Smooth Meshes for Sketch-based Freeform Modeling

Takeo Igarashi
Computer Science Department, The University of Tokyo
takeo@is.s.u-tokyo.ac.jp

John F. Hughes
Computer Science Department, Brown University
jfh@cs.brown.edu

Abstract

This paper describes a framework for introducing visually smooth surfaces into sketch-based freeform modeling systems. An existing sketch-based freeform modeling system generates rough polygonal meshes with uneven triangulations after each operation. Our approach generates a dense, visually smooth polygonal mesh by beautifying and refining the original rough mesh. A *beautification* process generates near-equilateral triangles with a near-uniform distribution of vertices to mask the noise and bad sampling of the uneven mesh. The vertices are distributed on a smoothed surface that approximately interpolates the original mesh. *Refinement* generates a smooth, dense mesh by subdividing the beautified mesh and moving the vertices to the interpolative surface. The smooth interpolative surface is computed via implicit quadratic surfaces that best fit the mesh locally in a least-squares sense.

Keywords: Polygonal Meshes, Subdivision, Beautification, Skin, Implicit Surfaces, Sketch-based Modeling.

1 INTRODUCTION

Teddy [5] introduced a nice sketch-based modeling interface, but the resulting models were rough polygonal meshes. Their triangulations were uneven and the models had many undesirable small bumps and dents; such artifacts were introduced by almost all operations in the system. One could subdivide the mesh [9,20], but the resulting shape was not visually smooth because of the uneven triangulations. Our goal here is to introduce visually smooth surfaces like those seen in parametric and implicit models [17] to sketch-based modeling systems for free-form objects.

Our approach is to beautify and refine the irregular polygonal meshes resulting from the original Teddy algorithms (Figure 1). A *beautification* process, based on the Skin algorithm [11], generates near-equilateral triangles with a near-uniform distribution of vertices on the surface to hide irregularities in the original polygonal model; then *refinement* generates a dense polygonal mesh that smoothly interpolates the beautified mesh.

Beautification and refinement are guided by an implicit smooth surface that approximately interpolates the polygonal mesh. We compute implicit quadratic surfaces that best fit the mesh locally in a least-squares sense, and move the vertices to the surface during beautification and refinement. The implicit surfaces only *approximately* interpolate the mesh, and C^1 continuity among adjacent surface pieces is not guaranteed. This is not acceptable if one wants to use the implicit surface as final output, but works well for guiding the beautification and refinement of polygonal meshes. In addition, our framework is intended to apply to simple rotund objects without small details, such as those in Teddy.

One can smooth meshes with geometric fairing [1,6,14], but these methods are designed to remove high-frequency noise from dense polygonal meshes with fairly uniform vertex distributions, such as those arising from 3D scans; they do not work well for the uneven, coarse meshes seen in Teddy. They also tend to make the surface drift away from the original mesh. We avoid this problem by fitting smooth surfaces to the mesh in a least-squares sense.

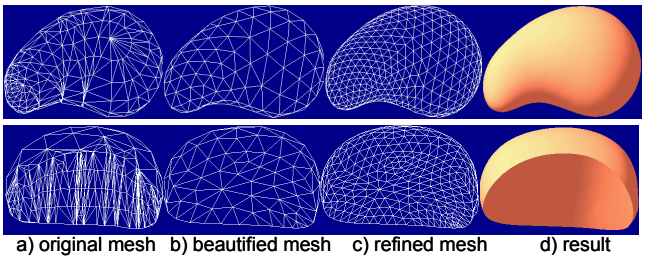


Figure 1: Overview of the algorithm. The system (a) constructs an uneven polygonal mesh from freeform strokes, (b) beautifies the mesh, (c) refines it, and (d) displays the refined mesh using smooth shading.

Our framework gives a basis for exploring various modeling operations with smooth surfaces. Given the built-in beautification mechanism, one can focus on the design of algorithms that construct arbitrary polygonal meshes without worrying about mesh quality or noise.

2 ALGORITHMS

Our basic representation for 3D geometry is a polygonal mesh. In response to editing operations, our system first generates an irregular polygonal mesh based on the algorithms introduced in Teddy [5]. Then we beautify the mesh internally and show the smoothly shaded refined mesh to the user. The algorithms have parameters that depend on the size of the models. The models are scaled to have their largest extent be 1.0.

