Large Mesh Deformation Using the Volumetric Graph Laplacian

Kun Zhou[1] Jin Huang[2*] John Snyder[3] Xinguo Liu[1] Hujun Bao[2] Baining Guo[1] Heung-Yeung Shum[1]

[1] Microsoft Research Asia [2] Zhejiang University [3] Microsoft Research

Abstract

We present a novel technique for large deformations on 3D meshes using the volumetric graph Laplacian. We first construct a graph representing the volume inside the input mesh. The graph need not form a solid meshing of the input mesh's interior; its edges simply connect nearby points in the volume. This graph's Laplacian encodes volumetric details as the difference between each point in the graph and the average of its neighbors. Preserving these volumetric details during deformation imposes a volumetric constraint that prevents unnatural changes in volume. We also include in the graph points a short distance outside the mesh to avoid local self-intersections. Volumetric detail preservation is represented by a quadric energy function. Minimizing it preserves details in a least-squares sense, distributing error uniformly over the whole deformed mesh. It can also be combined with conventional constraints involving surface positions, details or smoothness, and efficiently minimized by solving a sparse linear system.

We apply this technique in a 2D curve-based deformation system allowing novice users to create pleasing deformations with little effort. A novel application of this system is to apply nonrigid and exaggerated deformations of 2D cartoon characters to 3D meshes. We demonstrate our system's potential with several examples.

Keywords: differential domain methods, deformation retargeting, local transform propagation, volumetric details.

1 Introduction

Mesh deformation is useful in a variety of applications in computer modeling and animation. Many successful techniques have been developed to help artists sculpt stylized body shapes and deformations for 3D characters. In particular, multi-resolution techniques and recently introduced differential domain methods are very effective in preserving surface details, which is important for generating high-quality results. However, large deformations, such as those found with characters performing nonrigid and highly exaggerated movements, remain challenging today, and existing techniques often produce implausible results with unnatural volume changes.

We present a novel deformation technique that achieves convincing results for large deformations. It is based on the *volumetric graph Laplacian* (VGL), which represents *volumetric details* as the difference between each point in a 3D volume and the average of its neighboring points in a graph. VGL inherits the strengths of recent differential domain techniques [Yu et al. 2004; Sorkine et al. 2004]. In particular, it preserves surface details and produces visually-pleasing deformation results by distributing errors globally through least-squares minimization. But by working in the volumetric domain instead of on the mesh surface, VGL can effectively impose volumetric constraints to avoid unnatural volume changes and local self-intersections (Figure 1). Volumetric constraints are represented by a quadric energy function which can be efficiently minimized by solving a sparse linear system, and easily combined with other widely-used surface constraints (e.g., on surface positions, surface details [Sorkine et al. 2004], and surface smoothness [Botsch and Kobbelt 2004]).

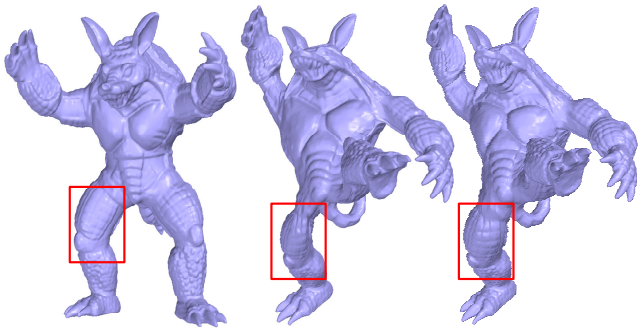

Figure 1: *Large deformation of the Stanford Armadillo. Left: original mesh; middle: deformed result using Poisson mesh editing; right: deformed result using our technique. Poisson mesh editing causes unnatural shrinkage especially in the model's right thigh.*

To apply the volumetric graph Laplacian to a triangular mesh, we construct a volumetric graph which includes the original mesh points as well as points derived from a simple lattice lying inside the mesh. These points are connected by graph edges which are a superset of the edges of the original mesh. The graph need not form a meshing (volumetric tessellation into tetrahedra or other finite elements) of the mesh interior. This flexibility makes it easy to construct. The deformation is specified by identifying a limited set of points on the original mesh, typically a curve, and where these points go as a result of the deformation. A quadric energy function is then generated whose minimum maps the points to their specified destination while maintaining surface detail and roughly preserving volume.

Our main contribution is to demonstrate that the problem of large deformation can be effectively solved by using a *volumetric* differential operator. Previous differential approaches [Yu et al. 2004; Sorkine et al. 2004] considered only surface operators. A naive way to extend these operators from surfaces to solids is to define them over a tetrahedral mesh of the object interior. However, solidly meshing a complex object is notoriously difficult. To our knowledge, available packages remesh geometry and disturb its connectivity, violating a common requirement in mesh deformation. Solid meshing also implies many constraints (e.g., that no tetrahedron be flipped and that each interior vertex remain in the visual hull of its neighbors) that make it harder to economically distribute interior points and add an "exterior shell" as we do to prevent local self-intersection. Our key insight is that the volumetric Laplacian operator can be applied to an easy-to-build volumetric graph without meshing surface interiors.

Using the method, we have developed an interactive deformation

*This work was done while Jin Huang was an intern at Microsoft Research Asia.

system based on 2D curves. Manipulating vertices in 3D space is tedious and requires artistic skill; our system allows novices to create pleasing results with a few, simple operations. A novel application of this system is to transfer the exaggerated deformations of 2D cartoon characters to 3D models by specifying a set of corresponding curves between the images and models. Our technique does not require the skeletons and key poses of the 3D models as input and can handle a wide range of nonrigid deformations.

2 Related Work

Mesh Deformation Energy minimization has long been used to design smooth surfaces [Welch and Witkin 1994; Taubin 1995]. Recently, a freeform modeling system allows users to define basis functions customized to a given design task [Botsch and Kobbelt 2004]. The resulting linear system handles arbitrary regions and piecewise boundary conditions with smoothness ranging continuously from C^0 to C^2.

Freeform deformation (FFD) is used in commercial software such as 3D Studio and Maya. A general treatment can be found in [Milliron et al. 2002]. FFD methods can be classified as lattice-based [Sederberg and Parry 1986; Coquillart 1990; MacCracken and Joy 1996], curve-based [Barr 1984; Singh and Fiume 1998], or point-based [Hsu et al. 1992; Bendels and Klein 2003]. Some approaches [Rappoport et al. 1995; Hirota et al. 1999] preserve the global volume of the object.

