Mesh Editing with Poisson-Based Gradient Field Manipulation

Yizhou Yu[†] Kun Zhou[‡] Dong Xu[‡§] Xiaohan Shi [‡§] Hujun Bao[§] Baining Guo[‡] Heung-Yeung Shum[‡]

[†] University of Illinois at Urbana-Champaign [‡] Microsoft Research Asia [§] Zhejiang University

Abstract

In this paper, we introduce a novel approach to mesh editing with the Poisson equation as the theoretical foundation. The most distinctive feature of this approach is that it modifies the original mesh geometry implicitly through gradient field manipulation. Our approach can produce desirable and pleasing results for both global and local editing operations, such as deformation, object merging, and smoothing. With the help from a few novel interactive tools, these operations can be performed conveniently with a small amount of user interaction. Our technique has three key components, a basic mesh solver based on the Poisson equation, a gradient field manipulation scheme using local transforms, and a generalized boundary condition representation based on local frames. Experimental results indicate that our framework can outperform previous related mesh editing techniques.

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

Keywords: Poisson Equation, Local Transform Propagation, Mesh Deformation, Object Merging, Mesh Filtering

1 Introduction

Mesh editing has long been an active research area in computer graphics. Most existing techniques, such as deformation, Boolean operations, detail editing and transfer, manipulate vertex positions explicitly. The user needs to be extremely careful to avoid artifacts in the resulting mesh. In this paper, we introduce a new mesh editing technique based on gradient field manipulation which implicitly modifies vertex positions. This technique is capable of producing desirable results with a small amount of user interaction.

The theoretical foundation of our technique is the Poisson equation which is able to reconstruct a scalar function from a guidance vector field and a boundary condition. The Poisson equation can also be viewed as an alternative formulation of a least-squares minimization. With these appealing characteristics, editing a function can be achieved by modifying its gradient field and boundary condition, and a succeeding reconstruction using the Poisson equation. The motivation of this approach is twofold. First, the gradient is a differential property that can be modified locally. Subsequent reconstruction from the modified gradient can give rise to a global effect which would otherwise require a larger amount of user interaction. Secondly, artifacts introduced during local editing can be removed during reconstruction because least-squares minimization tends to distribute errors uniformly across the function.

[‡]This work was done while Dong Xu and Xiaohan Shi were interns at Microsoft Research Asia.

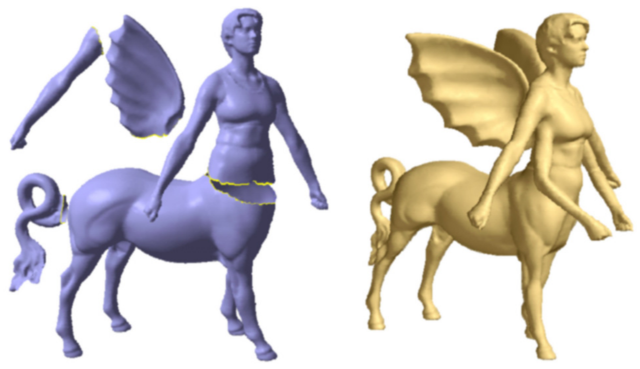
Figure 1: An unknown mythical creature. Left: mesh components for merging and deformation (the arm), Right: final editing result.

Despite the frequent appearance of the Poisson equation in various computational frameworks [Stam 1999; Perez et al. 2003], applying this equation to mesh editing is nontrivial. The unknown in the Poisson equation is always a scalar function while a mesh can be considered as a vector function that has unique differential properties, such as normals and curvature, with geometric interpretations. How can we apply the Poisson equation to mesh geometry? The Poisson equation involves gradient fields and a boundary condition which are mathematical concepts. It is not obvious how to modify these two types of information to achieve desirable mesh editing effects.

Our key contribution is a mesh processing engine based on gradient field manipulation and boundary condition editing. The engine is equipped with novel techniques designed to overcome the aforementioned difficulties. First, we regard the mesh geometry (coordinates) as scalar functions defined on a mesh surface and introduce the Poisson equation as a basic mesh solver. Second, we design a gradient editing scheme for these scalar functions using local transformations. Third, we propose generalized boundary conditions enforced by propagating local frame changes. These techniques will be elaborated in Section 3.

Our mesh processing engine has been successfully applied to mesh deformation, merging, as well as anisotropic smoothing. These operations have been integrated into a mesh editing system with a few novel interactive tools. Our system exhibits several desirable features. Both large-scale and detailed deformation can be achieved conveniently by locally manipulating a curve or vertex on the mesh. Merging meshes with drastically different open boundaries have been made easier. The shapes of the merged meshes are globally adjusted to be more compatible with each other. Details of these applications will be introduced in Section 4.

2 Background and Related Work

The Poisson Equation. Originally emerging from Isaac Newton's law of gravitation [Tohline 1999], the Poisson equation with Dirichlet boundary condition is formulated as

$$\nabla^2 f = \nabla \cdot \mathbf{w}, \quad f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{1}$$

where f is an unknown scalar function, $\mathbf{w}$ is a *guidance* vector field, f^* provides the desirable values on the boundary $\partial\Omega$, $\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ is the Laplacian operator, $\nabla \cdot \mathbf{w} = \frac{\partial w_x}{\partial x} + \frac{\partial w_y}{\partial y} + \frac{\partial w_z}{\partial z}$ is the divergence of $\mathbf{w} = (w_x, w_y, w_z)$.

Vector Field Decomposition. The Poisson equation is closely related to Helmholtz-Hodge vector field decomposition [Abraham et al. 1988] which uniquely exists for a smooth 3D vector field $\mathbf{w}$ defined in a region Ω:

$$\mathbf{w} = \nabla\phi + \nabla \times \upsilon + \mathbf{h}, \tag{2}$$

where ϕ is a scalar potential field with $\nabla \times (\nabla\phi) = 0$, υ is a vector potential field with $\nabla \cdot (\nabla \times \upsilon) = \mathbf{0}$, and $\mathbf{h}$ is a field that is both divergence and curl free. The uniqueness of this decomposition requires proper boundary conditions. The scalar potential field ϕ from this decomposition happens to be the solution of the following least-squares minimization

$$\min_{\phi} \int\int_{\Omega} \|\nabla\phi - \mathbf{w}\|^2 dA, \tag{3}$$

whose solution can also be obtained by solving a Poisson equation, $\nabla^2\phi = \nabla \cdot \mathbf{w}$.

