Computers & Graphics 35 (2011) 800–809

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Real-time ray casting of algebraic B-spline surfaces☆

Feifei Wei, Jieqing Feng *

State Key Laboratory of CAD&CG, Zhejiang University, 310058, China

ARTICLE INFO

Article history:
Received 25 October 2010
Received in revised form
10 April 2011
Accepted 19 April 2011
Available online 10 May 2011

Keywords:
Piecewise algebraic surface
B-spline
Newton–Raphson method
Regula falsi
Polygonization
Polar surface
Silhouette

ABSTRACT

Piecewise algebraic B-spline surfaces (ABS surfaces) are capable of modeling globally smooth shapes of arbitrary topology. These can be potentially applied in geometric modeling, scientific visualization, computer animation and mathematical illustration. However, real-time ray casting the surface is still an obstacle for interactive applications, due to the large amount of numerical root findings of nonlinear polynomial systems that are required. In this paper, we present a GPU-based real-time ray casting method for ABS surfaces. To explore the powerful parallel computing capacity of contemporary GPUs, we adopt iterative numerical root-finding algorithms, e.g., the Newton–Raphson and regula falsi algorithms, rather than recursive ones. To facilitate convergence of the Newton–Raphson or regula falsi algorithm, their initial guesses are determined through rasterization of the isotopic isosurface, and the isosurface is generated based on regular criteria for surface domain subdivision. Meanwhile, polar surfaces are adopted to identify single roots or to isolate different roots, i.e., ray and surface intersections. As an important geometric feature, the silhouette curve is elaborately computed to floating-point accuracy, which can be applied in further anti-aliasing processes. The experimental results show that the proposed method can render thousands of piecewise algebraic surface patches of degrees 6–9 in real time.

1. Introduction

Because of its advantages in intersection, blending, offset, and arbitrary topology, the algebraic polynomial representation is an alternative to parametric surfaces in geometric modeling. The piecewise algebraic surface in terms of B-splines, which is abbreviated as ABS in this paper, was first introduced by Patrikalakis and Kriezis [23]. The ABS surface is capable of modeling globally smooth shapes of arbitrary topology without explicitly specifying smoothing constraints as in A-patch [3] or NURBS surface [6] modeling. Compared to subdivision surfaces, which can also model globally smooth shapes of arbitrary topology, the ABS surface has an analytical expression, which is important to perform geometry computations and shape interrogation in geometric modeling, e.g., evaluations of normal, curvatures, offset, intersection. Jüttler and Felis [10], and Tong et al. [29] present ABS surface fitting algorithms for reconstruction of scatter data. However, shape modeling via ABS surfaces is beyond the scope of this paper.

Unlike its parametric counterpart, high quality ABS surface display is not trivial. The ray casting approach can reveal elaborated surface topology and geometric details, thus, it is free of sampling artifacts in pixel accuracy. Furthermore, only polynomial coefficients of the surface are required to use a ray casting algorithm, which leads to lower memory and bus bandwidth consumption than does the use of prevalent triangle rendering pipelines. However, the ray casting approach involves a large number of ray and surface intersections. As far as we know, there are no real-time ray casting methods for ABS surfaces.

In this paper, we present a GPU-based real-time ray casting method for ABS surfaces via iterative root-finding algorithms, e.g., the Newton–Raphson and regula falsi. The initial guess of the root-finding algorithm is first given via rasterization of an isosurface obtained from an isotopic ABS surface polygonization. The first root is isolated by analyzing configurations between ABS surfaces and their polar surfaces. Finally, the Newton–Raphson algorithm or regula falsi algorithm is used to compute the first hit point. Meanwhile, the polar surface is also employed to compute silhouette points to floating-point accuracy. And the silhouette points can be applied for further anti-aliasing. The hit point is computed in object space, and thus the overhead of functional composition between the ray and surface is avoided. Furthermore, the surface normal for shading is a byproduct of the iterative root-finding algorithm. The proposed method can display an ABS surface of degrees 6–9, which is composed of thousands of piecewise patches, in real time.

The contributions of the proposed method are summarized as follows:

- real-time ray casting of ABS surfaces on GPU;
- rather than solving a univariate polynomial equation, we iteratively solve a nonlinear tri-variate polynomial system in

☆This article was recommended for publication by D. Gutierrez.
* Corresponding author.
E-mail addresses: weifeifeicad@gmail.com (F. Wei), jqfeng@cad.zju.edu.cn (J. Feng).

0097-8493/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2011.04.002

object space to find the first hit point, where an initial guess is determined via isotopic polygonization;

- roots isolation by using polar surfaces;
- explicitly computing the silhouette and anti-aliasing along the silhouette.

In the following, we first review related work in Section 2. Then we describe the ABS surface in Section 3 and our algorithm in Section 4. Isotopic polygonization and applications of the polar surface are described in Section 4.2, 4.3 and 4.4, respectively. Details of our implementation are given in Section 5. Numerical examples, comparison and discussion are given in 6. Finally we draw the conclusion and propose future work.

2. Related work

Currently, there are two kinds of methods used to display an algebraic surface: sampling it and displaying proxy geometries, e.g., triangles or particles and directly ray tracing or ray casting it. Displaying an algebraic surface via discrete proxy geometries is more suitable for prevalent triangle rendering pipelines. With the rapid development of graphics hardware, parallelism of triangle rendering pipelines is exploited thoroughly. Even for very high throughput, the state of the art rendering pipelines are able to produce stunningly realistic synthesis images in real time. The prevalent approaches to approximate or sample an algebraic surface are Wyvill's polygonization [31], marching cubes [17], Bloomenthal's polygonization [5] and the particle system approach [30]. However, sampling artifacts are the main flaw of these approaches. Level-of-detail techniques can be employed to alleviate the artifacts to some extent, which will cause additional overhead. A comparison between ray casting and polygonization is shown in Fig. 2. The polygonization resolution of the left surface is up to 200^3, however, it is still plagued by sampling artifacts.

Although ray casting can address the above problems, it usually cannot meet the demands of real-time applications. Its most time-consuming procedure is to find the first hit point between a ray and the surface. In general, this is a root-finding problem of polynomials. Analytical solutions exist for polynomials of degree 4 or lower. According to this observation and some fundamental work in [4], Loop and Blinn [16] proposed a real-time ray casting algorithm for A-patches up to degree 4. However, it is not trivial to apply this approach to render ABS surfaces directly since the configurations of a cube projection on the screen plane are more complex than those of a tetrahedron. It is also not efficient to process overlapped domain cubes with complex depth relationships.

In the past decades, many numerical root-finding algorithms have been employed to ray cast parametric or algebraic surfaces of degrees above 4. Kajiya [11] reduced ray tracing of parametric surfaces to root finding of a high degree polynomial using Laguerre's method. Hanrahan [8] implemented ray tracing of algebraic surfaces by using the Descartes rule of signs for root isolation and Newton's method for root refinement. Kalra and Barr [12] employed a Lipschitz constant and surface gradient to guarantee correct ray and surface intersection. Interval arithmetic constructs a convex hull around a function over a given domain. By using this property, Mitchell [19] isolated the roots using repeated interval bisection until an interval contains a single root. Bézier clipping, proposed by Nishita et al. [21], finds the roots of a Bernstein polynomial equation based on the convex hull property. Recently, Mørken and Reimers [20] presented an unconditionally convergent root-finding method via knot insertion for splines.