2.1 Overview

The system maintains three polygonal mesh representations for each 3D model (Figure 2). The first is the *skin mesh*, which is the primary mesh for representing the target 3D shape. It adjusts itself over time through *beautification*. The second is the *skeleton mesh*, which is the irregular polygonal mesh created directly from the input strokes and serves as the reference for guiding the skin mesh during beautification. The third is the *visible mesh*, which is a dense, smooth polygonal mesh displayed on the screen as feedback to the user. The visible mesh is created from the skin mesh by *refinement* and is rendered using smooth shading. It is important to separate the visible mesh and skin mesh for efficient computation of the geometry. We describe beautification and refinement in detail in the following sections.

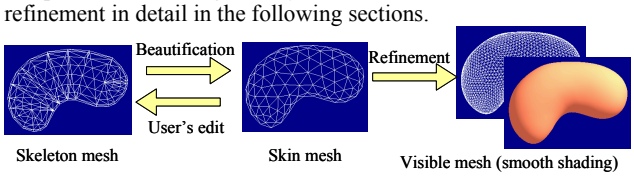


Figure 2: Three mesh representations.

When the user performs an editing operation, a copy of the skin mesh is modified to reflect the new geometry (Figure 3). This new geometry (whose triangulation is uneven and contains bumps and dents) is used as a *new* skeleton mesh; a new skin (which starts from this new skeleton mesh and gradually beautifies itself) is created from it. The user always sees the smooth visible mesh

obtained through refinement.

When a modeling task is finished, the system stores the skin mesh as output. The user can use the mesh as a lightweight polygonal model or as a control mesh for subdivision[1], and can also store the visible mesh if a dense polygonal mesh is desired.

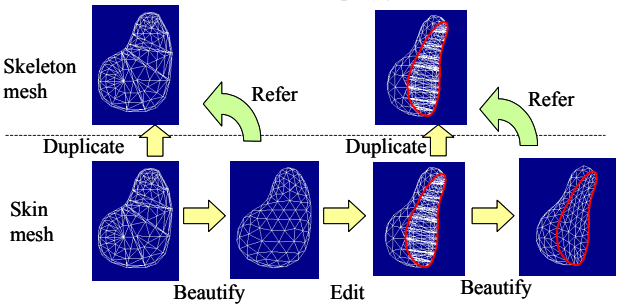


Figure 3: An editing sequence.

Edges along curves representing sharp ridges and creases are labeled as *sharp*. For example, the edges along the intersection loop resulting from a cut are labeled as sharp. We avoid blending surface normals of surrounding polygons at sharp edges so that smooth shading does not mask the sharp features. The Skin algorithm maintains the constraint that the sharp edges remain aligned along the curve [11].

2.2 Mesh Beautification

Mesh beautification aims to generate a mesh with near-equilateral triangles and a near-uniform vertex distribution while preserving some original overall shape, including sharp edges. Our algorithm is based on the Skin algorithm [11]. The vertices of our skin mesh move as particles around the skeleton, repeatedly adjusting their position and connectivity. Each skin vertex is associated with the nearest point on the skeleton mesh (called the *tracking point*). The main difference between our representation and that of Skin is that while Skin generates a *distance surface* around the skeleton with a certain offset, our beautification process tries to generate a surface that approximately *interpolates* the original skeleton mesh. One can obtain similar results simply by setting the offset to zero, but in the original Skin algorithm this actually shrinks the mesh (Figure 4 top). The amount of shrinkage is small if the skeleton mesh is dense, but is still problematic because the shrinkage accumulates through repeated edit-beautification cycles. This also occurs in other topological fairing techniques [7,16] because they insert new vertices on the existing polygonal surface.

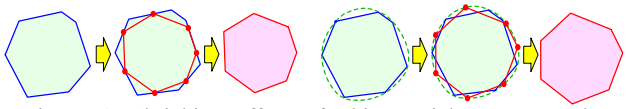


Figure 4: Shrinking effect. If Skin particles stay on the skeleton mesh, the resulting mesh gets smaller than the original (left). To prevent shrinking, the particles must move along an interpolative smooth surface (right).

To address this issue, we move the Skin particles along a smooth surface that approximately interpolates the skeleton mesh (Figure 4 bottom); we describe this surface in the next section.

