbf{x})$ and $|V(\mathbf{x})| = 0$) which degrades convergence.

An alternative search shown in Figure 18 finds the location in space and time for singular points in the family of implicit surfaces. where the value of a critical point changes sign. Such occurrences

Procedure Track-n-Search
Track critical points.
Search $\mathbf{X}$ for $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}\}$.

Figure 16: The "Track-n-Search" critical point determination algorithm.

Procedure Track-n-SearchDegenerate
Track critical points.
Search $\mathbf{X} \times [t_0, t_1]$ for $\{\mathbf{x} : \nabla f(\mathbf{x}) = \mathbf{0}, |V(\mathbf{x})| = 0\}$.

Figure 17: The "Track-n-SearchDegenerate" critical point determination algorithm.

occur as rarely as degenerate critical points, so the interval search can quickly guarantee that such points do not exist. However, when they do exist, as with degenerate critical points, they appear as double zeros which degrades the rate of convergence.

7 Conclusion

Using techniques from catastrophe theory and Morse theory, the preceding sections developed (1) a new polygonization algorithm that can guarantee the topology of the polygonization matches that of the implicit surface, and (2) a new implicit surface modeling system capable of maintaining a topologically-accurate polygonized representation of the implicit surface during direct manipulation at interactive update rates.

Section 4 improves previous *ad hoc* geometry-only techniques [24, 4] by describing a mathematically sound method for using the separatrix to identify the polygons affected by a topology change, and robust algorithms for reconnecting the vertices of the polygonization.

Section 5 uses these techniques to polygonize an implicit surface. This method improves previous geometry-based interval methods [28] in that it is faster and does not return a large number of unnecessarily small polygons. The interval search is also guaranteed to find all critical points, which overcomes the uncertainty of previous methods [4], and also properly polygonizes hollow bubbles when they appear within an implicit surface.

Some initial experiments revealed that performance dropped below ten frames per second on scenes containing combinations of four or more interacting blobby ellipsoids. Modeling sessions that string a chain of blobby components operate in real time, but sessions with densely packed arrangements of blobby components appear sluggish in the current prototype implementation of the system. Even apparently simple configurations of blobby components can yield numerous nearly-degenerate critical points, and their detection is required for accurate topology management. This performance was measured using the SearchSingularity interaction loop, but is similar to the performance of the other interaction loop critical point search/tracking methods. This procedure becomes noticeably slow near topology changes. While any speed degradation and inconsistency is undesirable, the algorithm does focus its computation on the time and space where it is most needed.

Procedure SearchSingularity
Search $\mathbf{X} \times [t_0, t_1]$ for $\{\mathbf{x} : f(\mathbf{x}) = 0, \nabla f(\mathbf{x}) = \mathbf{0}\}$.

Figure 18: The "SearchSingularity" algorithm.

7.1 Some Implementation Tricks

One of the topological guarantee's restrictions was the lack of degenerate critical points. However, for speed, we were able to implement a C^2 cubic approximation to the exponential blobby model. The kernel of this approximation is uniform away from the center, and results in a three-dimensional degenerate critical set. We overcame this problem by assuming that the derivative intervals with zero for one endpoint did not contain a critical point, but instead contained a portion of this 3-D critical set. We avoided the possibility that a critical point fell on the boundary of the interval by expanding each interval by a small percentage.

Occasionally, the program errs in its attempt to process a topology change. In such cases, the system automatically initiates a full "inflation" repolygonization.

Further implementation details can be found in the dissertation [29].

7.2 Future Work

Tracking critical points is much faster than searching for them, but does not account for the pairs of critical points that can be created spontaneously. Tracking all of the derivatives of the function could detect degenerate critical points. This is not possible for exponentials because they are infinitely differentiable, and their derivatives become increasingly complex. Piecewise polynomials have finitely many derivatives that become increasingly simple, and might offer the opportunity to attempt such tracking.

Implicit surfaces still offer many challenges in modeling, texturing and animation due to the flexibility of their topology. This research solved the problem of interactive polygonization. Understanding the dynamics of critical points might lead to further solutions to other implicit surface problems, such as maintaining a consistent texturing during a topology change.

This research focused on 3-D implicit surfaces. Its application to the polygonization and modeling of 2-D implicit curves would be a useful, though perhaps now trivial, simplification.

7.3 Acknowledgments

This research was supported in part by the NSF Research Initiation Award #CCR-9309210. This research was performed in the Imaging Research Laboratory. The authors would like to thank the SIGGRAPH reviewers for their constructive criticism and positive comments. Further thanks are due to Dan Asimov, Jules Bloomethal and Jim Kajiya for their help in tracking down theorems in Morse theory. Special thanks to Andrew Glassner and Scott Lang for their assistance with the photoready copy of this paper.

REFERENCES

[1] BLINN, J. F. A generalization of algebraic surface drawing. *ACM Transactions on Graphics 1*, 3 (July 1982), 235–256.

[2] BLOOMENTHAL, J. Polygonization of implicit surfaces. *Computer Aided Geometric Design 5*, 4 (Nov. 1988), 341–355.

[3] BLOOMENTHAL, J., AND WYVILL, B. Interactive techniques for implicit modeling. *Computer Graphics 24*, 2 (Mar. 1990), 109–116.

[4] BOTTINO, A., NUIJ, W., AND VAN OVERVELD, K. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 53–72.

[5] CHENG, K.-P. Using plane vector fieldsto obtain all the intersection curves of two general surfaces. In *Theory and Practice of Geometric Modeling* (New York, 1989), Springer-Verlag.

[6] DE FIGUEIREDO, L. H., DE MIRANDA GOMES, J., TERZOPOULOS, D., AND VELHO, L. Physically-based methods for polygonization of implicit surfaces. In *Proceedings of Graphics Interface '92* (May 1992), pp. 250–257.

[7] DELMARCELLE, T., AND HESSELINK, L. The topology of symmetric, second-order tensor fields. *Proceedings IEEE Visualization '94* (October 1994), 140–147.

[8] DESBRUN, M., TSINGOS, N., AND GASCUEL, M.-P. Adaptive sampling of implicit surfaces for interactive modeling and animation. *Implicit Surfaces '95 Proceedings* (April 1995), 171–185.

[9] FLEISCHER, K. W., LAIDLAW, D. H., CURRIN, B. L., AND BARR, A. H. Cellular texture generation. In *Computer Graphics (Annual Conference Series)* (Aug. 1995), pp. 239–248.

[10] HANSEN, E. A globally convergent interval method for computing and bounding real roots. *BIT 18* (1978), 415–424.

[11] HANSEN, E. R., AND GREENBERG, R. I. An interval newton method. *Applied Mathematics and Computation 12* (1983), 89–98.

[12] HART, J. C. Morse theory for computer graphics. Tech. Rep. EECS-97-002, Washington State University, May 1997. Also in: SIGGRAPH '97 Course #14 Notes "New Frontiers in Modeling and Texturing".

[13] HELMAN, J. L., AND HESSELINK, L. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications* (May 1991), 36–46.

[14] KALRA, D., AND BARR, A. H. Guaranteed ray intersections with implicit surfaces. *Computer Graphics 23*, 3 (July 1989), 297–306.

[15] KERGOSIEN, Y. L. Generic sign systems in medical imaging. *IEEE Computer Graphics and Applications 11*, 5 (Sep. 1991), 46–65.

[16] LORENSEN, W. E., AND CLINE, H. E. Marching cubes: A high resolution 3-d surface construction algorithm. *Computer Graphics 21*, 4 (July 1987), 163–170.

[17] MITCHELL, D. Three applications of interval analysis in computer graphics. In *Frontiers of Rendering*. SIGGRAPH '91 Course Notes, 1991.

[18] MITCHELL, D., AND HANRAHAN, P. Illumination from curved reflectors. *Computer Graphics 26*, 2 (July 1992), 283–291.

[19] MOORE, R. E. *Interval Analysis*. Prentice Hall, 1966.

[20] NING, P., AND BLOOMENTHAL, J. An evaluation of implicit surface tilers. *Computer Graphics and Applications 13*, 6 (Nov. 1993), 33–41.

[21] NISHIMURA, H., HIRAI, M., KAWAI, T., KAWATA, T., SHIRAKAWA, I., AND OMURA, K. Object modeling by distribution function and a method of image generation. In Proc. of *Electronics Communication Conference '85* (1985), pp. 718–725. (Japanese).

[22] NORTON, A. Generation and rendering of geometric fractals in 3-D. *Computer Graphics 16*, 3 (1982), 61–67.

[23] RATSCHEK, H., AND ROKNE, J. *Computer Methods for the Range of Functions*. John Wiley and Sons, 1984.

[24] RODRIAN, H.-C., AND MOOCK, H. Dynamic triangulation of animated skeleton-based implicit surfaces. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 37–52.

[25] ROSCH, A., RUHL, M., AND SAUPE, D. Interactive visualization of implicit surfaces with singularities. In *Proc. Implicit Surfaces '96* (Oct. 1996), pp. 73–87.

[26] SEDERBERG, T. W., AND GREENWOOD, E. A physically based approach to 2-D shape blending. *Computer Graphics 26*, 2 (July 1992), 25–34.

[27] SHINAGAWA, Y., KUNII, T. L., AND KERGOSIEN, Y. L. Surface coding based on morse theory. *IEEE Computer Graphics and Applications 11*, 5 (Sep. 1991), 66–78.

[28] SNYDER, J. *Generative Modeling for Computer Graphics and CAD*. Academic Press, 1992.

[29] STANDER, B. T. *Polygonizing Implicit Surfaces with Guaranteed Topology*. PhD thesis, School of EECS, Washington State University, May 1997.

[30] SUFFERN, K., AND FACKERELL, E. Interval methods in computer graphics. In *Proc. AUSGRAPH 90* (1990), pp. 35–44.

[31] SZELISKI, R., AND TONNESEN, D. Surface modeling with oriented particle systems. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 185–194.

[32] TAYLOR, A. E. *Advanced Calculus*. Ginn and Company, 1955.

[33] TURK, G. Generating textures for arbitrary surfaces using reaction-diffusion. In *Computer Graphics (SIGGRAPH '91 Proceedings)* (July 1991), T. W. Sederberg, Ed., vol. 25, pp. 289–298.

[34] VAN OVERVELD, C., AND WYVILL, B. Shrinkwrap: an adaptive algorithm for polygonizing and implicit surface. Tech. Rep. 93/514/19, University of Calgary, Dept. of Computer Science, March 1993.

[35] WITKIN, A. P., AND HECKBERT, P. S. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)* (July 1994), pp. 269–278.

[36] WYVILL, G., MCPHEETERS, C., AND WYVILL, B. Data structure for soft objects. *Visual Computer 2*, 4 (1986), 227–234.

Using Particles to Sample and Control More Complex Implicit Surfaces

John C. Hart, Ed Bachta, Wojciech Jarosz, Terry Fleury
University of Illinois
{jch|bachta|wjarosz|tfleury}@uiuc.edu

Abstract

In 1994, Witkin and Heckbert developed a method for interactively modeling implicit surfaces by simultaneously constaining a particle system to lie on an implicit surface and vice-versa. This interface was demonstrated to be effective and easy to use on example models containing a few blobby spheres and cylinders. This system becomes much more difficult to implement and operate on more complex implicit models. The derivatives needed for the particle system behavior can become laborious and error-prone when implemented for more complex models. We have developed, implemented and tested techniques for automatic and numerical differentiation of the implicit surface function. Complex models also require a large number of parameters, and the management and control of these parameters is often not intuitive. We have developed adapters, which are special shape-transformation operators that automatically adjust the underlying parameters to yield the same effect as the transformation. These new techniques allow constrained particle systems to sample and control more complex models than before possible.

1 Introduction

Witkin and Heckbert [14] revolutionized implicit surface modeling by using a particle system to both display and control an implicit surface. Their treatment used a real function $F : \mathbf{R}^3 \times Q \to \mathbf{R}$ over model space $\mathbf{R}^3$ and a continuous parameter space Q. This real function yielded an implicit surface as the solution points x such that $F(\mathbf{x}, \mathbf{q}) = 0$ for a fixed, given vector of parameters $\mathbf{q} \in Q$.

A particle $\mathbf{p}^i$ constrained to the implicit surface of F such that $F(\mathbf{p}^i, \mathbf{q}) = 0$ is called a *floater*. This constraint is enforced by setting its original velocity $\dot{\mathbf{P}}^i$ to a legal velocity $\dot{\mathbf{p}}^i$ by subtracting any illegal components normal to the implicit surface

$$\dot{\mathbf{p}}^i = \dot{\mathbf{P}}^i - \frac{F^i_{\mathbf{x}} \cdot \dot{\mathbf{P}}^i + F_{\mathbf{q}} \cdot \dot{\mathbf{q}} + \phi F^i}{F^i_{\mathbf{x}} \cdot F^i_{\mathbf{x}}} F^i_{\mathbf{x}}. \qquad (1)$$

(Note that here $\dot{\mathbf{P}}$ denotes the desired velocity instead of $\mathbf{P}$ [14].) These illegal components are due to either particle velocities ($F^i_{\mathbf{x}} \cdot \dot{\mathbf{P}}^i$) or parameter velocities ($F_{\mathbf{q}} \cdot \dot{\mathbf{q}}$) that change the resulting value of F. Hence we need the derivative of F with respect to both its embedding $F_{\mathbf{x}}$ and its parameterization $F_{\mathbf{q}}$.

The constrained particle system displayed the implicit surface with a collection of disks centered at the particles oriented according to the surface normal. These oriented disks provide a usable and highly responsive display of the underlying implicit surface, and also yield a quasi-volumetric display of the surface that reveals interior structure in the gaps between disks. The visual edge noise created by the disks can sometimes be distracting, but this can be overcome by connecting the particles into a polygonization using a topological guarantee [12].

Some floater particles can be selected as control particles, which means the implicit surface is constrained to pass through these particles. Control particles can be dragged to new locations, and the implicit surface deforms to accomodate its new position. This deformation occurs by changing the parameters $\mathbf{q}$ of the implicit surface using the parameter velocity

$$\dot{\mathbf{q}} = \dot{\mathbf{Q}} - \sum_j \lambda^j F^j_{\mathbf{q}}. \qquad (2)$$

The index j is the index of the control particle, and $\dot{\mathbf{Q}}$ is the desired unconstrained parameter velocity. The λ^j are Lagrange multipliers found by solving the system

$$\sum_j \left(F^i_{\mathbf{q}} \cdot F^j_{\mathbf{q}}\right) \lambda^j = F^i_{\mathbf{q}} \cdot \dot{\mathbf{Q}} + F^i_{\mathbf{x}} \cdot \dot{\mathbf{p}}^i + \phi F^i. \qquad (3)$$

The desired parameter velocity $\dot{\mathbf{Q}}$ is usually zero, such that (2) and (3) dissipate control particle velocities $\dot{\mathbf{p}}$ into parameter velocities $\dot{\mathbf{q}}$.

Witkin and Heckbert [14] alluded to an object-oriented implicit surface class hierarchy, where new implicit model objects need to implement $F, F_{\mathbf{x}}, F_{\mathbf{q}}$ and a bounding box (to keep the particles from following a non-compact manifold indefinitely). Most implicit modeling systems implement the function, its gradient and a bounding box, so the only

new information needed is the derivative of the implicit surface function with respect to its parameters.

The goal of the work described in this paper is to integrate the particle system into a full-featured implicit surface modeling system. Section 2 reviews some previous implicit surface modeling systems. To our knowledge, no one has yet implemented such an object hierarchy in a full-featured implicit modeling system based on the control particle interface.

We have developed such an object-oriented class hierarchy specifically for inclusion into a particle-based modeling system. Section 3 describes the flexibility of our system that includes a large variety of implicit surface primitives and operators.

Witkin and Heckbert [14] suggest (once F_q is implemented) the application of particle-based interaction to complex hierarchical implicit surface models is straightforward. In the process of implementing such a system we have found several setbacks, specifically in implementing and debugging the function derivatives $F_{\mathbf{x}}$ and $F_{\mathbf{q}}$, and managing large numbers of parameters for manipulating complex composite implicit surface models, as demonstrated by the 30+ parameters shown in Figures 1 and 2.

One of the challenges of implementing a full-featured implicit surface representation for a particle-based modeler is the development of the derivatives needed. The particle-based modeler requires each primitive and operator to have a derivative with respect to its embedding space and its parameters. Section 4 describes several methods available to ease the programming of these derivatives.

Complex hierarchies of implicit operators and primitives quickly grow to become overwhelming. Section 5 describes methods for modeling that use special operators called adapters that reparameterize a complex implicit model. This reparameterization simplifies the modeling process by providing more intuitive parameters to the user.

2 Other Complex Implicit Surface Modeling Systems

This paper develops a modeling system capable of creating complex models of a wide variety of implicit primitives and operators using a particle system for display and control. Several other modeling systems have been previously developed to create complex implicit surface models, but each of these system has been limited to a specific subset of implicit primitives and operations.

Witkin and Heckbert [14] used a prototype implemention of a particle-based implicit surface modeler that proved the concept. Their implementation was never released, and only supported small collections of a few primitives, including blobby spheres and cylinders. Pedersen [8] later released a more robust, flexible and freely available particle system based on (1) of floater particles for displaying implicit surfaces, but it did not include an implementation of (2) and (3) that provided the control particles needed to interactively change the surface parameters. Stander and Hart [12] later described how to use interval analysis and Morse theory to connect surface constrained floater particles into a polygonization. Their implementation was also a proof-of-concept prototype that was never released. It was limited to axis-aligned Gaussian ellipsoids, and the dynamic mesh that connected the particles into a polygonization was very fragile.

The Blob Tree integrated multiple implicit modeling tools into a single scene description hierarchy for implicit surface modeling [15]. The Blob Tree was based on piecewise-polynomial blobby sphere primitives that blend when placed in proximity to each other. The Blob Tree organized objects into groups and groups can be combined with a smooth surface blend or a creased CSG operation. The blend tree also supported other implicit surface operations, for example non-linear deformations.

The Hyperfun system supported collaborative modeling of implicit surfaces [1]. Hyperfun is based on R-functions which generalize blending between primitives to include non-blended CSG operations. Its components include a scene description hierarchy, a declarative language for defining functions, new file formats for communicating implicit surface descriptions, and integration of the results into common graphics systems such as POVRay.

Implicit surfaces are also supported by the VTK toolkit [10, 11]. These implicit surfaces could be used a sources for volumetric "structured-points" data pipelines. These pipelines can be made arbitrarily complex, and have been used to model context geometry to aid in the visualization of acquired volumetric data [9]. Implicit surface sources are integrated into the VTK toolkit by defining the function and its gradient, though the VTK tools typically sample the source into a volume before performing any further processing.

3 A General Implicit Object Model

Many previous implicit surface modeling systems have focused only on a subset of the available implicit surface models. Our goal is to design a general full-featured implicit surface system that could support any implicit surface representation or operation.

The `Implicit` class sits at the root of our implicit surface hierarchy, and contains three key member functions. The member function `proc`($\mathbf{x}$) returns the scalar result $F(\mathbf{x}, \mathbf{q})$. The member function `grad`($\mathbf{x}$) returns the 3-vector result $F_{\mathbf{x}}(\mathbf{x}, \mathbf{q})$. The member function `procq`($\mathbf{x}, \mathbf{dq}$) puts the derivative $F_{\mathbf{q}}(\mathbf{x}, \mathbf{q})$ into the vector $\mathbf{dq}$. Each of these functions assumes $\mathbf{q}$ is constant and held

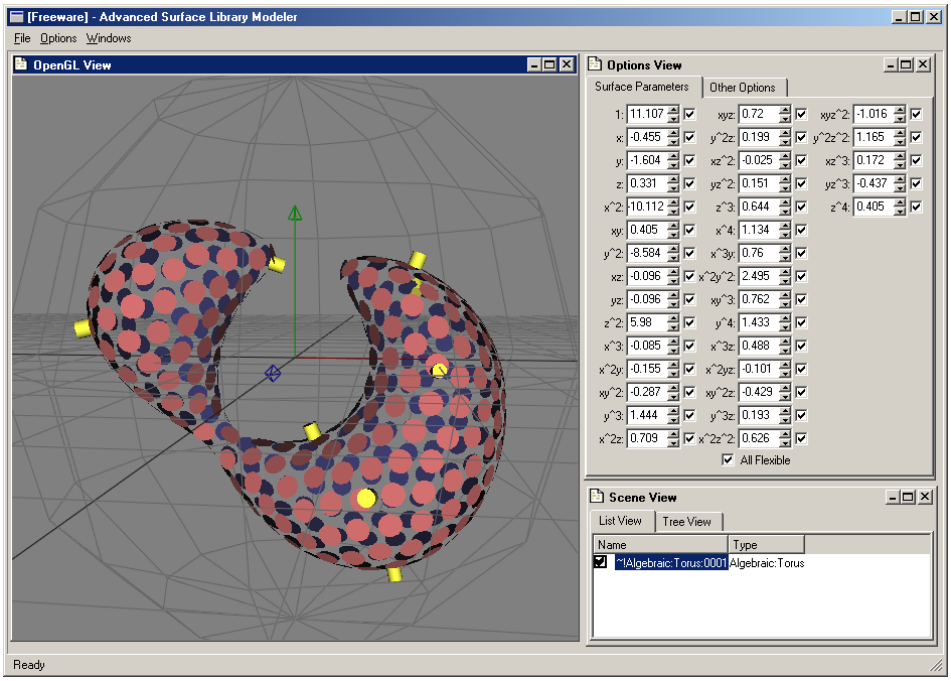


Figure 1. A sample screen of our particle-based modeler directly manipulating a quartic surface. This quartic surface has 35 continuous parameters shown on the right that can be entered individually or selected to be set by the controller particle constraints.

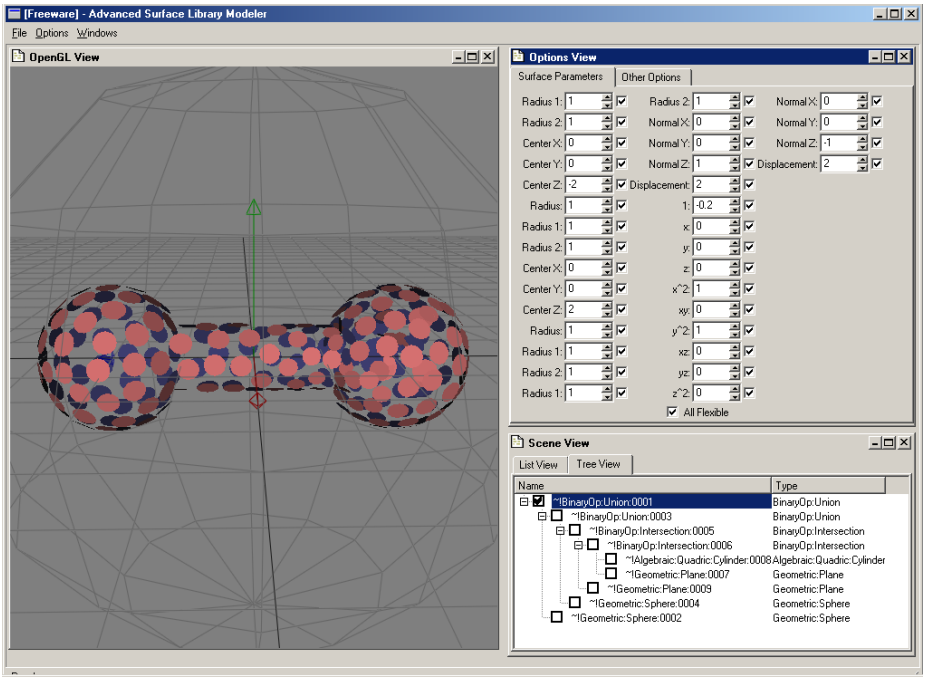


Figure 2. The particle-based modeler directly manipulating a dumbell modeled using R-functions. This composite surface has 34 continuous parameters shown on the right that can be entered individually or selected to be set by the controller particle constraints.

within the `Implicit` as part of its internal state.

We provide member access to the continuous parameters through an interface uniform across all classes derived from `Implicit`. The member function `getq(q)` puts the current parameters in vector $\mathbf{q}$, whereas the member `setq(q)` sets the current parameters to those in vector $\mathbf{q}$. This parameter access allows the particle system to query and adjust the continuous parameters as necessary to apply the floater and control particle constraints. The actual implementation of a specific implicit object need not store its parameters in a vector, and can provide additional specialized access methods to its parameters[1].

We have also implemented implicit surface operators (e.g. offset, scale, union, etc.). These operators affect the inputs and/or the outputs of one or more operand implicit objects. The `getq` and `setq` functions for the operators place the operator's parameters at the beginning of the vector, and follow them with the parameters of its operands. Hence, one can access all of the parameters of a hierarchically-defined implicit object through the `getq`/`setq` interface of the root operator object in the hierarchy.

4 Differentiation

One of the main obstacles in our implementation of a variety of implicit surface models has been in the derivation, implementation and debugging of derivatives of the functions.

4.1 Operator-Level Automatic Differentiation

The derivatives of operators, namely `grad` and `procq`, are implemented using use the chain rule, eventually calling `grad` and `procq` of the operands. The operations can be considered an arithmetic on implicit objects, which makes the chain-rule `grad` and `procq` implementations a partial form of automatic differentiation,

One could define arithmetic operations as Implicit operators. For example, the operator `Times` implements the function FG as

```
Times::proc(x) {
  return F->proc(x) * G->proc(x);
}
Times::grad(x) {
  return F->grad(x)*G->proc(x) +
         F->proc(x)*G->grad(x);
}
```

[1]For example, we have derived an `Algebraic` class from Implicit that provides access to the $d^3/6 + d^2 + 11d/6 + 1$ coefficients of a degree d trivariate polynomial. The coefficients are returned sorted by degree, then by x, y and z exponents. However, access to these coefficients is much easier by providing the exponents of x, y and z than providing a single coefficient index.

Given a sufficient set of these arithmetic operations, one could implement any implicit model. Implicit models constructed by composing these operators would have their differential functions automatically defined, since these objects compute their own derivatives. However, such an implementation would be cumbersome and inefficient.

Rather than implement high-level shape operators using automatically differentiated low-level arithmetic operators, we chose instead to automatically differentiate the high-level shape operators. A good example is our implementation of an abstract `Blend` class. Our blend class assumes the blend combines the isocontours of its operand implicit objects F and G [6]. The blend is thus defined by a real bivariate function $h(f, g)$ of the values f, g returned by the operands. The blend surface is thus implemented as

```
Blend::proc(x) {
  return h(F->proc(x),G->proc(x));
}
```

Specific blends are derived from the `Blend` class, and define the pure virtual function h. The blobby model can blend arbitrary implicit surfaces using the function

$$h(f, g) = T - e^{-f-a} - e^{-g-b} \tag{4}$$

where T, a and b control the "blobbiness" of the blend [4]. The superelliptical blend is given in this form as

$$h(f, g) = \frac{(f-a)^d}{a^d} + \frac{(g-b)^d}{b^d} - 1 \tag{5}$$

where a and b are continuous parameters controlling the extent of the blend and the discrete parameter d is the degree of the blending function [6]. R-functions provide a continuous neighborhood for CSG operations, and are given by

$$h(f, g) = (f + g + s\sqrt{f^2 + g^2})(f^2 + g^2)^{d/2} \tag{6}$$

where the discrete parameters $s = \pm 1$ differentiates between union and intersection, and d again provides a degree of smoothness [7].

Using the chain rule, we define

```
Blend::grad(x) {
  return hf(F->proc(x),G->proc(x))*F->grad(x) +
         hg(F->proc(x),G->proc(x))*G->grad(x);
}
```

using the partial derivative methods `hf` and `hg`. Thus, classes derived from `Blend` need not implement `proc`, `grad` and `procq`, but must implement the simpler method `h` and its partial derivatives `hf` and `hg`.

4.2 Code-Level Automatic Differentiation

Given a `proc` implementation, another technique for automatically generating the derivatives `grad` and `procq` is

to apply the automatic differentiation method to the `proc` procedure source code. Computational differentiation is an automatic differentiation applied to algorithms by declaring some variables as dependent and others as independent, and synthesizing the source code necessary to yield the derivatives of the dependent variables with respect to the independent variables. For example, recent tools exist (ADOL-C,ADIC) that differentiate C language source code [5, 3]. Performing automatic differentiation at compile time yields faster derivatives than automatically differentating at run time.

4.3 Numerical Differentiation

We can also use numerical techniques to evaluate the derivatives. Forward differencing of the spatial derivative is implemented as

$$F_{\mathbf{x}}(\mathbf{x},\mathbf{q}) = \begin{pmatrix} F(\mathbf{x}+\epsilon\mathbf{e}_0,\mathbf{q}) - F(\mathbf{x},\mathbf{q}), \\ F(\mathbf{x}+\epsilon\mathbf{e}_1,\mathbf{q}) - F(\mathbf{x},\mathbf{q}), \\ F(\mathbf{x}+\epsilon\mathbf{e}_2,\mathbf{q}) - F(\mathbf{x},\mathbf{q}) \end{pmatrix} \quad (7)$$

where $\mathbf{e}_i$ is a unit vector in the ith dimension direction. Using this notation, the parameter derivative can be similarly derived

$$F_{\mathbf{q}}(\mathbf{x},\mathbf{q}) = (\ldots, F(\mathbf{x},\mathbf{q}+\epsilon\mathbf{e}_i) - F(\mathbf{x},\mathbf{q}), \ldots)\,. \quad (8)$$

We have found that the constrained particle system remains stable even when numerical versions of $F_{\mathbf{x}}$ and $F_{\mathbf{q}}$ are used. The symbolic $F_{\mathbf{x}}$ runs about four times as fast as the numerical $F_{\mathbf{x}}$ because the forward differencing implementation calls the implicit surface function four times. Similarly, the symbolic $F_{\mathbf{q}}$ implementation is $|\mathbf{q}|$-times faster than its numerical version.

The virtual methods `grad` and `procq` of our `Implicit` object default to the forward differences approximations. Hence, we can add a new implicit surface model into our library as a black-box by implementing the method `proc`. This task is a less daunting than deriving $F_{\mathbf{q}}$ by hand, as has been previously suggested [14].

5 Parameterization

A second problem with using particles to manipulate complex implicit surface models is the management of the parameters $\mathbf{q}$ of the implicit surface model. This problem can be decomposed into two specific issues. The first issue is the conceptual disconnection between an object's intuitive parameters (such as location and orientation) and its actual parameters (such as the coefficients of an algebraic). The second issue is that there are often an overwhelmingly large number of free parameters, even for a moderately complex implicit surface model.

One problem we have found is that it is difficult to translate the ellipsoid

$$\begin{aligned} F(x,y,z,q_0,\ldots,q_9) \;=\; & q_4x^2 + q_5xy + q_6y^2 + \\ & q_7xz + q_8xy + q_9z^2 + \\ & q_1x + q_2y + q_3z + q_0. \end{aligned} \quad (9)$$

using the control particles. We have the ability to select which parameters the control particles affect, but identifying which of the ten quadric coefficients control translation is not intuitive.

In order to translate the ellipsoid by $\mathbf{o} = (o_x, o_y, o_z)$, we need to apply the domain transformation $F(x-o_x, y-o_y, z-o_z, \mathbf{q})$. Evaluating (9) and collecting terms shows that translation does not affect the parameters q_4 through q_9, but affects q_0 through q_3 as

$$\begin{aligned} q_0 \;\leftarrow\; & q_0 + q_4o_x^2 + q_5o_xo_y + q_6o_y^2 + \\ & q_7o_xo_z + q_8o_yo_z + q_9o_z^2 + \\ & q_1o_x + q_2o_y + q_3o_z, \end{aligned} \quad (10)$$

$$q_1 \;\leftarrow\; q_1 - 2q_4o_x - q_5o_y - q_7o_z, \quad (11)$$

$$q_2 \;\leftarrow\; q_2 - q_5o_x - 2q_6o_y - q_8o_z, \quad (12)$$

$$q_3 \;\leftarrow\; q_3 - q_7o_x - q_8o_y - 2q_9o_z. \quad (13)$$

Enabling only these four coefficients to be changed by the control particles actually causes the entire ellipsoid to deform instead of translate because (2) and (3) dissipate particle velocity evenly among the parameter velocities. The velocities of the ellipsoid parameters due to translation are in fact $\dot{\mathbf{q}} = (q_1o_x + q_2o_y + q_3o_z + q_4o_x^2 + q_5o_xo_y + q_6o_y^2 + q_7o_xo_z + q_8o_yo_z + q_9o_z^2, -2q_4o_x - q_5o_y - q_7o_z, -q_5o_x - 2q_6o_y - q_8o_z, -q_7o_x - q_8o_y - 2q_9o_z, 0, 0, 0, 0, 0, 0)$.

5.1 Adapters

We solve this problem with the construction of a special kind of operator called an adapter. Whereas operators are designed to remain in the model, adapters are temporary and are used to control the parameters of a model during interactive editing.

An operator creates its parameter vector from its parameters and the parameters of its operands. This assumes that the operator's parameters are independent of the operands' parameters (e.g. the radius of an offsetting operation). The parameters of an adapter are assumed to be related to a subset of the parameters of its operands. The parameter vector of the adapter contains only the parameters of the adapter, and ignores the parameters of the adapter's operands.

For example, the adapter `Mover` is defined by

```
Mover::proc(x) { return F->proc(x - o) }
```

where F is the operand of `Mover` and $\mathbf{o}$ is an offset vector. The parameters specific to `Mover` consist only of the

offset vector. If `Mover` was an ordinary operand, then its parameter vector $\mathbf{q}$ would be $\mathbf{o}$ catenated with whatever parameters operand object F may have. Assuming F has been sufficiently parameterized, the additional components to $\mathbf{q}$ offered by the offset vector $\mathbf{o}$ would be redundant.

Since `Mover` is an adapter, we mask all of its operand's parameters, such that `Mover::procq()` returns only $\mathbf{o}$. This restricted parameter vector allows the constrained particle system to move an object using a single control particle. But in order for the object to remain in its new location, the adapter must remain attached. One can imagine a model becoming quite complex with adapters every time a subset of the model needs to be positioned.

We can remove an adapter if its parameters are set to its identity configuration, (zeroed in the case of `Mover`). Hence we need a way of transferring changes in parameters in the adapter to parameters in the adapter's operand.

An adapter implements some deformation function $D : \mathbf{R}^3 \rightarrow \mathbf{R}^3$. We can apply the deformation D to the implicit surface of $F(\mathbf{x}, \mathbf{q}_0)$ as a domain transformation, yielding the implicit surface of $F(D^{-1}(\mathbf{x}), \mathbf{q}_0)$. We need to find a new set of parameters $\mathbf{q}_1$ such that

$$F(D^{-1}(\mathbf{x}), \mathbf{q}_0) = F(\mathbf{x}, \mathbf{q}_1) \;\; \forall \mathbf{x} \in \mathbf{R}^3. \qquad (14)$$

The adapter function D has its own set of parameters $\mathbf{q}_D$. (In the `Mover` example, $\mathbf{q}_D = \mathbf{o}$). Let the parameters of F be denoted $\mathbf{q}_F$. We can use the Jacobian $d\mathbf{q}_F/d\mathbf{q}_D$ to find how changes in $\mathbf{q}_D$ affect $\mathbf{q}_F$. But this Jacobian would need to be derived and implemented to interface between every adapter and every `Implicit` primitive and operator in the modeling system.

Equations (2) and (3) provide a more general solution. We construct a collection of control particles and apply the deformation D to them, which causes the effect of the distortion to be applied to the original parameters of F.

We create a special array of particles $\mathbf{p}^i$. Even though each particle constrains the surface to pass through a three-dimensional point, the constraint restricts only one degree of freedom, since the particle may freely move across the two degrees of freedom along the surface. Hence, the number of particles n in the array should be $|\mathbf{q}_F|$. We assume that $|\mathbf{q}_D| << |\mathbf{q}_F|$ and F is flexible enough to find the solution of a much less flexible deformation D.

We then solve a variation of (3) specifically for processing the effect of D into the parameters of F,

$$\sum_j \left(F^i_{\mathbf{q}} \cdot F^j_{\mathbf{q}}\right) \lambda^j = F^i_{\mathbf{x}} \cdot (D(\mathbf{p}^i) - \mathbf{p}^i) + \phi(F^i - F(\mathbf{p}^i_0, \mathbf{q}_0)). \qquad (15)$$

The resulting Lagrangian multipliers λ^j provide the solution as

$$\dot{\mathbf{q}}_F = -\sum_j \lambda^j F^j_{\mathbf{q}}. \qquad (16)$$

We have assumed no desired parameter velocity ($\dot{\mathbf{Q}} = 0$).

Equation 15 has a slightly different feedback term that allows $F(\mathbf{p}^i)$ to be fixed to an arbitrary value, instead of just zero as was the case in (3). This feedback term makes sure the values at the particles do not drift away from their original values, which are found by evaluating F at the pre-deformation particle locations $\mathbf{p}^i_0$. Hence we can place particles anywhere in space to capture the field of F instead of just its implicit surface.

We sprinkle these particles randomly in space instead of across the surface. The implicit surface of aF is the same as that of F for any $a \neq 0$. Using particles on the surface constrained to $F = 0$ could yield an aF result with $a \neq 1$[2].

5.2 Implementation

We implemented (15) and (16) in the `setq` method of the adapter. When a surface control particle is dragged, (2) and (3) determine new parameters for the adapter's deformation through Euler integration of $\dot{\mathbf{q}}$. These new adapter parameters are then set by the particle system calling `setq`.

The `setq` of the adapter performs the following algorithm.

1. Set the state of its deformation D to use the parameters from the parameter vector $\mathbf{q}$ passed to it.
2. Use the deformation D to evaluate (15) and (16) to find the resulting parameter velocity $\dot{\mathbf{q}}_F$ of its operand F.
3. Perform an Euler step on the velocity $\dot{\mathbf{q}}_F$ by adding a fraction of it to the operand's parameter vector returned by `F->getq`, and store the result back in the operand's parameter vector via `F->setq`.

Since the parameters have been passed from the adapter to its operand, the adapter then returns its parameters to their original state. Hence, when an adapter's control particle is moved on an implicit surface, the adapter's parameters remain fixed and its operand's parameters change instead. This is the primary difference between an adapter and an operator in our modeling system.

5.3 Results

Figure 3 demonstrates this process on the `Mover` adapter applied to an ellipsoid that was originally placed at the origin. We have placed the translation adapter on the object and dragged the resulting composite object with a single particle. The paremeters of the translation are automatically propagated to the parameters of the underlying implicit ellipsoid primitive.

[2]This is also an issue when constructing implicit surfaces using radial basis functions. One or more constraint points are placed inside or outside the desired surface to indicate a desired interior or a desired local surface orientation [13].

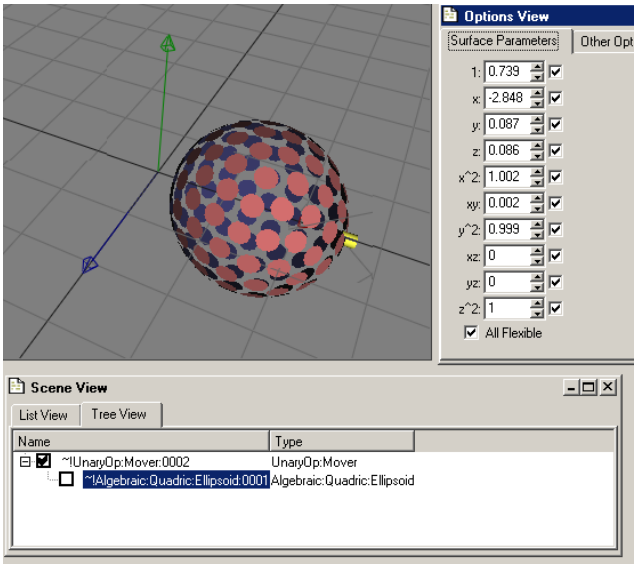


Figure 3. The effects of moving an ellipsoid by the single yellow particle on the coefficients of the quadric representation.

Careful examination of the parameters in Figure 3 reveals some numerical noise leaking into q_4, q_5 and q_6. This is most likely due to numerical error from the Euler integration. These inaccuracies may also contain some discretization error from the finite stochastic point-based sampling of the effects of the distortion.

This numerical noise causes the `Mover` operand to deform the ellipsoid as it translates. The feedback term is designed to reduce this distortion, but it is difficult to use this feedback term during the Euler integration because the randomly-positioned field particles do not actually move into their appropriate intermediate location as the parameter vector moves closer to the desired parameter vector. As a result, our implementation worked best when we took a large "predictor" step without feedback, followed by several "corrector" steps containing only the feedback term.

6 Conclusion

We have developed a implicit surface representation that is flexible enough to include a wide variety of different implicit modeling primitives and operations. We have built our system around a particle-based display and control system, and have explored several method for easily programming the derivatives needed for the particle dynamics and constraints.

We have also developed adapters which deform an implicit surface but embed the deformation in the parameter space of the implicit surface. These adapters provide the user with more intutive modeling parameters during interactive manipulation, but do not increase the complexity of the model.

6.1 The Implicit Modeling System

The implicit surface representation described in this paper is a subset of our actual system. Our full system serves as a base for our surface modeling research, and was designed to include a wide variety of different surface models. Figure 4 (generated automatically by *Doxygen* and *dot*) shows our present class hierarchy, demonstrating the range of implicit surface representations this class supports. Implicit surfaces are only one of the base representation classes supported by the system.

