Harmonic Functions for Quadrilateral Remeshing of Arbitrary Manifolds

S. Dong *, S. Kircher, M. Garland

University of Illinois at Urbana–Champaign, Department of Computer Science, 201 North Goodwin Avenue, Urbana, IL 61801, USA

Abstract

In this paper, we propose a new quadrilateral remeshing method for manifolds of arbitrary genus that is at once general, flexible, and efficient. Our technique is based on the use of smooth harmonic scalar fields defined over the mesh. Given such a field, we compute its gradient field and a second vector field that is everywhere orthogonal to the gradient. We then trace integral lines through these vector fields to sample the mesh. The two nets of integral lines together are used to form the polygons of the output mesh. Curvature-sensitive spacing of the lines provides for anisotropic meshes that adapt to the local shape. Our scalar field construction allows users to exercise extensive control over the structure of the final mesh. The entire process is performed without computing an explicit parameterization of the surface, and is thus applicable to manifolds of any genus without the need for cutting the surface into patches.

Key words: quad-dominant remeshing, gradient flow tracing, harmonic fields, harmonic 1-forms

1 Introduction

For a great many applications in computer graphics, surface meshes are used as the fundamental representation of geometry. In practice, these meshes are frequently generated by automated acquisition systems; laser scanning and isosurface extraction are particularly common examples. Unfortunately, the meshes produced by these systems are often undesirable in a number of ways.

* Corresponding author.
Email addresses: shendong@uiuc.edu (S. Dong), kircher@uiuc.edu (S. Kircher), garland@uiuc.edu (M. Garland).

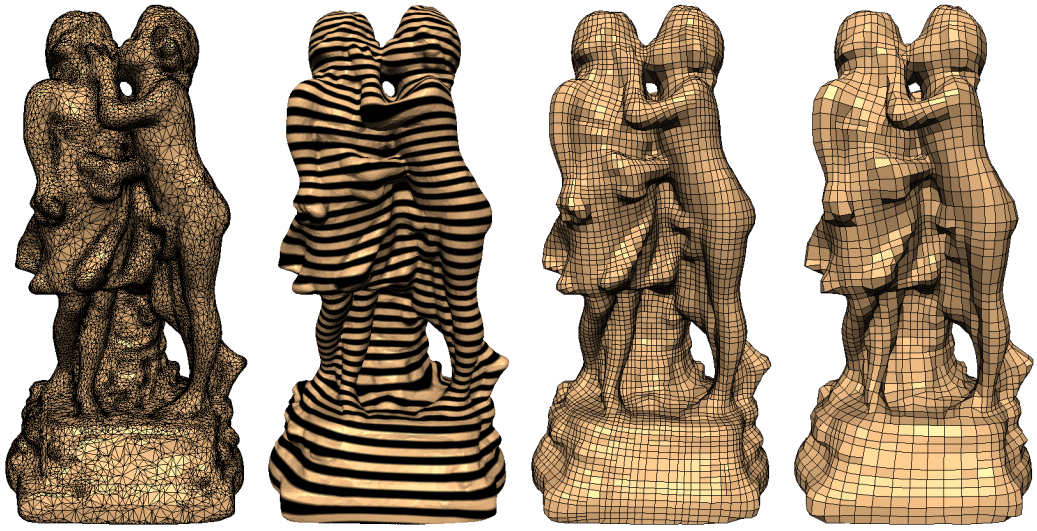


Fig. 1. Overview of our remeshing algorithm. From a triangulated manifold of arbitrary genus, we compute a harmonic scalar field (shown by its isobars) over the mesh. The gradient and isoparametric lines of this field guide our remeshing; two different sampling rates are shown.

For example, they may have a size complexity far greater than required by any particular application. Their tessellations are also frequently insensitive to the actual shape of the surface.

These widespread problems with many raw meshes have lead to a great deal of work aimed at automatically improving their quality. There is now a substantial literature devoted to surface simplification methods that can efficiently produce meshes with far fewer elements while preserving the shape of the object. There are also several examples of remeshing algorithms that seek to produce meshes that resample the surface at some desired resolution, whether simpler or more complex than the original. The method we present here falls into this latter category.

Our goal is to take a given triangulated manifold of arbitrary genus and to produce a new quad-dominant mesh that both preserves the shape of the original and obeys a spacing function provided by the user. The general approach that we take to this problem is to compute two smooth orthogonal direction fields over the surface. The mesh is computed by tracing integral lines through these direction fields. An overview of our method is shown in Figure 1.

The system that we have developed is based on computing a single scalar field over the surface. From this scalar field, we compute both the gradient and an orthogonal vector field. Specifically, we construct harmonic scalar fields, subject to boundary constraints provided by the user. Because the direction field we use is the gradient field of a harmonic scalar field, our direction fields are

guaranteed (by construction) to be very smooth and to be free of extraneous critical points.

Our approach to remeshing has several attractive properties. Unlike many remeshing methods, it can operate on manifolds of any genus and does not require an explicit parameterization. It is also very efficient, processing fairly large models in a matter of seconds, and is easily implemented. We also demonstrate that it produces high quality output meshes and can conform quite well to user-specified spacing requirements.

2 Related Work

The need to improve pre-existing meshes is widespread in computer graphics, as well as most engineering disciplines that rely on finite element and related methods. Depending on the problem domain, there are numerous ways in which a given mesh might be considered unsuitable. There might be too many or too few polygons, too much noise in the vertex positions, too many large angles, and so on. Consequently, a great deal of work, spanning many disciplines, has focused on the problem of producing "good" surface meshes. In this section, we survey the most salient results for our remeshing system.

2.1 Smoothing

Essentially all meshes scanned from real-world objects are inherently noisy. Consequently, many methods have been developed for smoothing mesh surfaces. Laplacian smoothing is one of the simplest and oldest smoothing algorithms [Taubin, 2000]. It both smooths the geometry of the surface as well as the distribution of vertices, tending to produce roughly equilateral triangles. It is also well-known to produce potentially severe shrinkage in the surface, ultimately converging to the barycenter of the mesh. This shrinkage can be essentially eliminated by appropriate filter design [Taubin, 1995]. The parametric smoothing that is characteristic of simple Laplacian smoothing can also be removed by use of the appropriate discrete harmonic weights in the definition of the umbrella operator [Desbrun et al., 1999]. It is from Laplacian smoothing methods such as these that we arrive at the discretization of the Laplacian operator Δu used in constructing our harmonic fields.

2.2 Simplification

The excessive density and shape-insensitive sampling of most scanned meshes has stimulated a considerable amount of work in the area of automatic surface simplification [Garland, 1999]. Of the many methods that have been developed, the majority have been based on either the iterative removal of vertices [Schroeder et al., 1992] or edges [Hoppe, 1996, Garland and Heckbert, 1997, Lindstrom and Turk, 1998]. Other important classes of simplification methods include those based on spatial clustering of the vertex set [Rossignac and Borrel, 1993] and on mesh clustering [Kalvin and Taylor, 1996, Cohen-Steiner et al., 2004].

2.3 Remeshing

Simplification methods strictly reduce the complexity of the initial mesh. In contrast, remeshing methods are typically focused on achieving some particular target sampling density in the output mesh. This may require local refinement in some areas of the mesh, and local coarsening in others. Remeshing methods are frequently categorized as either *isotropic* or *anisotropic* to indicate whether the desired density is a function only of position (isotropic) or of position plus direction (anisotropic).

Some of the earliest simplification methods were in fact remeshing methods. The retiling method developed by Turk [1992] distributed a new set of points over the input surface, replacing the initial mesh with a tessellation of these new vertices. Hoppe's mesh optimization system [1993] also sought to adapt the local mesh density in order to produce the best possible output mesh.

Several isotropic remeshing methods have been described recently in the computer graphics literature. One class of methods operates by parameterizing the surface into the plane, distributing vertices in the parameter domain, and then tessellating these vertices in the plane [Alliez et al., 2002, 2003b]. Such methods must both (1) cut the manifold into a disk and (2) map it into the plane, usually via a conformal mapping.

Taking a somewhat different approach, Surazhsky and Gotsman [2003] remesh the surface using local mesh operations and local parameterizations, obviating the need for mapping the entire manifold into the plane. They subsequently combined their approach with the vertex sampling scheme of Alliez *et al.* [2003b], and demonstrated that this could work quite effectively on large models [Surazhsky et al., 2003].

Sifri *et al.* [2003] also remesh the surface without the need for a parameteri-

zation of the surface. They base their method on computing a scalar distance field over the mesh, associating with each vertex its geodesic distance to a single root vertex. They remesh the surface using an expanding front method which produces strips of triangles aligned with the isocontours of the distance field. In order to construct these strips, they must first segment the mesh into regions where the distance field is monotonic.

Kobbelt *et al.* [1999] explicitly "shrink wrap" a specified mesh onto the input geometry. This method has the benefit of allowing the new connectivity to be precisely specified; the authors demonstrate its use for remeshing with subdivision connectivity. However, even though no global parameterization is used, the shrink wrapping approach requires that the surface be genus 0.

One of the few anisotropic remeshing methods proposed in the computer graphics literature, and also one of the few that produces quad-dominant meshes, was developed by Alliez *et al.* [2003a]. From a smoothed version of the curvature tensor field, they extract two vector fields corresponding to the (smoothed) principal direction fields. The mesh is defined by tracing streamlines through these vector fields. All of this is done in a planar parameter domain, again requiring that the surface be cut and conformally mapped to the plane. The streamline tracing is complicated by the fact that the vector fields may potentially have a large number of critical points, and the direction fields are undefined near umbilic points. Marinov and Kobbelt [2004] have recently extended this work by tracing integral lines directly along the surface, thus dispensing with the need for a global parameterization.

A number of quad-oriented remeshing algorithms have been proposed in the mesh generation literature [Bern and Eppstein, 1995, Owen, 1998]. Notable examples of such methods include square packing [Shimada et al., 1998] and paving [White and Kinney, 1997]. The focus of these methods is to produce quad or quad-dominant meshes for use in finite element analysis. As such, their goals are somewhat different than ours. They emphasize producing meshes that will produce numerically stable simulations, whereas our focus is on preserving the shape of the input manifold with an economical and well-structured mesh.

2.4 Design of Vector Fields on Manifolds

There are several graphics applications in which it becomes necessary to allow the user to design a vector field covering a manifold surface. This is particularly common in texturing applications where the vector field is used to align the texture on the surface. Typically, these systems are designed to allow the user to specify a few constraints on the vector field, which is subsequently extended

over the entire surface automatically.

Several methods for example-driven texture synthesis on surfaces make use of vector fields for local texture alignment. In their lapped textures system, Praun *et al.* [2000] allow the user to specify a few tangent vectors and then interpolate a vector field from these constraints. Turk [2001] uses a combination of relaxation and interpolation to fill in the vector field. Wei and Levoy [2001] exclusively use relaxation. In contrast to these methods, Zelinka and Garland [2003] also allow the user to specify source and sink points for the vector fields, thus exerting greater control over where the singularities occur.

Zhang *et al.* [2004] describe a much more general system for vector field design. In addition to allowing the user to specify critical points, they also provide a mechanism for the user to move and/or cancel critical points.

2.5 Parameterization

The design of automatic parameterization methods has been an area of substantial interest of late and many significant advances have been made [Floater and Hormann, 2004]. As we have already discussed, several existing remeshing algorithms rely on constructing suitable parameterizations of the input manifold. Isotropic methods in particular generally rely on conformal parameterizations — mappings that preserve angles.

Lévy *et al.* [2002] describe a technique for finding conformal mappings by least squares minimization of a "conformal energy". Desbrun *et al.* [2002] formulate a discrete conformal parameterization (DCP) as the solution of a linear system. These two methods are theoretically equivalent [Cohen-Steiner and Desbrun, 2002] but may produce numerically different results in practice.

The linear system derived by Desbrun *et al.* [2002] is structurally identical to the convex combination maps proposed by Floater [1997]. Both solve a constrained Laplacian system. The DCP method uses the well-known discrete harmonic weights [Pinkall and Polthier, 1993, Duchamp et al., 1997]. Floater [1997] originally proposed what he called "shape-preserving" weights, but more recently developed the much simpler mean value weights [Floater, 2003].

Gu and Yau [2003] investigated the construction of a global conformal structure for a given manifold of arbitrary genus. They formalize this as finding a basis for the linear space of all holomorphic 1-forms defined over the edges of the manifold. As we will see, our vector fields are in essence a holomorphic 1-form. A sufficiently uniform holomorphic 1-form can be used to directly remesh the surface using a fully regular geometry image [Gu and Yau, 2003].

Jin *et al.* [2004] provide an algorithm for finding the most uniform holomorphic 1-form given a particular 1-form basis set.

3 Algorithm Overview

We assume that we are given as input a triangulated manifold mesh $M = (V, F)$, composed of a set of vertices V and a set of triangles F. Each vertex i is assigned a position $\mathbf{x}_i \in \mathbb{R}^3$ in a 3-D Euclidean space. While we require that the mesh be manifold, it may be of arbitrary genus, and we place no restriction on the number of boundary curves.