Utilizing the GPU, it is possible to achieve real-time ray tracing of algebraic surfaces. Pabst et al. [22] and Kanamori et al. [13] implemented an iterative version of the Bézier clipping with a fixed-size stack on GPU for ray casting trimmed NURBS surfaces and metaballs, respectively. Knoll et al. proposed an interval bisection algorithm for real-time ray casting general algebraic surfaces via SSE instruction-level optimization [15] and GPU [14], respectively. Both an interval arithmetic and the Bézier clipping require recursive subdivision until a tolerance is satisfied. Interval evaluation of the Bernstein polynomials is numerically instable due to large numbers of MAD (addition and multiplication) operations on GPU. Meanwhile, rounded interval arithmetic is conservative [18], but is inefficient on GPU. Reimers and Seland [25] performed a functional composition between ray and surface via polynomial interpolation and employ B-spline knot insertion for root-finding. The root-finding algorithm is iterative and more suitable for the SIMD architecture of GPU than the Bézier clipping approach. However, functional composition is an additional overhead. Singh and Narayanan [27] proposed an adaptive marching point method to find the first hit point, wherein the step size is determined by distance from the surface and its closeness to the silhouette. Although it is very efficient on GPU, root isolation via rule of signs is not always robust. Furthermore, threshold estimations of step size are not a trivial work for piecewise surfaces, since they are patch dependent.

3. Algebraic B-spline surfaces

Let $\mathbf{X}=[x_0,x_1,\ldots,x_{M+m+1}]$, $\mathbf{Y}=[y_0,y_1,\ldots,y_{N+n+1}]$ and $\mathbf{Z}=[z_0,z_1,\ldots,z_{Q+q+1}]$ be three non-decreasing knot vectors along the x, y and z directions. An ABS surface is defined as

$$F_{ABS}(x,y,z)=\sum_{i=0}^{M}\sum_{j=0}^{N}\sum_{k=0}^{Q} w_{ijk}N_i^m(x)N_j^n(y)N_k^q(z)=0 \tag{1}$$

where $N_i^m(x)$,$N_j^n(y)$ and $N_k^q(z)$ are the B-spline basis functions of degrees m, n and q, defined by the knot vectors $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$. The scalars $\{w_{ijk}\}$ are weight coefficients. They have similar functions to control points in their parametric counterpart and can be used to adjust the shape of the ABS surface. In this paper, the degree of the ABS surface along each direction is no greater than 3. They can model the C^0, C^1 or C^2 smooth shapes, which means that the shape has common position, tangent and curvature values along the boundaries of the patches, respectively. In general, they can meet the requirements in the practical applications, e.g., C^0 or C^1 smooth shapes for computer animation and computer games, and C^1 or C^2 shapes for product design. By using the knot insertion algorithm of B-splines, the ABS surface defined on a knot box $[x_i,x_{i+1}]\times[y_j,y_{j+1}]\times[z_k,z_{k+1}]$ $(m\le i\le M, n\le j\le N, q\le k\le Q)$ can be converted into an algebraic surface patch in terms of tensor-product Bernstein polynomials:

$$F(x,y,z)=\sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{q} f_{ijk}B_i^m(u(x))B_j^n(v(y))B_k^q(w(z))=0 \tag{2}$$

where

$$u(x)=\frac{x-x_i}{x_{i+1}-x_i},\quad v(y)=\frac{y-y_j}{y_{j+1}-y_j},\quad w(z)=\frac{z-z_k}{z_{k+1}-z_k}$$

f_{ijk} are coefficients similar to w_{ijk} of ABS surfaces, and $B_i^m(u)$, $B_j^n(v)$ and $B_k^q(w)$ are the Bernstein basis functions of degrees m, n and q. As a result, we can obtain $(M-m+1)\times(N-n+1)\times(Q-q+1)$ piecewise algebraic surfaces in the Bernstein form.

4. Rendering algorithm

4.1. Algorithm overview

In general, ray and surface intersection can be reduced to a root-finding problem of a univariate polynomial equation via functional composition. Functional composition between an ABS surface and a ray is a complex procedure [7], and is impractical for the SIMD architecture. In our setting, ray and surface intersection is performed in object space as a root-finding problem of a tri-variate nonlinear system as follows:

$$\begin{cases} F(x,y,z) & =0 \\ a_h x + b_h y + c_h z + d_h & =0 \\ a_v x + b_v y + c_v z + d_v & =0 \end{cases} \tag{3}$$

where F is a piecewise tensor-product Bernstein polynomial described in Eq. (2), and the two linear equations are the horizontal and vertical scan planes that pass both the viewpoint and current pixel, respectively. Reimers and Seland adopted functional composition to reduce the ray and the algebraic surface intersection to a root-finding problem of univariate polynomial [25], which may contain many evaluations of the algebraic surface. In our approach, Eq. (3) is solved in the object space directly. Thus additional computational costs for functional compositions are saved.

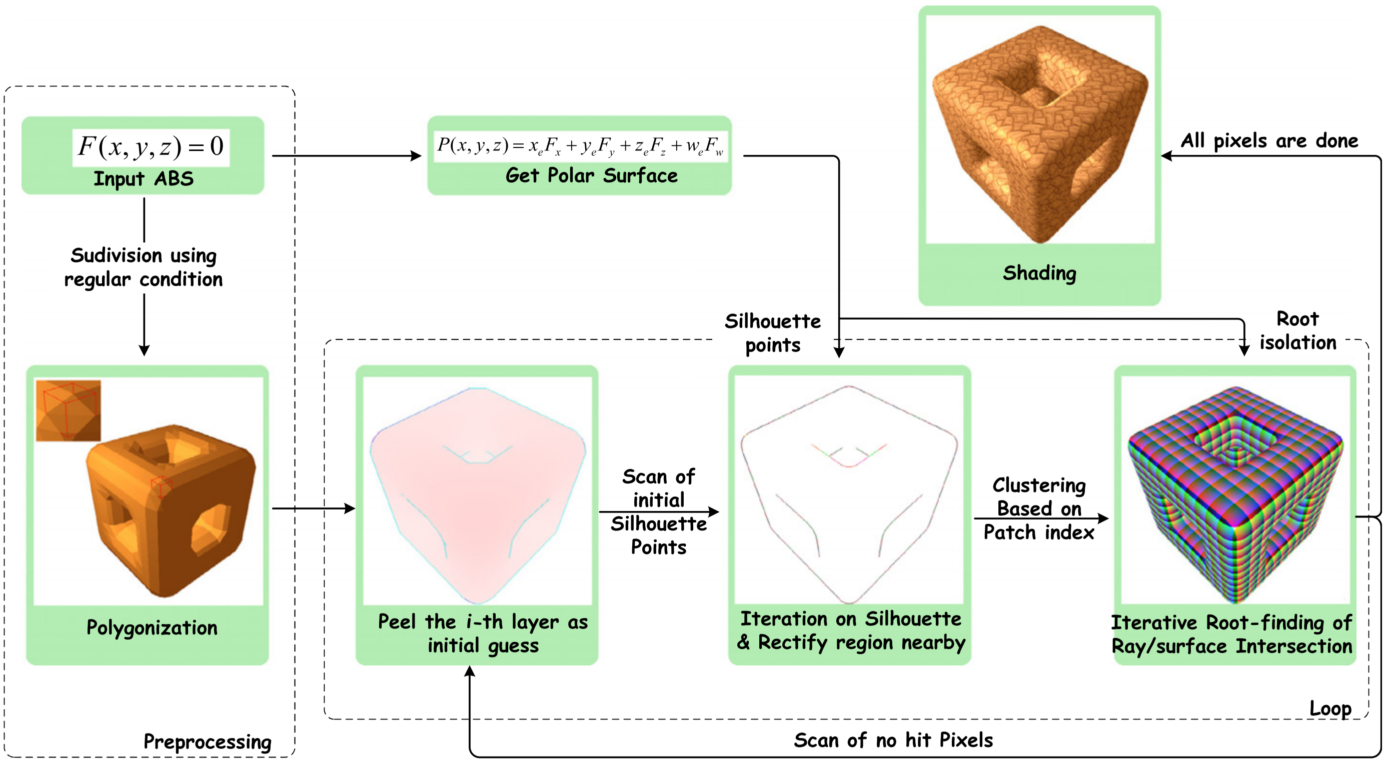
The nonlinear system (3) can be solved via the Newton–Raphson method, which can converge quadratically if a good initial guess is given. A rough initial guess can be obtained via rasterization of an isosurface of the underlying ABS surface, which is an isotopic polygonization as stated in Section 4.2. However, the initial guess cannot guarantee convergence of the root finding algorithm near the silhouette due to ill-conditioned computations. To overcome this problem, the silhouette, which is the intersection between the underlying ABS surface and its polar surface, is processed specially and robustly. For regions near the silhouette, the polar surface is employed again to isolate an interval containing the first hit point. The regula falsi method is adopted to find the first hit point in this interval. For other regions, the Newton–Raphson method can achieve quadratic convergence robustly. The flow chart for the proposed method is shown in Fig. 1, and its description is given in Algorithm 1.

