FiberMesh: Designing Freeform Surfaces with 3D Curves

Andrew Nealen
TU Berlin

Takeo Igarashi
The University of Tokyo / PRESTO JST

Olga Sorkine
TU Berlin

Marc Alexa
TU Berlin

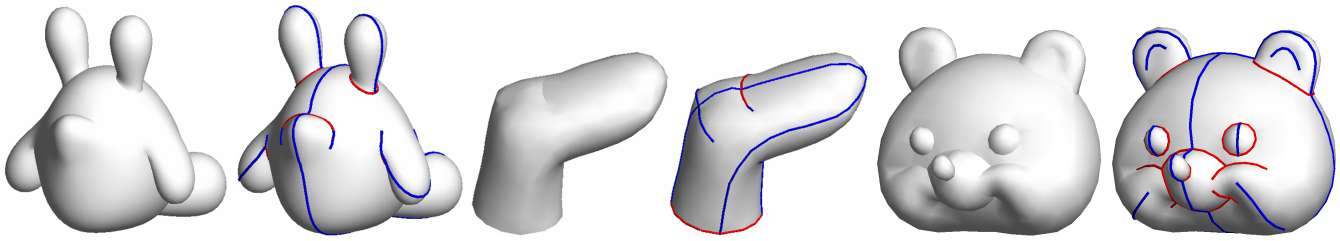


Figure 1: *Modeling results using* FIBERMESH. *The user interactively defines the control curves, combining sketching and direct manipulation, and the system continuously presents fair interpolative surfaces defined by these curves (blue = smooth curve, red = sharp curve).*

Abstract

This paper presents a system for designing freeform surfaces with a collection of 3D curves. The user first creates a rough 3D model by using a sketching interface. Unlike previous sketching systems, the user-drawn strokes stay on the model surface and serve as handles for controlling the geometry. The user can add, remove, and deform these control curves easily, as if working with a 2D line drawing. The curves can have arbitrary topology; they need not be connected to each other. For a given set of curves, the system automatically constructs a smooth surface embedding by applying functional optimization. Our system provides real-time algorithms for both control curve deformation and the subsequent surface optimization. We show that one can create sophisticated models using this system, which have not yet been seen in previous sketching or functional optimization systems.

Keywords: Sketch Based Interfaces and Modeling, Differential Representations, Sketching, Deformations, Fair Surface Design

1 Introduction

Current tools for free-form design, and the resulting design process, can be roughly categorized into two groups. The group of professional modeling packages makes use of parametric patches or subdivision surfaces [Maya 2007; 3ds Max 2007], where the user has to lay out the coarsest level patches in an initial modeling stage, and then modify control points to generate details. Because it is difficult for inexperienced users to generate the control structure for an intended shape from scratch, a group of research tools [Igarashi et al. 1999; Igarashi and Hughes 2003; Schmidt et al. 2005; Karpenko and Hughes 2006; Kara and Shimada 2007] as well as in-game character editors [Maxis 2007; Gingold 2007] are built around intuitive modeling metaphors such as sketching, trying to hide the mathematical subtleties of surface description from the user. However, some of these tools lack a high-level control structure, making it difficult to iteratively refine the design, or re-use existing designs.

We try to bridge the gap by using curves, a universally accepted modeling metaphor, as an interface for designing a surface. Notice that curves appear in both tools mentioned above: they appear as parameter lines, or seams where locally parameterized patches meet; they are sketched to generate or modify shape, or they are extracted from the current shape and used as handles. Also note that traditional design is mostly based on drawing characteristic curves.

Yet, design is a process. We cannot expect a user to draw the control (or characteristic) curves of a shape into free space. Our first fundamental idea is to let the user **define control curves by drawing them onto the shape** in its current design stage. These curves can be used as handles for deformation right after their definition, as in other tools, or at any other time in the design process. Of course, the effect of control curves can be modified (i.e. smooth vs. sharp edge), they can be removed from the current design, and there are no restrictions on their placement and topological structure. Specifically, they may be connected to or intersect other curves, or not; this is more general than recent developments for parameterized surfaces [Sederberg et al. 2003; Schaefer et al. 2005]).

The second fundamental principle is that **the shape is defined by the control curves** at any stage of the design process. While we found it important to serve the process of construction, and this is also what defines the topology of the surface, the result should be independent of when a control curve was modified. We achieve this by defining the surface to minimize certain functions of its differentials [Moreton and Séquin 1992; Welch and Witkin 1994], while constraining it by the control curves.

It is crucial that both the modification of curves as well as the computation of surface geometry allow for an interactive and smoothly responding system. For this we build on the recent advances in discrete Laplacian [Sorkine et al. 2004; Yu et al. 2004; Botsch and Kobbelt 2004] and other higher order or non-linear functionals for surface processing [Huang et al. 2006; Botsch et al. 2006; Wardetzky et al. 2007].

Uniquely combining interface metaphors (Section 2) with geometry processing techniques (Section 3), our contributions are

- A fair surface definition based on curve constraints, and an accompanying functional optimization algorithm, which runs at interactive rates.
- A detail preserving, real-time 3D curve editing and peeling interface, and a curve deformation algorithm based on discrete co-rotational methods.
- The generation and smooth embedding of initial surface components by sketching a planar control curve on a canvas.
- An interface that enables the design of 3D models with 3D control curves. The user's 2D sketching operations turn into 3D curves, and they serve as handles for subsequent editing.

Welch and Witkin [1994] propose an interactive modeling system where the user can cut up surfaces and paste them together, while the system continuously generates a fair interpolative surface. They demonstrate the capability of the approach by showing topologically non-trivial shapes such as branching surfaces and a Klein bottle. Our work extends this approach, allowing the user to design more practical models such as 3D characters (Fig. 1), by introducing a high-level user interface for curve control.

Discrete differential surface representations are an active research area [Sorkine 2006]. One of the key driving forces is the use of highly efficient sparse linear solvers [Toledo 2003; Davis 2004]. They can efficiently solve matrix systems of tens of thousands of entries, which makes it possible to process interesting 3D meshes in real-time. Most efforts have been dedicated to the editing of *existing* 3D models. Our work is an attempt to apply these techniques and tools to surface *creation* from scratch.