At a high level, our remeshing algorithm consists of the following basic steps:

(1) Build a piecewise-linear scalar field $u : V \to \mathbb{R}$ over the vertices of M.
(2) From u derive two orthogonal piecewise-constant tangent vector fields $\mathbf{g}_1, \mathbf{g}_2 : F \to \mathbb{R}^3$ over the faces of M.
(3) Form a net of polygons over the surface by tracing integral lines of the vector fields $\mathbf{g}_1$ and $\mathbf{g}_2$.

This will produce a non-conforming quad-dominant mesh covering the input manifold M. We conclude with an optional post-processing step.

(4) (Optional) Eliminate all T-junctions and triangulate polygons with more than 4 vertices.

The heart of our system is thus the construction of the vector fields $\mathbf{g}_1$ and $\mathbf{g}_2$, as they will determine the structure of the mesh. Our approach is to derive both vector fields from a single harmonic scalar field u. We construct the scalar field u by solving the Laplace equation $\Delta u = 0$ subject to user-specified constraints. We choose the first vector field $\mathbf{g}_1 = \nabla u$ to be the gradient of this scalar field. The second orthogonal field is then $\mathbf{g}_2 = \mathcal{R}\mathbf{g}_1$, where $\mathcal{R}\mathbf{g}_1$ indicates a counter-clockwise rotation by $\pi/2$ in the local tangent plane [1].

Our choice of harmonic scalar fields to drive the remeshing process has a number of important advantages. Of primary importance is that it guarantees a high degree of smoothness. As the mesh lines will follow the field, this means that the mesh will flow smoothly over the surface. It also guarantees that the vector fields will be free of extraneous critical points. Thus our choice of harmonic scalar fields simplifies the process of tracing integral lines considerably.

Throughout the discussion that follows, we show scalar fields by texturing isobars onto the input surface mesh. This provides a concise depiction of both

[1] This $\mathcal{R}$ operator is also known as the *Hodge star* operator.

the gradient ($\mathbf{g}_1$) and isoparametric ($\mathbf{g}_2$) vector fields. Unless noted otherwise, we show remeshing results without the optional post-process. Therefore, the meshes shown will in general contain a number of non-conforming vertices.

4 Building the Fields

We seek to construct a scalar function $u : V \to \mathbb{R}$ assigning scalar values to each vertex of M. We assume that this function is extended into a piecewise linear function over the entire mesh via linear interpolation within each triangle. Specifically, we are interested in constructing a *harmonic* function, satisfying the Laplace equation $\Delta u = 0$ subject to Dirichlet boundary conditions.

Our formulation of this problem follows that proposed by Ni *et al.* [2004]. It is a scalar analog of linear parameterization methods [Floater, 1997, Desbrun et al., 2002], a connection that we will revisit in Section 9.4. For the moment, we assume that we are provided with a set C of vertices whose values should be constrained. These constraints may be specified by the user in order to control the structure of the mesh, or they may be constructed automatically. We provide the details of determining these constraints in Section 7.

4.1 Harmonic Scalar Fields

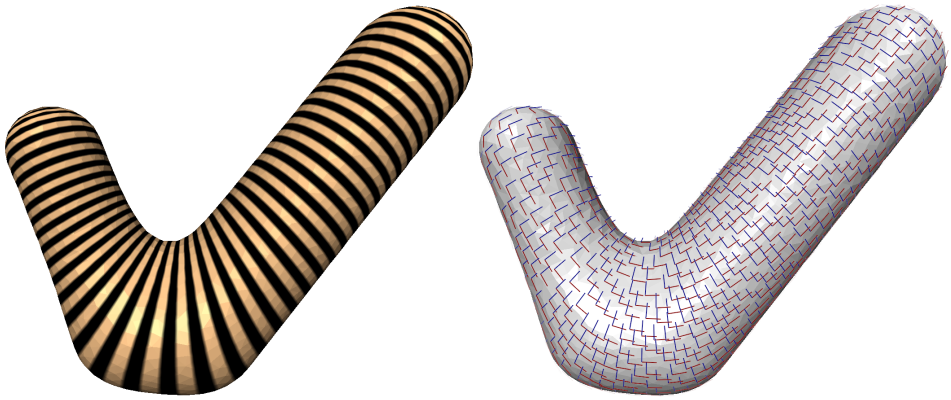
Our goal is to construct a scalar field u such that

$$\Delta u = \nabla^2 u = 0, \tag{1}$$

where Δ is the Laplace-Beltrami operator, subject to the Dirichlet boundary conditions that vertices in the set $C \subset V$ of *constrained vertices* take on the prescribed values:

$$u_i = c_i \quad \text{for all } i \in C. \tag{2}$$

A function u satisfying this constrained Laplace equation is a discrete *harmonic* function. An important consequence of the harmonicity of u is that it will have no local extrema other than at constrained vertices. Furthermore, if all constrained minima are assigned the same global minimum value and all constrained maxima are assigned the same global maximum value, then all the constraints will be guaranteed to be extrema in the resulting field. This is important from the user's perspective as it implies that the mesh will converge precisely at the specified constraint points and will flow smoothly everywhere else.



(a) Scalar field (b) Gradient field

Fig. 2. A harmonic field (shown as isobars) determined by one minimum (at left) and one maximum (at right). The gradient field $\mathbf{g}_1$ (red) and the isoparametric field $\mathbf{g}_2$ (blue) are both shown (with normalized magnitudes).

On a triangulated manifold, the usual discretization of the Laplacian operator is

$$\Delta u_i = \sum_{j \in N_i} w_{ij}(u_j - u_i) \tag{3}$$

where N_i is the set of vertices adjacent to vertex i and w_{ij} is a scalar weight assigned to the directed edge (i, j) such that $\sum_j w_{ij} = 1$. The standard choice for the weights w_{ij} are the *discrete harmonic weights*

$$w_{ij} = -\frac{1}{2}(\cot \alpha_{ij} + \cot \beta_{ij}) \tag{4}$$

where α_{ij} and β_{ij} are the angles opposite the edge. This choice of weights guarantees that u for which $\Delta u = 0$ has minimal Dirichlet energy [Pinkall and Polthier, 1993, Duchamp et al., 1997]. It further has the property that $\Delta \mathbf{x}$ is a discretization of the Laplace-Beltrami operator, meaning that $\Delta \mathbf{x}$ is the gradient of the 1-ring surface area [Desbrun et al., 2002] and approximates the mean curvature normal $\bar{\kappa}\mathbf{n}$ [Meyer et al., 2003].

If we represent the function u by a column vector of its values at all vertices $\mathbf{u} = [u_1 \ u_2 \ \ldots \ u_n]^\mathsf{T}$, we can rewrite the Laplacian operator as a matrix equation:

$$\Delta \mathbf{u} = -\mathbf{L}\mathbf{u} \tag{5}$$

where the matrix $\mathbf{L}$ has entries:

$$L_{ij} = \begin{cases} 1 & \text{if } i = j, \\ -w_{ij} & \text{if } j \in N_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

We seek to solve the linear system

$$\mathbf{A}\mathbf{u} = \mathbf{b} \tag{7}$$

where

$$A_{ij} = \begin{cases} \delta_{ij} & \text{if } i \in C, \\ L_{ij} & \text{otherwise.} \end{cases} \qquad b_i = \begin{cases} c_i & \text{if } i \in C, \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

with δ_{ij} being the Kronecker delta function. This is a sparse system and any efficient sparse linear solver can be used [2]. For very large meshes, the system can be solved more efficiently using a simple multigrid technique [Ni et al., 2004]. It will have a unique non-trivial solution as long as we are provided with at least two distinct constraints. Figure 2a shows an example harmonic field determined by exactly two constraints (one minimum and one maximum).

4.2 Constructing the Vector Fields

Having computed the scalar field u, we can now easily define both orthogonal tangent vector fields. Recall that we wish to find $\mathbf{g}_1 = \nabla u$ and $\mathbf{g}_2 = \mathcal{R}\mathbf{g}_1$.

Let us consider a triangle (i, j, k) whose corners lie at the corresponding positions $\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k \in \mathbb{R}^3$. Furthermore, let $\mathbf{n}$ be a unit normal vector perpendicular to the plane of the triangle. We can easily find the gradient vector $\mathbf{g}_1 = \nabla u$ by solving the linear system:

$$\begin{bmatrix} \mathbf{x}_j - \mathbf{x}_i \\ \mathbf{x}_k - \mathbf{x}_j \\ \mathbf{n} \end{bmatrix} \begin{bmatrix} \\ \mathbf{g}_1 \\ \\ \end{bmatrix} = \begin{bmatrix} u_j - u_i \\ u_k - u_j \\ 0 \end{bmatrix}. \tag{9}$$

Once we have the gradient field, we can construct the orthogonal vector field $\mathbf{g}_2$, which we refer to as the *isoparametric* field. Again, for a given triangle with unit normal $\mathbf{n}$, the orthogonal vector is simply:

$$\mathbf{g}_2 = \mathcal{R}\mathbf{g}_1 = \mathbf{n} \times \mathbf{g}_1 \tag{10}$$

Figure 2 shows a visualization of both the gradient and isoparametric fields derived from the shown scalar field. Note that the vectors have been drawn with their magnitudes normalized. And to avoid clutter we have not drawn vectors in every triangle of the mesh.

[2] Our implementation uses the SuperLU library [Demmel et al., 1999].

The field construction process that we have described is always valid, regardless of the topology of the input manifold. The user may specify any number of two or more extremal points and may freely chose whether any particular point is to be a local minimum or maximum. No matter the choice, a field u satisfying (7) exists.

The Euler characteristic relates the genus g of the manifold to the number of its vertices, edges, and faces

$$\chi = 2 - 2g = |V| - |E| + |F| \tag{11}$$

By applying Morse theory, we can also relate the Euler characteristic to the number of *critical points* — the minima, maxima, and saddle points [3] — of the function u

$$\chi = n_{\min} - n_{\text{saddle}} + n_{\max} \tag{12}$$

The effect of the choice of extremal constraints is thus clear: the more extremal points constrained by the user, the more saddle points will exist in the field. As saddle points require extra care on the part of the remeshing procedure it is desirable to minimize their number. One of the important properties of our harmonic scalar fields is that they provably minimize the number of saddle points for a given number of extremal constraints [Ni et al., 2004, Steiner and Fischer, 2001].

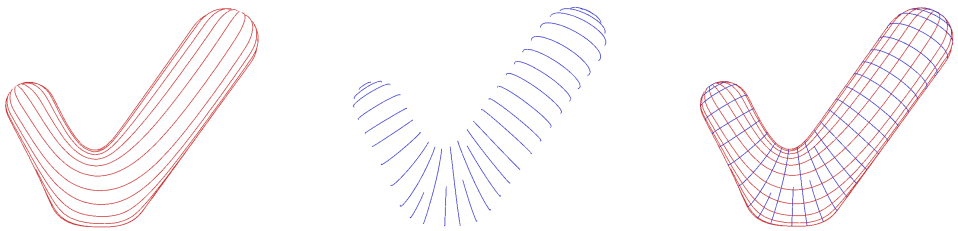
5 Flow Line Tracing

At this point, we have constructed two piecewise constant vector fields $\mathbf{g}_1, \mathbf{g}_2$ that are everywhere orthogonal. We will build the new surface mesh by tracing the locally orthogonal families of integral lines of these vector fields. Our basic approach is inspired by the tracing procedure used in the anisotropic remeshing algorithm of Alliez *et al.* [2003a]. However, the details of our technique are rather different, as we perform this tracing directly on the surface rather than in a planar parameter domain.

The first vector field $\mathbf{g}_1$ is defined by the gradient of the scalar function u. Therefore, we refer to its integral lines as the *gradient flow*. The integral lines of the second vector field $\mathbf{g}_2$ coincide with the isoparameter lines of u, thus we refer to them as the *isoparametric flow*. Both flows cover the entire surface,

[3] For simplicity, we assume saddles of multiplicity 1 and no flat edges. Ni *et al.* [2004] provide details on removing these restrictions.

and they will ultimately form the edges of the polygons in the output mesh. Figure 3 shows a simple example of their structure.



(a) Gradient flow (b) Isoparametric flow (c) Overlayed

Fig. 3. Tracing integral lines of the given vector fields produces the (a) gradient and (b) isoparametric flows. They will combine (c) to form the edges of the mesh.

In this section, we address the tracing of these flow lines. We use a simple and efficient method for tracing integral lines of the vector field. The spacing of these lines is controlled by a spacing function, locally adapted in response to the curvature of the surface. Once both families of flow lines have been traced, they are used to construct the output mesh. Details of this process are given in Section 6.

5.1 Tracing A Single Flow Line

A *flow line* is a piecewise linear curve over the surface that is an integral line of one of our underlying vector fields. Each flow line consists of a sequence of *flow nodes* connected by straight-line segments. These flow nodes, which form the vertices of the flow lines, coincide with the intersection of the integral line with the edges of the original mesh.