While energy minimization and FFD methods work well for smooth surfaces, multiresolution editing [Zorin et al. 1997; Kobbelt et al. 1998; Guskov et al. 1999] is better suited for detailed geometry such as that acquired from scanning devices. A model is first decomposed into a smooth base shape and a set of geometric details, represented as displacements in a local coordinate frame. After modifying the base shape with some freeform deformation, the details can be re-inserted. The problem with these methods is that the displacement vectors are manipulated independently at each vertex. Artifacts can appear in highly deformed regions because details are not coupled and preserved uniformly over the whole surface.

Displacement volumes [Botsch and Kobbelt 2003] extend the multi-resolution approach from surfaces to volumes, and coin the term "volumetric details" which we borrow here. Though the method addresses problems with local self-intersection, it may concentrate errors and thus artifacts in highly deformed regions such as the bend shown in its Figure 6c and Figure 7c. These artifacts are exacerbated by the iterative relaxation performed to enforce the nonlinear volumetric constraints.

Our approach builds on recent work that encodes surface details differentially; i.e., as local differences or derivatives. Differential domain methods, including ours, minimize an energy function representing how well the details are preserved after a deformation and can be solved as a sparse linear system. Poisson meshes [Yu et al. 2004] manipulate gradients of the mesh's coordinate functions using an FFD and then reconstruct the surface from the Poisson equation. Laplacian coordinates [Alexa 2003; Lipman et al. 2004; Sorkine et al. 2004] represent surface details as differences from a local mean. We extend these ideas to the volumetric domain to solve the problem of large deformations.

Mesh deformation is closely related to shape interpolation and morphing. Morphing can be extended from surfaces to solids by minimizing distortions in a local volume [Alexa et al. 2000]. A tetrahedral mesh must be constructed for the input triangular mesh, which we avoid by using a simpler volumetric graph. [Sheffer and Kraevoy 2004] propose a morphing and deformation method based on *pyramid coordinates*. Reconstruction from pyramid coordinates to vertex coordinates requires solving a nonlinear system.

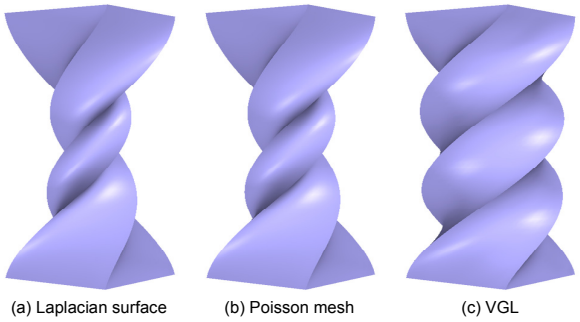

Figure 2: *Large twist deformation.*

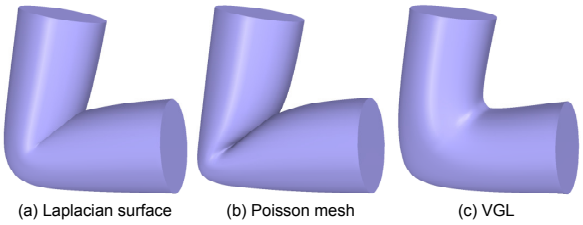

Figure 3: *Large bend deformation.*

2D Curve-based Deformation Since manipulating 3D vertices is tedious, some methods modify 3D objects by 2D curve editing. The Teddy system [Igarashi et al. 1999] allows users to create 3D objects by sketching 2D curves. It also supports a global deformation operation based on stroke warping. Recently, curve analogies [Hertzmann et al. 2002] have been extended to surfaces by applying to 3D models the transformation determined by two curves [Zelinka and Garland 2004]. The authors further propose a sketch-based interface [Kho and Garland 2005], which allows users to bend and twist models by sketching 2D curves.

Deformation Retargeting Reusing the deformation created for one 2D or 3D shape to deform another is often useful, especially for movie production. [Bregler et al. 2002] capture the affine deformation from existing 2D cartoon animations and retarget it onto 2D drawings and 3D shapes. [Favreau et al. 2004] animate 3D models of animals from existing live video sequences. Both methods require the skeleton and key poses of the model as input. Recently, [Sumner and Popović 2004] propose a technique to transfer the deformation of a source triangle mesh onto a target triangle mesh.

3 Deformation on Volumetric Graphs

Let $M = (V,K)$ be the triangular mesh we want to deform, where V is a set of n point positions $V = \{p_i \in \mathrm{R}^3 | 1 \le i \le n\}$, and K is an abstract simplicial complex which contains all the vertex connectivity information. There are three types of elements in K, vertices $\{i\}$, edges $\{i,j\}$ and faces $\{i,j,k\}$.

3.1 Laplacian Deformation on Abstract Graphs

Suppose $G = (P,E)$ is a graph, where P is a set of N point positions $P = \{p_i \in \mathrm{R}^3 | \, 1 \le i \le N\}$, and $E = \{(i,j) | \; p_i \text{ is connected to } p_j\}$ is the set of edges. The Laplacian of a graph is analogous to the Laplace operator on manifolds [Chung 1997] and computes the difference between each point p_i in the graph G and a linear combination of its neighboring points:

$$\delta_i = \mathscr{L}_G(p_i) = p_i - \sum_{j \in \mathscr{N}(i)} w_{ij} p_j, \qquad (1)$$

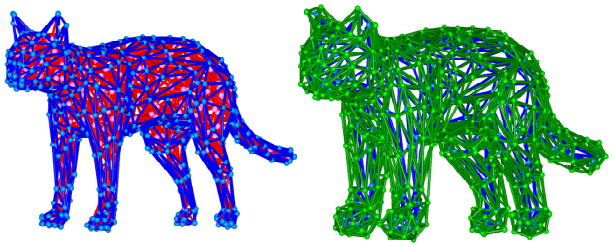

Figure 4: *Volumetric graph example. Left: G_{in}; right: G_{out}. The edges of the input mesh are marked in blue.*

where $\mathcal{N}(i) = \{j \mid \{i,j\} \in E\}$ are the edge neighbors, w_{ij} is the weight for point p_j, and δ_i is the *Laplacian coordinate* of the point p_i in graph G. $\mathcal{L}_G$ is called the Laplace operator of the graph G.