2 User Interface

From the user's point of view, our system can be seen as an extension to a freeform modeling system based on silhouette sketching, such as Teddy [Igarashi et al. 1999]. The user interactively draws the silhouette of the desired geometry and the system automatically constructs a (rotund) surface via functional optimization, such that its silhouette matches the user's sketch. However, unlike previous systems, the user's original stroke *stays* on the model surface and serves as a handle for further geometry control. The user can push and pull these curves interactively and the surface geometry changes accordingly. In addition, the user can freely add and remove control curves on the surface. These extensions enable the design of far more elaborate shapes than those possible with sketching alone. Fig. 2 shows an overview of the process.

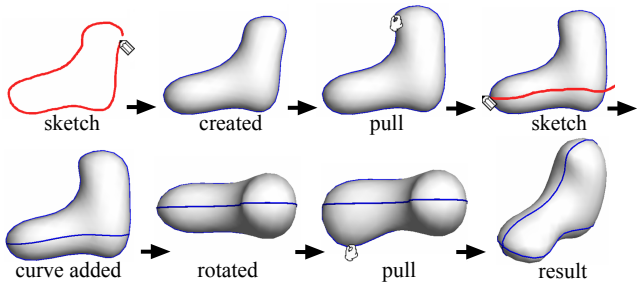


Figure 2: *An example modeling sequence.*

In a sense, our modeling process is similar to traditional modeling methods, such as parametric patches and subdivision surfaces: the user also defines nets of curves and the system automatically generates a smooth surface based on these. An advantage of our interface is that the user does not need to worry about the topology of the curves. Traditional methods require the user to cover the entire surface with triangle or quad regions. Our method is much more flexible: curves need not be connected to other curves and much fewer curves can represent simple geometry. It is also important that, instead of providing individual points as an interface, our interface treats curves as continuous entities. We believe this can help smooth the "skill transfer" from 2D drawing to 3D modeling.

Various interactive modeling methods have been proposed in research contexts, including direct 3D editing [Perry and Frisken 2001], spatial deformation operations [Singh and Fiume 1998; Angelidis et al. 2006; von Funck et al. 2006] and surface based deformation tools [Zorin et al. 1997; Kobbelt et al. 1998; Nealen et al. 2005]. These modeling methods provide reduced degrees-of-freedom handles (curves and control meshes) for surface control. The user can focus on the high-level control and the system automatically maintains aesthetic consistency. Our approach is unique in that we use curves also as the *definition* of the surface, not only as temporal handles for deformation.

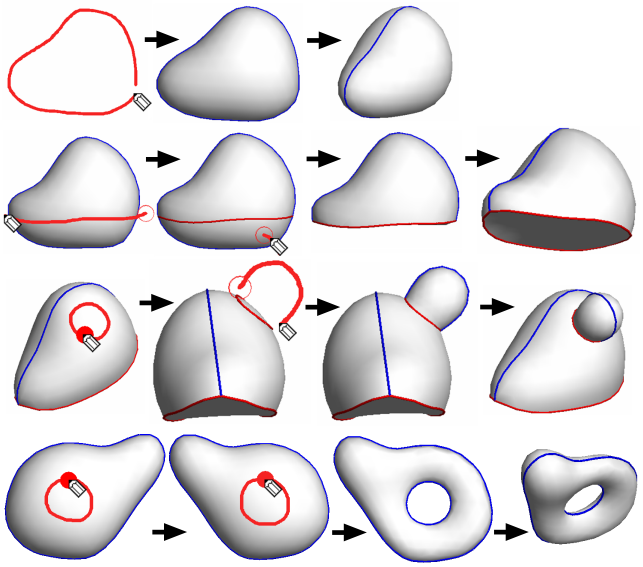


Figure 3: *Sketching operations (from top to bottom): creation, cut, extrusion and tunnel.*

Our current modeling interface consists of five tools (modes): sketching tool, deformation tool, rubbing tool, erasing tool, and type change tool. The user switches between these tools via menu selection or a keyboard shortcut.

2.1 Sketching Tool

Our system provides five kinds of sketching operations: creation, cut, extrusion, tunnel (Fig. 3), and add-control-curve (Fig. 4). When the user draws a closed stroke on a blank canvas, the system automatically inflates the closed area and presents an initial 3D model. The user draws a stroke crossing the model to cut it. Drawing a closed stroke on the object surface followed by a silhouette stroke creates an extrusion. If the user draws another closed loop on the opposite side of the surface, the system generates a tunnel. These operations are borrowed from the original Teddy system, but the difference is that the user's original strokes stay on the model surface as control curves. These control curves literally define the surface shape (as positional constraints in the surface optimization), and the user can modify the shape by deforming these control curves. New control curves can be added by drawing an open stroke on the object surface, drawing a closed stroke followed by clicking, and by drawing a cutting stroke followed by clicking (Fig. 4). The last method is very useful during the early stages of model creation, since it allows the user to quickly generate a convenient handle to adjust the amount of inflation (or *fatness*).

The control curves are divided into two types: smooth curves (blue) and sharp curves (red). A smooth curve constrains the surface to be smooth across it, while a sharp curve only places positional constraints with C^0 continuity. These types are automatically assigned to the newly added curves according to the sketching operation the user applied. The creation operation generates a smooth curve that

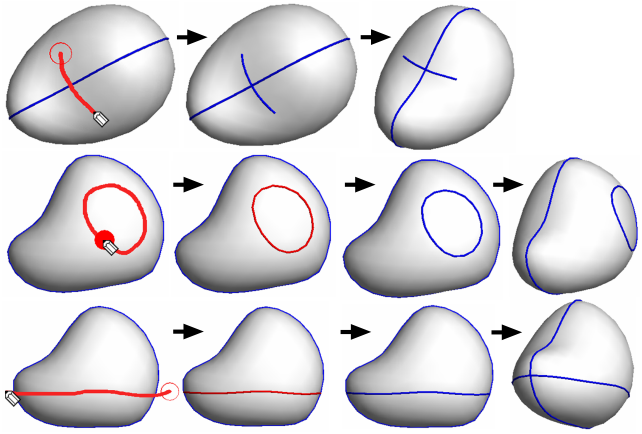


Figure 4: *Adding control curves: open stroke (top), closed stroke (middle) and cutting stroke (bottom). The user needs to click after drawing a stroke to make it a control curve in case of closed stroke and cutting stroke.*

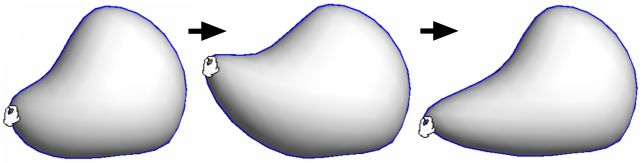


Figure 5: *Pulling a curve. The deformed curve segment is determined by how much of it the user peels off.*

corresponds to the silhouette. A cutting operation generates a sharp curve. An extrusion generates one sharp curve along the base, and one smooth curve along the silhouette. When the user paints a new curve on the surface, it is initially defined as a smooth curve. The type of these curves can be freely changed afterwards using the type change tool.