The placement of flow lines is controlled by the location of certain *seed points*. Given a particular seed point, our goal is to trace out the integral line on which it lies. We will revisit the problem of placing these seed points shortly. For the moment, let us assume that we are given a single seed point and that we wish to trace its corresponding integral line.

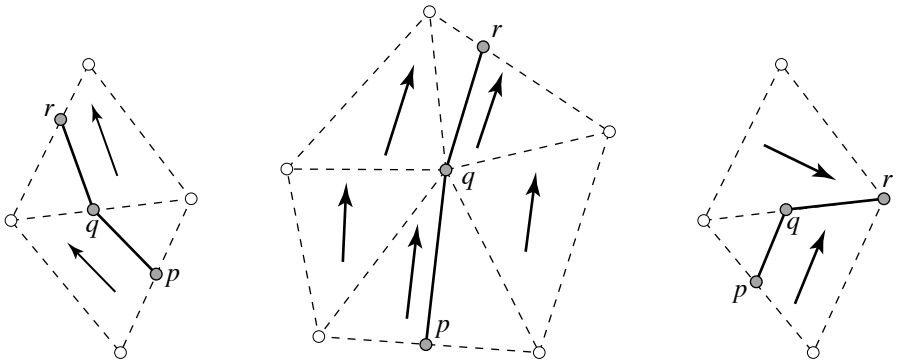
Accurately tracing the integral lines of an arbitrary vector field, even in the plane, is in general a difficult problem. It requires careful use of stable numerical methods (e.g., the Runge–Kutta integration used by Alliez *et al.* [2003a]). Integration of the vector field directly on a 2-manifold further complicates this process. In our specific case, the expense and complexity of sophisticated integration schemes is unnecessary. The gradient field is extremely smooth and all integral lines converge at well-defined extremal points. The orthogonal field

consists of the isoparameter lines of a known scalar field, eliminating the need for explicit integration entirely.

Because of the special structure of the fields we are tracing, we can use a straightforward scheme to find the flow lines. This is both efficient and stable. Even for meshes with hundreds of thousands of faces, all flow lines can be traced in less than 10 seconds. Flow line tracing can also be trivially performed directly on the triangulated surface, avoiding the need for parameterizing the surface. Because we do not require even a local parameterization, our method can also easily handle surfaces of arbitrary genus.

5.1.1 Gradient Flow

A gradient flow line will always begin at a local minimum of u and then follow the gradient field $\mathbf{g}_1 = \nabla u$ until it reaches a local maximum of u. Given an arbitrary seed point on the surface, we must therefore trace a flow line in both the positive and negative gradient directions until we reach the bounding maximum and minimum, respectively.



(a) Regular case (b) Vertex case (c) Edge case

Fig. 4. Flow lines propagate through the gradient field by following the gradient vector across triangles. Special cases arise when the flow intersects a vertex or when the vector field converges toward an edge.

In the general case, the endpoint q of the flow line will lie along some edge of the input mesh. The flow line will have already crossed one of the triangles incident on this edge, and we wish to extend it across the other incident triangle. This situation is illustrated in Figure 4a. Given that segment $\overrightarrow{pq}$ is the last segment of the flow line, we wish to extend the flow line with segment $\overrightarrow{qr}$. In the regular case, we can construct this segment simply by walking across the triangle from q in the direction of the gradient vector within this triangle. Thus both $\overrightarrow{pq}$ and $\overrightarrow{qr}$ will be parallel to the gradient vectors in their respective triangles.

Two special cases can arise in the tracing of the gradient field. First, a flow node may coincide with a vertex of the mesh (as in Figure 4b). In this case, we must examine each of the triangles incident on this vertex. We advance the flow line through the triangle in which we can move the furthest in the gradient direction. Second, the flow field may converge on an edge (as in Figure 4c). In this case, we cannot extend the segment $\overrightarrow{qr}$ through the neighboring face, and must instead simply follow the edge connecting the two triangles. This is the only case in which the flow fails to follow the vector field exactly. It is a fairly uncommon occurrence, as this case typically only arises near saddle points and our field construction automatically minimizes the number of saddle points in the field.

5.1.2 Isoparametric Flow

The isoparametric flow lines can be traced using an even simpler process than the gradient flow lines. This is because they are, by definition, the isoparameter curves of a *known* scalar field. Within any triangle, we can thus compute exactly where the isoparameter lines should cross a particular edge.

This fact is also important for guaranteeing that the isoparametric flow has the right structure. Except in the presence of open boundaries, each isoparametric flow line is obviously a closed curve. In a general integration scheme, accumulated numerical error might cause a traced isoparameter line to fail to connect with itself. We can avoid such errors entirely. Because we explicitly follow the exact isoparameter curve, we can guarantee that the resulting curve is always closed.

5.2 Placing Flow Lines

As mentioned previously, tracing of a specific flow begins at a seed point. Herein, we describe how we place these seeds so as to effectively sample the surface. This relies on the existence of two piecewise linear, positive real-valued functions h_1 and h_2, defined over the vertices of the mesh, which specify the desired spacing of the gradient and isoparametric flows, respectively. The generation of h_1 and h_2 will be addressed in the next section. Here, our goal is to place flow lines such that they obey the spacing requirements of these two functions. That is, the distance between neighboring gradient flows is governed by h_1; and the distance between isoparametric flow lines is governed by h_2. Throughout this section, the distance between two neighboring flows is given by the distance along the surface in the orthogonal flow direction. This is the most relevant distance to measure, because distance along the orthogonal direction reflects the length of the edges of the output mesh. Since h_1 and

h_2 are independent, our method can produce both isotropic and anisotropic meshes.

The path of a flow line is determined completely by the underlying vector field. Thus, we need only determine where to start tracing a flow (i.e., where to place seed points) and when to stop. Stopping is fairly simple, so it will be described first. Since we want flow lines to be spaced a certain distance apart (given by either h_1 or h_2) we stop tracing a new flow when it gets too close to an existing flow of the same type. To get good layout of flows, we stop tracing a flow when it gets closer than τh_1 to an existing flow, where τ is a user controllable tolerance parameter. We employ an octree data structure to accelerate the detection of nearby flow lines. All meshes shown in this paper were generated with $\tau = 0.5$, which we have found to give good results. We also discard flows that are too short to avoid fragmented flows.

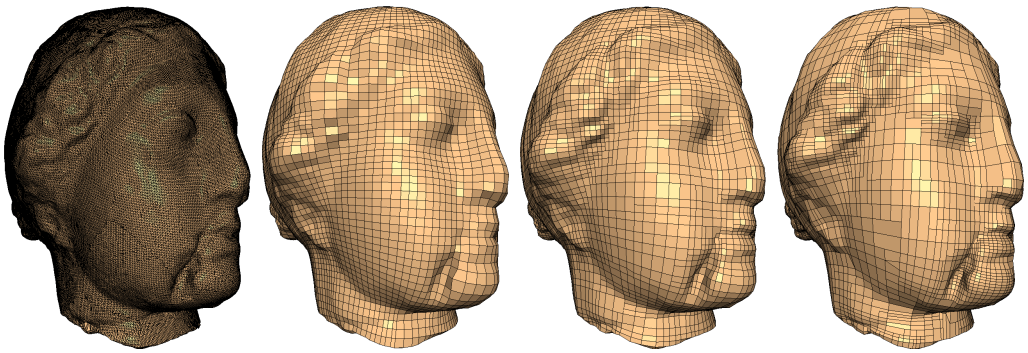
To place seed points, we extend the evenly spaced streamline method described in [Jobard and Lefer, 1997] to work directly on a surface in 3D. We place an initial seed at each extremal point and one on each feature corner (see §5.4). This guarantees that each extremum and each corner will be properly covered. Saddle points can cause instability in the flow line tracing. Therefore we isolate them by placing a seed point in each of the up and down directions around the saddle — a total of 4 seeds for regular saddle points. This guarantees that we will properly sample the saddle point and the presence of these seeds will prevent the tracing of other flow lines through the saddle.

Once we have the initial seeds, we can begin tracing flow lines one at a time. After each flow line is traced, we place additional seed points to either side of it, at even intervals along the flow line. The distances from the flow line to the newly placed seed points are given by h_1 (or h_2 if we are tracing isoparametric flows). The spacing between seeds generated by the same flow is not crucial (although a minimum number of seeds should be generated by each flow). The important aspect here is that seeds are placed the proper distance away from the flow, as measured along the orthogonal flow direction.

All the seeds are stored in a priority queue, with the key being the distance to its generating flow. Each time we select the seed with the largest key, because they are located in regions with larger value of the sampling distance function (h_1 or h_2), and are therefore more likely to produce longer flow lines.

5.3 Sampling Distance Functions

The sampling distance functions h_1 and h_2 can be completely specified by the user. However, usually a user desires either a uniform, isotropic mesh, or a curvature-sensitive, anisotropic mesh (the benefits of anisotropy in remeshing



(a) Input (b) $\alpha = 0$ (c) $\alpha = 10$ (d) $\alpha = 300$

Fig. 5. Comparison of different values of the curvature sensitivity parameter, α. The output meshes have the same face count, but different α values. For high α value, notice how faces cluster near curved regions such as the dent near the mouth and in the hair.

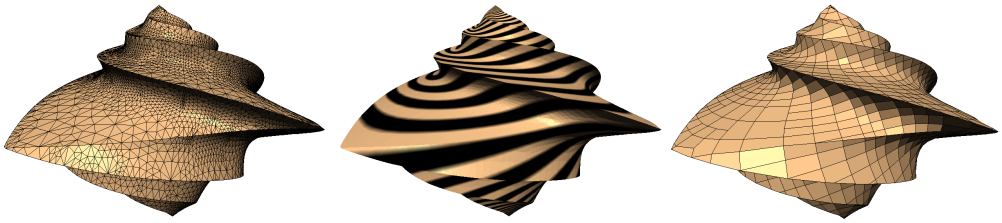
have been described extensively by Alliez *et al.* [2003a]). In this section, we provide our default functions for computing h_1 and h_2 from the local curvature of the surface and two user-supplied parameters. These heuristic functions are chosen so that the user can easily control the overall (isotropic) density of the output, as well as the degree of anisotropic curvature sensitivity.

We begin with a user-supplied isotropic sampling distance function h. This can be either a single constant (i.e., a uniform sampling distance), or it can vary over the mesh. A second user parameter, α governs the degree of anisotropic curvature sensitivity. We then construct h_1 and h_2 by

$$h_1 = \frac{h}{1 + \alpha \log_{10}(1 + \kappa_n^2)} \qquad h_2 = \frac{h}{1 + \alpha \log_{10}(1 + \kappa_n^1)} \tag{13}$$

where κ_n^1 and κ_n^2 are the normal curvatures of the surface in the directions of $\mathbf{g}_1$ and $\mathbf{g}_2$, respectively. The normal curvatures are estimated by intersecting a normal plane with the surface, in the direction of interest. We then perform a few Laplacian smoothing steps on the curvature values to remove high-frequency variations. Since h_1 governs the spacing *between* flow lines through $\mathbf{g}_1$, it is related to κ_n^2, not κ_n^1. We use the log scaling function because we have found, empirically, that it provides a pleasing feel to the user as they vary the sensitivity parameter α. Generally α is set to be less than 20, but can be much higher (see Figure 5). If $\alpha = 0$, then the remeshing is isotropic ($h_1 = h$ and $h_2 = h$).

Note that gradient flow lines converge at extrema. Thus, only a few flows will actually get very close to the extrema, as they must obey the spacing requirements given by h_1. Even though this behavior is correct, allowing flow lines to get closer to each other near extrema usually produces nicer results.



(a) Original (b) Field (c) Remesh

Fig. 6. Preservation of automatically detected feature lines. Notice that the sharp edges are preserved even though the field is not necessarily aligned with them. Two constraints have been placed, one on each of the top and bottom tips.

Therefore, by default, we decrease the isotropic spacing h slightly near all critical points.

5.4 Features

A good number of meshes have sharp features that must be preserved when remeshing the surface. This is particularly true of mechanical parts, but even relatively smooth meshes, like the hand, often have sharp features (see the base of the hand in Figure 14). Given a set of feature edges, we construct a feature skeleton consisting of *chains*. A *chain* is a sequence of neighboring feature edges beginning and ending at *darts* (vertices with only one feature edge neighbor) or *corners* (vertices with at least three feature edge neighbors), or forming a closed loop. If a chain is too short, it is considered noise and is thrown away.

When the flow lines are traced, they may cross feature edges. In this case, a flow node will actually lie on a feature edge, so we mark this flow node as a "feature". During the mesh construction phase (§6), additional edges will be added to connect all the neighboring "feature" flow nodes of the same feature chain. Note that corner vertices will automatically be preserved in the output mesh, since they are included in the initial seeding.

A number of methods have been proposed in the literature for detecting feature edges (e.g., [Watanabe and Belyaev, 2001, Page et al., 2002]). Our implementation currently uses a simple tagging scheme based on edge dihedral angle. Figure 6 shows an example of a model containing visually essential sharp features of non-trivial shape. As we can see, these feature lines are preserved very well by the remeshing system, even though they are not aligned with the flow lines.