The weights w_{ij} should be positive and satisfy $\sum_{j\in\mathcal{N}(i)} w_{ij} = 1$. The simplest weighting is uniform weighting $w_{ij} = 1/|\mathcal{N}(i)|$ [Taubin 1995; Sorkine et al. 2004]. We use a more complicated weighting scheme, described in Section 3.3.

To control a deformation, the user inputs the deformed positions $q_i, i \in \{1,\ldots,m\}$ for a subset of the N mesh vertices. This information is used to compute a new (deformed) Laplacian coordinate δ_i' for each point i in the graph. The deformed positions of the mesh vertices p_i' are then obtained by solving the following quadric minimization problem:

$$\min_{p_i'} \left(\sum_{i=1}^{N} \|\mathcal{L}_G(p_i') - \delta_i'\|^2 + \alpha \sum_{i=1}^{m} \|p_i' - q_i\|^2 \right). \qquad (2)$$

The first term represents preservation of local detail and the second constrains the positions of those vertices directly specified by the user. The parameter α balances these two objectives.

The deformed Laplacian coordinates are computed via

$$\delta_i' = T_i\,\delta_i$$

where δ_i is the Laplacian coordinate in the rest pose, defined in (1), and T_i transforms it into the deformed pose. A general transform T_i which includes anisotropic scaling is too powerful and can "fit away" local detail. The solution is to restrict T_i to a rotation and isotropic scale [Sorkine et al. 2004].

Given the deformed positions of a subset of the vertices q_i, many methods can be used to obtain T_i. We use a method, described in Section 3.3, which propagates the local transformation from the specified region of deformation to the entire mesh, blending the transform towards the identity away from the deformation site.

If the graph is a triangular mesh, the graph Laplacian is identical to the mesh Laplacian. Using the mesh Laplacian to encode surface details, [Alexa 2003; Lipman et al. 2004; Sorkine et al. 2004] preserve detailed geometric structure over a wide range of editing operations. For large deformations, these methods exhibit unnatural volume changes (Fig. 2a) or local self-intersections (Fig. 3a). The following section describes how to impose volumetric constraints which reduce such undesirable effects, by constructing a volumetric graph for the mesh.

3.2 Constructing the Volumetric Graph

Like [Botsch and Kobbelt 2003], our method avoids large volume changes and local self-intersections but does not guarantee elimination of global self-intersections, whose prevention must be managed by the user. We build two kinds of volumetric graphs: an inside graph G_{in} fills the interior volume of the mesh and prevents

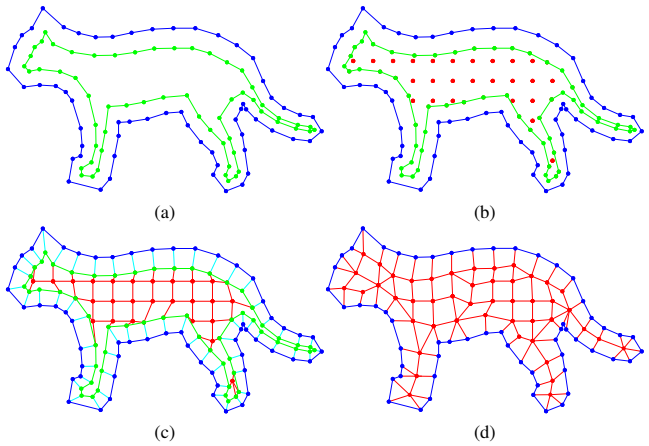

Figure 5: *Volumetric graph construction.*

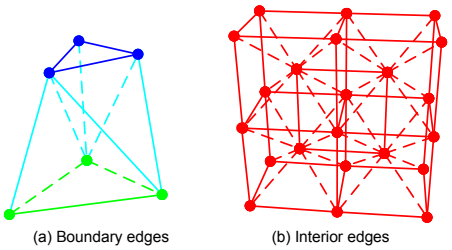

Figure 6: *Types of edge connections in the volumetric graph.*

large volume changes, while an outside graph G_{out} prevents local self-intersection.

A natural method for obtaining G_{in} is to tetrahedralize the interior volume of a surface mesh [Shewchuk 1998; Cutler et al. 2004; Bridson et al. 2004]. However, tetrahedral mesh generation is difficult to implement and computationally expensive (see the detailed survey by Owen [1998]). It is also hard to make robust and often produces too many or poorly shaped tetrahedra for complicated models [Shewchuk 1998]. We describe a simple method to produce the less-restrictive volumetric graph.

As Figure 5 illustrates, the algorithm consists of four steps:

- Construct an inner shell M_{in} for the mesh M by offsetting each vertex a distance in the direction opposite its normal (Fig.5a).
- Embed M_{in} and M in a body-centered cubic (BCC) lattice (Fig. 6b). Remove lattice nodes outside M_{in} (Fig. 5b).
- Build edge connections among M, M_{in}, and lattice nodes (Fig. 5c).
- Simplify the graph using edge collapse and smooth the graph (Fig. 5d).

The purpose of the inner shell M_{in} is to ensure that inner points are inserted even within thin features, like the tail of the cat, that may be missed by lattice sampling. To compute the inner shell, we use an an iterative method based on simplification envelopes [Cohen et al. 1996]. In each iteration, we attempt to move each vertex a fraction of the average edge length opposite to its normal vector. After moving a vertex, we test its adjacent triangles for intersections with each other and the rest of the model. If no intersections are found, we accept the step; otherwise, we reject it and move the vertex back. The iterations terminate when all vertices have moved the desired distance or can no longer move.

The BCC lattice consists of nodes at every point of a Cartesian grid

along with the cell centers (Figure 6b). Node locations may be viewed as belonging to two interlaced grids. This lattice occurs as a crystal structure in nature with desirable rigidity properties. Currently we set the grid interval to the average edge length.