Discrete Fields and Divergence. A prerequisite of solving the Poisson equation over a triangle mesh is to overcome its irregular connectivity in comparison to a regular image or voxel grid. One recent approach to circumvent this difficulty is to approximate smooth fields with discrete fields first and then redefine the divergence for the discrete fields [Polthier and Preuss 2000; Meyer et al. 2002; Tong et al. 2003]. A discrete vector field on a triangle mesh is defined to be a piecewise constant vector function whose domain is the set of points on the mesh surface. A constant vector is defined for each triangle, and this vector is coplanar with the triangle. A discrete potential field on a triangle mesh is defined to be a piecewise linear function, $\phi(\mathbf{x}) = \sum_i B_i(\mathbf{x})\phi_i$, with B_i being the piecewise-linear basis function valued 1 at vertex $\mathbf{v_i}$ and 0 at all other vertices, and ϕ_i being the value of ϕ at $\mathbf{v_i}$. For a discrete vector field $\mathbf{w}$ on a mesh, its divergence at vertex $\mathbf{v_i}$ can be defined to be

$$(\mathrm{Div}\mathbf{w})(\mathbf{v_i}) = \sum_{T_k \in N(i)} \nabla B_{ik} \cdot \mathbf{w}|T_k| \tag{4}$$

where $N(i)$ is the set of triangles sharing the vertex $\mathbf{v_i}$, $|T_k|$ is the area of triangle T_k, and ∇B_{ik} is the gradient vector of B_i within T_k. Note that this divergence is dependent on the geometry and 1-ring structures of the underlying mesh.

Mesh deformation and editing. Mesh processing includes a large variety of operations. We are going to focus on three areas most relevant to this paper, mesh deformation, object merging and mesh detail editing.

Mesh deformation can be classified as lattice-based free-form deformation (FFD) [Sederberg and Parry 1986; Coquillart 1990; MacCracken and Joy 1996], curve-based [Barr 1984; Chang and Rockwood 1994; Lazarus et al. 1994; Singh and Fiume 1998], or point-based [Hsu et al. 1992; Bendels and Klein 2003]. One representative curve-based method is WIRE [Singh and Fiume 1998] which directly attaches curves to mesh surfaces to achieve deformations. [Milliron et al. 2002] presents a general framework for geometric warps and deformations. [Llamas et al. 2003; Bendels and Klein 2003] demonstrate that the use of rotations in addition to translations can achieve certain deformations much more conveniently. [Pauly et al. 2003] supports multiple shape editing operations on point-sampled geometry which is not the focus of this paper.

Object modeling and deformations can be performed at various resolutions to achieve both global control and local editing [Kobbelt et al. 1998; Guskov et al. 1999; Kobbelt et al. 2000]. In particular, [Kobbelt et al. 1998] introduces a mesh deformation technique by solving a constrained minimization of the thin-plate energy at a desirable coarse resolution. The user specifies deformation constraints through a handle polygon. Original mesh details are added back to the resulting smooth mesh to produce a final solution. This technique only gives the user limited control over the mesh shape through sparse constraints on the handle polygon. The rest of the mesh geometry is uniquely determined by the minimization. In contrast, the method in this paper can achieve better shape control by specifying guidance vector fields as dense constraints over the editable mesh region. It does not require a multiresolution mesh representation which is only used for acceleration.

Mesh deformation is closely related to shape interpolation and morphing. [Alexa et al. 2000] introduces a shape interpolation technique for simplicial complexes. It considers the interiors of the given shapes and minimizes distortion in local volumes. It is nontrivial to generalize this technique to mesh deformation since triangle meshes are not simplicial complexes in three-dimensional spaces. With additional constraints imposed on admissible local transformations, [Sorkine et al. 2004] proposes such a generalization using Laplacian coordinates.

Boolean operations are often applied to obtain new models from a set of original ones [Biermann et al. 2001]. Continuity at intersection curves can be improved by local blending or smoothing. [Museth et al. 2002] applied the level set method to such tasks. The method in [Lévy 2003] can also perform merging by extrapolating parameterizations. In this paper, we perform more general object merging without 2D parameterizations by connecting objects at their open boundaries which may have very different shapes.

There have also been various techniques for mesh detail editing. A signal processing approach was presented in [Taubin 1995]. Methods [Taubin 2001; Desbrun et al. 2000; Meyer et al. 2002; Tasdizen et al. 2002; Bajaj and Xu 2003; Yagou et al. 2003; Fleishman et al. 2003; Jones et al. 2003] have been developed to remove noise while preserving important features by generalizing anisotropic diffusion [Perona and Malik 1990] onto meshes. Local details can be smoothed or sharpened by using curvature flows and the level set method [Museth et al. 2002].

3 A Framework for Poisson Mesh Editing

Given the definitions of discrete fields and their divergence introduced in Section 2, the discrete Poisson equation [Tong et al. 2003] can be expressed as

$$\mathrm{Div}(\nabla\phi) = \mathrm{Div}\mathbf{w}, \tag{5}$$

which is actually a sparse linear system,

$$\mathbf{Af} = \mathbf{b}, \tag{6}$$

that can be solved numerically using the conjugate gradient method. We still call (5) the Poisson equation for convenience. Note that the unknown in the discrete Poisson equation is still a scalar potential field. It would be straightforward to exploit (5) for surface properties defined on a mesh, such as textures.

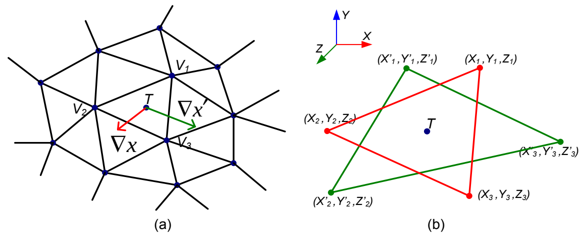

Figure 2: (a) A parameterization mesh, (b) a triangle (red) in the original mesh and its locally transformed version (green). Each of its coordinates is a scalar function on $\triangle V_1V_2V_3$ in (a). In addition, vertex (X_i, Y_i, Z_i) corresponds to V_i. ∇x (red) and $\nabla x'$ (green) in (a) are the gradient vectors of the x-component of the original and transformed triangles, respectively. They are coplanar with $\triangle V_1V_2V_3$.

How can we use the discrete Poisson equation to solve or modify the mesh geometry itself? Let us describe the fundamentals of this problem as well as our solution to boundary condition editing.

