Direct Manipulation of Subdivision Surfaces on GPU

Abstract

We present an algorithm for interactive deformation of subdivision surfaces, including displaced subdivision surfaces and subdivision surfaces with geometric textures. Our system lets the user directly manipulate the surface using freely-selected surface points as handles. During deformation the control mesh vertices are automatically adjusted such that the deforming surface satisfies the handle position constraints while preserving the original surface shape and details. To best preserve surface details, we develop a gradient domain technique that incorporates the handle position constraints and detail preserving objectives into the deformation energy. For displaced subdivision surfaces and surfaces with geometric textures, the deformation energy is highly nonlinear and cannot be handled with existing iterative solvers. To address this issue, we introduce a shell deformation solver, which replaces each numerically unstable iteration step with two stable mesh deformation operations. Our deformation algorithm only uses local operations and is thus suitable for GPU implementation. The result is a real-time deformation system running orders of magnitude faster than the state-of-the-art multigrid mesh deformation solver. We demonstrate our technique with a variety of examples, including examples of creating visually pleasing character animations in real-time by driving a subdivision surface with motion capture data.

Keywords: subdivision surface, detail preservation, displacement mapping, geometric texture.

1 Introduction

Subdivision surfaces have been widely used in movie production, commercial modelers and game engines [DeRose et al. 1998; Warren and Weimer 2002]. Constructing surfaces through subdivision elegantly addresses many issues that computer graphics practitioners are confronted with, such as arbitrary topology, scalability, uniformity of representation, numerical stability and code simplicity [Zorin et al. 2000]. Traditional subdivision surfaces are mainly suitable for modeling piecewise smooth surfaces; the displaced subdivision surface introduced in [Lee et al. 2000] enhances that expressive power by integrating displacement mapping [Cook 1984] into the subdivision framework. Recently, researchers further added geometric textures to subdivision surfaces to make them truly powerful tools for modeling surfaces with complex details [Peng et al. 2004; Porumbescu et al. 2005].

In this paper we present an algorithm for interactive deformation of subdivision surfaces. Our algorithm has the following features:

- ◇ **direct manipulation:** Instead of using the control mesh, the user is free to select points on the target surface as handles for direct manipulation. To deform the surface, the user simply drags the handles to new positions and our algorithm automatically adjusts the control mesh to satisfy the handle position constraints.
- ◇ **detail preserving:** Our algorithm is effective in preserving surface details while generating visually pleasing deformation.

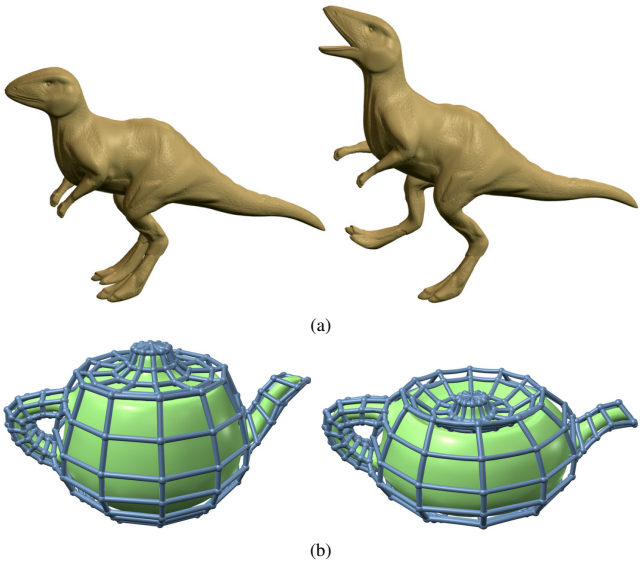


Figure 1: *Subdivision surface deformation. (a) Deformation of a displaced subdivision surface. The control mesh, base mesh and displacement map are shown in Fig. 2. (b) Deformation of a subdivision surface with geometric textures.*

- ◇ **real-time performance:** Our algorithm can be implemented on the GPU, with real-time performance ($>$ 100 FPS) for moderate-sized subdivision surface meshes.

Preserving details is important for subdivision surface deformation. Without detail preserving, the deformed surface can exhibit severe distortion as shown in Fig. 8. This motivated us to develop a gradient domain deformation algorithm for subdivision surfaces. Gradient domain techniques, introduced recently for mesh deformation and editing [Alexa 2003; Yu et al. 2004; Sorkine et al. 2004], are well-known for their ability to preserve surface details and generate visually pleasing results.

An immediate issue with a gradient domain algorithm for subdivision surfaces is that of maintaining subdivision surface representation. Like existing gradient domain techniques, we wish to manipulate the subdivision surface mesh directly and preserve details. Unlike existing techniques, which only generate a deformed mesh, we need to generate a new subdivision control mesh to ensure that the deformation result is actually a subdivision surface. We achieve this by projecting the deformation energy from the surface mesh to the control mesh, using the *subdivision detail function* that determines surface mesh vertices from control mesh vertices.

A much more challenging issue is the preservation of surface details during deformation. For a subdivision surface without displacement maps or geometry textures, the subdivision detail function is simply the linear function defined by the subdivision matrix [Warren and Weimer 2002]. The deformation energy in this case is only moderately nonlinear and can be minimized using a Gauss-Newton iterative method. This is similar to the situation with the subspace deformation technique [Huang et al. 2006], which uses the mean-value interpolation [Ju et al. 2005] to obtain a stable and fast solution.

For displaced subdivision surfaces and subdivision surfaces with geometry textures, the subdivision detail function is nonlinear. This leads to a highly nonlinear deformation energy and the Gauss-

Newton iteration used in [Huang et al. 2006] no longer converges. To handle this highly nonlinear energy, we introduce a *shell deformation solver*. A displaced subdivision surface or a subdivision surface with geometric texture is created by using the subdivision surface mesh as a *base mesh* over which displacement maps or geometric textures are mapped to generate a *detail mesh*. The base and detail meshes form the inner and outer boundaries of a shell. Our shell deformation solver operates within this shell, replacing each numerically unstable Gauss-Newton iteration with two stable deformation operations: one for optimizing the base mesh and the other for the detail mesh. By alternately optimizing the base and detail meshes, our technique essentially uses the deformation of the base mesh to compute a good initial estimation of the highly nonlinear components of the deformation energy and thus makes it trackable.