Three types of edge connections form an initial graph. First, each vertex in M is connected to its corresponding vertex in M_{in} (Figure 6a). The shorter diagonal for each prism face is included as well. Second, each inner node of the BCC lattice is connected with its eight nearest neighbors in the other interlaced grid (Figure 6b). Third, connections are made between M_{in} and nodes of the BCC lattice. For each edge in the BCC lattice that intersects M_{in} and has at least one node inside M_{in}, we connect the BCC lattice node inside M_{in} to the point in M_{in} closest to this intersection.

Simplification and smoothing on the initial graph make it more uniform. We visit the graph edges in increasing order of length. If the length of an edge is less than a threshold (half the average edge length of M), it is collapsed to the edge's midpoint. After simplification, several smoothing iterations (three in our implementation) are performed in which each point is moved to the average of its neighbors. Note that neither simplification nor smoothing are applied to the vertices of M.

Construction of G_{out} is simpler. We use the iterative normal-offset method described previously, but toward the outside rather than inside the surface, to form M_{out}. Then we build the connection between M and M_{out} in the same way as between M and M_{in}.

Note that both G_{in} and G_{out} are intermediate data structures never directly viewed by the user and discarded after the user interaction. They serve only to constrain the deformation of the mesh surface. Though intersections of M_{in} and M_{out} with themselves and with M can occur, especially on meshes containing regions of high curvature, we find this causes no difficulty in our interactive system.

3.3 Deforming the Volumetric Graph

To balance between preserving the original surface's details and constraining the volume, we modify the energy function in Equation (2) to the following general form:

$$\sum_{i=1}^{n} \|\mathscr{L}_M(p_i') - \varepsilon_i'\|^2 + \alpha \sum_{i=1}^{m} \|p_i' - q_i\|^2 + \beta \sum_{i=1}^{N} \|\mathscr{L}_{G'}(p_i') - \delta_i'\|^2 \quad (3)$$

where the first n points in graph G belong to the mesh M. $\mathscr{L}_M$ is the discrete mesh Laplacian operator [Desbrun et al. 1999; Meyer et al. 2002; Sorkine et al. 2004]. G' is the sub-graph of G formed by removing those edges belonging to M. For points on the original mesh M, ε_i' $(1 \le i \le n)$ are the mesh Laplacian coordinates in the deformed coordinate frame. For points in the volumetric graph G', δ_i' $(1 \le i \le N)$ are the graph Laplacian coordinates in the deformed frame. Energy is thus decomposed into three terms corresponding to preservation of surface details, enforcement of the user's chosen deformation locations, and preservation of volumetric details/rigidity.

β balances between surface and volumetric details. We actually specify $\tilde{\beta}$ where $\beta = n\tilde{\beta}/N$. The n/N factor normalizes the weight so that it is insensitive to the lattice density of the volumetric graph. With this normalization, we find that $\tilde{\beta} = 1$ works well for preserving volume and preventing self-intersections. The α parameter is not normalized because we want the constraint strength to depend on the number of constrained points relative to the total number of mesh points. We find $0.1 < \alpha < 1$ works well for our examples. It is set to 0.2 by default.

Note that our volumetric constraint in Equation (3) could also be combined with the quadric smoothness energy in [Botsch and Kobbelt 2004]. We do not do this because we focus on deforming models with significant geometric detail.

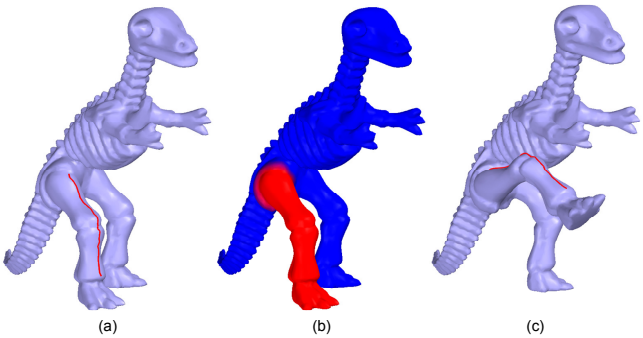

Figure 7: *Curve-based deformation. (a) original mesh and the control curve; (b) strength field (red=1, blue=0); (c) deformed mesh.*

Propagation of Local Transforms To obtain the local transforms T_i that take the Laplacian coordinates in the rest frame, δ_i and ε_i, to the new Laplacian coordinates δ_i' and ε_i' in the deformed frame, we adopt the WIRE deformation method [Singh and Fiume 1998]. A sequence of mesh vertices forming a curve is selected and then deformed to a new state. This curve controls the deformation and defines the q_i (Figure 7a).

The control curve only specifies where vertices *on the curve* deform to. The propagation algorithm first determines where neighboring graph points deform to, then infers local transforms at the curve points, and finally propagates the transforms over the whole mesh. We begin by finding mesh neighbors of the q_i and obtaining their deformed positions using WIRE. To review this method, let $C(u)$ and $C'(u)$ be the original and deformed control curves respectively, parameterized by arc length $u \in [0,1]$. Given some neighboring point $p \in R^3$, let $u_p \in [0,1]$ be the parameter value minimizing distance between p and the curve $C(u)$. The deformation maps p to p' such that C maps to C' and points nearby move analogously:

$$p' = C'(u_p) + R(u_p)\left(s(u_p)(p - C(u_p))\right).$$

$R(u)$ is a 3×3 rotation matrix which takes a tangent vector $t(u)$ on C and maps it to its corresponding tangent vector $t'(u)$ on C' by rotating around $t(u) \times t'(u)$. $s(u)$ is a scale factor. It is computed at each curve vertex as the ratio of the sum of lengths of its adjacent edges in C' over this length sum in C, and then defined continuously over u by linear interpolation.

We now have the deformed coordinates for each point on the control curve and for its 1-ring neighbors on the mesh. We proceed to compute the T_i at each point on the control curve. A rotation is defined by computing a normal and a tangent vector as the perpendicular projection of one edge vector with this normal. The normal is computed as a linear combination weighted by face area of face normals around the mesh point i. The rotation is represented as a quaternion, which means the rotation angle should be less than 180 degrees. The scale factor of T_i is given by $s(u_p)$.

The transform is then propagated from the control curve to all graph points p via a deformation strength field $f(p)$ which decays away from the deformation site (Figure 7b). Constant, linear, and gaussian strength fields can be chosen and are based on the shortest edge path (discrete geodesic distance) from p to the curve.