2.2 Deformation Tool

The deformation tool lets the user grab a curve at any point and pull it to the desired location. The curve deforms accordingly, preserving local details as much as possible (see Fig. 5 and Section 3.1). Editing operations are always applied to the control curves, not directly to the surface. If the user wants more control, new control curves must be added on the surface. Explicit addition of control curves exposes the surface structure in a clear way, and the curves serve as a convenient handle for further editing.

We use a peeling interface for the determination of the deformed curve segment (region of interest, ROI) [Igarashi et al. 2005]. The size of the curve segment to be deformed is proportional to the amount of pulling. The more the user pulls, the larger the deformed curve segment becomes. This frees the user from manually specifying the ROI before starting deformation and enables dynamical adjustment of the ROI during deformation. This peeling effect propagates to the other curves connected to the deformed curve, which allows the user to deform a larger area of the surface.

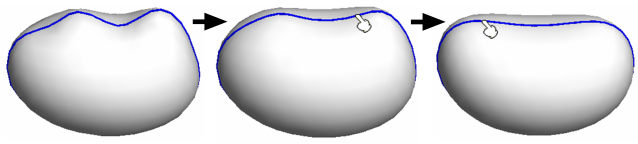


Figure 6: *Rubbing (smoothing) a curve.*

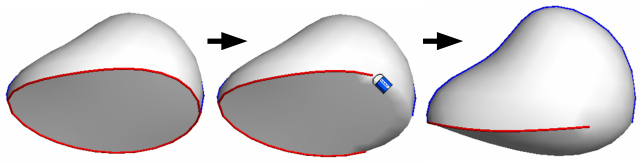


Figure 7: *Erasing a control curve (left: before erasing, middle: immediately after erasing, right: after surface optimization).*

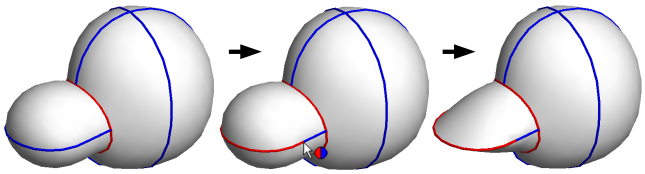


Figure 8: *Changing the curve type (left: before the change, middle: immediately after the change, right: after surface optimization).*

2.3 Rubbing Tool

The rubbing tool is used for smoothing a curve. As the user drags the mouse back and forth (rubs) near the target curve, the curve gradually becomes smooth. The more the user rubs, the smoother the curve becomes (Fig. 6). This tool is very important because the curves resulting from sketching can contain noise, and localized deformation can also introduce jaggy parts. It might be possible to automatically apply denoising after each user interaction, but it is not clear to which extent smoothing should be applied. Our rubbing tool provides an intuitive and convenient interface for specifying the target area to apply smoothing to, as well as the amount of smoothing. Our current implementation moves each vertex being rubbed one by one so that it locally improves inner and outer fairness.

2.4 Erasing Tool and Type Change Tool

The erasing tool works as a standard curve segment eraser: the user drags the cursor along a control curve to erase it. This is equivalent to removing constraints that define the surface. The system optimizes the surface when the user finishes an erasing operation (releases the mouse button, Fig. 7). The type change tool is for changing the type of a control curve. Like the erasing tool, the user drags the cursor along a curve to change the property. If the curve is a sharp curve, it converts it to a smooth curve (or curve segment), and vice versa. As with the erasing tool, the system updates the surface geometry according to the property change and presents the result after the user finishes the operation (releases the mouse button, Fig. 8).

3 Algorithm

To implement the described interface we propose an algorithm which consists of two main steps: curve deformation and surface

optimization. The additional steps, mesh construction and remeshing (Section 3.3), only occur at the end of the modeling operations creation, extrusion, cut, and deformation.

Instead of solving for both curve positions and fair surface simultaneously, we have found that decoupling the curve deformation from the surface optimization step is fast, intuitive, produces aesthetically pleasing results, and supports our fundamental principle of defining shape by control curves. The user first deforms (pulls) the curve(s) using the deformation tool (Section 3.1), after which the new curve positions are fed to the surface optimization step as positional constraints (Section 3.2). During curve pulling, these two operations are performed sequentially to achieve interactive updates of both the curves and the surface they define.

3.1 Curve Deformation

The user interface for curve deformation is a usual direct manipulation method: the user grabs and drags a point on a curve, and the curve deforms smoothly within the peeled ROI. The current implementation always moves the grabbed point parallel to the screen.

The algorithm we use is a variant of detail-preserving deformation methods using differential coordinates [Sorkine 2006], combined with co-rotational methods [Felippa 2007]. Geometry is represented using differential coordinates, and the final result is obtained by solving a sequence of linear least-squares problems, which satisfy the positional constraints given by the user. The main challenge in this framework is the computation of appropriate rotations for the differential coordinates. One approach is to explicitly compute rotations beforehand, typically by smoothly interpolating the prescribed orientation constraints defined by the user [Yu et al. 2004; Lipman et al. 2005; Zhou et al. 2005; Zayer et al. 2005]. These methods are not applicable in our setting because the user should only need to drag a vertex without specifying rotations. Another approach is to implicitly compute rotations as a linear combination of target vertex positions [Sorkine et al. 2004; Fu et al. 2007]. Our technique is similar to these methods, but we explicitly represent rotation matrices as separate free variables. This is due to the fact that neighboring vertices along a curve are nearly collinear and inappropriate for deriving rotations from them.

