Direct (Re)Meshing for Efficient Surface Processing

John Schreiner, Carlos E. Scheidegger, Shachar Fleishman and Cláudio T. Silva

UUSCI-2006-013

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

March 27, 2006

Abstract:

Many geometry processing techniques exploit the connection between shape and connectivity in triangle meshes to enable advanced processing of the data. These techniques face the challenge that triangle meshes, whether generated by a modeling tool or by 3D scanning, typically contain low quality triangles. The process of constructing *good* meshes out of sub-optimal ones is termed *surface remeshing*. Some form of surface remeshing is often applied as a pre- or post-processing step in many geometry processing algorithms, even if only implicitly. We propose a novel surface remeshing algorithm. While many remeshing algorithms are based on global parametrization or local mesh optimization, our algorithm is closely related to surface reconstruction techniques and it requires no explicit parameterization. Our approach is based on the advancing-front paradigm, and it can be used to both incrementally remesh the complete surface, or simply to remesh a portion of it with a high-quality mesh. It is accurate, fast, robust, and suitable for use with interactive mesh processing applications that require local remeshing. We show a number of applications, including matching the resolution of meshes when doing boolean operations such as unions and intersections. We also show how to adapt the algorithm to blend and merge mixed-mode objects for example, to compute the union of a point-set surface and a triangle mesh.

Direct (Re)Meshing for Efficient Surface Processing

John Schreiner Carlos E. Scheidegger Shachar Fleishman Cláudio T. Silva

Abstract

Many geometry processing techniques exploit the connection between shape and connectivity in triangle meshes to enable advanced processing of the data. These techniques face the challenge that triangle meshes, whether generated by a modeling tool or by 3D scanning, typically contain low quality triangles. The process of constructing *good* meshes out of sub-optimal ones is termed *surface remeshing*. Some form of surface remeshing is often applied as a pre- or post-processing step in many geometry processing algorithms, even if only implicitly. We propose a novel surface remeshing algorithm. While many remeshing algorithms are based on global parametrization or local mesh optimization, our algorithm is closely related to surface reconstruction techniques and it requires no explicit parameterization. Our approach is based on the advancing-front paradigm, and it can be used to both incrementally remesh the complete surface, or simply to remesh a portion of it with a high-quality mesh. It is accurate, fast, robust, and suitable for use with interactive mesh processing applications that require local remeshing. We show a number of applications, including matching the resolution of meshes when doing boolean operations such as unions and intersections. We also show how to adapt the algorithm to blend and merge mixed-mode objects — for example, to compute the union of a point-set surface and a triangle mesh.

1 Introduction

Triangle meshes are ubiquitous in digital geometry processing, and as such, triangulated models are many times only intermediate steps in more complex systems [9, 13, 14, 17]. Often, the efficiency and robustness of these geometric operations depend directly on mesh quality. Unfortunately, many geometric algorithms tend to generate low-quality triangulations to start with, or to lower triangulation quality after each application. This need for high-quality meshes, and in particular for improving mesh quality, has driven much of the development of remeshing algorithms [3].

Certain operations require meshes with roughly the same triangle size and reasonable triangle shape to work well. Mesh-editing systems are the typical example: boolean operations, such as union and intersection between meshes work much better when the triangle sizes between the meshes are similar. This is particularly true for the recently developed advanced mesh-editing techniques based on Laplacian coordinates [26] or on the Poisson equation [29]. These techniques allow automatic smooth stitching of two object parts. For these techniques to work, it is necessary to perform a remeshing step that matches the resolution and produces a single high-quality mesh before the computations. This problem gained little attention in the past and is so far solved by combining shapes of similar triangle size and manual work [29].

Many of these geometric applications are not well served by current remeshing techniques. Some approaches require parameterization of the surface to a planar region, which limits the techniques to surfaces homeomorphic to disks. If more general surfaces are used, cutting and stitching is necessary, and the quality of the meshing tends to suffer near the seams of the parameterization. Also, performing global parametrization of very large objects can be challenging. Techniques based on local mesh-improving operations (e.g., edge splits, edge collapses, and vertex repositioning) tend to be more practical, but need to be performed with care not to modify the overall geometric shape of the model. See [3] for an excellent survey.

We introduce a new remeshing algorithm that is based on a surface-reconstruction approach. Our technique builds a high-quality triangulation by directly resampling the geometry and topology of the input geometry, as though it was performing surface reconstruction. Our remeshing algorithm can be applied locally to a region of interest as in the case of editing operations between two meshes. Also, we can remesh the entire input to obtain a mesh that is optimally sampled in terms of triangle quality and Hausdorff distance between an input surface and the remesh and with an intuitive user control over the allowed error. Because our technique is based on surface reconstruction, it is not limited to triangle meshes. As Figure 1 shows, it also allows for mixed-mode operations, e.g., the union of a triangle mesh with a point-set surface.