Algorithm 1.

```
1.  Input: ABS surface
2.  Isotopic polygonization of ABS surface;
3.  For each patch of ABS surface, check existence of its
    polar surface via signs of control coefficients;
4.  repeat
5.    Peel $i$-th layer of visible isosurface rendering result as
      initial guess;
6.    Compute silhouette points and rectify the initial
      guess near the silhouette;
7.    Scan the initial guess and Sort them by patch index
      to generate pixel clusters;
8.    for each pixel cluster do
9.      if there is no polar surface then
10.       Compute first hit point between ray and patch via
          the Newton–Raphson method by using the initial
          guess;
11.       Record surface normal in last iteration;
12.     else
13.       Compute first hit point between ray and polar
          surface by using the Newton–Raphson method;
14.       Compute first hit point between ray and patch via
          regula falsi method in the interval bracketed by ray
          entry and polar surface;
15.       Calculate surface normal;
16.     end if
17.   end for
18. until No intersection is available.
19. Shading and anti-aliasing.
```

4.2. Isotopic polygonization

Topologically consistent polygonization of an implicit surface is not trivial. Several algorithms have been proposed to address this



$F(x,y,z)=0$
Input ABS
Sudivision using regular condition
Polygonization
Preprocessing
$P(x,y,z)=x_eF_x+y_eF_y+z_eF_z+w_eF_w$
Get Polar Surface
Shading
All pixels are done
Silhouette points
Root isolation
Peel the i-th layer as initial guess
Scan of initial Silhouette Points
Iteration on Silhouette & Rectify region nearby
Clustering Based on Patch index
Iterative Root-finding of Ray/surface Intersection
Loop
Scan of no hit Pixels

Fig. 1. Flow chart of real-time rendering of ABS surfaces.

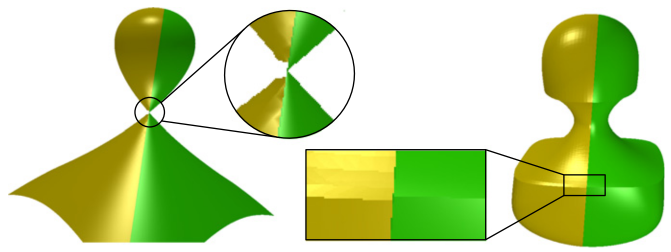


Fig. 2. Comparison between our methods and polygonization of the surface in Fig. 6. The yellow parts are rendered by polygonization, while green by our method. The examples illustrate that the ray casting images can always reveal the details of the ABS surface more elaborately, such as the singular point and the sharp edge, while the polygonization may miss these fine features. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

problem, which are based on the Morse theory, stratification theory, etc. [28,1,2]. Because the ABS is transformed to the Bernstein basis representations in rectangular grids as in Eq. (2), and Alberti et al. [1] proposed a regular criteria according to a generalization of Descartes' rule that can guarantee that the resulting mesh is isotopic to the algebraic surfaces. We keep subdividing the representation until the surface in each box boils down to the case where the implicit object is isotopic to its linear approximation in the cell. Thus, we can estimate the maximum subdivision resolution of the ABS surfaces conservatively by adopting the maximum resolution among all the algebraic patches. Then the ABS surface is polygonized in parallel on the GPU according to this resolution. Due to the tensor-product property of the ABS surface and its low degree (less than 4 along each direction), we augment the polygonization by use of accurate vertices in the isosurface. The accurate vertex, i.e., the intersection between the ABS surface and voxel edge, can be analytically evaluated by using Blinn's efficient GPU algorithm [4]. The additional cost of augmented polygonization is less than 10 ms in general, according to our implementations. However, accuracy improvement in polygonization will be helpful to enhance the performance of ray and surface intersection computation.

As indicated by Alberti et al. [1], the above isotopic polygonization is confined to a smooth surface with finite singular points, where a singular point is defined as $f = f_x = f_y = f_z = 0$. Thus we also assume that the ABS surface has finite singular points in this paper. The singular points can be found robustly via the projected polyhedron algorithm [24]. The isosurface near a singular point is described as a cone-like shape [1]. We suppose that the ABS surfaces are potentially more suitable to model smooth shapes, rather than shapes with sharp features such as infinite singular points or self-intersections.

For the same viewpoint as that of the following recasting, the isosurface is first sent to the traditional polygonal rendering pipelines. Then, a rasterized image together with its z-buffer is obtained. Each z-value in the z-buffer together with its corresponding pixel (x, y) is coalesced as a coarse initial guess for the following root-finding iteration in the raycasting.

4.3. Silhouette and polar surfaces

The silhouette plays an important role in the high quality display of an algebraic surface. Sederberg and Zundel [26] first proposed the concept of the polar surface to compute the silhouette. Its definition is described in homogenous coordinate space. For an algebraic surface $F(x,y,z,w) = 0$ of degree n in homogeneous coordinate space, its polar surface $P(x,y,z,w)$ with respect to viewpoint $E(x_e,y_e,z_e,w_e)$ is of degree $n-1$ and is defined as

$$P(x,y,z,w) = x_eF_x + y_eF_y + z_eF_z + w_eF_w = 0 \qquad (4)$$

At this point the polar surface is rewritten in the Euclidean space by setting $w = w_e$. If the projection is orthogonal, $w_e = 0$; if perspective, $w_e = 1$. For the sake of convenience, the polar surface is still noted as $P(x,y,z) = 0$. The degree of the ABS surface in Eq. (1) is $(m+n+q-1)$. To facilitate the following computations on GPU, its degree is elevated to $m \times n \times q$. The silhouette curves with respect to a viewpoint is defined as the intersection between the ABS surface and its polar surface [26]. We then perform scan conversion of silhouette points horizontally or vertically according to the following system:

$$\begin{cases} F(x,y,z) & = 0 \\ P(x,y,z) & = 0 \\ a_kx + b_ky + c_kz + d_k & = 0, \quad k = v \text{ or } h \end{cases} \qquad (5)$$

where the third linear equation represents the scan plane.

Eq. (5) is solved via the Newton–Raphson method on GPU as indicated in the sixth step of Algorithm 1. The silhouette of the isosurface can be adopted as its initial guess, namely the silhouette initial guess, via edge detection of its rasterized image. To perform a well-conditioned iteration, the horizontal or vertical scan plane should be selected carefully in Eq. (5). The gradient vector of $F(x,y,z)$ at the silhouette initial guess is projected onto the screen plane, which is noted as (n_x,n_y). If $|n_x| \approx |n_y|$, both the horizontal plane and vertical scan plane should be selected as shown in Fig. 3. There are two silhouette points obtained, which is important for the convex silhouette. If $|n_x| > |n_y|$, the vertical scan plane should be selected; otherwise, the horizontal scan plane should be chosen. If a silhouette curve segment is convex, then there will be a gap region between the isosurface silhouette and surface silhouette. In this gap region, there is no initial guess definition for ray and surface intersection. The missed initial guess can be obtained via linear interpolation of the initial guess at the isosurface silhouette and the actual silhouette point.