Conceptually, what we want to solve is the following error minimization problem

$$\arg\min_{\mathbf{v},\mathbf{R}} \Big\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \mathbf{R}_i\delta_i\|^2 + \sum_{i\in C_1} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{i,j\in E} \|\mathbf{R}_i - \mathbf{R}_j\|_F^2 + \sum_{i\in C_2} \|\mathbf{R}_i - \mathbf{R}_i'\|_F^2 \Big\}, \tag{1}$$

where $\mathbf{L}(\cdot)$ is the differential operator, $\mathbf{v}_i$ represents the vertex coordinates, $\mathbf{R}_i$ represents rotations associated with these vertices in the deformed curve, $\|\cdot\|_F$ is the Frobenius norm, E is the set of curve edges, C_1 and C_2 are the sets of constrained vertices, and primed values are given constraints. The first term minimizes the difference between the resulting differential coordinates and the rotated original differential coordinates δ_i. The second term represents positional constraints (we use three constrains: two at the boundary of the ROI and one at the handle). The third term ensures that the rotations are smoothly varying along the curve [Allen et al. 2003; Sumner and Popović 2004; Fu et al. 2007], and the last term represents rotational constraints (we use two constraints at the boundary of the ROI). These four terms also need to be appropriately weighted to obtain visually pleasing results. We have omitted these weights in the above equation for simplicity.

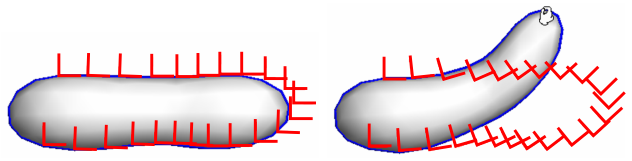


Figure 9: *Rotated local coordinate frames (red) after curve deformation by pulling a single vertex.*

A problem with this approach is that $\mathbf{R}$ is not linear. Unconstrained transformation includes shearing, stretching, and scaling, which is undesirable for our application. Similar to [Sorkine et al. 2004], we therefore use a linearized rotation matrix to represent small rotations. In order to accommodate large rotations, we iteratively compute the gross rotation by concatenating small delta rotations obtained by solving a linear system at each step.

In summary, what we solve in each step is the following minimization problem

$$\arg\min_{\mathbf{v},\mathbf{r}} \Big\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \mathbf{r}_i\mathbf{R}_i\delta_i\|^2 + \sum_{i\in C_1} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{i,j\in E} \|\mathbf{r}_i\mathbf{R}_i - \mathbf{r}_j\mathbf{R}_j\|_F^2 + \sum_{i\in C_2} \|\mathbf{r}_i\mathbf{R}_i - \mathbf{R}_i'\|_F^2 \Big\}, \tag{2}$$

where $\mathbf{R}_i$ is the gross rotation obtained from the previous iteration step and fixed in each minimization step. $\mathbf{r}_i$ is a linearized incremental rotation represented as a skew symmetric matrix with three unknowns

$$\mathbf{r}_i = \begin{bmatrix} 1 & -r_{iz} & r_{iy} \\ r_{iz} & 1 & -r_{ix} \\ -r_{iy} & r_{ix} & 1 \end{bmatrix}.$$

As a whole, this minimization problem amounts to the solution of a sparse linear system and it returns optimal vertex positions and delta rotations $\mathbf{r}_i$. We update target gross rotations as $\mathbf{R}_i \leftarrow \mathbf{r}_i\mathbf{R}_i$, and also orthonormalize them using polar decomposition [Fu et al. 2007]. Figure 9 shows the resulting gross rotations obtained using this algorithm.

One remaining issue is the choice of differential coordinates L. We have tested two options: first order differentials (L_0) and second order differentials (L_1)

$$L_0 = \mathbf{v}_i - \mathbf{v}_{i-1}, \qquad L_1 = \mathbf{v}_i - \frac{1}{|N_i|}\sum_{j\in N_i} \mathbf{v}_j.$$

L_1 seems to be the popular choice for surface deformation. However in our case, we found that L_1 is not appropriate for the estimation of rotations because it almost always degenerates (i.e. is close to zero) in a smooth curve. On the other hand, L_0 always has certain length in an appropriately sampled curve and serves as a reliable guide for estimating rotations. One problem with L_0 based geometry computation is that it causes C^1 discontinuities on the boundaries of the ROI. Therefore, we first use L_0 for the iterative process of rotation estimation, and then switch to L_1 for computing the final vertex positions using the estimated rotations. This combination is a bit complicated, but gives the best results in our experiments.

Physically inspired methods, such as PriMo [Botsch et al. 2006], have become very popular in the context of surface modeling. For comparison and experimentation purposes we have implemented a

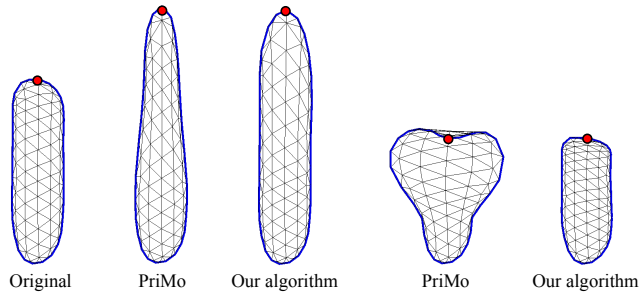


Figure 10: *Comparison of our curve deformation to a physically based approach inspired by PriMo.*

variant of PriMo tailored to 3D curves, i.e. each curve segment defines a (3D) prism. While PriMo is an excellent choice for the simulation of physically plausible deformation, we found it to be unsuitable for our curve editing tool. When the curve is compressed it shows undesirable buckling, while when stretched it loses local detail. Both phenomena result from length preservation, inherent to all physically inspired curve deformation algorithms. In contrast, our uniform discretization of the Laplacian (L_1) tolerates some scaling (see Fig 10).

3.2 Surface Optimization

It is important to provide real-time visual feedback to the user during control curve deformation. This constraint necessitates the use of a fast surface optimization algorithm. An intuitive choice appears to be discrete surfaces defined as the solution of sparse linear systems [Sorkine 2006; Botsch and Sorkine 2007]. If we kept the system matrix constant during interaction, updating the positions would only require back-substitution, which is very fast. Unfortunately, in our setting we have encountered a shortcoming inherent to these algorithms, which is due to the absence of normal constraints along the curves. Specifically, if the positional constraints lie in a subspace, the solution will also be constrained to lie in this subspace. In our tool, the initially sketched curve is planar, so the resulting mesh geometry is also planar, see Fig. 11. We prove this property for the constructions of [Botsch and Kobbelt 2004] and [Sorkine and Cohen-Or 2004] in [Nealen and Sorkine 2007]. Even in the presence of non-planar positional constraints, surfaces from [Sorkine and Cohen-Or 2004] or the linearized thin plate surfaces of [Botsch and Kobbelt 2004] seem to concentrate curvature near the curves, see Fig. 12 (left column).