Our work is based on the technique of Scheidegger et al. [24]. The main idea of their advancing-front algorithm is to grow a triangulation over a point-set surface using a guidance field that dictates the appropriate triangle size. They use a finite set of samples from the surface curvature that, when queried appropriately, tends to overcome the excessive locality of decision-making by advancing-front methods. We extend their approach in significant ways. First, we have a more principled way to compute the guidance field, which can be used to prove that the induced function is Lipschitz continuous. As a result, we generate fewer and better triangles. The algorithm is also significantly more robust. Additionally, we show that the algorithm works on any surface for which we can compute curvature and project points on the surface. For example, this includes a variety of point set surface definitions. This allows us to generate a mesh that is the result of some operation between different types of surface representations, or allowing users to perform mesh operations which results in a triangle soup, i.e. a mesh that may contain triangle flips, overlaps, holes, etc. then one can sample the affected region with some points and simply remesh that region with our algorithm. We exploit the local nature of the code to handle sharp features in input meshing.

A key contribution of this work is a simple solution to a problem that is encountered by most implementations of mesh processing algorithms: the requirement of high quality triangulations and control of their resolutions. Our method produces meshes that balance three requirements well: high geometric fidelity, low triangle count and good triangle shape. Our remeshing algorithm requires no parameterization, is indifferent to the topology of the objects or their intersection curve and is simple.

2 Related work

The need to generate discrete representations of continuous geometry for computational, imaging and other purposes is widespread, and has generated substantial amount of literature in surface polygonization, meshing, and, more recently, remeshing. For implicit surfaces, seminal work is reviewed in [6]. A good example of this early work is the Marching Cubes algorithm [19, 22], which samples functions at a grid of fixed resolution, and uses a table of possible configurations of range signs to create a triangulated surface out of those samples. The main strengths of Marching Cubes are its generality, simplicity and robustness, which have made it one of the most used meshing algorithms in practice (e.g., 3D photography [18]). The main problem with Marching Cubes is the inherent

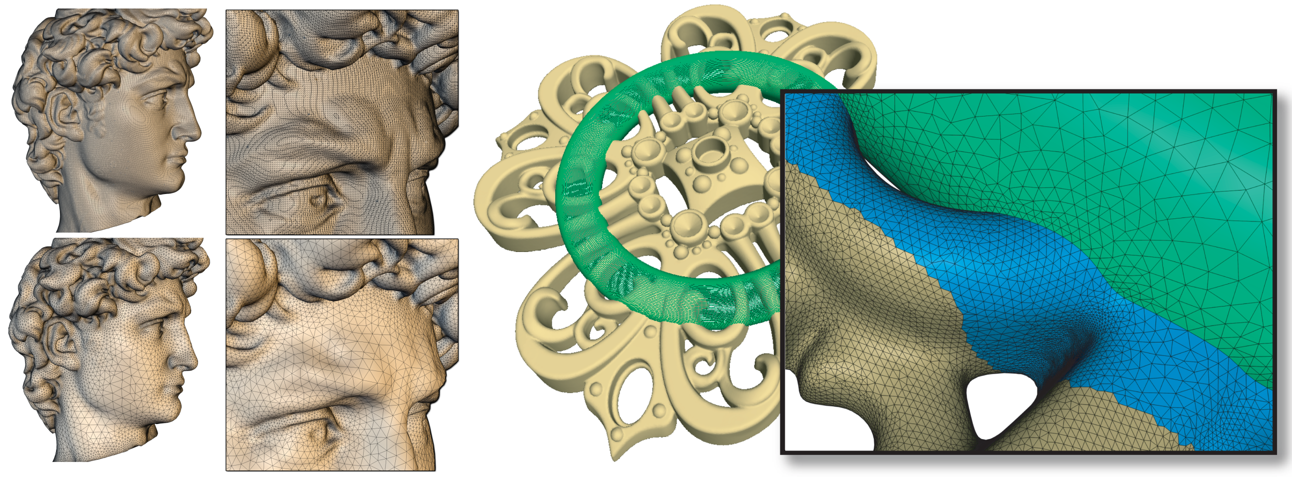


Figure 1: Many surface processing tasks require *good* meshes. At the same time, many meshes created automatically exhibit bad triangulations. Our remeshing algorithm is based on surface reconstruction and requires no parameterization. It generates graded triangle meshes of excellent quality, and can also be used for high-quality CSG operations. Because the technique is very general, we also use it to create *mixed-mode* models: CSG operations between a mesh and a point-set, for example. In the left, we show a remesh of a portion of the Michelangelo David. On the right, we compute the union of a triangle mesh with genus 48 and a torus defined by a point set, with a final genus of 68. For CSG operations, we do not remesh the entire surface: only a part of the original mesh, shown in blue, is retriangulated. Note that triangles meet one another in the intersection curves with the same resolution, yielding high triangle quality.

bias caused by placing vertices on all intersections between grid edges and the surface. This also implies that the sampling density is proportional to the grid resolution, and not to any intrinsic surface properties. Meshes from Marching Cubes are typically over-tessellated, and contain many bad triangles.