4.4. Root isolation via polar surface

To compute the intersection between a ray and the ABS surface, we solve Eq. (3) via the Newton–Raphson method by using the initial guess, which is obtained from isosurface rasterization in Section 4.2 or linear interpolation in Section 4.3. However, near the silhouette, a ray and the ABS surface tends to intersect at two points P_1 and P_2, as shown in Fig. 4. Thus the Newton–Raphson iterations with the initial guess P_m may not

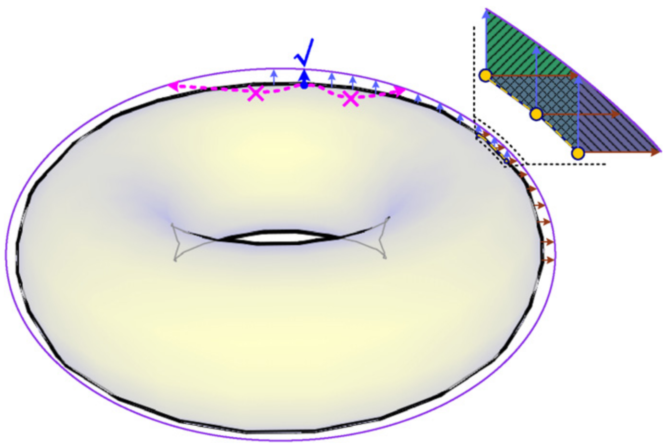


Fig. 3. Illustration of classifications and iteration of silhouette computation. The horizontal and vertical arrows indicate the horizontal and vertical scan planes. The initial guesses in the gaps near the silhouette are obtained via linear interpolation.

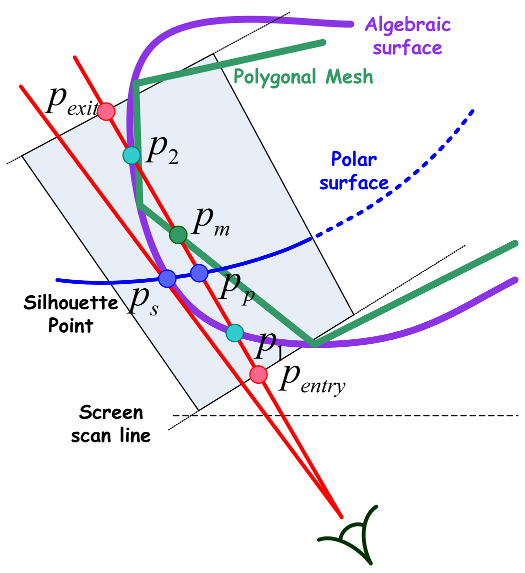


Fig. 4. Root isolation using the polar surface near the silhouette.

converge to the first hit point P_1. To address this problem, we first isolate the first hit point by using the polar surface. Then, the regular falsi method is employed to compute the first hit point. The details of root isolation via polar surfaces are provided in Appendix. Here we give two corollaries of the polar surface:

Corollary 1. *If the polar surface* $\{(x,y,z) : P(x,y,z)=0\}$ *is null, the ray will intersect the surface at one point at most, i.e., Eq.* (3) *has at most one root.*

Corollary 2. *The intersection point between the ray and the polar surface can isolate two adjacent intersection points between the same ray and the corresponding algebraic surface.*

As stated in Algorithm 1, two iterative root-finding methods are adopted: the Newton–Raphson method and a two-phases method (root isolation plus regula falsi). According to Corollary 1, if there is no polar surface defined in $\Re^3$, a ray and the ABS surface will intersect in one point at most. In this case, the Newton–Raphson method with a rough initial guess will converge quadratically. Otherwise, the initial guess will be close to the silhouette as shown in Fig. 4, and the Newton–Raphson iterations will be ill-conditioned due to a near singularity of the Jacobian matrix of Eq. (3). To overcome this problem, we isolate the first hit point P_1 of the ray and the ABS surface by using the polar surface according to Corollary 2. Because the ray tends to be orthogonal to the polar surface near the silhouette, we can compute the ray and the polar surface intersection p_p by using the Newton–Raphson method. Let the ray and the patch domain intersect at two points, i.e., p_{entry} and p_{exit}. The interval $[p_{entry},p_p]$ or $[p_p,p_{exit}]$ is selected as the initial interval for the regula falsi method, in which $F(x,y,z)$ has opposite signs at the interval ends. The regula falsi method is robust, as the first root has been bracketed, even for the pixels near a singular point. An example is shown in Fig. 5.

4.5. Shading and anti-aliasing

The surface normal is important for illumination calculations. In functional-composition-based ray casting approaches, the surface normal is computed via an additional procedure, either analytically or approximately. The root-finding procedure in our approach is performed in object space thoroughly, and the extra overhead of normal computation can be avoided. For example, in the Newton–Raphson iteration, the surface normal ∇F is computed in each step. The ∇F computed in the final iteration can be adopted as the surface normal.

In Section 4.3, the silhouette points have been computed to floating-point accuracy. In screen space, the slope at each silhouette point can be obtained via projection of the normal. In this way, each silhouetting pixel can be anti-aliased by using a percentage coverage filtering methodology.

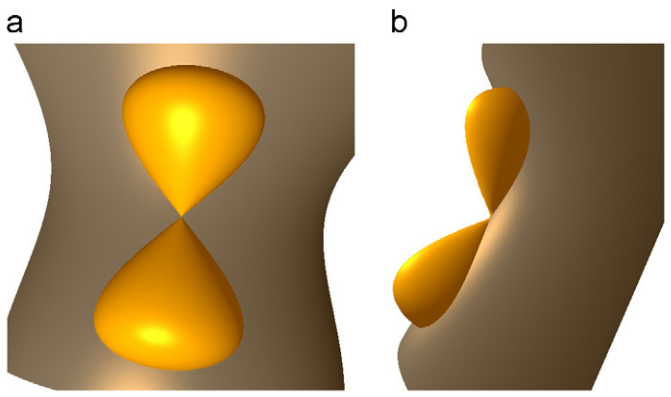


Fig. 5. An hourglass shape surface with one singular point and its polar surface. The polar surface isolates two intersection points of the ray and surface: (a) front view and (b) side view.

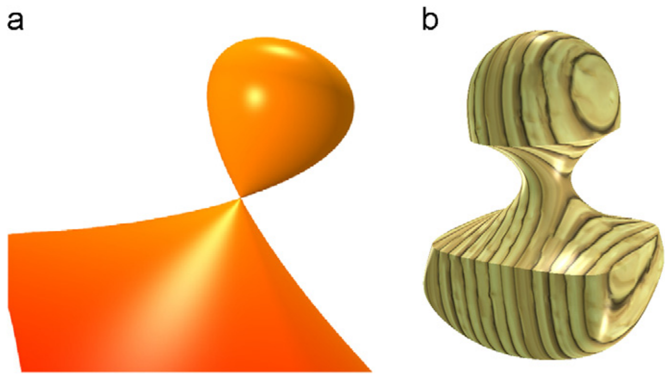


Fig. 6. (a) is an ABS patch of degree 7 with a singular point constructed from the "Dingdong" surface. The sharp edge of (b) is constructed via multiple knots: (a) 1 patch, G^0, 57.62 fps and (b) 3 patches, G^0, 97.65 fps.