6 Constructing the Output Mesh

Once we have a complete set of flow lines for the vector fields $\mathbf{g}_1$ and $\mathbf{g}_2$, we are ready to construct the output mesh. The vertices of this mesh will lie at the intersection of unique gradient and isoparametric flow lines. Its edges will follow the flow lines themselves, thus determining the polygons as well. Our meshing algorithm is done directly on the surface, without the help of a parameter domain.

If we were presented with an arbitrary collection of flow lines, constructing the output mesh in this fashion would essentially be a generalization of finding arrangements of lines in the plane [Edelsbrunner, 1987, Guibas and Sharir, 1993]. However, the set of flow lines we have are highly structured, so we are able to extract the mesh quite efficiently.

6.1 Extracting Connectivity

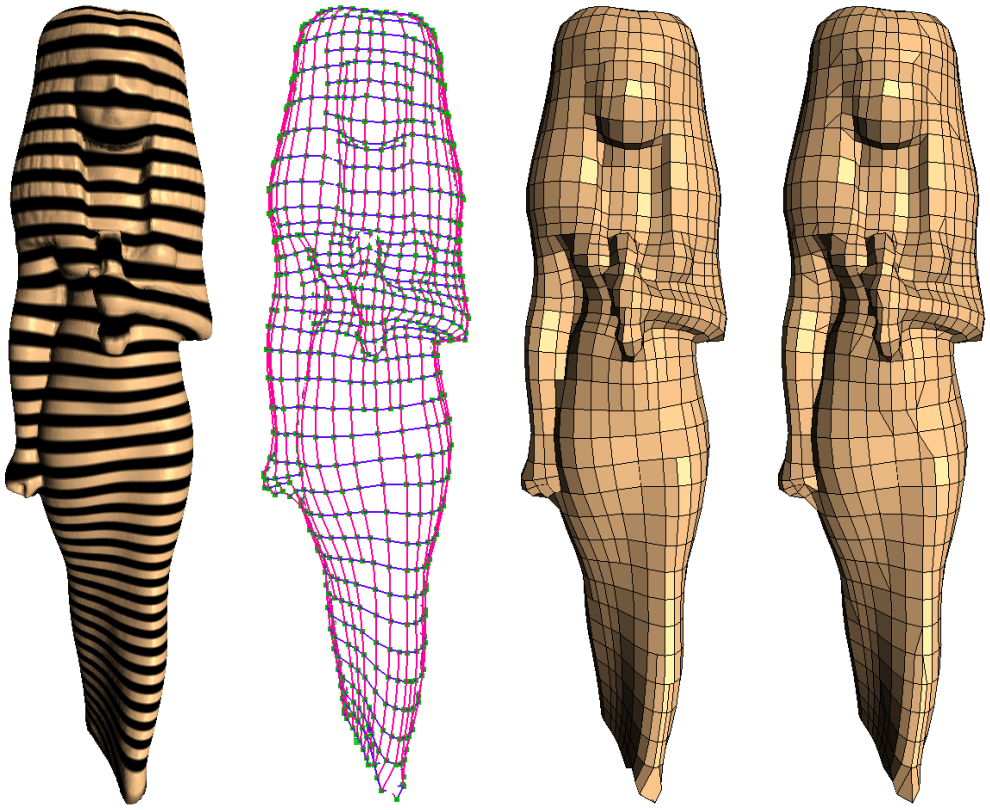
We begin extracting the connectivity of the output mesh by computing all intersections of gradient and isoparametric flow lines. For efficiency, we consider each triangle of the input mesh separately. Within each triangle, the net of flow lines is a (sparse) planar grid of line segments. The set of intersection points within each triangle are easily computed. We exclude all critical points from the set of crossings, as the flow lines either converge or diverge in these areas. Instead, we treat critical points specially during polygon creation.

If a flow node has been marked as a "feature", it is also added to the list of crossings, even though two flow lines do not necessarily cross at that point. Such crossings are called *feature crossings*.

Once the crossings have been detected, we must correctly identify the connections between them. For each flow line, we create a list of the crossings along this flow line sorted by their order of occurrence. Feature crossings are also ordered by their position along the feature chain.

In the normal case each crossing will have 4 neighbors, two each along the gradient and isoparametric flow lines. In rare cases, a crossing will have only 1 neighbor. Such dangling crossings are deleted to provide more well-formed meshes. Also there may be normal crossings that are very close to a feature crossing. In such cases, we eliminate the non-feature crossing by contracting it into the feature crossing.

At this point, we have constructed a graph over the surface, as in Figure 7b. The green knots are the crossings. And the edges connect consecutive crossings



(a) Field (b) Crossings (c) Remesh (d) Post-process

Fig. 7. The 3 stages of mesh generation. First, we build the set of crossings along each flow line. Next we extract a non-conforming mesh from this net of flow crossings. A final post-process produces a conforming mesh composed solely of triangles and quadrilaterals.

along flow lines.

6.2 Polygon generation

Once the connectivity graph is complete (as in Figure 7b), we are ready to generate the polygons of the output mesh (as in Figure 7c). Extracting the faces of this graph is quite straightforward, as the flow lines provide an explicit ordering and orientation for all edges. We traverse each face ring of the graph in counter-clockwise order, emitting a polygon for each such ring.

Because extremal points are excluded from the crossing lists, we must generate additional polygons around each of them in order to close up the mesh. To do this, we simply find the isoparametric flow line closest to a given extremal vertex. We extract the list of vertices along this flow line, and connect

consecutive pairs to the extremal point, forming a triangle fan.

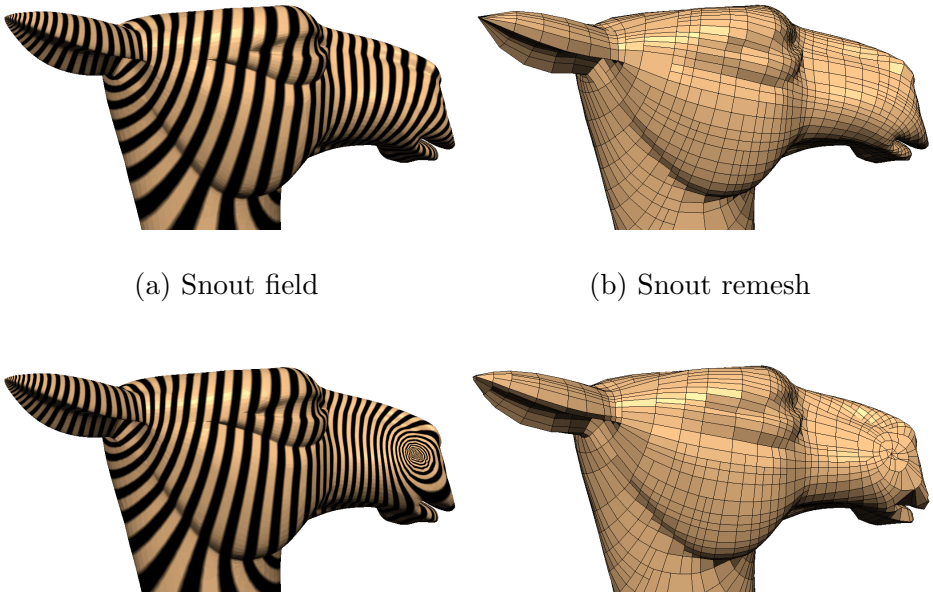
At this point, the mesh is complete. We have generated one polygon for each face ring of the connectivity graph and a triangle fan around each extremal point. The vertices of the final mesh are positioned at their respective locations on the original surface (e.g., where two flow lines crossed). Therefore, all vertices of the output are always guaranteed to lie on the original surface mesh.

As mentioned before, the resulting mesh is generally non-conforming. There will be so-called T-junctions where flow lines were terminated by the local spacing control. This is quite apparent in Figure 7c. For those applications in which non-conforming meshes are undesirable, we perform an optional post-process to remove all T-junctions, adopting the same template-based strategy as Alliez *et al.* [2003a]. We also triangulate polygons with more than 4 sides using a simple ear-cutting algorithm. After post-processing, the mesh consists solely of quadrilaterals and triangles, and is strictly conforming. Figure 7 shows an example of a mesh both (c) before and (d) after post-processing.

7 Placing Extrema

Our remeshing method provides the user a great deal of control over the output mesh. The alignment of quad edges and the location of singularities can both be controlled by the user, through their selection of constraint points for the harmonic field. Usually it is desirable to have several constraints, to ensure that the field follows the geometric structure of the mesh, but only two (a minimum and a maximum) are required. Placing these constraints by hand is generally quick and painless, as the user need only click on a few locations on the mesh. In this section we provide a few guidelines to help the user in deciding where to place extrema for best results. Later, we discuss a way in which the extrema can be found automatically.

The minima and maxima of the scalar field are sources and sinks for the gradient field. Gradient flow lines will flow from a minimum towards a maximum. Thus, for example, we can specify that the quad edges in the remeshing output should flow along the mesh from one protrusion to another, or in any other manner we choose. This gives the user a great deal of control over the way the quad edges are aligned in the output mesh. Generally it is desirable to place minima and maxima on the tips of protrusions, so that the gradient flow lines follow the shape of the surface. In addition, the gradient flow lines will converge at minima and maxima, so placing the extrema on tips of protrusions looks more natural. For example, placing an extrema on the tip of the camel's snout looks better than placing it on the side of the camel's cheek (see Figure



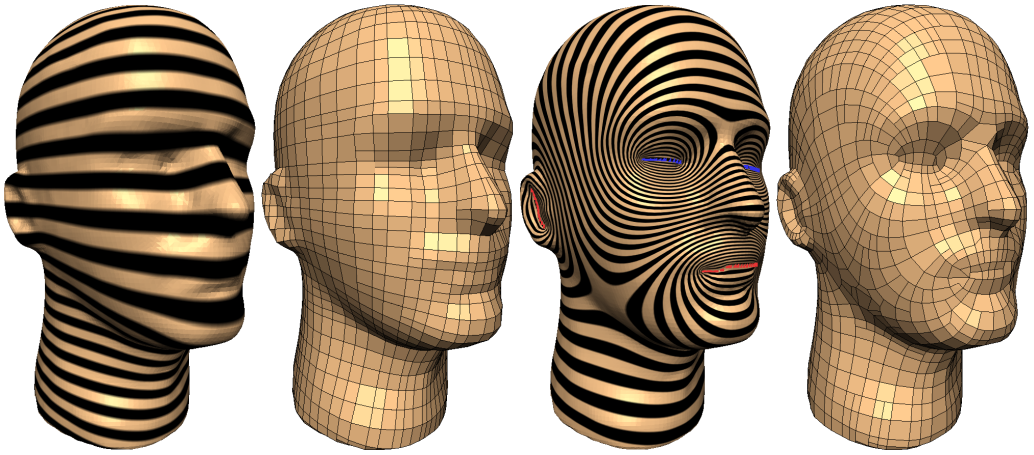
(a) Snout field (b) Snout remesh

(c) Cheek field (d) Cheek remesh

Fig. 8. Different choices of constraints produce different Laplacian fields, and hence different remeshing outputs. When placed manually, extrema should generally be located on local feature points.

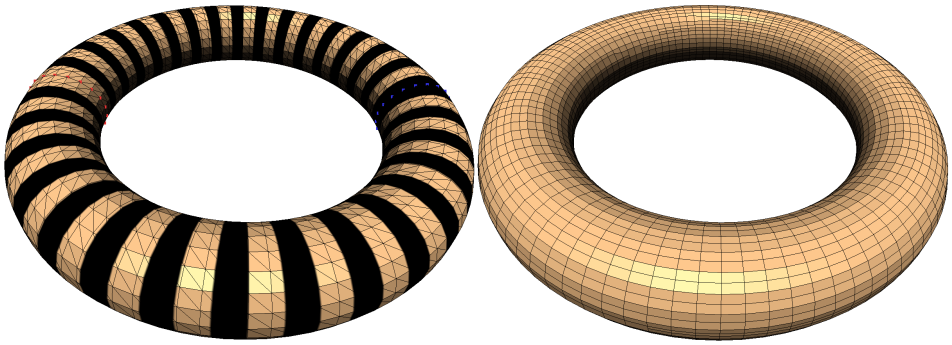
8). There is also some theoretical support for using this "tip of protrusions" heuristic, in terms of reducing the stretching of the harmonic field, as analyzed in [Jin et al., 2004].

Even greater control over the alignment of the field can be achieved by using *curve* or *loop* constraints on the model, rather than only selecting isolated points. A curve constraint consists of a connected sequence of vertices such that each vertex has one or two neighbors constrained to the same value. A loop constraint is a closed curve constraint (each vertex in the loop has exactly two constrained neighbors). Curve and loop constraints are usually unnecessary, but they can be useful in certain situations. For example, they can be used to orient an otherwise lopsided field so that the isoparametric flow lines form shortest path circles around a cylindrical portion of a mesh. They can also be used to force the remeshing to follow the important lines of a human face (see Figure 9). Another very important use of loop constraints is to allow the remeshing of genus-1 surfaces without any singularities (see Figure 10). In our examples, we manually identify these loops, but they can also be computed automatically [Gu and Yau, 2003, Steiner and Fischer, 2004].