The growing field of mesh processing has developed a number of advanced surface processing techniques, e.g., mesh editing [25,29], mesh deformation [27], cloth simulation and compression. For these, and other applications, good triangle meshes are required since often the mesh is assumed to be a piecewise approximation of a smooth function. As explained in the excellent survey of Alliez et al. [3], there is a need to improve *raw* meshes with oversampled and redundant geometry. This has given rise to the sub-field of *remeshing*. Although no formal definition of remeshing exists, it is roughly the process of creating a mesh with certain *improved* properties from a pre-existing mesh. Typically, the goal is to optimize sampling, grading, regularity, size, and shape of elements, while keeping the overall geometry of the model the same.

A number of remeshing techniques work by first computing a parametrization of the input geometry. Then, the surface is resampled in the parameter domain, and the samples are reprojected into 3D space. Unless the surface is homeomorphic to a disk, this requires cutting and stitching of the surface. Surazhsky and Gotsman [28] point out that such remeshing techniques are sensitive to the specific global parameterization used and can be slow. Not only that, but robust implementation of these techniques is non-trivial, since numerical precision issues often arise from the distortion induced by the parameterizations. Instead, they advocate iteratively applying local optimizations directly on the existing mesh until some quality criteria are fulfilled. We point the reader to [3] for coverage of existing remeshing techniques.

Our technique is most related to recent developments in surface reconstruction, triangulation, and remeshing [2,7,24]. Given a two-dimensional orientable manifold surface $\mathcal{S}$ embedded in $\mathbb{R}^3$ that supports a projection operator and curvature computation, we produce a high quality triangulation that accurately captures details. Our final reconstruction has bounded error from the original mesh, and most of the triangles exhibit excellent quality – the user defines how close to equilateral they must be. In particular, our work is inspired by the advancing-front algorithm of Scheidegger et al. [24], who extended the technique of Karkanis and Stewart [16] to the triangulation of point-set surfaces. Their key innovation was the use of a global guidance field (also called a sizing field in other works [4, 7]) to avoid missing features of the underlying surface. Section 5 contains a detailed comparison to their work.

3 The Advancing Front Algorithm

We state the basic remeshing problem as follows. Given a surface $\mathcal{S}$ as input, defined as a projection operator $\mathcal{P} : \mathbb{R}^3 \to \mathcal{S}$, we want to construct a triangulation $\mathcal{T}$ such that the distance between $\mathcal{T}$ and $\mathcal{S}$ is bounded. We also want to control the number of triangles used, and the quality of the generated triangles. There are two user-defined parameters ρ and η which will control approximation accuracy and triangle quality, respectively. Both parameters affect triangle count, as can be seen in Figure 9. For each surface, we define $\iota_{\mathcal{S}} : \mathcal{S} \to \mathbb{R}^+$ to be a function that gives the local ideal edge size for a triangle of $\mathcal{T}$ on $\mathcal{S}$. As illustrated on Figure 4, we define $\iota_{\mathcal{S}}(x)$ to be the edge that subtends an angle ρ on the osculating circle of minimum radius situated at x:

$$\iota_{\mathcal{S}}(x) = \frac{2\sin(\rho/2)}{\kappa_{\max}}$$

where $\kappa_{\max}$ is the maximum curvature at x. One critical problem for triangulation quality is that of grading: good triangle meshes are possible when triangle sizes change slowly throughout the surface [23]. We solve this problem by using a *guidance field* $g_{\mathcal{S}}(x) : \mathbb{R}^3 \to \mathbb{R}^+$ which will determine the triangle size. The guidance field will be constructed so that triangle grading in $\mathcal{T}$ is adequate, and triangles are no larger than $\iota_{\mathcal{S}}$ mandates.

The algorithm begins with a set of initial fronts, which are lists of vertices with normals. These initial fronts could be resamplings of the intersection curves computed from a *constructive solid geometry* (CSG) operation (see Figure 8), a boundary loop of a patch of the surface to be locally remeshed, or simply a single seed edge to begin remeshing the entire surface. The fronts are iteratively modified with *edge growth* and *connection* operations until all fronts have been closed. New triangles are always created from an edge

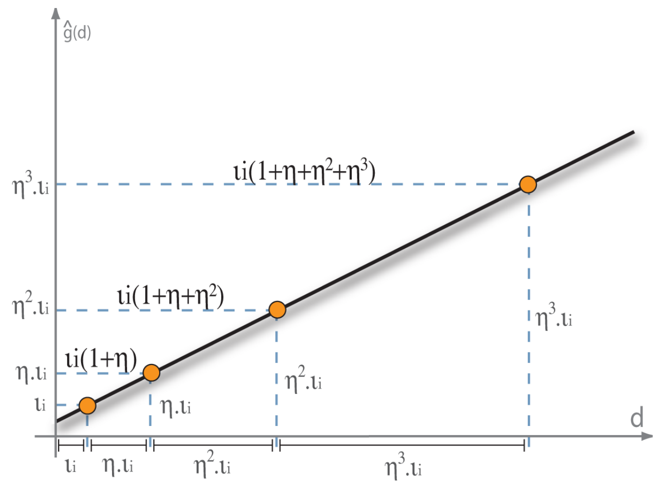


Figure 2: Illustration of $\hat{g}_i(x)$, the function that defines the correct edge size as a function of the distance to a sample $\tilde{s}_i$ of the surface.