Our algorithm can be implemented on the GPU for real-time performance. A key observation about our algorithm is that it is completely designed using local operations and thus suitable for GPU implementation. This is an important feature that distinguishes our algorithm from other nonlinear deformation algorithms such as [Huang et al. 2006], which relies on a global scheme, the mean-value interpolation. To balance the workload between the CPU and GPU and take advantage of parallel execution streams in the GPU, we organize the control mesh of subdivision in texture memory following [Shiue et al. 2005]. We also precompute the matrix inversion needed and load the results into the GPU as texture images. This way the whole iterative solver rests on the GPU, resulting in high performance. Our GPU implementation runs orders of magnitude faster than the state-of-the-art fast deformation solver using multigrid [Shi et al. 2006].

With the proposed algorithm, good-quality deformation results can be achieved with high performance on the GPU. Fig. 1 provides deformation examples of our algorithm. We will demonstrate our technique with more examples. We will also show that with our GPU deformation algorithm, an animator can create visually pleasing, real-time animations from a static subdivision surface and motion capture data.

2 Related Work

[Welch and Witkin 1992] presents a variational algorithm for direct manipulation of B-spline surfaces based on energy minimization. Using the variational approach, [Boier-Martin et al. 2004] introduces a deformation technique for subdivision surfaces. To preserve surface details, they optimize the energy of a deformation vector field instead of the deformation energy of vertex positions. With their technique the deformation result is always a fine mesh with a deformation vector associated with each vertex. This can be quite inconvenient when working with subdivision surfaces without displacements. Furthermore, since displacements at vertices are not texture-mapped from a displacement map, this approach does not scales up well as the subdivision level increases. Most importantly, the above technique cannot handle geometry textures that are not displacement maps.

Freeform deformation (FFD) [Sederberg and Parry 1986] embeds an object inside a volume lattice. The user deforms the object by manipulating the lattice points. Several extensions have been proposed to provide a more intuitive user interface by directly manipulating points [Hsu et al. 1992] or curves [Singh and Fiume 1998] on the object surface. Recent approach [Botsch and Kobbelt 2005] uses volume-based radial basis function to deform the object. Real-time performance on large meshes has been achieved for deformation with predefined handles.

Multiresolution editing techniques [Zorin et al. 1997; Kobbelt et al. 1998; Guskov et al. 2000] can preserve surface details by decomposing a mesh into several frequency bands. A deformed mesh is obtained by first manipulating the low-frequency mesh and later adding back the high frequency details as displacement vectors. Recently, [Marinov et al. 2006] mapped a two-band multiresolution deformation framework to the GPU. These methods do not support direct manipulation of the original surface. Also, the displacement vectors are inserted back independently at each vertex. As a result, artifacts can appear in highly deformed regions because details are not coupled and preserved uniformly over the surface.

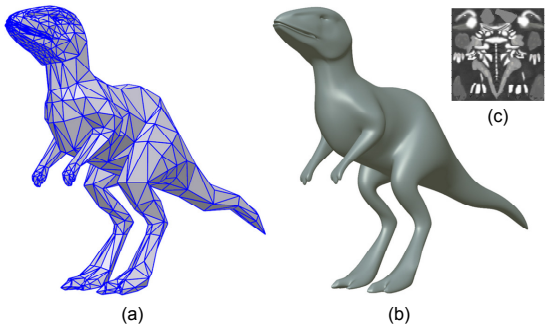


Figure 2: *(a) Control mesh shown in blue. (b) Base mesh. (c) Displacement map. See Fig. 1 for the detail mesh of the subdivision surface.*

Gradient domain mesh deformation techniques [Alexa 2003; Yu et al. 2004; Sorkine et al. 2004; Sheffer and Kraevoy 2004; Zhou et al. 2005; Lipman et al. 2005; Nealen et al. 2005; Zayer et al. 2005; Au et al. 2006; Huang et al. 2006; Lipman et al. 2006] cast deformation as an energy minimization problem. The energy function incorporates position constraints as well as terms for detail preservation. Minimization of this energy distributes errors globally over the entire mesh and thus leads to high quality deformation results. The user can directly manipulates the mesh surface and use the region of interest to control the scale of manipulation.

Our algorithm combines the strengths of gradient domain techniques and subdivision surfaces to achieve visually pleasing deformation and high performance. Recently, [Shi et al. 2006] presents a fast multigrid solver for gradient domain mesh deformation. Unfortunately, their GPU implementation does not run much faster than on the CPU version due to the unstructured nature of a general mesh. Thanks to the regular connectivity and locality-preserving data access of subdivision surfaces, our deformation algorithm can be efficiently implemented on the GPU, resulting in a real-time system which runs orders of magnitude faster than the multigrid solver of [Shi et al. 2006]. We deem our algorithm a nice complement to existing GPU-based subdivision techniques [Bolz and Schröder 2004; Shiue et al. 2005].

Deformation is an active research area and the above review only summarizes techniques most relevant to our work. Other deformation approaches include example-based mesh deformation [Sumner et al. 2005; Der et al. 2006], vector field based shape deformation [von Funck et al. 2006], and volumetric prism based deformation [Botsch et al. 2006].

3 Subdivision Surface Deformation

In this paper, a triangle mesh M is represented by a tuple (K,V), where K is an abstract simplicial complex containing mesh connectivity information and $V = (v_1,...,v_m)^T$ is a $3m$-dimensional vector with each $v_i \in R^3$ representing a vertex position.

3.1 Laplacian Deformation

We first derive a formulation for direct manipulation of subdivision surfaces following the Laplacian surface editing approach for

meshes [Sorkine et al. 2004; Yu et al. 2004]. Let the control mesh of the subdivision surface be M_c with vertices V_c as shown in Fig. 2. From the control mesh, a *base mesh* M_b is obtained by subdividing V_c to a desired level. A *detail mesh* M_d with vertices V_d is then generated on top of the base mesh by applying either a displacement map [Lee et al. 2000] or geometric texture [Peng et al. 2004].