The simplest propagation scheme assigns to p a rotation and scale from the point q_p on the control curve closest to p. A smoother result is obtained by computing a weighted average over all the vertices on the control curve instead of the closest. Weighting by the

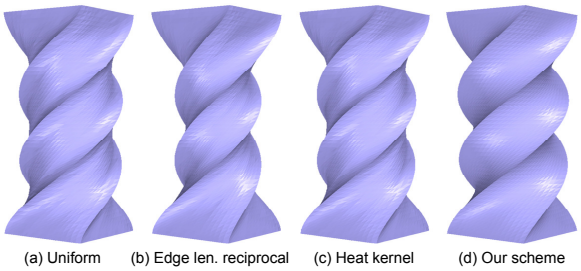

Figure 8: *Weighting schemes.*

reciprocal of distance $1/\|p-q_i\|_g$ or by a Gaussian function

$$\exp\left(-\frac{(\|p-q_i\|_g-\|p-q_p\|_g)^2}{2\sigma^2}\right)$$

works best in our experiments. $\|p-q\|_g$ denotes the discrete geodesic distance from p to q. σ controls the width of the Gaussian. Weighting between multiple curves is similar, except that the quaternion and scale must be accumulated over multiple curves.

The final transform matrix at point p is:

$$T_p = f(p)\,\tilde{T}_p + (1-f(p))\,I$$

where $\tilde{T}_p$ is p's weighted average transform. This formula simply blends that transform with the identity using the strength field. Laplacian coordinates thus approach their original (rest) state outside the deformation's influence region.

This propagation scheme is similar to the method in [Yu et al. 2004]. The difference is that we compute the transform for each graph vertex and apply it to its Laplacian coordinate. [Yu et al. 2004] compute a transform for each triangle and apply it to the triangle's vertices. Independently transforming each triangle disconnects it from its neighbors in the mesh, but solving the Poisson equation stitches triangles back together, preserving each triangle's orientation and scale in a least-squares sense. Extending this idea to a volumetric domain requires a tetrahedral mesh.

Rather than computing transforms at the deformation site and propagating them away from it, [Sorkine et al. 2004] introduce additional degrees of freedom by defining an unknown, least-squares optimal transform which takes a local neighborhood of points from the rest state to the deformed state. The transform is restricted to rotations and scales in order to prevent loss of local detail, as is the case for us too. For the system to remain quadratic and thus easily solvable, rotations are defined using the small-angle approximation. This is a poor approximation for large deformations, which then require more complicated, iterative refinement.

Weighting Scheme While uniform weighting was effective in [Sorkine et al. 2004], we find that a different scheme improves results (see Figure 8). Our geometric models come from modeling software and scanning devices; many are not uniformly tessellated. It may also be that our method of local transform propagation is more sensitive to the weighting.

For the mesh Laplacian $\mathscr{L}_M$, we use the cotangent weights [Desbrun et al. 1999]:

$$w_{ij} \propto (\cot\alpha_{ij} + \cot\beta_{ij}),$$

where $\alpha_{ij} = \angle(p_i, p_{j-1}, p_j)$ and $\beta_{ij} = \angle(p_i, p_{j+1}, p_j)$.

For the graph Laplacian $\mathscr{L}_{G'}$, we compute the weights by solving a *quadratic programming problem*. Independently for each graph vertex i, the following problem is solved to obtain the weights w_{ij} (for clarity we drop the i subscript):

$$\min_{w_j}\left(\|p_i - \sum_{j\in\mathscr{N}(i)} w_j p_j\|^2 + \lambda\left(\sum_{j\in\mathscr{N}(i)} w_j\,\|p_i - p_j\|\right)^2\right)$$

$$\text{subject to} \quad \sum_{j\in\mathscr{N}(i)} w_j = 1 \text{ and } w_j > \xi.$$

The first energy term aims at weights that generate Laplacian coordinates of smallest magnitude. The second term is based on a *scale-dependent umbrella operator* [Fujiwara 1995; Desbrun et al. 1999] which prefers weights in inverse proportion to the edge lengths. The parameter λ balances these two objectives, while the parameter ξ prevents small weights. Setting λ and ξ both to 0.01 achieves good results in all our experiments.

Figure 8 compares weighting schemes, including uniform (a), reciprocal of edge length (b), and heat kernel (decaying exponential function of squared distance) (c). Our result (d) is smoother and more uniform.

Quadric Energy Minimization Given the new Laplacian coordinates we can minimize the quadric energy in Equation (3). We solve the following equations:

$$\mathscr{L}_M(p'_i) + \beta\,\mathscr{L}_{G'}(p'_i) = \varepsilon'_i + \beta\,\delta'_i, \quad i \in 1,\ldots,n,$$

$$\beta\,\mathscr{L}_{G'}(p'_i) = \beta\,\delta'_i, \quad i \in n+1,\ldots,N,$$

$$\alpha\,p'_i = \alpha\,q'_i, \quad i \in 1,\ldots,m$$

This is a sparse linear system $Ax = b$. The matrix A is dependent only on the graph before deformation while b is also dependent on the current Laplacian coordinates and position constraints. Therefore, A is fixed as long as we do not switch the mesh or graph and the control points while b changes constantly during interactive deformation. Thus, we precompute A^{-1} using LU decomposition and dynamically execute the back substitution step to obtain $A^{-1}b$.

Multiresolution Methods Multiresolution editing can be used to accelerate differential methods, especially for very large models [Yu et al. 2004]. For example, the Stanford armadillo model contains 170K vertices. Its volumetric graph then generates almost six times as many variables as vertices in the linear system to be solved. Solving such a large system is expensive for an interactive system. To reduce computation, we use the method of [Guskov et al. 1999]. A simplified mesh with fewer vertices (15K for the armadillo) is generated. After deforming this mesh using our method, details can be added back to get the deformed high resolution mesh.

4 Deformation From 2D Curves

4.1 2D Curve-based Deformation System

The basic mode of interaction in our system is as follows. The user first specifies a curve on the mesh surface, called the *original control curve*, by selecting a sequence of mesh vertices which are connected by the shortest edge (Dijkstra) path. This 3D curve is projected onto one or more planes to obtain 2D curves which can be edited easily. After editing, the modified 2D curves are projected back to 3D to get the *deformed control curve*, which forms the basis for deformation of the previous section.