in a front. *Free* triangles are the ones where the remaining vertex is a newly chosen sample of $\mathcal{S}$. If this free triangle were to encroach on an existing front, we instead pick an existing vertex of $\mathcal{T}$, and call it a *connection* triangle. If the edge and the connection vertex are part of the same front, the front gets split into two. Otherwise, the two different fronts are merged into one. A front is closed when it only contains three vertices, which are used in a single triangle. As the front advances, every free triangle is placed before any connection triangles. Within each class of triangles, we prioritize the ones which would result in larger ratios of the incircle radius to the circumcircle radius. Note that there is a correspondence between these operations and handlebody decompositions [20]. Therefore, advancing front techniques naturally cope with high-genus manifold reconstruction.

At the heart of our algorithm is the guidance field $g_{\mathcal{S}}(x)$. The guidance field determines the choice of triangle size on the surface. It prevents large triangles from being created near small ones by "looking ahead" and gradually shrinking edges before getting to a detailed area. Limiting the rate of edge length change also bounds the aspect ratio of free triangles. When placing a free triangle, we query the guidance field at each of the existing edge vertex locations. These values, combined with the normals and ordering of the two front vertices, determine a tentative location for the new vertex. This vertex is then projected onto the surface and inserted into the front.

3.1 Constructing the guidance field

Here, we show how to construct the guidance field $g_{\mathcal{S}}(x)$ from a finite set of samples $\tilde{s}_i$ taken from $\mathcal{S}$, each associated with an ideal length ι_i. We will discuss only the construction of the guidance field for meshing a single surface, but will explain in later sections how to extend the definition for more advanced operations.

We assume the surface we want to remesh is smooth. Then, a Taylor expansion of the surface around a point shows that up to second order the surface is locally characterized by the shape operator [10]. From the operator we compute $\kappa_{\max}$. In other words, we do not take anisotropy into account, and approximate the surface assuming it is locally spherical. We impose two conditions on the size of triangles:

1. The triangles placed over a patch of surface must be a good approximation for the surface. In other words, the triangle edge size at s_i should be at most ι_i.

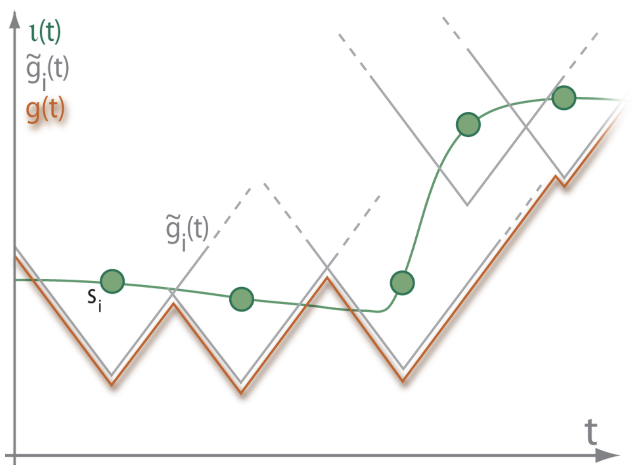


Figure 3: The guidance field $g(t)$ on a curve $t : \mathbb{R} \to \mathcal{S}$. $g(t)$ is the minimum over all $\tilde{g}$. At the sample points $\tilde{s}_i$, $\tilde{g}_i$ is minimum, and it grows linearly as the distance from $\tilde{s}_i$ increases. Note that if the sampling is too coarse, $g(t)$ might not bound $\iota(t)$, and that some of the samples might be unnecessary. Since each $\tilde{g}_i$ is Lipschitz, so is $g(t)$.

2. Triangle quality throughout the triangulation must be adequate. Specifically, we require that any two edges e_i and e_j incident to a common vertex have a ratio bounded by a user-defined parameter η:

$$\eta^{-1} \leq |e_i|/|e_j| \leq \eta \tag{1}$$

Given a set of surface samples $\tilde{s}_i \in \mathcal{S}$, we compute $\kappa_i = \kappa_{\max}$ applied at $\tilde{s}_i$, calling $\iota_i = 2\sin(\rho/2)/\kappa_i$ (see Figure 4). Each $\tilde{s}_i$ defines a constraint on $g_{\mathcal{S}}(x)$, namely $g_{\mathcal{S}}(x) \leq \tilde{g}_i(x)$, and the guidance field will be, at each point, the maximum value that satisfies all constraints.