The problem could be solved by asking the user to specify normal constraints for the curves. In most mesh editing tools, normal constraints are implemented by fixing n-rings of adjacent vertices. We have not considered this as an option, since it is our strict design goal to keep the interface simple. Instead, we have chosen to implement a solution, which generates a fair surface that interpolates the control curves by means of nonlinear functional optimization. There are a variety of possible objective functions to choose from. Welch and Witkin [1994] minimize the integral of squared principle curvatures

$$E_p = \int_S (\kappa_1^2 + \kappa_2^2)\, dA, \tag{3}$$

also known as thin-plate energy, while Bobenko and Schröder [2005] minimize the closely related Willmore energy

$$E_w = \int_S (\kappa_1 - \kappa_2)^2\, dA, \tag{4}$$

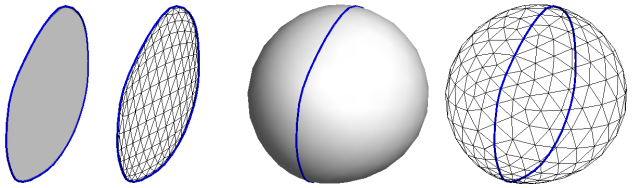


Figure 11: *The results of least-squares meshes (left) and our nonlinear solution (right) for a planar curve.*

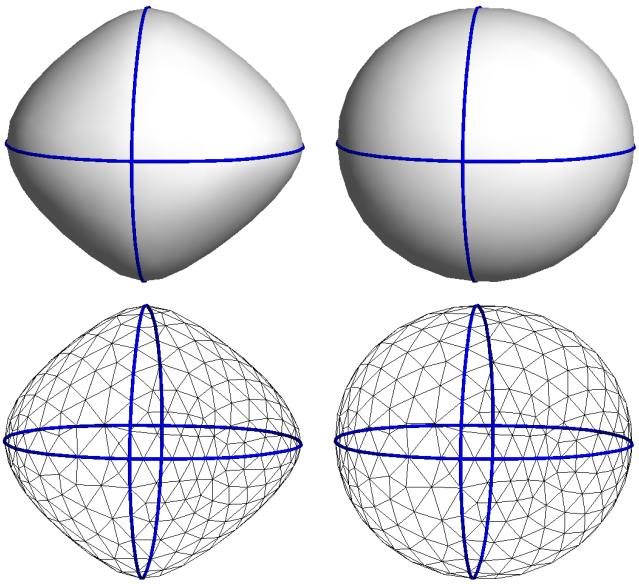


Figure 12: *Least squares mesh (= linearized thin plate surface $\Delta^2\mathbf{x} = 0$, left) and the results of our nonlinear solution (right).*

implemented as a flow. They use this flow for smoothing and hole filling. Moreton and Séquin [1992] minimize variation of curvature

$$E_c = \int_S \left(\frac{d\kappa_n}{d\hat{e}_1}\right)^2 + \left(\frac{d\kappa_n}{d\hat{e}_2}\right)^2 dA, \tag{5}$$

which is the integral of (squared) partial derivatives of normal curvature κ_n w.r.t. the directions $\hat{e}_1$, $\hat{e}_2$ of principal curvatures.

Each objective function has its own strengths and weaknesses, which also heavily depend on how it is implemented. Based on these previous results and our own experiences, we chose to compute a surface, which results from a sequence of optimization problems. This is inspired by a surface construction method presented by Schneider and Kobbelt [2001]. The PDE governing fairness in their work is defined as $\Delta_B H = 0$, where Δ_B is the discrete Laplace-Beltrami operator, and $H = (\kappa_1 + \kappa_2)/2$ is the mean curvature. Their basic idea is to factorize this fourth order problem into two second order problems and solve them sequentially. First they compute target mean curvatures (scalars) that smoothly interpolate the curvatures specified at the boundary, and then move the vertices, one vertex at a time, to satisfy the target curvatures.

However, the second stage of their technique is not fast enough to provide interactive updates of the geometry when the user pulls the curve. In addition, we are lacking curvature information at the boundaries. Our idea for a faster computation, is to cast both second order problems as sparse linear systems that use a constant system

matrix. This allows factoring the matrices once and then performing only back-substitution during the iterations.

In particular, in the first second-order system we replace the geometry dependent Laplace-Beltrami operator by the uniformly discretized Laplace operator and solve the following least-squares minimization problem

$$\arg\min_{c} \left\{ \sum_i \|\mathbf{L}(c_i)\|^2 + \sum_i \|c_i - c_i'\|^2 \right\}, \qquad (6)$$

where $\mathbf{L}(\cdot)$ denotes the discrete graph Laplacian, to obtain a set of smoothly varying Laplacian magnitudes (LMs) $\{c_i\}$, which approximate scalar mean curvature values. The first term requires that the neighboring LMs vary smoothly and the second term requires the LMs at all vertices to be near the current LM c_i'. In the first iteration we set target LMs only for the constrained curves using the scalar mean curvatures along these curves. Unlike [Schneider and Kobbelt 2001], where the curvature is fixed at the boundary, these initial target LMs are likely to change in subsequent iterations.

To obtain a geometry that satisfies these target LMs we use the uniformly discretized Laplacian as an estimator of the *integrated* mean curvature normal [Wardetzky et al. 2007]. The integrated target Laplacian $\delta_i = A_i \cdot c_i \cdot \mathbf{n}_i$ per vertex is given as the product of an area estimate A_i for vertex i, the target LM c_i and an estimate of the normal $\mathbf{n}_i$ from the current face normals. Then new positions could then be computed by solving the following global least-squares system

$$\arg\min_{\mathbf{v}} \left\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \delta_i\|^2 + \sum_{i \in C} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 \right\}, \qquad (7)$$

where the first term requires that the vertex Laplacians are close to the integrated target Laplacians, and the second term places positional constraints on all vertices in the control curve set C.

However, our assumption that the uniformly discretized Laplacian is a reasonable estimate for the integrated mean curvature normal does not hold when the edges around a vertex are not of equal length. Rather than using a geometry dependent discretization, which would require recomputation of the system matrix in each iteration, we try to achieve equal edge lengths by prescribing target edge vectors. For this, we first compute desired scalar edge lengths, similar to the computation of desired target LMs, by solving

$$\arg\min_{e} \left\{ \sum_i \|\mathbf{L}(e_i)\|^2 + \sum_i \|e_i - e_i'\|^2 \right\}, \qquad (8)$$

for a smooth set $\{e_i\}$ of target average edge lengths, from the current set of the average lengths e_i' of edges incident on vertex i. Again, we start the iterations by using only the edge lengths along the given boundary curve. Note that the matrix for this linear system is identical to the system for computing target LMs, so that we can re-use the factored matrix.