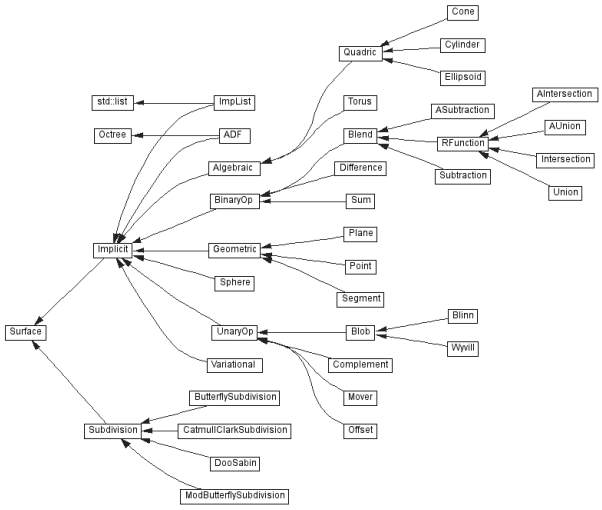


Figure 4. A current snapshot of our `Implicit` class hierarchy.

In the full system, our implicit objects include a second-derivative Hessian matrix for interrogating the curvature of the implicit surface. We plan to use the curvature to vary the size of particles across the surface, such that areas of high curvature receive many smaller particles. The `Implicit` class also includes interval extensions of many of the methods, to provide guarantees on properties over regions of space. We also have developed a toolkit for interval-based root finding and plan to implement topological polygonization guarantees on a larger selection of implicit surfaces than has previously before been accomplished [12].

Our goal for this system is to provide a publically-available open-source common environment for implicit surface research. The environment is available online at:

`http://graphics.cs.uiuc.edu/projects/surface`

The core of this representation was a group project of a class on advanced surface modeling taught in the Fall Semester 2000 at the University of Illinois. Each of the students was assigned a component of the library to implement as a project for the class. This format for a class project had several advantages. The student projects were not discarded at the end of class, which gives the students a stronger sense of accomplishment. The student projects were also distinct, which encouraged cooperation and teamwork instead of competition among the students. The interdependencies among the components of this library meant that it was in a student's best interest to help any other student that might be falling behind. This exercise provided production programming experience in an environment similar to the one many will find in their first jobs.

6.2 Future Work

The application of program differentiation tools on a given code segment is itself a complex task. It would be useful to develop a simpler subset of existing program differentation tools specifically for automatically translating `proc` methods into `grad` and `procq` methods.

The ability to "flatten" chains of multiple operators into a single operators is often employed in other hierarchical models in computer graphics. Depending on the success of implementations on implicit surfaces, it may be interesting to reparameterize the interfaces of other shape representations.

We have implemented the `Mover` adapter to verify the derivations in Section 5. We are in the process of implementing other adapters to perform deformations such as scale, rotation, taper, twist and bend. Barr [2] introduced the latter non-affine deformations, and it will be interesting to see how well some implicits, such as high-degree algebraics, can simulate the deformation effects within their parameterizatons.

Implicit surfaces are still *slippery* [14]. We have found that it is much easier to "pull" a convex surface than to "push" it. The *slipperiness* of the surface appears related to the flow gradient of the surface in the direction of the user-exerted force on a control particle. Constraining a control particle to a given position on the surface relative to nearby features could reduce the slippery feel of this method of modeling.

6.3 Acknowledgments

The core of our implicit surface library was coded by CS497JCH students Ed Bachta, Lennie Brown, Nate Carr, Jeff Decker, Bill Nagel and Steve Zelinka. Bill Lorensen, Will Schroeder and Ross Whitaker provided valuable insights into object oriented libraries from their experience with the vtk project. This research is supported in part by the NSF grant CCR-0196226 and the University of Illinois Department of Computer Science.

References

[1] V. Adzhiev, R. Cartwright, E. Fausett, A. Ossipov, A. Pasko, and V. Savchenko. Hyperfun project: a framework for collaborative multidimensional f-rep modeling. *Proc. Implicit Surfaces '99*, pages 59–69, Sept. 1999.

[2] A. H. Barr. Global and local deformations of solid primitives. *Computer Graphics*, 18(3):21–30, July 1984.

[3] C. H. Bischof, L. Roh, and A. J. Mauer-Oats. ADIC: an extensible automatic differentiation tool for ANSI-C. *Software: Practice and Experience*, 27(12):1427–1456, 1997.

[4] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[5] A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A package for the automatic differentiation of algorithms written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, June 1996.

[6] C. Hoffman and J. Hopcroft. Automatic surface generation in computer aided design. *Visual Computer*, 1:92–100, 1985.

[7] A. Pasko, V. Adzhiev, A. Sourin, and V. Savchenko. Function representation in geometric modeling: concepts, implementation and applications. *Visual Computer*, 11:429–446, 1995.

[8] H. K. Pedersen. imp. Source code available via implicit.eecs.wsu.edu, 1997.

[9] W. Schroeder, W. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. *Proc. Visualization '94*, pages 40–45, Oct. 1994.

[10] W. Schroeder, K. Martin, and W. Lorensen. *The Visualization Toolkit: An Object-Oriented Appriach to 3D Graphics*. Prentice Hall, Dec. 1997.

[11] W. J. Schroeder, K. M. Martin, and W. E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. *IEEE Visualization '96*, pages 93–100, Oct. 1996.

[12] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Computer Graphics (Annual Conference Series)*, pages 279–286, Aug. 1997.

[13] G. Turk and J. O'Brien. Shape transformation using variational implicit functions. *Computer Graphics (Proc. SIGGRAPH 99)*, pages 335–342, Aug. 1999.

[14] A. P. Witkin and P. S. Heckbert. Using particles to sample and control implicit surfaces. In *Computer Graphics (Annual Conference Series)*, pages 269–277, July 1994.

[15] B. Wyvill, E. Galin, and A. Guy. Extending the CSG tree: Warping, blending and boolean operations in an implicit surface modeling system. *Computer Graphics Forum*, 18(2):149–158, June 1999.

Isosurfaces and Level-Set Surface Models [a]

Ross T. Whitaker

Technical Report UUCS-02-010

[a] Versions of these notes and the accompanying talk appeared in tutorials at IEEE Visualization 2000 and 2001, and ACM SIGGRAPH 2001 and 2002.

School of Computing
University of Utah
Salt Lake City, UT 84112 USA

April 3, 2002

Abstract

This paper is a set of notes that present the basic geometry of isosurfaces and the basic methods for using level sets to model deformable surfaces. It begins with a short introduction to isosurface geometry, including curvature. It continues with a short explanation of the level-set partial differential equations. It also presents some practical details for how to solve these equations using up-wind scheme and sparse calculation methods. This paper presents a series of examples of how level-set surface models are used to solve problems in graphics and vision. Finally, it presents some examples of implementations using *VISPack*, an object oriented, C++ library for doing volume processing and level-set surface modeling.

1 Introduction

1.1 Motivation

These notes address mechanisms for analyzing and processing volumes in a way that deals specifically with *isosurfaces*. The underlying philosophy is to use isosurfaces as a modeling technology that can serve as an alternative to parameterized models for a variety of important applications in visualization and computer graphics. This paper presents the mathematics and numerical techniques for describing the geometry of isosurfaces and manipulating their shapes in prescribed ways. We start with a basic introduction into the notation and fundamental concepts and then presents the geometry of isosurfaces. We describe the method of level sets, i.e., moving isosurfaces, and present the mathematical and numerical methods they entail. This paper concludes with some application examples and describes *VISPACK*, a *C++*, object-oriented library the performs volume processing and level-set modeling.

1.2 Isosurfaces

1.2.1 Modeling Surfaces With Volumes

When considering surface models for graphics and visualization, one is faced with a staggering variety of options including meshes, spline-based patches, constructive solid geometry, implicit blobs, and particle systems. These options can be divided into two basic classes — explicit (parameterized) models and implicit models. With an implicit model, one specifies the model as a *level set* of a scalar function,

$$\begin{array}{rccc} \phi: & U & \mapsto & \mathbb{R} \\ & x, y, z & & k \end{array}, \tag{1}$$

where $U \subset \mathbb{R}^3$ is the domain of the volume (and the *range* of the surface model). Thus, a surface $\mathcal{S}$ is

$$\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\} . \tag{2}$$

The choice of k is arbitrary, and ϕ is sometimes called the *embedding*. Notice that surfaces defined in this way divide U into a clear inside and outside—such surfaces are always closed wherever they do not intersect the boundary of the domain.

Choosing this implicit strategy begs the question of how to represent ϕ. Historically, implicit models are represented using linear combinations of *basis* functions. These basis or

potential functions usually have several degrees of freedom including 3D position, size, and orientation. By combining these functions, one can create complex objects. Typical models might contain several hundred to several thousands of such primitives. This is the strategy behind the "blobby" models proposed by Blinn [1].

While such an implicit modeling strategy offers a variety of new modeling tools, it has some limitations. In particular, the global nature of the potential functions limits ones ability to model *local* surface deformations. Consider a point $\boldsymbol{x} \in \mathcal{S}$ where $\mathcal{S}$ is the level surface associated with a model $\phi = \sum_i \alpha_i$, and α_i is one of the individual potential functions that comprise that model. Suppose one wishes to move the surface at the point $\boldsymbol{x}$ in a way that maintains continuity with the surrounding neighborhood. With multiple, global basis functions one must decide which basis function or combination of basis functions to alter and at the same time control the effects on other parts of the surface. The problem is generally ill posed — there are many ways to adjust the basis functions so that $\boldsymbol{x}$ will move in the desired direction and yet it may be impossible to eliminate the effects of those movements on other disjoint parts of the surface. These problems can be overcome, however they usually entail heuristics that tie the behavior of the surface deformation to the choice of representation [2].

An alternative to using a small number of *global* basis functions is to use a relatively large number of *local* basis functions. This is the principle behind using a volume as an implicit model. A volume is a discrete sampling of the embedding ϕ. It is also an implicit model with a very large number of basis functions, as shown in Figure 1. The total number of basis functions is fixed, as are their positions (grid points) and extent. One can change only the magnitude of each basis function, i.e., each basis function has only one degree of freedom. A typical volume of size $128 \times 128 \times 128$ contains over a million such basis functions. The shape of each basis function is open to interpretation — it depends on how one interpolates the values between the grid points. A trilinear interpolation, for instance, implies a basis function that is a piece-wise cubic polynomial with a value of one at the grid point and zero at neighboring grid points. Another advantage of using volumes as implicit models, is that for the purposes of analysis we can treat the volume as a continuous function whose values can be *set* at each point according to the application. Once the continuous analysis is complete we can map the algorithm into the discrete domain using standard methods of numerical analysis. The sections that follow discuss how to compute the geometry of surfaces that are represented as volumes and how to manipulate the shapes of those surfaces by changing the gray-scale values in the volume.

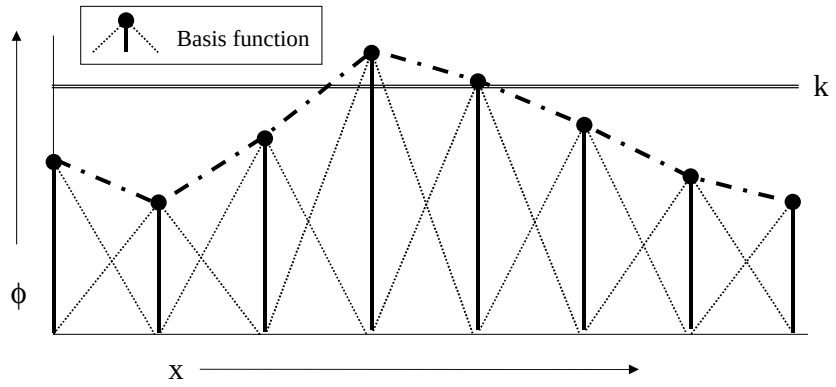


Figure 1: A volume can be considered as an implicit model with a large number of local basis functions.

1.2.2 Isosurface Extraction and Visualization

This paper addresses the question of how to use volumes as surface models. Depending on the application, however, a 3D grid of data (i.e. a volume) may not be a suitable model representation. For instance, if the goal is make measurements of an object or visualize its shape, an explicit model might be necessary. In such cases it is beneficial to convert between volumes and other modeling technologies.

For instance, the literature proposes several methods for scan converting polygonal meshes or solid models [3, 4]. Likewise a variety of methods exist for extracting parametric models of isosurfaces from volumes. The most prevalent method is to locate isosurface crossings along grid lines in a volume (between voxels along the 3 cardinal directions) and then to link these points together to form triangles and meshes. This is the strategy of "marching cubes" [5] and other related approaches. However, extracting a parametric surface is not essential for visualization, and a variety of direct methods [6, 7] are now computationally feasible and arguably superior in quality. These notes do not address the issue of extracting or rendering isosurfaces, but rather studies the geometry of isosurfaces and how to manipulate them directly by changing the grey-scale values in the underlying volume. Thus, we propose volumes as a mechanism for studying and deforming surfaces, regardless of the ultimate form of the output. Their are many ways of rendering or visualizing them and and these techniques are beyond the scope of this discussion.

2 Surface Normals

The surface normal of an isosurface is given by the normalized gradient vector. Typically, we identify a surface normal with a point in the volume domain D. That is

$$\boldsymbol{n}(\boldsymbol{x}) = \frac{\nabla\phi(\boldsymbol{x})}{|\nabla\phi(\boldsymbol{x})|} \quad \text{where} \quad \boldsymbol{x} \in D. \tag{3}$$

The convention regarding the direction of this vector is arbitrary; the negative of the normalized gradient magnitude is also normal to the isosurface. The gradient vector points toward that side of the isosurface which has greater values (i.e. brighter). When rendering, the convention is to use *outward pointing* normals, and the sign of the gradient must be adjusted accordingly. However, for most applications any consistent choice of normal vector will suffice. On a discrete grid, one must also decide how to approximate the gradient vector (i.e., first partial derivatives). In many cases central differences will suffice. However, in the presence of noise, especially when volume rendering, it is sometimes helpful to compute first derivatives using some smoothing filter (e.g., convolution with a Gaussian). When

using the normal vector to solve certain kinds of partial differential equations, it is sometimes necessary to approximate the gradient vector with discrete, one-sided differences, as discussed in successive sections.

Note that a single volume contains families nested isosurfaces, arranged like the layers of an onion. We specific the normal to an isosurface as a function of the position within the volume. That is, $\boldsymbol{n}(\boldsymbol{x})$ is the normal of the (single) isosurface that passes through the point $\boldsymbol{x}$. The k value associated with that isosurface is $\phi(\boldsymbol{x})$.

3 Second-Order Structure

In differential geometric terms, the second-order structure of a surface is characterized by a quadratic patch that shares first- and second-order contact with the surface at a point (i.e., tangent plane and osculating circles). The *principal directions* of the surface are those associated with the quadratic approximation, and the *principal curvatures*, k_1, k_2, are the curvatures in those directions.

The second-structure of the isosurface can be computed from the first- and second-order structure of the embedding, ϕ. All of the isosurface shape information is contained field of normals given by $\boldsymbol{n}(\boldsymbol{x})$. The 3×3 matrix of derivatives of this vector,

$$N = -\left[\boldsymbol{n}_x \;\; \boldsymbol{n}_y \;\; \boldsymbol{n}_z.\right] \tag{4}$$

The projection of this derivative onto the tangent plane of the isosurface gives the shape matrix, β. Let P denote normal projection operator, which is defined as

$$P = \boldsymbol{n} \otimes \boldsymbol{n} = \frac{1}{||\nabla\phi||^2} \begin{pmatrix} \phi_x^2 & \phi_x\phi_y & \phi_x\phi_z \\ \phi_y\phi_x & \phi_y^2 & \phi_y\phi_z \\ \phi_z\phi_x & \phi_z\phi_y & \phi_z^2 \end{pmatrix}. \tag{5}$$

The tangential projection operator is $I - P$, and thus the shape matrix is

$$\beta = NT = TH_\phi T, \tag{6}$$

where H_ϕ is the Hessian of ϕ. The shape matrix β has 3, real, eigenvalues which are

$$e_1 = k_1, e_2 = k_2, e_3 = 0. \tag{7}$$

The corresonding eigenvectors are the principle directions (in the tangent plane) and the normal, respectively.

The *mean curvature* is the mean of the two principal curvatures, which is one half of the trace of β, which is equal to the trace of N:

$$\begin{aligned} H &= \frac{k_1 + k_2}{2} = \frac{1}{2}\mathrm{Tr(N)} \\ &= \frac{\phi_x^2(\phi_{yy} + \phi_{zz}) + \phi_y^2(\phi_{xx} + \phi_{zz}) + \phi_z^2(\phi_{xx} + \phi_{yy}) - 2\phi_x\phi_y\phi_{xy} - 2\phi_x\phi_z\phi_{xz} - 2\phi_y\phi_z\phi_{yz}}{2(\phi_x^2 + \phi_y^2 + \phi_z^2)^{3/2}} \end{aligned} \tag{8}$$

The *Gaussian curvature* is the product of the principal curvatures:

$$\begin{aligned} K &= k_1 k_2 = e_1 e_2 + e_1 e_3 + e_2 e_3 = 2\mathrm{Tr(N)}^2 - \frac{1}{2}||\mathrm{N}|| \\ &= \frac{\begin{array}{c}\phi_z^2(\phi_{xx}\phi_{yy} - \phi_{xy}\phi_{xy}) + \phi_y^2(\phi_{xx}\phi_{zz} - \phi_{xz}\phi_{xz}) + \phi_x^2(\phi_{yy}\phi_{zz} - \phi_{yz}\phi_{yz}) \\ + 2(\phi_x\phi_y(\phi_{xz}\phi_{yz} - \phi_{xy}\phi_{zz}) + \phi_x\phi_z(\phi_{xy}\phi_{yz} - \phi_{xz}\phi_{yy}) + \phi_y\phi_z(\phi_{xy}\phi_{xz} - \phi_{yz}\phi_{xx}))\end{array}}{(\phi_x^2 + \phi_y^2 + \phi_z^2)^2}. \end{aligned} \tag{9}$$

The total curvature, also called the deviation from flatness, D, is the root sum of squares of the two principal curvatures, which is the Euclidean norm of the matrix β.

Notice, these measures exist at every point in U, and at each point they describe the geometry of the particular isosurface that passes through that point. All of these quantities can be computed on a discrete volume using finite differences, as described in successive sections.

4 Deformable Surfaces

This section begins with mathematics for describing surface deformations on parametric models. The result is an evolution equation for a surface. Each of the terms in this evolution equation can be re-expressed in a way that is independent of the parameterization. Finally, the evolution equation for a parametric surface gives rise to an evolution equation (differential equation) on a volume, which encodes the shape of that surface as a level set.

4.1 Surface Deformation

A regular surface $\mathcal{S} \subset \mathbb{R}^3$ is a collection of points in 3D that can be be represented *locally* as a continuous function. In geometric modeling a surface is typically represented as a two-parameter object in a three-dimensional space, i.e., a surface is local a mapping $\boldsymbol{S}$:

$$\boldsymbol{S} : \underset{r}{V} \times \underset{s}{V} \mapsto \underset{x,\, y,\, z,}{\mathbb{R}^3} \tag{10}$$

where $V \times V\mathbb{R}^2$, and the bold notation refers specifically to a parameterized surface (vector-valued function). A deformable surface exhibits some motion over time. Thus $\boldsymbol{S} = \boldsymbol{S}(r, s, t)$, where $t \in \mathbb{R}^+$. We assume second-order-continuous, orientable surfaces; therefore at every point on the surface (and in time) there is surface normal $\boldsymbol{N} = \boldsymbol{N}(r, s, t)$. We use $\mathcal{S}_t$ to refer to the entire set of points on the surface.

Local deformations of $\boldsymbol{S}$ can be described by an evolution equation, i.e., a differential equation on $\boldsymbol{S}$ that incorporates the position of the surface, local and global shape properties, and responses to other forcing functions. That is,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{G}\left(\boldsymbol{S}, \boldsymbol{S}_r, \boldsymbol{S}_s, \boldsymbol{S}_{rr}, \boldsymbol{S}_{rs}, \boldsymbol{S}_{ss}, \ldots\right), \tag{11}$$

where the subscripts represent partial derivatives with respect to those parameters. The evolution of $\boldsymbol{S}$ can be described by a sum of terms that depends on both the geometry of $\boldsymbol{S}$ and the influence of other functions or data.

There are a variety of differential expressions that can be combined for different applications. For instance, the model could move in response to some directional "forcing" function [8, 9], $\boldsymbol{F} : U \mapsto \mathbb{R}^3$, that is

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{F}(\boldsymbol{S}). \tag{12}$$

Alternatively, the surface could expand and contract with a spatially-varying speed. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = G(\boldsymbol{S})\boldsymbol{N} \tag{13}$$

where $G : \mathbb{R}^3 \mapsto \mathbb{R}$ is a signed speed function. The evolution might also depend on the surface geometry itself. For instance,

$$\frac{\partial \boldsymbol{S}}{\partial t} = \boldsymbol{S}_{rr} + \boldsymbol{S}_{ss} \tag{14}$$

describes a surface that moves in way that is becomes more *smooth* with respect to its own parameterization. This motion can be combined with the motion of Equation 12 to produce a model that is pushed by a forcing function but maintains a certain smoothness in its shape and parameterization. There are myriad terms that depend on both the differential geometry of the surface and outside forces or functions to control the evolution of a surface.

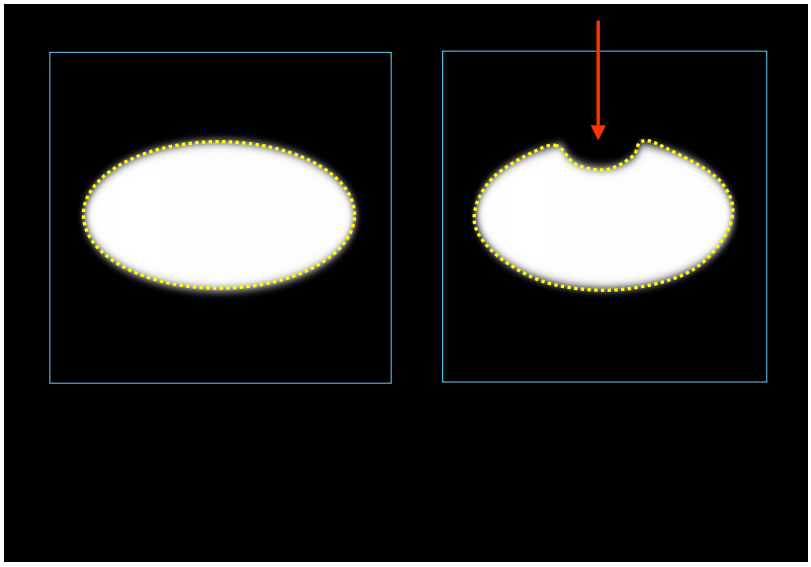


Figure 2: Level-set models represent curves and surfaces implicitly using greyscale images: a) an ellipse is represented as the level set of an image, b) to change the shape we modify the greyscale values of the image.

5 Deformation: The Level Set Approach

The method of level-sets, proposed by Osher and Sethian [10] and described extensively in [11], provides the mathematical and numerical mechanisms for computing surface deformations as time-varying iso-values of ϕ by solving a partial differential equation on the 3D grid. That is, the level-set formulation provides a set of numerical methods that describe how to manipulate the greyscale values in a volume, so that the isosurfaces of ϕ move in a prescribed manner (shown in Figure 2).

We denote the movement of a point on a surface as it deforms as $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$, and we assume that this motion can be expressed in terms of the position of $\boldsymbol{x} \in U$ and the geometry of the surface at that point. In this case, there are generally two options for representing such surface movements implicitly:

Static: A single, static $\phi(\boldsymbol{x})$ contains a family of level sets corresponding to surfaces as different times t. That is,

$$\phi(\boldsymbol{x}(t)) = k(t) \quad \Rightarrow \quad \nabla\phi(\boldsymbol{x}) \cdot \frac{\partial \boldsymbol{x}}{t} = \frac{\mathrm{d}k(t)}{\mathrm{d}t}. \tag{15}$$

To solve this static method requires constructing a ϕ that satisfies equation 15. This is a boundary value problem, which can be solved somewhat efficiently starting with a single surface using the fast marching method of Sethian [12]. This representation has some significant limitations, however, because (by definition) a surface cannot pass back over itself over time, i.e., motions must be strictly monotonic — inward or outward.

Dynamic: The approach is to use a one-parameter *family* of embeddings, i.e., $\phi(\boldsymbol{x}, t)$ changes over time, $\boldsymbol{x}$ remains on the k level set of ϕ as it moves, and k remains constant. The behavior of ϕ is obtained by setting the total derivative of $\phi(\boldsymbol{x}(t), t) = k$ to zero. Thus,

$$\phi(\boldsymbol{x}(t), t) = k \;\; \Rightarrow \;\; \frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}. \tag{16}$$

This approach can accommodate models that move forward and backward and cross back over their own paths (over time). However, to solve this requires solving the initial value problem (using finite forward differences) on $\phi(\boldsymbol{x}, t)$ — a potentially large computational burden. The remainder of this discussion focuses on the dynamic case, because of its superior flexibility.

All surface movements depend on position and geometry, and the level-set geometry is expressed in terms of the differential structure of ϕ. Therefore the dynamic formulation from equation 16 gives a general form of the partial differential equation on ϕ:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = -\nabla \phi \cdot \boldsymbol{F}(\boldsymbol{x}, \mathrm{D}\phi, \mathrm{D}^2\phi, \ldots), \tag{17}$$

where $D^n\phi$ is the set of order-n derivatives of ϕ evaluated at $\boldsymbol{x}$. Because this relationship applies to every level-set of ϕ, i.e. all values of k, this equation can be applied to all of U, and therefore the movements of *all* the level-set surfaces embedded in ϕ can be calculated from Equation 17.

The level-set representation has a number of practical and theoretical advantages over conventional surface models, especially in the context of deformation and segmentation. First, level-set models are topologically flexible, they can easily represent complicated surface shapes that can, in turn, form holes, split to form multiple objects, or merge with other objects to form a single structure. These models can incorporate many (millions) of degrees of freedom, and therefore they can accommodate complex shapes. Indeed, the shapes formed by the level sets of ϕ are restricted only by the resolution of the sampling. Thus, there is no need to reparameterize the model as it undergoes significant deformations.

Such level-set methods are well documented in the literature [10, 13] for applications such as computational physics [14], image processing [15, 16], computer vision [17, 18], medical image analysis [19, 18], and 3D reconstruction [20, 21]. For instance, in computational

physics level-set methods are a a powerful tool for modeling moving interfaces between different materials (see Osher and Fedkiw [14] for a nice overview of recent results). Examples are water-air and water-oil. In such cases, level-set methods can be used to compute deformations that minimize surface area while preserving volumes for materials that *split and merge* in arbitrary ways. The method can be extended to multiple, non-overlapping objects.

Level-set methods have also been shown to be effective in extracting surface structures from biological and medical data. For instance Malladi *et al.* [18] propose a method in which the level-sets form an expanding or contracting contour which tends to "cling" to interesting features in 2D angiograms. At the same time the contour is also influenced by its own curvature, and therefore remains smooth. Whitaker *et al.* [19, 22] have shown that level sets can be used to simulate conventional deformable surface models, and demonstrated this by extracting skin and tumors from *thick-sliced* (e.g. clinical) MR data, and by reconstructing a fetal face from 3D ultrasound. A variety of authors [23, 24, 16, 25] have presented variations on the method and presented results for 2D and 3D data. Sethian [11] gives several examples of level-set curves and surface for segmenting CT and MR data.

5.1 Deformation Modes

In the case of parametric surfaces, one can choose from a variety of different expressions to construct an evolution equation that is appropriate for a particular application. For each of those parametric expressions, there is a corresponding expression that can be formulated on ϕ, the volume in which the level-set models are embedded. In constructing evolutions on levels sets, there can be no reference to the underlying surface parameterization (terms depending on r and s in Equations 10 through 14). This has two important implications: 1) only those surface movements that are normal to the surface are represented—any other movement is equivalent to a reparameterization 2) all of the derivatives with respect to surface parameters r and s must be expressed in terms of invariant surface properties that can be derived without a parameterization.

Consider the term $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ from equation 14. If r, s is an orthonormal parameterization, the effect of that term is based purely on surface shape, not on the parameterization, and the expression $\boldsymbol{S}_{rr} + \boldsymbol{S}_{ss}$ is twice the *mean curvature*, H, of the surface. The corresponding level-set formulation is given by Equation 8.

Table 1 shows a list of expressions used in the evolution of parameterized surfaces and their equivalents for level-set representations. Also given are the assumptions about the parameterization that give rise to the level-set expressions.

	Effect	Parametric Evolution	Level-Set Evolution	Parameter Assumptions
1	External force	$\boldsymbol{F}$	$\boldsymbol{F} \cdot \nabla\phi$	None
2	Expansion/ contraction	$G(\boldsymbol{x})\boldsymbol{N}$	$G(\boldsymbol{x})\|\nabla\phi(\boldsymbol{x},t)\|$	None
3	Mean curvature	$S_{rr} + S_{ss}$	$H\|\nabla\phi\|$	Orthonormal
4	Gauss curvature	$S_{rr} \times S_{ss}$	$K\|\nabla\phi\|$	Orthonormal
5	Second order	S_{rr} or S_{ss}	$\left(H \pm \sqrt{H^2 - K}\right)\|\nabla\phi\|$	Principal curvatures

Table 1: A list of evolution terms for parametric models has a corresponding expression on the embedding, ϕ, associated with the level-set models.

6 Numerical Methods

By taking the strategy of embedding surface models in volumes, we have converted equations that describe the movement of surface points to nonlinear, partial differential equations defined on a volume, which is generally a rectilinear grid. The expression $u^n_{i,j,k}$ refers to the nth time step at position i, j, k, which has an associated value in the 3D domain of the continuous volume $\phi(x_i, y_j, z_k)$. The goal is to solve the differential equation consisting of terms from Table 5.1 on the discrete grid $u^n_{i,j,k}$.

The discretization of these equations raises two important issues. First is the availability of accurate, stable numerical schemes for solving these equations. Second is the problem of computational complexity and the fact that we have converted a *surface* problem to a *volume* problem, increasing the dimensionality of the domain over which the evolution equations must be solved.

The level-set terms in Table 1 are combined, based on the needs of the application, to create a partial differential equation on $\phi(\boldsymbol{x}, t)$. The solutions to these equations are computed using finite differences. Along the time axis solutions are obtained using finite *forward* differences, beginning with an initial model (i.e., volume) and stepping sequentially through a series of discrete times steps (which are denoted as superscripts on u). Thus the update equation is:

$$u^{n+1}_{i,j,k} = u^n_{i,j,k} + \Delta t \Delta u^n_{i,j,k}, \tag{18}$$

The term $\Delta u^n_{i,j,k}$ is a discrete approximation to $\partial\phi/\partial t$, which consists of a weighted sum

of terms such as those in Table 5.1. Those terms must, in turn, be approximated using finite differences on the volume grid.

6.1 Up-wind Schemes

The terms in Table 1 fall into two basic categories: the first-order terms (items 1 and 2 in Table 1) and the second-order terms (items 3 through 5). The first-order terms describe a moving wave front with a space-varying velocity (expression 1) or speed (expression 2). Equations of this form cannot be solved with a simple finite forward difference scheme. Such schemes tend to overshoot, and they are unstable. To address this issue Osher and Sethian [26] have proposed an *up-wind* scheme. The up-wind method relies on a one-sided derivative that looks in the up-wind direction of the moving wave front, and thereby avoids the over-shooting associated with finite forward differences.

We denote the type of discrete difference using superscripts on a difference operator, i.e., $\delta^{(+)}$ for forward differences, $\delta^{(-)}$ for backward differences, and δ for central differences. For instance, differences in the x direction on a discrete grid, $u_{i,j,k}$, with domain X and uniform spacing h are defined as

$$\delta_x^{(+)} u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i,j,k})/h, \tag{19}$$

$$\delta_x^{(-)} u_{i,j,k} \triangleq (u_{i,j,k} - u_{i-1,j,k})/h, \quad \text{and} \tag{20}$$

$$\delta_x u_{i,j,k} \triangleq (u_{i+1,j,k} - u_{i-1,j,k})/(2h), \tag{21}$$

$$\tag{22}$$

where we have left off the time superscript for conciseness. Second-order terms are computed using the *tightest-fitting* central difference operators. For example,

$$\delta_{xx} u_{i,j,k} \triangleq \left(u_{i+1,j,k} + u_{i-1,j,k} - 2u_{i,j,k}\right)/h^2, \tag{23}$$

$$\delta_{zz} u_{i,j,k} \triangleq \left(u_{i,j,k+1} + u_{i,j,k-1} - 2u_{i,j,k}\right)/h^2, \quad \text{and} \tag{24}$$

$$\delta_{xy} u_{i,j,k} \triangleq \delta_x \delta_y u_{i,j,k} \tag{25}$$

The discrete approximation to the first-order terms of in Table 5.1 are computed using the up-wind proposed by Osher and Sethian [10]. This strategy avoids overshooting by approximating the gradient of ϕ using a one-sided differences in the direction that is up-wind of the moving level-set thereby ensuring that no *new* contours are created in the process of updating $u_{i,j,k}^n$ (as depicted in Figure 3). The scheme is separable along each axis (i.e., x, y, and z).

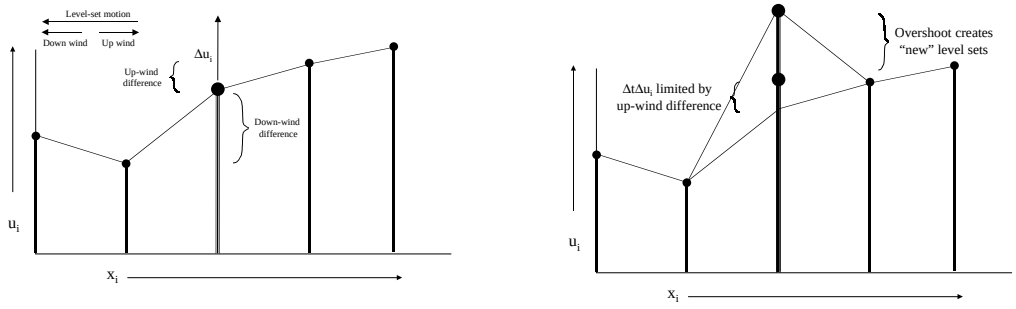


Figure 3: The up-wind numerical scheme uses one-sided derivatives to prevent overshooting and the creation of new level sets.

Consider Term 1 in Table 5.1. If we use superscripts to denote the vector components, i.e.,

$$\boldsymbol{F}(x,y,z) = (F^{(x)}(x,y,z), F^{(y)}(x,y,z), F^{(z)}(x,y,z)), \tag{26}$$

the up-wind calculation for a grid point $u^n_{i,j,k}$ is

$$\boldsymbol{F}(x_i,y_i,z_i)\cdot\nabla\phi(x_i,y_j,z_k,t) \approx \sum_{q\in\{x,y,z\}} F^{(q)}(x_i,y_i,z_i) \begin{cases} \delta_q^+ u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) > 0 \\ \delta_q^i u^n_{i,j,k} & F^{(q)}(x_i,y_i,z_i) < 0 \end{cases} \tag{27}$$

The time steps are limited—the fastest moving wave front can move only one grid unit per iteration. That is

$$\Delta t_{\boldsymbol{F}} \leq \frac{1}{\sum_{q\in\{x,y,z\}} \sup_{i,j,k\in X}\{|\nabla F^{(q)}(x_i,y_j,z_k)|\}}. \tag{28}$$

For Term 2 in Table 5.1 the direction of the moving surface depends on the normal, and therefore the same up-wind strategy is applied in a slightly different form.

$$\begin{aligned} &G(x_i,y_j,z_k)|\nabla\phi(x_i,y_j,z_k,t)| \approx \\ &\sum_{q\in\{x,y,z\}} G(x_i,y_i,z_i) \begin{cases} \max^2(\delta_q^+ u^n_{i,j,k},0) + \min^2(\delta_q^- u^n_{i,j,k},0) & G(x_i,y_i,z_i) > 0 \\ \min^2(\delta_q^+ u^n_{i,j,k},0) + \max^2(\delta_q^- u^n_{i,j,k},0) & G(q)(x_i,y_i,z_i) < 0 \end{cases} \end{aligned} \tag{29}$$

The time steps are, again, limited by the fastest moving wave front:

$$\Delta t_G \leq \frac{1}{3\sup_{i,j,k\in X}\{|\nabla G(x_i,y_j,z_k)|\}} \tag{30}$$

To compute approximation the update to the second-order terms in Table 5.1 requires only central differences . Thus, the mean curvature is approximated as:

$$\begin{aligned} H^n_{i,j,k} \;=\; & \frac{1}{2}\left(\left(\delta_x u^n_{i,j,k}\right)^2+\left(\delta_y u^n_{i,j,k}\right)^2+\left(\delta_z u^n_{i,j,k}\right)^2\right)^{-1}\Big[\left(\left(\delta_y u^n_{i,j,k}\right)^2+\left(\delta_z u^n_{i,j,k}\right)^2\right)\delta_{xx}u^n_{i,j,k} \\ & +\left(\left(\delta_z u^n_{i,j,k}\right)^2+\left(\delta_x u^n_{i,j,k}\right)^2\right)\delta_{yy}u^n_{i,j,k}+\left(\left(\delta_x u^n_{i,j,k}\right)^2+\left(\delta_y u^n_{i,j,k}\right)^2\right)\delta_{zz}u^n_{i,j,k} \\ & -2\delta_x u^n_{i,j,k}\delta_y u^n_{i,j,k}\delta_{xy}u^n_{i,j,k}-2\delta_y u^n_{i,j,k}\delta_z u^n_{i,j,k}\delta_{yz}u^n_{i,j,k}-2\delta_z u^n_{i,j,k}\delta_x u^n_{i,j,k}\delta_{zx}u^n_{i,j,k}\Big] \end{aligned} \tag{31}$$

Such curvature terms can be computing by using a combination of forward and backward differences as described in [27]. In some cases this is advantageous—but the details are beyond the scope of this paper.

The time steps are limited, for stability, to

$$\Delta t_H \leq \frac{1}{6}. \tag{32}$$

When combining terms, the maximum time steps for each terms is scaled by one over the weighting coefficient for that term.

6.2 Narrow-Band Methods

If one is interested in only *a single level set*, the formulation described previously is not efficient. This is because solutions are usually computed over the entire domain of ϕ. The solutions, $\phi(x, y, z, t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

Fortunately, the evolution of a single level set, $\phi(\boldsymbol{x}, t) = k$, is not affected by the choice of embedding. The evolution of the level sets is such that they evolve independently (to within the error introduced by the discrete grid). Furthermore, the evolution of ϕ is important only in the vicinity of that level set. Thus, one should perform calculations for the evolution of ϕ only in a neighborhood of the surface $\mathcal{S} = \{\boldsymbol{x} | \phi(\boldsymbol{x}) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see Figure 4). Of course, as the surface moves, that subset of grid points must change to account for its new position.

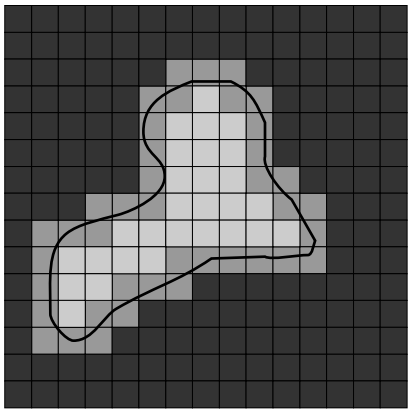


Figure 4: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.

Adalsteinson and Sethian [28] propose a *narrow-band* approach which follows this line of reasoning. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is truncated, i.e, computed over a finite width of only m points that lie within a specified distance to the level set. The remaining points are set to constant values to indicate that they do not lie within the narrow band, or *tube* as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by calculating the evolution of u only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band (see Figure 5), they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the kth level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of [28]) where one would like to have a band that is only as wide as necessary to calculate the derivatives of u near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the band, where some grid points are undergoing the evolution and nearby neighbors are static.

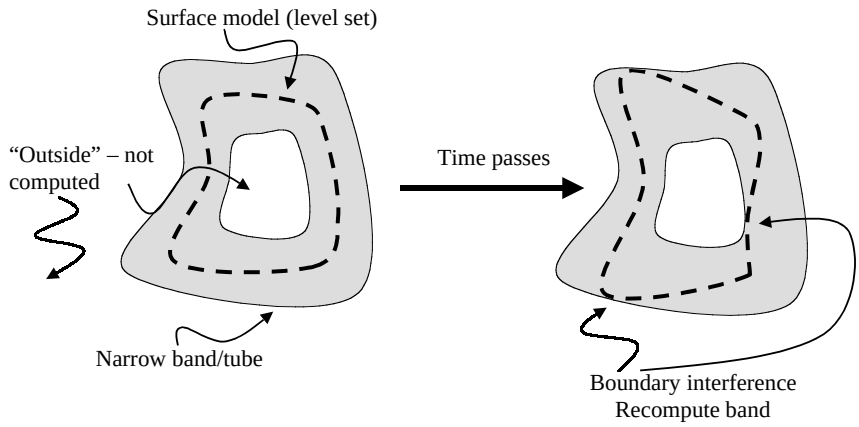


Figure 5: The narrow band scheme limits computation to the vicinity of the specific level set. As the level-set moves near the edge of the band the process is stopped and the band recomputed.

6.3 The Sparse-Field Method

The basic premise of the narrow band algorithm is that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process. The strategy proposed here is to use an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Computation of the evolution equation is computed on a band of grid points that is only on point wide. The embedding is extended from the active points to a neighborhood around those points that is precisely the width needed at each time. This extension is done via a fast distance transform approximation.

This approach has several advantages. First, the algorithm does precisely the number of calculations needed to compute the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. Because the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm visits only those points adjacent to the k-level curve. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive.

The *sparse-field* algorithm is analogous to a locomotive engine that lays down tracks before it and picks them up from behind. In this way the number of computations increases with the surface area of the model rather than the resolution of the embedding. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for various kinds of "model fitting" applications.