We will construct $\tilde{g}_i(x)$ by making use of a simpler function. $\hat{g}_i : \mathbb{R}^+ \to \mathbb{R}^+$ will define the constraint on edge size as a function of the distance to $\tilde{s}_i$: $\hat{g}_i(|\tilde{s}_i - x|) = \tilde{g}_i(x)$. To use as few triangles as possible, $\hat{g}_i(x)$ should be as large as possible. Furthermore, $\hat{g}_i$ should be monotonically increasing so that triangle edge sizes always increase as the edges move away from $\tilde{s}_i$. To satisfy condition 1, we need

$$\hat{g}_i(\iota_i) = \iota_i. \tag{2}$$

In other words, edges close to the sample must be sufficiently small. While Condition 1 restricts the correct size for the closest edge to the sample, Condition 2 will restrict the remaining edge sizes. Maximizing $\hat{g}_i(x)$ along with equation (2) directly implies

$$\begin{aligned} \hat{g}_i(\iota_i(1+\eta)) &= \eta\iota_i \\ \hat{g}_i(\iota_i(1+\eta+\eta^2)) &= \eta^2\iota_i \\ &\vdots \\ \hat{g}_i\left(\iota_i\frac{1-\eta^k}{1-\eta}\right) &= \eta^{k-1}\iota_i, k>0 \end{aligned} \tag{3}$$

Equation 3 defines $\hat{g}_i(x)$ at a set of discrete values. Since we need to evaluate it at any distance $d \in \mathbb{R}^+$, we must extend its definition appropriately. There are many such functions. Since $\hat{g}_i$ will directly determine edge sizes, we look for one that minimizes

$$\int_0^\infty \left(\hat{g}_i''(x)\right)^2 dx \tag{4}$$

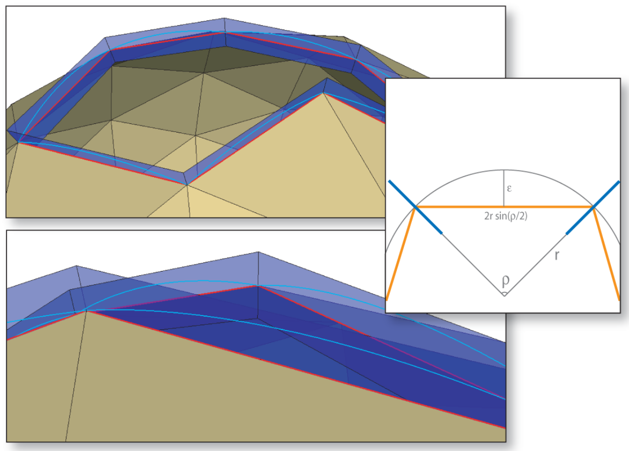


Figure 4: To robustly detect front interference, we use a set of *fences*: extensions of the front curve in the normal direction of the surface. We exploit the bound on the Hausdorff error to determine the correct fence height – the inset on the right shows the argument in two dimensions. $r = \kappa_{\max}^{-1}$.

This will minimize the change in the grading determined by the function. Consider the following expression for $\hat{g}_i(x)$:

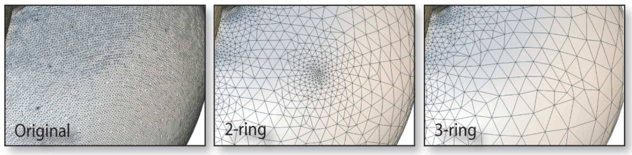
$$\hat{g}_i(x) = (1 - \eta^{-1})x + \eta^{-1}\iota_i \tag{5}$$

This expression interpolates all values given by Equation 3. Since its second derivative is zero everywhere in the open interval $(0, \infty)$, the integral is zero, and so it is the global minimizer of Equation 4. Figure 2 illustrates the situation. Finally, we simply say that $g_{\mathcal{S}}(x) = \min_i \tilde{g}_i(x)$. Each $\hat{g}_i(x)$ is clearly Lipschitz, and so are the $\tilde{g}_i(x)$. Since the minimum of a set of Lipschitz functions is Lipschitz, a $g_{\mathcal{S}}(x)$ constructed in this way will be Lipschitz, regardless of the ι_i or the sampling density. Notice that the Lipschitz order is directly related to the allowed rate of change of triangle edges. Figure 3 illustrates a plot of an example guidance field.

3.2 Surfaces

Our algorithm only requires a few properties of $\mathcal{S}$. $\mathcal{S}$ must admit only a projection operator $\mathcal{P} : \mathbb{R}^3 \to \mathcal{S}$, and the computation of curvature at any given point $\tilde{s} \in \mathcal{S}$. The curvature is used for computing ι, and $\mathcal{P}$ projects the new points of free triangles onto the surface. We now describe how to compute these for a few surface types we have implemented.

Triangle Meshes Since triangle meshes are ubiquitous in all areas of computer graphics, they are an obvious target for our triangulator. We take the surface samples $\tilde{s}_i$ to be the mesh vertices. We estimate the curvature at a point by fitting a quadratic polynomial to a neighborhood of the mesh [12], even though other estimations are possible [21]. We have found that using the 3-ring neighborhood of a vertex to compute the curvature works well for all the meshes we have experimented with. Using a smaller neighborhood will typically lead to erroneous curvature estimations as most meshes have poorly represented areas.