2.2.1. Implicit quadratic surfaces

The many algorithms for creating interpolative parametric surfaces generally exhibit some artifacts due to the lack of global continuity [10]. Global optimization techniques can generate beautiful surfaces, but they are generally very slow [12]. Variational surfaces, represented by radial basis functions, are also globally (generically) smooth surfaces [17], but it is difficult to maintain a particular topology with them, and they sometimes exhibit unintuitive oscillations. Our approach is to compute implicit quadratic surfaces that best fit the mesh locally in a least-squares sense. This quadratic representation effectively eliminates small bumps and dents because of its limited degrees of freedom, and the least-squares fitting to neighboring vertices generates an aesthetically pleasing smooth surface from a coarse polyhedron.

Levin's approach [8] also uses least-squares fitting, but it locally computes a parametric surface while we locally fit implicit surfaces in 3D space (which makes it possible to fit shapes like ellipsoids perfectly). His approach also requires repeatedly solving a minimization problem when computing multiple positions on a surface. This would be prohibitively expensive when moving the skin vertices on the surface. On the other hand, the approach avoids the shrinkage problem mentioned above.

The implicit quadratic surface is computed for each skeleton vertex using nearby vertices as fitting targets. The quadratic function is formulated as

$$f(p) = f(x, y, z) = Ax^2+By^2+Cz^2+Dxy+Eyz+Fzx+Gx+Hy+Iz+J,$$

and the surface is implicitly defined as $f(p)=0$. We use the nearest 13 vertices around the vertex (including the vertex itself) as targets for fitting.[2] These are collected by a local search around the target vertex, which stops at edges labeled as sharp (Figure 5 left). To establish an orientation and to increase robustness, we also include extra low-weight constraints in the computation. These are obtained by moving each vertex in the direction of its temporary normal (the average of the surrounding polygon normals) with predefined offsets (±0.05 units). The system tries to fit the surface so that $f(p)$ becomes 0 at the target vertices, 1 at outside constraints, and –1 at inside constraints (Figure 5 right). Constraints are given smaller weights (0.01).

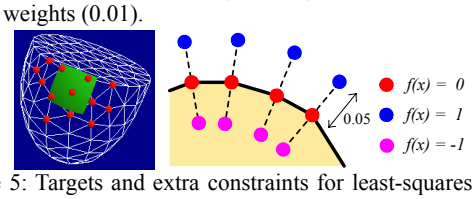
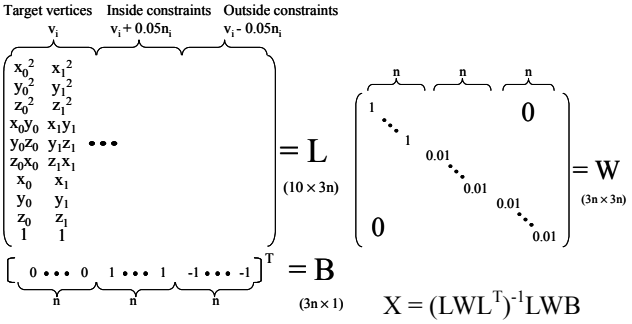


Figure 5: Targets and extra constraints for least-squares fitting. Red points indicate target vertices and the green surface represents the resulting implicit quadratic surface (left). We use 13 target vertices and additional in and out constraints (right).

The objective function for the least-square fitting is formulated as

$$E(f) = \sum_{p\in\text{targets}} (f(p))^2 + 0.01\{ \sum_{p\in\text{outsideconstraints}} (f(p)-1)^2 + \sum_{p\in\text{insideconstraints}} (f(p)+1)^2 \}$$

In matrix form,



The actual least-square fitting is done by solving the matrix system above, where X denotes the unknown vector of coefficients ($X^T=\{A,B,C,D,E,F,G,H,I,J\}$). The weighted overconstrained fitting

[1] The result of subdivision is slightly smaller than the visible mesh. For more accurate results, one can optimize the control mesh so that the result of subdivision faithfully matches the visible mesh [3].

[2] In a near-equilateral mesh, they are the vertices of the six triangles around the center and those of the six triangles around them.

problem is $WL^t X = WB$; multiplying by L on both sides leads to a solvable system. Such a system is solved once per vertex of the mesh.

Once we have the quadratic function for each skeleton vertex, we compute the target position for each skin vertex based on its tracking point on the skeleton mesh. If the tracking point is at a vertex, we simply use the quadratic function associated with the vertex. If the tracking point is at an edge, we compute the target position using each of the two quadratic functions associated with the edge's end points and linearly interpolate them according to the position on the edge. Similarly, the system uses quadratic functions associated with the three corners when the tracking point is on a triangle. To compute the position on the implicit quadratic surface, we apply a simple Newton's method three times, using the tracking point as initial value. This works reasonably well because the initial value is already close to the solution. For vertices lying on a sharp edge, we compute two implicit quadratic surfaces, and then move the vertex to one surface and then to the other in sequence using Newton's method.