3.1 A Basic Poisson Mesh Solver

To apply the discrete Poisson equation to mesh processing, we need to consider the three coordinates of a *target* mesh as three scalar fields defined on a *parameterization* mesh [1]. Since triangle meshes are piecewise linear models, such scalar fields are actually piecewise linear, and satisfy the definition of discrete potential fields. The target and parameterization meshes should have the same topology (vertex connectivity), and their vertices should have one-to-one correspondence.

The purpose of applying the Poisson equation is to solve an unknown target mesh with known topology but unknown geometry (vertex coordinates). To obtain the unknown vertex coordinates, the Poisson equation requires a discrete guidance vector field for each of the three coordinates. Once a discrete guidance vector field is introduced over the parameterization mesh, its divergence, defined in (4), at a vertex of the parameterization mesh can be computed. The vector $\mathbf{b}$ in (6) is obtained from the the collection of divergence values at all vertices. The coefficient matrix $\mathbf{A}$ in (6) is independent of the guidance field, and can be obtained using the parameterization mesh only. The resulting linear system is solved to obtain one specific coordinate for all vertices simultaneously. This process is repeated three times to obtain the 3D coordinates of all vertices. This whole process looks like ”mesh cloning”, and the guidance fields encode the desired properties of the target mesh.

In principle, different parameterization meshes give rise to different target meshes. Due to the nature of the least-squares minimization in (3), the general rule is that guidance vectors associated with larger triangles in the parameterization mesh are better approximated than those associated with smaller triangles. The areas of the triangles serve as the weighting scheme. During mesh editing, the original mesh is given and the goal is to obtain an edited mesh. Therefore, it is most convenient to treat the original mesh as the parameterization mesh and the edited one as the target mesh without any 2D parameterizations.

[1]The concept of a parameterization mesh has previously been used in [Taubin 1995; Karni and Gotsman 2000], however, not in the context of the Poisson equation. In addition, their parameterization meshes only have topology while ours are real 3D meshes.

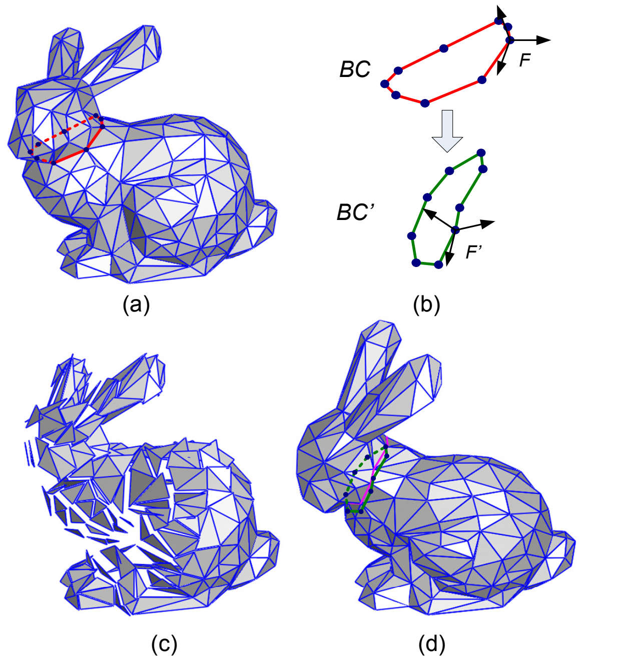

Figure 3: (a) A BUNNY mesh with a curve around its neck. (b) This curve is a generalized boundary condition, BC, for the Poisson equation. Its edited version is BC'. Local frame changes on the curve are propagated to other triangles. (c) Each triangle is locally transformed by the transformation it receives from the propagation. The triangles become disconnected. (d) The Poisson equation stitches together the triangles again in the new pose defined by BC'.

3.2 Gradient Field Editing Using Local Transforms

In this section, we discuss mesh editing by manipulating its original gradient fields. The Poisson equation relies on both guidance vector fields and boundary conditions. Our key observation is that if we edit the boundary conditions while keeping the original gradient fields of a mesh as the guidance fields, most part of the resulting mesh will not reflect the changes in the boundary conditions, causing undesirable artifacts as shown in the middle image of Fig. 7. Therefore, we need to alter the original gradient fields.

Gradient editing is achieved by applying local transformations to the triangles and obtaining new gradient vectors of the altered triangles. Note a triangle has three gradient vectors for three coordinates. A local transformation is defined on a per triangle basis. The three vertex positions of a triangle are altered by the same transformation (Fig. 2). Since gradient vectors are translation invariant, the local transformation is typically carried out in a canonical local frame at the center of the triangle. Examples of transformations include rotation and scaling. The new gradient vectors define three new vector fields over the parameterization mesh. Since the local transform applied to each triangle may be different, the original mesh is torn apart and the triangles are not connected any more (Fig. 3(c)). Therefore, the new vector fields are not likely to be gradient fields of any scalar functions. To reconstruct a mesh from these vector fields, we need to consider them as the guidance fields in the Poisson equation. Given a parameterization mesh and three guidance vector fields, the steps in the previous section can be followed to reconstruct the new target mesh. Intuitively, solving the Poisson equation is analogous to stitching together the previously disconnected triangles again.

Manipulating mesh gradient fields is a key component of our mesh editing system. This editing mode will be exploited in the following section as well as Section 4.3. Note that it would be tedious to interactively define a local transform for every triangle of a mesh. Automatic schemes to obtain such local transforms will be discussed wherever gradient manipulation is needed.

3.3 Boundary Condition Editing

We would like to achieve local or global mesh editing by conveniently manipulating a small number of local features such as curves or vertices. In this section, we discuss how to satisfy such editing requests. Details about user interaction will be discussed in Section 4. In terms of the Poisson equation, both a curve or a vertex anywhere on a mesh is a boundary condition in the sense that a unique solution to the Poisson equation exists because this equation is translation invariant. There is still something more to meshes than to scalar functions. Geometrically, a set of neighboring vertices on a mesh provides information such as normal orientation, curvature and scale in addition to the vertex positions themselves. Therefore, there is a need to generalize the concept of a boundary condition for a mesh.

We formally define a generalized boundary condition of a mesh as a combination of five components $BC = (I, P, F, S, R)$ where I is the index set of a set of connected vertices on the mesh, P is the set of 3D vertex positions, F is a set of local frames which define the local orientations of the vertices, S is the set of scaling factors associated with the vertices, and R is a strength field. A vertex is *constrained* if it belongs to at least one boundary condition; otherwise, it is a *free* vertex. As usual, a local frame at a vertex is defined by three orthogonal unit vectors one of which should be the unit normal. If a vertex belongs to a curve, its local frame should be defined by the normal, the tangent and the binormal of the curve. The scaling factor at a vertex only reflects the scale change before and after an editing operation, and can be initialized to one at the beginning. The strength field defines the influence of the boundary condition at every free vertex. The influence (strength) is a function of the minimal distance between the free vertex and the constrained vertices in the boundary condition. All free vertices receiving a nonzero strength define the *influence region* of the boundary condition.