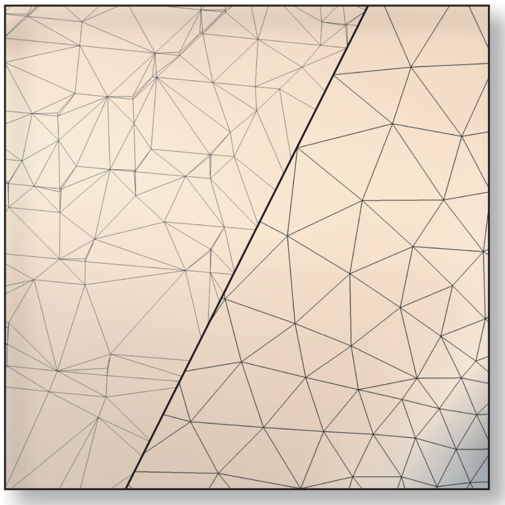
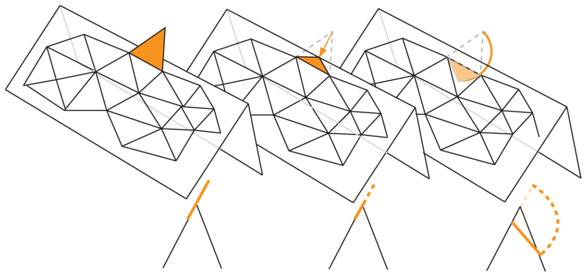


Figure 5: A closeup of the original Buddha mesh and our result. Our remesher is independent of the quality of the input, and outputs well-shaped triangles that respect approximation constraints.

One possible definition of $\mathcal{P}$ for mesh surfaces is simply a nearest point projection. Traditional advancing front techniques use a projection procedure where a *tentative point* in created, usually through some sort of prediction, and then projected on the surface. This procedure tends to cluster point samples on non-smooth parts of the surface, leading to unexpected changes in triangle edge lengths, and triangulation artifacts. Instead, we use a procedure which guarantees that the edge lengths of the new triangle are exactly the desired length. Consider creating a free triangle from a front edge with vertices $\mathbf{v}_1$ and $\mathbf{v}_2$ and new vertex $\mathbf{x}$. We want $||\mathbf{x} - \mathbf{v}_1|| = g_{\mathcal{S}}(\mathbf{v}_1)$ and $||\mathbf{x} - \mathbf{v}_2|| = g_{\mathcal{S}}(\mathbf{v}_2)$. Each of these equations defines a sphere. We find the intersections between these spheres and the mesh, and define the projection to be the point for which the new triangle normal is closest to the front normals. The following figure illustrates, respectively, the original triangle, the result of a closest point projection, and the solution we use.



MLS Surfaces Moving-Least Squares surfaces are a popular way to define a smooth surface from a finite set of samples [1, 2, 5, 11]. Most of them work precisely in terms of a projection operator, the fixed points of which are taken to be the resulting surface. These definitions are in general suitable for our algorithm. The original MLS surface [2] is defined in terms of a non-linear optimization, and so it is non-trivial to compute any differential properties. We then estimate normals and curvatures as described in [24].

Linear MLS Surfaces We have also implemented a linear formulation of MLS surfaces, which take points and oriented normals as input [1]. We exploit here the formulation as an implicit surface