The sparse-field algorithm takes advantage of the fact that a k-level surface, S, of a discrete image u (of any dimension) has a set of cells through which it passes, as shown in Figure 4. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points.* As a first-order approximation, the distance of the level set from the center of any active point is proportional to the value of u divided the gradient magnitude at that point. Because all of the derivatives (up to second order) in this approach are computed using nearest neighbor differences, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process. The strategy is to compute the evolution given by equation 17 on the active set and then update neighborhood around the active set using a fast distance transform. Because active points must be adjacent to the level-set model, their positions lie within a fixed distance to the model. Therefore the values of u for locations in the active set must lie within a certain range. When active-point values move out of this *active range*

they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is important to the operation of the algorithm.

The values of the points in the active set can be updated using the up-wind scheme for first-order terms and central differences for the mean-curvature flow, as described in the previous sections. In order to maintain stability, one must update the neighborhoods of active grid points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. Grid points should be removed from the active set when they are no longer the nearest grid point to the zero crossing. If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, $x_m = (i, j, k)$, to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see Figure 6). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just x_m is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points pass out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. In [21] the author gives some formal definitions of active sets and the operations that affect them, which show that active sets will always form a boundary between positive and negative regions in the image, even as control of the level set passes from one set off active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control the behavior of non-active grid points to which they are adjacent. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots L_{+N}$ and $L_{-1}, \ldots L_{-N}$, where the i indicates the distance (city block distance) from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. Most of the level-set work relies on surface normals and curvature, which require only second-order derivatives of ϕ. Second-order derivatives are calculated using a $3 \times 3 \times 3$ kernel (city-block distance 2 to the corners). Therefore only five layers are necessary (2 inside layers, 2 outside layers, and the active set). These layers are denoted L_1, L_2, L_{-1}, L_{-2}, and L_0.

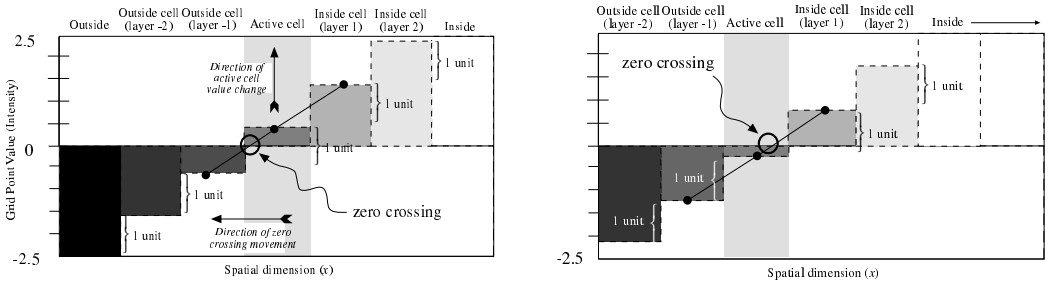


Figure 6: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed one grid point to another.

The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in Figure 6). Thus the values of layer L_i fall in the interval $[i - \frac{1}{2}, i + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are totally inside and outside are $N + \frac{1}{2}$ and $-N - \frac{1}{2}$, respectively. The procedure for updating the image and the active set based on surface movements is as follows:

1. For each active grid point, $x_m = (i, j, k)$, do the following:
   (a) Calculate the local geometry of the level set.
   (b) Compute the net change of u_{x_m}, based on the internal and external forces, using some stable (e.g., up-wind) numerical scheme where necessary.
2. For each active grid point x_j add the change to the grid point value and decide if the new value $u_{x_m}^{n+1}$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put x_m on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u_{x_m}^{n+1} > 1$ or $u_{x_m}^{n+1} < -1$, respectively.
3. Visit the grid points in the layers L_i in the order $i = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{i \mp 1}$. If more than one $L_{i \mp 1}$ neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_i has no $L_{i \mp 1}$ neighbors, then it gets demoted to $L_{i \pm 1}$, the next level away from the active set.
4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$ do the following:

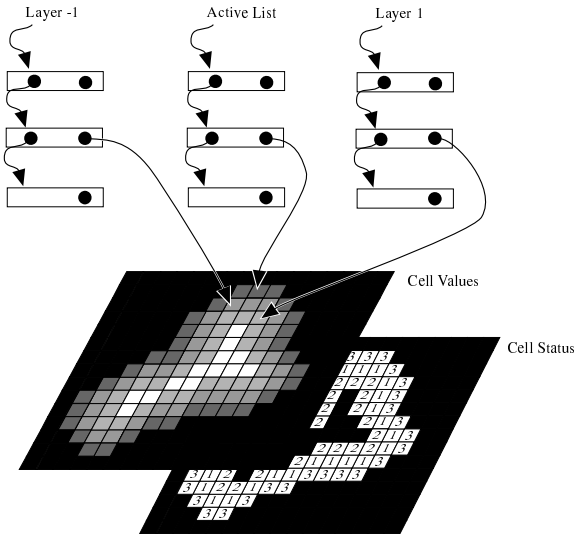


Figure 7: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

(a) For each element x_j on the status list S_i, remove x_j from the list $L_{i\mp1}$, and add it to the L_i list, or, in the case of $i = \pm(N+1)$, remove it from all lists.

(b) Add all $L_{i\mp1}$ neighbors to the $S_{i\pm1}$ list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in Figure 7. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. The computation time grows as m^{n-1}, where m is the number of grid points along one dimension of u (sometimes called the resolution of the discrete sampling). Computation time for dense-field approach increases as m^n. The m^{n-1} growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the domain, rather than the range.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. The time steps are limited by the speed of the "fastest" moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

$$\sup_{x \in \mathcal{A} \subset X} (g(x)) \leq \sup_{x \in X} (g(x)), \tag{33}$$

where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

Results from previous work [21] have demonstrated several important aspects of the sparse-field algorithm. First, the manipulations of the active set and surrounding layers allow the active set to "track" the deformable surface as it moves. The active set always divides the inside and outside of the objects it describes (i.e., it stays closed). Empirical results show significant increases in performance relative to both the computation of full domain and the narrow-band method, as proposed in the literature. Empirical results also show that the sparse-field method is about as accurate as both the full, discrete solution, and the narrow-band method. Finally, because the method positions level sets to sub-voxel accuracy it avoids aliasing problems and is more accurate then these other methods when it comes to *fitting* level-set models to other surfaces. This sub-voxel accuracy is important aspect of the implementation, and will significantly impact the quality of the results for the applications that follow.

7 Applications

This section describes several examples of how level-set surface models can be used to address problems in graphics, visualization, and computer vision. These examples are a small selection of those available in the literature. All of these examples where implemented using the sparse-field algorithm and the VISPack library, which is described in the section that follows.

7.1 Surface Morphing

This section summarizes the work of [29], which describes the use of level-set surface models to perform 3D shape metamorphosis. The *morphing* of 3D surfaces is the process of constructing a series of 3D models that constitute a smooth transition from one shape to another (i.e., a homotopy). Such a capability is interesting for creating animations and as a tool for geometric modeling. There is not yet a single, general method for generating such transitional shapes. However, there are several desirable aspects of morphing algorithms that allow us to compare the adequacy of different approaches to surface morphing. Several desirable properties of 3D surface morphing are:

1. The transition process should begin with an *initial* surface and end with a specified *target* surface.

2. The morphing algorithm should apply to a wide range of shapes and topologies.

3. Intermediate surfaces should undergo continuous 3D transitions (rather than continuity only in the image space).

4. A 3D morphing algorithm should incorporate user input easily but should degrade gracefully without it.

5. Transitional shapes should depend only on the surface geometry of the two input shapes and user input.

These requirements are not exhaustive, but they capture many of the practical aspects of 3D morphing.

In this section we show how level-set models provide an algorithm for 3D morphing which meets most of these criteria and compare favorably with existing algorithms. Furthermore, this algorithm is a natural extension of the mathematical principles discussed in previous sections. The strategy is to allow a free-form deformation of one surface (called the *initial* surface) using the signed distance transform of a second surface (the *target* surface). This free-form deformation is combined with an underlying coordinate transformation that gives either a rough global alignment of the two surfaces, or one-to-one relationships between a finite set of landmarks on both the initial and target surfaces. The coordinate transformation can be computed automatically or using user input (as in [30]).

Much of the previous 3D morphing work has focused on morphing parametric models [31, 32] and applies to only very limited classes of shapes and topologies. Several authors have described volumetric techniques. Hughes [33] demonstrates how volumes can provide topological flexibility in surface morphing. Lerios et al. [30] followed up with a volume-based scheme which incorporates user input via underlying coordinate transformations (a known generalization the image warping technique that is often used in image morphing). Neither of these approaches have dealt with the deeper issue of deforming the level sets of a volume, but rather rely on the properties of the embedding. Payne and Toga [34] as well as Cohen-Or *et al.* [35] fix the embedding problem by using a signed distance transform to create volumes from surfaces. However, interpolating distance transforms can introduce artifacts that violate the previously stated properties, and both of these methods use a discrete distance transform which introduces volume aliasing.

7.1.1 Free-Form Deformations

The distance transform gives the nearest Euclidean distance to a set of points, curve, or surface. For closed surfaces in 3D, the signed distance transform gives a positive distance for points inside and negative for points outside (one can also choose the opposite sign convention).

If two connected shapes overlap then the initial surface can expand or contract using the distance transform of the target. The steady state of such a deformation process is a shape consisting of the zero set of the distance transform of the target. That is, the initial object becomes the target. This is the basis of the proposed 3D morphing algorithm.

Let $D(\boldsymbol{x})$ be the signed distance transform of the target surface, B, and let A be the initial surface. The evolution process which takes a model S from A to B is defined by

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(\boldsymbol{x}), \tag{34}$$

where $\boldsymbol{x}(t) \in \mathcal{S}_t$ and $\mathcal{S}_{t=0} = A$. The free-form deformations can be combined with an underlying coordinate transformation. The strategy is to use a coordinate transformation (for instance a translation and rotation) to position the two surfaces near each other. These transformations can capture gross similarities in shape as well as user input. A coordinate transformation is given by

$$\boldsymbol{x}' = T(\boldsymbol{x}, \alpha), \tag{35}$$

where $0 \leq \alpha \leq 1$ parameterizes a continuous family of these transformations that begins with identity, i.e. $\boldsymbol{x} = T(\boldsymbol{x}, 0)$. The evolution equation for a parametric surface is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{N}\ D(T(\boldsymbol{x}, 1)), \tag{36}$$

and the corresponding level-set equation is

$$\frac{\partial \Phi(\boldsymbol{x}, t)}{\partial t} = |\nabla \Phi(\boldsymbol{x}, t)|\ D(T(\boldsymbol{x}, 1)). \tag{37}$$

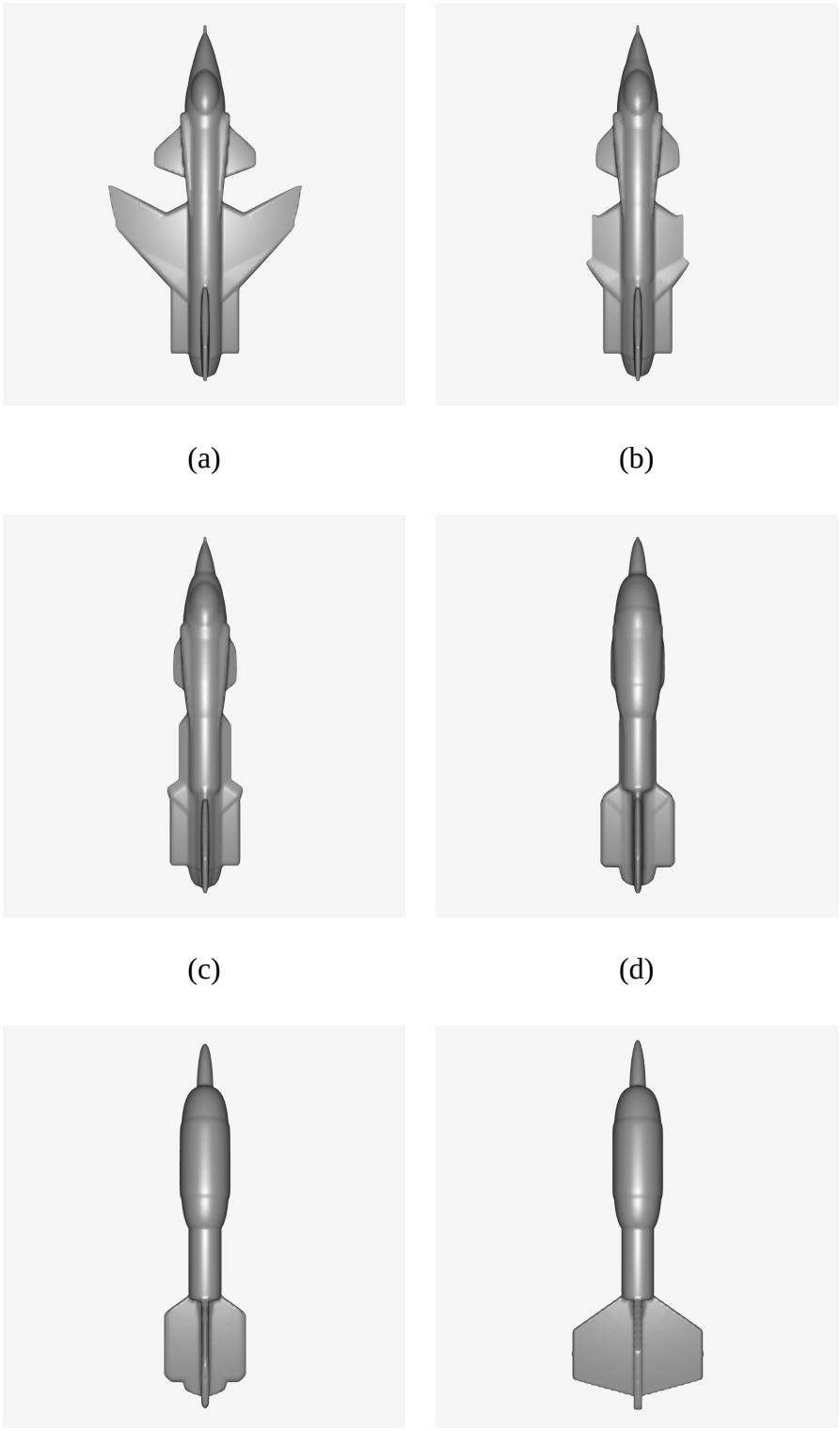
This process produces a series of transition shapes (parameterized by t). The coordinate transformation can be a global rotation, translation, or scaling, or it might be a *warping of the underlying 3D space* as was used by [30]. Incorporating user input is important for any surface morphing technique, because in many cases finding the best set of transition surfaces depends on context. Only users can apply semantic considerations to the transformation of one object to another. However, this underlying coordinate transformation

Figure 8: A 3D model of a jet that was built using Clockworks, a CSG modeling system.

can, in general, achieve only some finite similarity between the "warped" initial model and the target, and even this may require a great deal of user input. In the event that a user is not able or willing to define every important correspondence between two objects, some other method must "fill in" the gaps remaining between the initial and target surface. In [30] they propose alpha blending to achieve that smooth transition—really just a fading from one surface to the other. We are proposing the use of the free-form deformations, implemented with level-set models, to achieve a continuous transition between the shapes that result from the underlying coordinate transformation. We have also experimented with ways of automatically orienting and scaling objects, using 3D moments, in order to achieve a significant correspondence between two objects.

Figure 8 shows a 3D model of a jet that was built using Clockworks [36], a CSG modeling system. Lerios et al. [30] demonstrate the transition of a jet to a dart, which was accomplished using 37 user-defined correspondences, roughly a hundred user-defined parameters. Figure 9 shows the use of level-set models to construct a set of transition surfaces between a jet and a dart. The triangle mesh is extracted from the volume using the method of marching cubes [5]. These results are obtained without any user input. Distance transforms on the CSG models are computed near the level surface using an analytical description and extended into the volume using a level-set method [37].

The application in this section shows how level-set models moving according to the first-order term given in expression 2 in Table 1 can "fit" other objects by moving with a speed that depends on the signed distance transform of the target object. The application in the



(a) (b)

(c) (d)

(e) (f)

Figure 9: The deformation of the jet to a dart using a level-set model moving with a speed defined by the signed distance transform of the target object.

next section relies on expression 5 of Table 1, a second-order flow that depends on the principal curvatures of the surface itself.

7.2 Filleting and Blending Solid Objects

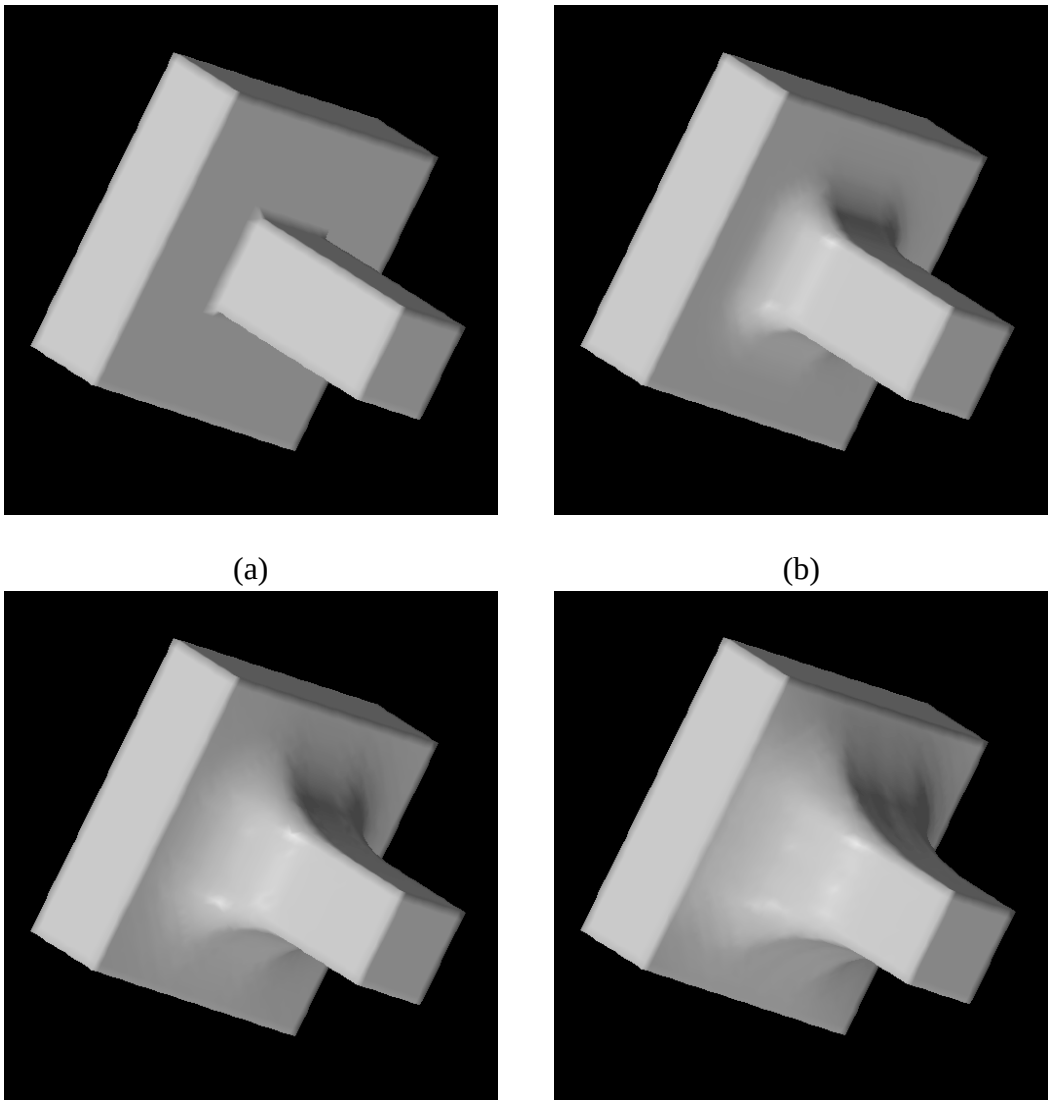
The construction of blending surfaces is an important tool in solid modeling. Geometric solid primitives and their intersections often produce sharp corners or creases that are often not consistent with the real-world objects that they are intended to represent. This section shows how blending can be described as a deformation process, where surfaces move under a geometric flow that can add or remove material based on local curvature information. The result is a method for solid object blending that does not depend on any particular model representation. Thus this method is not restricted to a specific class of shapes or topologies. Additionally, the results are invariant; they do not depend on arbitrary choices of coordinate systems or bases. The only requirement is that the blended objects must be closed surfaces with some known inside-outside function.

Surface blending techniques are typically tied very closely to the choice of geometric primitives. For instance, Middleditch and Sears [38] propose a set-theoretic method for blending solids which relies on low-order algebraic primitives. A fillet at the joint of two tori requires the solution of a degree 32 polynomial. Bloomenthal and Shoemake [39] propose a modeling system based on convolutions, which relies on a skeletonized representation of objects. In general the use of convolution to achieve deformations on implicit shapes results in shapes that reflect both the shape of the model and the embedding, Φ.

The blending method proposed in this section implements an interative smoothing scheme that smooths only along the level set; the final result is independent of the embedding. Consider the case of fillets. We propose that a fillet can be constructed from a process of "filling in" material in places of high curvature. The curvature of a level-set model can be calculated from the embedding, and the deformation of the level set is well defined by the curvature terms in Table 1.

The strategy is to construct a curvature term, k_p, that consists of only positive curvatures. [1] The principal curvatures of the level sets of Φ are functions of Φ and its derivatives. For a specific Φ the principal curvatures are functions of 3-space $k_1(\boldsymbol{x})$ and $k_2(\boldsymbol{x})$. For *adding* material the joint between two objects, we consider only the positive curvature components,

[1] The sign of curvature is defined by the direction of the normals— in this work normals point into the volume enclosed by the object.



(a) (b)

(c) (d)

Figure 10: Two rectangular solid models are joined by a volumetric fillet that is created from a positive curvature flow.

i.e.,

$$\frac{\partial \Phi}{\partial t} = |\nabla \Phi| k_p = |\nabla \Phi| k_1^+ + |\nabla \Phi| k_2^+, \tag{38}$$

where k^+ consists of only the positive parts of k and is defined as zero elsewhere. Because the use of separate curvature terms can cause over-shooting, the up-wind scheme (treating k_p as a space-varying velocity in the normal direction) is used for this evolution.

Figure 10 shows how the positive-curvature flow can be used to construct fillets. No knowledge of the underlying models is necessary. The fillets grow larger as more time passes. The physical extent or position of the fillet can be controlled by either specifying a region of action or by placing a small blob of deformable material in the joint that requires a fillet. Figure 11 shows how such a blending capability can be useful in animation. In this case a pair of superquadrics undergo a rigid transformation that controls their relative positions. Level-set models with a positive-curvature flow are used to create a smooth joint between these two primitives. Notice that the positive curvature method does not suffer from the growth or expansion artifacts that are often associated with distance-based blending methods [40].

Thus, a second-order flow can create smooth blends between objects in a way that does not require specific knowledge of the shapes or topologies of the object involved. The application in the next section, 3D scene reconstruction, shows how a combination of first-order and second-order terms from Table 1 are combined to create technique that fits models to data while maintaining certain smoothness constraints and thereby offsetting the effects of noise.

7.3 3D Reconstruction from Multiple Range Maps

Level-set models are useful for problems related to 3D reconstruction. Previous work has presented level-set results derived from noisy 3D data such as MRI [19] and ultrasound [41]. In [42] we have shown how the reconstruction of objects from multiple range maps can be formulated as a problem of finding the surface that optimizes the posterior probability given a set of measurements (noisy range maps) and some information about the a-priori probability of different kinds of surfaces. That optimization problem can be expressed as a volume integral which can be solved with level-set models. This section presents the mathematical expressions that result from those formulations and presents some new results: the reconstruction of entire scenes by fitting level-set models to the data from a scanning LADAR (laser ranging and detection) system.

A *range map* is a collection of range measurements taken along different directions (lines

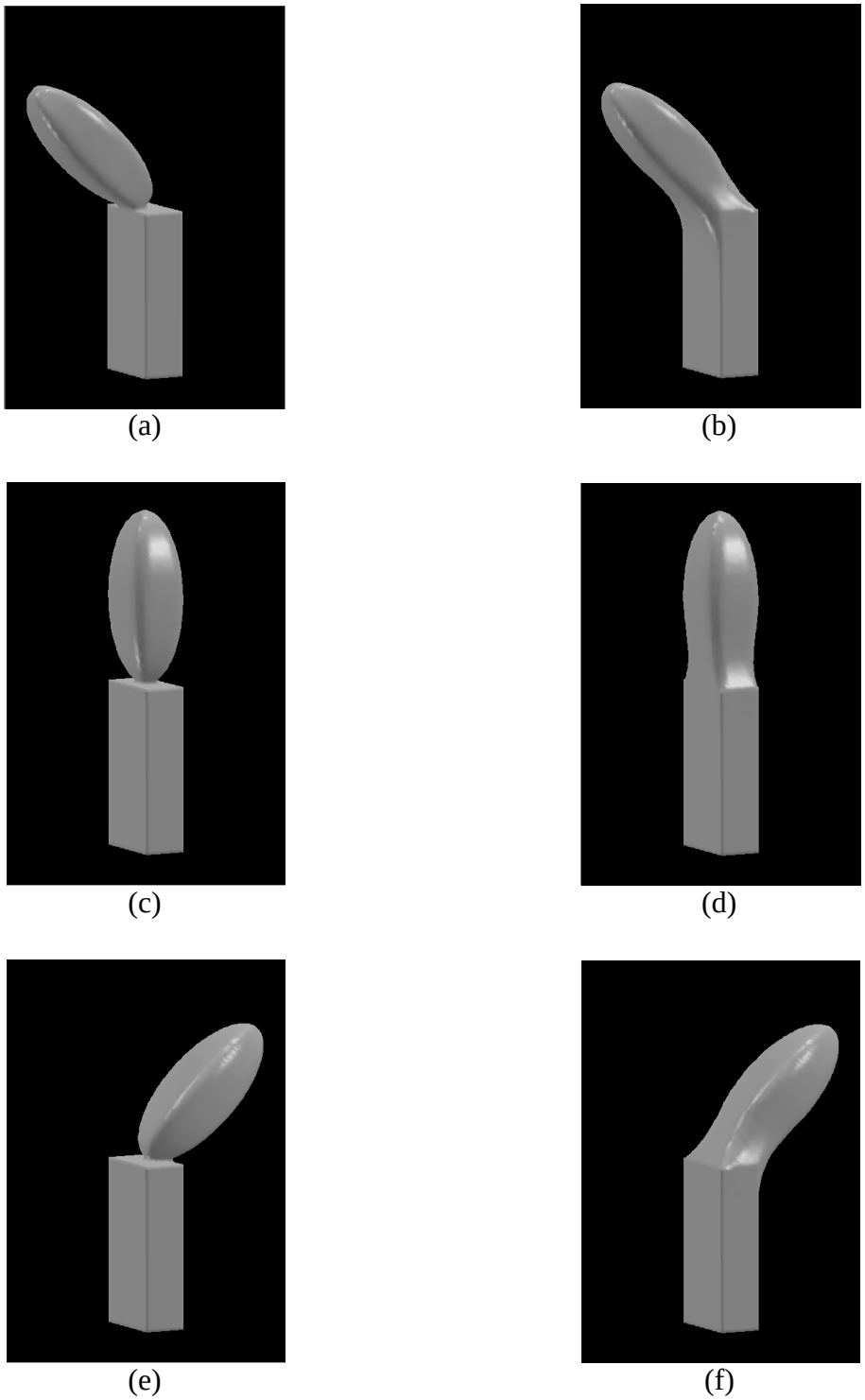


Figure 11: A short animation is created by specifying the relative motion between two superquadric components of an object. A positive-curvature flow (applied frame by frame to the joint between the two 3D models) creates a smooth, flexible object.

of sight) but from a single point of view. Range maps could come from any number of different sources including laser scanners, structured light depth systems, shape from stereo, or shape from motion. We assume that such range maps are noisy and uncertain. The goal is to combine a number of range maps from different points of view to create a 3D structure that reflects the collective confidence and depth measures.

Several examples in the literature have applied parametric models to this task. Turk and Levoy [43], for instance, "zip" together triangle meshes in order to construct 3D objects from sequences of range maps from a laser range finder. They perform minor adjustments to the surface position in order account for ambiguity in the range maps. Their approach assumes very little noise in the input, which is reasonable given the high quality of their range maps. Chen and Medioni [44] use a parametric (triangle mesh) model which expands inside a sequence of range maps. Curless and Levoy [45] describe a volume-based technique for combining range data. They use the signed distance transform to encode volume elements with data that represent the averages (with some allowance for outliers) of multiple measurements. Surfaces of objects are the level sets of volumes. Related approaches are given in [46, 47]. Bajaj et. al. [48] use a Delaunay triangulation to impose a topology on a set of unordered 3D points and then fit trivariate Bernstein-Bezier patches—i.e. a higher-order implicit model—to the data. Muraki [2] uses implicit or blobby models to reconstruct objects from range data. The individual blobs are spherically symmetric 3D potentials that are combined linearly so that they blend together. The resulting models, with approximately 400 primitives are quite coarse.

This work differs from previous work in two ways. First, rather than heuristics, our reconstruction strategy is based on a strategy that solves for the optimal surface estimate. This optimal estimate includes information about one's expectations of the likelihood of different surfaces. The result is not a closed-form solution, but an iterative process that seeks to fit a level-set model to the data while enforcing a kind of smoothness on the data.

7.3.1 Objective function for multiple range maps

The evolution equation for the estimation of optimal surfaces is shown in [42] to consist of two parts:

$$\frac{\partial \boldsymbol{x}}{\partial t} = -G(\boldsymbol{x})\boldsymbol{N} + \rho(\mathcal{S}). \tag{39}$$

This first part, $-G(\boldsymbol{x})\boldsymbol{N}$, is the data term, which is a movement with variable speed (as in expression 2 from Table 1) that is the cumulative effect from all of the individual range maps. The second part is the prior, which describes the likelihood of the surface indepen-

dent of the data. The data term is

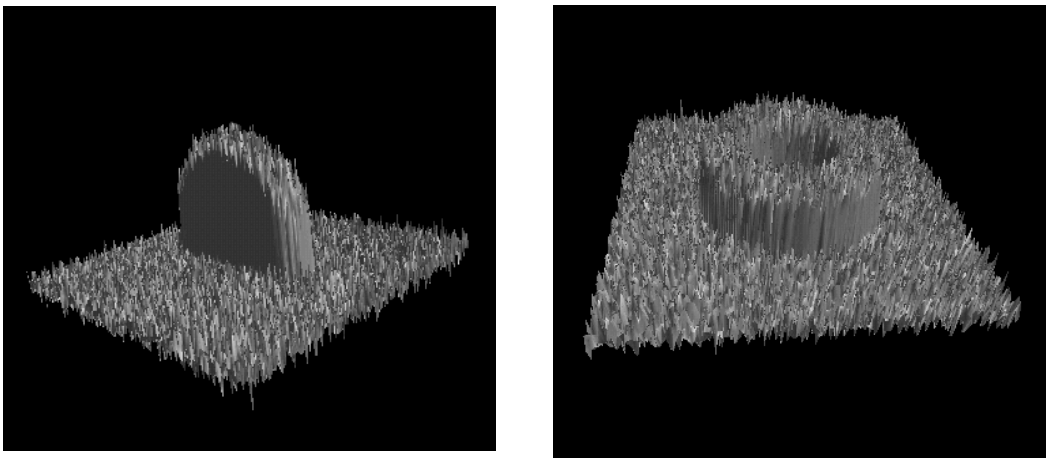
$$G(\boldsymbol{x}) = \sum_j c^{(j)}(\boldsymbol{x})\ D^{(j)}(\boldsymbol{x})\ \omega\left(D^{(i)}(\boldsymbol{x})\right)\ \gamma^{(j)}(\boldsymbol{x}), \tag{40}$$

where D_j is the signed distance along the line of sight from a range measurement in range map j associated passing through $\boldsymbol{x}$. The function $\omega : \mathbb{R} \mapsto \mathbb{R}$ is a windowing function that limits the penalty of any one range measurement, and $c(\cdot)$ is a confidence function, which is inversely proportional to the level of noise in the range measurement associated with the same line of sight. The term $\gamma(\cdot)$ is an integration constant that takes into account the curvilinear coordinate system of the range scanner.

Thus, a set of range maps creates a scalar function of 3D, which describes the movement of a surface model as it seeks the optimal surface position. In the absence of a prior, $\rho = 0$, the zero set of this function is the final position (steady state) of that evolving surface. Thus, in the absence of a prior, one could sample $g(\boldsymbol{x})$ and obtain an approximation to the optimal surface estimate. This strategy results in an algorithm that is very much like that of [45].

There are several reasons for going to an iterative scheme for finding optimal solutions. First is the use of a prior. In surface reconstruction, even a very low level of noise can degrade the quality of the rendered surfaces in the final result, and in such cases better reconstructions can be obtained by introducing a prior. Second is aliasing. Discretizing $g(\boldsymbol{x})$ and finding the zero crossings will cause aliasing in those places where the transition from positive to negative is particularly steep. A deformable model can place the surface much more precisely. The third reason for going to an iterative scheme is that despite the windowing function $\omega(\boldsymbol{x})$ there is interference between different range maps at places of high curvature. This problem is addressed by introducing a nonlinearity which is solved in an iterative scheme given by equation 39. In the work described in [21], the solution of the linear problem, the zero set of $g(\boldsymbol{x})$, serves as the initial estimate for the nonlinear, iterative optimization strategy that results from the inclusion of a prior and a nonlinear term that compensates for lack of any explicit model of self occlusions.

Equation 39 includes a *prior*, which is a likelihood function on surface shape. A reasonable choice of prior is one that models objects with less surface area as more likely than objects with more surface area. Alternatively, one could say that given a set of surfaces that are near the data, the algorithm should choose a surface that has less area. Often, but not always, this will be the smoother surface. The $\rho(\mathcal{S})$ that results from this prior is the mean curvature. Therefore the evolution of the surface, using the level-set formulation, that seeks to maximize the posterior probability (given a set of range maps and a prior that penalizes



(a) (b)

Figure 12: Range maps: Synthetic range data 200×200 pixels with 20% Gaussian white noise of a torus end (a) and side (b).

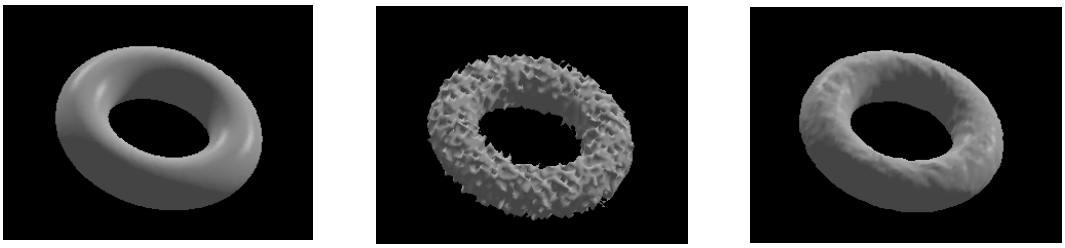
surface area) is

$$\frac{\partial \Phi(\boldsymbol{x},t)}{\partial t} = |\nabla \Phi(\boldsymbol{x})| \sum_j \left(D^{(j)}(\boldsymbol{x})\, \omega\left(D^{(i)}(\boldsymbol{x})\right) \times\ \gamma^{(j)}(\boldsymbol{x})\ c^{(j)}(\boldsymbol{x}) \frac{\left(\nabla \Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})\right)^{+}}{\nabla \Phi \cdot \boldsymbol{n}^{(j)}(\boldsymbol{x})} \right) + \beta H, \tag{41}$$

where $\boldsymbol{n}^{(j)}(\boldsymbol{x})$ is the line of sight from a range finder to a 3D point, $\boldsymbol{x}$, β is a free parameter that controls the level of smoothing in the model, and H is the expression for the mean curvature given in equation 8.

Figure 12 shows a pair of simulated range maps constructed from an analytical description of a torus. These 200×200 pixel range maps are corrupted with additive Gaussian noise that has a standard deviation of 20% (as a function of the smaller of the two radii). Six synthetic noise-corrupted viewpoints of a torus are combined to create a level-set reconstruction of a torus. Figure 13(a) shows the initial model (80×80×40 voxels) used for fitting a level-set models to the range data. Figure 13(b) shows the result of the level-set models that uses 13(a) as an initial state and has a value of β equal to 0.5. The result is a reasonable reconstruction of the noiseless model (Figure 13(c)) which combines the six points of view and the smoothing function.

Figure 14(a) shows a range map taken with the Perceptron model P5000, an infra-red, time-of-flight laser range finder with a pan-tilt mechanism. Figure 14(b) shows the amplitudes associated with the return signal (an *intensity*), and 14(c) shows a surface plot of the range

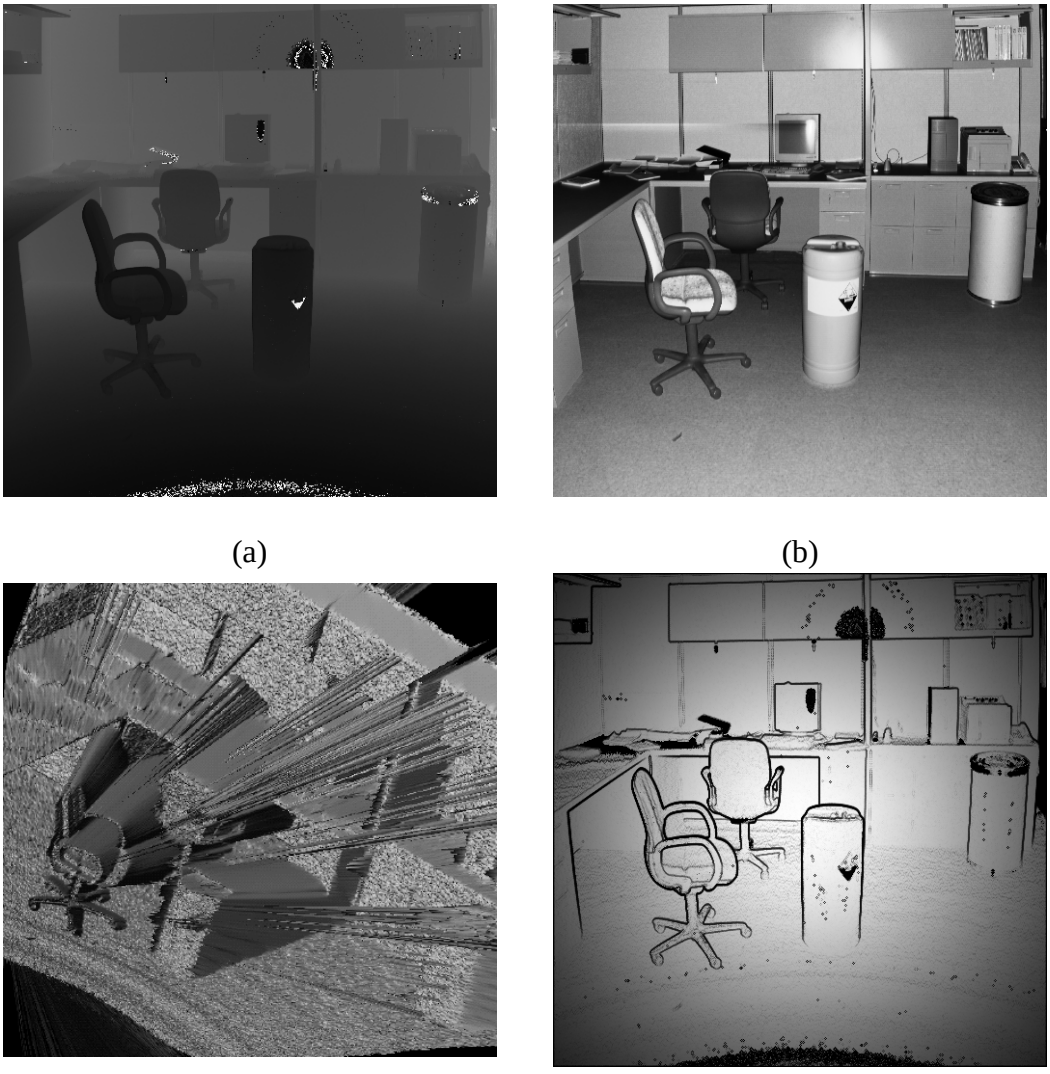


(a) (b) (c)

Figure 13: (a) An analytically-defined model of a torus. (b) An initial model ($80\times80\times40$ voxels) is constructed by combining six points of view of a torus and solving for $g(\boldsymbol{x}) = 0$. (c) The model, which is attracted to the range data but subject to internal forces, evolves and settles into a smoother steady state.

map to demonstrate the degree of noise (additive and outliers). Figure 14(d) shows the confidence values associated with those range measurements. These confidence values are derived from empirical data about the level of noise in the range finder (which depends on the return amplitude), and some analysis, from first principles, about the effects of uncertainty in the 3D positions of the scans and the model — which results in the lower confidence at edges as described in [42]. We combined twelve such views from different locations in the room to generate the results that follow.

Figure 15(a) shows the initial estimate based on the zero crossings of $g(\boldsymbol{x})$, and 15(b) shows the result of 32 iterations with the prior term and the correction for the surface normal direction. The size of the volume is $300 \times 150 \times 180$ voxels, and the resolution is 1.8 cm/voxel. These results show the ability of the statistically-based approach to overcome the noise in the scanner, and they show that the inclusion of iterative, model-fitting scheme helps create more accurate reconstructions. The resolution of the model falls below that of the scans, because it was limited by the random-access-memory available on our workstation. Some small features, such as the arm rests of the chairs, are lost because of the inaccuracies in the registration of the individual range maps.



(a) (b)

(c) (d)

Figure 14: (a) One of twelve range maps (b) The associated amplitude map (c) A surface plot of the range data to show the level of noise. (d) The confidence measures associated with those range values.