Once a boundary condition $BC = (I, P, F, S, R)$ needs editing, we create a modified version $BC' = (I, P', F', S', R)$ (Fig. 3(b)). A constrained vertex position $\mathbf{v}_c \in P$ may have a different position, local frame and scale in BC'. The difference between the new and old local frames at $\mathbf{v}_i$ can be uniquely determined by a single rotation which is represented as a unit quaternion in practice. The difference in scale is represented as a ratio. Thus, each constrained vertex in BC has its associated quaternion and ratio to represent the local frame and scale changes.

We propagate the local frame and scale changes from the constrained vertices to all the free vertices in the influence region to create a smooth transition. When there is only one single boundary condition, $BC_0 = (I_0, P_0, F_0, S_0, R_0)$, and its edited version, BC'_0, we first compute the geodesic distance from each free vertex, denoted by $\mathbf{v}_f$, in the original mesh to all the constrained vertices in BC_0 [2]. Suppose $\mathbf{v}_{min}$ is the constrained vertex in BC_0 that is closest to $\mathbf{v}_f$. That is, $\mathbf{v}_{min} = \arg\min_{\mathbf{v}_c \in P_0} dist(\mathbf{v}_f, \mathbf{v}_c)$. The simplest scheme, called the *Nearest* scheme, directly assigns the quaternion and scale ratio at $\mathbf{v}_{min}$ to $\mathbf{v}_f$. In practice, smoother results can be obtained by assigning to $\mathbf{v}_f$ the weighted average of the quaternions and scale ratios at all constrained vertices in BC_0. We designed three weighting schemes: *Uniform*, *Linear*, and *Gaussian*. In the Uniform scheme, the transforms from all constrained vertices in BC_0 are weighed equally. In the Linear scheme, the transform from a constrained vertex, $\mathbf{v}_c$, in BC_0 is weighed by the inverse of $dist(\mathbf{v}_f, \mathbf{v}_c)$. In the Gaussian scheme, the transform from a constrained vertex, $\mathbf{v}_c$, in BC_0 is weighed by the Gaussian function $\exp\left(-\frac{(dist(\mathbf{v}_f,\mathbf{v}_c)-dist(\mathbf{v}_f,\mathbf{v}_{min}))^2}{2\sigma_d^2}\right)$, where σ_d is a user-specified parameter to indicate the width of the Gaussian. In our experiments, the Linear and Gaussian weighting schemes typically produce better results.

When there are multiple boundary conditions, $BC_i, i = 1, ..., m$, a free vertex $\mathbf{v}_f$ receives a quaternion $\mathbf{q}_i$ from each of the boundary conditions. We define a weight w_i for each quaternion $\mathbf{q}_i$ using the strength of BC_i at $\mathbf{v}_f$. The strength of a boundary condition in its influence region can be constant, linearly decreasing or a cosine wave function. The final quaternion assigned to $\mathbf{v}_f$ is a weighted average, $\frac{\sum_i w_i q_i}{\sum_i w_i}$ [3]. The final scale ratio can be defined similarly using the geometric mean.

We proceed to define a local transform for each triangle in the mesh. An average quaternion based on the three quaternions at the three vertices represents the rotation component. The scale ratio represents the scaling factor. Both rotation and scaling can be integrated into a single linear transform applied to the triangle as in Section 3.2 to obtain new guidance vectors. The new mesh geometry obtained using the solver in Section 3.1 best approximates the orientations and scales imposed by the modified generalized boundary conditions (Fig. 3(d)).

[2]This is actually a distance transform that can be computed by the level set method [Sethian 1999].

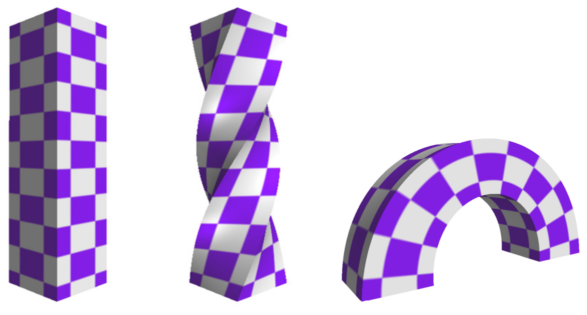

Figure 4: (a) Original model (2040 vertices and 4000 faces), (b) twisting by rotating the top rectangular boundary around the vertical axis of the PRISM (running time = 578 ms), (c) bending by rotating the top boundary around a horizontal axis in addition to a translation (running time = 609 ms).

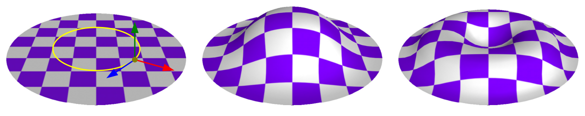

Figure 5: (a) Original model (1281 vertices and 2480 faces), (b)-(c) simultaneous normal rotation around their respective tangents using cosine functions with two different phase angles as their strength fields. The running time for (b) is 230ms and (c) 240ms.

[3]Note that we frequently use (weighted) averages of quaternions. Since quaternions are not commutative, such a weighted average is implemented by a sequence of spherical-linear interpolations in a fixed order. For example, $\frac{w_1\mathbf{q}_1+w_2\mathbf{q}_2+w_3\mathbf{q}_3}{w_1+w_2+w_3}$ is actually interpreted as $\frac{w_1+w_2}{w_1+w_2+w_3}\left(\frac{w_1}{w_1+w_2}\mathbf{q}_1 + \frac{w_2}{w_1+w_2}\mathbf{q}_2\right) + \frac{w_3}{w_1+w_2+w_3}\mathbf{q}_3$.