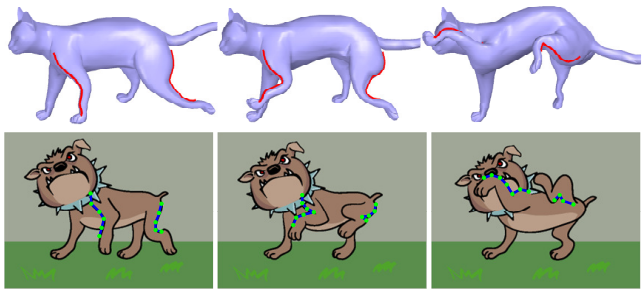

Figure 9: *Deformation retargeting. Two control curves in red are shown on the 3D mesh (top row). Their deformation is driven by cartoon curves selected by the user and shown in the bottom row.*

Curve Projection Given the original control curve, the system automatically selects the projection plane based on its average normal and principal vectors. The principal vectors are computed as the two eigenvectors corresponding to the largest eigenvalues from a principal component analysis (PCA) over the curve's vertices. In most cases, the cross product of the average normal and the first principal vector provides a satisfactory plane. When the length of the average normal vector is small, as for a closed planar curve, we use the two principal vectors instead. The user can also directly choose or modify the projection chosen by the system.

Curve Editing Projected 2D curves inherit geometric detail from the original mesh which complicates editing. Multiresolution curve editing [Finkelstein and Salesin 1994] provides one solution for B-spline curves. We use an editing method for discrete curves based on Laplacian coordinates [Sorkine et al. 2004]. The Laplacian coordinate of a curve vertex is the difference between its position and the average position of its two adjacent neighbors, or single neighbor for terminal vertices.

The discrete 2D curve to be edited is denoted as C. A cubic B-spline curve C_b is first computed as a least-squares fit to C, representing C's low frequencies. Then a discrete version of C_b, denoted C_d, is computed by mapping each vertex of C onto C_b using proportional arc-length mapping. The simple B-spline curve C_b can now be edited conveniently. After editing, we obtain the modified B-spline curve C_b' and a new discrete version C_d'. These curves indicate the user's desired deformation but lack the original curve's detail. To restore it, at each vertex of C we find the unique rotation and scale that map its location from C_d to C_d'. Applying this transformation to the Laplacian coordinates on C and solving equation (2) (without the second point constraint term) over the simple curve graph generates a new curve C' which preserves the details of C but follows the user's coarse-scale modification. Essentially, this is a trivial version of local transform "propagation" for deforming a mesh, but for the curve case it can be defined independently per point and need not be propagated.

This algorithm can be posed as a curve analogy [Hertzmann et al. 2002]: given a pair of source curves, C_s and C_s', and a target curve C, generate a new curve C' such that the relationship between C and C' is analogous to the relationship between C_s and C_s'. While [Hertzmann et al. 2002] transforms the vertices directly, we transform their differential coordinates, which distributes errors more uniformly across the whole curve.

Initially, C_b has two knots at the curve endpoints. The user can add knots to perform editing at a more detailed level. Our system supports two editing modes: one manipulates a coarse-scale fit to the original curve, C_b, and the other sketches an entirely new curve. In the latter case, correspondence between the sketched curve and the control curve is achieved by arclength by default. The user can also specify a series of corresponding points between the two curves.

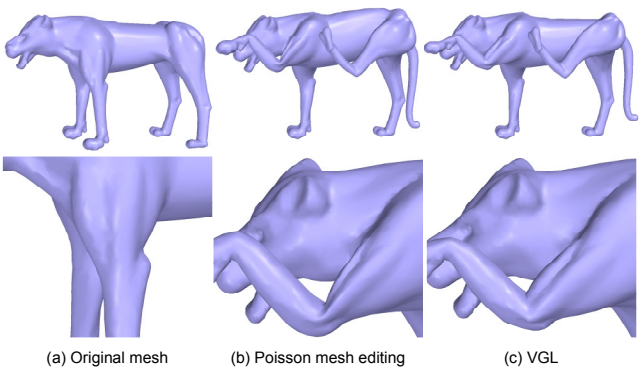

Figure 10: *Large deformation comparison.*

4.2 Deformation Retargeting From 2D Cartoons

Retargeting the deformation of 2D cartoons to 3D meshes is a direct application of our 2D sketch-based deformation. Users specify one or more 3D control curves on the mesh along with their projection planes and, for each curve, a series of 2D curves in the cartoon image sequence that drive its deformation (see Figure 9). Suppose that C_i is the projection of a 3D control curve, and its corresponding curves in the cartoon sequence are $C_{i,j}, j \in \{1,...,k\}$. The index i is for different control curves, driving different parts of the model like arms and legs. The index j is the frame index – the same control curve has a corresponding cartoon curve for each frame. Our system automatically derives a deformation sequence mapping C_i to successive $C_{i,j}$.

Two details require further explanation. First, it is not necessary to generate a deformation from scratch at every frame. Users can select just a few key frames and specify cartoon control curves just for these rather than the entire sequence. An automatic interpolation technique based on differential coordinates [Alexa 2003] is then used to interpolate between key frames. Suppose we have two meshes M and M' with the same connectivity, representing the deformed mesh at two key frames. We begin by computing the Laplacian coordinates for each vertex on the two meshes. A rotation and scale in the local neighborhood of each vertex p is computed taking the Laplacian coordinate from its location in M to M' (see Section 3.3). Denote the transformation by T_p. By interpolating each transformation from the identity to T_p over time, we get a smoothly varying Laplacian coordinate from M to M'. Solving equation (2) provides a sequence of meshes from M to M'.

Second, the 2D cartoon curves only specify how the deformed curve projects in a single plane, leaving unspecified its shape perpendicular to the plane. We therefore allow users to select other projection planes to specify these extra degrees of freedom, if desired.

5 Experimental Results

We have experimented with large deformations on models from scanning devices (armadillo and dinosaur) and modeling software (dog, cat and lioness). With surface-based methods like the Poisson mesh [Yu et al. 2004], pinching and other artifacts happen frequently as models are deformed. Our technique eliminates these artifacts as shown in Figures 1 and 10.