5. Implementation on GPU

The proposed algorithm is designed for GPU with SIMD architecture. Except for isosurface rendering via a traditional graphics pipeline, the proposed algorithm is implemented completely in NVIDIA CUDA. CUDA provides a C language interface for general-purpose programming on the GPU. It exposes many important parallel computing capabilities of the GPU so as to facilitate data-parallel computations, e.g., an efficient scatter-gather mechanism. The common operations, such as scan, reduce and sort, have been released in CUDPP [9], which can be reused conveniently as shown in Algorithm 1.

The major difference between the rendering of single algebraic surfaces and ABS surfaces is that ABS surfaces have a large number of control coefficients. The manner in which the piecewise surface patch data is accessed seriously affects the performance of the rendering algorithm.

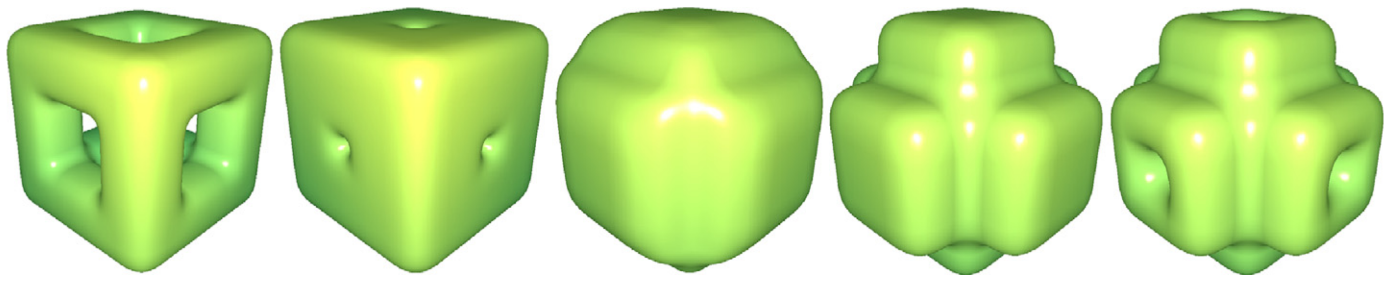


Fig. 7. Morphing between two ABS surfaces of degree 6 with different topologies. The morphing is generated by interpolating the control coefficients of the source and destination surfaces on-the-fly. This can be achieved at approximately 29–35 fps with a 1024×768 resolution.

Table 1
The ABS surfaces information and a comparison of different methods, where "segments" is the number of valid domains of the ABS surface when transformed into the Bernstein form as in Eq. (2), i.e., $(M-m+1)*(N-n+1)*(Q-q+1)$, "#(patches)" is the number of non-empty algebraic patches in terms of the Bernstein basis, "sub" refers to the subdivision resolution of the isotopic polygonization, "acc(ms)" and "lerp(ms)" represent the polygonization run time, where the vertices of the mesh are computed accurately or linearly approximated, respectively.

Figure	Degree	Segments	#(Patches)	Polygonization			Frame rate (fps)			Speedup	
				sub	acc(ms)	lerp(ms)	KSN08	RS08	Ours	$\frac{\text{Ours}}{\text{KSN08}}$	$\frac{\text{Ours}}{\text{RS08}}$
6 (a)	$2\times2\times3$	$1\times1\times1$	1	16	0.67	0.28	11.59	13.10	57.62	4.97	4.39
6(b)	$2\times2\times2$	$1\times1\times3$	3	4	0.94	0.54	23.62	27.44	97.65	4.13	3.55
10(a)	$2\times2\times2$	$17\times17\times17$	1636	4	6.96	1.82	10.66	11.54	49.57	4.65	4.29
10(b)	$2\times2\times2$	$13\times9\times9$	566	4	3.20	0.79	13.80	14.71	55.32	4.00	3.76
10(c)	$2\times2\times2$	$16\times16\times16$	2188	8	7.95	2.20	7.88	8.40	42.07	5.33	5.00
10(d)	$2\times2\times2$	$13\times13\times5$	248	4	0.98	0.80	14.03	14.95	60.05	4.27	4.01
10(e)	$2\times2\times2$	$14\times24\times24$	1185	4	6.88	2.32	28.26	32.19	86.17	3.04	2.67
10(f)	$3\times3\times3$	$4\times18\times18$	648	4	5.87	0.88	11.12	12.67	50.77	4.56	4.00
11(a)	$2\times2\times2$	$19\times15\times15$	1145	4	6.39	1.71	18.13	19.67	70.50	3.88	3.58
11(b)	$2\times2\times2$	$9\times20\times20$	1376	1	4.66	1.32	23.69	25.11	78.07	3.29	3.10
11(c)	$2\times2\times2$	$37\times38\times38$	3622	1	6.25	1.35	16.78	17.90	49.75	2.96	2.77
11(d)	$2\times2\times2$	$23\times19\times19$	1063	1	3.97	1.13	28.69	30.80	92.35	3.21	2.99
11(e)	$2\times2\times2$	$18\times18\times18$	937	4	7.52	2.27	16.66	18.20	66.47	3.98	3.65
11(f)	$2\times2\times2$	$19\times19\times19$	1494	4	7.27	1.81	13.13	14.08	56.17	4.27	3.98

According to the specification of CUDA, allocating more threads per block is better for efficient time slicing. However, the more threads per block, the fewer registers are available per thread due to limited register resources on GPU. As a result, the coefficients of each surface patch and its polar surface could cause high memory throughput. If we implement the Newton–Raphson method trivially, it will load more coefficients when the iterations jump from one patch onto its neighbor. Thus it is impractical to load surface patch data into registers for every pixel. On the other hand, the simultaneous memory access of adjacent pixels could not be coalesced into a single memory transaction, which would result in a performance penalty.

To address this problem, we propose a patch index-based clustering scheme. First, each initial guess is associated with a patch index according to its position. We sort and cluster the initial guesses by their associated patch index. Then we launch these clusters in blocks. In this way, threads in each block can share the same patch data, which can be loaded into the on-chip shared memory. This scheme provides efficient memory access as fast as registers, meanwhile limited registers are saved and could be applied to deeper parallelism.

During the implementation of the Newton–Raphson or regula falsi method, the iteration could cross the boundary between two patches. In this case, we still use current patch data via surface patch extrapolation. After the iteration terminates, we scan the pixels and cluster those that actually converge to the adjacent patch. Then we refine these with additional iterations. Such pixels are typically found near the patch boundary, and thus the pixel number is limited in general. The additional iterations can also be performed efficiently even if we read the patch data through texture memory fetching. The operations of "scan" and "sort" are both implemented by using the CUDPP library on GPU.

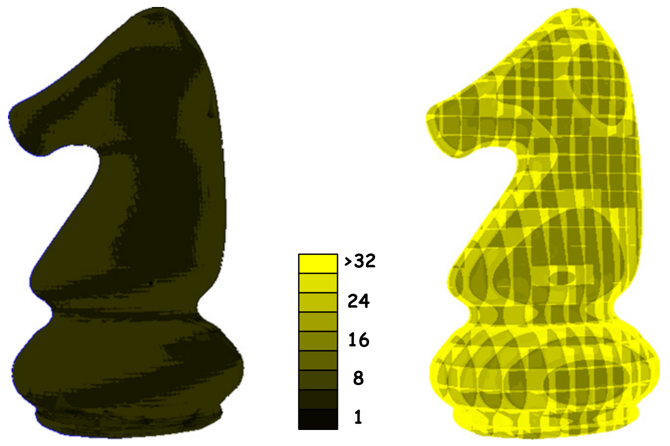


Fig. 8. The numbers of function evaluations in our method on the left and in the **RS08** method on the right, while the **RS08** and **KSN08** methods have similar convergence properties.