From these target average edge lengths, we derive target edge vectors for a subset B of the edges in the mesh

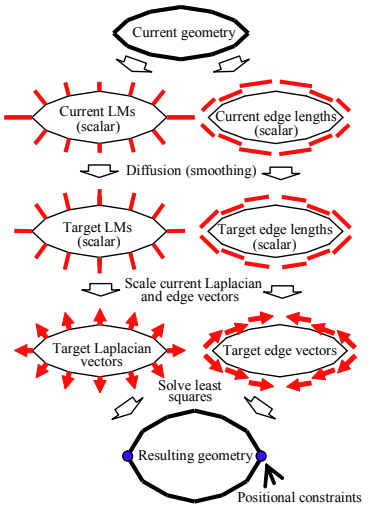
$$\eta_{ij} = (e_i + e_j)/2 \cdot (\mathbf{v}_i - \mathbf{v}_j)/\|\mathbf{v}_i - \mathbf{v}_j\|. \qquad (9)$$

Using this set of target edge vectors, we modify the linear system in Eqn. 7 to derive the updated vertex positions as follows:

$$\arg\min_{\mathbf{v}} \Bigg\{ \sum_i \|\mathbf{L}(\mathbf{v}_i) - \delta_i\|^2 + \sum_{i \in C} \|\mathbf{v}_i - \mathbf{v}_i'\|^2 + \sum_{(i,j) \in B} \|\mathbf{v}_i - \mathbf{v}_j - \eta_{ij}\|^2 \Bigg\}. \qquad (10)$$

We have found that it is sufficient to only constrain edges incident to the constrained curves, because setting the uniformly discretized Laplacian equal to vectors in normal direction automatically improves inner fairness at all free vertices [Nealen et al. 2006].

The two-step process, consisting of solving for target LMs and edge lengths and then updating the positions, is repeated until convergence. In practice, we observed that the computation converges rather quickly, in approximately 5 to 10 iterations. The system needs to repeatedly solve a few sparse linear systems, but the expensive matrix factorizations are required only once at the beginning (because left-hand side matrices remain unchanged during iteration). The system only needs to run back-substitutions during the iterations, which is very fast. See the figure on the right for an overview of one iteration.



While the algorithm described above is stable and robust, it is not entirely independent of tessellation, since we use the uniformly weighted graph Laplacian as an approximation of the integrated mean curvature normal to avoid matrix factorization in every iteration. To overcome this, we could compute a minimal energy surface as Schneider and Kobbelt [2001] propose. As an experiment with a non-linear solution that is independent of surface tessellation, we have implemented the inexact Newton method for Willmore flow described in [Wardetzky et al. 2007]. We have found that our algorithm tends to generate very similar results if the discretization is near-regular and that, as expected, there are situations where unequal edge lengths along the fixed boundaries would benefit from the discretization-independent solution. However, not only are these techniques significantly slower to an extent that makes them unsuitable for most interactive editing situations, we have also encountered that the solution can become unstable when using insufficient boundary constraints, i.e. curves without normals (this is expected and mentioned in [Wardetzky et al. 2007]).

3.3 Meshing and re-meshing implementations

The system generates a new mesh after the creation, cut, and extrusion operations. In the case of cut, the system flattens the intersection (it is always developable) and generates a 2D mesh inside of it. In the cases of creation and extrusion, the system generates a 2D mesh on the image plane within the region surrounded by the input stroke. The system first resamples the input stroke and then smoothes it by moving each vertex to the mid point of adjacent vertices. It is possible to skip this process, but the resulting mesh is nicer for our purpose because it ends up generating more triangles in high curvature areas. The resampled stroke is intersected with a regular triangular grid mesh, and each point of the resampled stroke is connected to the nearest grid vertex. Both front and back sides are created from the same 2D mesh and stitched together at the common boundary (Fig. 13). Note that this merely defines the mesh connectivity, not the actual geometry, which is computed subsequently as described in the previous section.

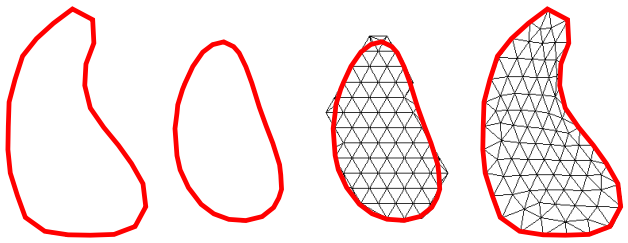


Figure 13: *Initial mesh generation. The sketch curve (left) is resampled, smoothed (2nd from left) and intersected with a regular triangular grid (3rd from left), resulting in the mesh topology used for surface optimization (right).*

As the geometry is changed by the user, it may become necessary to remesh the surface. One possible approach is to interleave mesh topology updates and vertex position updates as in [Welch and Witkin 1994]. Unfortunately, our surface optimization algorithm heavily relies on pre-factorization of the Laplacian matrix, which is defined by mesh topology. If the mesh topology changes during optimization, the factorization must be computed again, which is a significant overhead. Therefore, we apply remeshing only when a large change occurs, and use a constant topology mesh during (continuous) deformation and surface optimization to provide real-time feedback. Specifically, we apply remeshing after each sketch-based modeling operation, and when the user releases the mouse button after a deformation (pulling) operation. We use a modified version of explicit remeshing [Surazhsky and Gotsman 2003] in our system.

4 Results

Fig. 14 shows shapes which are difficult to model with implicit representations [Turk and O'Brien 2002; Karpenko et al. 2002; Schmidt et al. 2005]. CSG operations allow the user to represent *closed* sharp curves along a boundary [Schmidt et al. 2005], but it is problematic to specify an *open* sharp curve starting in the middle of a smooth surface. It is also difficult to represent *point sharp* (e.g., the tip of a cone) using the standard implicit representation. Both can be modeled with our system (Fig. 14). Markosian et al. [1999] show that it is possible to include some creases on the surface by tracing the implicit surface with an explicit mesh, but the basic geometry is defined by a user-defined polygonal skeleton, not by surface curves.