The detail mesh M_d is the subdivision surface that we wish to deform through direct manipulation. We can carry out Laplacian deformation of M_d by minimizing the following energy function:

$$\min_{V_d} \left(\|LV_d - \hat{\delta}(V_d)\|^2 + \|CV_d - U\|^2 \right), \tag{1}$$

where L is the Laplacian operator matrix of M_d, $\hat{\delta}(V_d)$ is the Laplacian coordinates of V_d, and C is the positional constraint matrix and U is the target positions of the constrained vertices (i.e., vertices under direct manipulation). $\hat{\delta}(V_d)$ is a nonlinear function of the vertex positions because it includes local rotations.

We must ensure that the deformation result is still a subdivision surface. For this purpose we rewrite Eq. (1) in terms of the control mesh vertices V_c. Through subdivision, displacement mapping, and geometric texture mapping, the vertices of the detail mesh M_d may be computed from V_c as follows:

$$V_d = f(V_c),$$

where the function f is determined by the subdivision rules, displacement mapping and texture mapping procedures. We call f the *subdivision detail function*. For a subdivision surface without displacement maps or geometry textures, f is simply the linear function defined by the subdivision matrix, as we shall see. In general f is a complex nonlinear function.

By replacing V_d with $f(V_c)$ we turn Eq. (1) into

$$\min_{V_c} \left(\|Lf(V_c) - \hat{\delta}(f(V_c))\|^2 + \|Cf(V_c) - U\|^2 \right). \tag{2}$$

This is the basic formulation for the Laplacian deformation of subdivision surfaces. The deformation proceeds by first solving Eq. (2) for the control mesh V_c and then applying subdivision rules, displacement maps, and geometric textures to arrive at the deformed detail mesh M_d. Fig. 3 provides an example of subdivision surface deformation by direct manipulation.

In the following we present our technique for solving Eq. (2), starting with the simple case in which displacement mapping and geometric textures are absent and M_d is simply the base mesh M_b.

3.2 Base Surface Deformation

What makes this case simple is the fact that the subdivision detail function $f(V_c)$ is a linear function. Let $M_b = M_l$ be the l^{th}-level subdivision mesh of the control mesh $M_c = M_0$ and

$$V_b = S_l S_{l-1} \ldots S_1 V_c = S_b V_c.$$

The *subdivision matrix* for $M_c \rightarrow M_b$ is $S_b = S_l S_{l-1} \ldots S_1$, where S_i is the *subdivision matrix* for $M_{i-1} \rightarrow M_i$ $(i = 1, \ldots, l)$. The subdivision detail function $f(V_c) = S_b V_c$ is the linear function defined by subdivision matrix.

With a linear $f(V_c)$ Eq. (2) becomes

$$\min_{V_c} \|AV_c - b(V_c)\|^2, \text{ where} \tag{3}$$

$$A = \begin{pmatrix} L_b S_b \\ C S_b \end{pmatrix}, \quad b(V_c) = \begin{pmatrix} \hat{\delta}(S_b V_c) \\ U \end{pmatrix},$$

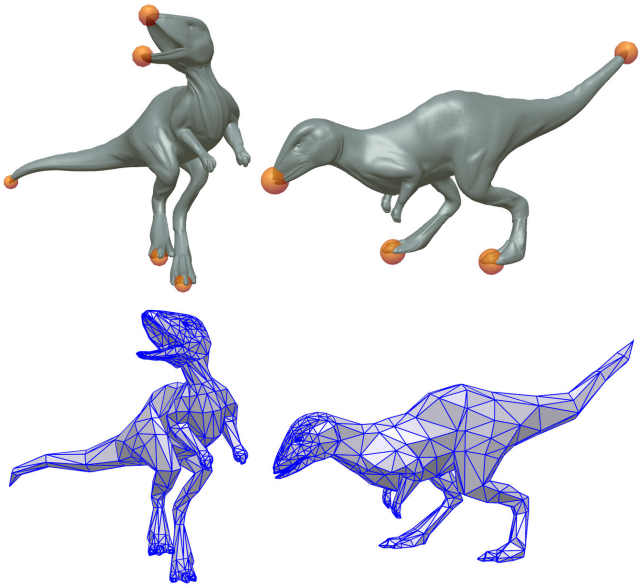


Figure 3: *Subdivision surface deformation via direct manipulation. Top row: The user deforms the detail mesh using freely selected surface points as handles (shown as orange dots). Bottom row: Our algorithm automatically adjusts the control mesh accordingly.*

where L_b is the Laplacian operator matrix of the base mesh M_b since we are examining the case when $M_d = M_b$. Here we use L_b instead of L to emphasize its relationship to M_b. Note that $b(V_c)$ is a nonlinear function of V_c because of the nonlinear $\hat{\delta}$.

As in [Huang et al. 2006], Eq. (3) can be solved using an inexact Gauss-Newton method [Steihaug 1995],

$$\min_{V_c^{k+1}} \|AV_c^{k+1} - b(V_c^k)\|^2. \tag{4}$$

In each iteration, $b(V_c^k)$ is known and Eq. (4) is solved as a least squares problem

$$V_c^{k+1} = (A^T A)^{-1} A^T b(V_c^k). \tag{5}$$

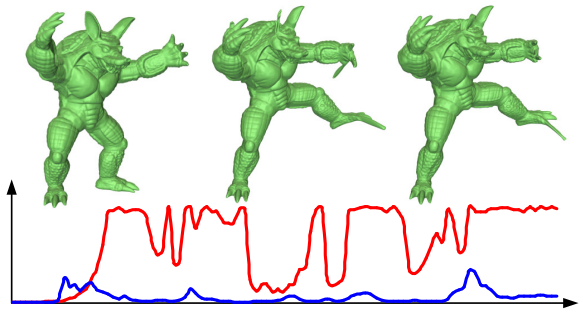
3.3 Shell Deformation Solver

The inexact Gauss-Newton method for base surface deformation essentially uses the following linearization:

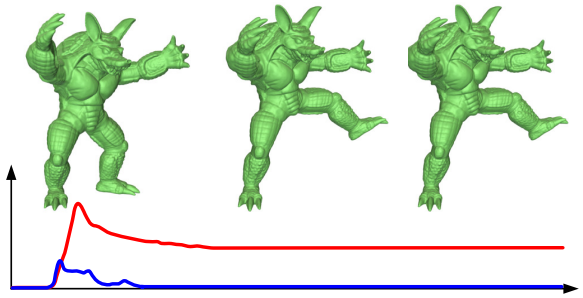
$$\begin{aligned} AV_c^{k+1} - b(V_c^{k+1}) &\approx AV_c^k - b(V_c^k) + (A - J_b(V_c^k))(V_c^{k+1} - V_c^k) \\ &\approx AV_c^k - b(V_c^k) + A(V_c^{k+1} - V_c^k) \\ &= AV_c^{k+1} - b(V_c^k), \end{aligned} \tag{6}$$

where J_b is the Jacobian of b. This approximation is accurate when either $\|J_b(V_c^k)\| \ll \|A\|$ or the step size $\|V_c^{k+1} - V_c^k\|$ is very small. In practice, the step size is not always small because large step sizes at the beginning of the iterative process are usually necessary for fast convergence. Fortunately, we do have $\|J_b(V_c^k)\| \ll \|A\|$ because the subdivision detail function $f(V_c)$ is linear and the nonlinearity of $b(V_c)$ is solely caused by the nonlinear Laplacian coordinates $\hat{\delta}$. In this case, $b(V_c)$ is only moderately nonlinear and the above one-step linearization method suffices. Our experiments indicate that $\|J_b^T J_b)\| / \|A^T A\|$ is in the range of $< 1.0e^{-3}$.

In general, the detail mesh M_d and the base mesh M_b differ and the subdivision detail function $f(V_c)$ is nonlinear. This nonlinear $f(V_c)$ leads to highly nonlinear deformation energy functions that cannot be minimized using the above one-step linearization method. To



(a) [Huang et al. 2006]



(b) our shell deformation solver

Figure 4: *The convergence of the subspace deformation solver [Huang et al. 2006] and our shell deformation solver. The horizontal axis is for time. The red curves indicate the deformation energy while the blue curves indicate the iteration step sizes. See the companion video for animated deformation sequences.*

handle such deformation energy functions, we developed the shell deformation solver. For simplicity, we first describe the shell deformation solver for displaced subdivision surfaces.

Suppose the detail mesh M_d is created by applying a displacement map to the base mesh M_b. Each vertex v_i on M_b is displaced by a distance h_i along the normal $n_i \in R^3$. The vertex positions V_d of the detail mesh may be computed as follows:

$$V_d = V_b + HN_b,$$

where $N_b = (n_1, ..., n_m)^T$ is a nonlinear function of the vertex positions V_b. Since $V_b = S_b V_c$, N_b is also a nonlinear function of V_c. H is a $m \times m$ diagonal matrix with $H(i,i) = h_i$. Using $V_b = S_b V_c$, we can compute V_d from V_c as follows:

$$V_d = f(V_c) = S_b V_c + HN_b.$$

Now the subdivision detail function f is nonlinear because of the nonlinear function N_b. With this new f, Eq. (2) can be turned into

$$\min_{V_c} \|DV_c - d(V_c)\|^2, \tag{7}$$

$$D = \begin{pmatrix} L_d S_b \\ C S_b \end{pmatrix}, \quad d(V_c) = \begin{pmatrix} \hat{\delta}(f(V_c)) - L_d H N_b \\ U - CHN_b \end{pmatrix},$$

where L_d the Laplacian operator matrix of the detail mesh M_d – we use L_d instead of L to emphasize its relationship with M_d. The function $d(V_c)$ is highly nonlinear due to the nonlinearity of both $\hat{\delta}$ and N_b. Under this circumstance, the one-step linearization method in [Huang et al. 2006] no longer suffices and the corresponding Gauss-Newton solver usually runs into convergence problems as shown in Fig. 4.

The shell deformation solver is an iterative solver for Eq. (7). The base mesh M_b and the detail mesh M_d form a thin shell with inner boundary M_b and outer boundary M_d. In each iteration, the shell deformation solver optimizes M_b and M_d. An iteration starts with deforming the inner boundary M_b using the position constraints imposed on the outer boundary M_d. This is done by solving Eq. (4) according to the *inferred* position constraints on M_b. The inferred constraints are derived from U, the *given* constraints on M_d. After deforming the base mesh M_b, M_b is used to evaluate the nonlinear Laplacian coordinates $\hat{\delta}$ and displacement normals N_b in $d(V_c)$ in Eq. (7). Finally, the deformation of the detail mesh M_d is computed for the current iteration using $d(V_c)$ so evaluated.

Specifically, at each iteration k, we first compute an initial guess of the control mesh vertices $V_c^{k+\frac{1}{2}}$ using Eq. (5) and obtain

$$V_c^{k+\frac{1}{2}} = (A^T A)^{-1} A^T b'(V_c^k) \tag{8}$$

where $b'(V_c^k) = \begin{pmatrix} \hat{\delta}(S_b V_c) \\ U' \end{pmatrix}$ with U' representing the inferred position constraints on the base mesh M_b. U' is inferred from the original position constraints U as follows. Suppose a vertex v_i on the detail mesh M_d is constrained to move by Δv_i according to U. Then v_i's corresponding vertex on the base mesh M_b should be constrained to move by the same amount Δv_i according to U'.

To calculate the deformation of M_d for the current iteration, we compute V_c^{k+1} using the deformed base mesh control vertices $V_c^{k+\frac{1}{2}}$ by solving

$$\min_{V_c^{k+1}} \|DV_c^{k+1} - d(V_c^{k+\frac{1}{2}})\|^2.$$

The result is

$$V_c^{k+1} = (D^T D)^{-1} D^T d(V_c^{k+\frac{1}{2}}). \tag{9}$$

Fig. 3 shows the deformation results of a displaced subdivision surface. The user directly manipulates the points on the detail mesh. The control mesh is automatically adjusted and the surface details are nicely preserved.

Fig. 6 demonstrates the importance of preserving geometric details. The dinosaur model is a displaced subdivision surface created from the original scanned model using the algorithm described by [Lee et al. 2000]. Fig. 6 (c) is the result of deformation without preserving the details of the displacement map. This is generated by first deforming the base mesh using the algorithm described in Section 3.2 and then applying the displacement map to the deformed base mesh. Fig. 6 (d) is the result of the shell deformation solver, which preserves the geometric details of the detail mesh. As we can see from the zoomed versions in Fig. 6 (e) and (f), the geometric details in Fig. 6 (c) are heavily compressed compared to that of Fig. 6 (d). Fig. 8 provides another example that demonstrates the importance of preserving details.