6. Results and comparison

The algorithm has been implemented on a PC with an Intel Core2 Q9550 2.83 GHz CPU and an NVIDIA GeForce GTX 570 GPU. The rendering resolution is 1024×768, and the ABS surfaces roughly fill the screen. The termination criteria in the root-finding of Eq. (3) is that the value of $|F(x,y,z)|$ or the iterative step of consecutive iterations is less than 1.0×10^{-6}, which is the threshold of single precision floating-point arithmetic.

We constructed several examples using the ABS surfaces. The shapes in Figs. 5 and 6(a) both have one singular point, while Fig. 6(b) has sharp edges. Fig. 10(f) is C^2 smooth, and all the others are C^1 smooth in all directions. Their topologies vary, e.g., genus 0 in Fig. 10(e), genus 1 in Figs. 10(b) and (d), genus 5 in Fig. 10(f) and even genus 12 in Fig. 10(c). Some of these are regular shapes, as the torus and revolution surface in Fig. 11(f); some are free-form shapes, e.g., bunny and knot in Fig. 11. From these examples, we conclude that the ABS surface can not only represent globally smooth shapes, but also able to model shapes with sharp features. However, the modeling of the ABS surfaces goes beyond the scope of the paper and is omitted here.

Because the algebraic surface is capable of processing a dynamic topology, an example of topological morphing between two ABS surfaces is given in Fig. 7, where the source and the destination ABS surfaces have genus 0 and 5, respectively. Because the proposed algorithm is purely on-the-fly, it can be applied to render dynamic ABS surfaces and explore the topological changes.

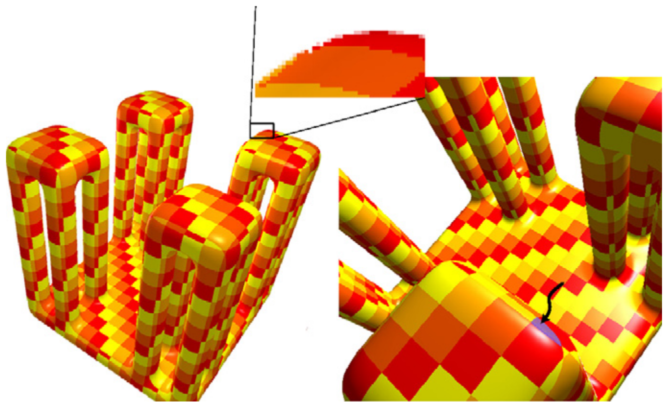


Fig. 9. Artifacts caused by ill-conditioned computations.

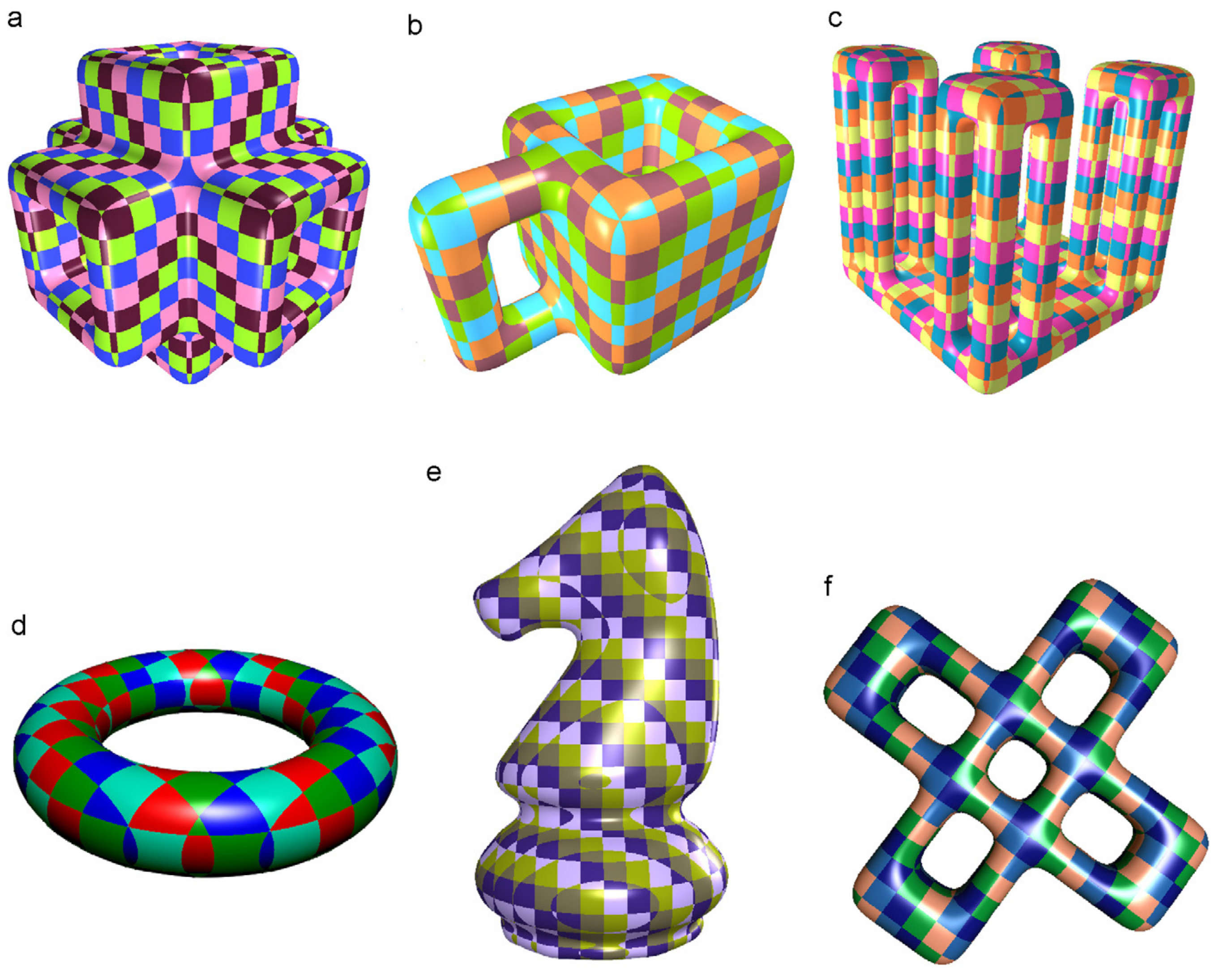


Fig. 10. Rendering results. (f) is globally C^2 smooth, and all others are C^1 smooth: (a) 1636 patches, C^1, 49.57 fps, (b) 566 patches, C^1, 55.32 fps, (c) 2188 patches, C^1, 42.07 fps, (d) 248 patches, C^1, 60.05 fps, (e) 1185 patches, C^1, 86.17 fps and (f) 648 patches, C^2, 50.77 fps.

Information regarding these ABS surfaces, the subdivision resolution of the isotopic polygonization, running time statistics and comparisons are listed in Table 1. In our algorithm, only the maximum subdivision resolution and singular point detection in the isotopic polygonization are evaluated by the CPU. The successive polygonization is computed once or on-the-fly for the static or dynamic ABS surface on GPU, respectively. All remaining parts of the algorithm are performed on-the-fly on GPU.