Our 2D curve-based deformation system has an intuitive interface (see the accompanying video) that makes it easy to drive 3D deformations from 2D cartoons. Figures 9, 11, 12 and 13 show deformation retargeting results from cartoon characters. We do not aim at deforming the 3D model into precisely the same pose as the 2D cartoon's. This is difficult because their shapes are so different

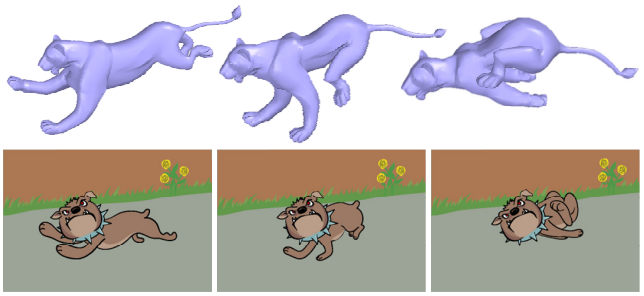

Figure 11: *Deformation transfer from a running dog to a lioness.*

	arma	dino	cat	lioness	dog
# mesh vertices	15,002	10,002	7,207	5,000	10,002
# graph points	28,142	15,895	14,170	8,409	17,190
graph generation	2.679s	1.456s	1.175s	1.367s	1.348s
LU decomposition	0.524s	0.286s	0.348s	0.197s	0.118s
back substitution	0.064s	0.028s	0.030s	0.019s	0.011s
# control curves	6	5	4	5	
# key frames	10	9	8	8	
session time (min)	∼120	∼90	∼30	∼90	

Table 1: *Statistics and timing.*

and because cartoons are drawings that may not be reflective of the motion of 3D geometry. Instead, our goal is to transfer the quality of the cartoon's motion to the 3D model. As the animations in the accompanying video show, we successfully obtain motions that are remarkably similar to the cartoon's.

Table 1 shows the data statistics and timings for models presented in this paper. The timing is measured on a 3.0 GHz Intel Pentium 4 workstation. The session time for deformation transfer varies from about half an hour to two hours for an untrained graduate student, and depends on the number of control curves and image key frames.

6 Conclusion and Future Work

Differential domain methods preserve surface details as a mesh is deformed but produce objectionable pinching and intersection artifacts when the deformation is large. We solve this problem by preserving volumetric details represented by the volumetric graph Laplacian. Our solution avoids the intricacies of solidly meshing complex objects. We show the value of this idea by building a "Teddy-like" system that allows novice users to easily specify mesh deformations, and to re-target cartoon motions to complicated 3D models.

Automatically inferring good local transforms for point-based rather than curve-based deformation is an area for future work. Our current system does support a limited form of point-based deformation. If only a single point moves, we set the local transform to the identity everywhere because our method of inferring local transforms using WIRE depends on an original and deformed curve pair. This works well for small deformations. For large deformations, the results are often poor because details are not preserved in the expected local coordinate frame but are sheared along the vector between the original and deformed point.

We currently fix graph connectivity during a deformation. Adaptive connectivity [Kobbelt et al. 2000] is necessary for very large deformations and represents an area of future work. Another enhancement would be to automatically track curves in cartoon sequences.

Acknowledgements

We would like to thank Stanford University, Cyberware and Robert W. Sumner (MIT) for providing the 3D models, and Disney Feature Animation for giving us the permission to use and reprint some of the famous Disney Animation Cells. The cartoon drawings shown in Figures 9, 11 and 13 are provided by Dongyu Cao. Thanks to Xin Sun and Jianwei Han for using our system to create the deformation results. Thanks to Steve Lin, Xin Sun and Bo Zhang for their help in video production. The authors are grateful to the anonymous reviewers for their helpful suggestions and comments. Hujun Bao was supported by NSFC (No. 60021201 and 60033010) and 973 Program of China (No. 2002CB312102).

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *SIGGRAPH 2000 Conference Proceedings*, 157–164.

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

BARR, A. 1984. Global and local deformations of solid primitives. *SIGGRAPH 84 Conference Proceedings 18*, 3, 21–30.

BENDELS, G. H., AND KLEIN, R. 2003. Mesh forging: editing of 3d meshes using implicitly defined occluders. In *Symposium on Geometry Processing*, ACM SIGGRAPH / Eurographics, 207–217.

BOTSCH, M., AND KOBBELT, L. 2003. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum 22*, 3.

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform-modeling. *ACM Trans. on Graphics 23*, 3, 630–634.

BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. In *SIGGRAPH 2002 Conference Proceedings*, 399–407.

BRIDSON, R., TERAN, J., MOLINO, N., AND FEDKIW, R. 2004. Adaptive physics based tetrahedral mesh generation using level sets. *Engineering with Computers*, to appear.

CHUNG, F. R. K. 1997. Spectral graph theory. *CBMS 92*, AMS.

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*, 119–128.

COQUILLART, S. 1990. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. *SIGGRAPH 90 Conference Proceedings 24*, 4, 187–196.

CUTLER, B., DORSEY, J., AND MCMILLAN, L. 2004. Simplification and improvement of tetrahedral models for simulation. In *Symposium on Geometry Processing*, ACM SIGGRAPH / Eurographics, 95–104.

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

FAVREAU, L., REVERET, L., DEPRAZ, C., AND CANI, M.-P. 2004. Animal gaits from video. In *Symposium on Computer Animation*, ACM SIGGRAPH / Eurographics.

FINKELSTEIN, A., AND SALESIN, D. H. 1994. Multiresolution curves. In *SIGGRAPH 94 Conference Proceedings*, 261–268.

FUJIWARA, K. 1995. Eigenvalues of laplacians on a closed riemannian manifold and its nets. In *Proceedings of AMS 123*, 2585–2594.

GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *SIGGRAPH 99 Conference Proceedings*, 325–334.

HERTZMANN, A., OLIVER, N., CURLESS, B., AND SEITZ, S. M. 2002. Curve analogies. In *Proceedings of the 13th Eurographics Workshop on Rendering*, 233–245.

HIROTA, G., MAHESHWARI, R., AND LIN, M. C. 1999. Fast volume preserving free form deformation using multi-level optimization. In *Proceedings of Solid Modeling and Applications*, 234–245.