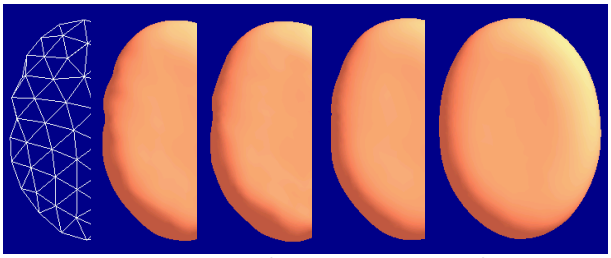


Figure 6: Comparison of various interpolations. The original mesh is subdivided twice and the vertices are moved to the surface defined by each interpolation scheme. Figures are rendered using smooth shading.

Figure 6 demonstrates the advantages of our approach. Local parametric interpolation (PN triangles [18]) and interpolative subdivision (butterfly subdivision [20]) exhibit small dents near the ridge that topological fairing techniques [7,16] cannot hide. Interpolation using radial basis functions [17] and our quadratic fitting both efficiently recover the smooth surface. An alternative solution to the problems arising in this example is to control the meshing so that edges are aligned to ridges; then silhouette problems are not so evident [19]. But this is difficult to do in general, and is in conflict with the behavior of skin.

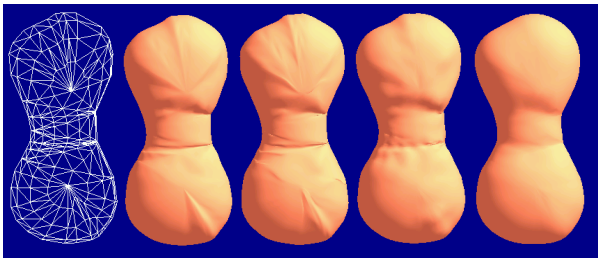


Figure 7: Comparison of various interpolations for the original mesh. This figure is generated in the same way as Figure 6. Existing schemes interpolate the original mesh *exactly*, which inevitably amplifies the small noise in the original mesh.

The piecewise quadratic surface approach may be unsuitable for some applications because of its approximating nature, but it works well for our purpose for several reasons. First, it generates a smooth surface from an uneven mesh with small bumps and dents; other methods are deliberately sensitive to these irregularities (Figure 7). Second, it is reasonably fast for interactive operation. Third, the implicit representation lets us move particles to the desired surface quickly, which can be difficult when using an interpolative subdivision scheme [6,9,20].

2.2.2. Computation of target edge length

The skin algorithm requires a target edge length for guiding the remeshing process; edges should be shorter at high-curvature regions and longer at low-curvature regions. A typical approach to computing surface curvature is to use the immediate neighbors of each vertex [13,15,16], but this can be unstable when applied to uneven meshes. We therefore use the implicit quadratic surface described in the previous section to compute the local curvature. The curvature for a skeleton vertex p is computed as follows. We compute the Hessian matrix $Hf(p)$ – the array of all second partial derivatives of f – and then the eigenvalues of

$$A = \begin{pmatrix} b_1^t Hf(p) b_1 & b_1^t Hf(p) b_2 \\ b_2^t Hf(p) b_1 & b_2^t Hf(p) b_2 \end{pmatrix}$$

where $\{b_1, b_2\}$ is an arbitrary orthonormal basis for the tangent plane at p. The principal curvature k_m is then $e_1 / ||\nabla f(p)||$ where e_1 denotes the larger eigenvalue of A [2, 4]. For vertices along a sharp curve, we use the curvature of the curve. Given k_m, we set the target edge length to $0.8/ k_m$. To prevent excessively long or short edges, we clamp to a minimum and maximum edge length[3].

This procedure determines the desired target edge length for each vertex, but these values may not be appropriate from a more global point of view. Figure 8 illustrates the problem. The low-curvature point v suggests a long edge length, but the long edges at v fail to represent the high curvature region *near* v. To prevent this, we impose the following constraint to the target edge length, using $L(p)$ to denote the target edge length at vertex p: "For every vertex u whose distance to a vertex v is smaller than $L(v)$, $L(u)$ must be equal to or larger than $L(v)$." To satisfy the constraint, the system searches the neigbors U of each vertex v and sets $L(v)$ to max $(L(u), |v\text{-}u|)$ if $L(u) < L(v)$ and $u \in U$. We use mesh distance as the measure of distance between vertices.