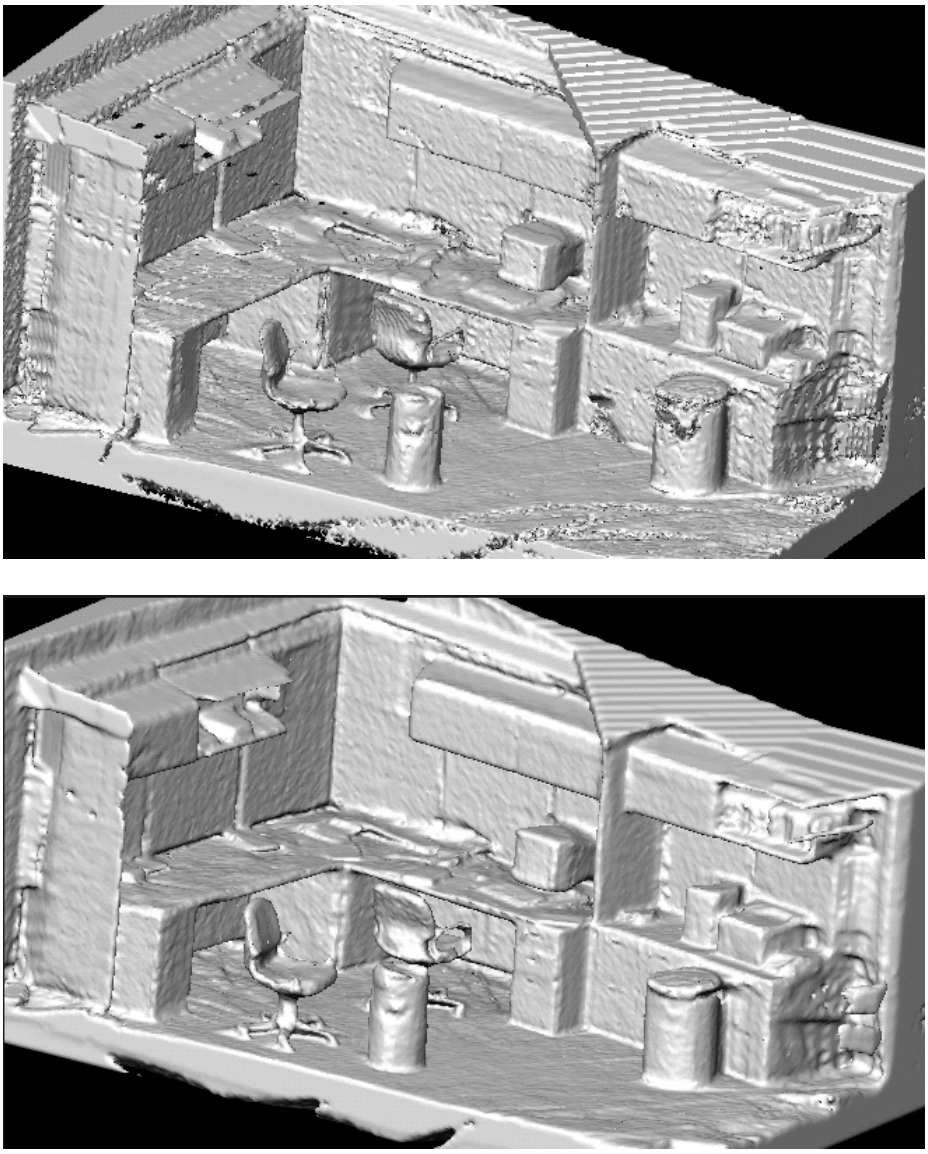


Figure 15: (top) The 3D reconstruction resulting from the zero crossings of $g(\boldsymbol{x})$ gives some averaging, but includes no prior. (bottom) The result of 32 iterations with the iterative scheme includes the prior and excludes influences of data on surfaces that face away from the scanner.

8 VISPACK

8.1 Introduction

VISPACK is a set of *C++*, object-oriented libraries for image processing, volume processing, and level-set surface modeling. It consists of five libraries: Matrix, Image, Volume, Util, and Voxmodel (level-set modeling). These libraries can be used separately or together when creating applications.

VISPACK incorporates eight basic design attributes. These are

Data Handles/Copy on Write: VISPack is an object-oriented library, and as such we allow the objects to handle memory management, and relieve the programmer (in most cases) from having to worry pointers and the corresponding memory allocation/deallocation problems. For this we use the data handles with a *copy on write protocol*. Copy constructors perform a shallow copy with reference counting until a *non const* operation on the underlying buffers forces a deep copy. Thus deep copies are performed only when necessary, but all memory is maintained by the objects and objects behave as "variables" rather than pointers.

Modified Data Hiding: Access to data in objects is generally through access methods, however, pointers to buffers for fast implementations are available.

Templates: VISPack utilizes the templating construct of C++ virtually throughout. Many of the objects, including images, volumes, lists, and arrays, are intended to support a wide range of data types. Thus, via templating programmers can define the pixels of different images of different types, such as floating point, 24-bit color, and 16-bit greyscale.

Use of Standard File Formats: When appropriate VISPack uses standard file formats. We choose formats that are well known and have publicly available libraries that can be distributed with our libraries. The matrix library uses a simple text format. The image library uses TIFF and FITS file formats. Because no standard format exists for saving volumes of data we do use a *raw* file format.

Operator Overloading: Proper use of operator overloading gives users a convenient way to execute operations on an object. When compined with the copy-on-write convention, operator overloading allows programmers to treat many heavy-weight objects (e.g. images and volumes) as variables. For instance, the following code computes non-maximal edges in a on a filtered volume.

```
Volume<float> dx, dy, dz;
Volume<float> vol_gauss = vol.gauss(0.5);
Volume<float> vol_out = (((dx = vol_gauss.dx()).power(2)
       *vol_gauss.dx(2)
     + ((dy = vol_gauss.dy()).power(2)*vol_gauss.dy(2)
     + ((dz = vol_gauss.dz()).power(2)*vol_gauss.dz(2)
     + dx*dy*(dx).dy() + dx*dz*(dx).dz())
     + dy*dz*(dy).dz()) ).zeroCrossings()
       && ((dx.power(2) + dy.power(2)) > T*T));
```

8.2 Level-Set Surface-Modeling Library

The Level-Set Surface-Modeling (LSSM) Library is an implementation of the level-set technique [10, 13] specifically for deforming surface models embedded in volumes. The implementation uses the sparse-field method described in [20]. The library implements all of the basic numerical algorithms and handles all of the data structures required to perform LSSM. The strategy for using this library is to subclass the object `VoxModel`, set some parameters, define a set of simple virtual functions that control the deformation process, initialize the model, and then direct the model to iteratively deform according to those equations. This section describes the relationship between the mathematics of previous sections and the VISPack library. Its also presents an example of using VISPack libarary to do 3D shape metamorphosis as described in Section 7.1.

8.2.1 Surface Deformation

The LSSM library allows one to solve for surface deformations, as a function of time, for general level-set surface movements of the form:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x})) + \beta G(\boldsymbol{x}, \boldsymbol{N}(\boldsymbol{x}))\boldsymbol{N}(\boldsymbol{x}) + \gamma \boldsymbol{N}(\boldsymbol{x}) + \eta E\left(k_1(\boldsymbol{x}), k_2(\boldsymbol{x})\right), \qquad (42)$$

where $\boldsymbol{x}$ is a point on the surface. This equation is solved by representing the surface as the kth level set of an implicit function $\phi(\boldsymbol{x}, t) : \mathbb{R}^3 \times \mathbb{R}^+ \mapsto \mathbb{R}$. This gives

$$\frac{\partial \phi}{\partial t} = \alpha \boldsymbol{F}(\boldsymbol{x}, \nabla\phi)) \cdot \nabla\phi + \beta G(\boldsymbol{x}, \nabla\phi)|\nabla\phi| + \gamma|\nabla\phi| + \eta E(D\phi, \mathrm{D}^2\phi), \qquad (43)$$

where $\mathrm{D}\phi$ and $\mathrm{D}^2\phi$ are collections first and second derivatives of ϕ, respectively. This equation is solved on a discrete grid using an *up-wind* scheme gradient calculations, central differences for the curvature, and forward finite differences in time. The LSSM library uses the *sparse-field* method described in Section 6.3 and in [21].

Thus, the LSSM library offers the following capabilities:

1. Creates an initial model (with associated active set) from a volume.
2. Calculates $\Delta u^n_{i,j,k}$ and Δt using virtual functions (defined by subclasses) that describe $\boldsymbol{F}$ and G, and parameters (values set by the subclass) α, β, γ, and η.
3. Performs an update on the values of $u^n_{i,j,k}$.
4. Maintains the list of active grid points and updates the *layers* around those points in order to maintain a neighborhood from which to calculate subsequent updates.
5. Provides access to the volume that defines $u^n_{i,j,k}$ and the linked list of active grid points.

Given the volume defining $u^n_{i,j,k}$, one can then rely on the functionality of the volume library for subsequent processing, file I/O, or surface extraction.

8.2.2 Structure and Philosophy of the LSSM Library

The library is organized (mostly for ease of development) into a base class, `LevelSetModel`, and a derived class, `VoxModel`. The base class does all of the book keeping associated with the active set and surrounding *layers*, the link lists associated with those sets, and initializing the model. Thus it adds and removes voxels from the active set (and surrounding layers) in response to an update operation. The base class assumes that the subclasses know how to update individual voxels. Applications are built by subclassing `VoxModel` and redefining a small set of virtual functions that control the movement of the model.

The subclass, `VoxModel`, performs update on the grid points in the active set of the form given in Equation 18, using functions $\boldsymbol{F}$ and G and parameters α, β, γ, and η. It also calculates the maximum Δt that ensures stability. Thus a user who wishes to perform a surface deformation using the LSSM library, would create subclass of `VoxModel` and define the appropriate virtual functions and set the parameters to achieve the desired behavior.

8.2.3 The `LevelSetModel` Object

The `LevelSetModel` contains a volume of values, a volume of status flags, five lists (one active list, two inside lists, and two outside lists), and three parameters that determine the origin of the coordinate system form which the model performs its calculations.

There are two constructors, `LevelSetModel()` and `LevelSetModel( const VISVolume<float> &)`. The first simply initializes the data structure, and the second also set the values of the model volume (`_values`) to the input. Once the values have been set, one can create an initial volume from those values by calling `constructLists()`, which can also take a floating-point argument that controls the scaling of the input relative to a local distance transform near the zero set.

The list that keeps track of the active set, called `_active_list`, keeps track of the location of those grid points and a single floating-point value, which stores the change in their values from one iteration to the next.

Another important methods for users of this object is `update(float)`, which changes the grey-scale values of the grid for the active set according to the values stored in `_active_list`, and updates the status of elements on the active list as well as the values and status of nearby layers (2 inside and 2 outside). The floating point argument is the value of Δt from Equation 18, and the return value is the maximum change that occurred on the active set. Finally, the method `iterate()` calls the virtual method `calculate_change`, a virtual function which sets the values of $\Delta u^n_{i,j,k}$ and returns the maximum value of Δt for stability, and then calls `update`. For this object the function `calculate_change` performs some trivial (i.e., useless) operation.

8.2.4 The `VoxModel` Object

The `VoxModel` object is a subclass of `LevelSetModel`, and it add three things to the base class.

1. `calculate_change()` is redefined to implement the surface deformation described in Equation 43.
2. The virtual functions are declared for F (called `force`) and G (called `grow`). These functions are defined to return zero for this object.

3. The parameters that control the relative influence of the various terms are read from file by a routine `load_params`.

4. A method `rescale(float)` is defined, which resamples the volume of grid-point values into a new volume with different resolution and redefines the lists (and thereby the model) in this new volume. This method is for performing coarse-to-fine deformation procedures.

8.3 Example: 3D Shape Metamorphasis

The `Morph` object allows one to construct a sequence of volumes or surface meshes using the 3D shape metamorphasis technique described in Section 7.1, which was first proposed by Whitaker and Breen [20]. This technique relies distance transforms for both the source and target objects and uses a LSSMs to manipulate the shape of the source so that it coincides with the target. The surface deformation that describes this behavior is

$$\frac{\partial \boldsymbol{x}}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) \boldsymbol{N}(\boldsymbol{x}), \tag{44}$$

where $G(\boldsymbol{x})$ is simply the distance transform (or some monotonic function thereof) of the target, and T is a coordinate transformation that aligns the source and target objects. The level-set formulation of this is

$$\frac{\partial \phi(\boldsymbol{x}, t)}{\partial t} = \beta G\left(T(\boldsymbol{x})\right) |\nabla \phi|. \tag{45}$$

The morphing process consists of several steps:

1. Read in distance transforms (in the form of volumes) for both source and target.

2. Initialize the LSSM by fitting it to the zero set of the source distance transform.

3. Update the LSSM according to Equation 45.

4. Save intermediate volumes/surfaces at regular intervals.

The remainder of this section lists the code and comments for three files, morph.h (which declares the `Morph` object), morph.C (which defines the methods) and main.C (which performs all of the I/O and uses the `Morph` object to construct a sequence of shapes.

8.4 Morph.h

```
//
// morph.h
//
//

#ifndef iris_morph_h
#define iris_morph_h

#include "voxmodel/voxmodel.h"
#include "matrix/matrix.h"

#define INIT_STATE 0
#define MORPH_STATE 1
//
// This is the morph object.  It uses all of the machinery of the base
// class to manipulate level sets.  It needs to have an initial volume
// and a final volume (which would typically be the distance transform,
// it might need a 3D transformation, and it needs to redefine the
// virtual function "grow", which takes 6 floats as input, the position
// followed by the normal vectors (all will calculated and passed into
// this method by the base class).  It might also have a state, that
// indicates whether or not it's been initialized.
//
// Functions not defined here should be defined in "morph.C"
//
class Morph: public VoxModel
{
  protected:
    VISVolume<float> _dist_source;
    VISVolume<float> _dist_target;
    VISMatrix _transform;
//
// This is the function that is used by the base class to manipulate the
level
// set.  You can define it to by anything you want.  For this object, it
will
// return a value from the distance transform of the target.
//
```

```
    virtual float grow(float x, float y, float z,
         float nx, float ny, float nz);

// There are two states.  In the first state, the model is trying to fit
// to the input data.  In this way the models starts by looking just like

// the input data
    int _state;

  public:

    Morph(const Morph& other)
    {
 _dist_target = other._dist_target;
 _initial = other._initial;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    Morph(VISVolume<float> init, VISVolume<float> d)
 :VoxModel()
    {
 _dist_target = d;
 _initial = init;
 _state = MORPH_STATE;
 _transform = VISVISMatrix(3, 3);
 _transform.identity();
// initialize();
    }

    void initialize();

// for this object I assume that the transform is just a matrix.
// but it could be anything
    void transform(const VISVISMatrix& t)
    { _transform = t;}

    const VISVISMatrix& transform()
    { return(_transform);}
```

```
    void distance(const VISVolume<float> d)
    { _dist_target = d;}
    VISVolume<float> distance()
    { return(_dist_target);}

};
#endif
```

8.5 Morph.C

```
#include "morph.h"
#include "util/geometry.h"
#include "util/mathutil.h"


//
// this is the virtual function, that  is the guts of it all.
//

float Morph::grow(float x, float y, float z,
    float nx, float ny, float nz)
{

// this says you are in the morph state (things have been initialized)
    if (_state == MORPH_STATE)
 {
 float xx, yy, zz;
 VISPoint p(4u);
 p.at(0) = x;
 p.at(1) = y;
 p.at(2) = z;
 p.at(3) = 1;
 VISPoint p_tmp;
// this is where you could put some other transform.
 p_tmp = _transform*p;

 xx = p_tmp.x();
 yy = p_tmp.y();
```

```
 zz = p_tmp.z();

// make sure you are not out of the bounds
// of your distance volume.
 if (_dist_target.checkBounds(xx, yy, zz))
// if not, get the distance (use trilinear interpolation).
     return(_dist_target.interp(xx, yy, zz));
 else
     return(0.0f);
    }
else
    {
// if you are still initializing, then move toward the zero set of
// your initial case
 if (_initial.checkBounds(x, y, z))
     return(_initial.interp(x, y, z));
 else
     return(0.0f);
    }
}

// this makes the model look like the input.
#define INIT_ITERATIONS 5
void Morph::initialize()
{
    _values = _initial;
    int state_tmp = _state;
    _state = INIT_STATE;
    construct_lists(DIFFERENCE_FACTOR);
// these couple of iterations are required to make sure that the zero
// sets of the model match the zero sets of the
//
    for (int i = 0; i < INIT_ITERATIONS; i++)
 {
// limit the dt to 1.0 so that the model settles in to a solution
     update(::min(calculate_change(), 1.0f));
 }
    _state = state_tmp;
}
```

8.6 Main.C

```
#include "vol/volume.h"
#include "vol/volumefile.h"
#include "image/imagefile.h"
#include "morph.h"
#include <string.h>


const int V_HEIGHT = (40);
const int V_WIDTH = (40);
const int V_DEPTH = (40);

#define XY_RADIUS (12)  // this matches the 2.5D data generated in
torus.C
#define T_RADIUS (4)  // this matches the 2.5D data generated in torus.C
#define S_RADIUS (12)  // radius of a sphere

#define B_WIDTH (20.0f)
#define B_HEIGHT (60.0f)
#define B_DEPTH (20.0f)

#define B_CENTER_X (12.0f)
#define B_CENTER_Y (32.0f)
#define B_CENTER_Z (12.0f)

float sphere(unsigned x, unsigned y, unsigned z);
float torus(unsigned x, unsigned y, unsigned z);
float cube(unsigned x, unsigned y, unsigned z);


// This is a program that does the morph.  If you give it two
// arguments, it reads the initial model and the dist trans for the
// final model from the two file names given, otherwise, it makes a
sphere
// and deforms it into a torus

main(int argc, char** argv)
{
```

```
VISVolume<float> vol_source, vol_target;
VISVolumeFile vol_file;
int i;
char fname[80];

vol_source = VISVolume<float>(25,65,25);
vol_source.evaluate(cube);

if (argc > 2)
    {
// read in the sourceing model
 vol_source = VISVolume<float>(vol_file.read_float(argv[1]));
// read in the dist trans of the final model
     vol_target = VISVolume<float>(vol_file.read_float(argv[2]));
    }
else
// make up some volumes
    {
 vol_source = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_source.evaluate(sphere);
 vol_target = VISVolume<float>(V_WIDTH, V_HEIGHT, V_DEPTH);
 vol_target.evaluate(torus);
    }

// create morph object
Morph morph(vol_source, vol_target);
// loads in some parameters (for morphing these are all zero but one)
// i.e.
//
//
//
//
morph.load_parameters("morph_params");
morph.initialize();
vol_file.write_float(morph.values(), "morph0.flt");

float dt;

// do 150 iterations for your model to get from start to finish
// probably don’t need this many iterations
```

```
for (i = 0; i < 150; i++)
    {
 dt = morph.calculate_change();
// limit dt to 0.5 so that model never overshoots goal
 dt = min(dt, 0.5f);
 morph.update(dt);

 printf("iteration %d dt %f\n", i, dt);

 if (((i + 1)%10) == 0)
 {
// save every tenth volume
     sprintf(fname, "morph_out.%d.dat", i + 1);
     vol_file.write_float(morph.values(), fname);
 }
    }

// save a surface model (i.e. marching cubes).
vol_file.march(0.0f, morph.values(), ‘‘morph_final.iv’’);

printf("done\n");

}
```

References

[1] J. Blinn, "A generalization of algebraic surface drawing," *ACM Trans. on Graphics*, vol. 1, pp. 235–256, March 1982.

[2] S. Muraki, "Volumetric shape description of range data using "blobby model"," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 227–235, July 1991.

[3] G. Taubin, "An accurate algorithm for rasterizing algebraic curves and surfaces," *IEEE Computer Graphics & Applications*, March 1994.

[4] D. Breen, S. Mauch, and R. Whitaker, "3d scan conversion of csg models into distance, closest-point and colour volumes," in *Volume Graphics* (M. Chen, A. Kaufman, and R. Yagel, eds.), pp. 135–158, London: Springer, 2000.

[5] W. Lorenson and H. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1982.

[6] M. Levoy, "Display of surfaces from volume data," *IEEE Computer Graphics and Applications*, vol. 9, no. 3, pp. 245–261, 1990.

[7] R. A. Drebin, L. Carpenter, and P. Hanrahan, "Volume rendering," in *SIGGRAPH '88 Proceedings*, pp. 65–74, August 1988.

[8] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active contour models," *International Journal of Computer Vision*, vol. 1, pp. 321–323, 1987.

[9] D. Terzopoulos and K. Fleischer, "Deformable models," *The Visual Computer*, vol. 4, pp. 306–331, December 1988.

[10] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[11] J. Sethian, *Level Set Methods and Fast Marching Methods*. Cambridge: Cambridge University Press, second ed., 1999.

[12] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proc. Nat. Acad. Sci.*, vol. 93, no. 4, pp. 1591–1595, 1996.

[13] J. A. Sethian, *Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Sciences*. Cambridge University Press, 1996.

[14] S. Osher and R. Fedkiw, "Level set methods: An overview and some recent results," Tech. Rep. 00-08, UCLA Center for Applied Mathematics, Department of Mathematics, University of California, Los Angeles, 2000.

[15] L. Alvarez and J.-M. Morel, "A morphological approach to multiscale analysis: From principles to equations," in *Geometry-Driven Diffusion in Computer Vision* (B. M. ter Haar Romeny, ed.), pp. 4–21, Kluwer Academic Publishers, 1994.

[16] V. Caselles, R. Kimmel, and G. Sapiro, "Geodesic active contours," in *Fifth Int. Conf. on Comp. Vision*, pp. 694–699, IEEE, IEEE Computer Society Press, 1995.

[17] B. B. Kimia and S. W. Zucker, "Exploring the shape manifold: the role of conservation laws.," in *Shape in Picture: the mathematical description of shape in greylevel images* (Y.-L. O, A. Toet, H. Heijmans, D. H. Foster, and P. Meer, eds.), Springer-Verlag, 1992.

[18] R. Malladi, J. A. Sethian, and B. C. Vemuri, "Shape modeling with front propagation: A level set approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 2, pp. 158–175, 1995.

[19] R. T. Whitaker and D. T. Chen, "Embedded active surfaces for volume visualization," in *SPIE Medical Imaging 1994*, (Newport Beach, California), 1994.

[20] R. Whitaker and D. Breen, "Level-set models for the deformation of solid objects," in *The Third International Workshop on Implicit Surfaces*, pp. 19–35, Eurographics, 1998.

[21] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[22] R. T. Whitaker, "Algorithms for implicit deformable models," in *Fifth Intern. Conf. on Comp. Vision*, IEEE, IEEE Computer Society Press, 1995.

[23] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi, "Gradient flows and geometric active contour models," in *Fifth Int. Conf. on Comp. Vision*, pp. 810–815, IEEE, IEEE Computer Society Press, 1995.

[24] A. Yezzi, S. Kichenassamy, A. Kumar, P. Olver, and A. Tannenbaum, "A geometric snake model for segmentation of medical imagery," *IEEE Transactions on Medical Imaging*, vol. 16, pp. 199–209, April 1997.

[25] L. Lorigo, O. Faugeraus, W. Grimson, R. Keriven, and R. Kikinis, "Segmentation of bone in clinical knee MRI using texture-based geodesic active contours," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI '98)* (W. Wells, A. Colchester, and S. Delp, eds.), pp. 1195–1204, October 1998.

[26] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations," *Jrnl. of Comp. Phys.*, vol. 79, pp. 12–49, 1988.

[27] X. Xue and R. Whitaker, "Variable-conductance, level-set curvature for image denoising," in *IEEE International Conference on Image Processing*, p. To Appear., October 2001.

[28] D. Adalstein and J. A. Sethian, "A fast level set method for propagating interfaces," *Jrnl. of Comp. Phys.*, pp. 269–277, 1995.

[29] D. Breen and R. Whitaker, "A level-set approach to 3D shape metamorphosis," *IEEE Transactions on Visualization and Computer Graphics*, p. To Appear., 2001.

[30] A. Lerios, C. D. Garfinkle, and M. Levoy, "Feature-Based volume metamorphosis," in *SIGGRAPH '95 Conference Proceedings* (R. Cook, ed.), Annual Conference Series, pp. 449–456, Addison Wesley, Aug. 1995.

[31] J. Kent, W. Carlson, and R. Parent, "Shape transformation for polyhedral objects," in *SIGGRAPH '92 Proceedings*, pp. 47–54, July 1992.

[32] J. Rossignac and A. Kaul, "AGRELS and BIPs: Metamorphosis as a bezier curve in the space of polyhedra," *Computer Graphics Forum (Eurographics '94 Proceedings)*, vol. 13, pp. C–179–C–184, September 1994.

[33] J. F. Hughes, "Scheduled Fourier volume morphing," in *Computer Graphics (SIGGRAPH '92 Proceedings)* (E. Catmull, ed.), vol. 26, pp. 43–46, July 1992.

[34] B. Payne and A. Toga, "Distance field manipulation of surface models," *IEEE Computer Graphics and Applications*, vol. 12, no. 1, pp. 65–71, 1992.

[35] D. Cohen-Or, D. Levin, and A. Solomivici, "Three-dimensional distance field metamorphosis," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 116–141, 1998.

[36] P. Getto and D. Breen, "An object-oriented architecture for a computer animation system," *The Visual Computer*, vol. 6, pp. 79–92, March 1990.

[37] D. Breen, S. Mauch, and R. Whitaker, "3D scan conversion of CSG models into distance volumes," in *Proceedings of the 1998 Symposium on Volume Visualization*, pp. 7–14, ACM SIGGRAPH, October 1998.

[38] A. Middleditch and K. Sears, "Blend surfaces for set theoretic volume modeling systems," in *SIGGRAPH '85 Proceedings*, pp. 161–170, July 1985.

[39] J. Bloomenthal and K. Shoemake, "Convolution surfaces," in *SIGGRAPH '91 Proceedings* (T. W. Sederberg, ed.), pp. 251–257, July 1991.

[40] M. Desbrun and M.-P. Gascuel, "Animating soft substances with implicit surfaces," in *SIGGRAPH '95 Proceedings*, pp. 287–290, August 1995.

[41] R. T. Whitaker, "Volumetric deformable models: Active blobs," in *Visualization In Biomedical Computing 1994* (R. A. Robb, ed.), (Mayo Clinic, Rochester, Minnesota), pp. 122–134, SPIE, 1994.

[42] R. T. Whitaker, "A level-set approach to 3D reconstruction from range data," *Int. Jrnl. of Comp. Vision*, vol. October, no. 3, pp. 203–231, 1998.

[43] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proc. of SIGGRAPH '94*, pp. 311–318, ACM SIGGRAPH, August 1994.

[44] Y. Chen and G. Médioni, "Fitting a surface to 3-D points using an inflating ballon model," in *Second CAD-Based Vision Workshop* (A. Kak and K. Ikeuchi, eds.), vol. 13, pp. 266–273, IEEE, 1994.

[45] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. of SIGGRAPH '96*, pp. 303–312, ACM SIGGRAPH, August 1996.

[46] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *Computer Graphics*, vol. 26, no. 2, pp. 71–78, 1992.

[47] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt, "Reliable surface reconstruction from multiple range images," in *Euro. Conf. on Comp. Vision*, Springer-Verlag, 1996.

[48] C. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3D scans," in *SIGGRAPH '95 Proceedings*, pp. 109–118, August 1995.

A Level-Set Approach to 3D Reconstruction From Range Data

Ross T. Whitaker
School of Computing
University of Utah
Salt Lake City, UT 84112-9205
email: whitaker@cs.utah.edu

Abstract

This paper presents a method that uses the level sets of volumes to reconstruct the shapes of 3D objects from range data. The strategy is to formulate 3D reconstruction as a statistical problem: find that surface which is mostly likely, given the data and some prior knowledge about the application domain. The resulting optimization problem is solved by an incremental process of deformation. We represent a deformable surface as the level set of a discretely sampled scalar function of 3 dimensions, i.e. a volume. Such *level-set models* have been shown to mimic conventional deformable surface models by encoding surface movements as changes in the greyscale values of the volume. The result is a voxel-based modeling technology that offers several advantages over conventional parametric models, including flexible topology, no need for reparameterization, concise descriptions of differential structure, and a natural scale space for hierarchical representations. This paper builds on previous work in both 3D reconstruction and level-set modeling. It presents a fundamental result in surface estimation from range data: an analytical characterization of the surface that maximizes the posterior probability. It also presents a novel computational technique for level-set modeling, called the sparse-field algorithm, which combines the advantages of a level-set approach with the computational efficiency and accuracy of a parametric representation. The sparse-field algorithm is more efficient than other approaches, and because it assigns the level set to a specific set of grid points, it positions the level-set model more accurately than the grid itself. These properties, computational efficiency and sub-cell accuracy, are essential when trying to reconstruct the shapes of 3D objects. Results are shown for the reconstruction objects from sets of noisy and overlapping range maps.

Appears in: *The International Journal of Computer Vision* , 29(3), October, 1998, pp. 203–231.

1 Introduction

There are two aspects to this research. The first aspect is the application, which is the development of 3D models from range data. That problem is formulated using a statistical approach, which maximizes the posterior probability of a surface. The result is a surface that is described by a nonlinear objective function. That optimization is solved using a variational approach and a gradient-descent method. This optimization produces an evolution equation for deforming a surface so that it matches a given set of data. The work in this paper uses the level sets of volumes as a means of representing and manipulating object shapes. The second aspect of the research is the enhancement of the underlying level-set technology in order to make it suitable for problems in 3D reconstruction. This section introduces the issues surrounding those two aspects of the work, and lays out the overall structure of the paper.

1.1 3D Reconstruction

When investigating 3D reconstruction, it is necessary to describe the kind of data being considered. This work deals primarily with data that is from range or distance sensors that have some kind of "scanning capability". Three different properties of this data are important. First, the data is range or distance (i.e., direct 3D) compared to intensity or luminance, as one would expect from an X-ray or a video camera. Second, the data is relatively dense, rather than a sparse scattering of 3D points. This density is obtained by rotating or translating a device that takes a discrete set of 3D measurements. From a single point of view the range finder sweeps out a volume, and there is a 2D topology on these range measurements that is induced by the motion of the scanning device. The third important aspect of this data is that it is noisy. Thus, the 3D reconstruction problem is not strictly geometric; it is also statistical. For this work we assume that algorithms for calibrating the range sensor and registering the multiple views are taken from the variety of automated and semiautomated techniques that are in the literature — e.g. (Besl and McKay 1992, Chen and Medioni 1992, Zhang 1994).

Under the circumstances described above, approaches to 3D reconstruction can be distinguished based on the "level" of the models that one uses. High-level models have been used extensively in computer vision — some example are given in (Jain and Jain 1990, Jain and Flynn 1993). High-level models are those that represent specific objects or classes of objects. Such approaches can work with relatively little information (such as sparse sets of features), and when they work they can lead directly to higher level processes such as recognition. High-level approaches usually work best in situations where the domain is restricted and well characterized, and where the distinguishing shape characteristics of a particular object can be captured by relatively few parameters. Typically, such reconstructions do not capture those details that are unique to a particular object.

Another class of approaches could be characterized as "mid level". Mid-level approaches to reconstruction are characterized by the use of geometric primitives that can be combined in various ways to form more complicated objects. The assumption is that the objects in the domain can be decomposed into finite set of such primitives. This strategy can work quite well on certain domains, such as man-made objects—many of which were designed using such primitives. The fitting of primitives can also provide important structural information such as part/sub-part decomposition. The results of such systems are generally some mixture of the data and the models, but they tend to strongly reflect the shapes of the models or primitives that are chosen.

Finally, one could characterize the class of "low level" approaches to 3D reconstruction. These approaches use a few general assumptions about the domain; assumptions are often at the level of basic surface geometry, e.g. continuity. Low-level approaches can capture more detail but often provide very little higher level information. For this reason such approaches are well suited for applications that are geared to visualization by a human, rather than recognition by some automated system. These approaches form surfaces from sets of 3D data by performing a fusion of different scans, but they typically impose very little structure on the data. The performance of low-level approaches tends to degrade gradually as the assumptions about the environment and the sensor are violated. However, low-level approaches typically require more data in order to give useful results. The voxel-based approach in this paper is a low-level approach and is therefore best suited to applications that provide a relatively large amount of data that must be fused to produce a result with a great amount of detail.

1.2 Level-set models

The strategy taken within this research is to pose the reconstruction problem as the process of finding the surface or set of surfaces that are most likely to have given rise to the data. Thus, it is a Maximum A Posteriori (MAP) strategy which can combine the data with known properties or tendencies of the surfaces being measured. Finding those surfaces with the highest likelihood is generally a nonlinear optimization problem.

A common and often effective strategy for solving such optimization problems is to use a variational approach. That is, start with an initial solution and perturb or deform that solution so that it improves the overall likelihood. Thus, the variational approach requires a modeling technology that can undergo such goal-driven deformations. Such models, often called *deformable models*, are used extensively in the computer graphics and computer vision literature, e.g. (Kass, Witkin and Terzopoulos 1987, Staib and Duncan 1992, Miller, Breen, Lorensen, O'Bara and Wozny 1991, Terzopoulos, Witkin and Kass 1988). These models typically undergo evolution processes which implement some type of hill-climbing strategy on the objective function.

Conventional geometric models are parametric. That is, they are mappings from one space, which coincides with the dimensionality of the model, to another, which is the range. Typically, these parameterizations rely on a set of basis functions that span some finite subspace of all possible shapes. Thus when considering deformations, parametric models have several drawbacks which make them inadequate for certain kinds of applications. The dependency on the parameterization and an associated basis is critical; it limits the kinds of shapes a model can represent. Models typically do not deform far from their initial conditions without some sort of reparameterization. Such reparameterizations are often inefficient and developed on the basis of heuristics rather than a consistent mathematical foundation. The parameterization can also make it difficult to measure the intrinsic geometry of the model, as with polygonal meshes for instance.

An alternative to a parametric model is an implicit model, i.e., specifying a model as a level set of a scalar function, ϕ. This scalar function can be sampled on a discrete rectilinear grid. Such a *level-set* representation (Sethian 1996) has a number of practical and theoretical advantages over conventional parametric models, especially in the context of deformation and reconstruction. First, level-set models are topologically flexible, that is, the models can "split" into pieces to form multiple objects (Malladi, Sethian and Vemuri 1995, Whitaker and Chen 1994). Second the evolution of the embedding ϕ is a differential expression that is invariant to orthogonal group transformations (rotations and translations). The shapes formed by the level sets of ϕ are restricted only by the resolution of the discrete sampling used to represent ϕ. Finally, the representation of deformable models in terms of a *multidimensional image* provides a method for multi-scale or multi-grid solutions. For instance solutions can start on a relatively coarse grid and then proceed to progressively finer grids as the energy reaches a minimum (Whitaker 1995). Such a multigrid strategy reduces computation time, controls the relative importance of large and small-scale structures in the input image, and helps to simplify the objective function.

Despite these advantages, level-set models suffer from several drawbacks compared with parametric models. One disadvantage is the large number of computations needed to solve these equations over the entire range. Surface deformations in 3D, for example, require solutions to a 3D, nonlinear, partial differential equation: a significant computational task. Another drawback of level-set models is the absence of any direct, efficient representation of the surface during deformation. Such direct representations are useful when incorporating forces that depend on the multi-local or global information. A third drawback is the finite resolution that is imposed by the discrete approximation to the scalar field. Level-set models are limited by resolution rather than shape.

These problems are significant in the context of 3D reconstruction. The sparse-field algorithm described in this paper addresses these issues by representing a surface as both a discretely-sampled 3D field and a subset of that field which consists of a set of grid points (or voxels), called *active points*, through which the model passes. At each time step in the deformation process only a thin layer of voxels near the active points are visited and updated. In this way the computational complexity of the algorithm grows with the dimensionality of the surface, rather than the dimensionality of the volume. The set of active points are kept in a linked list which enables quick and efficient access to a set of points near or on the model.

The remainder of this paper proceeds as follows. After a brief review in Section 2 of other relevant work, Sec-

tion 3 presents the formulation of 3D reconstruction as a process of finding the most likely surface given a set of range data and some general knowledge about the relative likelihoods of different surface properties. Solving that problem requires the ability to manipulate or deform surface shape in a systematic manner. Section 4 shows how this deformation can be achieved via an implicit representation, using the level sets of a discretely-sampled scalar function of 3D. That section discusses the numerical methods that are used to solve the equations of motion. Section 5 describes an efficient computational technique, the sparse-field algorithm, and presents the results of an empirical analysis which shows that the sparse-field algorithm gives solutions that compare favorably to previously proposed algorithms. Additionally, the sparse-field algorithm is shown to overcome the aliasing artifacts associated with other level-set approaches and thereby achieves greater accuracy compared to those approaches. Section 6 shows how the level-set technology can be applied to the problem of 3D reconstruction. It shows how using level-set models within the MAP reconstruction framework provides new capabilities for surface reconstruction.

2 Related work

Several low- and mid-level reconstruction approaches described in the literature are worth noting. Turk and Levoy (1994) describe a "zippering" algorithm that combines triangle meshes, each representing a range map of the same object from a different point of view. Chen and Médioni (1994) use a triangle mesh which expands inside a sequence of range maps. This strategy is similar to that of Miller et al. (1991) which was demonstrated on 3D medical data.

Several low-level volumetric approaches have been proposed. Chien and Aggarwal (1989) have used binary volumes with the aid of oct-trees to reconstruct objects from multiple silhouettes. The focus is on using silhouettes and object features to do recognition. Muraki (1991) uses implicit or blobby models to reconstruct objects from range data. The individual blobs are spherically symmetric 3D potentials that are combined linearly so that they blend together. The global nature of the potentials makes updates somewhat expensive, thus limiting the number of primitives that can be efficiently computed.

Hoppe, DeRose, Duchamp, McDonald and Stuetzle (1992) proposed an implicit strategy which constructs a 3D field based on the signed distance transform of tangent planes generated from unordered 3D data. Regions are always associated with the nearest points, and therefore there is little or no averaging. Curless and Levoy (1996) describe a volume-based technique for combining range data. They use the signed distance transform to encode volume elements with data that represent the averages (with some allowance for outliers) of multiple measurements. Surfaces of objects are the level sets of volumes. They show that the resolution of the scanner can be overcome by such an averaging technique. Hilton, Stoddart, Illingworth and Windeatt (1996) describe a similar algorithm which is an extension to in (Hoppe et al. 1992) that accounts for interference at occluding contours and thin objects.

Another volumetric approach, which is more of a "mid-level" strategy, is that of (DeCarlo and Metaxas 1995), in which they use a combination of primitives that can deform, split, and even change topology in order to match the input data. The computer vision literature shows numerous mid-level approaches that fit volumetric models to range data—see (Dickinson, Metaxas and Pentland 1997), for example.

In robotics several researchers (Elfes 1989, Moravec 1988) have investigated the use of *occupancy grids*. The sensor model and the methods for combining sensor measurements from different points of view follow from a Bayesian formulation. Strictly speaking an occupancy grid does not give a 3D reconstruction; occupancies do not give the most likely surfaces.

With regard to deformable models, there are several parametric approaches that seek to overcome the usual topological limitations of parametric models. DeCarlo and Metaxas (1995) allow the primitives associated with an object to make discrete changes as a result of a set of rules and thresholds. Szeliski, Tonnesen and Terzopoulos (1993) propose systems of oriented particles that join together, disconnect, and rejoin in order to change topology. McInerny and Terzopoulos (1995) propose parametric models that are reparameterized based on a local grid structure that allows for changes in topology. Despite these advances, for 3D reconstruction the level-set approach offers several practical advantages, which are discussed in more detail in the sections that follow.

With regard to level-set models, this work draws on a number of recent developments in computational physics

and computer vision. Osher and Sethian (1988) have proposed the embedding of wavefront propagations that model physical systems, and they have developed numerical schemes, called *up-wind* schemes, to solve the resulting equations. In image processing Alvarez, Guichard, Lions and Morel (1992) have proposed a morphological scale space for planar curves which relies on geometric invariant curve evolutions. In computer vision Kimia, Tannenbaum and Zucker (1992) have proposed a reaction-diffusion space in which singularities represent basic properties of 2D shape. They use the same strategy as (Osher and Sethian 1988) of embedding planar curves in order to create geometric flows that are independent of any particular parameterization.

Malladi et al. (1995) have proposed a segmentation scheme based on a wavefront propagation that allows seed points to contract or expand. The author (Whitaker and Chen 1994) has shown that image "forces" (the term used in the sense of first-order physics) can drive level sets toward interesting features in images or volumes, as with conventional deformable models, while geometric flows can be used to enforce smoothness and continuity. Caselles, Kimmel and Sapiro (1995) have reached a similar formulation using a conformal mapping and have shown that the equations resulting from moving level sets in this manner are well posed. Adalstein and Sethian (1995) have proposed a narrow-band scheme, with a finite band of 6-12 grid points on either side of the level set, for reducing the computations associated with level-set models. Results in successive sections of this paper will show the ability to reduce the width of this narrow band to a single grid point will improve computational performance and accuracy.

This paper makes several contributions to previous work in this field. One is a statistical formulation of the 3D reconstruction problem that gives an optimal surface estimate while making very few assumptions about the scene. This formulation incorporates a noise model and may include some prior knowledge about the relative likelihoods of different kinds of surfaces. The result is quite general, but it is particularly well suited for level-set models. This paper presents examples that show effectiveness of the level-set approach for this application. Another contribution is the numerical scheme for implementing the level-set models. This new scheme compares favorably with previously proposed methods both in terms of computational efficiency and accuracy.

3 A MAP Reconstruction Strategy

A range finder is a device that produces a distance measurement along a particular line of site. The distance measurement and, to a lesser degree, the direction of the line of sight are corrupted by noise. For the purposes of this work we assume that the noise is additive and Gaussian. When the range finder is combined with a scanning mechanism, it produces a 2D array of data, and the noise is often assumed to be independent from one element of the array to another. In the event that the noise is non-stationary, we assume that the variability of the noise process is known (e.g. as a function of object distance or scanner orientation). The scanning mechanism is typically calibrated, enabling one to calculate the direction of the line of sight associated with each measurement in the 2D array. In the event that the scanning mechanism does not produce a 2D array, but rather an unordered list of range measurements taken in different directions, one can impose a 2D topology on the data by the use of nearest-neighbor relationships such as Delaunay triangulation. A single 2D array of range data taken from a single location of the scanner is called a *range map*. One of the goals of this work is to combine the information from a collection of range maps.