While a complete analysis of the stability of the shell deformation solver is beyond the scope of this paper, the intuition behind the solver is not difficult to understand. The shell deformation solver essentially uses the deformation of the base mesh to compute a good initial estimation of $d(V_c)$ and thus makes the highly nonlinear $d(V_c)$ trackable. As noted in [Steihaug 1995], the numerical stability of the Gauss-Newton method heavily depends the nonlinearity of $d(V_c)$. $d(V_c)$ includes on two nonlinear components, N_b and $\hat{\delta}$, and $\hat{\delta}$ further depends on N_b. This complex nonlinearity makes the one-step linearization method numerically unstable even with small step sizes. The shell deformation solver replaces each Gauss-Newton iteration with two numerically stable deformation operations. The deformation of the base mesh M_b is stable because it only involves the nonlinearity of $\hat{\delta}$ and thus can be handle with the one-step linearization method. The deformation of the detail

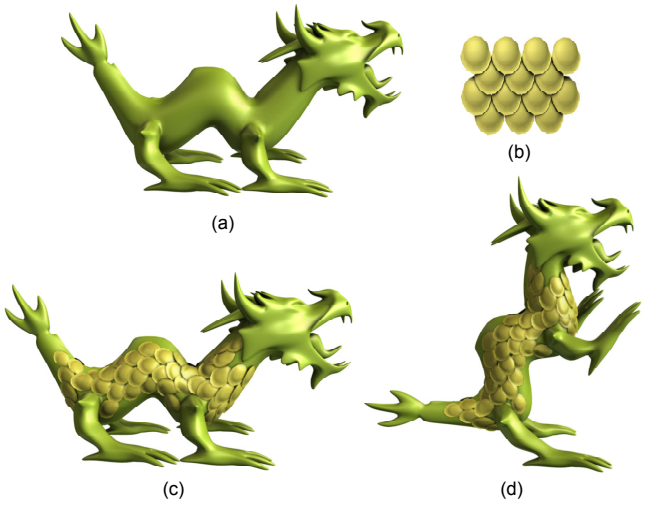


Figure 5: *Deformation of a subdivision surface with a complex geometric texture. (a) Base mesh. (b) Geometric texture. (c) The detail mesh. (d) A deformation result.*

mesh M_d is stable because the deformed M_b provides good initial estimations of V_c and N_b.

We have verified the numerical stability through a wide variety of experiments. Fig. 4 compares the stability of our shell deformation solver with the Gauss-Newton solver used in [Huang et al. 2006]. As we can see, our solver converges fast while the Gauss-Newton solver diverges with oscillations.

Our algorithm can be extended to support displacements along arbitrary directions, although only displacements along the normal direction are implemented currently. For general displacements, the displacement direction of each vertex of the base mesh is represented as a vector in the local frame defined by the vertex normal and the tangent vectors. These local vectors may be stored as an additional texture. At run time, we simply compute the global displacement directions using the local displacement directions and the local frames and feed the results to the shell deformation solver.

3.4 Handling Geometric Textures

The shell deformation solver can also handle subdivision surfaces with geometric textures. Fig. 5 shows a dragon model mapped with a squama geometric texture.

We use the shell map [Porumbescu et al. 2005] to map a geometric texture to the shell space over the base mesh M_b. First we construct a shell space over the base mesh M_b. An offset mesh M_t, which has the same number of vertices and the same mesh connectivity as M_b, is created using the envelope generation algorithm introduced in [Cohen et al. 1996]. As with displacement mapping, each vertex v_i of M_b is moved by a distance h_i along the normal direction at v_i. Thus the vertex positions of the offset mesh M_t can be expressed as

$$V_t = V_b + HN_b,$$

where $N_b = (n_1, ..., n_m)^T$, with each $n_i \in R^3$ being a unit normal vector. H is a diagonal matrix with $H(i,i) = h_i$.

We define a shell map by decomposing both the shell space (the space between M_b and M_t) and the texture space into two sets of corresponding tetrahedra. The shell map is defined by the barycentric coordinates of the corresponding tetrahedra. Given a point in the texture space, we can easily locate the tetrahedron it belongs to and compute its barycentric coordinates. Its corresponding point in the shell space is located in the corresponding tetrahedron with the same barycentric coordinates.

With the shell map, the vertex positions of the detailed mesh M_d can be represented as a linear combination of V_b and V_t:

$$V_d = (W_b \;\; W_t) \begin{pmatrix} V_b \\ V_t \end{pmatrix} = W_b V_b + W_t V_t = (W_b + W_t) V_b + W_t H N_b,$$

where $(W_b \;\; W_t)$ is the matrix of barycentric coordinates. Replacing V_b with $S_b V_c$, we get

$$V_d = f(V_c) = (W_b + W_t) S_b V_c + W_t H N_b.$$

This subdivision detail function f has essentially the same form as that of the displaced subdivision surface.

The detail mesh M_d can be deformed by solving Eq. (7) with a new matrix D and nonlinear function $d(V_c)$

$$D = \begin{pmatrix} L_d(W_b + W_t) S_b \\ C(W_b + W_t) S_b \end{pmatrix}, \quad d(V_c) = \begin{pmatrix} \hat{\delta}(f(V_c)) - L_d W_t H N_b \\ U - C W_t H N_b \end{pmatrix}.$$

Again, the nonlinear structures of $d(V_c)$ is the same for displacement maps and geometric textures. As a result, the shell deformation solver can be applied here.

Note that we can also use other algorithms such as [Peng et al. 2004] for constructing the offset surface M_t. With [Peng et al. 2004], the displacements from the base mesh vertices to the offset mesh vertices can be arbitrary. We can represent such displacements using the local frames defined by the vertex normals and the tangent vectors, storing the result as a texture. This is the same as a displacement map with arbitrary displacement directions, which we discussed earlier.