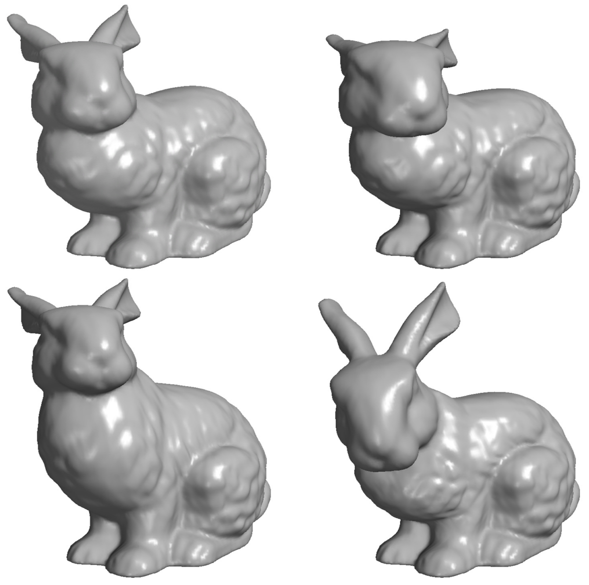

Figure 6: Interactive mesh deformation. The top row, from left to right, shows the original model and the result from rotating normals of a curve around their respective tangents. The bottom row shows the results by applying a translation or rotation to the whole curve. The curve is around the neck, and the weighting scheme for all the constrained vertices on the curve is Gaussian.

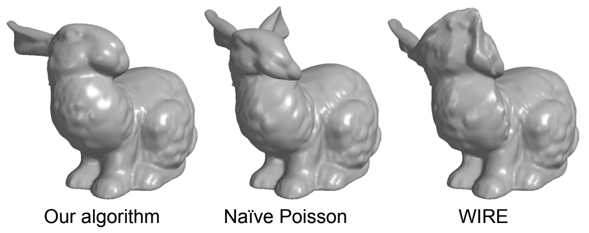

Figure 7: Deformation comparison. From left to right are results from our algorithm, naive Poisson editing and WIRE, respectively.

4 Applications

4.1 Mesh Deformation

Since boundary conditions are a powerful means to influence the shape of an object, we implemented a method to perform mesh deformation through interactive boundary condition editing. For the convenience of user control, we only adopt open or closed *curves* or single *vertices* as boundary conditions. We further distinguish *fixed* boundary conditions from *editable* boundary conditions during an interactive session. Fixed boundary conditions include vertices on the mesh that the user wishes to hold still during the whole session while editable boundary conditions include vertices that the user wishes to modify through direct manipulation. The rest are free vertices whose positions and local frames are indirectly controlled by both types of boundary conditions as discussed in Section 3.3. Note that vertices in fixed boundary conditions have the identity matrix as their local transform.

The vertices on the same editable curve can be modified either individually or simultaneously. Individual editing is able to change small scale details on a mesh while simultaneous editing is a very powerful operation that can introduce large scale deformations via minimal user interaction. Our system supports two important types

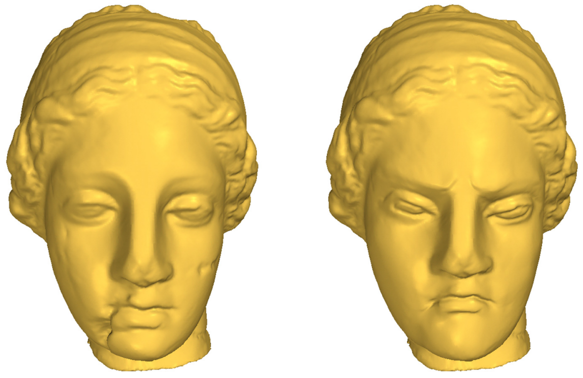

Figure 8: Left: the original Cyberware Igea model; Right: the edited model. Detail editing is applied to the eyes, eyebrows and lips to change the facial expression. Local smoothing is applied to the cheeks and the groove on the lower left part of the face.

of simultaneous editing operations:

• Simultaneous translation, rotation and/or scaling applies the same transformation to all the vertices on the same curve. Translation only changes vertex positions while rotation and scaling need to change local frames and scaling factors as well. Fig. 4 shows a PRISM deformed by simultaneous editing applied to the rectangular boundary on the top.

• Simultaneous rotation of all the vertex normals around their respective tangent directions by the same degree. Since the tangent directions at the vertices differ, the resulting quaternions also differ. Fig. 5(b)&(c) show a circular disk deformed by simultaneous normal rotation around a curve with different strength fields.

Although simultaneous translation does not change local frames, individually translating vertices on a curve does induce orientation changes. To uniquely determine the quaternions in the latter case, we use the algorithm in [Singh and Fiume 1998] to obtain an intermediate deformed mesh which satisfies the individual vertex displacements only. The local frames from this intermediate mesh are compared to the original local frames to obtain the quaternions which are propagated before obtaining the final deformed mesh using (6).

Fig. 6 shows the deformation results on the BUNNY model by applying the above two types of simultaneous editing to a curve around the neck when the bottom part of the model is fixed. Even with large-scale deformations, the BUNNY's head and body exhibit nice elastic appearances while preserving small-scale features. Fig. 7 gives a comparison among three editing methods. The middle one is from naive Poisson editing which does not propagate local frame and scale changes. It has severe distortions because the Poisson equation by default can only enforce modified vertex positions in the boundary condition, but suppresses orientation and scale changes at the free vertices. As a result, such changes are confined to narrow regions surrounding the constrained vertices. The rightmost result in Fig. 7 is from WIRE [Singh and Fiume 1998]. We designed this experiment according to the author's suggestions [Singh 2004]. The original shape of the BUNNY's head is not well-preserved especially under rotations. This is because WIRE only considers changes in curve tangents which cannot uniquely determine 3D rotations alone. Ambiguities in curve rotations lead to discontinuous behaviors on the rest of the mesh. On the other hand, our algorithm can uniquely determine rotations using local frames with three axes, one of which is the surface normal on the curve.

Fig. 8 demonstrates detail editing by individually manipulating vertices on curves as well as local smoothing which will be discussed in Section 4.3.

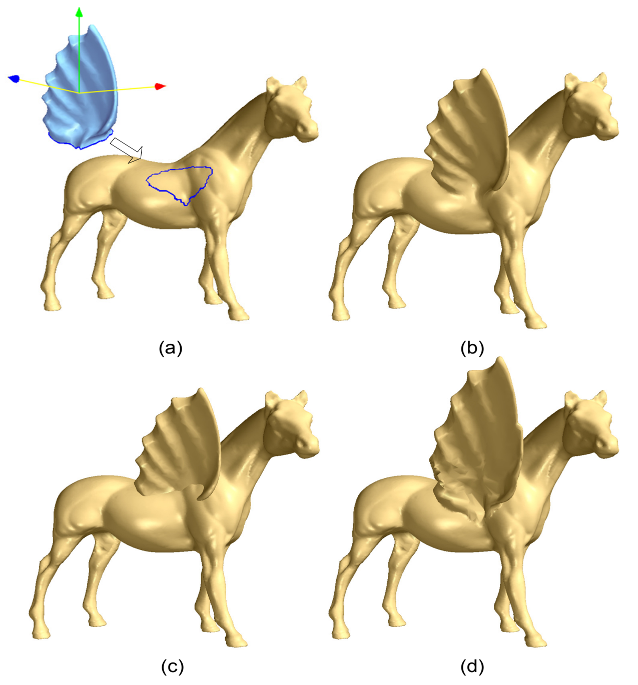

Figure 9: (a) the boundary on the WING (2000 faces) is projected along a user-defined direction to define the boundary on the HORSE model (100K faces), (b) the result from our projection scheme (running time = 400 ms), (c) Boolean operation, (d) WIRE.