The above sensor model has some significant shortcomings. First, the noise is rarely additive Gaussian. Outliers, for instance, are a problem. The noise can often be related to surface characteristics, and therefore could be correlated. In the case of structured light sensors, such as the one used for some of the results in this paper, the data contains occluded regions for which there is no data at all. As a theoretical consideration, Gaussian noise allows finite probabilities for measurements behind the scanner. In cases where the noise level is some appreciable fraction of the the distance from the scanner (very rare in practice), a log-normal distribution for the noise might be more appropriate.

In this work, we start with the simple sensor model, i.e. independent, additive, Gaussian noise, to generate the basic algorithm and then modify that algorithm to account for some of these other complicating factors. The outliers, for instance, are handled within the minimization process rather than the underlying model. Areas where the sensor failed (either by lack of signal or occlusion) are handled by giving such points low confidence, i.e. large variance, so that they have little effect on the final results. The formulation that results from these assumptions is optimal for

independent, additive, Gaussian noise, but later sections will show that it is useful when the noise is more complex (such as in the case of outliers).

The strategy in this work is to use a maximum a posteriori (MAP) approach to construct an expression for the surface likelihood conditional on the measured data. The total error associated with a model converts into a volume integral, and the Euler-Lagrange of that volume integral defines the motion of a deformable model that seeks to maximize this likelihood.

Let $\mathcal{S}$, a compact subset of 3D, be the surface that encloses the object Ω, i.e., $\mathcal{S} = \partial\Omega$. Let $\{R^{(1)}, \ldots, R^{(n)}\}$ be a set of range maps, each of which consists of set of 3D points $R^{(i)} = \{\mathbf{r}_1^{(i)}, \ldots, \mathbf{r}_m^{(i)}\}$, which can arranged in a 2D grid, with coordinates u, v, defined by the scanning mechanism. Associated with each grid point is a ray, called the line of sight, along which the range measurement is taken. Assume that the rays within a single range map do not intersect, as is the case with the spherical, cylindrical, or linear projective scanning mechanisms that are used on most range finders.

The posterior probability of $\mathcal{S}$ is given by Bayes rule:

$$P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)}) = \frac{P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S})P(\mathcal{S})}{P(R^{(1)}, \ldots, R^{(n)})}. \tag{1}$$

The goal is to find the most likely surface model $\mathcal{S}$ to give rise to a particular collection of range data,

$$\sup_{\mathcal{S}} \left[P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)})\right] = \sup_{\mathcal{S}} \left[P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S})P(\mathcal{S})\right] \tag{2}$$

where the denominator $P(R^{(1)}, \ldots, R^{(n)})$ is removed because it is a normalization factor that does not depend on $\mathcal{S}$. The term $P(\mathcal{S})$ is the prior, which reflects the fact that within the set of possible surfaces, some are more likely than others, independent of the data.

The range maps, conditional on the surface position, are independent random variables (this ignores the effects of surface properties on error, as discussed above), and thus,

$$P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S}) = P(R^{(1)}|\mathcal{S}) \ldots P(R^{(n)}|\mathcal{S}) = \prod_i P(R^{(i)}|\mathcal{S}). \tag{3}$$

Each $R^{(i)}$ consists of an array of 3D range data that can be represented as $\mathbf{r}_{u,v}^{(i)}$. Within a given range map each range measurement is a independent random variable, and therefore

$$P(R^{(1)}, \ldots, R^{(n)}|\mathcal{S}) = \prod_i \prod_{u,v} P(\mathbf{r}_{u,v}^{(i)}|\mathcal{S}). \tag{4}$$

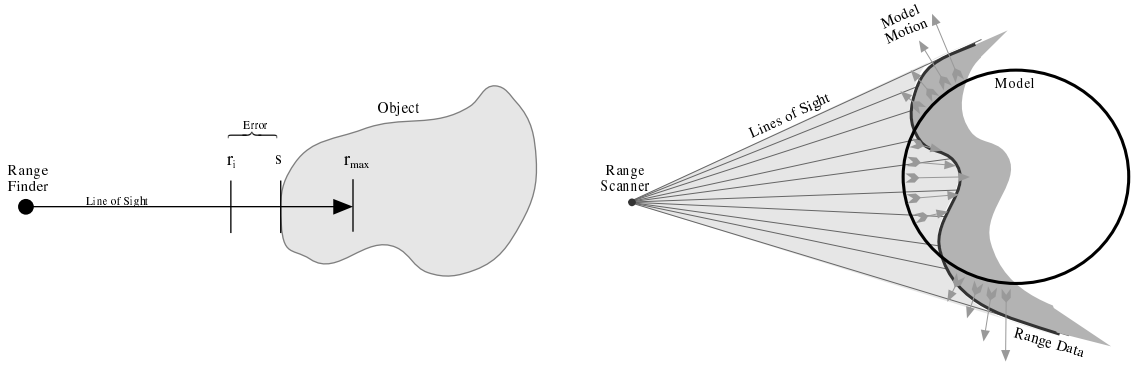
As is typical with such MAP algorithms, the maximization is performed as a minimization on the negative logarithm of the probability:

$$\sup_{\mathcal{S}} \left\{\ln\left(P(\mathcal{S}|R^{(1)}, \ldots, R^{(n)})\right)\right\} = \inf_{\mathcal{S}} \left\{-\sum_i \sum_{u,v} \ln P(\mathbf{r}_{u,v}^{(i)}|\mathcal{S}) - \ln P(\mathcal{S})\right\}. \tag{5}$$

The right-hand side of equation (5) reflects the combination of terms that is typical with MAP reconstructions. The terms are of two different types: those terms that enforce the solution to look like the data and those terms that enforce the prior, i.e., that encourage the solutions to conform to those expectations that are independent of the data.

Each individual range measurement is dependent not on the surface $\mathcal{S}$ as a whole, but on that particular point on the surface that is first intersected along the line of sight from the scanner location. Consider a single range reading r_i, represented as a distance along its associated line of sight. Suppose that the distance to the surface along that line of sight is given by s_i. The Gaussian noise model gives the following probability density function for individual range readings,

$$-\ln P(r_i|\mathcal{S}) = \frac{1}{\sigma_i^2}(s_i - r_i)^2 + \frac{1}{\sqrt{2\Pi\sigma}} = c_i(s_i - r_i)^2 + k(c_i), \tag{6}$$



(a) (b)

Figure 1: The MAP reconstruction strategy: (a) The squared error along the line of sight can be written as a an integral that includes a binary membership function for the object. (b) A single range scan, consisting of a number of measurements along lines of sight emanating from the point at the left, creates surface in 3D (shown here as a curve in the plane). A circular model deforms to fit the data (motion indicated by arrows).

where the variance σ_i^2 associated with a range reading r_i is represented as a confidence measure $c_i = 1/\sigma_i^2$. The term $k(c_i)$ can be ignored, because it does not depend on surface position and will not affect the outcome of the optimization. The squared distance of the surface to the range measurement can be represented as an integral

$$(s_i - r_i)^2 = \int_0^{r_{\max}} (\alpha - r_i)\, I_s(\alpha) \mathrm{d}\alpha - \int_{r_i}^{r_{\max}} (\alpha - r_i)\, \mathrm{d}\alpha \tag{7}$$

where $I_s(\alpha)$ is a function that is unity inside the object and zero outside, and $r_{\max}$ indicates the maximum effective range of the range finder. The term $r_{\max}$ serves as a bound on the error if the line of sight fails to intersect the surface, and it also serves to limit the range of influence of a particular range reading. That is, surface points that lie beyond $r_{\max}$ have no impact on the conditional likelihood associated with that range measurement.

Strictly speaking, this formulation is valid only if the surface intersects the line of sight at most once between the scanner and $r_{\max}$, as shown in figure 1(a). Thus, the segment along the line of sight with length $r_{\max}$ should be long enough to penetrate the model (as it deforms) but not so long that it emerges from the other side or enters the object again, as could easily happen with self occlusions. Typically, $r_{\max}$ should depend on r_i, e.g. $r_{\max} = r_i + \varepsilon$. The use of $r_{\max}$ and ε serves to limit the effects of the range data and thereby simplifies the mathematical model by ignoring self occlusions of the object. Unfortunately, in order for this to work properly, ε must be chosen so that scans to do not interfere with each other in areas where the object self occludes. This is not always possible, but with modifications to the minimization algorithm, discussed in successive sections, this problem can be all but eliminated.

Instead of putting bound $r_{\max}$ on the integral, we can use a *windowing function*, ω, that nullifies the effects of surface that lie beyond $r_{\max}$, i.e.,

$$\omega(r_{\max}, \alpha) = \begin{cases} 1 & \text{if } r_{\max} - \alpha \geq 0 \\ 0 & \text{if } r_{\max} - \alpha < 0 \end{cases} \tag{8}$$

The error becomes

$$(s_i - r_i)^2 = \int_0^{\infty} (\alpha - r_i)\, I_s(\alpha) \omega(r_i + \varepsilon, \alpha) \mathrm{d}\alpha - \int_{r_i}^{\infty} (\alpha - r_i)\, \omega(r_i + \varepsilon, \alpha) \mathrm{d}\alpha. \tag{9}$$

Given this formulation, the windowing function $\omega(\cdot)$ need not be binary; it could implement a fuzzy cutoff of the influence of the range data. Results in later sections use a Gaussian centered at the range reading with standard deviation ϵ.

To extend this formulation to include all of the samples in a single scan, R, we represent the distance along the line of sight in 3D. Let $\mathbf{n}_i$ be the unit vector along the line of sight and $\mathbf{r}_i = r_i \mathbf{n}_i$ be the 3D location of the ith range reading. Then

$$\begin{aligned} -\ln\left(P(R|\mathcal{S})\right) &= -\ln\left(P(\mathbf{r}_1, \ldots, \mathbf{r}_n|\mathcal{S})\right) \\ &= \sum_i c_i \int_0^\infty \left((\alpha\mathbf{n}_i - \mathbf{r}_i)\cdot\mathbf{n}_i\right) \omega\left((\alpha\mathbf{n}_i - \mathbf{r}_i)\cdot\mathbf{n}_i\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}_i)\mathrm{d}\alpha - k \\ &= \sum_{u,v} c_{u,v} \int_0^\infty \left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right) \omega\left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}_{u,v})\mathrm{d}\alpha - k \end{aligned} \tag{10}$$

where $I_{\mathcal{S}} : \mathbb{R}^3 \mapsto \mathbb{R}$ is unity inside the object and zero otherwise, and

$$k = \sum_{u,v} c_{u,v} \int_{r_{u,v}}^\infty \omega\left((\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\right)\ (\alpha\mathbf{n}_{u,v} - \mathbf{r}_{u,v})\cdot\mathbf{n}_{u,v}\ \mathrm{d}\alpha, \tag{11}$$

which does not depend on the surface. The notation u, v refers to the fact that the measurements from a single range map are arranged in a 2D grid. If we assume that this grid is relatively dense we can approximate the likelihood of the jth scan as an integral:

$$\begin{aligned} -\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int\int\int & c(\varphi,\theta)\left((\alpha\mathbf{n}(\varphi,\theta) - \mathbf{r}(\varphi,\theta))\cdot\mathbf{n}(\varphi,\theta)\right) \\ & \omega\left((\alpha\mathbf{n}(\varphi,\theta) - \mathbf{r}(\varphi,\theta))\cdot\mathbf{n}(\varphi,\theta)\right)\ I_{\mathcal{S}}(\alpha\mathbf{n}(\varphi,\theta))\ \mathrm{d}\alpha\ \mathrm{d}\varphi\ \mathrm{d}\theta - k, \end{aligned} \tag{12}$$

where φ and θ are the continuous versions of u and v, respectively, and $\mathbf{n}(\cdot)$, $\mathbf{r}(\cdot)$, $c(\cdot)$, are continuous functions derived from some suitable interpolation (e.g., bilinear) of their discrete counterparts.

Because the rays that define the lines of sight associated with a single scan do not cross, there is unique mapping from each point within the 3D subspace swept out by the range finder to the point (φ, θ) within the range map that has a line of sight passing through that 3D point. Therefore, the volume integral of equation (13) can be reformulated in Cartesian coordinates:

$$-\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int_{\mathcal{R}} c(\mathbf{x})\ D(\mathbf{x})\ \omega\left(D(\mathbf{x})\right)\ I_{\mathcal{S}}(\mathbf{x})\gamma(\mathbf{x})\ \mathrm{d}\mathbf{x}\ - k, \tag{13}$$

where $\gamma(\mathbf{x})$ is an integration factor, such that $\gamma(\mathbf{x})\mathrm{d}\mathbf{x} = \mathrm{d}\varphi\mathrm{d}\theta\mathrm{d}\alpha$. The term $D(\mathbf{x}) = \left((\mathbf{x} - \mathbf{r}(\mathbf{x}))\cdot\mathbf{n}(\mathbf{x})\right)$ is the signed distance from the surface position to the range measurement, and $\mathcal{R}$ is the volume swept out by the range finder. If we define the confidence $c(\mathbf{x})$ to be zero for any point outside the space swept out by the scanner, then one can extend bounds of the integral to $\mathbb{R}^3$. Because, $I_{\mathcal{S}}$ is a binary function that is one only within the object, the integral can be re-expressed over the object itself

$$-\ln\left(P(R^{(j)}|\mathcal{S})\right) \approx \int_{\Omega} c(\mathbf{x})\ D(\mathbf{x})\ \omega\left(D(\mathbf{x})\right)\ \gamma(\mathbf{x})\ \mathrm{d}\mathbf{x}\ - k. \tag{14}$$

The gradient descent on the posterior for a single range map is therefore

$$\frac{\partial\mathcal{S}}{\partial t} = -c(\mathcal{S})\ D(\mathcal{S})\ \omega\left(D(\mathcal{S})\right)\ \gamma(\mathcal{S})\ \mathcal{N}, \tag{15}$$

where $\mathcal{N}$ is the surface normal.

Equation (15) describes the motion of a model as it seeks to maximize its likelihood of giving rise to a single range map R, which is set of range measurements. A single range map creates a surface in 3D (shown in figure 1(b) as a curve in the plane), and the model expands or contracts, with a magnitude proportional to the distance along the line of sight to the range reading, so that it exactly fills the volume behind that surface. The windowing function $\omega(\cdot)$ controls the depth of the region behind the range surface over which this expanding or contracting action occurs. The

function $\gamma(\cdot)$ describes the unit-volume relationship between the scanner coordinates and cartesian coordinates. For the results in this paper we assume that the rays (lines of sight) emanating from the scanner are nearly parallel and regularly sampled, so that $\gamma(\cdot)$ is a constant.

The effect of multiple range maps is additive, and the Euler-Lagrange of the combined posterior is

$$\frac{\partial \mathcal{S}}{\partial t} = -g(\mathcal{S})\mathcal{N} + \rho(\mathcal{S}), \tag{16}$$

where

$$g(\mathbf{x}) = \sum_j c^{(j)}(\mathbf{x})\ D^{(j)}(\mathbf{x})\ \omega\left(D^{(i)}(\mathbf{x})\right)\ \gamma^{(j)}(\mathbf{x}), \tag{17}$$

and the superscripts indicate the range, line-of-sight, and confidence functions associated with a particular range map. The term $\rho(\mathcal{S}) = -\delta \ln P(\mathcal{S})$ is the Euler-Lagrange of the prior. In the absence of any prior (i.e., uniform prior), $\rho(\mathcal{S}) = 0$, and maximizing the combined effects of a sequence of range maps is linear; the solution is given by the zero crossings of the 3D function defined in equation (17). Indeed, the algorithms proposed by (Curless and Levoy 1996), as well as (Hoppe et al. 1992) and (Hilton et al. 1996), can be formulated as a special cases of this MAP approach, with a particular choice of $\omega(\cdot)$ and $c(\cdot)$, and with a uniform prior.

There are several reasons for going to an iterative scheme for finding optimal solutions. First is the use of a prior. In surface reconstruction, even a very low level of noise can degrade the quality of the rendered surfaces in the final result, and in such cases better reconstructions can be obtained by introducing a prior. Second is aliasing. Discretizing $g(\mathbf{x})$ and finding the zero crossings will cause aliasing in those places where the transition from positive to negative is particularly steep. A deformable model can place the surface much more precisely. The third reason for going to an iterative scheme is that despite the windowing function $\omega(\mathbf{x})$ there is interference between different range maps at places of high curvature. This problem is addressed by introducing a nonlinearity which is solved in an iterative scheme as described in Section 6.1. In this work, solutions of the linear problem will serve as the initial conditions for the nonlinear, iterative optimization strategy that results from the inclusion of a prior and a nonlinear term that compensates for lack of any explicit model of self occlusions.

3.1 The Prior

There are numerous possibilities for selecting priors that are consistent with the MAP formulation of the previous section. These possibilities range from high-level priors that bias the solution to look like certain shapes (Johnson 1993) to low-level priors that enforce some very general properties on those shapes. The goal of this work is to obtain a somewhat general reconstruction algorithm that can be subsequently tuned to specific applications. Therefore, we use a low-level prior that biases solutions toward smooth, continuous surfaces, like those associated with many man-made objects or anatomical structures. Of course, the selection of a prior will depend strongly on the application; the choice of prior for the reconstruction of injection-molded, hand-held objects should probably differ from the prior needed to reconstruct the shapes of highly irregular objects, such as the human brain with its many folds and protrusions.

For many applications, a natural choice of prior is to penalize the integral of the normalized first derivatives on the object surface, i.e., penalize surface area. This choice is based on the heuristic that many physical processes, both synthetic and natural, tend to conserve surface area and produce objects that reflect, at some level of detail, this tendency to minimize area. Alternatively, one could say that given a set of surfaces that are near the data, the algorithm should choose a surface that has less area. Often, but not always, this will be the smoother surface. A probability distribution for the prior that reflects this principle is

$$P(\mathcal{S}) = \left(\frac{\beta}{\Pi}\right)^{\frac{1}{2}} \exp\left(-\beta \int_{\mathcal{S}} \mathcal{N} \cdot \mathcal{N} \mathrm{d}\mathbf{x}\right) \tag{18}$$

After the logarithm, the Euler-Lagrange of this quantity is the mean curvature of the surface:

$$\rho(\mathcal{S}(r,s,t)) = \beta H(\mathcal{S}) = \beta \left(\frac{\partial^2 \mathcal{S}}{\partial r^2} + \frac{\partial^2 \mathcal{S}}{\partial s^2} \right) \tag{19}$$

where r, s is a local, orthonormal parameterization.

The term β controls the probability distribution of surfaces in the prior. A larger β means that the prior favors more strongly surfaces that are smoother. The parameter β could be set in any number of different ways. One way would be to "calibrate" the system by scanning a number of known objects and determine which values of β give the most accurate/reliable reconstructions. Another method would be to take a collection of objects that represent the application domain, model them, and generate a probability density function for surface area, possibly at different scales or resolutions. For this work, we treat β as a free parameter that must be tuned by the person using the algorithm.

The existence of a free parameter β represents a typical property of signal processing systems that must deal with noise in the absence of specific information about the signal. Results in the following sections will show that the algorithm is somewhat robust with respect to this free parameter, β, and that the algorithms fails gradually as this parameter is set too high or low. This paper will also show that very small values of β can reduce the effects of uncorrelated noise without distorting the overall shapes of the objects being reconstructed.

Despite the usefulness of this second-order smoothing term, it is somewhat limited in its effectiveness because it can interpolate only position and tends to create straight lines, flat surfaces, or singularities where there is little data. The topic of developing more effective low-level priors (Whitaker 1995), as well as incorporating high-level priors, is an area of ongoing investigation. Fourth-order terms, for example, could create structures with smoothly varying normals and allow the MAP approach to operate with less data (Sethian 1996). This paper will show that despite the limitations of the second-order smoothing given in equation (19), that prior is an improvement over none at all, especially in cases of noisy surface data.

4 Level-Set Models

The hill-climbing optimization strategy described in the previous section requires a modeling technology that is capable of accommodating incremental changes surface shape in an efficient manner. The equation of motion given by (16) makes no assumptions about the type surface model used to achieve the reconstruction. The MAP formulation could be adapted to any number of conventional, parametric surface models by expressing changes in surface position in terms of the parameters that control surface shape.

However, there are some drawbacks to using parametric deformable models. For instance, as models evolve and undergo large deviations from their original shapes, surface parameterizations often introduce de facto constraints. The expansion of polygonal models can create a kind of coarseness which prevents the model from capturing smaller structures; thus the evolution of polygonal models requires the creation and deletion of polygons (Miller et al. 1991, MacDonald, Avis and Evans 1994, Chen and Médioni 1994), or alternatively, a reparameterization (McInerny and Terzopoulos 1995, DeCarlo and Metaxas 1995). Also the choice a surface model with relatively few degrees of freedom, such as superquadric or superelipsoid, restricts solutions to the relatively small space of shapes that are captured by those modeling technologies.

An alternative to a parametric model, is a level-set model (Sethian 1996), i.e., a model that treats a 3D surface as the level-set of a discretely-sampled volume. Previous work has shown promising results with this modeling technology for 3D reconstruction, primarily in the context of 3D medical data (Malladi et al. 1995, Whitaker and Chen 1994, Whitaker 1994). This section gives the formulation for this modeling strategy, starting with an equation of motion for a deformable surface (Whitaker and Chen 1994). This formulation differs somewhat from that of (Caselles et al. 1995), which they develop by applying a conformal mapping to the energy function defined over the range.

The strategy is to rely on the properties of regular surfaces, which have local parameterizations that can be ex-

pressed in terms of intrinsic quantities such as arc length, curvature, etc. If one starts with a parameterized surface and the associated equations of motion, and then removes the parameterization from the deformable model, all that remains is intrinsic geometry, which is expressed in terms of the embedding, ϕ. The strategy for embedding active surfaces consists of four steps:

1. Express the equations of motion for a deformable model in terms of some unspecified parameterization (as was done in Section 3).
2. Describe the parameterization in terms of the differential structure of the model.
3. Assume the model is the level set of a function ϕ.
4. Express the geometry of the level set in terms of the differential structure of ϕ and create an evolution equation for ϕ.

In order to apply this strategy to a deformable surface $\mathcal{S}$, represent $\mathcal{S}$ as a level set of an 3D scalar function, $\phi : \mathbb{R}^3 \times \mathbb{R}^+ \mapsto \mathbb{R}$, which evolves over time. The evolution equations of the individual level surfaces imply corresponding evolution equations for the scalar function $\phi(\mathbf{x}, t)$, where $\mathbf{x} \in \mathbb{R}^3$.

A parametric deformable model, $\mathcal{S}(r, s, t)$ where $r, s \in U \subset \mathbb{R}$, can be represented

$$\mathcal{S} = \{\mathbf{x} | \phi(\mathbf{x}, t) = k\}. \tag{20}$$

The surface $\mathcal{S}$ remains a level set of ϕ over time, and therefore the time derivative is zero:

$$\frac{\partial \phi(\mathcal{S}, t)}{\partial t} + \nabla \phi(\mathcal{S}, t) \cdot \frac{\partial \mathcal{S}}{\partial t} = 0, \tag{21}$$

where

$$\nabla \phi(\mathbf{x}) \triangleq \left(\frac{\partial \phi}{\partial x}, \frac{\partial \phi}{\partial y}, \frac{\partial \phi}{\partial z} \right). \tag{22}$$

Thus,

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\partial \mathcal{S}}{\partial t} = |\nabla \phi| \frac{\partial \mathcal{S}}{\partial t} \cdot \mathcal{N}, \tag{23}$$

where $\mathcal{N} = -\nabla\phi / |\nabla\phi|$ is the surface normal.

The data term for reconstruction from equation (16) takes the form

$$\frac{\partial \mathcal{S}}{\partial t} = -g(\mathbf{x}) \mathcal{N} \tag{24}$$

where $g(\cdot)$ is the cumulative effect from all of the individual range maps as in equation (17). The level-set formulation, without the prior, becomes

$$\frac{\partial \phi}{\partial t} = g(\mathbf{x}) |\nabla \phi| \tag{25}$$

The mean curvature, used for the prior, from equation (19) is computed directly from the first- and second-order structure of ϕ,

$$\begin{aligned} |\nabla \phi| \ (\mathcal{S}_{rr} + \mathcal{S}_{ss}) \cdot \mathcal{N} \quad = \quad & \left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)^{-\frac{1}{2}} \left[\left(\phi_y^2 + \phi_z^2\right) \phi_{xx} + \left(\phi_z^2 + \phi_x^2\right) \phi_{yy} \right. \\ & \left. + \left(\phi_x^2 + \phi_y^2\right) \phi_{zz} - 2\phi_x \phi_y \phi_{xy} - 2\phi_y \phi_z \phi_{yz} - 2\phi_z \phi_x \phi_{zx} \right]. \end{aligned} \tag{26}$$

In the numerical implementation, the derivatives are calculated using centralized differences (Sethian 1996).

Combining the data term and the prior gives the following level-set formulation:

$$\begin{aligned} \frac{\partial \phi}{\partial t} \quad = \quad & g(\mathbf{x}) |\nabla \phi| + \beta \left(\phi_x^2 + \phi_y^2 + \phi_z^2\right)^{-\frac{1}{2}} \left[\left(\phi_y^2 + \phi_z^2\right) \phi_{xx} + \left(\phi_z^2 + \phi_x^2\right) \phi_{yy} \right. \\ & \left. + \left(\phi_x^2 + \phi_y^2\right) \phi_{zz} - 2\phi_x \phi_y \phi_{xy} - 2\phi_y \phi_z \phi_{yz} - 2\phi_z \phi_x \phi_{zx} \right] \end{aligned} \tag{27}$$

Equation (27) is invariant under certain kinds of geometric transformations. First, it is invariant to orthogonal group transformations on $\mathbf{x}$. Thus the position and orientation of the model or the data has no impact (to within the

error introduced by the representation of ϕ) on the solution. Equation (27) is also invariant to monotonic transformations on ϕ; that is, equation (27) acts only on level sets and treats each level set of ϕ as an individual surface evolving under the same set of set of priors and forces as all other level sets of ϕ.

4.1 Numerical Algorithms

The differential equation described by equation (27) has two parts. The first part is a first-order term that has the form $g(\mathbf{x})|\nabla\phi(\mathbf{x})|$. It is a moving wave front with a velocity that depends on position. This expression cannot be solved with a simple forward finite difference scheme. Such schemes tend to overshoot, and they are unstable. To solve this problem Osher and Sethian (1988) propose an *up-wind* scheme. The up-wind method uses a one-sided derivative that looks in the up-wind direction of the moving wavefront, and thereby avoids the overshooting associated with forward finite differences.

The one-sided derivatives are denoted by $\mathrm{d}^{(+)}$ and $\mathrm{d}^{(-)}$. Let $u_{x,y,z}$ with domain X be an approximation to ϕ defined on a discrete rectilinear grid with spacing h. The one-sided derivatives are

$$\mathrm{d}_x^{(+)} u_{x,y,z} \triangleq (u_{x+h,y,z} - u_{x,y,z})/h, \tag{28}$$

$$\mathrm{d}_x^{(-)} u_{x,y,z} \triangleq (u_{x,y,z} - u_{x-h,y,z})/h. \tag{29}$$

For the first term of equation (27) the up-wind scheme has the following form (Osher and Sethian 1988):

$$\begin{aligned} g(x,y,z)|\nabla u| = \Big(& \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_x^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_x^{(-)}u\right]^2 \\ & + \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_y^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_y^{(-)}u\right]^2 \\ & + \left[\max\left(g(x,y,z),0\right)\ \mathrm{d}_z^{(+)}u + \min\left(g(x,y,z),0\right)\ \mathrm{d}_z^{(-)}u\right]^2 \Big)^{\frac{1}{2}}, \end{aligned} \tag{30}$$

where the time steps are limited by the speed of the fastest moving wavefront,

$$\Delta t \leq \frac{1}{\sup_{x,y,z\in X}\{|\nabla g(x,y,z,t)|\}} \tag{31}$$

The second term in equation (27) is a diffusion term that can be solved using a finite forward difference scheme and does not require the up-wind method.

4.2 Narrow-Band Methods

If one is interested in only a single level set, the formulation described previously is not computationally efficient. This is because solutions are usually computed over the entire domain of u. The solutions, $u_{x,y,z}(t)$ describe the evolution of an embedded family of contours. While this dense family of solutions might be advantageous for certain applications, there are other applications that require only a single surface model. In such applications the calculation of solutions over a dense field is an unnecessary computational burden, and the presence of contour families can be a nuisance because further processing might be required to extract the level set that is of interest.

When solving for only a single level set, $\phi(\mathbf{x},t) = k$, the evolution of ϕ is important only in the vicinity of that level set. The evolution of the implicit models is such that the level sets evolve independently (to within the error introduced by the discrete grid). Thus, one should perform calculations for the evolution of ϕ only in a neighborhood of the surface $\mathcal{S} = \{\mathbf{x}|\phi(\mathbf{x}) = k\}$. In the discrete setting, there is a particular subset of grid points whose values control a particular level set (see figure 2).

Adalstein and Sethian (1995) propose a *narrow-band* approach that takes advantage of the fact that the movement of a particular level set is a local phenomenon. The narrow-band technique constructs an embedding of the evolving curve or surface via a signed distance transform. The distance transform is computed over a finite width of only m points, and the remaining points are set to constant values to indicate that they do not lie within the narrow band, or *tube* as they call it. The evolution of the surface (they demonstrate it for curves in the plane) is computed by

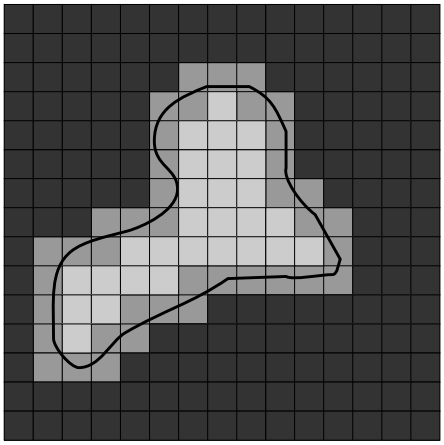


Figure 2: A level curve of a 2D scalar field passes through a finite set of cells. Only those grid points nearest to the level curve are relevant to the evolution of that curve.

calculating the evolution of u only on the set of grid points that are within a fixed distance to the initial level set, i.e. within the narrow band. When the evolving level set approaches the edge of the band, they calculate a new distance transform and a new embedding, and they repeat the process. This algorithm relies on the fact that the embedding is not a critical aspect of the evolution of the level set. That is, the embedding can be transformed or recomputed at any point in time, so long as such a transformation does not change the position of the kth level set, and the evolution will be unaffected by this change in the embedding.

Despite the improvements in computation time, the narrow-band approach is not optimal for several reasons. First it requires a band of significant width ($m = 12$ in the examples of Adalstein and Sethian (1995)) where one would like to have a band that is only as wide as necessary to calculate the derivatives of u near the level set (e.g. $m = 2$). The wider band is necessary because the narrow-band algorithm trades off two competing computational costs. One is the cost of stopping the evolution and computing the position of the curve and distance transform (to sub-cell accuracy) and determining the domain of the band. The other is the cost of computing the evolution process over the entire band. The narrow-band method also requires additional techniques, such as smoothing, to maintain the stability at the boundaries of the band, where some grid points are undergoing the evolution and nearby neighbors are stationary.

5 Sparse-Field Solutions

The narrow-band algorithm reduces computation time by restricting the updates to a *band* of grid points that lie near the level set. However, that strategy is based on the assumption that computing the distance transform is so costly that it cannot be done at every iteration of the evolution process — the band of computation must be wide enough to justify this costly update of the embedding.

The sparse-field algorithm proposed in this section uses an approximation to the distance transform that makes it feasible to recompute the neighborhood of the level-set model at each time step. Thus, it takes the narrow-band strategy to the extreme; it computes updates on a band of grid points that is only one point wide. The values of the points in the active set can be updated using the up-wind scheme and the mean-curvature flow described in the previous sections. When computing updates on so few points, however, one must be careful to maintain a neighborhood around those points so that the derivatives that control the process can be computed with sufficient accuracy. The strategy is to extend the embedding from the active points outward in layers to create a neighborhood around those points that is precisely the width needed to calculate the derivatives for the next time step.

This approach has several advantages. The algorithm does precisely the number of calculations needed to compute

the next position of the level curve. It does not require explicitly recalculating the positions of level sets and their distance transforms. For large 3D data sets, the very process of incrementing a counter and checking the status of all of the grid points is prohibitive. In the sparse-field algorithm the number of points being computed is so small, it is feasible to use a linked-list to keep track of them. Thus, at each iteration the algorithm *visits* only those points adjacent to the k-level curve. Also, the sparse-field approach identifies a single level set with a specific set of points whose values control the position of that level set. This allows one to compute external forces to an accuracy that is better than the grid spacing of the model, resulting in a modeling system that is more accurate for 3D reconstruction.

The k-level surface, S, of a function u defined on a discrete grid has a set of cells through which it passes, as shown in figure 2. The set of grid points adjacent to the level set is called the *active set*, and the individual elements of this set are called *active points*. As the model deforms the active will change. All of the derivatives (up to second order) required to calculate the update of u are computed using nearest neighbor differences. Therefore, only the active points and their neighbors are relevant to the evolution of the level-set at any particular time in the evolution process.

One important aspect of the sparse-field algorithm is the mechanism to control membership in the active set. In order to maintain stability, one must update the active set and neighboring points in a way that allows grid points to enter and leave the active set without those changes in status affecting their values. This mechanism can be understood as follows. Active points must be adjacent to the level-set model. Therefore their positions lie within a fixed distance to the model. Because the embedding is a distance transform, the values of u for locations in the active set must lie within a certain range of values. When active-point values move out of this *active range* they are no longer adjacent to the model. They must be removed from the set and other grid points, those whose values are moving into the active range, must be added to take their place. The precise ordering and execution of these operations is critical to the proper operation of the algorithm.

If we assume that the embedding u is a discrete approximation to the distance transform of the model, then the distance of a particular grid point, x_j, to the level set is given by the value of u at that grid point. If the distance between grid points is defined to be unity, then we should remove a point from the active set when the value of u at that point no longer lies in the interval $[-\frac{1}{2}, \frac{1}{2}]$ (see figure 3). If the neighbors of that point maintain their distance of 1, then those neighbors will move into the active range just as x_j is ready to be removed.

There are two operations that are significant to the evolution of the active set. First, the values of u at active points change from one iteration to the next. Second, as the values of active points move out of the active range they are removed from the active set and other, neighboring grid points are added to the active set to take their place. The appendix of this paper gives some formal definitions of active sets and the operations that affect them, and it shows that active sets will always form a boundary between positive and negative regions in the discrete sampling u, even as control of the level set passes from one set of active points to another.

Because grid points that are near the active set are kept at a fixed value difference from the active points, active points serve to control changes in the nonactive grid points to which they are adjacent. The neighborhoods of the active set are defined in *layers*, $L_{+1}, \ldots L_{+N}$ and $L_{-1}, \ldots L_{-N}$, where the i indicates the city-block distance from the nearest active grid point, and negative numbers are used for the outside layers. For notational convenience the active set is denoted L_0.

The number of layers should coincide with the size of the footprint or neighborhood used to calculate derivatives. In this way, the inside and outside grid points undergo no changes in their values that affect or distort the evolution of the zero set. The work in this paper uses first- and second-order derivatives computed on a a $3 \times 3 \times 3$ kernel (city-block distance 2 to the corners). Therefore only five layers are necessary: 2 inside layers, 2 outside layers, and the active set. These layers are denoted L_1, L_2, L_{-1}, L_{-2}, and L_0, respectively.

The active set has grid point values in the range $[-\frac{1}{2}, \frac{1}{2}]$. The values of the grid points in each neighborhood layer are kept 1 unit from the next layer closest to the active set (as in figure 3). Thus the values of layer L_i fall in the interval $[i - \frac{1}{2}, i + \frac{1}{2}]$. For $2N + 1$ layers, the values of the grid points that are not in any of the layers are either inside all of the layers, with a value of $N + \frac{1}{2}$, or outside all of the layers, with a value of $-N - \frac{1}{2}$. The procedure for updating the image and the active set based on surface movements is as follows:

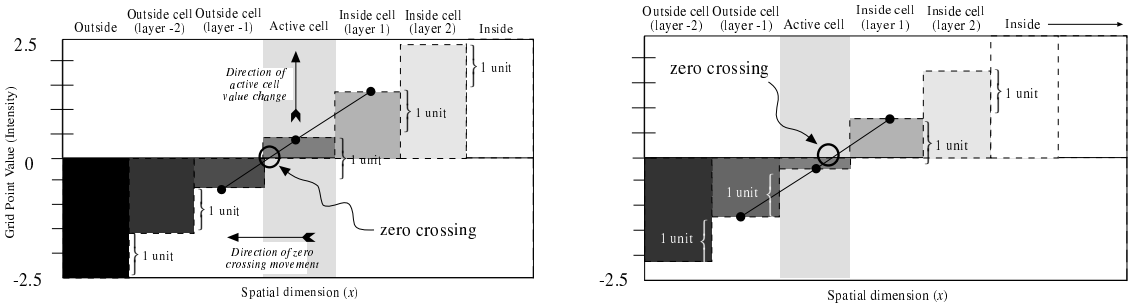


Figure 3: The status of grid points and their values at two different points in time show that as the zero crossing moves, *activity* is passed from one grid point to another.

1. For each active grid point, x_j, do the following:
   (a) Calculate the local geometry of the level set.
   (b) Compute the net change of u_{x_j}, based on the internal and external forces, using some stable (e.g., upwind) numerical scheme where necessary.
2. For each active grid point x_j add the change to the grid point value and decide if the new value $u_{x_j}(t + \Delta t)$ falls outside the $[-\frac{1}{2}, \frac{1}{2}]$ interval. If so, put x_j on lists of grid points that are changing status, called the *status list*; S_1 or S_{-1}, for $u_{x_j}(t + \Delta t) > \frac{1}{2}$ or $u_{x_j}(t + \Delta t) < -\frac{1}{2}$, respectively.
3. Visit the grid points in the layers L_i in the order $i = \pm 1, \ldots \pm N$, and update the grid point values based on the values (by adding or subtracting one unit) of the next inner layer, $L_{i \mp 1}$. If more than one $L_{i \mp 1}$ neighbor exists then use the neighbor that indicates a level curve closest to that grid point, i.e., use the maximum for the outside layers and minimum for the inside layers. If a grid point in layer L_i has no $L_{i \mp 1}$ neighbors, then it gets demoted to $L_{i \pm 1}$, the next level away from the active set.
4. For each status list $S_{\pm 1}, S_{\pm 2}, \ldots, S_{\pm N}$ do the following:
   (a) For each element x_j on the status list S_i, remove x_j from the list $L_{i \mp 1}$, and add it to the L_i list, or, in the case of $i = \pm(N + 1)$, remove it from all lists.
   (b) Add all $L_{i \mp 1}$ neighbors to the $S_{i \pm 1}$ list.

This algorithm can be implemented efficiently using linked-list data structures combined with arrays to store the values of the grid points and their states as shown in figure 4. This requires only those grid points whose values are changing, the active points and their neighbors, to be visited at each time step. Therefore computation time grows as m^{n-1}, where m is the number of grid points along one dimension of u. Computation time for dense-field approach increases as m^n. The m^{n-1} growth in computation time for the sparse-field models is consistent with conventional (parameterized) models, for which computation times increase with the resolution of the domain, rather than the range.

Another important aspect of the performance of the sparse-field algorithm is the larger time steps that are possible. In the numerical schemes for updating level-set models, the time steps are limited by the speed of the "fastest" moving level curve, i.e., the maximum of the force function. Because the sparse-field method calculates the movement of level sets over a subset of the image, time steps are bounded from below by those of the dense-field case, i.e.,

$$\sup_{x \in \mathcal{A} \subset X} (g(x)) \leq \sup_{x \in X} (g(x)), \tag{32}$$

where $g(x)$ is the space varying speed function and $\mathcal{A}$ is the active set.

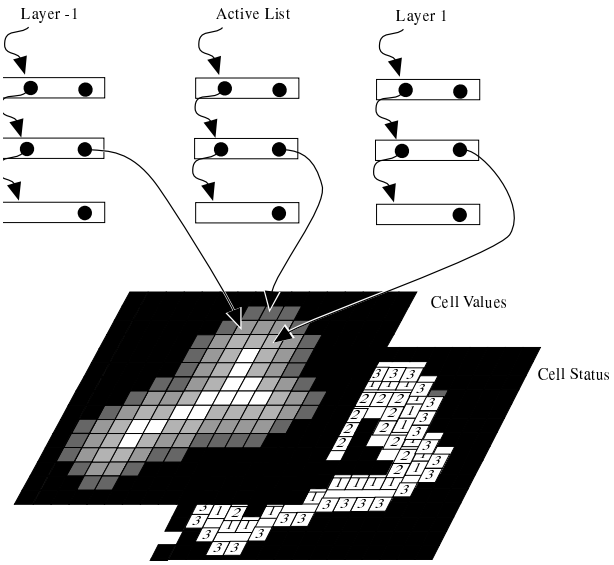


Figure 4: Linked-list data structures provide efficient access to those grid points with values and status that must be updated.

5.1 Empirical results of The Sparse-Field Algorithm

5.1.1 Error analysis

The sparse-field algorithm described above is based on an important approximation: grid points adjacent to the active points are assumed to undergo the same change in value as their nearby active-set neighbors. The dense-field algorithms treat each level set (and each grid point) separately. Because proximate level sets can have different shapes and undergo different forces, nearby grid points can undergo different changes in value. The question is how the sparse-field approximation affects the evolution of the zero-level set. The results in this section show that the evolution of the zero-level set in the sparse-field algorithm introduces an error that is consistent with that of the dense-field approach and that both errors are significantly smaller than the grid spacing h.