(a) (b) (c) (d)

Fig. 9. Usage of curve constraints to improve the artistic properties of the remeshing. (a-b) Single minimum and single maximum on the top and bottom of the head. (c-d) Lines of minima (blue) and maxima (red) have been placed on the eyes, ears, and mouth, generating a more aesthetically pleasing remeshing result.



(a) Input mesh with field (b) Output

Fig. 10. By placing two ring constraints on the torus, we can generate a smooth remeshing result with no singularities. It should be noted that, since we do not make use of the magnitudes of the gradient or isoparametric fields, the ring constraints need not be symmetrically placed.

7.1 Semi-Automatic Constraints

Our method affords the user a significant amount of control over the remeshing, since the placement of the constraints governs the flow of the generated quad edges. However, it is often the case that this amount of control is unnecessary. In addition, using identical values for all extrema of the same type can lead to slightly misplaced saddle points, or undesirable warping, on meshes

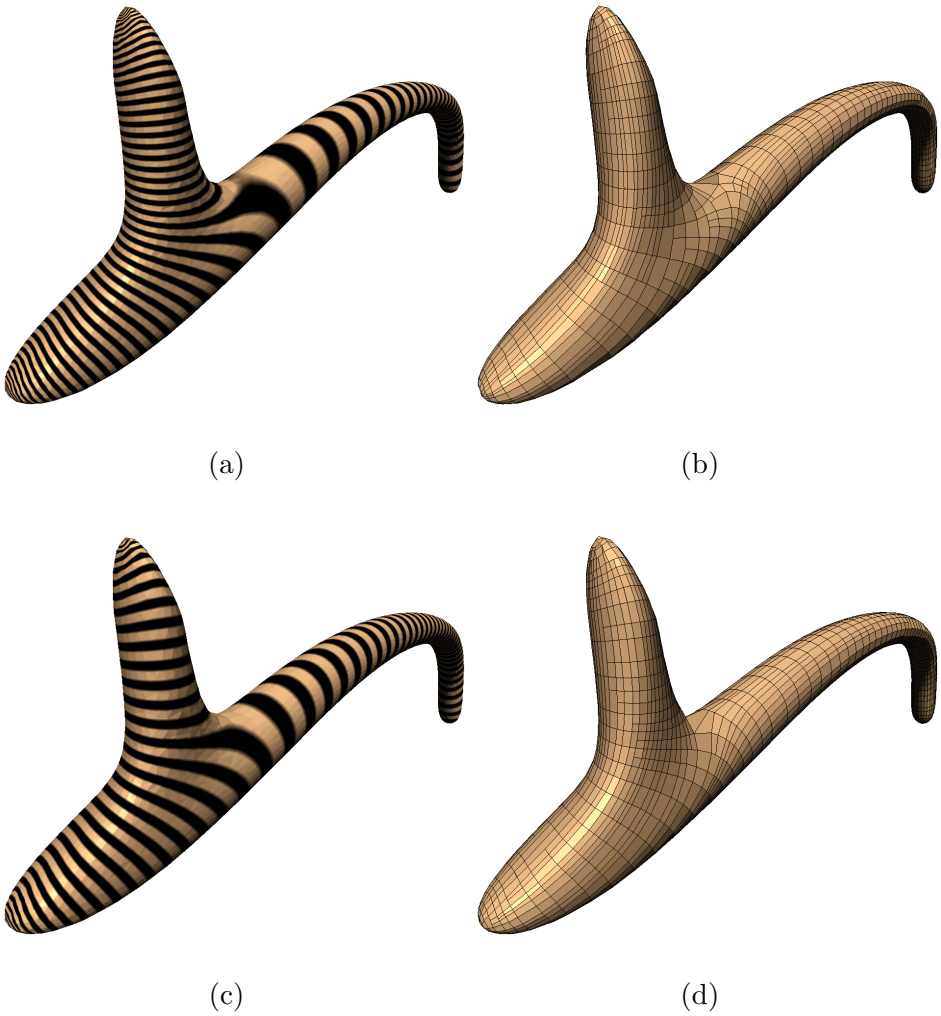


Fig. 11. (a-b) Using equal values for all maxima can result in slightly misshapen fields. (c-d) By using a larger constraint value on the longer protrusion, we obtain the desired behavior. This constraint value was obtained from our Poisson field.

with large asymmetries (as in Figure 11). Therefore, we present a method for automatically determining both the positions and appropriate values for a set of constraints, with minimal user input. This method solves a Poisson equation to determine an initial (non-harmonic) scalar field on the mesh. The extrema of this field are extracted and used as constraints in our harmonic field construction.

We wish to automatically construct an initial scalar field that follows the overall shape of the mesh, from which we will extract appropriate extrema. As mentioned in section 4.1, the magnitude of the discretized Laplace-Beltrami operator is an approximation of the mean curvature of the surface [Meyer et al., 2003]. We seek a scalar field f whose "curvature" matches that of the

surface. To find such a field, we solve a Poisson equation of the form

$$\Delta f = -||\Delta \mathbf{x}||. \tag{14}$$

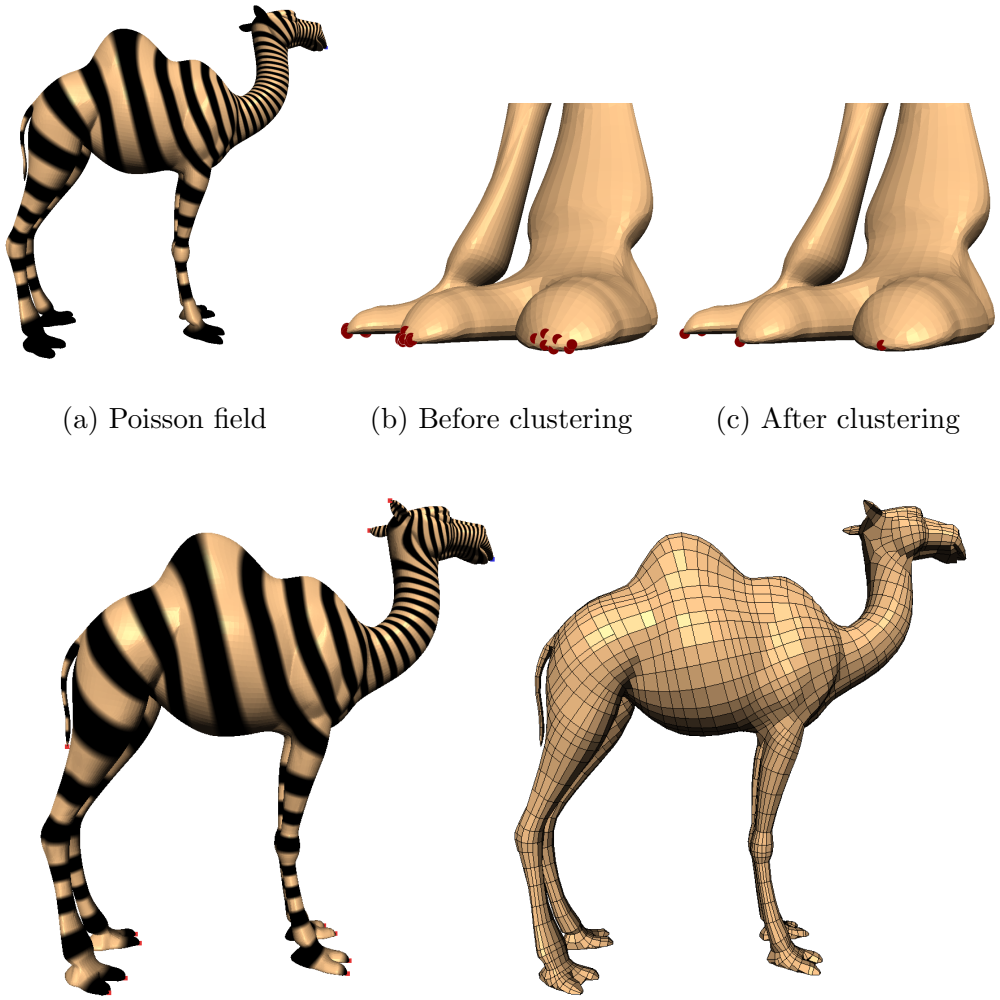
Note that the sign of the right hand side is arbitrary; switching the sign would simply reverse the roles of minima and maxima in the following discussion.

Equation (14) can be discretized and solved in exactly the same manner as equation (1). Only the right hand side of the linear system has changed. For the system to have a unique solution, we must specify one constraint point. Specifying additional constraints is actually undesirable as it generally leads to unwanted undulations in the field. We shall call the solution to equation (14) the *Poisson field*. An example of such a field is shown in Figure 12a.

Assuming that the mean curvature is not everywhere zero, there will be a unique solution to (14) if we specify a single constraint point. This initially specified point will always be a local minimum. Furthermore, it will be the only minimum (ignoring numerical issues). This follows directly from the discretization of (14) and the fact that $||\Delta \mathbf{x}|| \geq 0$. The value of f at a vertex i must be greater than or equal to the weighted average of its 1-ring neighbors, for all i *except* the constrained vertex. Hence, the constrained vertex is the only possible minimum, and is therefore a global minimum (note that this also assumes the weights used in the discretization of the Laplacian operator are strictly positive). In addition, we have observed that the points that are locally most distant from the constrained vertex (measured along the surface) will tend to end up as local maxima.

While it is possible to run the remeshing process directly on the Poisson field, this has a number of potential problems. While these fields tend to conform very well to the shape of the surface, they can develop regions of very low gradient, which can in turn lead to multiple extraneous local extrema (see Figure 12b). In general, such regions have ill-defined gradient directions, and produce undesirable results when remeshed. To solve this problem, we instead extract only the most significant extrema, and construct a harmonic field (via equation (1)) using these extrema.

We use clustering to extract the most significant extrema. The maxima are clustered based on their proximity to one another, and a representative is selected from each cluster. Since we are clustering maxima, we can assume that the most significant extrema in each cluster is that with the highest value of f. These representative extrema are used as the constrained vertices in the subsequent harmonic field construction (Figure 12d). The only two user inputs to this process are the initial constraint point (which generally should be placed on the most prominent protrusion) and the cluster spacing (which governs how close two extrema have to be before they are combined).



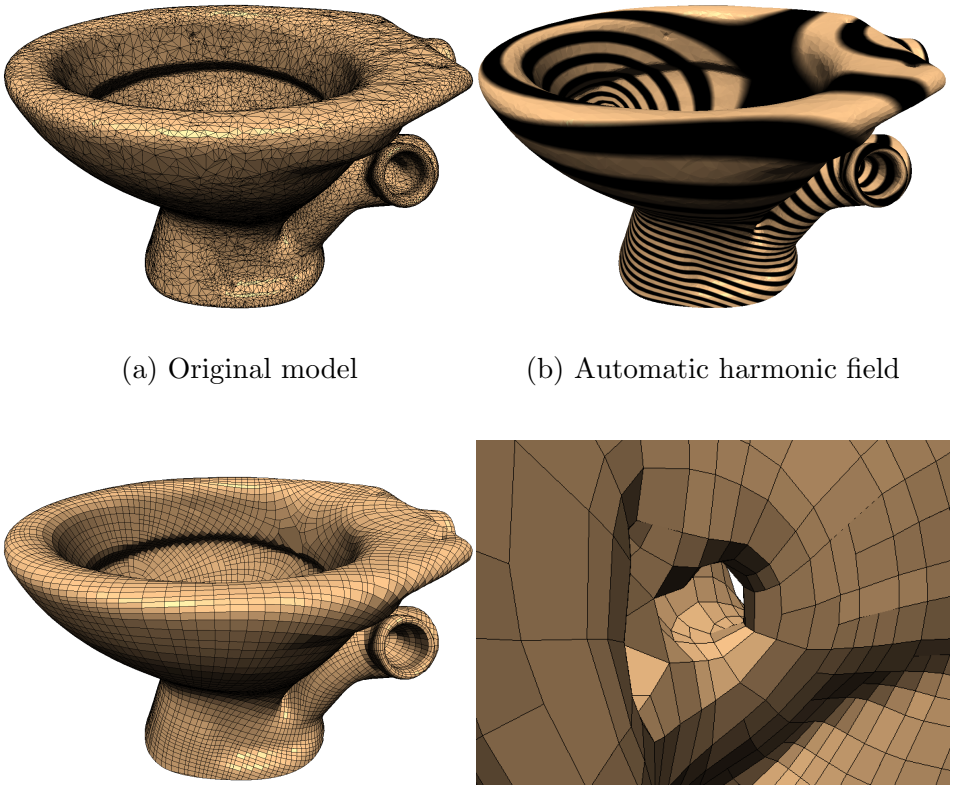
(a) Poisson field (b) Before clustering (c) After clustering

(d) Harmonic field (e) Resulting remesh

Fig. 12. Overview of our constraint detection method. (a) The user places a single constraint point (tip of the camel's nose), and a Poisson field is generated. (b) Due to low gradients and numerical error, multiple extrema (red dots) may occur in locations where there should only be a single maximum. (c) The extrema are clustered to extract a single representative maximum. (d) The 16 extracted maxima and the original minimum are used as constraints for the Harmonic field. (e) The rest of the remeshing algorithm remains unchanged.