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

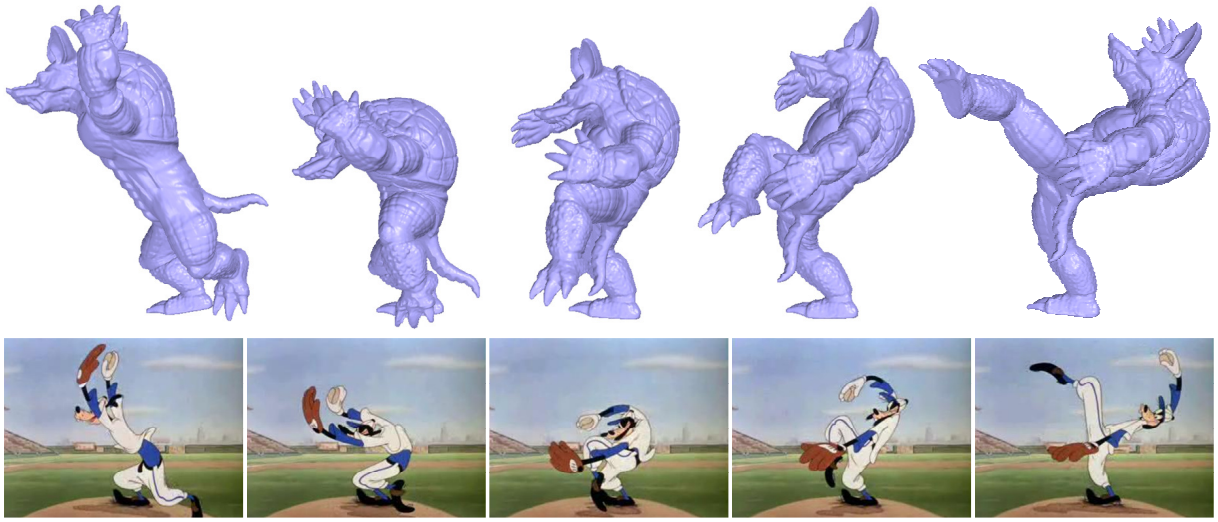

Figure 12: *Deformation transfer from Goofy to armadillo. ©Disney*

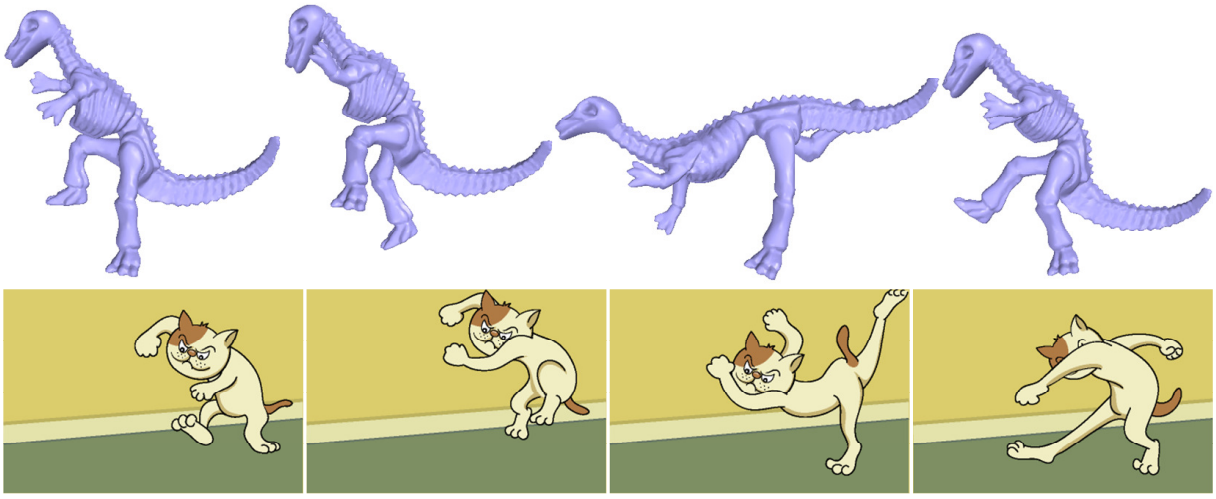

Figure 13: *Deformation transfer from a kicking cat to dinosaur.*

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *SIGGRAPH 99 Conference Proceedings*, 409–416.

KHO, Y., AND GARLAND, M. 2005. Sketching mesh deformations. In *Proceedings of the ACM Symposium on Interactive 3D Graphics*.

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.

KOBBELT, L., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum 19*, 3, 249–260.

LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*, IEEE Computer Society Press, 181–190.

MACCRACKEN, R., AND JOY, K. 1996. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH 96 Conference Proceedings*, 181–188.

MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. 2002. Discrete differential-geometry operators for triangulated 2-manifolds. In *Proc. VisMath*.

MILLIRON, T., JENSEN, R., BARZEL, R., AND FINKELSTEIN, A. 2002. A framework for geometric warps and deformations. *ACM Trans. Graphics 21*, 1, 20–51.

OWEN, S. J. 1998. A survey fo unstructured mesh generation technology. In *7th International Mehsing Roundtable*, 239–267.

RAPPOPORT, A., SHEFFER, A., AND BERCOVIER, M. 1995. Volume preserving free-form solid. In *Proceedings of Solid modeling and applications*, 361–372.

SEDERBERG, T., AND PARRY, S. 1986. Free-form deformation of solid geometric models. *SIGGRAPH 86 Conference Proceedings 20*, 4, 151–160.

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

SHEWCHUK, J. R. 1998. Tetrahedral mesh generation by delaunay refinement. In *Proceedings of the 14th Annual Symposium on Computational Geometry*, 86–95.

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

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Symposium on Geometry Processing*, ACM SIGGRAPH / Eurographics, 179–188.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. on Graphics 23*, 3, 399–405.

TAUBIN, G. 1995. A signal processing approach to fair surface design. In *SIGGRAPH 95 Conference Proceedings*, 351–358.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *SIGGRAPH 94 Conference Proceedings*, 247–256.

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. on Graphics 23*, 3, 644–651.

ZELINKA, S., AND GARLAND, M. 2004. Mesh modelling with curve analogies. In *Proceedings of Pacific Graphics*, 94–98.

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