4.1.1 Acceleration for Interactive Deformation

The Poisson equation is a sparse linear system that can be efficiently solved by the conjugate gradient method. However, according to the timings shown in Fig. 4 and 5, when the number of vertices in the mesh becomes large, it is impossible to achieve interactive rates by solving the equation at the original resolution. Since interactive performance is critical for user-guided deformation, we introduce a few acceleration schemes particularly for this task.

If we look at the linear system in (6), matrix $\mathbf{A}$ is only dependent on the parameterization mesh and the original target mesh before editing while $\mathbf{b}$ is also dependent on the current guidance vector field. Therefore, $\mathbf{A}$ is fixed as long as we do not switch the parameterization mesh while $\mathbf{b}$ changes constantly during interactive mesh editing. Thus, we can precompute $\mathbf{A}^{-1}$ using LU decomposition, and only dynamically execute the back substitution step to obtain $\mathbf{A}^{-1}\mathbf{b}$ at every frame. Our experiments indicate that this scheme alone can achieve a three to six fold speedup. Note that LU decomposition is less stable than conjugate gradient, and does not preserve the sparse structure of matrix $\mathbf{A}$. The latter implies that storing the result of LU decomposition requires more memory and reduces the largest mesh size a machine can handle. This acceleration scheme is only used during interactive sessions, and the user can request the system to produce a final version of the deformed mesh using conjugate gradient.

The second acceleration scheme exploits multiresolution meshes. We build a multiresolution mesh pyramid for large meshes using the algorithm presented in [Guskov et al. 1999] and only perform Poisson mesh editing at the coarsest resolution. At every frame, the pyramid is collapsed to add high frequency details back onto the modified coarsest level to obtain a modified high resolution mesh for display. The pyramid collapse operation can be performed very efficiently. Therefore, this scheme is much more efficient than directly solving the Poisson equation at the highest resolution. Because the level of precision tolerance increases with the scale of deformation, we use multiresolution acceleration for large-scale deformations like those shown in Fig. 6 where the finest BUNNY model for display has 70K faces and its corresponding coarsest model only has 2000 faces.

At the coarsest level of the BUNNY model, our LU-based acceleration took 29 milliseconds on an Intel Xeon 1.5GHz processor for each editing operation while the non-accelerated version took 105 milliseconds. With both acceleration schemes, our system only took around 100 milliseconds at the finest level where the non-accelerated version took multiple seconds. The size of the BUNNY model approaches the limit we can handle interactively (10fps) on the machine we use.

In our system, small-scale editing is directly performed on the finest level, but confined to a small surface region. With most of the mesh vertices fixed, editing in a small region can still be performed in real-time as well. The result shown in Fig. 8 was obtained in the finest level.

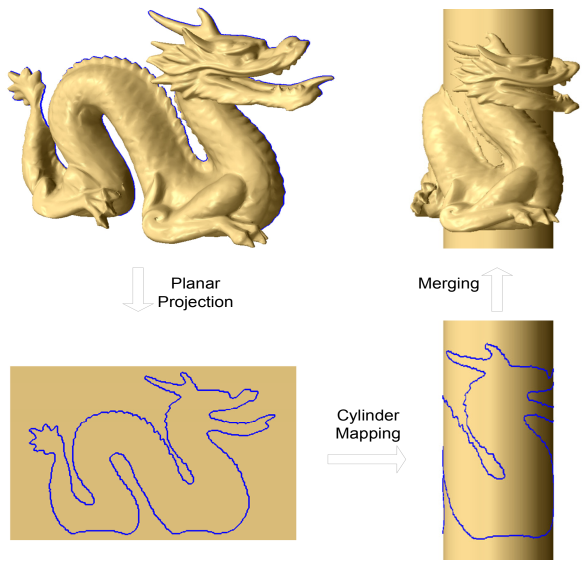

Figure 10: Object merging using mapping. Only the front facing half of the DRAGON model (18K faces) is mapped with its open boundary onto a plane which is then mapped onto the CYLINDER (60K faces). The running time is 5 seconds.

4.2 Mesh Merging and Assembly

Merging meshes to assemble a complete object is another important application of our framework. The partial meshes are merged at their open (mesh) boundaries which truly serve as Poisson boundary conditions this time! Merging two meshes involves the following steps: *i)* obtain a mesh boundary on each mesh and the vertex correspondence between them; *ii)* compute or custom design the local frames along the two boundaries; *iii)* obtain an intermediate boundary, including both vertex positions and local frames, by either interpolating the original two using the vertex correspondence or simply choosing one of the original two; *iv)* change the mesh connectivity along the boundaries of both meshes according to the intermediate boundary. *iv)* compare the local frames at the intermediate boundary with the local frames at the original two boundaries to obtain two sets of quaternions; *v)* propagate the two sets of quaternions towards the interior of both meshes, respectively; *vi)* set up the linear system in (6) for all the vertices from both meshes and solve it to obtain a merged mesh.

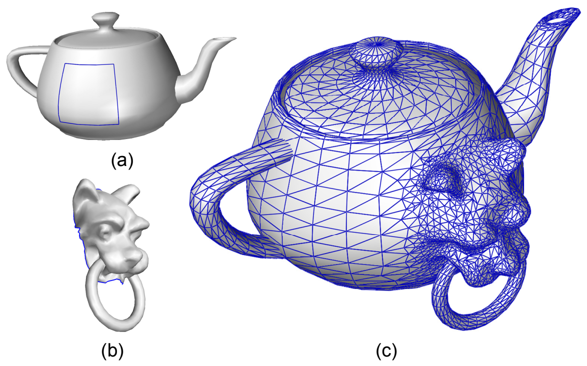

Figure 11: Object merging by interactively specifying sparse key vertex correspondences between two boundaries. GARGOYLE has 4000 faces, TEAPOT has 2000 faces, and the running time is 890 ms.