An important point in the extraction of local extrema is that the value of the Poisson field at each extremal point is used as the value of the constrained point in the harmonic field construction. This is as opposed to setting all the maxima to the same global value, helping to better position the saddle points, since longer protrusions will tend to get higher field values than short ones (see Figure 11).

The usefulness of the automatic constraint method is demonstrated on the



(a) Original model (b) Automatic harmonic field

(c) Remeshing output (d) Closeup of topological hole

Fig. 13. Semi-automatic constraint method demonstrated on the toilet model. Notice that the complexity of the toilet's shape is captured nicely. The only user specified constraint point was in the middle of the base of the toilet. From this, six distinct maxima were found. The toilet is, in fact, genus four. A closeup of one of the holes (near the bottom of the inside of the basin) is shown in (d), thus demonstrating that the semi-automatic constraint method works on higher genus models as well.

toilet model in Figure 13. This model has a rather complicated shape, and unlike models with arms, the proper location of extrema is less obvious. Thus, having an automatic method is desirable. Notice how the lines in the remeshing output follow the shape of the bowls and pipes. Only one extrema was placed by the user, on the base of the toilet; the remaining six extrema were found automatically.

8 Results

We have already shown several examples of our results. To demonstrate the flexibility of our algorithm, and the quality of the results that it generates, we show several more complex examples in this section. Note that all out-

Model	Fig	h	α	$\lvert V_{\text{in}}\rvert$	$\lvert V_{\text{out}}\rvert$	Running time (sec) Solve	Init	Trace	Mesh	**Total**
sculpture	1(c)	39	4	25,382	8,434	6.4	3.9	5.0	0.6	15.9
	1(d)	27	4	25,382	3,789	6.4	4.0	3.3	0.2	13.9
isis	7(c)	20	6	188,496	1,538	81.6	31.1	5.5	0.1	118.3
head	9(d)	30	4	39,955	2,376	11.6	5.8	3.1	0.1	20.6
hand	14(b)	51	4	38,218	6,070	9.9	5.6	5.0	0.2	20.7
	14(c)	26	4	38,218	1,660	9.9	5.8	2.5	0.1	18.3
	14(d)	12	4	38,218	473	9.9	7.9	1.4	0.0	19.2
octopus	16(c)	30	9	16,554	3,785	3.1	2.7	2.8	0.2	8.8
	16(d)	21	9	16,554	2,107	3.1	3.1	2.1	0.1	8.4
bull	18(b)	29	4	5,002	3,188	1.1	0.8	1.7	0.2	3.8

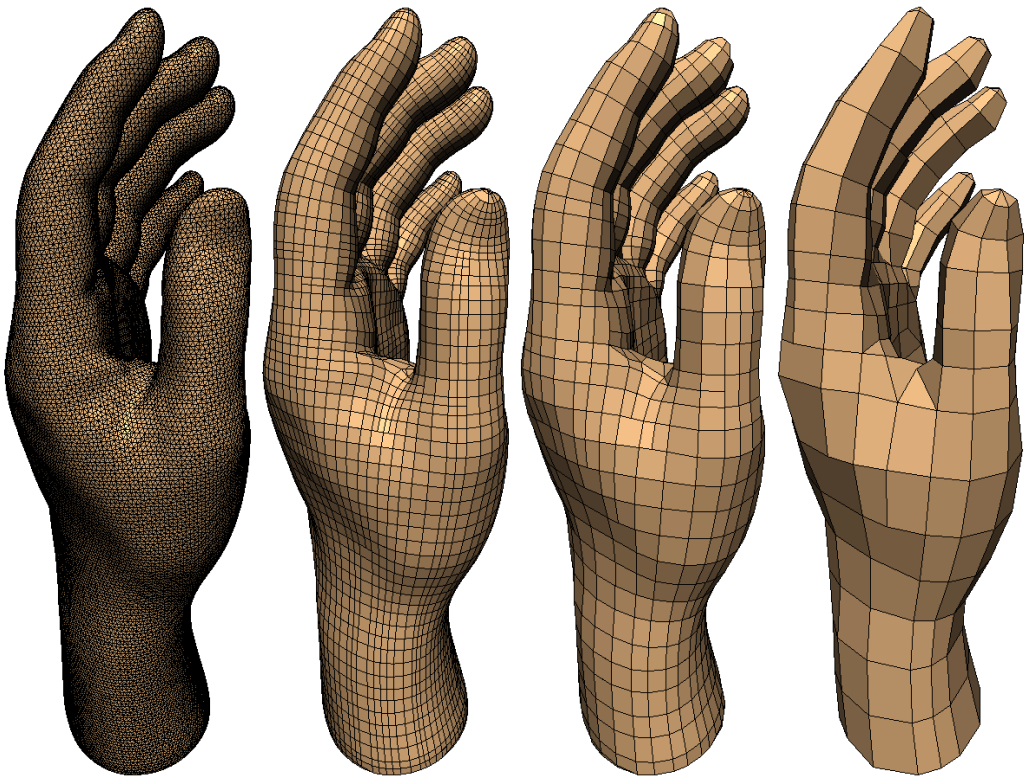
Table 1
Remeshing times for several models. The isotropic spacing (h) and curvature sensitivity (α) parameters are also shown.

put meshes in this section are shown without post-processing, and thus are generally non-conforming. All of the remeshing results were generated using user-specified field constraints, and the specification of these constraints generally required less than 20 seconds.

We begin with a summary of timing results for our experimental implementation. All times were measured on a 2.66 GHz Pentium IV, with 512MB of RAM. The results are shown in Table 1. Running times are split into several categories. *Solution time* includes the assembly and solution of the Laplacian linear system. *Initialization time* measures the time necessary to compute the sample spacing functions h_1, h_2 over the surface and to distribute the initial seed points. *Tracing time* is the time required to actually trace all flow lines over the mesh, and *meshing time* is the time taken to generate the output polygons. All times are reported in seconds, and are exclusive of file I/O time. Overall, our system is quite fast, requiring less than a minute to remesh any model under 100,000 vertices.

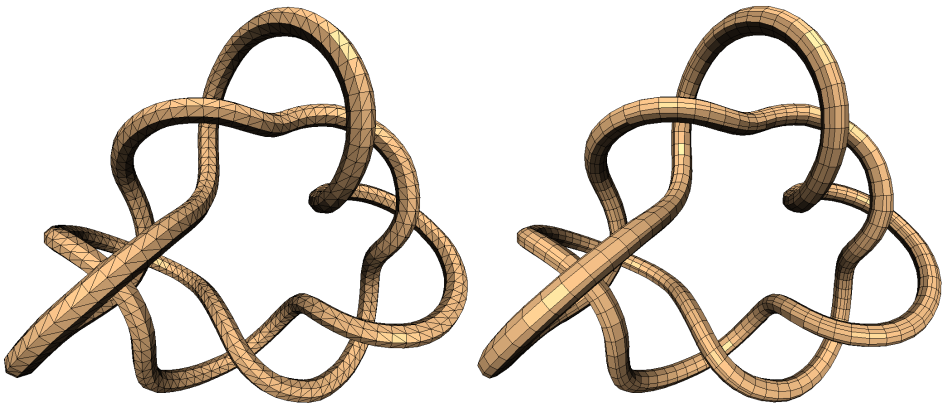
Our first example is a mannequin hand, shown in Figure 14. We see the results of our remeshing algorithm at three different resolutions; the exact choice of the spacing parameter is summarized in Table 1. For this model, we placed a single minimum in the middle of the base of the hand and maxima at the tip of each finger. As we can clearly see, the quad edges align quite nicely with the natural "flow" of the surface. In addition, the shape of the surface remains well preserved even at the lowest sampling density. When producing multiple remeshings of the same model in this way, we can improve efficiency by computing the harmonic field (and the resulting vector fields) only once. In this particular case, this would reduce the total time for subsequent remeshings by roughly 50%.

Despite its complicated geometry, the knot shown in Figure 15a is topologically equivalent to a torus. Thus, as in Figure 10, we can remesh it without any singularities (Figure 15b). Note how we can recover a model with flow lines



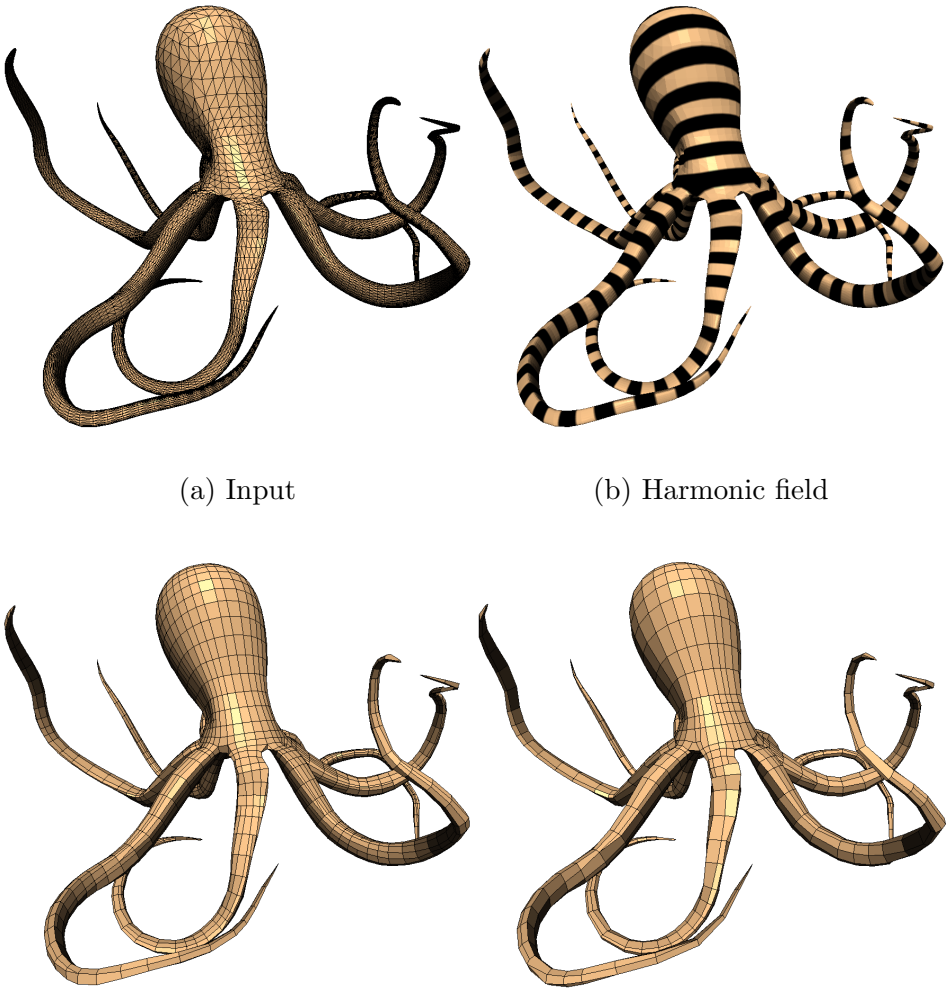
(a) Input (b) Fine (c) Medium (d) Coarse

Fig. 14. Multiple remeshings of a hand capture the shape of the fingers very well, even at low sampling rates.



(a) Input (b) Remesh

Fig. 15. This genus-1 knot can be remeshed without any singularities. In addition, the spacing of the quad edges adapts to the local curvature of the surface.