We implemented the **KSN08** [13] method and the **RS08** [25] method as a comparison. The rendering times and the speedup of our methods are shown is Table 1. The **RS08** method is extended to render the ABS surface by adopting depth peeling as in **KSN08**. According to our experiments, the Bézier clipping method [13] is slightly slower than the knot insertion method [25] due to an increase in the number of operations in each step. Aside from root-finding, the functional composition itself will consume many function evaluations of the ABS surface F, the surface normal ∇F and the univariate polynomial. These are the most time-consuming parts in both our method and functional composition methods. The color maps in Fig. 8 show the number of these function evaluations in their methods and ours before a hitting point is determined for an ABS surface of degree 6. Their methods involve many more function evaluations than ours. As a result, our algorithm achieves a 3–5-fold speedup as shown in Table 1. Even when runtime of the isotopic polygonization in the preprocess step is taken into account, our method is faster than the functional composition methods. Two key reasons for the speedup are the initial guesses provided by the rasterization of the isosurface and the root isolation by adopting polar surfaces, and both of these can save us from heavy overhead functional composition for piecewise algebraic surfaces, because the functional composition contains large number of the function evaluations.

7. Conclusion and future work

In this paper, we presented a GPU-based real-time ABS surface rendering algorithm, which performs ray and surface intersection computations in object space. According to regular termination criteria of domain subdivision of an ABS surface, an enhanced polygonization is performed on GPU, where the isosurface has accurate vertices and is isotopic to the ABS surface. The rasterization result of the isosurface via traditional rendering pipelines provides a rough initial guess for numerically solving for the ray and ABS surface intersection. The silhouette, defined as the intersection between the ABS surface and its polar surface is scan-converted in floating-point accuracy via the Newton–Raphson method. Then, the initial guess in the gap region between the ABS surface and isosurface can be obtained via linear interpolation. The polar surface is also employed to isolate the root near the silhouette so as to improve the robustness of the root-finding procedure. Apart from the silhouette and its vicinity, most of the hit points between the ray and ABS surface can be solved via the quadratically convergent Newton–Raphson method. Furthermore, the surface normal that is important during illumination computation is the byproduct of the Newton–Raphson method. We also design an efficient and flexible data access to facilitate the performance on GPU via the programming language CUDA. The experimental results show that the proposed method can render the ABS surfaces of degrees 6–9 composed of thousands of patches in real time.

Several aspects of the proposed method can be improved in future research. In the isotopic polygonization, each patch corresponds to a grid resolution. We should choose the maximum resolution, so that the isosurface is free of cracks. This will lead to additional costs. Aliasing artifacts may arise due to ill-conditioned calculations. For example, when one patch is almost

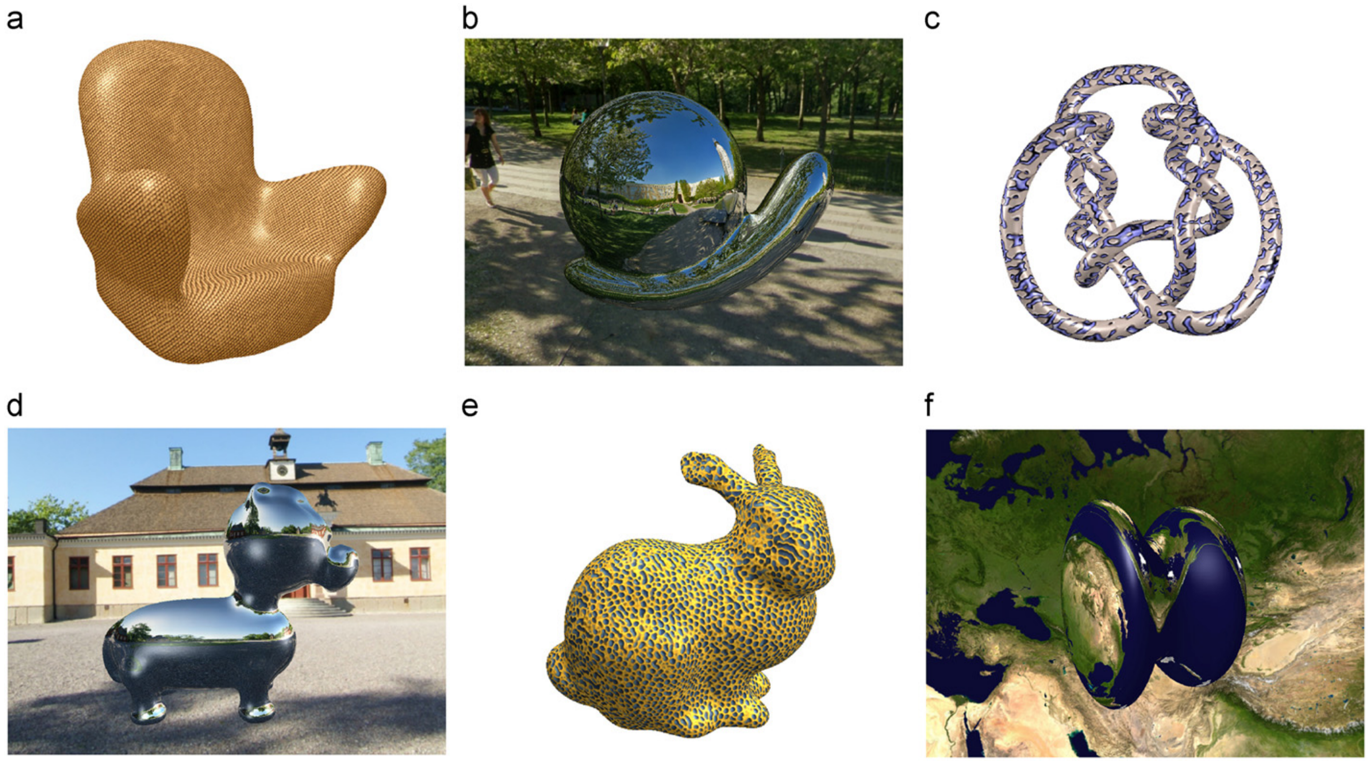


Fig. 11. Rendering results. The models are constructed by approximation of triangle meshes and are all C^1 smooth: (a) 1145 patches, C^1, 70.50 fps, (b) 1376 patches, C^1, 78.07 fps, (c) 3622 patches, C^1, 49.75 fps, (d) 1063 patches, C^1, 92.35 fps, (e) 937 patches, C^1, 66.47 fps and (f) 1494 patches, C^1, 56.17 fps.

planar, and is viewed from an extremely flat angle, silhouette computation will be very sensitive to numerical noise as shown in Fig. 9. Currently, other root-finding methods also struggle to handle this case well because the nonlinear system will be under-determined. However, the region with artifacts is very small in general. Nevertheless, we are looking forward to seeking a more robust and precise root-finding algorithm to address this problem. The criteria used to determine whether a polar surface is null or not is conservative. As a result, some overheads for root isolation are required even in the case of a single root. The situation can be improved by the use of more accurate function estimation.

Acknowledgment

The research is funded by the NSF of China (60933007, 60873046), the 973 program of China (2009CB320801). We would like to thank Dr. Aaron Knoll for his valuable suggestions, which clarify the advantages and difficulties in root-finding.

Appendix A. Proof of root isolation via polar surface

Let $\mathbf{E}(x_e,y_e,z_e)$ and $\mathbf{d}(d_x,d_y,d_z)$ be a viewpoint and ray direction, respectively. Then, the ray originated from $\mathbf{E}$ can be expressed as

$$\begin{cases} x(t)=x_e+t\cdot d_x \\ y(t)=y_e+t\cdot d_y \\ z(t)=z_e+t\cdot d_z \end{cases} \tag{A.1}$$

The ray and surface intersection can be obtained by solving the following univariate polynomial equation:

$$f(t)=F(x(t),y(t),z(t))=0 \tag{A.2}$$

Its derivative polynomial is

$$\begin{aligned} f'(t) &= F_x\cdot d_x+F_y\cdot d_y+F_z\cdot d_z = F_x\left(\frac{x-x_e}{t}\right)+F_y\left(\frac{y-y_e}{t}\right)+F_z\left(\frac{z-z_e}{t}\right) \\ &= \frac{nF-w_eF_w-x_eF_x-y_eF_y-z_eF_z}{t} = \frac{nF-P}{t} \end{aligned} \tag{A.3}$$

where the third step is obtained according to Euler's law, which is the relationship between a homogeneous polynomial of degree n and its derivative:

$$F(x,y,z,w)=\frac{xF_x+yF_y+zF_z+wF_w}{n} \tag{A.4}$$

By using Eq. (A.3), the functional composition between the ray (A.1) and the polar surface (4) can be written as

$$p(t)=nf(t)-tf'(t) \tag{A.5}$$

This is a linear combination of $f(t)$ and its derivative $f'(t)$.