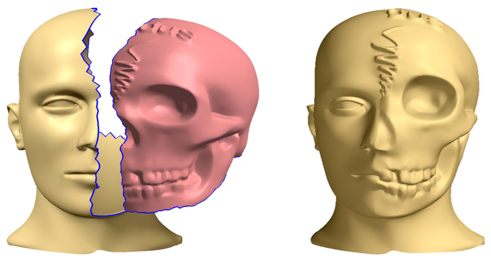

Figure 12: Two mesh components are merged at their jagged boundaries.

The vertex correspondence between boundaries is not automatically available in our method. We designed the following three interactive tools from which the user can choose. The first tool is quite restrictive but only needs little user interaction while the last one is most powerful but requires a larger amount of interaction.

i) The boundary on the first mesh is projected onto the second mesh to obtain the second boundary. The user only needs to interactively define a projection direction (Fig. 9(a)). Vertex correspondence is obtained by extending every vertex on the first boundary into a ray whose nearest vertex on the second mesh is then located. Fig. 9(b) shows a model merged using this tool.

ii) A planar parameterization of the boundary curve on the first mesh is first obtained. Then this 2D boundary is mapped onto the second mesh using a mapping scheme, such as the cylindrical mapping or more general parameterizations. A model merged using this tool is shown in Fig. 10.

iii) The user interactively defines sparse key vertex correspondences between the two boundaries and a dense correspondence is obtained through interpolation. Fig. 11 and 12 show two examples generated using this scheme.

Using our method to perform mesh merging has the following major advantages:

• It allows the two mesh boundaries to have very different shapes, sizes and roughness. For example, in Fig. 9(a), because the boundary of the WING is projected along an oblique direction onto the HORSE surface which has undulations, the two boundaries have different shapes and sizes. The two boundaries in Fig. 12 also have different shapes and are jagged.

• The propagation of local frame changes can globally adjust the two meshes so their shapes become more compatible with each other. This is demonstrated in Fig. 9(b), 11 and 12. Note that the example in Fig. 11 would be difficult for parameterization-based merging [Lévy 2003] since one of the components has genus greater than zero.

A comparison is given in Fig. 9 among three approaches. Fig. 9(b) shows our merging result. Fig. 9(c) is from Boolean intersection. To extract a closed intersection curve between the two partial meshes, we had to lower the WING. As a result, the undulations on the HORSE model hide a large portion of it. The result from WIRE [Singh and Fiume 1998] is shown in Fig. 9(d) where the WING exhibits the same type of distortions and discontinuities as in deformation.

An example with both deformation and merging is shown in Fig. 1. Multiple components are merged and the ARM is deformed before being merged.

Although Poisson mesh deformation and merging are powerful and flexible, they do not guarantee G^1 continuity between constrained and free vertices. Fortunately, continuity at these places can be significantly improved by Poisson normal smoothing, which will be introduced in the next section.

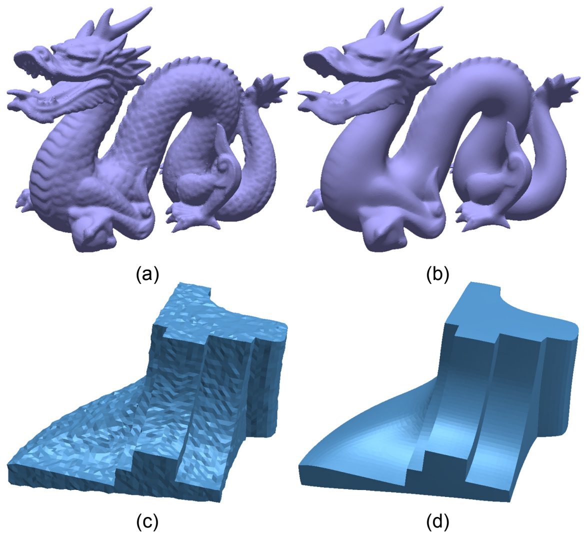

Figure 13: Denoising results. (a) Original model (150K vertices), (b) smoothed model of (a) after only one iteration with $\sigma_f = 4.0$ and $\sigma_g = 0.2\pi$. (c) Noisy model (Gaussian noise), (d) smoothed model of (c) after three iterations with $\sigma_f = 3.0$ and $\sigma_g = 0.2\pi$.

4.3 Mesh Smoothing and Denoising

The Poisson equation can be applied to perform mesh filtering operations in addition to interactive editing. As an example, we demonstrate a mesh smoothing and denoising algorithm in this section. Our algorithm does feature-preserving mesh smoothing via normals, which have previously been investigated [Taubin 2001; Tasdizen et al. 2002; Yagou et al. 2003]. In practice, we conduct bilateral filtering on the normals instead of the vertex positions [Jones et al. 2003; Fleishman et al. 2003] to preserve sharp features. Using normals to preserve features on a mesh is more intuitive since normals typically change abruptly at edges and creases. Actually [Jones et al. 2003] does perform normal smoothing as a preprocessing step. The bilateral filters in our method also have two parameters σ_f and σ_g. σ_f controls the spatial weight which is also used in [Jones et al. 2003; Fleishman et al. 2003] while σ_g defines the amount of normal variation allowed. Once smoothed normals have been obtained, our algorithm shifts vertex positions using the Poisson equation to reflect the altered normals while [Jones et al. 2003] performs a revised bilateral filtering on the vertices. Since

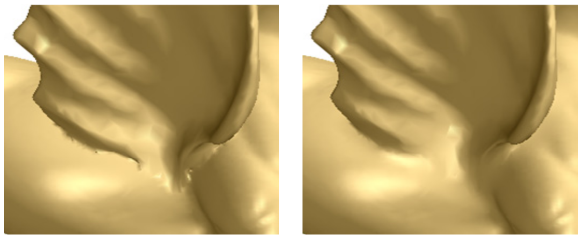

Figure 14: Smoothing merging boundary. Left: before smoothing, Right: after smoothing.

the normal of a triangle is a nonlinear function of its vertex positions, reconstructing a mesh from predefined normals is a classic nonlinear optimization problem [Yagou et al. 2003] which is both expensive and prone to local suboptimal solutions.

Our engine facilitates a *linear* method to obtain vertex positions from normals. Consider one triangle with its original normal $\mathbf{n_i}$. Suppose we have defined its new normal which is $\mathbf{n_i}'$. To incorporate this change, we define a local rotation matrix from the minimal rotation angle and its associated axis that can transform the original normal to the new one. This local rotation matrix serves as the local transform that should be applied to the original triangle to obtain a new triangle and its new gradient vectors. We perform this step over all triangles with altered normals to define new guidance fields as in Section 3.2. With these new guidance fields, the new vertex positions of the mesh can be obtained as in Section 3.1.