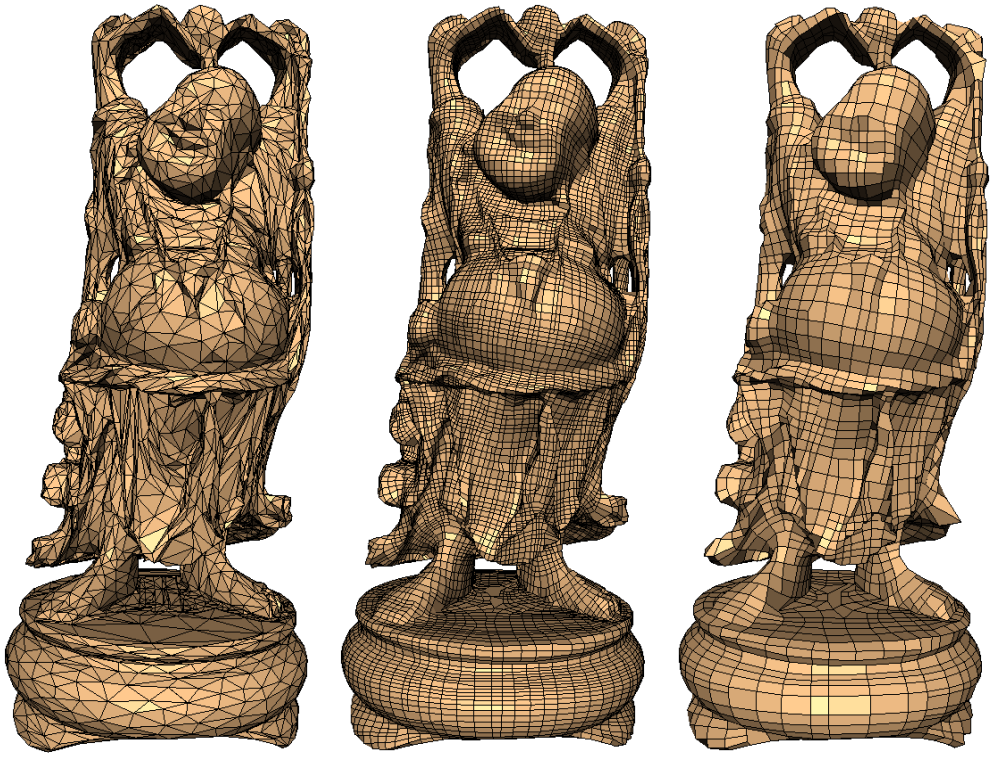
(a) Input (b) Harmonic field

(c) Finer remesh (d) Coarser remesh

Fig. 16. Remeshing results on the octopus. Notice that the quad-edges are aligned along the flow of the tentacles, and that the tips of the tentacles are preserved.

very similar to that of the original mesh (which is composed of triangulated quads). However, in the remeshed version, the density of the quads can be seen to be adapting to the local curvature of the model. For example, notice how the mesh on the rightmost bend is denser than the mesh on the leftmost bend, which has a larger radius of curvature. To generate this result, as with the torus, two minimum radius loop constraints were placed at different (but fairly arbitrary) locations along the knot.

The octopus model shown in Figure 16 presents an interesting challenge because of its long skinny tentacles. This model is particularly problematic for remeshing methods that rely on parameterization of the surface. When flattening this surface into the plane, the long thin tentacles can result in extremely high parametric distortions, making the task of the remesher difficult. In con-



(a) Input (b) Fine remesh (c) Coarse remesh

Fig. 17. Using our method, this genus-6 Buddha statue can be remeshed without the cutting required by parametric methods.

trast, our method requires no parameterization and is therefore unaffected by this difficulty. Nevertheless, the tentacles provide a good test of our edge tracing method, as minor errors in line placement can lead to significant damage to the shape of the tentacles. As we can see from the output meshes shown, our method handles this surface successfully. We placed a total of only 10 extrema: 1 on the top of the head, 1 in its mouth on the underside of the surface, and 1 at the tip of each of the 8 tentacles. In the resulting mesh, the quad edges are well-aligned with the shape of the tentacles, and the position of their tips is preserved. In addition, the effect of our anisotropic sampling can be observed in the stretching of quads along the tentacles.

As we have noted previously, one of the primary advantages of our method is that it does not require a parameterization of the surface. The main consequence of this fact is that our method is able to easily handle manifolds of any genus. As an example, consider the Buddha statue shown in Figure 17. This is a genus-6 manifold which is also geometrically complex. We show two example remeshings of this input surface. The first is actually denser than the input. The second is significantly coarser, yet it still captures the original

(a) Input (b) Field (c) Remesh

Fig. 18. Our field construction and polygon generation algorithms are stable even on extremely irregular meshes.

shape quite well. To achieve such results on this rather complicated surface, we needed to specify only 3 constraints. We placed a single extremum both at the very top of the mesh and in the center of the underside of the base stand. In order to get the best alignment of the mesh with the base, we also placed a single ring constraint around the upper rim of the stand.

The previous example demonstrates that our technique produces stable results even when presented with surfaces that are topologically and geometrically complex. It is also robust in the face of poorly constructed input meshes. Figure 18 shows a mesh, describing the surface of a bull, that is an extremely irregular tessellation. The distribution of the vertices is obviously non-uniform and rather haphazard. Only a small minority of the triangles are well-shaped; the rest all have either very large or very small angles. Despite this ill-formed input mesh, our field construction and polygon generation algorithms both produce pleasingly smooth results. We placed extremal constraints on the nose, horns, feet, and tail. The resulting harmonic field, shown in Figure 18b, is quite smooth given the input mesh. Similarly, the final mesh shown in Figure 18c is also both smooth and faithful to the original shape of the bull.

One natural application of remeshing methods such as ours is to produce coarser approximations of an initially fine mesh (as in Figure 14). This is also the primary goal of the fairly extensive literature on surface simplification. We of course take a slightly different view of the problem here. Simplification methods generally focus on minimizing approximation error and generally consider mesh quality as a secondary concern, if at all. In contrast, producing good quality meshes is our primary concern.

In Figures 19 and 20 we examine the performance of our remeshing system as compared to the QSlim simplification method [Garland and Heckbert, 1997]. The difference in emphasis of error vs. mesh quality is readily apparent in the results. We generated multiple approximations of the hand model shown in

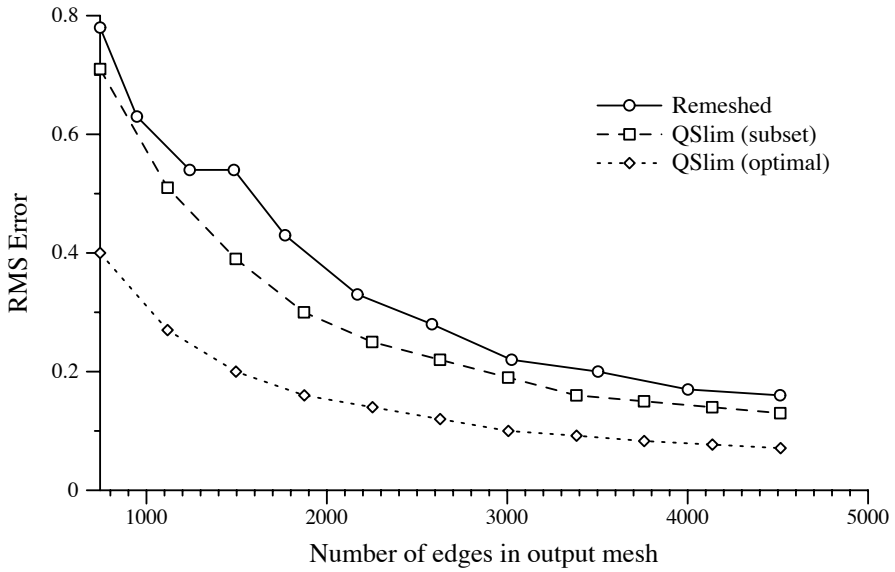
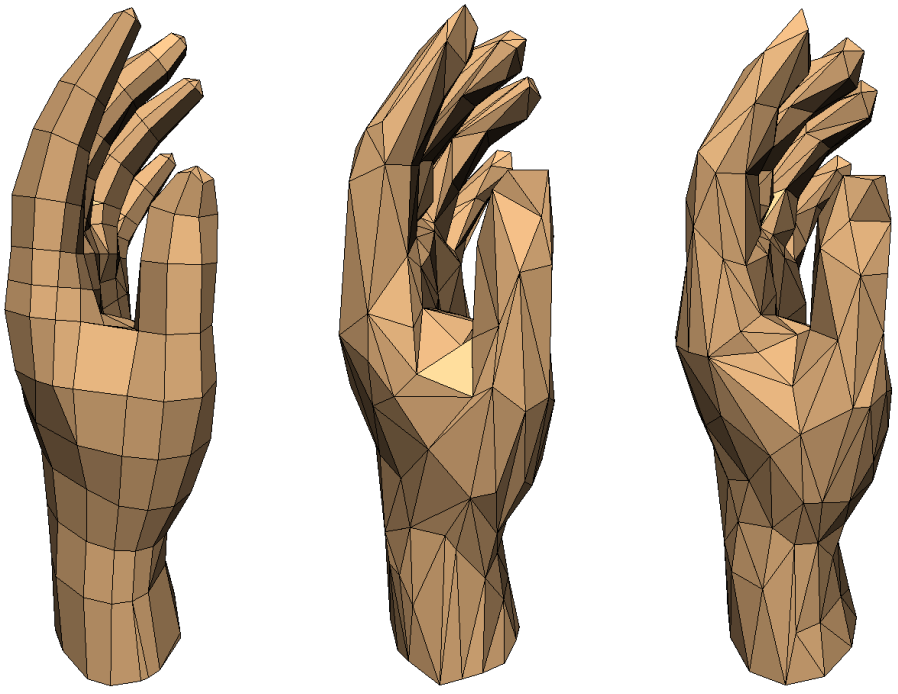


Fig. 19. Comparing approximation error of our remesher with the QSlim simplification system. Our system, which does not explicitly measure error, produces somewhat higher RMS error than the error-based QSlim method.



(a) Remeshed (b) QSlim (subset) (c) QSlim (optimal)

Fig. 20. Approximations of the hand dataset (Fig. 14) with 743 edges each. Although the RMS error is slightly higher, the mesh produced by our system is noticeably more appealing.

Figure 14 using our remesher, QSlim with subset placement, and QSlim with optimal placement. Subset placement is the most direct point of comparison as, like our remesher, it will place all output vertices on the input surface. For a given number of edges, our remeshings have roughly 20% higher error than subset placement and 50% higher error than optimal placement. However, if we look at the actual meshes, the output of our remesher is noticeably more appealing. The mesh is far more regular and shape distortions are more equally distributed. As we would expect, our system, which focuses exclusively on mesh quality and has no explicit notion of approximation error, produces nicer meshes but higher approximation error than a method such as QSlim, which focuses on approximation error and has no notion of mesh quality. For applications such as building base domains for subdivision surfaces, our results are clearly superior.

9 Discussion

9.1 Complexity Analysis

In Section 8 we have seen that our system is efficient in practice. It is also asymptotically efficient as well. If we are given an input surface of n vertices and produce an output of m vertices, the space complexity of our system is $O(n + m)$. To analyze its time complexity, we consider the field solution and remeshing processes separately.

Much like parameterization-based remeshing systems, scalar field construction is the most expensive part of our method, since it involves solving an $n \times n$ linear system. This can require at worst $O(n^3)$ time. However, we can expect to do much better as the system we are solving is quite sparse and has only $O(n)$ non-zero entries. For the SuperLU sparse solver that we use in our implementation, the best case running time would be $O(n^{3/2})$, corresponding to the situation in which all non-zero entries could be arranged in a single $\sqrt{n} \times \sqrt{n}$ dense supernode [Demmel et al., 1999]. We can improve the expected running time to $O(n)$ by using our multigrid solver [Ni et al., 2004]. However, we can not provide a linear worst case bound as that would require a provably good mesh hierarchy, and no provably good surface simplification algorithms are yet known.

The field solution must only be performed once for a given input and constraint set. It is thus independent of the output size. In contrast, the complexity of the remeshing phase depends on both the input and output size. In the worst case, where the input mesh is extremely dense, compared to the output mesh, the measurement of distance along the surface can require $O(\sqrt{n})$ time.

Such measurement could conceivably be needed $O(\sqrt{nm})$ times; thus, the worst case tracing time is $O(n\sqrt{m})$. However, this worst cast behavior only occurs in pathological cases and for essentially all reasonable inputs, distance computation times are $O(1)$ and the total complexity is $O(n + m)$. As can be seen from the running times reported in Table 1, this linear behavior is exactly what we encounter in practice.

9.2 Comparison with Alternative Methods

The majority of existing remeshing and simplification techniques focus on generating new triangulated meshes from their input. In contrast, our focus is on producing polygonal meshes consisting predominantly of quadrilaterals. The sole alternative method with similar goals is the anisotropic quad remeshing method recently proposed by Alliez *et al.* [2003a]. The meshes produced by their method are, at a high level, structurally similar to those produced by our own. Indeed, we have adapted some of their techniques to our setting, especially their approach of tracing integral lines through vector fields to determine polygon connectivity.

Despite these similarities, there are also some significant differences in our approach to the remeshing problem. First and foremost, our method requires neither a parameterization nor any cutting of the input model. Consequently, our method can easily handle models of arbitrary genus. In part because of our non-parametric approach, our technique appears to be somewhat faster. Alliez *et al.* report run times of about 60 seconds to remesh a model similar[4] to the hand model (Figure 14) which our method remeshes in about 20 seconds. It should be noted, however, that the Alliez *et al.* method has been recently extended to replace the global parameterization with local ones, allowing remeshing of closed high genus models [Marinov and Kobbelt, 2004].

Our use of harmonic fields also allows us to avoid some of the more delicate aspects of their approach, especially concerning the handling of umbilic points and integrating lines through a vector field with many critical points. And while their method always tries to lay out the new mesh conforming to the curvature field, we are able to provide a user with extensive control over the flow of the mesh (e.g., Figure 9).

[4] The model they use does not contain the forearm.