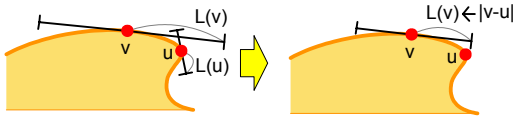


Figure 8: Postprocessing for target edge length.

2.3 Mesh Refinement

Mesh refinement generates a dense, smooth polygonal mesh from the skin mesh as feedback for the user. To do this, we subdivide the skin mesh, and then move the vertices to the quadratic surfaces fitted to the skin mesh. One can obtain smoother surfaces by applying the refinement process repeatedly, but we found that a single refinement generates visually satisfying results as feedback during editing operations.

3 IMPLEMENTATION AND RESULTS

We are developing a prototype modeling system based on our surface representation. The system uses a sketching interface like Teddy's, with some experimental smooth-surface editing operations such as filleting, creasing, and smoothly merging separate meshes (Figure 9). Filleting smooths the sharp corners

[3] The minimum edge length is set to 0.03 and the maximum to 0.3 in the current implementation. In the future we hope to find a way to compute these lengths from properties of the overall shape.

resulting from cutting or extrusion. We apply a geometric fairing algorithm [13] to the skeleton mesh for smoothing. Creasing puts a sharp crease where the user draws a stroke on the object surface. This is done by pushing the stroke edges inwards and labeling them as sharp [11]. For smooth merging we compute the union of the two meshes and put a fillet at the intersection. As in the original Teddy system, these operations simply edit the polygonal mesh; this is significantly easier to implement than it would be with parametric surfaces or implicit surfaces. The accompanying video demonstrates the behavior of the system from the user's point of view.

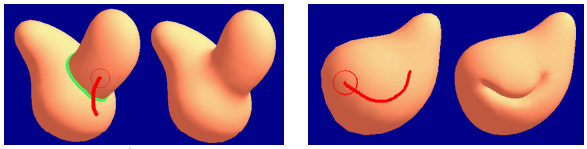


Figure 9: Experimental editing operations.

The system is implemented in Java™ (JDK1.4) and uses directX7 for 3D rendering. It takes a few seconds for skin algorithms to converge to a reasonably beautiful mesh after each editing operation on a high-end PC (AMD Athlon™ 1.54GHz). Figure 10 shows some example 3D models designed using the system (they show the visible mesh in our system). The duck's neck is smoothly merged with the head, and the four legs are smoothly merged to the octopus body. The palm and the bottom of the foot were made by putting fillets at intersections after cutting.

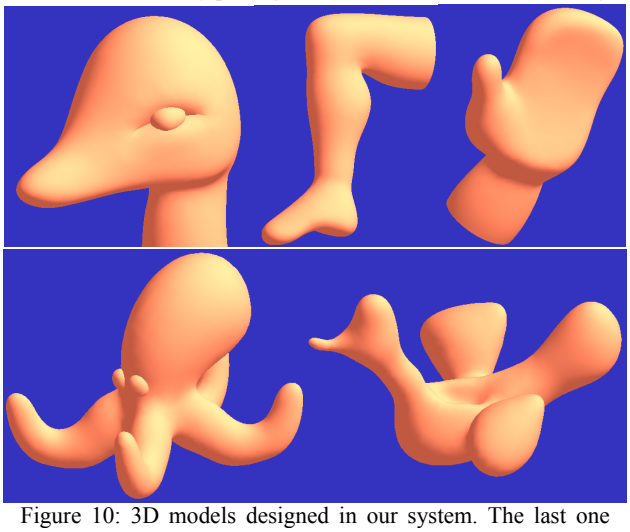


Figure 10: 3D models designed in our system. The last one was designed by a test user and the others by the author.

4 LIMITATIONS AND FUTURE WORK

There are some fundamental limitations in our technique. First, it works only for smooth, rounded surfaces. Second, it requires several empirically set constants. Third, there is as yet no theoretical guarantee of smoothness and robustness.

Our least-squares fitting finds good quadratic functions in most cases, but the resulting surface sometimes has a "discontinuity" in the middle of the target fitting area (Figure 11). This is a fundamental problem of implicit quadratics and our only solution so far is to have more vertices as fitting targets and to use "in" and "out" hints. This prevents the problem in almost all cases in our experience, but we clearly need a more complete solution.