Theorem 1. *Assuming that a polynomial $f(t)$ has two adjacent single roots t_0 and t_1 in an interval $[a, b]$. We define a new polynomial as*

$$g(t)=c_0f(t)+c_1f'(t) \tag{A.6}$$

where c_0 and c_1 are non-zero real numbers. Then for any given c_0 and c_1, $g(t)$ has at least one root in the interval (t_0,t_1).

Proof. Let the root of $g(t)$ in the interval (t_0,t_1) be noted as $r(c_0,c_1)$. When $c_1\to 0$, $r(c_0,c_1)$ can approach its lower and upper bounds t_0 and t_1, respectively.

If $t_0\neq t_1$, then according to Rolle's' theorem, $f'(t)$ has at least one root $t_2\in[t_0,t_1]$. Without loss of generality, we assume $f(t_2)>0$. There exist $t_3\in[t_0,t_2]$ and $t_4\in[t_2,t_1]$ such that $f(t_3)=f'(t_3)$ and $f(t_4)=-f'(t_4)$. Let $k=c_1/c_0$, because $f(t)$ and $f'(t)$ are continuous, there is at least one root $r(c_0,c_1)$ for $g(t)=0$ in

1. $[t_0,t_3]$ if $k\in(0,-1]$;
2. $[t_3,t_2]$ if $k\in[-1,-\infty)$;
3. $[t_2,t_4]$ if $k\in(\infty,1]$;
4. $[t_4,t_1]$ if $k\in[1,0)$.

It is similar to the case of $f(t_2)>0$. Therefore, $g(t)$ has at least one root in the interval (t_0,t_1). □

Appendix B. Supplementary material

Supplementary data associated with this article can be found in the online version of 10.1016/j.cag.2011.04.002.

References

[1] Alberti L, Comte G, Mourrain B. Meshing implicit algebraic surfaces: the smooth case. In: Schumaker L, Mæhlen M, Mørken K, editors. Mathematical methods for curves and surfaces: Tromsø'04, Nashboro, Tromsø Norway. p. 11–26.
[2] Alberti L, Mourrain B, Técourt JP. Isotopic triangulation of a real algebraic surface. J Symbolic Comput 2009;44:1291–310. Effective Methods in Algebraic Geometry.
[3] Bajaj CL, Chen J, Xu G. Modeling with cubic a-patches. ACM Trans Graph 1995;14:103–33.
[4] Blinn J. Jim Blinn's corner, how to solve a cubic equation. IEEE Comput Graph Appl 2007;27:78–89.
[5] Bloomenthal J. An implicit surface polygonizer. In: Graphics gems IV. San Diego, CA, USA: Academic Press Professional Inc.; 1994. p. 324–49.
[6] Che X, Liang X, Li Q. G1 continuity conditions of adjacent nurbs surfaces. Comput Aided Geom Des 2005;22:285–98.
[7] DeRose TD, Goldman RN, Hagen H, Mann S. Functional composition algorithms via blossoming. ACM Trans Graph 1993;12:113–35.
[8] Hanrahan P. Ray tracing algebraic surfaces. In: SIGGRAPH '83: proceedings of the 10th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1983. p. 83–90.
[9] Harris M, Owens J, Sengupta S, Zhang Y, Davidson A. Cudpp homepage, ⟨http://www.gpgpu.org/developer/cudpp/⟩; 2007.
[10] Jüttler B, Felis A. Least-squares fitting of algebraic spline surfaces. Adv Comput Math 2002;17:135–52, doi:10.1023/A:1015200504295.
[11] Kajiya JT. Ray tracing parametric patches. In: SIGGRAPH '82: proceedings of the ninth annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1982. p. 245–54.
[12] Kalra D, Barr AH. Guaranteed ray intersections with implicit surfaces. In: SIGGRAPH '89: proceedings of the 16th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1989. p. 297–306.
[13] Kanamori Y, Szego Z, Nishita T. Gpu-based fast ray casting for a large number of metaballs. Comput Graph Forum 2008;27:351–60.
[14] Knoll A, Hijazi Y, Kensler A, Schott M, Hansen C, Hagen H. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. Comput Graph Forum 2009;28:26–40.
[15] Knoll A, Wald I. Interactive ray tracing of arbitrary implicit functions. In: Proceedings of the second IEEE/EG symposium on interactive ray tracing, p. 11–8.
[16] Loop C, Blinn J. Real-time GPU rendering of piecewise algebraic surfaces. ACM Trans Graph 2006;25:664–70.
[17] Lorensen WE, Cline HE. Marching cubes: a high resolution 3D surface construction algorithm. SIGGRAPH Comput Graph 1987;21:163–9.
[18] Maekawa T, Patrikalakis NM. Computation of singularities and intersections of offsets of planar curves. Computer Aided Geometric Design 1993;10: 407–29.
[19] Mitchell DP, Robust ray intersection with interval arithmetic. In: Proceedings on graphics interface '90. Toronto, Ont., Canada: Canadian Information Processing Society; 1990. p. 68–74.
[20] Mørken K, Reimers M. An unconditionally convergent method for computing zeros of splines and polynomials. Math Comput 2007;76:845–65.
[21] Nishita T, Sederberg TW, Kakimoto M. Ray tracing trimmed rational surface patches. In: SIGGRAPH '90: proceedings of the 17th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1990. p. 337–45.
[22] Pabst HF, Springer JP, Schollmeyer A, Lenhardt R, Lessig C, Froehlich B. Ray casting of trimmed nurbs surfaces on the gpu. In: Proceedings of IEEE symposium on interactive ray tracing. Salt Lake City, UT, USA, 2006. p. 151–60.
[23] Patrikalakis NM, Kriezis GA. Representation of piecewise continuous algebraic surface in terms of B-splines. Vis Comput 1989;5:360–74.

[24] Patrikalakis NM, Maekawa T. Shape interrogation for computer aided design and manufacturing. Secaucus, NJ, USA: Springer-Verlag New York Inc.; 2002.
[25] Reimers M, Seland J. Ray casting algebraic surfaces using the frustum form. Comput Graph Forum 2008;27:361–70.
[26] Sederberg TW, Zundel AK. Scan line display of algebraic surfaces. In: SIGGRAPH '89: proceedings of the 16th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1989. p. 147–56.
[27] Singh JM, Narayanan PJ. Real-time ray tracing of implicit surfaces on the gpu. IEEE Trans Vis Comput Graph 2010;16:261–72.
[28] Stander BT, Hart JC. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In: SIGGRAPH '97: proceedings of the 24th annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 1997. p. 279–86.
[29] Tong W, Feng Y, Chen F. Hierarchical implicit tensor-product B-spline surface and its application in surface reconstruction. J Software 2006;17: 11–20.
[30] Witkin AP, Heckbert PS. Using particles to sample and control implicit surfaces. In: SIGGRAPH '94: proceedings of the 21st annual conference on computer graphics and interactive techniques. New York, NY, USA: ACM; 1994. p. 269–77.
[31] Wyvill G, McPheeters C, Wyvill B. Data structure for soft objects. Vis Comput 1986;2:227–34.