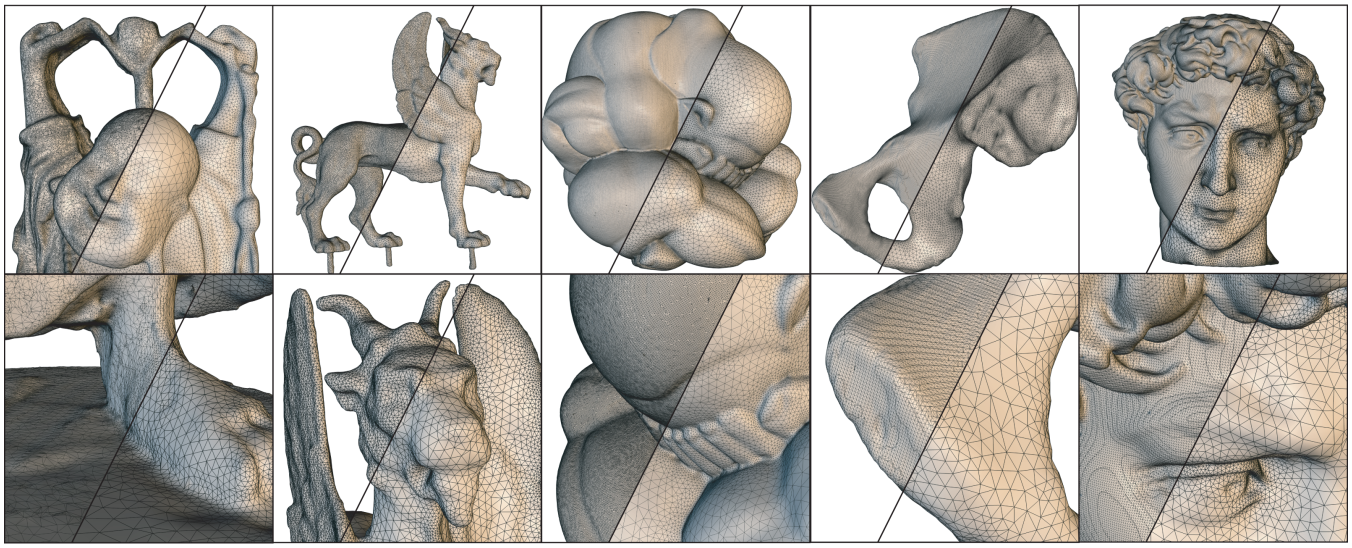


Figure 6: A sample of the results of our algorithm. From left to right we show remeshes of the Happy Buddha, the Feline, Pensatore, a Marching Cubes reconstruction of a pelvis bone used for a biomechanics simulation, and the head of Michelangelo's David.

to compute differential properties exactly. Implementing the computation of the Hessian of the projected point explicitly would be very tedious and error-prone. Instead, we use C++ metaprogramming to *automatically* compute the expressions. We use a class that essentially encodes the behavior of the chain rule in calculus. Additionally, since all the expansion is done at compile-time, the compiler can optimize redundant expressions. We have measured the performance to be similar to the previously described methods used in meshes and non-linear MLS.

3.3 Feature Preservation

Our advancing front algorithm allows for a simple extension to enable meshing of surfaces while preserving a certain class of surface features. We are able to preserve boundaries, creases, and intersection curves from CSG operations. In order to preserve them, we must ensure that no triangles are created which cross them. This is easiest done by initializing the advancing fronts *at* the features, essentially dealing with each piecewise smooth patch separately. Finding boundaries and creases is a hard problem in general. For mesh surfaces, finding boundaries is trivial. To identify creases, we simply threshold the dihedral angles, though advanced techniques could be used for identifying ridges and valleys in meshes [15], and point-set surfaces [11]. Once the feature curves have been found, we resample them with an algorithm analogous to our advancing front, albeit in one lower dimension. These resampled curves then become the initial fronts for the triangulation.

When constructing the guidance field for surfaces with features, we include the curvature of the feature curves in addition to the curvature of the surface. We then use the same guidance field for both resampling the feature curves and triangulating the surface. This ensures that all of the features are accurately represented while still maintaining the constraints on the triangle quality.

3.4 Implementation

Our system was designed to mesh (or remesh) a large class of surfaces, and the design of our prototype implementation reflects this. The main triangulation module handles the front advancement, with all intersection tests, merging and splitting of fronts. A set of abstract classes define the interface of the surface definition, and so the triangulation module is unaware of the underlying surface type. This simplifies the implementation of more advanced operations, such as mixed-mode remeshing.

A challenging aspect of implementing an advancing front algorithm is to determine when a free triangles encroaches a front in the triangulation. The fronts are only linear approximations of curves on the surface, so they can cross each other without strictly intersecting. If this happens, more than one triangle will cover a patch of the surface, making it topologically incorrect. Our solution is to extend the front curve along the normal surface direction, creating what we call *fences*, illustrated in Figure 4. To determine if a free triangle is encroaching, we test it for intersection against the fences. Since Equation 8 tells us the maximum distance between a point on $\mathcal{S}$ and the mesh, the fence heights simply need to be greater than $\varepsilon(\rho, \mathbf{x})$ to guarantee that the fronts will not cross. Additionally, we call a triangle encroaching if the newly placed vertex comes closer than half of its ideal step length to the fence.

Evaluating the guidance field is another operation that is central to our algorithm. It is impractical to compute $g_{\mathcal{S}}(x)$ by taking the minimum over all the points in $\mathcal{S}$, so we approximate it by densely sampling the surface with points $\tilde{s}_i \in \mathcal{S}$. Note that all $\tilde{g}_i$s grow at the same rate as we move away from the minima. This leads to the following simple and efficient procedure for evaluating $g(x)$ at any given point. First assume that we have an upper bound γ of $g(x)$. Then it can be shown that for all i such that $||\tilde{s}_i - x|| > \gamma/(1-\eta^{-1})$, $\tilde{g}_i(x) > \gamma$. That is, $\tilde{g}_i$ will not change our current estimate of $g(x)$. Therefore, there is no need to inspect $\tilde{g}_i(x)$ when $||\tilde{s}_i - x|| > \gamma/(1-\eta^{-1})$. These observations lead to the procedure outlined in Figure 7 for evaluating $g(x)$, which examines the fewest number of points possible by using a kd-tree to extract the points s_i in ordered distance from x.