3.5 Implementation Details

The Laplacian operator matrix L can be constructed using the cotangent form as introduced in [Desbrun et al. 1999]. For the Laplacian coordinates $\hat{\delta}$, we employ the rotation-invariant representation introduced in [Huang et al. 2006]. Given an inner vertex v_i on the undeformed mesh, its one-ring vertices $\{v_{i,1}, ..., v_{i,m_i}\}$ and incident triangles $\{t_{i,j} = \triangle(v_i, v_{i,j-1}, v_{i,j})\}_{j=1}^{m_i}$, its Laplacian coordinate before deformation, δ_i, is first computed using L. Since the Laplacian is a discrete approximation of the curvature normal, it lies in the linear space spanned by the normals of the incident triangles. A set of coefficients μ_{ij} is then computed such that $\delta_i = \sum_{j=1}^{m_i} \mu_{ij} \left((v_{i,j-1} - v_i) \times (v_{i,j} - v_i) \right)$. Note that both L and $\{\mu_{ij}\}$ are precomputed for the undeformed mesh.

In each Gauss-Newton iteration, we need to compute $\hat{\delta}$. Here we use $\hat{\delta}(S_b V_c^k))$ as an example to show how this is done. We first compute the vertex positions $V_b^k = S_b V_c^k$, then calculate the Laplacian at iteration k using $\{\mu_{ij}\}$:

$$\varepsilon_i(V_b^k) = \sum_{j=1}^{m_i} \mu_{ij} \left((v_{i,j-1}^k - v_i^k) \times (v_{i,j}^k - v_i^k) \right). \tag{10}$$

Then we scale the magnitude of $\varepsilon_i(V_b^k)$ to keep the length of the original Laplacian before deformation:

$$\hat{\delta}_i(V_b^k) = \gamma_i \frac{\varepsilon_i(V_b^k)}{\|\varepsilon_i(V_b^k)\|}, \tag{11}$$

where $\gamma_i = \|\delta_i\|$ is the length of the original Laplacian.

Our current system uses the Loop subdivision scheme [Loop 1987].

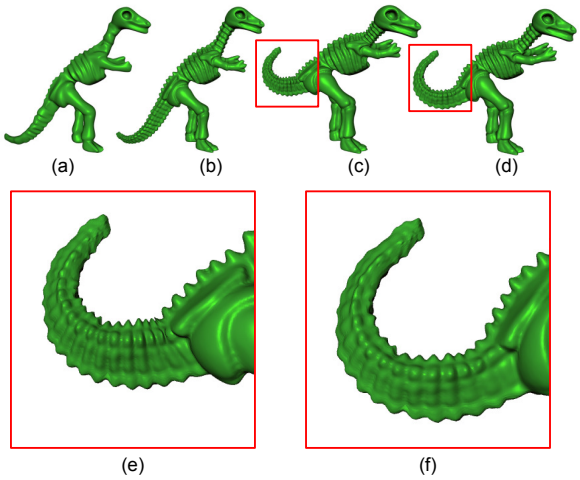


Figure 6: *Preserving details in a displaced subdivision surface. (a) Base mesh. (b) Detail mesh. (c) Deformation result without detail preservation. (d) Deformation result with detail preservation. (e) Zoomed version of (c). (f) Zoomed version of (d). As we can see from the zoomed versions, the geometric details in (c) are heavily compressed compared to that of (d).*

Model	# V_c	Subd. Level	# V_d	FPS
Dinosaur (Fig. 3)	721	4	184,066	125
Teapot (Fig. 1(b))	296	2	61,052	113
Dragon (Fig. 5)	1,157	2	22,706	122
Tower (Fig. 8)	68	2	29,995	122
Armadillo (Fig. 9)	1,202	4	307,202	103

Table 1: *Statistics for the examples used in the paper, including the numbers of vertices for the control and detail meshes, the subdivision levels, and the frame rates of the GPU implementation.*

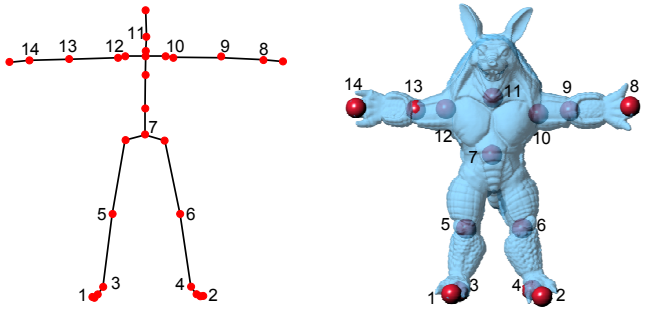


Figure 7: *The user can create real-time animations by driving a subdivision surface with motion capture data. Here is an example of setting up correspondences between the skeleton joints in the motion capture data and the handles on the subdivision surface.*

4 Real-Time Deformation on GPU

The main components of our deformation algorithm consist of local operations such as subdivision and Laplacian coordinates calculation, and matrix-vector multiplications. These operations can be efficiently implemented on a programable graphics hardware. In the following we use the base mesh deformation pipeline (Section 3.2) as an example to explain how this can be done.

According to Eq. (5), in each iteration we need to evaluate $V_c^{k+1} = (A^T A)^{-1} A^T b(V_c^{k+1})$. We precompute A^T and $(A^T A)^{-1}$ on the CPU and load the results into the GPU as two textures. Alternatively, we could precompute $(A^T A)^{-1} A^T$ and load it as a single texture. However, we choose not to do so for the following reasons. First, as A is a sparse matrix, computing $A^T b(V_c^k)$ only involves a sparse matrix-vector multiplication which is inexpensive. Second, we wish to keep the precomputation time short to facilitate user interaction, but calculating $(A^T A)^{-1} A^T$ would involve significantly more precomputation time due to the additional multiplication between a dense matrix $(A^T A)^{-1}$ and a sparse matrix A^T. Less precomputation time means quicker response to the user, because each time the user selects a new set of manipulation handles, we need to repeat the precomputation stage. To see this, note that A consists of two parts, $L_b S_b$ and $C S_b$. The first part is usually fixed because it only depends on the undeformed control mesh and the subdivision level. The second part, however, depends on the user selection of the manipulation handles which often change during a deformation session. Precomputing $(A^T A)^{-1}$ thus leads to quicker response to the user.

The calculation of $b(V_c^k)$ consists of two parts: $\hat{\delta}(S_b V_c^k)$ and U. Obtaining U is easy because it comes directly from the user input. To compute $\hat{\delta}(S_b V_c^k)$, we need to get the base mesh vertices $V_b^k = S_b V_c^k$ through subdivision, which can be efficiently performed using the subdivision kernel introduced in [Shiue et al. 2005]. The control mesh is first preprocessed into a set of fragment meshes. The fragment meshes that share the same lookup table are placed into a group and stored as a 2D texture using the spiral enumeration. Each fragment mesh in the group is mapped to a row in the 2D texture. Then in the fragment shader, the look-up table is used to fill the necessary subdivision stencil for each row. The subdivision results (vertex positions and normals) are either stored as 2D textures for subsequent processing or sent to pixel buffer objects (PBOs) for rendering. Once we get V_b^k, $\hat{\delta}(V_b^k)$ can be computed according to Eq. (10) and Eq. (11).