In order to measure the effects of these approximations we compare the movement of the level sets computed by the both the dense-field and sparse-field methods to the deformation of a circle, which can be computed analytically (Adalstein and Sethian 1995). The total error of a model is computed from the set of zero crossings along the lines connecting the grid. These points are found by using a linear interpolation between adjacent points that lie on either side of a zero crossing. The total error is the average squared distance of these zero crossings from the ideal.

Figure 5a shows the error of the dense- and sparse-field methods in 2D for a circle moving under its own curvature. Figure 5b shows the same results for the a circle moving in the direction of the inward normal at a uniform speed of 1. The 2D grid is 100×100, and the grid spacing is unity. The circle begins with a radius of 30 units. These results show that, on the whole, the sparse-field method and dense-field methods give comparable errors. The magnitude of these errors, when scaled appropriately for differences in time constants and grid spacing, are consistent with those documented in (Adalstein and Sethian 1995) even though the error metric is slightly different. Notice that in the case of constant inward velocity, both discrete level-set methods have high errors as the circle radius becomes quite small (it should be zero at $t = 30$). This is to be expected and is a inevitable consequence of using discrete methods to represent objects with sizes that are comparable to the grid spacing.

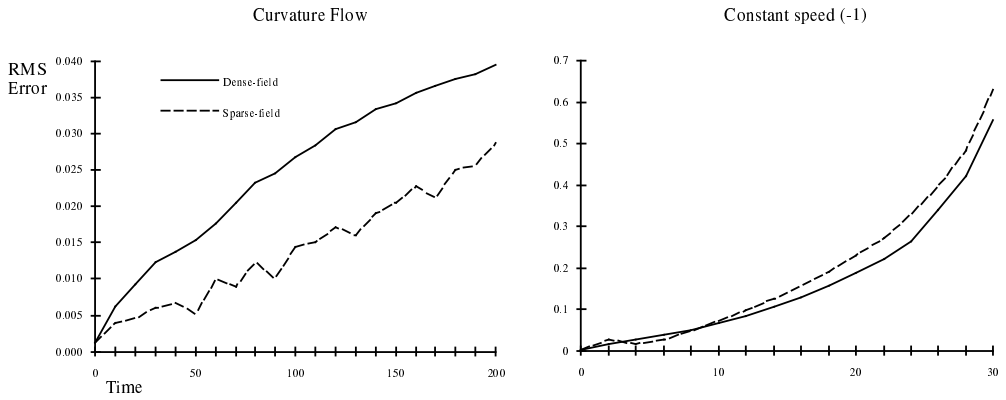


Figure 5: The rms errors, relative to ground truth, for a circle evolving under constant speed and curvature suggest that the errors associated with the sparse-field algorithm are no worse than those introduced by the discretization and level-set approximation.

5.1.2 Timing

Measurements of the timing bear out the expected improvements in performance with the sparse-field method. A rough calculation of the expected improvements is as follows. In 2D, an $n \times n$ image requires kn^2 calculations per update, where k is the number of calculations per grid point. The updates for the sparse field should be $k'n$, where k' includes the extra costs associated with updating and maintaining the neighborhood. Let $R = k/k'$, be the efficiency of the improvement. Of course, we expect R is less than one because of the extra overhead of maintaining the active set and the neighborhood around it. For a circle of radius $n/3$, the cardinality of the active set is approximately $2n$. The neighborhood is another 4 layers, each requiring 1 pass for an update. Let the cost of visiting maintaining the list and updating the neighborhood be denoted α. he Then the ratio of calculation times, R, for the two methods should be approximately

$$R = \frac{k}{k'} = \frac{k}{(4\alpha + 1)2k} = \frac{1}{8\alpha + 2}, \tag{33}$$

If α is of the same order magnitude as k (because it requires several floating point calculations involving nearest neighbors), then equation 33 gives some rough bounds on the efficiency: $0.012 \leq R \leq 0.36$.

Of course these numbers are estimates and depend quite heavily on the implementation. For the experiments of this section we have used a rather high-level, object-oriented, *C++* image processing library (developed in house), which uses in-line functions where practical as well as templated images and generic data structures. This library emphasizes ease of implementation rather than performance. The "inner loops" of the methods compared, i.e. those loops that do the calculations and updates at each pixel, are written with identical code where possible. All results are computed on a Sun Sparc 10.

Table 1 gives the average time per iteration, averaged over 25 iterations, for a circle of radius $n/3$ moving with first curvature and then a constant inward speed. The execution times and the efficiency factors confirm the expected improvements in performance from the sparse-field method.

Making direct comparisons of these computation times with the results of other researchers is difficult because of differences in implementations and hardware. However, the ratio of the dense-field computation time to that of the sparse-field algorithm gives a *performance ratio*. We have computed the same performance ratio for the narrow-band method from the data in (Adalstein and Sethian 1995). Figure 6 shows a graph of that ratio for the sparse-field approach and the narrow-band method for two distinct band widths, 6 and 12. These results show that the sparse-field method is somewhat faster, and the improvements in computation time grow as the domain size increases. For larger models, the difference in computation time between these methods is even more extreme. The models in these experiments are relatively small and the time difference should be more dramatic as one goes to 3D. For instance,

Image Width	Sparse-Field	Dense-Field	Efficiency Factor R
Circle moving under curvature.			
75	0.02	0.12	0.080
150	0.03	0.49	0.109
300	0.07	1.96	0.093
600	0.14	7.87	0.094
Circle moving at constant speed -1.			
75	0.02	0.11	0.073
150	0.05	0.44	0.059
300	0.10	1.74	0.058
600	0.20	6.97	0.058

Table 1: A comparison of execution times (seconds/iteration) for computing the evolution a circle

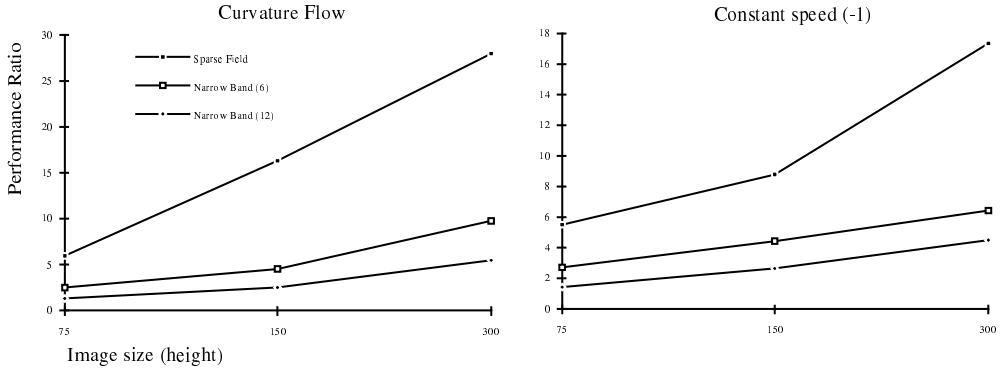


Figure 6: A comparison of the sparse-field method with the narrow-band results from (Adalstein and Sethian 1995) bears out the advantage of the sparse field method. The advantage is modest for small models but grows as the models get larger — an important trend when considering large, volumetric models.

(a) (b) (c)

Figure 7: The deformation of a cube moves downhill on the distance transform of a torus. In (a) the steady state of the dense-field approach shows that the torus suffers from aliasing due to the shocks which form near the boundary of the torus. In (b) the sparse-field algorithm bounces back and forth between grid points that have forces in opposite directions. In (c) the modified sparse-field algorithm uses a first-order approximation to position the zero-level set to subcell accuracy.

the circle in the 300×300 image contains approximately 600 pixels along its border. The surface of a sphere in a $300 \times 300 \times 300$ volume would contain about 40,000 voxels.

5.2 Subcell accuracy

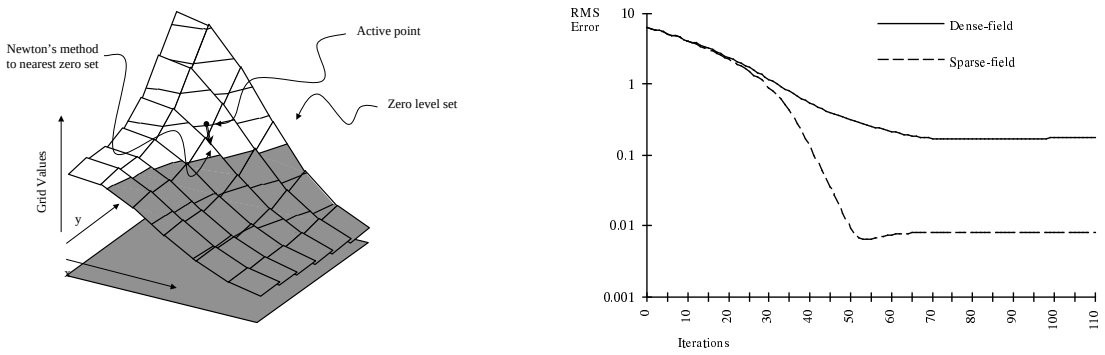
Results of the previous section demonstrate the accuracy and computational efficiency of the sparse-field algorithm. However, the level-set approach is still not appropriate for 3D reconstruction because of its limited spatial resolution. This section shows how the sparse-field algorithm can be modified to improve the overall accuracy of the level-set method.

It is well known that front propagations of the form of equation (25), without any smoothing term, form shocks where fronts moving in opposite directions meet. In the discrete domain, when using an upwind scheme, these shocks take the form of high contrast regions that form along those grid points that lie near the zero crossings of $g(\cdot)$. Unfortunately these shocks have an adverse side effect; they constrain the positions of level-set solutions to fall on the grid lines, half way between grid points. The high contrast regions associated with shock formation cause *aliasing* in the final results of level-set models.

This aliasing is not an inherent property of the implicit representation. Indeed, grid-point values can be manipulated to position zero crossings anywhere on the grid lines, and linear or higher order interpolation techniques can be used to construct parametric representations from level sets to within subcell accuracy. The aliasing associated with the moving wavefronts follows from the fact that the numerical schemes for propagating fronts sample the force function $g(\cdot)$ only at grid point locations; it is an inherent problem in the level-set numerical schemes that have been proposed to date. Any iterative deformation scheme that samples the forcing function at only a discrete set of grid locations will be limited in its ability to accurately locate the solution.

The problems associated with the discrete sampling of the forcing function are particularly troublesome when considering 3D reconstruction. Figure 7(a) shows the result of allowing a cube (sampled on a $50 \times 50 \times 50$ grid) to move on the distance transform of a torus. The distance transform of the torus (which serves as the $g(\mathbf{x})$ for the reconstruction) is computed analytically and then sampled on the same grid as the cube. The level-set model forms patterns that reflect the underlying grid structure because of the shocks that form between grid points that are inside and outside of the torus boundary. The problems of shock creation and aliasing are even more serious in cases where the forces, represented by $g(\cdot)$, have a greater resolution than the model — as in the case of recovering 3D models from high-resolution range maps. In such cases limiting the model position to the resolution of the grid is far from optimal; it introduces unnecessary artifacts.

The sparse-field algorithm provides a mechanism for positioning level sets to subcell accuracy. In the sparse-field algorithm the active grid points can be thought of as control points for a nearby zero-level set. The level set does not necessarily pass through the center of the grid point (except in the special case where the grid point has a value zero).



(a) (b)

Figure 8: The modified sparse-field algorithm: (a) For a curve in a plane, the position of the level set near to a grid point is found using Newton's method. (b) For a square being fit to a circle, the sparse-field method obtains subcell accuracy and almost perfect fit (less than a hundredth of a voxel error), while the dense-field scheme forms shocks and stabilizes at an error of 0.20 cells.

Newton's method gives a first-order approximation to the position of the nearby zero-level set, as in figure 8(a).

Let $\mathbf{x} \in \mathrm{I}^D$ be the position of a grid point, $\nabla u_{\mathbf{x}}$ be the vector of first derivatives of the model at some time t. The position of the closest zero-level set to the grid point $\hat{\mathbf{x}}$ is given by

$$\hat{\mathbf{x}} = \mathbf{x} - u_{\mathbf{x}} \frac{\nabla u_{\mathbf{x}}}{\nabla u_{\mathbf{x}} \cdot \nabla u_{\mathbf{x}}}, \tag{34}$$

except when $\nabla u_{\mathbf{x}} = 0$, which must be handled as a special case. Computing the forces on level-set locations away from the grid points is similar in philosophy to the method of *extending* the velocity fields of level sets described by Sethian (1996).

The numerical computation of $\nabla u_{\mathbf{x}}$ in equation (34) can proceed in one of several different ways. Although centralized differences are possible, the closest zero crossing can be found by finding the *steepest* one-sided derivative in each dimension. Define the function

$$\mathrm{MaxAbs}(a,b) \triangleq \begin{cases} a & |a| > |b| \\ b & |a| < |b| \\ \frac{a+b}{2} & |a| = |b| \end{cases} . \tag{35}$$

The direction of nearest zero crossing can be calculating by using

$$\nabla u_{\mathbf{x}} \triangleq \mathrm{MaxAbs}(\mathrm{d}_{x_1}^{(+)} u_{\mathbf{x}}, \mathrm{d}_{x_1}^{(+)} u_{\mathbf{x}}), \ldots, \mathrm{MaxAbs}(\mathrm{d}_{x_D}^{(+)} u_{\mathbf{x}}, \mathrm{d}_{x_D}^{(+)} u_{\mathbf{x}}). \tag{36}$$

Figure 7(c) shows that better results are obtained from the modified sparse-field method (first-order approximation to the level set location) than methods which sample force-field values only at grid point locations. The first-order approximation allows grid points to achieve grey-level values that reflect their distance from nearby features. This first-order improvement is essential in using the level-set paradigm for modeling 3D objects. The rendering of 3D models makes use of first-order derivatives of the volume data, which are sensitive to aliasing artifacts.

Figure 8(b) shows the error, using the same error measure described in section 5.1.1, for a square moving downhill on the distance transform of a circle. The error begins at about 7 units and slowly decreases as the model moves toward the circle. The difference in the sparse-field and dense-field methods demonstrates the subcell accuracy that is obtained with the first-order modifications. The dense-field method forms shocks, and the error stabilizes at about 0.20 cells. The dense-field scheme produces a binary image which is the inside-outside function for the circle. This result is representative of other numerical algorithms for level sets, such as that of (Adalstein and Sethian 1995),

which do not attempt to position level sets to sub-cell accuracy. The sparse-field method positions the zero-level set to subcell accuracy and eventually obtains very small errors in fitting the level set. The small dip in error at about 50 iterations is even more dramatic in the dense-field approach but is not easily visible on the logarithmic scale given in figure 8(b). This overshoot indicates that the model moves through the solution and slightly beyond before reaching a steady state.

This strategy of approximating the level-set position to sub-cell accuracy can generalized to higher orders. For instance, in two dimensions one can fit a second-order function to u in the neighborhood of the active grid point. Such higher order schemes are not pursued in this paper for two reasons. First, they would add considerably to the computational burden of the method. Second, the embedding of the level set is the distance transform implies that $|\phi| = 1$ almost everywhere, and therefore second derivative in the gradient direction is virtually 0 except at singularities (Bruce and Giblin 1986).

6 Level-Set Models for 3D Reconstruction

This section combines the level-set modeling technology from Sections 4 and 5 with the MAP reconstruction formulation of Section 3 to generate 3D reconstructions of objects from multiple range maps. The strategy is as follows. Construct a rather coarse volume that is the solution to the linear problem, i.e. the zero-level sets of $g(\cdot)$, without the prior. This volume serves as initialization for a level-set model which moves toward the data given by the range maps while undergoing a second-order flow to enforce the prior. After the rate of deformation slows to below some threshold, the resolution is increased, the volume resampled, and the process repeated. The coarse-to-fine strategy is intended to be a continuation method (as described by (Nielson 1997, Snyder, Han, Bilbro, Whitaker and Pizer 1995)) for both reducing computation and preventing the algorithm from converging on local minima.

There are several additional considerations which affect the performance of this algorithm. The first consideration is that the MAP formulation in Section 3 suffers from a problem; it ignores the nonlinearity resulting from the fact that the range scanner gives the depth reading for the *single closest surface point* along the line of sight. Therefore the solution given by the zero sets of $g(\cdot)$ could contain artifacts that result from surfaces interacting at occlusion boundaries. The use of $r_{\max}$ or the windowing function $\omega(\cdot)$ help alleviate this problem, but it is virtually impossible to stop this interaction in places where the object has high curvature near its occluding contour. An iterative scheme can eliminate interactions of surfaces near occluding contours by taking advantage of the knowledge that the objects are solids. That is, a surface cannot return a range reading if it is facing away from the scanner. If the surface normal is taken to be outward, then the dot product of the surface normal and the line of sight must be negative. The evolution equation, modified to include only those surface that face the scanner associated with a particular range map, is

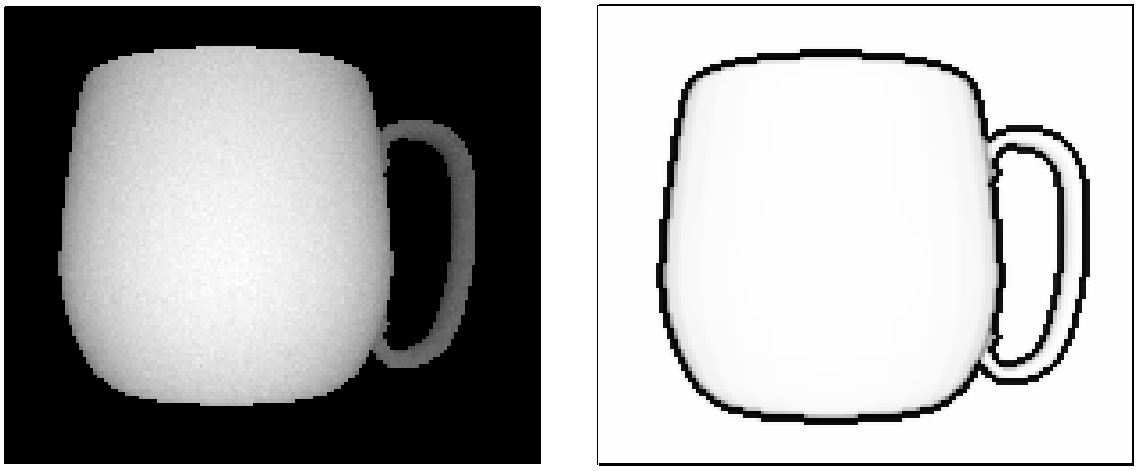
$$\frac{\partial \phi(\mathbf{x},t)}{\partial t} = |\nabla \phi(\mathbf{x})| \sum_j D^{(j)}(\hat{\mathbf{x}})\, \omega\left(D^{(i)}(\hat{\mathbf{x}})\right)\, \gamma^{(j)}(\hat{\mathbf{x}})\, c^{(j)}(\hat{\mathbf{x}}) \frac{\left(\nabla\phi \cdot \mathbf{n}^{(j)}(\hat{\mathbf{x}})\right)^+}{\nabla\phi \cdot \mathbf{n}^{(j)}(\hat{\mathbf{x}})} + \tag{37}$$

$$\beta\left(\phi_x^2+\phi_y^2+\phi_z^2\right)^{-\frac{1}{2}}\left[\left(\phi_y^2+\phi_z^2\right)\phi_{xx}+\left(\phi_z^2+\phi_x^2\right)\phi_{yy}\right. \tag{38}$$
$$\left.+\left(\phi_x^2+\phi_y^2\right)\phi_{zz}-2\left(\phi_x\phi_y\phi_{xy}+\phi_y\phi_z\phi_{yz}+\phi_z\phi_x\phi_{zx}\right)\right],$$

where $\mathbf{n}^{(j)}(\mathbf{x})$ is the line of sight from a range finder to a 3D point, $\mathbf{x}$.

A second practical consideration is the confidence associated with the range data. In Section 3 the confidence measure $c^{(j)}(\mathbf{x})$ was expressed as the inverse of the variance of the sensor noise associated with the range measurement along the line of sight $\mathbf{n}(\hat{\mathbf{x}})$. Other factors also affect this confidence metric. One such factor is the uncertainty in the position of the model that results from the discretization of the embedding ϕ. More specifically, each grid point being updated in the volume u, controls the movement a of a level surface nearby, whose position is known to only finite accuracy. Thus there is cloud of uncertainty around each grid point. The first-order approximation to the level-set position described in Section 5.2 improves matters considerably, but the 3D positional error cannot be discounted entirely. For this work we assume that uncertainty is isotropic with variance denoted $\sigma_{\mathbf{x}}^2$.

The positional error associated with the discrete sampling of a volume gives rise to an uncertainty about the line of



(a) (b)

Figure 9: The range map of a mug, viewed as an image (a), gives rise to a confidence measure (b) that combines both the noise of the scanner with the spatial uncertainty of the model.

sight and thereby uncertainty about the value of the associated range measurement. Using a first-order approximation to calculate this error we have

$$c^{(j)}(\mathbf{x}) = \frac{1}{\left(\sigma^{(j)}(\mathbf{x})\right)^2 + \left|\nabla D^{(j)}(\mathbf{x})\right|^2 \sigma_{\mathbf{x}}^2(\mathbf{x})}, \tag{39}$$

where $\sigma^{(j)}(\mathbf{x})^2$ is the variance associated with the range measurement along the line of sight that passes through $\mathbf{x}$ and $D^{(j)}(\mathbf{x})$, as in equation (13), is the signed distance along the line of sight between the range measurement and $\mathbf{x} \in \mathbb{R}^3$. Derivatives of $D^{(j)}(\mathbf{x})$ combine the geometry of the scanner with derivatives of the range map. That is,

$$\nabla D^{(j)}(\mathbf{x}) = \nabla \mathbf{x} - \nabla r^{(j)}(\mathbf{x}) = \nabla \mathbf{x} - \frac{\partial r^{(j)}(\theta,\varphi)}{\partial \theta}\nabla\theta(\mathbf{x}) - \frac{\partial r^{(j)}(\theta,\varphi)}{\partial \varphi}\nabla\varphi(\mathbf{x}). \tag{40}$$

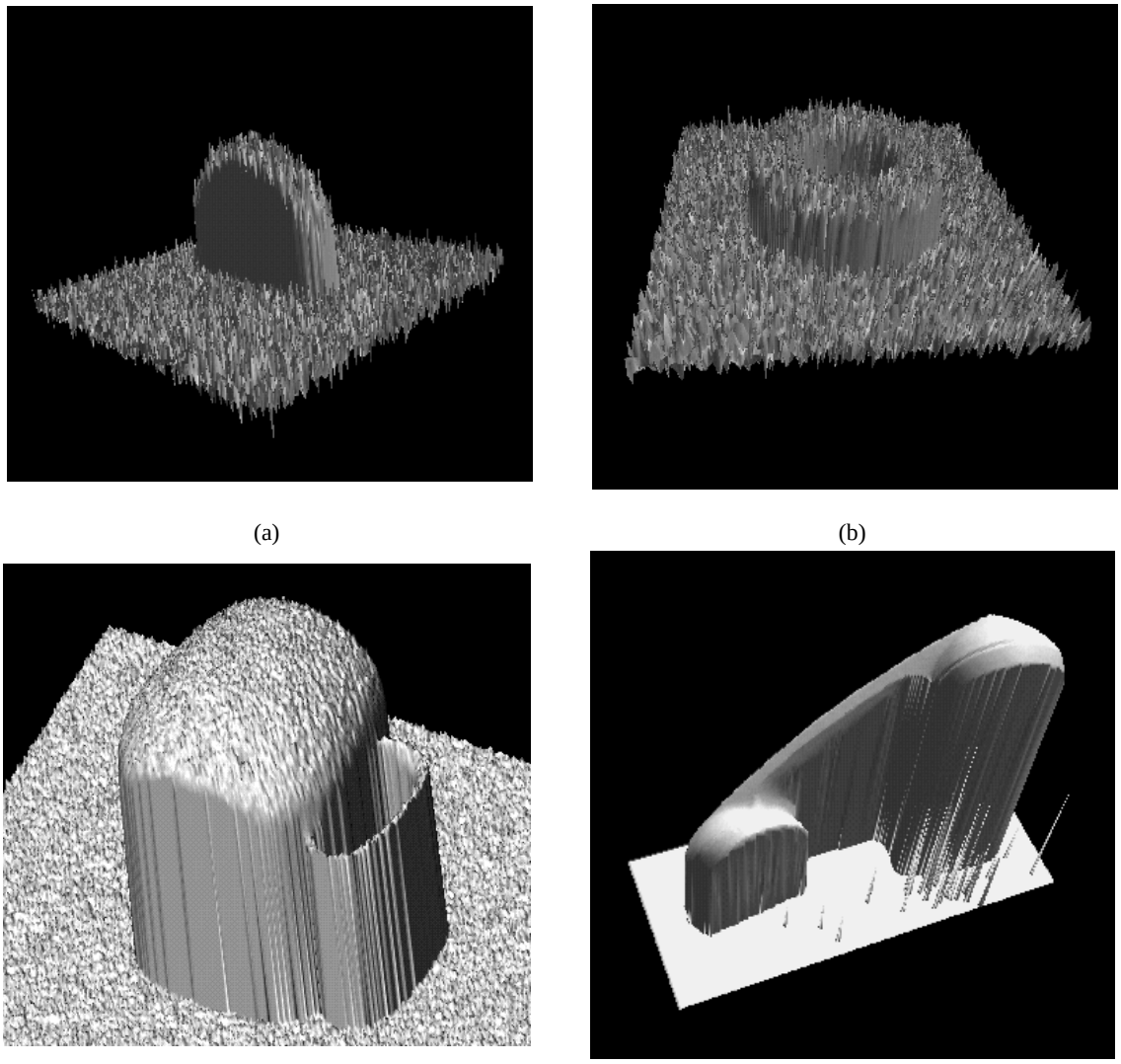
Thus, this formulation has the intuitive result of lowering the confidence near step edges in the range map. Some simple reasoning gives approximate values for $\sigma_{\mathbf{x}}$. If the surface position is known to within one half of a voxel (and voxels have unit length), then one should choose $\sigma_{\mathbf{x}}^2 = 1/12$. If the first-order approximation reduces that by an order of magnitude we have $\sigma_{\mathbf{x}}^2 = 1/120$. Figure 9a shows a noisy range map generated from the CAD model of a mug, and figure 9b shows the resulting confidence map with $\sigma_{\mathbf{x}}^2 = 1/120$ and a constant noise value for all range measurements of 1 unit.

6.1 Results

This section presents 3D reconstructions from several different sets of data. The first is synthetic data of a torus, shown as a surface in Figures 10(a) and 10(b). These 200×200 pixel depth maps are computed analytically and corrupted with 20% uncorrelated Gaussian noise. Six such views of the torus are combined in the examples that follow.

The second set of data is ten synthetic views computed from a CAD model of a mug as shown in figure 10(c). This mug has some smaller features such as the handle (particularly where it attaches to the body) and the top of the rim. Results will be shown with and without 50% additive Gaussian noise.

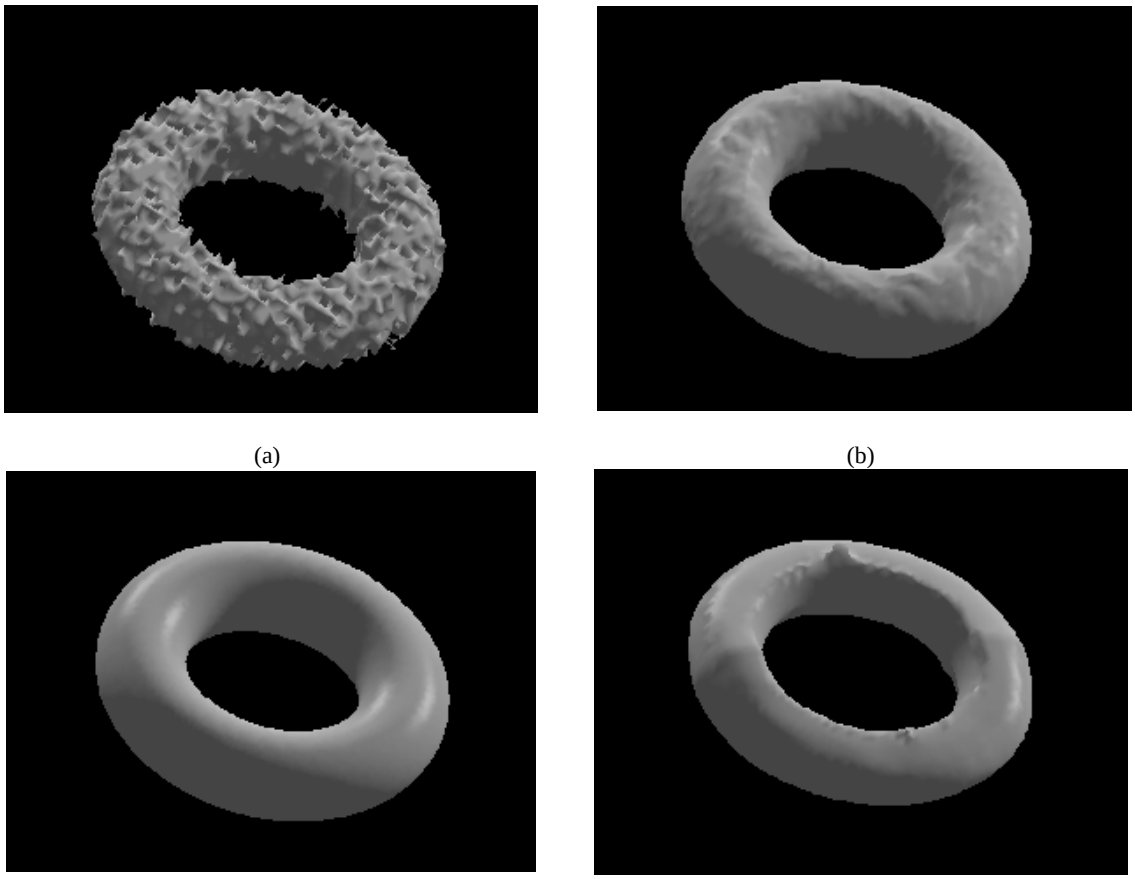
For real data, figure 10(d) shows a 400×500 pixel depth map taken from a telephone receiver using a triangulating laser range finder. Eight such views have been positioned relative to each other, initially by hand and then by an automated surface matching technique (Turk and Levoy 1994).



(a) (b)

(c) (d)

Figure 10: Range maps: Synthetic range data of a torus — 200×200 pixels with 20% Gaussian white noise (as a fraction of smaller diameter) taken of both end (a) and side (b). Synthetic range data of a mug (c) — 256×256 pixels with 50% Gaussian white noise (as a fraction of handle width). Triangulating laser range data of telephone (d).



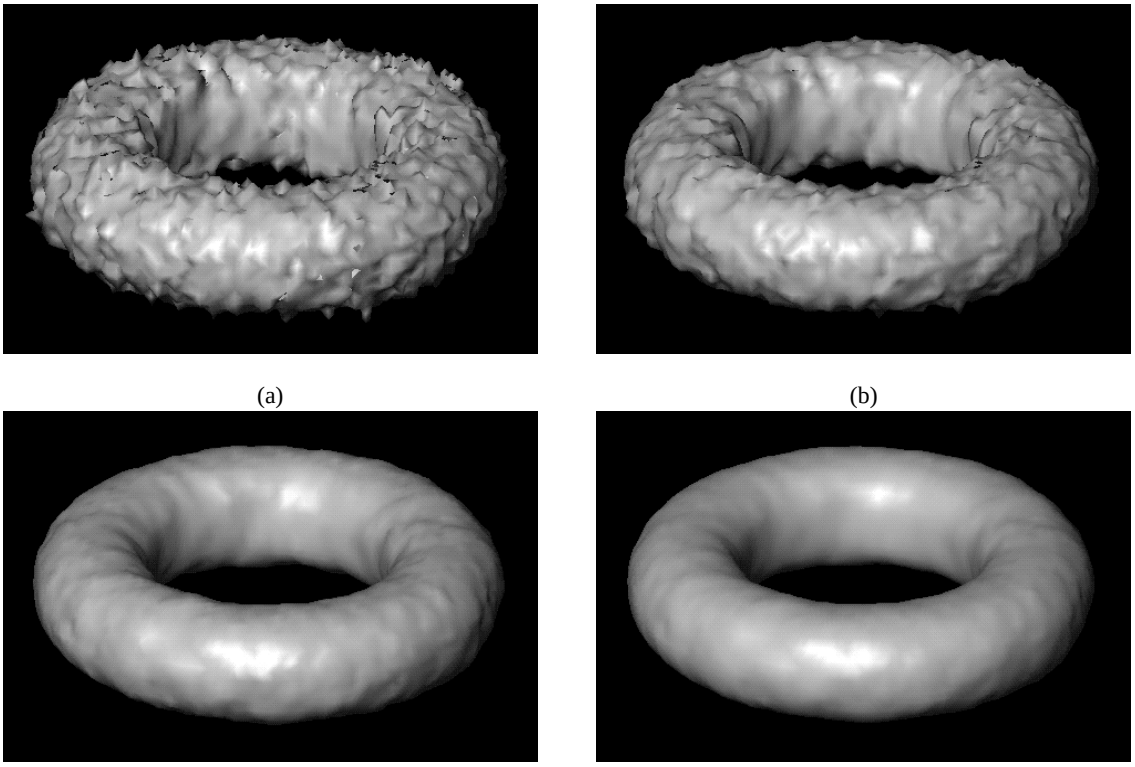
(a) (b)

(c) (d)

Figure 11: An initial model (a) is constructed by combining six points of view of a torus using the zero crossings of $g(\cdot)$. The model which is attracted to the range data but undergoes internal forces evolves and settles into a smoother steady state (b) which resembles the uncorrupted torus (c). Image-based smoothing produces view-dependent artifacts (d).

Figure 11(a) shows the initial model used for fitting the level-set model to the range data for the torus. The initial model is a $80\times80\times40$ voxels and is produced by finding the zero crossings of the data term, $g(\cdot)$. For all of the examples of this paper, 3D surface renderings of level sets are from polygonalized surfaces obtained from the volumes by the marching cubes method (Lorenson and Cline 1982). Figure 11(b) shows the result of the level-set model that uses 11(a) as an initial state and β value of 0.25. The resulting model, which combines the six points of view and the smoothing function, is a reasonable reconstruction of the original object (figure 11(c)). Figure 11(d) shows the result of the combined data term $g(\cdot)$ created by the six noisy torus range maps that have been smoothed with Gaussian blurring prior to their combination. Such an image-space blurring distorts the geometric structure of the range maps so that they no longer fit together; this results in view-dependent artifacts. The object-based smoothing, achieved by incorporating the prior of equation (19) into the level-set approach, does not produce such artifacts.

Figure 12 shows the effects of the parameter β on the final result. The parameter is not a threshold that must be set precisely; results change gradually as β varies over an order of magnitude. Figure 13 shows, for different values of β, the rms error of the reconstruction of the torus as a percentage of the magnitude of the noise added to the synthetic range maps. Even without the prior, the averaging obtained from the MAP method overcomes much of the initial noise in the data. The quality of the reconstruction does depend on the choice of β, but the use of the prior



(a) (b)

(c) (d)

Figure 12: Solutions to reconstruction described in figure 11 with values of β at (a) 0, (b) 0.04, (c) 0.32, and (d) 0.64.

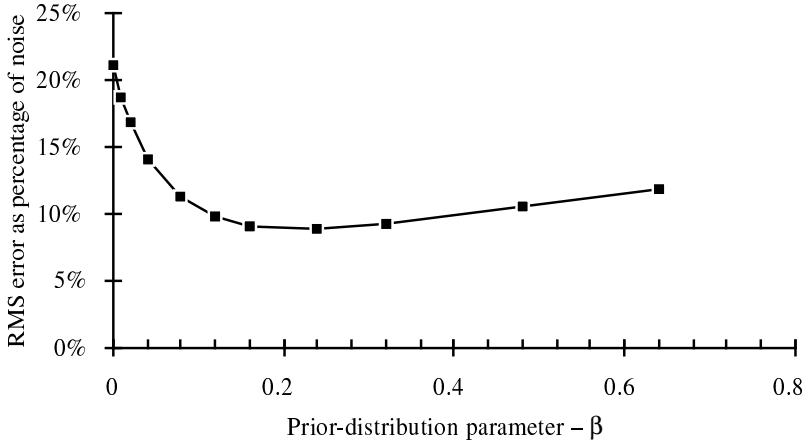


Figure 13: The rms error of the reconstruction of a torus, given as a percentage of the magnitude of the noise, for different values of the prior, β.

provides significant improvements in the reconstruction for any choice of beta that is within an order of magnitude of the optimum.

A 3D model of a mug, shown in figure 14(a), is used to generate 10 range maps. These ten views are then combined to reconstruct the 3D model (figure 14(b)). The reconstruction resembles the original in figure 14(c) to within the accuracy afforded by the discretization of the volume and some small shadowing artifacts around the handle, where it is difficult to completely map because of self occlusions. The final model is $140 \times 140 \times 140$ voxels. At this resolution the handle is less than two voxels thick, but one can still recover the fine details where the handle attaches to the main part of the mug. The facets are in the original, polygonal model. Such large volumes (almost 3 million voxels) necessitate the use of the sparse-field approach, which visits the entire data set only during the initialization, and visits only those voxels that lie near the surface thereafter. With no priors (i.e. $\beta = 0$) and range images that are corrupted by additive Gaussian noise, the surface takes on an rough appearance. Figure 14(d) shows how a small influence of the prior, $\beta = 0.5$, produces a smoother result. The result that includes the smoothing prior is also quantitatively better as determined by the rms error from the original mug model. The $140 \times 140 \times 140$ grid used for the mug required 16 iterations. The entire reconstruction process lasted about 20 minutes on a Sparc 10. Most of that time was spent on the initialization and resampling which requires visiting the entire volume.

For the workstation used in this work, this large model is at the limit of what can be stored in random access memory. As a result, models larger than this typically introduce thrashing and significantly longer computation times. Methods for efficiently representing sparse volumes are well documented in the literature, but access time penalties associated with current technologies (e.g. oct-trees) would make such methods inefficient for the iterative procedures used in this work. Reducing these memory requirements while maintaining run-time efficiency is an area of ongoing investigation.

Figure 15 shows how the algorithm is able to handle outliers. Figure 15(a) shows a single scan from the same mug data set corrupted with 1% replacement noise (i.e. outliers) instead of additive Gaussian noise. The resulting initialization shown in 15(b) is quite poor; it contains hundreds of holes. The combination of smoothing and sub-voxel accuracy obtained from the level-set models pulls the model away from the outliers and gives a convincing reconstruction.

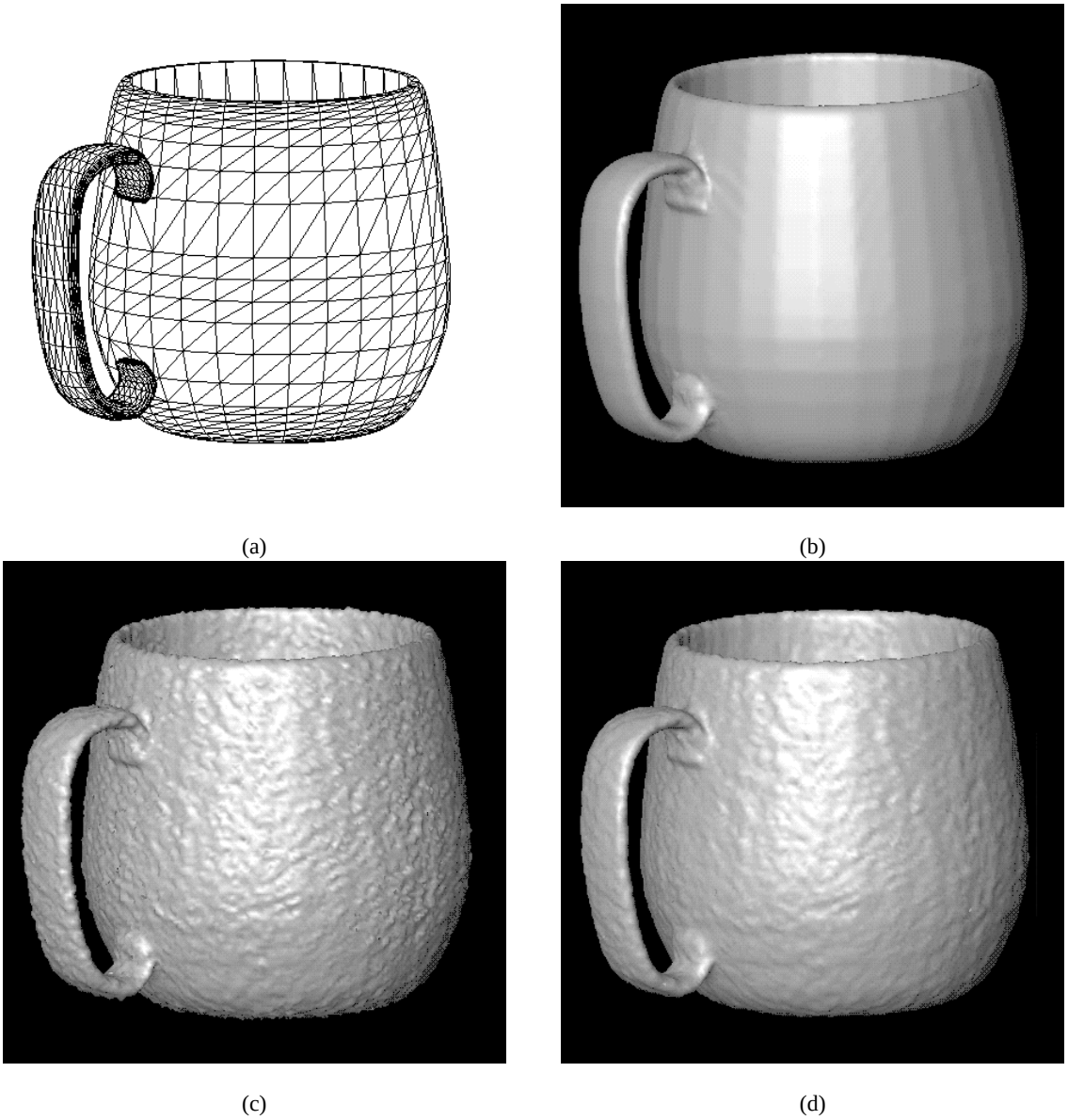
In figure 16 eight range maps of a telephone (taken with a triangulating laser range finder (Curless and Levoy 1995)) are combined to create a 3D reconstruction. Figure 16(a) shows the $40 \times 20 \times 20$ initial model used for fitting level-set models to the range data. That model serves as the initial conditions for the evolution given by equation (39). After the model settles down (change from one iteration to the next drops below a threshold), the volume is resampled onto a finer grid, and the process is repeated. The scanner images have artifacts which affect the result of the modeling as shown in figure 17(a). These artifacts result from false surfaces in the data, as well as misalignments in the range maps. The MAP reconstruction algorithm as it stands cannot realign the data, but to the extent these artifacts have a high-frequency character to them, they can be controlled by adjusting the smoothing parameter, β.