The current implementation can represent sharp edges but not the tip of a cone, i.e., we handle one-dimensional singularities but not zero-dimensional ones. The system automatically rounds off sharp tips in our current implementation. Although this might be acceptable in most cases, we plan to search for an appropriate representation of such points.

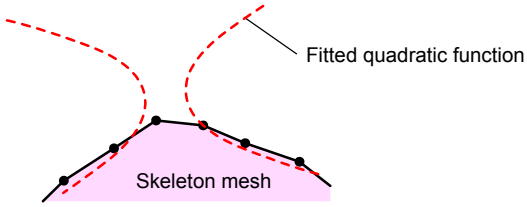


Figure 11: A limitation of quadratic fitting.

References

1. M. Desbrun, M. Mayer, P Schröder, and A.H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. *SIGGRAPH 99 Conference Proceedings*, pages 317-324, 1999.
2. P. Dombrowski. Krümmungsgrößen Gleichungsdefinierter Untermannigfaltigkeiten Riemannscher Mannigfaltigkeiten. *Mathematische Nachrichten*, vol. 38, pages 133-190. Berlin: Akademie Verlag, 1968.
3. H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. *SIGGRAPH 1994 Conference Proceedings*, pages 295-302, 1994.
4. J. Hughes, Differential Geometry of Implicit Surfaces in 3-Space – a Primer. *Technical Report CS-03-05*, Computer Science Dept., Brown University, 2003.
5. T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: a sketching interface for 3D freeform design. *SIGGRAPH 99 Conference Proceedings*, pages 409-416, 1999.
6. L. Kobbelt. Discrete fairing and variational subdivision for freeform surface design. *The Visual Computer, Vol. 16, Issue 3/4,* pages 142-158, 2000.
7. L. Kobbelt, T. Bareuther, H.P Seidel. Multiresolution shape deformations for meshes with dynamic vertex connectivity, *Computer Graphics Forum, Vol 19, No 3*, pages 249-260, 2000.
8. D. Levin. Mesh-independent surface interpolation. To appear in *Advances in Comp. Math*.
9. J. Maillot and J. Stam. A unified subdivision scheme for polygonal modeling. *Eurographics '01 proceedings*, 2001.
10. S. Mann, C. Loop, M. Lounsbery, D. Meyers, J. Painter, T. DeRose, and K. Sloan. A survey of parametric scattered data fitting using triangular interpolants. In Hans Hagen, editor, *Curve and Surface Design*, pages 145-172. SIAM, 1992.
11. L. Markosian, J.M. Cohen, T. Crulli, and J.F. Hughes. Skin: a constructive approach to modeling free-form shapes. *SIGGRAPH 99 Conference Proceedings*, pages 393-400, 1999.
12. H.P. Moreton and C.H. Sequin. Functional optimization for fair surface design. *SIGGRAPH 92 Conference Proceedings*, pages 167-176, 1992.
13. R. Schneider and L. Kobbelt. Geometric fairing of irregular meshes for free-form surface design. To appear in *Computer Aided Geometric Design*.
14. G. Taubin. A signal processing approach to fair surface design. *SIGGRAPH 95 Conference Proceedings*, pages 351-358, 1995.
15. G. Taubin. Estimating the tensor of curvature of a surface from a polyhedral approximation. *Fifth International Conference on Computer Vision,* pages 902-907, 1995.
16. G. Turk. Re-tiling polygonal surfaces. *Computer Graphics, Vol. 26, No. 2, (SIGGRAPH 92),* pages 55-64, 1992.
17. G. Turk and J. F. O'Brien. Variational implicit surfaces. *Technical Report GITGVU 9915*, Georgia Institute of Technology, May 1999.
18. A. Vlachos, J. Peters, C. Boyd, and J.L. Mitchell. Curved PN triangles. *Proc. of Interactive 3D Graphics,* pages 159-166, 2001.
19. J. Vorsatz, C. Rossl, L. Kobbelt, and H. Seidel. Feature sensitive remeshing. *Eurographics '01 proceedings*, pages 393-401, 2001.
20. D. Zorin, W. Sweldens, and P. Schröder. Interpolating subdivision for meshes of arbitrary topology. *SIGGRAPH 96 Conference Proceedings*, pages 189-192, 1996.