For the final evaluation of $V_c^{k+1} = (A^T A)^{-1} A^T b(V_b^k)$, we perform a sparse matrix-vector multiplication between A^T and $b(V_b^k)$ using the method described in [Bolz et al. 2003], followed by a dense matrix-vector multiplication between $(A^T A)^{-1}$ and $A^T b(V_b^k)$ which is carried out as in [Kruger and Westermann 2003].

5 Experimental Results

We have implemented the described algorithm on a 3.7Ghz PC with 2GB of memory and an NVidia 8800GTX graphics card. See the companion video for animated versions of the figures and other deformation examples. All video clips are captured live from our deformation system.

Fig. 3 and Fig. 6 show deformation results for displaced subdivision surfaces. Color textures are easily supported in our deformation system, as shown in Fig. 10. Fig. 1 (a) and Fig. 8 show the deformation results with geometric textures. The user directly manipulates the points on the geometric textures. The control mesh is automatically adjusted and the geometric details are nicely preserved. The tower in Fig. 8 is generated by mapping a geometric texture over a simple cylindric base mesh. This example demonstrates the importance of preserving surface details for high quality deformation results.

With our GPU deformation algorithm, an animator can create visually pleasing, real-time animations from a static subdivision surface and motion capture data (Fig. 9 and Fig. 10). The user simply selects vertices on the subdivision surface as handles and specifies a corresponding joint on the skeleton of motion capture data for each handle (see Fig. 7). Then the handle will move following the joint. Sometimes it is helpful to use a group of surface points as a handle. In that case, the centroid of the handle moves following the corresponding joint. Note that [Shi et al. 2006] also uses motion capture data to create mesh animations and they need to build a volumetric graph inside the mesh to get the rotation constraints from the

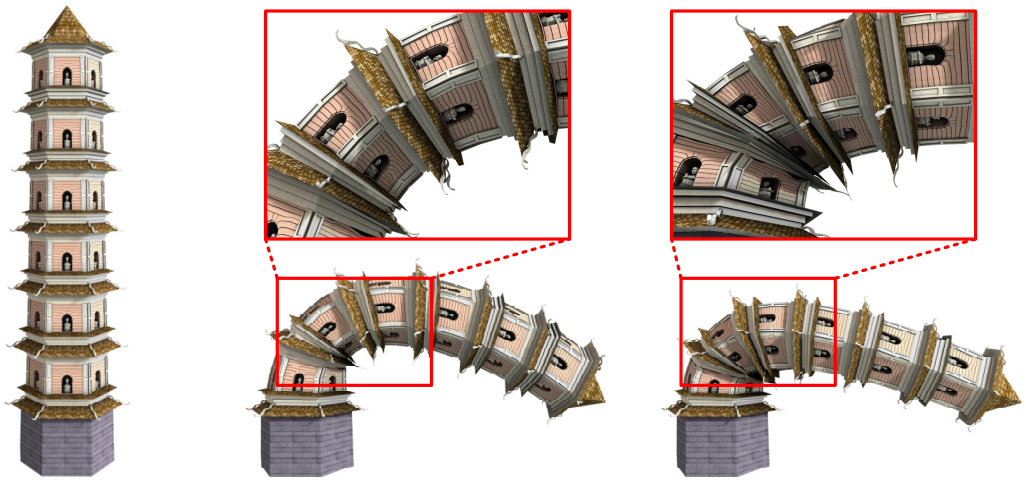


Figure 8: *Detail preservation for subdivision surface deformation. From left to right: the original model, deformation with detail preservation, and deformation result without detail preservation. The result of deformation without detail preservation is generated by first deforming the base mesh using the algorithm described in Section 3.2 and then mapping the geometric texture onto the deformed base mesh.*

bone transformation of motion capture data. We do not need such a volumetric graph because our algorithm can automatically infer rotations from handle translations. More importantly, [Shi et al. 2006] need several seconds to generate a frame frame while our GPU algorithm runs at real time.

Table 1 provides some statistics for the examples shown in the paper. For all examples, our deformation system can achieve real-time performance. As mentioned, when the user adds new handles or removes old handles, the positional constraint matrix C will change. Therefore, we need to re-compute the matrix inverse for $(A^TA)^{-1}$ and $(D^TD)^{-1}$. Fortunately, the dimensions of these matrices are decided by the vertex number of the control mesh, which is much smaller than the detail mesh. For all examples shown in this paper, this computation takes around $0.1 \sim 3$ seconds. The coefficients $\{\mu_{ij}\}$, the Laplacian and subdivision matrices are fixed during deformation and not affected by the handle selections. The precomputation time for these items are less than 7 seconds for all examples.

6 Conclusion

We have described an algorithm for interactive deformation of subdivision surfaces. This algorithm works for all commonly used subdivision surfaces, including displaced subdivision surfaces and subdivision surfaces with complex geometric textures. With our algorithm, the user can directly manipulate subdivision surfaces using freely-selected surface points as handles. The most important feature of our algorithm is that it combines the strengths of gradient domain techniques and subdivision surfaces to achieve both visually pleasing deformation and high performance. Specifically, our system automatically preserves surface details, generating high-quality deformation results by minimizing a deformation energy that incorporates both the Laplacian and handle position constraints. While significant computation is needed for minimizing a highly nonlinear deformation energy, our algorithm, designed with local operations and equipped with a novel shell deformation solver, achieves real-time performance on the GPU and is orders of magnitude faster than the state-of-the-art fast mesh deformation solver based on multigrid.

As a topic of future research, we plan to explore the use of adaptive subdivision in our system. Our current system only supports uniform subdivision. We are also interested in developing techniques for collision-free deformation with geometric textures. When deforming geometric textures, we do not update the offset surface and thus cannot guarantee collision-free deformation. If collision occurs locally, it is possible to prevent it by updating the offset surface interactively as described in [Peng et al. 2004]. However, a general solution to this problem merits further investigation.