This smoothing algorithm can be applied to a mesh either once or with multiple iterations. The solution obtained can be either used directly or as the initialization for further nonlinear optimization. The use of this algorithm includes mesh denoising and mesh smoothing. In the latter case, we replace the bilateral filter with a regular Gaussian filter for normals because we do not wish to preserve small features and artifacts. Fig. 13 shows two examples of feature-preserving mesh denoising. Fig. 14 shows the effect of local smoothing at the merging boundary while Fig. 8 demonstrates smoothing in user-specified local regions.

5 Conclusions

In this paper, we have developed a basic framework along with interactive tools for mesh editing. The core of the technique is a Poisson mesh solver which has a solid theoretical foundation. The computations and implementations involved are very straightforward. The interactive tools are intuitive, and do not require special knowledge about the underlying theory. The product is a versatile mesh editing system that can be used for high-end applications which require superior results.

In future, we would like to overcome the limitations of the framework presented in this paper. The Poisson equation can only guarantee C^0 continuity between constrained and free vertices although the C^0 effect is not obvious due to local transform propagation. It is also possible to improve the performance of our user-guided deformation by exploiting multi-grid methods.

Acknowledgments: We wish to thank Karan Singh (U. Toronto) for discussion on WIRE, Lin Shi (UIUC) for discussion on the Poisson equation, Yiying Tong (USC) for discussion on vector field decomposition, and the anonymous reviewers for their valuable comments. Thanks to Ran Zhou, Bo Zhang and Steve Lin (MSRA) for their help in video production. Yizhou Yu was supported by NSF (CCR-0132970). Hujun Bao was supported by NSFC (No. 60021201 and 60033010) and 973 Program of China (No. 2002CB312104).

References

ABRAHAM, R., MARSDEN, J., AND RATIU, T. 1988. *Manifolds, Tensor Analysis, and Applications*, vol. 75. Springer. Applied Mathematical Sciences.

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

BAJAJ, C., AND XU, G. 2003. Anisotropic diffusion on surfaces and functions on surfaces. *ACM Trans. Graphics 22*, 1, 4–32.

BARR, A. 1984. Global and local deformations of solid primitives. *Computer Graphics(SIGGRAPH'84) 18*, 3, 21–30.

BENDELS, G., AND KLEIN, R. 2003. Mesh forging: Editing of 3d-meshes using implicitly defined occluders. In *Symposium on Geometry Processing*.

BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. 2001. Approximate boolean operations on free-form solids. In *Proceedings of SIGGRAPH*, 185–194.

CHANG, Y.-K., AND ROCKWOOD, A. 1994. A generalized de casteljau approach to 3d free-form deformation. In *Proc. SIGGRAPH'94*, 257–260.

COQUILLART, S. 1990. Extended free-form deformation: A sculpturing tool for 3d geometric modeling. *Computer Graphics(SIGGRAPH'90) 24*, 4, 187–196.

DESBRUN, M., MEYER, M., SCHRÖDER, P., AND BARR, A. 2000. Anisotropic feature-preserving denoising of height fields and bivariate data. In *Proc. Graphics Interface*, 145–152.

FLEISHMAN, S., DRORI, I., AND COHEN-OR, D. 2003. Bilateral mesh denoising. *ACM Trans. Graphics 22*, 3, 950–953.

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

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

JONES, T., DURAND, F., AND DESBRUN, M. 2003. Non-iterative, feature-preserving mesh smoothing. *ACM Trans. Graphics 22*, 3, 943–949.

KARNI, Z., AND GOTSMAN, C. 2000. Spectral compression of mesh geometry. In *Proc. SIGGRAPH'00*, 279–287.

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

KOBBELT, L., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. In *Proc. Eurographics'2000*.

LAZARUS, F., COQUILLART, S., AND JANCENE, P. 1994. Axial deformations: An intuitive deformation technique. *Computer Aided Design 26*, 8, 607–613.

LÉVY, B. 2003. Dual domain extrapolation. *ACM TOG 22*, 3, 364–369.

LLAMAS, I., KIM, B., GARGUS, J., ROSSIGNAC, J., AND SHAW, C. D. 2003. Twister: A space-warp operator for the two-handed editing of 3d shapes. *ACM Trans. Graphics 22*, 3, 663–668.

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

MUSETH, K., BREEN, D., WHITAKER, R., AND BARR, A. 2002. Level set surface editing operators. *ACM Transactions on Graphics 21*, 3, 330–338.

PAULY, M., KEISER, R., KOBBELT, L., AND GROSS, M. 2003. Shape modeling with point-sampled geometry. *ACM Trans. Graphics 22*, 3, 641–650.

PEREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Trans. on Graphics 22*, 313–318.

PERONA, P., AND MALIK, J. 1990. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. Patt. Anal. Mach. Intell. 12*, 7, 629–639.

POLTHIER, K., AND PREUSS, E. 2000. Variational approach to vector field decomposition. In *Proc. Eurographics Workshop on Scientific Visualization*.

SEDERBERG, T., AND PARRY, S. 1986. Free-form deformation of solid geometric models. *Computer Graphics(SIGGRAPH'86) 20*, 4, 151–160.

SETHIAN, J. 1999. *Level Set Methods and Fast Marching Methods*. Cambridge University Press.

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

SINGH, K., 2004. personal communication.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. Tech. rep., March 2004.

STAM, J. 1999. Stable fluids. In *SIGGRAPH 99 Conference Proceedings*, 121–128.

TASDIZEN, T., WHITAKER, R., BURCHARD, P., AND OSHER, S. 2002. Geometric surface smoothing via anisotropic diffusion of normals. In *Proceedings IEEE Visualization*, 125–132.

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

TAUBIN, G. 2001. Linear anisotropic mesh filtering. Tech. rep., IBM Research Report RC2213.

TOHLINE, J., 1999. Origin of the poisson equation. http://www.phys.lsu.edu/astro/H_Book.current/Context/PGE/poisson.origin.text.pdf.

TONG, Y., LOMBEYDA, S., HIRANI, A., AND DESBRUN, M. 2003. Discrete multi-scale vector field decomposition. *ACM Trans. Graphics 22*, 3, 445–452.

YAGOU, H., OHTAKE, Y., AND BELYAEV, A. 2003. Mesh denoising via iterative alpha-trimming and nonlinear diffusion of normals with automatic thresholding. In *Proc. Computer Graphics Intl.*