Figs. 15 and 16 show some more complex results obtained with our modeling tool. While the models shown in Fig. 1 each took a trained user approximately 5-10 minutes to create, those depicted in Figs. 15 and 16 took between 10 minutes (arm) and 1 hour (torso). See the accompanying video for more details on the construction process.

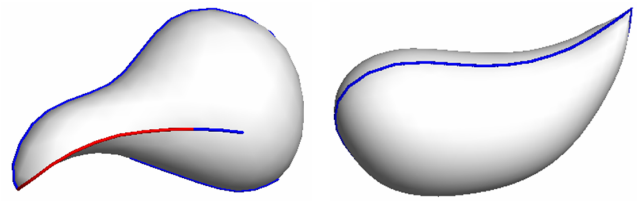


Figure 14: *Open sharp curve (left) and point-sharp curve (right).*

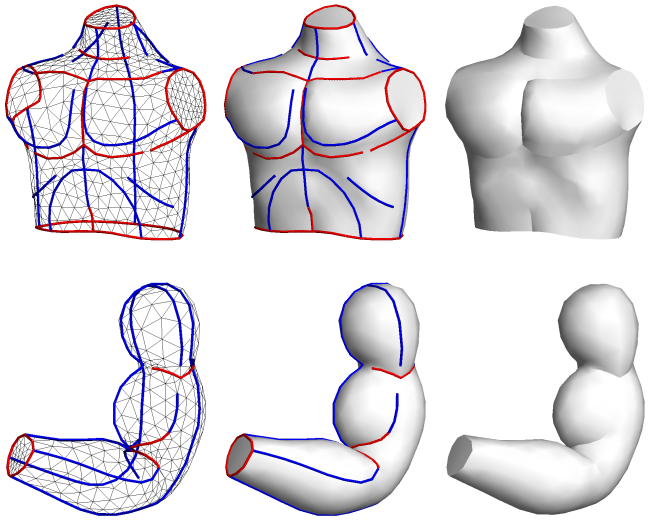


Figure 15: *Some results obtained using* FIBERMESH.

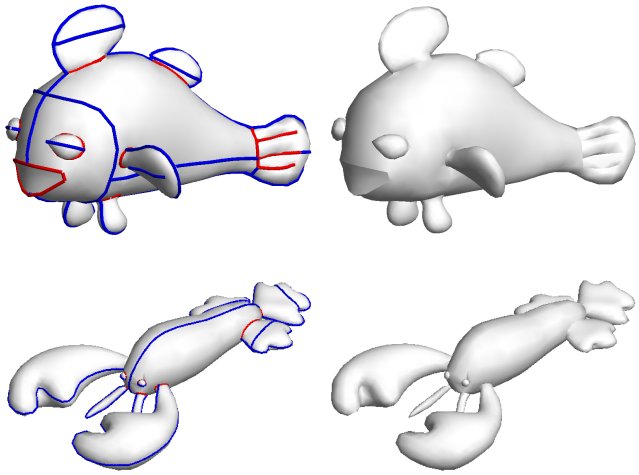


Figure 16: *Some* fishy *results obtained with the* FIBERMESH *tool.*

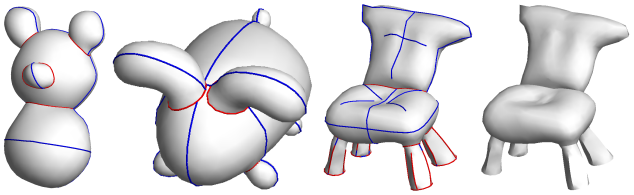


Figure 17: *Results obtained from first-time novice users. Model creation took 10, 10 and 20 minutes, respectively.*

We have conducted an informal user study to test FIBERMESH. We trained first-time novice users for approximately 10-15 minutes, and then let them create some models (Fig. 17). We also asked a professional 2D animation artist to evaluate our system (Fig. 18). To quote the artist: "One great thing about this system is that one

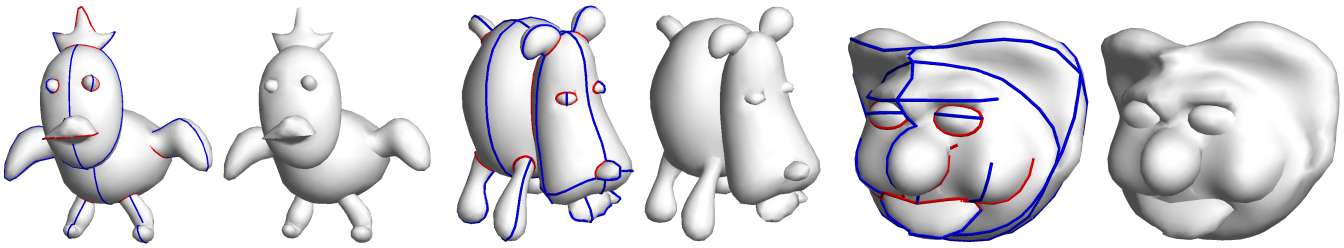


Figure 18: *Creations from a professional 2D animation artist. Modeling took 10, 20 and 20 minutes, respectively.*

can start doodling without having a specific goal in mind, as if doodling on paper. One can just create something by drawing a stroke, and then gradually deform it guided by serendipity, which is very important for creative work. Traditional modeling systems (parametric patches and subdivision surfaces) require a specific goal and careful planning before starting to work on the model, which can hinder the creative process."

Furthermore, we have learned that (a) FIBERMESH indeed supports the skill transfer from traditional 2D sketching to 3D modeling, (b) while the system does require some practice, the amount is reasonable and acceptable and (c) creating separate models first and then merging, as well as animation tools would be very useful.

Our current implementation is written in Java running on the Windows platform. Mesh processing routines are written in Java, but sparse matrix solvers are written in native code (linked via JNI). We are testing the system on an Intel Pentium M 1GHz machine, where it runs in interactive rates. Factorization takes less than a second, and interactive curve deformation (including surface optimization) works in 10-15 fps in most of our examples (600-2000 vertices). We currently process the entire mesh as a single system throughout the deformation, which causes some slowdown when the model becomes complicated. Note though, that it is straightforward to handle larger meshes by editing only a subset of the mesh, while fixing the rest.