7 Conclusions

This paper has presented a strategy for reconstructing 3D objects by combining range maps taken from different points of view. The strategy is to compute the surface which is mostly likely to have given rise to the data generated by the range scanner; it maximizes the posterior probability of the surface.

By using the independence of the sensor noise and assuming that the collection of data in any one map is dense relative to the object structure, the MAP formulation converts the error from individual range readings into a volume integral. The Euler-Lagrange of that volume integral gives the equation of motion for a surface that seeks to minimize its likelihood conditional on the data. The introduction of a prior leads to a full MAP formulation. This is a fundamental result in 3D reconstruction which poses surface reconstruction as an evolutionary process based on first principles.

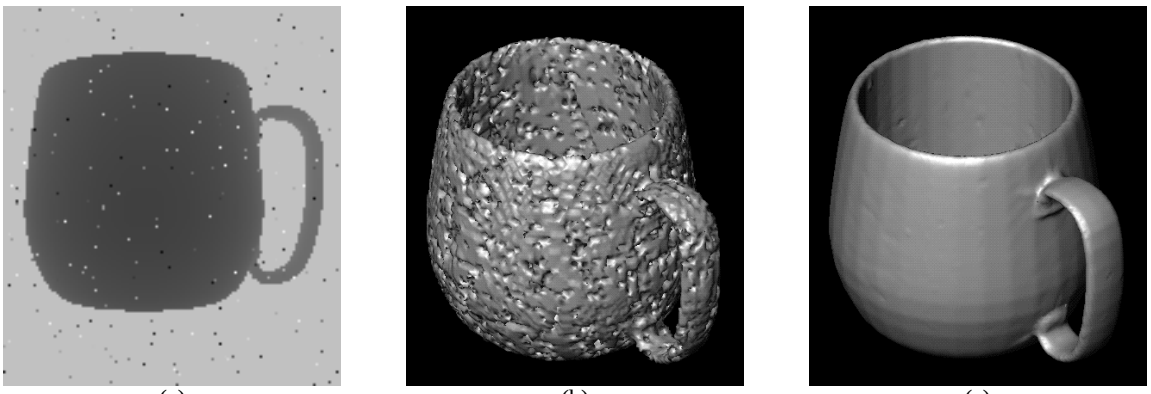
This paper has also proposed level-set models for computing the deformations associated with MAP estimation. The level-set modeling approach is well suited for the MAP reconstruction formulation because it offers flexible



(a) (b)

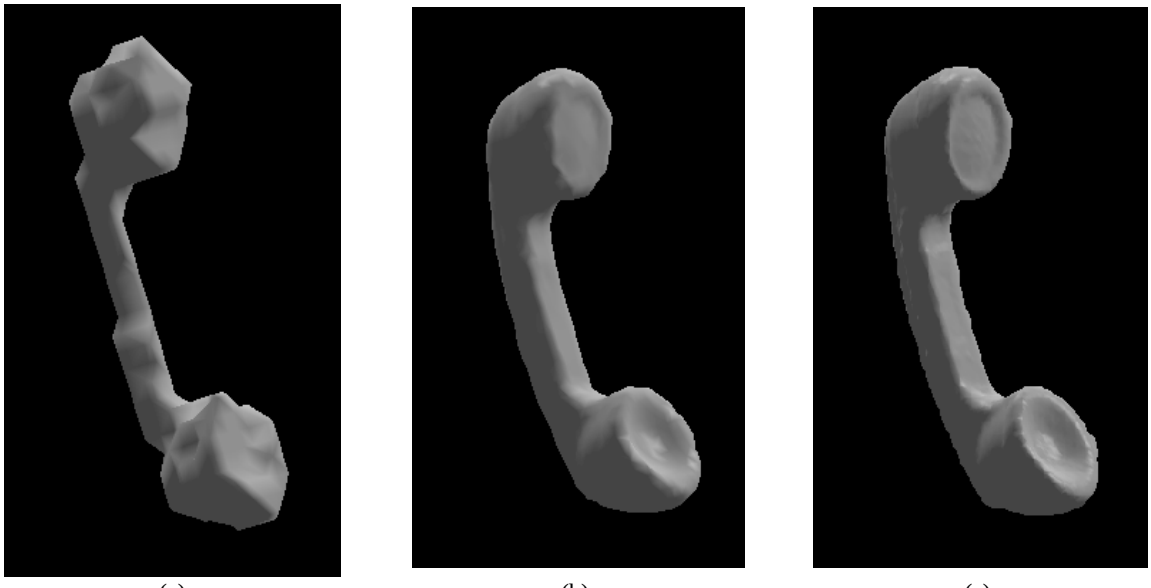
(c) (d)

Figure 14: Level-set MAP reconstruction of a mug using synthetic data generated from a 3D model (a). Without noise the reconstruction (b) is limited only by the resolution of the model ($140 \times 140 \times 140$). With noise, the surface appears rough (c). Including a prior of $\beta = 0.25$ improves the appearance of the reconstruction (d) and the rms error from the original.



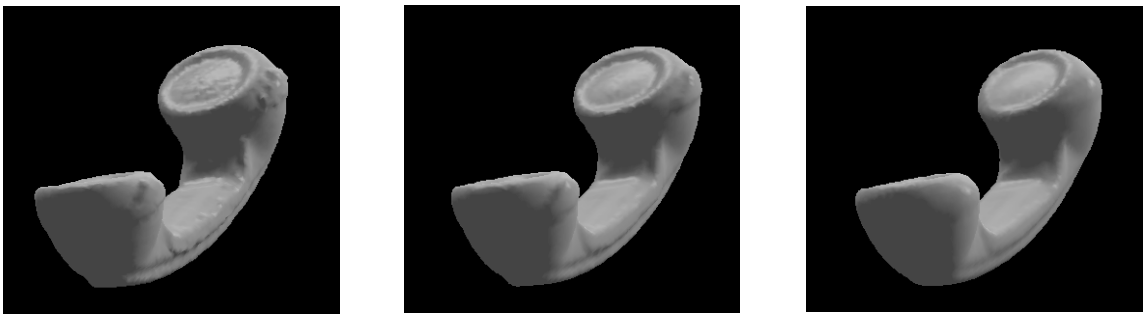
(a) (b) (c)

Figure 15: The effects of outliers: (a) The original mug data set corrupted by replacement noise. (b) The poor initialization that results from that data. (c) The results of the iterative, level-set formulation.



(a) (b) (c)

Figure 16: The results of the level-set MAP reconstruction approach: An initial model (a) is constructed by combining eight points of view of a telephone into a discrete occupancy grid of $40 \times 20 \times 20$ grid points. Results from the course model serve as initial conditions for successively finer models in (b) and (c).



(a) (b) (c)

Figure 17: Artifacts that produce discontinuities can be controlled by tuning the weight β of the smoothing term. In (a), (b), and (c), steady-state solutions with β = 0.1, 1.0, and 5.0, respectively.

topologies, multiscale/multigrid representations, many degrees of freedom, and efficient methods for calculating geometric surface properties. Despite these advantages, the level-set approach as described in the literature suffers from high computational costs and limited accuracy.

This paper has proposed an enhancement to level-set modeling technology in the form of a new algorithm, a sparse-field method. The sparse-field method performs calculations and updates only at those grid point locations that lie near the zero-level set (or any level set of interest). As a result this algorithm is computable in a fraction of the time required for other level-set approaches. This paper has shown that this new method introduces errors that are no worse than previous algorithms. The sparse-field algorithm also enables one to use a first-order approximation to the level-set position, and therefore it is actually more accurate under circumstances where the forcing function is defined to greater accuracy than the grid structure, as is the case with the proposed MAP formulation for 3D reconstruction.

Results on both real and synthetic data confirm the effectiveness of combining the MAP strategy with the level-set modeling method. The use of a very simple prior, which biases solutions toward those that have less surface area, can improve the reconstruction in the presence of uncorrelated noise and high-frequency artifacts.

There are, however, several areas that warrant further development. One area is the problem of large data sets. While the sparse-field method reduces the number of computations required for deforming, it does not alleviate the need to store data for the volume that covers the entire domain. Another area of ongoing investigation is the prior. The second-order smoothing used in this paper tends to distort the shapes of objects when applied too aggressively. This is particularly true when the objects have high curvature, such as the handle of the mug in figure 14. Priors, such as the bending energy, that rely on higher order geometry or priors that incorporate higher level knowledge about the objects could improve the quality of the results.

Acknowledgments

Thanks go to David Elsner for supplying the “mug” data set. The laser range data for the telephone was provided by the Stanford University Computer Graphics Laboratory. This work is supported by the Office of Naval Research grant N00014-97-0227 and the University Research Program in Robotics under the Department of Energy DOE-DE-FG02-86NE37968.

References

Adalstein, D. and Sethian, J. A.: 1995, A Fast Level Set Method for Propogating Interfaces, *Journal of Computational Physics* pp. 269–277.

Alvarez, L., Guichard, F., Lions, P.-L. and Morel, J.-M.: 1992, Axioms and fundamental equations of image processing, *Technical Report 9231*, Ceremade, Universite Paris-Dauphine, Place de Lattre de Tassigny, 75775 Paris Cedex 16 France.

Besl, P. J. and McKay, N. D.: 1992, A method for registration of 3-D shapes, *IEEE Trans. Pattern Anal. Mach. Intelligence* **14**, 239–256.

Bruce, J. and Giblin, P.: 1986, Growth, motion, and 1-parameter families of symmetry sets, *Proc. Roy. Soc. Edinburgh* **104A**, 179–204.

Caselles, V., Kimmel, R. and Sapiro, G.: 1995, Geodesic active contours, *in* ICC (1995), pp. 694–699.

Chen, Y. and Medioni, G.: 1992, Object modeling by registration of multiple range images, *Int. J. Image and Vision Computing* **10**, 145–155.

Chen, Y. and Médioni, G.: 1994, Fitting a surface to 3-d points using an inflating ballon model, *in* A. Kak and K. Ikeuchi (eds), *Second CAD-Based Vision Workshop*, Vol. 13, IEEE, pp. 266–273.

Chien, C.-H. and Aggarwal, J. K.: 1989, Model construction and shape recognition from occluding contours, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**(4), 372–390.

Curless, B. and Levoy, M.: 1995, Better optical triangulation through spacetime analysis, *in* ICC (1995), pp. 987–994.

Curless, B. and Levoy, M.: 1996, A volumetric method for building complex models from range images, *Computer Graphics (SIGGRAPH '96 Proceedings)* .

DeCarlo, D. and Metaxas, D.: 1995, Adaptive shape evolution using blending, *in* ICC (1995), pp. 834–839.

Dickinson, S., Metaxas, D. and Pentland, A.: 1997, The role of model-based segmentation in the recovery of volumetric parts from range data, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(3), 259–267.

Elfes, A.: 1989, Using Occupancy Grids for Mobile Robot Perception and Navigation, *Computer* **22**(6), 46–57.

Hilton, A., Stoddart, A. J., Illingworth, J. and Windeatt, T.: 1996, Reliable surface reconstruction from multiple range images, *Proceedings from the Sixth European Conference on Compter Vision*, Springer-Verlag.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J. and Stuetzle, W.: 1992, Surface reconstruction from unorganized points, *Computer Graphics* **26**(2), 71–78.

ICC: 1995, *Fifth International Conference on Computer Vision*, IEEE Computer Society Press.

Jain, A. and Flynn, P. (eds): 1993, *Three-Dimensional Object Recognition Systems*, Elsevier Science Publishers, The Netherlands.

Jain, R. and Jain, A. (eds): 1990, *Analysis and Interpretation of Range Images*, Springer-Verlag.

Johnson, V. E.: 1993, A framework for incorporating structural prior information into the estimation of medical images, *in* H. H. Barrett and A. F. Gmitro (eds), *Information Processing in Medical Imaging (IPMI'93)*, number 687 in *Lecture Notes in Computer Science*, Springer-Verlag, pp. 307–321.

Kass, M., Witkin, A. and Terzopoulos, D.: 1987, Snakes: Active contour models, *International Journal of Computer Vision* **1**, 321–323.

Kimia, B. J., Tannenbaum, A. R. and Zucker, S. W.: 1992, Shapes, shocks, and deformations I: The components of shape and the reaction-diffusion space, *Technical Report LEMS-105*, Division of Engineering, Brown University.

Lorenson, W. and Cline, H.: 1982, Marching cubes: A high resolution 3d surface construction algorithm, *Computer Graphics* **21**(4), 163–169.

MacDonald, D., Avis, D. and Evans, A.: 1994, Volumetric deformable models: Active blobs, *in* R. A. Robb (ed.), *Visualization In Biomedical Computing 1994*, SPIE—The International Society for Optical Engineering, Mayo Clinic, Rochester, Minnesota, pp. 160–169.

Malladi, R., Sethian, J. A. and Vemuri, B. C.: 1995, Shape modeling with front propogation: A level set approach, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(2), 158–175.

McInerny, T. and Terzopoulos, D.: 1995, Toplogically adaptable snakes, *in* ICC (1995), pp. 840–845.

Miller, J., Breen, D., Lorensen, W., O'Bara, R. and Wozny, M.: 1991, Geometrically deformed models: A method for extracting closed geometric models from volume data, *Computer Graphics (SIGGRAPH '91 Proceedings)* **25**(4), 217–226.

Moravec, H. P.: 1988, Sensor Fusion in Certainty Grids for Mobile Robots, *AI Magazine* **9**(2), 61–77.

Muraki, S.: 1991, Volumetric shape description of range data using "blobby model", *Computer Graphics (SIGGRAPH '91 Proceedings)* **25**(4), 227–235.

Nielson, M.: 1997, Graduated nonconvexity by functional focusing, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(5), 521–525.

Osher, S. and Sethian, J.: 1988, Fronts propogating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *Journal of Computational Physics* **79**, 12–49.

Sethian, J. A.: 1996, *Level Set Methods: Evolving Interfaces in Gometry, Fluid Mechanics, Computer Vision, and Material Sciences*, Cambridge University Press.

Snyder, W., Han, Y.-S., Bilbro, G., Whitaker, R. and Pizer, S.: 1995, Image relaxation: restoration and feature extraction, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17**(6), 620–624.

Staib, L. and Duncan, J.: 1992, Boundary finding with parametrically deformable models, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **14**, 1061–1075.

Szeliski, R., Tonnesen, D. and Terzopoulos, D.: 1993, Modeling surfaces of arbitrary topology with dynamic particles, *Proceedings Fourth International Conference on Computer Vision (ICCV'93)*, IEEE Computer Society Press, Berlin, Germany, pp. 82–87.

Terzopoulos, D., Witkin, A. and Kass, M.: 1988, Constraints on deformable models: Recovering 3d shape and nonrigid motion, *Artificial Intellegence* **36**(1), 91–123.

Turk, G. and Levoy, M.: 1994, Zippered polygon meshes from range images, *SIGGRAPH '94 Proceedings*, pp. 311–318.

Whitaker, R. T.: 1994, Volumetric deformable models: Active blobs, *in* R. A. Robb (ed.), *Visualization In Biomedical Computing 1994*, SPIE, Mayo Clinic, Rochester, Minnesota, pp. 122–134.

Whitaker, R. T.: 1995, Algorithms for implicit deformable models, *in* ICC (1995).

Whitaker, R. T. and Chen, D. T.: 1994, Embedded active surfaces for volume visualization, *SPIE Medical Imaging 1994*, Newport Beach, California.

Zhang, Z.: 1994, Iterative point matching for registration of free-form curves and surfaces, *Int. J. Computer Vision* **13**, 119–152.

Appendix

A formal definition of active sets and the operations performed on them gives some general properties regarding their behavior during the sparse-field algorithm. Let u be a discrete approximation to ϕ on a rectangular grid. On each of the grid points (or voxels in 3D), u has a value equal to that of ϕ. An active set is a set of grid points that are adjacent to the level set. For notational convenience the zero-level set is used, and thus active sets consist of grid points that lie near zero crossings. As the level set moves, two things happen to active sets. First, the values of active set and nearby points change. Second, points are added and removed from the active set, i.e., activity is *passed* on from one grid point to another. The remainder of this section serves to formalize these ideas and show that active sets maintain their important properties as the sparse-field algorithm progresses. In order to maintain generality, the discussion proceeds in n dimensions, where $n = 3$ for surface reconstruction problem at hand.

A volume $I = \{X, u\}$ of dimension n is a set of grid points, X, arranged in a rectilinear grid and a discrete mapping $u : X \mapsto \mathbb{R}$. A grid point $x_j \in X$ has an associated index $i(x_j) \in \mathrm{I}^n$ (i.e., $i = i_1, \ldots, i_n$ and $i_j \in \mathrm{I}$). This index has a basis, $e_1, \ldots, e_n$, which consists of unit vectors along the grid lines, i.e., $e_1 = (1, 0, \cdots), e_2 = (0, 1, 0, \cdots), \cdots, e_n = (0, \cdots, 1)$.

Each grid point has a set of $2n$ connected neighbors which are given by $C(x_j) = \{x_k | i(x_k) = i(x_j) \pm e_m, 1 <= m <= n\}$. That is, the $2d$ neighbors of x_j, given by $C(x_j)$, are those points that lie adjacent to x_j in one of the grid directions.

One minor point is the handling of boundary conditions on the domain. For this work we assume that every grid point has $2d$ adjacent grid points, meaning either a toroidal topology (wrap around) or an infinite array of grid points. All of the results described in this section can be extended to other topologies or boundary conditions.

The *adjacent* operator, $A(x_i, x_j)$ for $x_i, x_j \in X$, indicates whether or not two or more grid points are adjacent, i.e.,

$$A(x_i, x_j) \stackrel{\triangle}{=} x_j \in C(x_i). \tag{41}$$

The sign operator $\mathrm{O}(x_i, x_j, u)$ indicates whether or not the values of u associated with two grid points have values of opposite sign, i.e.,

$$\mathrm{O}(x_i, x_j, u) \stackrel{\triangle}{=} \left(u_{x_i} \geq 0 \text{ and } u_{x_j} 0\right) \text{ or } \left(u_{x_i} < 0 \text{ and } u_{x_j} \geq 0\right). \tag{42}$$

Note that both $\mathrm{O}(x_i, x_j)$ and $A(x_i, x_j, u)$ are commutative. An adjacent grid point pair with values that have opposite signs are said to lie on a *zero crossing*.

An *active set* $\mathcal{A}$ is a subset of X such that for every adjacent pair with opposite sign, at least one of the grid points of that pair is in the active set. That is, $\mathcal{A} \subset X$ is an active set if and only if $\forall x_i, x_j \in X$, $O(x_i, x_j, u)$ and $A(x_i, x_j) \implies \{x_i, x_j\} \cap \mathcal{A} \neq \emptyset$. This definition means that an active set could include grid points that are not necessary to satisfy the condition. An active set is said to be *efficient* if for all members of the active set have a neighbor of opposite sign, i.e., $\mathcal{A}$ is efficient if and only if $\forall x_j \in \mathcal{A}\ \exists\ x_i$ s.t. $O(x_i, x_j, u)$ and $A(x_i, x_j)$. Efficient active sets consist entirely of grid points that lie on zero crossings.

There are several properties of active sets that are important. First, an efficient active set can be constructed from a volume (if it is finite) by testing the values of each grid point and its neighbors and constructing a union of all of those grid points that lie on zero crossings. Second, an efficient active set can be constructed from any (finite) active set by successively removing those grid points that do not lie on zero crossings. Third, active sets divide images into enclosed positive and negative regions. This is true because, by the definition of an active set, there is no connected path (a sequence of adjacent grid points) from a grid point with positive value to a grid point with negative value that does not pass through an active grid point.

The sparse field algorithm works with active sets and performs two distinct actions on these sets; it moves the activity of grid points to adjacent grid points, and it changes the values of grid points. The following results show that sparse-field algorithm maintains the properties of active sets (they continue to divide positive and negative regions).

A *movement* of an active set $\mathcal{A}$ is a procedure for constructing a new set of grid points $\mathcal{A}' = M(\mathcal{A}, I)$. This procedure is to remove a grid point $x_j \in \mathcal{A}$ from $\mathcal{A}$ and add to $\mathcal{A}$ all of x_j's neighbors that have opposite sign. That

is,

$$M(\mathcal{A}, I) \triangleq \{\mathcal{A} - \{x_j\}\} \cup \{x_i | A(x_i, x_j) \text{ and } O(x_i, x_j, u)\}. \quad (43)$$

Proposition 1 *A set of grid points $\mathcal{A}'$ constructed by a movement procedure, $\mathcal{A}' = M(\mathcal{A}, I)$, is an active set.*

Proof Proposition 1 Let x_j denote the grid point that is removed. Assume that $\mathcal{A}'$ is not an active set. Then there exists a pair of grid points $\{x_i, x_k\}$ such that $O(x_i, x_j, u)$ and $A(x_i, x_k)$ and $\{x_i, x_k\} \cap \mathcal{A}' = \emptyset$. There are two cases to consider:

$x_j \notin \{x_i, x_k\}$ In this case neither of the grid points $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$. Thus $\mathcal{A}$ is not an active set, which violates one of the assumptions.

$x_j \in \{x_i, x_k\}$ This also violates the assumption because all opposite-sign neighbors of x_j are in $\mathcal{A}'$.

□

Proposition 2 *If $\mathcal{A}$ is an efficient active set, then the set of grid points $\mathcal{A}'$ constructed by a movement procedure $\mathcal{A}' = M(\mathcal{A}, I)$, is also an efficient active set.*

Proof Proposition 2 Let x_j denote the grid point that is removed. $\mathcal{A}'$ is an active set by Proposition 1. Thus only the proof of efficiency is necessary. Assume that $\mathcal{A}'$ is not efficient, then there exists an active grid point x_i with a neighborhood $N = \{x_k | A(x_i, x_k\}$ such that $\{x_k | O(x_i, x_k, u)\} \cap N = \emptyset$. There are two cases to consider:

$x_j \in \mathcal{A}$: This violates the assumption because $\mathcal{A}$ is assumed to be efficient.

$x_j \notin \mathcal{A}'$: In this case x_j is one of the grid points that is added to $\mathcal{A}'$ by the movement procedure. However, these grid points are added because they have a neighbor of opposite sign, namely x_j.

□

An *update* of an active set is the construction of a new image which has a new value for one of its active points, i.e., $I' = \{X, u'\}$ where $u' = U(u, \mathcal{A})$.

Proposition 3 *An update of I to I' with $I' = \{X, u'\}$ and $u' = U(u, \mathcal{A})$ preserves the active set $\mathcal{A}$, i.e., $\mathcal{A}$ is still an active set of I'.*

Proof Proposition 3 Let x_j be the grid point with a value that is changed. Assume that $\mathcal{A}$ is not an active set of I'. Then there exists a pair of grid points $\{x_i, x_k\}$ such that $O(x_i, x_j, u')$ and $A(x_i, x_k)$ and $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$. There are two cases to consider:

$x_j \notin \{x_i, x_k\}$ In this case neither of the grid points $\{x_i, x_k\} \cap \mathcal{A} = \emptyset$, which means that $\mathcal{A}$ is not an active set of X, u (since u and u' differ only in the value at x_j), which violates one of the assumptions.

$x_j \in \{x_i, x_k\}$ This also violates the assumption because x_j is by definition in the active set $\mathcal{A}$.

□

It is provable that the update procedure does not preserve the efficiency of an active set in all cases. Certain combinations of grid points can be shown to produce inefficient active sets after an update procedure. Pathological cases can produce dense sets of active points that fill whole regions and render the sparse-field method inefficient.

One solution to the problem of inefficiency is a *pruning* process which systematically removes unnecessary active points. A pruning procedure $P(u, \mathcal{A})$ is a procedure that produces a new active set $\mathcal{A}' = \mathcal{A} - \{x_j\}$ if the neighborhood N of x_j is of the same sign, i.e., $O(x_i, x_j, u) \forall x_i \in N$, and $\mathcal{A}' = \mathcal{A}$ otherwise. The pruning procedure can be repeated for each $x_j \in \mathcal{A}$ after an update step and can be shown to produce an efficient active set. The author has found that in practice such a pruning procedure is unnecessary, and that under normal situations active sets remain efficient except at singularities, i.e., where level sets break, join, or disappear.

Level Set Surface Editing Operators

Ken Museth* David E. Breen* Ross T. Whitaker† Alan H. Barr*

*Computer Science Department
California Institute of Technology

†School of Computing
University of Utah

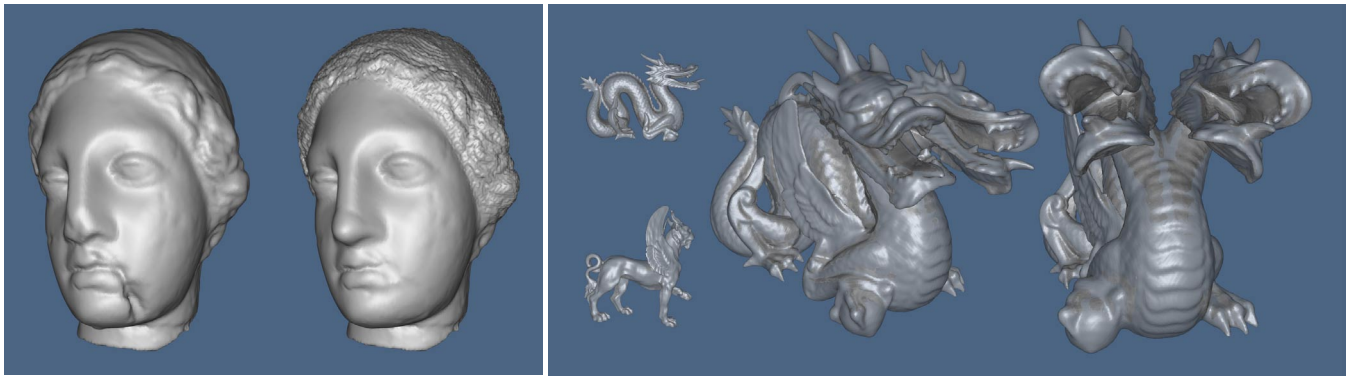


Figure 1: Surfaces edited with level set operators. Left: A damaged Greek bust model is repaired with a new nose, chin and sharpened hair. Right: An edited model is constructed from models of a griffin and dragon (small figures), producing a two-headed, winged dragon.

Abstract

We present a level set framework for implementing editing operators for surfaces. Level set models are deformable implicit surfaces where the deformation of the surface is controlled by a speed function in the level set partial differential equation. In this paper we define a collection of speed functions that produce a set of surface editing operators. The speed functions describe the velocity at each point on the evolving surface in the direction of the surface normal. All of the information needed to deform a surface is encapsulated in the speed function, providing a simple, unified computational framework. The user combines pre-defined building blocks to create the desired speed function. The surface editing operators are quickly computed and may be applied both regionally and globally. The level set framework offers several advantages. 1) By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. 2) Level set models easily change topological genus, and 3) are free of the edge connectivity and mesh quality problems associated with mesh models. We present five examples of surface editing operators: blending, smoothing, sharpening, openings/closings and embossing. We demonstrate their effectiveness on several scanned objects and scan-converted models.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Surface and object representations; I.3.4 [Computer Graphics]: Graphics Utilities—Graphics Editors;

Keywords: Deformations, geometric modeling, implicit surfaces, shape blending.

Computer Graphics Proceedings, Annual Conference Series (SIGGRAPH 2002), July 2002. © 2002 ACM pp. 330-338 reprinted with permission.

1 Introduction

The creation of complex models for such applications as movie special effects, graphic arts, and computer-aided design can be a time-consuming, tedious, and error-prone process. One of the solutions to the model creation problem is 3D photography [Bouguet and Perona 1999], i.e. scanning a 3D object directly into a digital representation. However, the scanned model is rarely in a final desired form. The scanning process is imperfect and introduces errors and artifacts, or the object itself may be flawed.

3D scans can be converted to polygonal and parametric surface meshes [Edelsbrunner and Mücke 1994; Bajaj et al. 1995; Amenta et al. 1998]. Many algorithms and systems for editing these polygonal and parametric surfaces have been developed [Cohen et al. 2001], but surface mesh editing has its limitations and must address several difficult issues. For example, it is difficult to guarantee that a mesh model will not self-intersect when performing a local editing operation based on the movement of vertices or control points, producing non-physical, invalid results. See Figure 2. If self-intersection occurs, it must be fixed as a post-process. Also, when merging two mesh models the process of clipping individual polygons and patches may produce errors when the elements are small and/or thin, or if the elements are almost parallel. In addition while it is not impossible to change the genus of a surface mesh model [Biermann et al. 2001], it is certainly difficult and requires significant effort to maintain the consistency/validity of the underlying vertex/edge connectivity structure.

1.1 New Surface Editing Operators

In order to overcome these difficulties we present a level set approach to implementing operators for locally and globally editing closed surfaces. Level set models are deformable implicit surfaces that have a volumetric representation [Osher and Sethian 1988]. They are defined as an iso-surface, i.e. a level set, of some implicit function ϕ. The surface is deformed by solving a partial differential equation (PDE) on a regular sampling of ϕ, i.e. a volume dataset. To date level set methods have not been developed for adaptive grids, a limitation of current implementations, but not of the mathematics. It should be emphasized that level set methods do not manipulate an explicit closed form representation of ϕ, but only

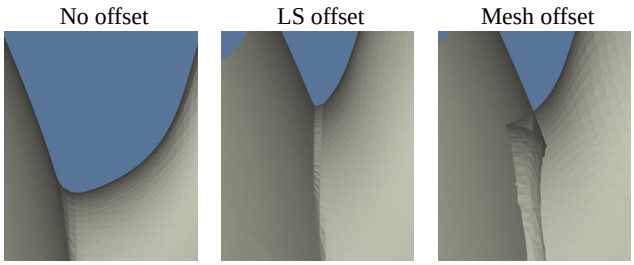


Figure 2: (left) A cross-section of the teapot model near the spout. (middle) No self-intersection occurs, by construction, when performing a level set (LS) offset, *i.e.* dilation, of the surface. (right) Self-intersections may occur when offsetting a mesh model.

a sampling of it. Level set methods provide the techniques needed to change the voxel values of the volume in a way that deforms the embedded iso-surface to meet a user-defined goal. The user controls the deformation of the level set surface by defining a speed function $\mathcal{F}(\boldsymbol{x}, \ldots)$, the speed of the level set at point $\boldsymbol{x}$ in the direction of the normal to the surface at $\boldsymbol{x}$. Therefore all the information needed to deform a level set model may be encapsulated in a single speed function $\mathcal{F}()$, providing a simple, unified computational framework.

We have developed a number of surface editing operators within the level set framework by defining a collection of new level set speed functions. The cut-and-paste operator (Section 5.1) gives the user the ability to copy, remove and merge level set models (using volumetric CSG operations) and automatically blends the intersection regions (See Section 5.2). Our smoothing operator allows a user to define a region of interest and smooths the enclosed surface to a user-defined curvature value. See Section 5.3. We have also developed a point-attraction operator. See Section 5.4. Here, a regionally constrained portion of a level set surface is attracted to a single point. By defining line segments, curves, polygons, patches and 3D objects as densely sampled point sets, the single point attraction operator may be combined to produce a more general surface embossing operator. As noted by others, the opening and closing morphological operators may be implemented in a level set framework [Sapiro et al. 1993; Maragos 1996]. We have also found them useful for performing global blending (closing) and smoothing (opening) on level set models. Since all of the operators accept and produce the same volumetric representation of closed surfaces, the operators may be applied repeatedly to produce a series of surface editing operations. See Figure 11.

1.2 Benefits and Issues

Performing surface editing operations within a level set framework provides several advantages and benefits. Many types of surfaces may be imported into the framework as a distance volume, a volume dataset that stores the signed shortest distance to the surface at each voxel. This allows a number of different types of surfaces to be modified with a single, powerful procedure. By construction, the framework always produces non-self-intersecting surfaces that represent physically-realizable objects, an important issue in computer-aided design. Level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with deforming and modifying mesh models. Additionally, some reconstruction algorithms produce volumetric models [Curless and Levoy 1996; Whitaker 1998; Zhao et al. 2001] and volumetric scanning systems are increasingly being employed in a number of diverse fields. Therefore volumetric models are becoming more prevalent and there is a need to develop powerful editing operators that act on these types of models directly.

There are implementation issues to be addressed when using level set models. Given their volumetric representation, one may be concerned about the amount of computation time and memory needed to process level set models. Techniques have been developed to limit level set computations to only a narrow band around the level set of interest [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] making the computational complexity proportional to the surface area of the model. We have also developed computational techniques that allow us to perform the narrow band calculations only in a portion of the volume where the level set is actually moving. Additionally, fast marching methods have been developed to rapidly evaluate the level set equation under certain circumstances [Tsitsiklis 1995; Sethian 1996]. Memory usage has not been an issue when generating the results in this paper. The memory needed for our results (512 MB) is available on standard workstations and PCs. We have implemented our operators in an interactive environment that allows us to easily edit a number of complex surfaces. Additionally, concerns have been raised that volume-based models cannot represent fine or sharp features. Recent advances [Frisken et al. 2000; Kobbelt et al. 2001] have shown that is is possible to model these kinds of structures with volume datasets, without excessively sampling the whole volume. These advances will also be available for our operators once adaptive level set methods, an active research area, are developed.

1.3 Contributions

The major contributions of our work are the following.

- The introduction of a unified approach to surface editing within a level set framework.
  - Editing operators defined by speed functions.
  - Results produced by solving a PDE.
- The definition of level set speed functions that implement blending, smoothing and embossing surface editing operators.
  - Blending is automatic and is constrained to only occur within a user-specified distance to an arbitrarily complex intersection curve.
  - Smoothing and embossing are constrained to occur within a user-specified region.
  - The user specifies the local geometric properties of the resulting surface modifications.
  - The user specifies if material should be added and/or removed during editing operations.
- The new techniques used to localize level set calculations.
- In Appendix B we present a new, numerically-stable curvature measure for level set surfaces.

2 Previous Work

Three areas of research are closely related to our level set surface editing work; volumetric sculpting, mesh-based surface editing/fairing and implicit modeling. Volumetric sculpting provides methods for directly manipulating the voxels of a volumetric model. CSG Boolean operations [Hoffmann 1989; Wang and Kaufman 1994] are commonly found in volume sculpting systems, providing a straightforward way to create complex solid objects by combining simpler primitives. One of the first volume sculpting systems is presented in [Galyean and Hughes 1991]. [Wang and Kaufman 1995] improved on this work by introducing tools for carving and sawing. More recently [Perry and Frisken 2001] implemented a volumetric sculpting system based on the Adaptive Distance Fields (ADF) [Frisken et al. 2000], allowing for models with adaptive resolution.

Performing CSG operations on mesh models is a long-standing area of research [Requicha and Voelcker 1985; Laidlaw et al. 1986].

Recently CSG operations were developed for multi-resolution subdivision surfaces by [Biermann et al. 2001], but this work did not address the problem of blending or smoothing the sharp features often produced by the operations. However, the smoothing of meshes has been studied on several occasions [Welch and Witkin 1994; Taubin 1995; Kobbelt et al. 1998]. [Desbrun et al. 1999] have developed a method for fairing irregular meshes using diffusion and curvature flow, demonstrating that mean-curvature based flow produces the best results for smoothing.

There exists a large body of surface editing work based on implicit models [Bloomenthal et al. 1997]. This approach uses implicit surface representations of analytic primitives or skeletal offsets. The implicit modeling work most closely related to ours is found in [Wyvill et al. 1999]. They describe techniques for performing blending, warping and boolean operations on skeletal implicit surfaces. [Desbrun and Gascuel 1995] address the converse problem of preventing unwanted blending between implicit primitives, as well as maintaining a constant volume during deformation.

Level set methods have been successfully applied in computer graphics, computer vision and visualization [Sethian 1999; Sapiro 2001], for example medical image segmentation [Malladi et al. 1995; Whitaker et al. 2001], shape morphing [Breen and Whitaker 2001], 3D reconstruction [Whitaker 1998; Zhao et al. 2001], and recently for the animation of liquids [Foster and Fedkiw 2001]

Our work stands apart from previous work in several ways. We have not developed volumetric modeling tools. Our editing operators act on surfaces that happen to have an underlying volumetric representation, but are based on the mathematics of deforming implicit surfaces. Our editing operators share several of the capabilities of mesh-based tools, but are not hampered by the difficulties of maintaining vertex/edge information. Since level set models are not tied to any specific implicit basis functions, they easily represent complex models to within the resolution of the sampling. Our work is the first to utilize level set methods to perform user-controlled editing of complex geometric models.

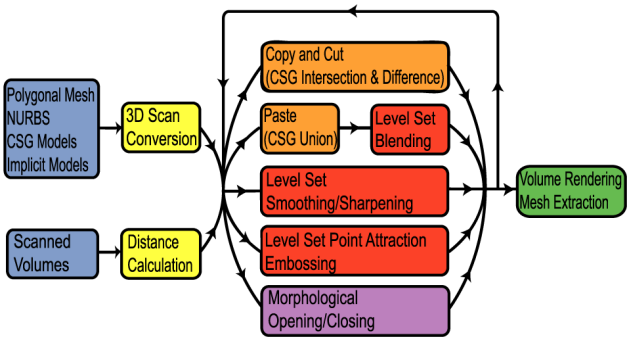


Figure 3: Our level set surface editing operators (red) fit into a larger editing framework. The pipeline consists of: input models (blue), pre-processing (yellow), CSG operations (orange), local LS operators (red), global LS operators (purple) and rendering (green).

3 Overview of the Editing Pipeline

The level set surface editing operators should be viewed as components of a larger modeling framework. The pipeline for this framework is presented in Figure 3. The red components contain the level set speed functions that we have developed for localized surface editing. The remaining components contain the data and operations needed for level set modeling, input models (blue), preprocessing (yellow), CSG operations (orange), global LS operators (purple) and rendering (green). The pipeline provides the context for the details of our speed functions.

3.1 Input and Output Models

We are able to import a wide variety of closed geometric models into the level set environment. We represent a level set model as an iso-surfaces embedded in a distance volume. Frequently we only store distance information in a narrow band of voxels surrounding the level set surface. As illustrated in Figure 3 we have developed and collected a suite of scan conversion methods for converting polygonal meshes, CSG models [Breen et al. 2000], implicit primitives, and NURBS surfaces into distance volumes. Additionally many types of scanning processes produce volumetric models directly, e.g. MRI, CT and laser range scan reconstruction. These models may be brought into our level set environment as is or with minimal pre-processing. We frequently utilize Sethian's Fast Marching Method [Sethian 1996] to convert volume datasets into distance volumes. The models utilized in this paper and their original form are listed in Table 1.

Table 1: Native representations of the input models and dimensions of the corresponding scan converted distance volumes.

Model	Representation	Dimensions
Dragon	volumetric reconstruction	$356 \times 161 \times 251$
Griffin	volumetric reconstruction	$312 \times 148 \times 294$
Greek bust	polygonal reconstruction	$221 \times 221 \times 161$
Human head	polygonal reconstruction	$256 \times 246 \times 193$
Utah teapot	NURBS surface	$156 \times 232 \times 124$
Supertoroid	implicit primitive	$91 \times 91 \times 31$

In the final stage of the pipeline we can either volume render the surface directly or render a polygonal mesh extracted from the volume. While there are numerous techniques available for both approaches, we found extracting and rendering Marching Cubes meshes [Lorensen and Cline 1987] to be satisfactory.

4 Level Set Surface Modeling

The Level Set Method, first presented in [Osher and Sethian 1988], is a mathematical tool for modeling surface deformations. A deformable (*i.e.* time-dependent) surface is implicitly represented as an iso-surface of a time-varying scalar function, $\phi(\boldsymbol{x}, t)$. A detailed description of level set models is presented in Appendix A.

4.1 LS Speed Function Building Blocks

Given the definition

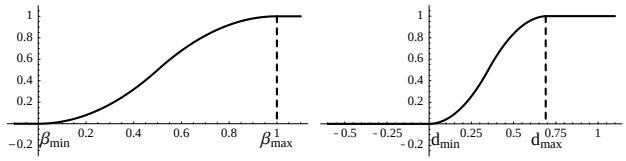
$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \equiv \boldsymbol{n} \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{1}$$

the fundamental level set equation, Eq. (12), can be rewritten as

$$\frac{\partial \phi}{\partial t} = |\nabla \phi| \, \mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi, \dots) \tag{2}$$

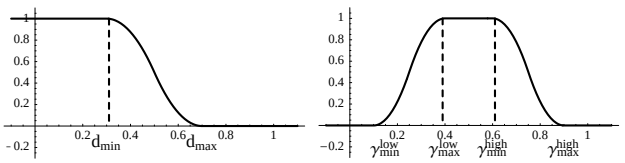
where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ and $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$ are the velocity and normal vectors at $\boldsymbol{x}$ on the surface. We assume a positive-inside/negative-outside sign convention for $\phi(\boldsymbol{x}, t)$, *i.e.* $\boldsymbol{n}$ *points outward*. Eq. (1) introduces the speed function $\mathcal{F}$, which is a user-defined scalar function that can depend on any number of variables including $\boldsymbol{x}$, $\boldsymbol{n}$, ϕ and its derivatives evaluated at $\boldsymbol{x}$, as well as a variety of external data inputs. $\mathcal{F}()$ is a *signed* scalar function that defines the motion (*i.e.* speed) of the level set surface in the direction of the local normal $\boldsymbol{n}$ at $\boldsymbol{x}$.