References

ALEXA, M. 2003. Differential coordinates for local mesh morphing and deformation. *The Visual Computer 19*, 2, 105–114.

AU, O. K.-C., TAI, C.-L., LIU, L., AND FU, H. 2006. Dual laplacian editing for meshes. *IEEE TVCG 12*, 3, 386–395.

BOIER-MARTIN, I., RONFARD, R., AND BERNARDINI, F. 2004. Detail-preserving variational surface design with multiresolution constraints. In *Proceedings of the Shape Modeling International 2004*, 119–128.

BOLZ, J., AND SCHRÖDER, P. 2004. Evaluation of subdivision surfaces on programmable graphics hardware. to appear.

BOLZ, J., FARMER, I., GRINSPUN, E., AND SCHRÖDER, P. 2003. Sparse matrix solvers on the gpu: Conjugate gradients and multigrid. *ACM Trans. Graph. 22*, 3, 917–924.

BOTSCH, M., AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. In *Eurographics 2005*, 611–621.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, 11–20.

COHEN, J., VARSHNEY, A., MANOCHA, D., TURK, G., WEBER, H., AGARWAL, P., BROOKS, F., AND WRIGHT, W. 1996. Simplification envelopes. In *SIGGRAPH 96 Conference Proceedings*, 223–231.

COOK, R. L. 1984. Shade trees. In *SIGGRAPH 84 Conference Proceedings*, 223–231.

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable modelss. *ACM Trans. Graph. 25*, 3, 1174–1179.

DEROSE, T., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *SIGGRAPH 98 Conference Proceedings*, 85–94.

DESBRUN, M., MEYER, M., SCHRODER, P., AND BARR, A. H. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 99 Conference Proceedings*, 317–324.

GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRODER, P. 2000. Normal meshes. In *SIGGRAPH 2000 Conference Proceedings*, 95–102.

HSU, W. M., HUGHES, J. F., AND KAUFMAN, H. 1992. Direct manipulation of free-form deformations. In *SIGGRAPH 92 Conference Proceedings*, 177–184.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3, 1126–1134.

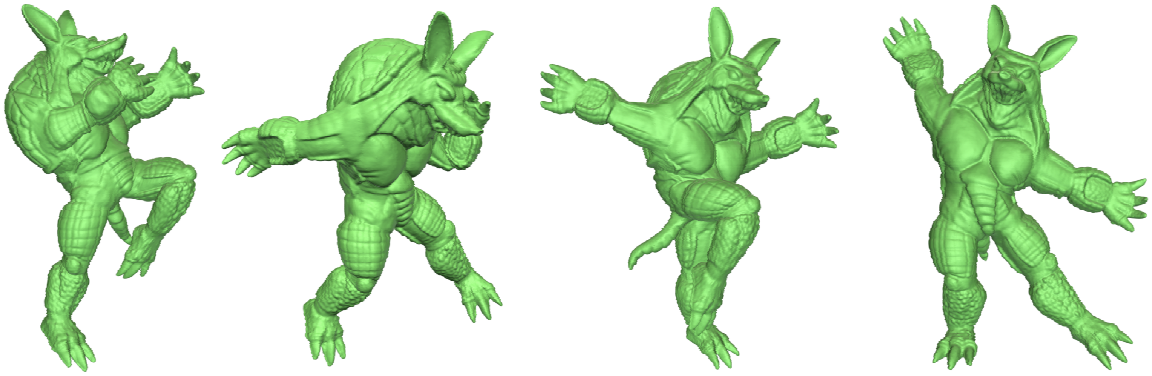


Figure 9: *Snapshots of a dancing Armadillo driven by motion capture data. See the companion video for the animation.*

Figure 10: *A frog and a dinosaur dancing together. Our GPU deformation algorithm is fast enough to deform multiple models simultaneously in real time.*

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3, 561–566.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, 105–114.

KRUGER, J., AND WESTERMANN, R. 2003. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. Graph. 22*, 3, 908–916.

LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *SIGGRAPH 2000 Conference Proceedings*, 85–94.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

LIPMAN, Y., COHEN-OR, D., GAL, R., AND LEVIN, D. 2006. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, to appear.

LOOP, C. T. 1987. Smooth subdivision surfaces based on triangles. Master's Thesis, Department of Mathematics, University of Utah.

MARINOV, M., BOTSCH, M., AND KOBBELT, L. 2006. Gpu-based multiresolution deformation using approximate normal field reconstruction. *Journal of Graphics Tools*, to appear.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3, 1142–1147.

PENG, J., KRISTJANSSON, D., AND ZORIN, D. 2004. Interactive modeling of topologically complex geometric detail. *ACM Trans. Graph. 23*, 3, 635–643.

PORUMBESCU, S. D., BUDGE, B., FENG, L., AND JOY, K. I. 2005. Shell maps. *ACM Trans. Graph. 23*, 3, 626–633.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *SIGGRAPH 86 Conference Proceedings*, 151–160.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proceedings of 3DPVT '04*, 68–75.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3, 1108–1117.

SHIUE, L.-J., JONES, I., AND PETERS, J. 2005. A realtime gpu subdivision kernel. *ACM Trans. Graph. 24*, 3, 1010–1015.

SINGH, K., AND FIUME, E. 1998. Wires: a geometric deformation technique. In *SIGGRAPH 98 Conference Proceedings*, 405–414.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Eurographics Symposium on Geometry Processing*, 175–184.

STEIHAUG, T. 1995. An inexact gauss-newton approach to mildly nonlinear problems. Tech. rep., Dept. of Mathematics, University of Linkoping.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.

WARREN, J., AND WEIMER, H. 2002. *Subdivision Methods for Geometric Design*. Morgan Kaufmann Publishers.

WELCH, W., AND WITKIN, A. 1992. Variational surface modeling. In *Proceedings of SIGGRAPH 92*, 157–166.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. In *Eurographics 2005*, 601–609.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3, 496–503.

ZORIN, D., SCHRÖDERR, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, 259–268.

ZORIN, D., SCHRÖDERR, P., DEROSE, T., KOBBELT, L., LEVIN, A., AND SWELDENS, W. 2000. Subdivision for modeling and animation. *Course notes of SIGGRAPH 2000*.