9.3 Mean Value Weights

One unfortunate property of the discrete harmonic weights (4) is that they may become negative in the presence of obtuse angles. This can result in local extrema at unconstrained vertices. While this does not in any way impair our remeshing process, it can be undesirable if too many additional extrema are created. This effect can be avoided by splitting all edges opposite internal angles larger than $\pi/2$. Alternatively, we can also avoid this problem by using the mean value weights [Floater, 2003] in place of the discrete harmonic weights:

$$w_{ij} = \frac{\lambda_{ij}}{\sum_{k \in N_i} \lambda_{ik}}, \qquad \lambda_{ij} = \frac{\tan(\theta_{ij}/2) + \tan(\phi_{ij}/2)}{\|\mathbf{x}_j - \mathbf{x}_i\|} \tag{15}$$

Here θ_{ij} and ϕ_{ij} are the two angles on either side of the edge (i, j) at vertex i. Using these weights, every edge (i, j) is guaranteed to be assigned a positive weight [Floater, 2003], and therefore local extrema cannot occur except at constrained vertices [Ni et al., 2004].

In practice, the choice of discrete harmonic versus mean value weights makes virtually no difference in the final remeshing results. However, this choice does impact the performance of the system. The mean value weights guarantee that no extraneous extrema will occur, and we have also found that they tend to improve the conditioning of the linear system. On the other hand, they are asymmetric. Thus twice as much memory is required to represent the linear system and certain methods that assume symmetric matrices (e.g., conjugate gradient) cannot be used. It is left to the user to decide whether they are willing to pay this increased cost.

9.4 Conformal Structure of the Field

By construction, the vector field $\mathbf{g}_1 = \nabla u$ is the gradient field of the scalar function u. It is natural to consider whether there is some scalar function v such that $\mathbf{g}_2 = \nabla v$. If we cut the surface M into a polygon [Gu and Yau, 2003, Ni et al., 2004, Erickson and Har-Peled, 2002] and require that v only be continuous at edge midpoints, then there does indeed exist a function $v : E \to \mathbb{R}$, unique up to an additive constant, for which $\mathbf{g}_2 = \nabla v$ [Polthier, 2000]. In this setting, we can rewrite the relationship governing $\mathbf{g}_1$ and $\mathbf{g}_2$ as:

$$\nabla v = \mathcal{R} \nabla u \tag{16}$$

Notice that this is simply one way of writing the well-known Cauchy–Riemann equations. This indicates that the mapping $(x, y, z) \to (u, v)$ is in fact a discrete conformal mapping of the cut manifold into the plane. The mapping is,

however, non-conforming as the field v is not necessarily continuous at the vertices of the mesh.

We can also see the vector fields that we compute as an instance of the holomorphic 1-forms used by Gu & Yau [2003] and Jin *et al.* [2004]. The scalar field u is harmonic everywhere except at the k constrained extremal points. Thus the vector fields $\mathbf{g}_1$ and $\mathbf{g}_2$, respectively, define a harmonic 1-form ω_{ij} and its conjugate 1-form $*\omega_{ij}$:

$$\omega_{ij} = (\mathbf{x}_j - \mathbf{x}_i)^\mathsf{T} \mathbf{g}_1 \tag{17}$$

$$*\omega_{ij} = (\mathbf{x}_j - \mathbf{x}_i)^\mathsf{T} \mathbf{g}_2 \tag{18}$$

Together, these 1-forms define a holomorphic 1-form over M with k points removed. Note that this is not *entirely* equivalent to the formulation proposed by Gu and Yau. As discussed above, our 1-form is non-conforming, whereas the 1-forms used by Gu and Yau are fully conforming.

10 Conclusion

We have presented a novel algorithm for quad-dominant mesh generation. Our method is both efficient and general. It does not require any parameterization of the surface and can process manifolds of any genus. The resulting meshes resample the geometry quite effectively. The user is provided control over the flow of the quadrilaterals, via scalar field constraints, and over the local quadrilateral size, via a spacing function. The final mesh interpolates the original geometry, in that all of its vertices lie on the original surface. All of this is based on our usage of a harmonic scalar field to define the flow of the mesh over the surface.

The technique we have described works very well. There are also a number of interesting avenues for further work that could extend or improve its current performance. As with many other remeshing techniques, we place sample vertices exactly on the piecewise linear surface of the original mesh. Allowing the system more freedom in placing vertices near but not on the surface could provide a better fit to the original shape. There are potentially many methods for automatically identifying extremal points. The *average geodesic distance* used for capturing "isolated" points [Hilaga et al., 2001, Zhang et al., 2003] is one particularly attractive alternative. Identifying extremal points using the eigenvectors of the Laplacian matrix might also prove effective. Finally, we have outlined (§9.4) the connection between our vector field construction and a global conformal structure [Gu and Yau, 2003] over the surface. We believe that this connection can be exploited for efficient compatible remeshing of multiple manifolds.

11 Acknowledgements

Thanks to the many groups and people who provided the surface meshes used in this work. The original scan of the happy buddha statue is due to the Stanford Graphics Lab; we have used a topologically clean version provided by Zoë Wood. The octaflower model was made by Yutaka Ohtake of the Max-Planck-Institut für Informatik. The knot is available from *The KnotPlot Site* maintained by Robert Scharein at the University of British Columbia. Paul Heckbert designed the blobby "V". The Igea head and the Isis statue are courtesy of Cyberware.

This research was funded in part by a grant from the National Science Foundation (CCR-0098170).

References

P. Alliez, D. Cohen-Steiner, O. Devillers, B. Lévy, and M. Desbrun. Anisotropic polygonal remeshing. *ACM Transactions on Graphics*, 22(3): 485–493, July 2003a.

P. Alliez, É. C. de Verdière, O. Devillers, and M. Isenburg. Isotropic surface remeshing. In *Proceedings of Shape Modeling International*, pages 49–58, 2003b.

P. Alliez, M. Meyer, and M. Desbrun. Interactive geometry remeshing. *ACM Transactions on Graphics*, 21(3):347–354, July 2002. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).

M. W. Bern and D. Eppstein. Mesh generation and optimal triangulation. In D.-Z. Du and F. K. Hwang, editors, *Computing in Euclidean Geometry*, number 4 in Lecture Notes Series on Computing, pages 47–123. World Scientific, second edition, 1995.

D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Transactions on Graphics*, 23(3), July 2004. Proceedings of SIGGRAPH 2004.

D. Cohen-Steiner and M. Desbrun. Hindsight: LSCM=DNCP. web: http://www-grail.usc.edu/pubs/CD02.pdf, June 2002.

J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 20(3):720–755, 1999.

M. Desbrun, M. Meyer, and P. Alliez. Intrinsic parameterizations of surface meshes. In *Eurographics 2002 Conference Proceedings*, 2002.

M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 317–324, Aug. 1999.

T. Duchamp, A. Certain, A. DeRose, and W. Stuetzle. Hierarchical computation of PL harmonic embeddings. preprint, July 1997.

H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.

J. Erickson and S. Har-Peled. Optimally cutting a surface into a disk. In *Proceedings of the 18th Annual ACM Symposium on Computational Geometry*, pages 244–253, 2002.

M. S. Floater. Parameterization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

M. S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19–27, Mar. 2003.

M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Multiresolution in Geometric Modelling*. Springer, 2004.

M. Garland. Multiresolution modeling: Survey & future opportunities. In *State of the Art Report*, pages 111–131. Eurographics, Sept. 1999.

M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH 97*, pages 209–216. ACM SIGGRAPH, Aug. 1997.

X. Gu and S.-T. Yau. Global conformal surface parameterization. In *Proc. Eurographics Symposium on Geometry Processing*, 2003.

L. Guibas and M. Sharir. Combinatorics and algorithms of arrangements. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, pages 9–36. Springer-Verlag, 1993.

M. Hilaga, Y. Shinagawa, T. Kohmura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 203–212, Aug. 2001.

H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH 96*, pages 99–108. ACM SIGGRAPH, Aug. 1996.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *SIGGRAPH '93 Proc.*, pages 19–26, Aug. 1993.

M. Jin, Y. Wang, S.-T. Yau, and X. Gu. Optimal global conformal surface parameterization. In *Proc. IEEE Visualization*, pages 267–274, 2004.

B. Jobard and W. Lefer. Creating evenly-spaced streamlines of arbitrary density. In P. Brunet and R. Scopigno, editors, *Proceedings of the Eurographics Workshop on Visualization in Scientific Computing*, pages 45–55. The Eurographics Association and Blackwell Publishers, 1997.

A. D. Kalvin and R. H. Taylor. Superfaces: Polygonal mesh simplification with bounded error. *IEEE Computer Graphics and Appl.*, 16(3), May 1996.

L. P. Kobbelt, J. Vorsatz, U. Labsik, and H.-P. Seidel. A shrink wrapping approach to remeshing polygonal surface. In P. Brunet and R. Scopigno, editors, *Computer Graphics Forum (Eurographics '99)*, volume 18(3), pages 119–130. The Eurographics Association and Blackwell Publishers, 1999.

B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps

for automatic texture atlas generation. *ACM Transactions on Graphics*, 21 (3):362–371, July 2002. (Proceedings of ACM SIGGRAPH 2002).

P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *Proceedings of IEEE Visualization 98*, pages 279–286,544, Oct 1998.

M. Marinov and L. Kobbelt. Direct anisotropic quad-dominant remeshing. In *Proc. Pacific Graphics*, 2004.

M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-man ifolds. In H.-C. Hege and K. Polthier, editors, *Visualization and Mathematics III*, pages 35–57. Springer-Verlag, Heidelberg, 2003.

X. Ni, M. Garland, and J. C. Hart. Fair Morse functions for extracting the topological structure of a surface mesh. *ACM Transactions on Graphics*, 23 (3):613–622, July 2004. Proceedings of SIGGRAPH 2004.

S. J. Owen. A survey of unstructured mesh generation technology. In *Proceedings 7th International Meshing Roundtable*, Oct. 1998.

D. L. Page, Y. Sun, A. F. Koschan, J. Paik, and M. A. Abidi. Normal vector voting: crease detection and curvature estimation on large, noisy meshes. *Graph. Models*, 64(3/4):199–229, 2002. ISSN 1524-0703.

U. Pinkall and K. Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

K. Polthier. Conjugate harmonic maps and minimal surfaces. Preprint No. 446, TU-Berlin, 2000. URL `http://www.zib.de/polthier/articles/harmonic/Harmonic_abstract.html`.

E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. *Proc. SIGGRAPH 2000*, pages 465–470, July 2000.

J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering complex scenes. In B. Falcidieno and T. Kunii, editors, *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, 1993.

W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):65–70, July 1992.

K. Shimada, J.-H. Liao, and T. Itoh. Quadrilateral meshing with directionality control through the packing of square cells. In *Seventh Intl. Meshing Roundtable*, pages 61–75, Oct. 1998.

O. Sifri, A. Sheffer, and C. Gotsman. Geodesic-based surface remeshing. In *12th Intl. Meshing Roundtable*, Sept. 2003.

D. Steiner and A. Fischer. Topology recognition of 3d closed freeform objects based on topological graphs. *Proc. Pacific Graphics*, pages 82–88, Oct. 2001.

D. Steiner and A. Fischer. Finding and defining the generators of genus-n objects for constructing topological and cut graphs. *The Visual Computer*, 20(4):266–278, June 2004.

V. Surazhsky, P. Alliez, and C. Gotsman. Isotropic remeshing of surfaces: A local parameterization approach. In *Proc. 12th International Meshing Roundtable*, Sept. 2003.

V. Surazhsky and C. Gotsman. Explicit surface remeshing. In *Proc. Euro-*

graphics Symposium on Geometry Processing, pages 20–30, 2003.

G. Taubin. A signal processing approach to fair surface design. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 351–358, Aug. 1995.

G. Taubin. Geometric signal processing on polygonal meshes. In *State of the Art Report*, pages 81–96. Eurographics, Sept. 2000.

G. Turk. Re-tiling polygonal surfaces. *Computer Graphics (SIGGRAPH '92 Proc.)*, 26(2):55–64, July 1992.

G. Turk. Texture synthesis on surfaces. In *Proceedings SIGGRAPH 2001*, Aug. 2001.

K. Watanabe and A. G. Belyaev. Detection of salient curvature features on polygonal surfaces. *Computer Graphics Forum (Procs. EUROGRAPHICS '01)*, 20(3):385–392, 2001.

L.-Y. Wei and M. Levoy. Texture synthesis over arbitrary manifold surfaces. In *Proceedings SIGGRAPH 2001*, Aug. 2001.

D. R. White and P. Kinney. Redesign of the paving algorithm: Robustness enhancements through element by element meshing. In *Proceedings 6th International Meshing Roundtable*, pages 323–335, Oct. 1997.

S. Zelinka and M. Garland. Interactive texture synthesis on surfaces using jump maps. In *Proceedings of the Eurographics Symposium on Rendering 2003*, pages 90–96. Eurographics Association, June 2003.

E. Zhang, K. Mischaikow, and G. Turk. Feature-based surface parameterization and texture mapping. Technical Report GVU 03-29, Georgia Tech., 2003.

E. Zhang, K. Mischaikow, and G. Turk. Vector field design on surfaces. Technical Report 04-16, Georgia Institute of Technology, 2004. URL `http://www.cc.gatech.edu/grads/z/Eugene.Zhang/vecfld_design.html`.