The speed function is usually based on a set of geometric measures of the implicit level set surface and data inputs. The challenge when working with level set methods is determining how to combine the building blocks to produce a local motion that creates a desired global or regional behavior of the surface. The general



(a) Building-block, $P(\beta)$, for the ROI functions, *cf.* Eq. (4).

(b) Function, $\mathcal{D}_s(d)$, of distance to a surface, *cf.* Eq. (6b).



(c) Function, $\mathcal{D}_p(d)$, of distance to point-set, *cf.* Eq. (6a).

(d) Function, $\mathcal{C}(\gamma)$, of geometric measure, γ, *cf.* Eq. (7).

Figure 4: Graph of region-of-influence (ROI) functions used to define the speed functions for our local level set operations, *cf.* Eq. (3).

structure for the speed functions used in our surface editing operators is

$$\mathcal{F}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \mathcal{D}_q(d)\mathcal{C}(\gamma)\mathcal{G}(\gamma) \tag{3}$$

where $\mathcal{D}_q(d)$ is a distance-based cut-off function which depends on a distance measure d to a geometric structure q. $\mathcal{C}(\gamma)$ is a cut-off function that controls the contribution of $\mathcal{G}(\gamma)$ to the speed function. $\mathcal{G}(\gamma)$ is a function that depends on geometric measures γ derived from the level set surface, e.g. curvature. Thus, $\mathcal{D}_q(d)$ acts as a region-of-influence function that regionally constrains the LS calculation. $\mathcal{C}(\gamma)$ is a filter of the geometric measure and $\mathcal{G}(\gamma)$ provides the geometric contribution of the level set surface. In general γ is defined as zero, first, or second order measures of the LS surface.

4.2 Regionally Constraining LS Deformations

Most of our surface operators may be applied locally in a small user-defined region on the edited surface. In order to regionally restrict the deformation during the level set computation, a technique is needed for driving the value of $\mathcal{F}()$ to zero outside of the region. This is accomplished in three steps. The first step involves defining the region of influence (ROI), *i.e.* the region where $\mathcal{F}()$ should be non-zero. This is done by either the user interactively placing a 3D object around the region, or by automatically calculating a region from properties of the surface. Both cases involve defining a geometric structure that we refer to as a "region-of-influence (ROI) primitive". The nature of these primitives will vary for the different LS operations and will explicitly be defined in Section 5. The second step consists of calculating a distance measure to the ROI primitive. The final step involves defining a function that smoothly approaches zero at the boundary of the ROI.

We define a region-of-influence function $\mathcal{D}_q(d)$ in Eq. (3), where d is a distance measure from a point on the level set surface to the ROI primitive q. The functional behavior of $\mathcal{D}_q(d)$ clearly depends on the specific ROI primitive, q, but we found the following piecewise polynomial function to be useful as a common speed function building block:

$$P(\beta) = \begin{cases} 0 & \text{for } \beta \leq 0 \\ 2\beta^2 & \text{for } 0 < \beta \leq 0.5 \\ 1 - 2(\beta - 1)^2 & \text{for } 0.5 < \beta < 1 \\ 1 & \text{for } \beta \geq 1. \end{cases} \tag{4}$$

$P(\beta)$ and its derivatives are continuous and relatively inexpensive to compute. See Figure 4(a). Other continuous equations with the same basic shape would also be satisfactory. We then define

$$\mathcal{P}(d; d_{min}, d_{max}) \equiv P\left(\frac{d - d_{min}}{d_{max} - d_{min}}\right) \tag{5}$$

where d_{min} and d_{max} are user-defined parameters that define the limits and sharpness of the cut-off. Let us finally define the following region-of-influence functions

$$\mathcal{D}_p(d) = 1 - \mathcal{P}(d; d_{min}, d_{max}) \tag{6a}$$

$$\mathcal{D}_s(d) = \mathcal{P}(d; 0, d_{max}) \tag{6b}$$

for a point-set, p, and a closed surface, s.

In Eq. (6a) d denotes the distance from a point on the level set surface to the closet point in the point-set p. In Eq. (6b) d denotes a signed distance measure from a point on the level set surface to the implicit surface s. The signed distance measure does not necessarily have to be Euclidean distance - just a monotonic distance measure following the positive-inside/negative-outside convention. Note that $\mathcal{D}_p(d)$ is one when the shortest distance, d, to the point-set is smaller than d_{min}, and decays smoothly to zero as d increases to d_{max}, after which it is zero. $\mathcal{D}_s(d)$, on the other hand, is zero everywhere outside, as well as on, the surface s ($d \leq 0$), but one inside when the distance measure d is larger than d_{max}.

An additional benefit of the region-of-influence functions is that they define the portion of the volume where the surface cannot move. We use this information to determine what voxels should be updated during the level set deformation, significantly lowering the amount of computation needed when performing editing operations. This technique allows our operators to be rapidly computed when modifying large models.

4.3 Limiting Geometric Property Values

We calculate a number of geometric properties from the level set surface. The zero order geometric property that we utilize is shortest distance from the level set surface to some ROI primitive. The first order property is the surface normal, $\boldsymbol{n} \equiv -\nabla\phi/|\nabla\phi|$. Second order information includes a variety of curvature measures of the LS surface. In Appendix B we outline a new numerical approach to deriving the mean, Gaussian and principle curvatures of a level set surface. Our scheme has numerical advantages relative to traditional central finite difference schemes for computing the second order derivatives. We found the mean curvature to be a useful second order measure [Evans and Spruck 1991].

Another desirable feature of our operators is that they allow the user to control the geometric properties of surface in the region being edited. This feature is implemented with another cut-off function, $\mathcal{C}()$, within the level set speed function. $\mathcal{C}()$ allows the user to slow and then stop the level set deformation as a particular surface property approaches a user-specified value. We reuse the cut-off function, Eq. (5), defined in the previous section, as a building block for $\mathcal{C}()$. We define

$$\mathcal{C}(\gamma) = \begin{cases} \mathcal{P}(\gamma; \gamma_{min}^{low}, \gamma_{max}^{low}) & \text{for } \gamma \leq \overline{\gamma} \\ 1 - \mathcal{P}(\gamma; \gamma_{min}^{high}, \gamma_{max}^{high}) & \text{for } \gamma > \overline{\gamma} \end{cases} \tag{7}$$

where $\overline{\gamma} \equiv (\gamma_{max}^{low} + \gamma_{min}^{high})/2$. The four parameters γ_{min}^{low}, γ_{max}^{low}, γ_{min}^{high}, and γ_{max}^{high} define respectively the upper and lower support of the filter, see Figure 4(d).

4.4 Constraining the Direction of LS Motions

Another important feature of the level set framework is its ability to control the direction of the level set deformation. We are able to restrict the motion of the surface to only add or remove material

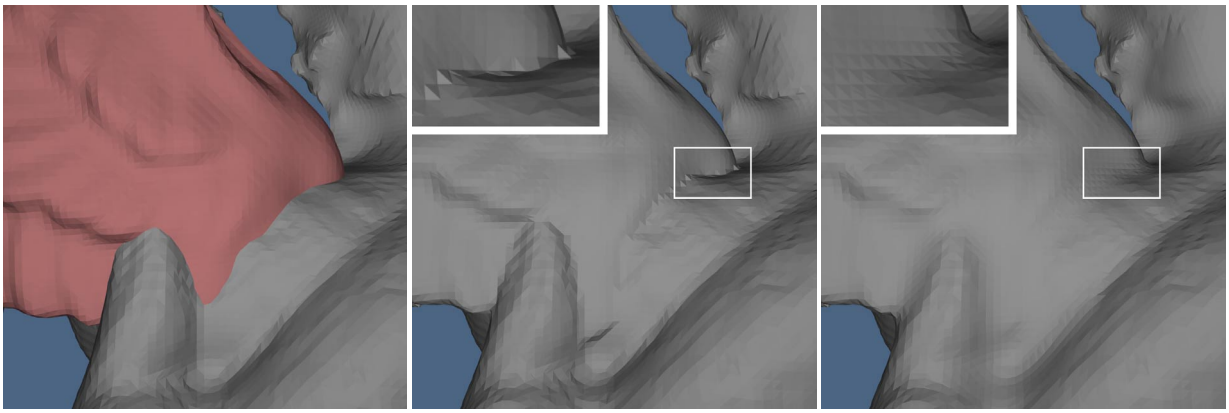


Figure 5: Left: Positioning the (red) wing model on the dragon model. Middle: The models are pasted together (CSG union operation), producing sharp, undesirable creases, a portion of which is expanded in the box. Right: Same region after automatic blending based on mean curvature. The blending is constrained to only move outwards. The models are rendered with flat-shading to highlight the details of the surface structure.

during the level set editing operations. At any point the level set surface can only move in the direction of the local surface normal. Hence, we can simply redefine the speed function as $\min(\mathcal{G}, 0)$ to remove material (inward motion only) and $\max(\mathcal{G}, 0)$ to add material (outward motion only). In the case of curvature driven speed functions this produces min/max flows [Sethian 1999]. Of course no restriction on the direction of the motion need be imposed.

5 Definition of Surface Editing Operators

Given the building blocks described in the previous section, the level set surface editing operators outlined in Figure 3 may be defined. We begin by defining the well-known CSG operations that are essential to most editing systems. We then define the new level set speed functions that implement our surface editing operators by combining the geometric measures with the region-of-influence and cut-off functions.

5.1 CSG Operations

Since level set models are volumetric, the constructive solid geometry (CSG) [Hoffmann 1989] operations of union, difference and intersection may be applied to them. This provides a straightforward approach to copy, cut and paste operations on the level set surfaces. In our level set framework with a positive-inside/negative-outside sign convention for the distance volumes these are implemented as min/max operations [Wang and Kaufman 1994] on the voxel values as summarized in Table 2. Any two closed surfaces represented as signed distance volumes can be used as either the main edited model or the cut/copy primitive. In our editing system the user is able to arbitrarily scale, translate and rotate the models before a CSG operation is performed.

Table 2: Implementation of CSG operations on two level set models, A and B, represented by distance volumes V_A and V_B with positive inside and negative outside values.

Action	CSG Operation	Implementation
Copy	Intersection, $A \cap B$	$\text{Min}(V_A, V_B)$
Paste	Union, $A \cup B$	$\text{Max}(V_A, V_B)$
Cut	Difference, $A - B$	$\text{Min}(V_A, -V_B)$

5.2 Automatic Localized LS Blending

The surface models produced by the CSG paste operation typically produces sharp and sometimes jagged creases at the intersection of the two surfaces. We can dramatically improve this region of the surface by applying an automatic localized blending. The method is automatic because it only requires the user to specify a few parameter values. It is localized because the blending operator is only applied near the surface intersection region. One possible solution to localizing the blending is to perform the deformation in regions near both of the input surfaces. However, this naive approach would result in blending the two surfaces in *all* regions of space where the surfaces come within a user-specified distance of each other, creating unwanted blends. A better solution, and the one we use, involves defining the region of influence based on the distance to the *intersection curve* shared by both input surfaces. A sampled representation of this curve is the set of voxels that contains a zero distance value (within some sub-voxel value ϵ) to both surfaces. We have found this approximate representation of the intersection curve as a point-set to be sufficient for defining a shortest distance d for the region-of-influence function, $\mathcal{D}_p(d)$, *cf.* Eq. (3). Representing the intersection curve by a point-set allows the curve to take an arbitrary form - it can even be composed of multiple curve segments without introducing any complication to the computational scheme.

The blending operator moves the surface in a direction that minimizes a curvature measure, $\mathcal{K}$, on the level set surface. This is obtained by making the speed function, $\mathcal{G}$, Eq. (3), proportional to $\mathcal{K}$, leading to the following blending speed function:

$$\mathcal{F}_{blend}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha\, \mathcal{D}_p(d)\, \mathcal{C}(\mathcal{K})\, \mathcal{K} \tag{8}$$

where α is a user-defined positive scalar that is related to the rate of convergence of the LS calculation, $\mathcal{D}_p(d)$ is defined in Eq. (6a) where d is the shortest distance from the level set surface to the intersection curve point set, and $\mathcal{C}(\mathcal{K})$ is given by Eq. (7) where $\mathcal{K}$ is one of the curvatures define in Appendix B. Through the functions $\mathcal{D}_p$ and $\mathcal{C}$ the user has full control over the region of influence of the blending (d_{min} and d_{max}) and the upper and lower curvature values of the blend (γ_{min}^{low}, γ_{max}^{low} and γ_{min}^{high}, γ_{max}^{high}). Furthermore we can control if the blend adds or removes material, or both as described in Section 4.4.

Automatic blending is demonstrated in Figure 5. A wing model is positioned relative to a dragon model. The two models are pasted together and automatic mean curvature-based blending is applied to smooth the creased intersection region.

5.3 Localized LS Smoothing/Sharpening

The smoothing operator smooths the level set surface in a user-specified region. This is accomplished by enclosing the region of interest by a geometric primitive. The "region-of-influence primitive" can be any closed surface for which we have signed inside/outside information, e.g. a level set surface or an implicit primitive. We use superellipsoids [Barr 1981] as a convenient ROI prim-

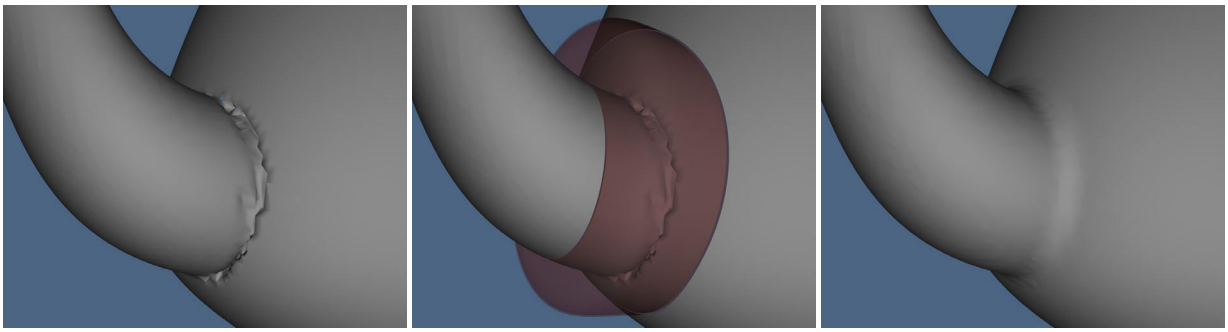


Figure 6: (left) Scan conversion errors near the teapot spout. (middle) Placing a (red) superellipsoid around the errors. (right) The errors are smoothed away in 15 seconds. The surface is constrained to only move outwards.

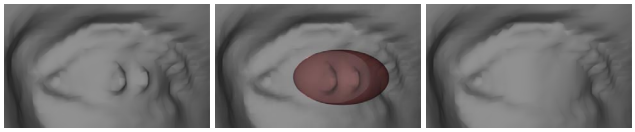


Figure 7: Regionally constrained smoothing. Left: Laser scan reconstruction with unwanted, pointed artifacts in the eye. Middle: Defining the region to be smoothed with a (red) superellipsoid. Right: Smoothing the surface within the superellipsoid. The surface is constrained to only move inwards.

itive, a flexible implicit primitive defined by two shape parameters. The surface is locally smoothed by applying motions in a direction that reduces the local curvature. This is accomplished by moving the level set surface in the direction of the local normal with a speed that is proportional to the curvature. Therefore the speed function for the smoothing operator is

$$\mathcal{F}_{smooth}(\boldsymbol{x}, \boldsymbol{n}, \phi) = \alpha \mathcal{D}_s(d)\mathcal{C}(\mathcal{K})\mathcal{K}. \tag{9}$$

Here d denotes the signed value of the monotonic inside/outside function of the ROI primitive s evaluated at $\boldsymbol{x}$. As before, $\mathcal{C}_s(d)$ ensures that the speed function smoothly goes to zero as $\boldsymbol{x}$ approaches the boundary of the ROI primitive.

Figure 7 demonstrates our smoothing operator applied to a laser scan reconstruction. Unwanted artifacts are removed from an eye by first placing a red superellipsoid around the region of interest. A smoothing operator constrained to only remove material is applied and the spiky artifacts are removed. Figure 6 demonstrates our smoothing operator applied to a preliminary 3D scan conversion of the Utah teapot. Unwanted artifacts are removed from the region where the spout meets the body of the teapot by first placing a superellipsoid around the region of interest. A smoothing operator constrained to only add material is applied and the unwanted artifacts are removed. In our final, artificial smoothing example in Figure 9 a complex structure is completely smoothed away. This examples illustrates that changes of topological genus and number of disconnected components are easily handled within a level set framework during smoothing.

We obtain a sharpening operator by simply inverting the sign of α in Eq. (9) and applying an upper cut-off to the curvature in $\mathcal{C}()$ in order to maintain numerical stability. The sharpening operator has been applied to the hair of the Greek bust in Figure 1.

5.4 Point-Set Attraction and Embossing

We have developed an operator that attracts and repels the surface towards and away from a point set. These point sets can be samples of lines, curves, planes, patches and other geometric shapes, *e.g.* text. By placing the point sets near the surface, we are able to emboss the surface with the shape of the point set. Similar to the smoothing operator, the user encloses the region to be embossed with a ROI primitive *e.g.* a superellipsoid. The region-of-interest function for this operator is $\mathcal{D}_s(d)$, Eq. (6b).

Figure 8: Left: Three types of single point attractions/repulsions using different ROI primitives and γ values. Right: Utah teapot embossed with 7862 points sampling the "SIGGRAPH 2002" logo.

First, assume that all of the attraction points are located outside the LS surface. $\boldsymbol{p}_i$ denotes the closest attraction point to $\boldsymbol{x}$, a point on the LS surface. Our operator only allows the LS surface to move towards $\boldsymbol{p}_i$ if the unit vector, $\boldsymbol{u}_i \equiv (\boldsymbol{p}_i - \boldsymbol{x})/|\boldsymbol{p}_i - \boldsymbol{x}|$, is pointing in the same direction as the local surface normal $\boldsymbol{n}$. Hence, the speed function should only be non-zero when $0 < \boldsymbol{n} \cdot \boldsymbol{u}_i \leq 1$. Since the sign of $\boldsymbol{n} \cdot \boldsymbol{u}_i$ is reversed if $\boldsymbol{p}_i$ is instead located inside the LS surface we simply require $\gamma = -\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i$ to be positive for any closest attraction point $\boldsymbol{p}_i$. This amounts to having only positive cut-off values for $\mathcal{C}(\gamma)$. Finally we let $\mathcal{G} = -\alpha\phi(\boldsymbol{p}_i, t)$ since this will guarantee that the LS surface, $\boldsymbol{x}$, actually stops once it reaches $\boldsymbol{p}_i$. The following speed function implements the point-set attraction operator:

$$\mathcal{F}_{point}(\boldsymbol{x}, \boldsymbol{n}, \phi) = -\alpha \mathcal{D}_s(d)\mathcal{C}(-\text{sign}[\phi(\boldsymbol{p}_i, t)]\boldsymbol{n} \cdot \boldsymbol{u}_i)\phi(\boldsymbol{p}_i, t), \tag{10}$$

where d is a signed distance measure to a ROI primitive evaluated at $\boldsymbol{x}$ on the LS surface, and p_i is the closest point in the set to $\boldsymbol{x}$. The shape of the primitive and the values of the four positive parameters in Eq. (7) define the footprint and sharpness of the embossing. See Figure 8, left. Point-repulsion is obtained by making α negative. Note that Eq. (10) is just one example of many possible point-set attraction speed functions.

In Figure 8, right, the Utah teapot is embossed with 7862 points that have been acquired by scanning an image of the SIGGRAPH 2002 logo and warping the points to fit the shape of the teapot.

5.5 Global Morphological Operators

The new level set operators presented above were designed to perform localized deformations of a level set surface. However, if the user wishes to perform a global smoothing of a level set surface, it is advantageous to use an operator other than $\mathcal{F}_{smooth}$. For a global smoothing the level set propagation is computed on the whole volume, which can be slow for large volumes. However, in this case morphological opening and closing operators [Serra 1982] offer faster alternatives to global smoothing of level set sur-

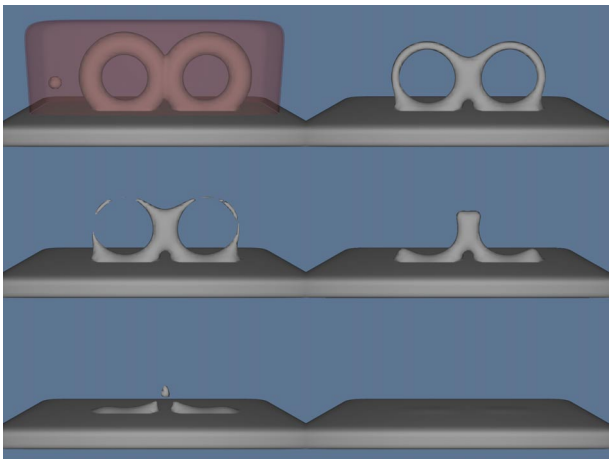


Figure 9: Changes in topological genus and the number of disconnected components are easily handled within a level set framework during smoothing. The superellipsoid defines the portion of the surface to be smoothed. The surface is constrained to move only inwards.

faces. While we are not the first to explore morphological operators within a level set framework [Sapiro et al. 1993; Maragos 1996], we have implemented them and find them useful. Morphological openings and closings consist of two fundamental operators, dilations D_ω and erosions E_ω. Dilation creates an offset surface a distance ω outward from the original surface, and erosion creates an offset surface a distance ω inwards from the original surface. The morphological opening operator O_ω is an erosion followed by a dilation, i.e. $O_\omega = D_\omega \circ E_\omega$, which removes small pieces or thin appendages. A closing is defined as $C_\omega \phi = E_\omega \circ D_\omega \phi$, and closes small gaps or holes within objects. Morphological operators may be implemented by solving a special form of the level set equation, the Eikonal equation, $\partial\phi/\partial t = \pm|\nabla\phi|$, up to a certain time t, utilizing Sethian's Fast Marching Method [Sethian 1996]. The value of t controls the offset distance from the original surface of $\phi(t = 0)$. Figure 10 contains a model from a laser scan reconstruction that has been smoothed with an opening operator with ω equal to 3.

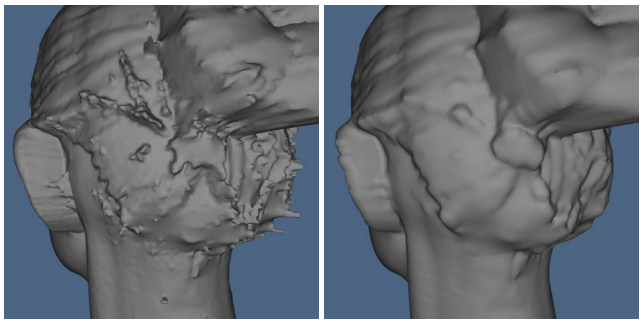


Figure 10: Applying a morphological opening to a laser scan reconstruction of a human head. The opening performs global smoothing by removing protruding structures smaller than a user-defined value.

5.6 Editing Session Details

Figure 11 contains a series of screen shots taken of our level set modeling program while constructing the two-headed winged dragon. The first shows the original dragon model loaded into the system. A cylindrical primitive is placed around its head and it is cut off. The model of the head is duplicated and the two heads are positioned relative to each other. Once the user is satisfied with their orientation, they are pasted together and an automatic blending is performed at the intersection seam. The combined double head model is positioned over the cropped neck of the dragon body. The double head is pasted and blended onto the body. The griffin model is loaded into the LS modeling system. A primitive is place around one of its wings. The portion of the model within the primitive is copied, being stored in a buffer. Several cutting operations are used to trim the wing model (not shown). The double-headed dragon model is loaded, and the wing is positioned, pasted and blended onto it. A mirror copy of the wing model is created. It is also positioned, pasted and blended onto the other side of the double-headed dragon. We then added a loop onto the dragon's back as if designing a bracelet charm. This is accomplished by positioning, pasting, and blending a scan-converted supertoroid, producing the final model seen in the bottom right.

The Greek bust model was repaired by copying the nose from the human head model of Figure 10, and pasting and blending the copied model onto the broken nose. A piece from the right side of the bust was copied, mirrored, pasted and blended onto the left side of her face. Local smoothing operators were applied to various portions of her cheeks to clean minor cracks. Finally, the sharpening operator was applied within a user-defined region around her hair.

Table 3: Typical operator execution times on a R10K 250MHz MIPS processor.

Operation	Objects	sub-volume	Time
Paste	wing on dragon	$316 \times 172 \times 215$	33 sec.
Blend	wing on dragon	$82 \times 48 \times 63$	98 sec.
Smooth	teapot spout	$60 \times 55 \times 31$	15 sec.
Opening	human head	$256 \times 246 \times 193$	22 sec.
Emboss	single point	$21 \times 29 \times 29$	1.5 sec.

Table 4: Parameters used in examples. γ_{min}^{high} and γ_{min}^{high} are only used during sharpening. Their values are 0.8 and 0.9. No upper limit is placed on γ in the other examples.

Example	d_{min}	d_{max}	γ_{min}^{low}	γ_{max}^{low}
Wing Blending	7	9	0.04	0.06
Eye Smoothing	0.9	1	0.04	0.07
Spout Smoothing	0.9	1	0.1	0.13
Hair Sharpening	0.9	1	0.01	0.013
Teapot Embossing	0.9	1	0.8	0.9

6 Conclusion and Future Work

We have presented an approach to implementing surface editing operators within a level set framework. By developing a new set of level set speed functions automatic blending, localized smoothing and embossing may be performed on level set models. Additionally we have implemented morphological and volumetric CSG operators to fill out our modeling environment. All of the information needed to deform a level set surface is encapsulated in the speed function, providing a simple, unified computational framework. The level set framework offers several advantages. By construction, self-intersection cannot occur, which guarantees the generation of physically-realizable, simple, closed surfaces. Additionally, level set models easily change topological genus, and are free of the edge connectivity and mesh quality problems associated with mesh models.

Several issues still must be addressed to improve our work. Currently level set implementations are based on uniform samplings of space, a fact that effectively limits the resolution of the objects that can be modeled. The development of adaptive level set methods would allow our operators to be applied to adaptive distance fields. It is possible to shorten the time needed to edit level set surfaces. Incrementally updating the mesh used to view the edited surface,

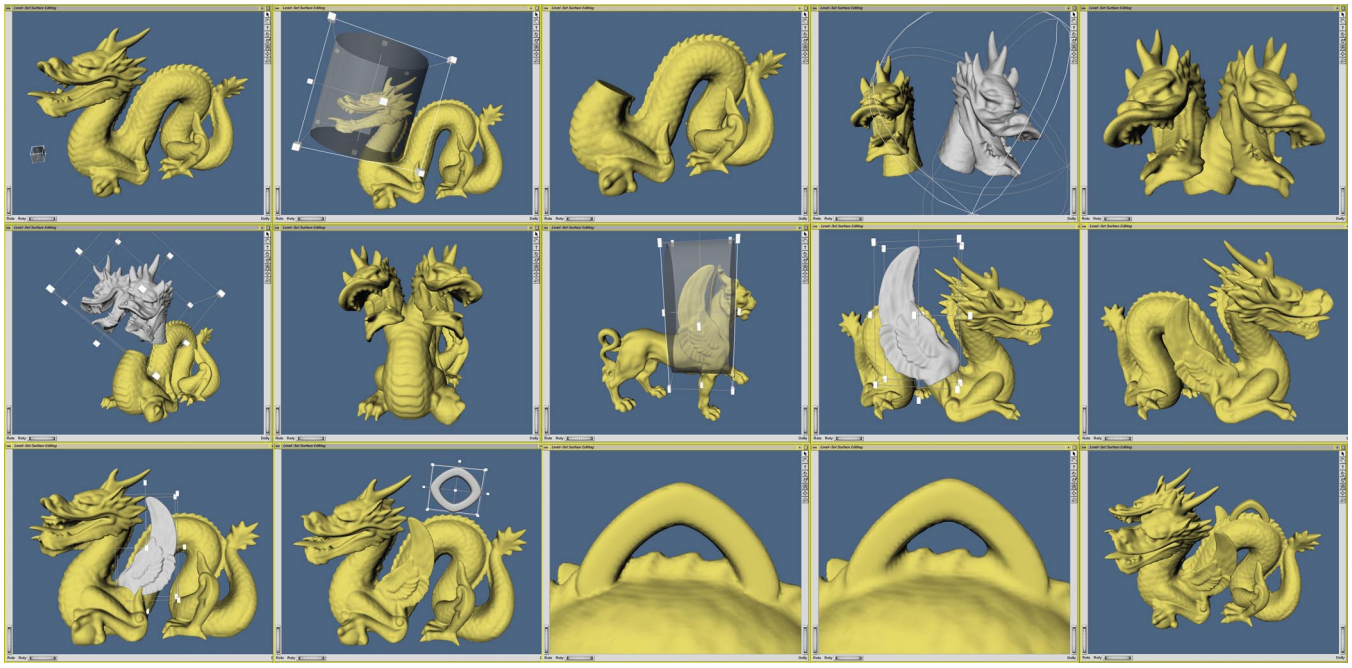


Figure 11: Series of operations used to create the winged two-headed dragon of Figure 1. First the head is cut off, pasted and blended back onto the body. Next a wing is copied from a different model and blended onto one side of the dragon. The same wing is then mirrored and blended onto the other side. Finally a scan converted supertoroid is blended onto the dragon's back to form the loop of a bracelet charm.

utilizing direct volume rendering hardware, parallelizing the level set computations, and exploring multiresolution volumetric representations will lead to editing operations that require only a fraction of a second, instead of tens of seconds.

We have presented five example level set surface editing operators. Given the generality and flexibility of our framework many more can be developed. We intend to explore operators that utilize Gaussian and principal curvature, extend embossing to work directly with lines, curves and solid objects, and ones that may be utilized for general surface manipulations, such as dragging, warping, and sweeping.

7 Acknowledgements

We would like to thank Mathieu Desbrun for his helpful suggestions, and Cici Koenig and Katrine Museth for helping us with the figures. The Greek bust and human head models were provided by Cyberware Inc. The dragon and griffin models were provided the Stanford Computer Graphics Laboratory. The teapot model was provided by the University of Utah's Geometric Design and Computation Group. This work was financially supported by National Science Foundation grants ASC-89-20219, ACI-9982273 and ACI-0083287.

References

ADALSTEINSSON, D., AND SETHIAN, J. 1995. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 269–277.

AMENTA, N., BERN, M., AND KAMVYSSELIS, M. 1998. A new voronoi-based surface reconstruction algorithm. In *Proc. SIGGRAPH '98*, 415–421.

BAJAJ, C., BERNARDINI, F., AND XU, G. 1995. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proc. SIGGRAPH '95*, 109–118.

BARR, A. 1981. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications 1*, 1, 11–23.

BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. 2001. Approximate Boolean operations on free-form solids. In *Proc. SIGGRAPH 2001*, 185–194.

BLOOMENTHAL, J., ET AL., Eds. 1997. *Introduction to Implicit Surfaces*. Morgan Kaufmann, San Francisco.

BOUGUET, J.-Y., AND PERONA, P. 1999. 3D photography using shadow in dual space geometry. *International Journal of Computer Vision 35*, 2 (Nov/Dev), 129–149.

BREEN, D., AND WHITAKER, R. 2001. A level set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics 7*, 2, 173–192.

BREEN, D., MAUCH, S., AND WHITAKER, R. 2000. 3D scan conversion of CSG models into distance, closest-point and color volumes. In *Volume Graphics*, M. Chen, A. Kaufman, and R. Yagel, Eds. Springer, London, 135–158.

COHEN, E., RIESENFELD, R., AND ELBER, G. 2001. *Geometric Modeling with Splines*. A K Peters, Natick, MA.

CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proc. SIGGRAPH '96*, 303–312.

DESBRUN, M., AND GASCUEL, M. 1995. Animating soft substances with implicit surfaces. In *Proc. SIGGRAPH 95 Conference*, 287–290.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proc. SIGGRAPH '99*, 317–324.

DO CARMO, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, NJ.

EDELSBRUNNER, H., AND MÜCKE, E. 1994. Three-dimensional alpha shapes. *ACM Trans. on Graphics 13*, 1, 43–72.

EVANS, L., AND SPRUCK, J. 1991. Motion of level sets by mean curvature, i. *Journal of Differential Geometry*, 635–681.

FOSTER, N., AND FEDKIW, R. 2001. Practical animation of liquids. In *Proc. SIGGRAPH 2001*, 23–30.

FRISKEN, S., PERRY, R., ROCKWOOD, A., AND JONES, T. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *SIGGRAPH 2000 Proceedings*, 249–254.

GALYEAN, T., AND HUGHES, J. 1991. Sculpting: An interactive volumetric modeling technique. In *Proc. SIGGRAPH '91*, 267–274.

HOFFMANN, C. 1989. *Geometric and Solid Modeling*. Morgan Kaufmann, San Francisco.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. SIGGRAPH '98*, 105–114.

KOBBELT, L. P., BOTSCH, M., SCHWANECKE, U., AND SEIDEL, H.-P. 2001. Feature sensitive surface extraction from volume data. In *Proc. SIGGRAPH 2001*, 57–66.

LAIDLAW, D., TRUMBORE, W., AND HUGHES, J. 1986. Constructive solid geometry for polyhedral objects. In *Proc. SIGGRAPH '86*, vol. 20, 161–170.

LORENSEN, W., AND CLINE, H. 1987. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH '87*, 163–169.

MALLADI, R., SETHIAN, J., AND VEMURI, B. 1995. Shape modeling with front propagation: A level set approach. *IEEE Trans. on Pattern Analysis and Machine Intelligence 17*, 2, 158–175.

MARAGOS, P. 1996. Differential morphology and image processing. *IEEE Trans. on Image Processing 5*, 6 (June), 922–937.

OSHER, S., AND FEDKIW, R. 2001. Level set methods: An overview and some reecent results. *Journal of Computational Physics 169*, 475–502.

OSHER, S., AND SETHIAN, J. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics 79*, 12–49.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H.-K., AND KANG, M. 1999. A PDE-based fast local level set method. *Journal of Computational Physics 155*, 410–438.

PERRY, R., AND FRISKEN, S. 2001. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH 2001*, 47–56.

REQUICHA, A., AND VOELCKER, H. 1985. Boolean operations in solid modeling: Boundary evaluation and merging algorithms. *Proceedings of the IEEE 73*, 1, 30–44.

RUDIN, L., OSHER, S., AND FATEMI, C. 1992. Nonlinear total variation based noise removal algorithms. *Physica D 60*, 259–268.

SAPIRO, G., KIMMEL, R., SHAKED, D., KIMIA, B., AND BRUCKSTEIN, A. 1993. Implementing continuous-scale morphology via curve evolution. *Pattern Recognition*, 9, 1363–1372.

SAPIRO, G. 2001. *Geometric Partial Differential Equations and Image Analysis*. Cambridge University Press, Cambridge, UK.

SERRA, J. 1982. *Image Analysis and Mathematical Morphology*. Academic Press, London.

SETHIAN, J. 1996. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Science*, vol. 93 of 4, 1591–1595.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*, second ed. Cambridge University Press, Cambridge, UK.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *Proc. SIGGRAPH '95*, 351–358.

TSITSIKLIS, J. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Trans. on Automatic Control 40*, 9, 1528–1538.

WANG, S., AND KAUFMAN, A. 1994. Volume-sampled 3D modeling. *IEEE Computer Graphics and Applications 14*, 5 (September), 26–32.

WANG, S., AND KAUFMAN, A. 1995. Volume sculpting. In *1995 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 151–156.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *Proc. SIGGRAPH '94*, 247–256.

WHITAKER, R., AND XUE, X. 2001. Variable-conductance, level-set curvature for image denoising. In *Proc. IEEE International Conference on Image Processing*, 142–145.

WHITAKER, R., BREEN, D., MUSETH, K., AND SONI, N. 2001. Segmentation of biological datasets using a level-set framework. In *Volume Graphics 2001*, M. Chen and A. Kaufman, Eds. Springer, Vienna, 249–263.

WHITAKER, R. 1998. A level-set approach to 3D reconstruction from range data. *Int. Jrnl. of Comp. Vision October*, 3, 203–231.

WYVILL, B., GALIN, E., AND GUY, A. 1999. Extending the CSG tree. warping, blending and Boolean operations in an implicit surface modeling system. *Computer Graphics Forum 18*, 2 (June), 149–158.

ZHAO, H.-K., OSHER, S., AND FEDKIW, R. 2001. Fast surface reconstruction using the level set method. In *Proc. 1st IEEE Workshop on Variational and Level Set Methods*, 194–202.

A Level Set Models

A deformable (*i.e.* time-dependent) surface, $\mathcal{S}(t)$, is implicitly represented as an iso-surface of a time-varying[1] scalar function, $\phi(\boldsymbol{x}, t)$, embedded in 3D, *i.e.*

$$\mathcal{S}(t) = \{\boldsymbol{x}(t) \mid \phi(\boldsymbol{x}(t), t) = k\}, \tag{11}$$

where $k \in \Re$ is the iso-value, $t \in \Re^+$ is time, and $\boldsymbol{x}(t) \in \Re^3$ is a point in space on the iso-surface. It might seem inefficient to implicitly represent a surface with a 3D scalar function; however the higher dimensionality of the representation provides one of the major advantages of the LS method: the flexible handling of changes in the topology of the deformable surface. This implies that LS surfaces can easily represent complicated surface shapes that can, form holes, split to form multiple objects, or merge with other objects to form a single structure.

The fundamental level set equation of motion for $\phi(\boldsymbol{x}(t), t)$ is derived by differentiating both sides of Eq. (11) with respect to time t, and applying the chain rule giving:

$$\frac{\partial \phi}{\partial t} = -\nabla \phi \cdot \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t}, \tag{12}$$

where $\mathrm{d}\boldsymbol{x}/\mathrm{d}t$ denotes the speed vectors of the level set surface. A number of numerical techniques by [Osher and Sethian 1988; Adalsteinsson and Sethian 1995] make the initial value problem of Eq. (12) computationally feasible. A complete discussion of the details of the level set method is beyond the scope of this paper. We instead refer the interested reader to [Sethian 1999; Osher and Fedkiw 2001]. However, we will briefly mention two of the most important techniques: the first is the so called "up-wind scheme" which addresses the problem of overshooting when trying to solve Eq. (12) by a simple finite forward difference scheme. The second is related to the fact that one is typically only interested in a single solution to Eq. (12), say the $k = 0$ level set. This implies that the evaluation of ϕ is important only in the vicinity of that level set. This forms the basis for "narrow-band" schemes [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] that solve Eq. (12) in a narrow band of voxels containing the surface. The "up-wind scheme" makes the level set method numerically robust, and the "narrow-band scheme" makes its computational complexity proportional to the level set's surface area rather than the size of the volume in which it is embedded.

B Curvature of Level Set Surfaces

The principle curvatures and principle directions are the eigenvalues and eigenvectors of the *shape matrix* [do Carmo 1976]. For an implicit surface, the shape matrix is the derivative of the normalized gradient (surface normals) projected onto the tangent plane of the surface. If we let the normals be $\boldsymbol{n} = \nabla\phi/|\nabla\phi|$, the derivative of this is the 3×3 matrix

$$\boldsymbol{N} = \left(\frac{\partial \boldsymbol{n}}{\partial x} \;\; \frac{\partial \boldsymbol{n}}{\partial y} \;\; \frac{\partial \boldsymbol{n}}{\partial z} \right)^T. \tag{13}$$

The projection of this derivative matrix onto the tangent plane gives the shape matrix [do Carmo 1976] $\boldsymbol{B} = \boldsymbol{N}(I - \boldsymbol{n} \otimes \boldsymbol{n})$, where $\otimes$ is the exterior product. The eigenvalues of the matrix $\boldsymbol{B}$ are k_1, k_2 and zero, and the eigenvectors are the principle directions and the normal, respectively.

[1] Our work uses the dynamic level set equation, which is more flexible than the corresponding stationary equation, $\phi(\boldsymbol{x}) = k(t)$, see [Sethian 1999] for more details.

Because the third eigenvalue is zero, we can compute k_1, k_2 and various differential invariants directly from the invariants of $\boldsymbol{B}$. Thus the weighted curvature flow is computing from $\boldsymbol{B}$ using the identities $D = ||\boldsymbol{B}||_2$, $H = \mathrm{Tr}(\boldsymbol{B})/2$, and $K = 2H^2 - D^2/2$. The choice of numerical methods for computing $\boldsymbol{B}$ is discussed in the following section. The principle curvature are calculated by solving the quadratic

$$k_{1,2} = H \pm \sqrt{\frac{D^2}{2} - H^2}. \tag{14}$$

In many circumstances, the curvature term, which is a kind of directional diffusion, which does not suffer from overshooting, can be computed directly from first- and second-order derivatives of ϕ using central difference schemes. However, we have found that central differences do introduce instabilities when computing flows that rely on quantities other than the mean curvature. Therefore we use the method of *differences of normals* [Rudin et al. 1992; Whitaker and Xue 2001] in lieu of central differences. The strategy is to compute normalized gradients at staggered grid points and take the difference of these staggered normals to get centrally located approximations to $\boldsymbol{N}$. The shape matrix $\boldsymbol{B}$ is computed with gradient estimates from central differences. The resulting curvatures are treated as speed functions (motion in the normal direction), and the associated gradient magnitude is computed using the up-wind scheme.