5 Future Work

Our current implementation uses a curve only as a series of positional constraints. However, we can expect that curves have more information. For example, when an artist defines a shape with curves, it is often the case that these curves indicate the principal curvature direction of the surface. It is also natural to expect that the character lines form curvature extrema. It might be possible to obtain better (more intuitive and aesthetically pleasing) surfaces by taking these issues into account during optimization. One interesting direction to explore would be to create a quad mesh that follows the direction of the curves. Quad meshes naturally represent principal curvature directions and would make it possible to handle minimum and maximum principal curvatures separately. Quad meshes are also desirable when the user wants to export the resulting model from our system and continue editing it in a standard modeling package.

Multi-resolution (hierarchical) structure would be necessary to construct more complicated models than those shown in this paper. Our current implementation can successfully handle individual body parts such as torso, finger, and face, but the construction of an entire body consisting of these parts would require some mechanism to handle the part hierarchy. One interesting approach would be to allow the user to add a "detailed mesh" on top of a "base mesh" as in multi-resolution approaches. Traditional multi-resolution meshes require fixed mesh topology, but our optimization framework might be able to introduce a topologically more flexible structure.

In a similar vein, we exclusively focused on surface-based control (curves on the surface) in this work. However, in practical modeling purposes, a skeleton based approach might be better in some cases, such as a modeling of simple tube-like arms and legs. Welch and Witkin [1994] actually combined surface-based control and skeleton based control. It might be interesting to explore further into this direction, especially in the context of character animation.

6 Acknowledgements

This work was supported in part by grants from the Japan Society for the Promotion of Science, the Alexander von Humboldt Foundation and the German Academic Exchange Service (DAAD). We would like to thank Alexander Bobenko, Klaus Hildebrandt, Leif Kobbelt, Boris Springborn and Max Wardetzky for helpful discussions, Kenshi Takayama, Yuki Mori and UrumaDelvi for model creation, and the anonymous reviewers for their valuable comments and suggestions.

References

3DS MAX, 2007. Autodesk, `http://www.autodesk.com/3dsmax`.

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3, 587–594.

ANGELIDIS, A., CANI, M.-P., WYVILL, G., AND KING, S. 2006. Swirling-sweepers: constant volume modeling. *Grap. Models 68*, 4, 324–332.

BOBENKO, A. I., AND SCHROEDER, P. 2005. Discrete Willmore flow. In *Eurographics Symposium on Geometry Processing*, 101–110.

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

BOTSCH, M., AND SORKINE, O. 2007. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics*. To appear.

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

DAVIS, T. A. 2004. UMFPACK V4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw. 30*, 2, 196–199.

FELIPPA, C., 2007. Nonlinear finite element methods. `www.colorado.edu/engineering/CAS/courses.d/NFEM.d/`.

FU, H., AU, O. K.-C., AND TAI, C.-L. 2007. Effective derivation of similarity transformations for implicit Laplacian mesh editing. *Computer Graphics Forum 21*, 1, 34–45.

GINGOLD, C., 2007. SPORE's magic crayons. Game Developers Conference.

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

IGARASHI, T., AND HUGHES, J. F. 2003. Smooth meshes for sketch-based freeform modeling. In *ACM Symposium on Interactive 3D Graphics*, 139–142.

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

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3, 1134–1141.

KARA, L. B., AND SHIMADA, K. 2007. Sketch-based 3D shape creation for industrial styling design. *IEEE Computer Graphics and Applications 27*, 1, 60–71.

KARPENKO, O. A., AND HUGHES, J. F. 2006. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Trans. Graph. 25*, 3, 589–598.

KARPENKO, O., HUGHES, J. F., AND RASKAR, R. 2002. Free-form sketching with variational implicit surfaces. *Computer Graphics Forum 21*, 3, 585–594.

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

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

MARKOSIAN, L., COHEN, J. M., CRULLI, T., AND HUGHES, J. 1999. Skin: a constructive approach to modeling free-form shapes. In *ACM SIGGRAPH*, 393–400.

MAXIS, 2007. SPORE ™. Electronic Arts, `www.spore.com`.

MAYA, 2007. Autodesk, `http://www.autodesk.com/maya`.

MORETON, H. P., AND SÉQUIN, C. H. 1992. Functional optimization for fair surface design. In *ACM SIGGRAPH*, 167–176.

NEALEN, A., AND SORKINE, O., 2007. A note on boundary constraints for linear variational surface design. Technical Report, TU Berlin.

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

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2006. Laplacian mesh optimization. In *ACM GRAPHITE*, 381–389.

PERRY, R. N., AND FRISKEN, S. F. 2001. Kizamu: a system for sculpting digital characters. In *ACM SIGGRAPH*, 47–56.

SCHAEFER, S., WARREN, J., AND ZORIN, D. 2005. Lofting curve networks with subdivision surfaces. In *Symposium on Geometry Processing*, 105–116.

SCHMIDT, R., WYVILL, B., SOUSA, M., AND JORGE, J. 2005. ShapeShop: Sketch-based solid modeling with blobtrees. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 53–62.

SCHNEIDER, R., AND KOBBELT, L. 2001. Geometric fairing of irregular meshes for free-form surface design. *Computer Aided Geometric Design 18*, 4, 359–379.

SEDERBERG, T. W., ZHENG, J., BAKENOV, A., AND NASRI, A. 2003. T-Splines and T-NURCCs. *ACM Trans. Graph. 22*, 3, 477–484.

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

SORKINE, O., AND COHEN-OR, D. 2004. Least-squares meshes. In *Shape Modeling International*, 191–199.

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

SORKINE, O. 2006. Differential representations for mesh processing. *Computer Graphics Forum 25*, 4, 789–807.

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

SURAZHSKY, V., AND GOTSMAN, C. 2003. Explicit surface remeshing. In *Eurographics Symposium on Geometry Processing*, 20–30.

TOLEDO, S. 2003. TAUCS*: A Library of Sparse Linear Solvers*. Tel Aviv University.

TURK, G., AND O'BRIEN, J. F. 2002. Modelling with implicit surfaces that interpolate. *ACM Trans. Graph. 21*, 4, 855–873.

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

WARDETZKY, M., BERGOU, M., HARMON, D., ZORIN, D., AND GRINSPUN, E. 2007. Discrete quadratic curvature energies. *CAGD (to appear)*.

WELCH, W., AND WITKIN, A. 1994. Free-form shape design using triangulated surfaces. In *ACM SIGGRAPH*, 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. Graph. 23*, 3, 644–651.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. *Computer Graphics Forum 24*, 3, 601–609.

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

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