4 Experimental Results and Applications

We have tested our implementation by remeshing a large number of models of different characteristics, including models arising from isosurfaces, 3D photography, etc. Several representative examples are shown in Figure 6. Statistics regarding our remeshes are listed in Table 1. It relates ρ to the number of output triangles, the Hausdorff error, and the execution time. The system used to gener-

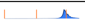
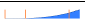
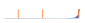
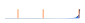
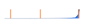
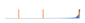
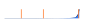
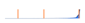
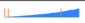
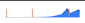
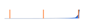
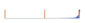
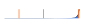
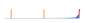
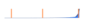
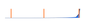
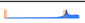
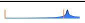
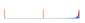
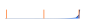
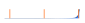
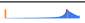
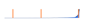
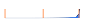
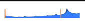
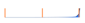
Mesh	ρ	Time	Error	Quality	In	Out	Mesh	ρ	Time	Error	Quality	In	Out
Bunny					69.4		Max					200.0	
	0.5	28.1s	0.24%			34.2		0.5	50.6s	0.12%			45.0
	1.0	18.2s	0.47%			14.0		1.0	36.5s	0.33%			18.7
	1.5	15.2s	0.57%			8.3		1.5	33.0s	0.44%			11.4
Dragon					100.0		Feline					151.0	
	0.5	92.6s	0.23%			120.7		0.5	80.6s	0.10%			130.7
	1.0	49.4s	0.34%			51.3		1.0	45.0s	0.26%			57.5
	1.5	38.8s	0.43%			31.2		1.5	34.7s	0.28%			37.7
Pelvis					529.8		David					1300.8	
	0.5	182s	0.08%			68.5		1.5	286s	0.59%			133.6
	1.0	97.2s	0.14%			26.6	Pensatore					1995.7	
	1.5	93.0s	0.27%			16.2		1.5	371s	0.19%			92.9
							Buddha					1087.7	
								0.8	228s	0.90%			182.9

Table 1: Summary of results of our algorithm. Error is measured as Hausdorff distance, in percent of the bounding box diagonal. Input and output size is measured in thousands of triangles. Quality is shown as a histogram of circle ratios. The three vertical bars show the worst triangle, the first half percentile, and the median.

```
$g(\mathbf{x})$
1  $\gamma \leftarrow \infty$
2  repeat
3          $\mathbf{p} \leftarrow$ NEXTCLOSESTPOINT$(\mathbf{x})$
4          $\gamma \leftarrow \min(\gamma, \tilde{g}_{\mathbf{p}}(\mathbf{x}))$
5     until $||\mathbf{p}-\mathbf{x}|| > \gamma/(1-\eta^{-1})$
6  return $\gamma$
```

Figure 7: Algorithm to compute the guidance field at a given point.

ate these results is a 3.0GHz Pentium D with 2GB of RAM. The amount of RAM required for our experiments ranged from 17MB for the bunny with 69 thousand triangles to 470MB for the Pensatore mesh which has almost two million triangles. Memory consumption is mostly dependent on the data structures used for keeping the guidance field and do not grow significantly with the density of the output mesh as the fronts are the only other data structure that remains in memory. Unless stated otherwise, we use $\eta = 1.2$ for the results described.

One remarkable aspect of our algorithm is that it produces a very consistent distribution of triangle qualities that does not depend on the input mesh. This distribution can be seen in Figure 10, which includes 80 of our remeshes with exceedingly similar quality histograms. Figure 11 shows that our triangulations also have a low Hausdorff error. For ρs large enough to produce fewer output triangles than input triangles, the measured error is about half of our predicted error bound. Figure 9 demonstrates the user's ability to control the output of our algorithm. In addition to being used to generate full remeshes, we can apply our algorithm to other novel applications.

4.1 Local Remeshing for Boolean Operations

Many interactive tools require local remeshing after an operation has been performed. One such example is CSG operations on two meshes $\mathcal{S}_1$ and $\mathcal{S}_2$. Since these operations need to stitch two incompatible triangulations together, they typically create many thin triangles and high valence vertices. A local remeshing is sufficient since only the triangles involved in the intersection are affected.

The procedure for CSG operations begins by finding the intersections of $\mathcal{S}_1$ and $\mathcal{S}_2$, which can be done robustly without user input. We call Λ the set of intersection loops of $\mathcal{S}_1$ and $\mathcal{S}_2$. The local remeshing then begins by marking all triangles within a user defined distance to Λ. These are the triangles that will be deleted and replaced by our triangulation. We call Ω the boundaries between the marked triangles and the unchanged portions of the input meshes. To remesh the marked area, we create a initial front for each loop in Ω, and two fronts with opposing orientation for each loop in Λ. The combination of these fronts surround the area to be remeshed to ensure that only the local region is changed.

Since the surface $\mathcal{S}$ we are now triangulating is defined by $\mathcal{S}_1 \oplus \mathcal{S}_2$, where $\oplus$ is a CSG operation, we must change the guidance field in several ways. In addition to bounding the rate of change of the edge lengths within a single mesh, it must also ensure a smooth gradation between $\mathcal{S}_1$ and $\mathcal{S}_2$. To achieve this, we simply use $\tilde{s}_i \in \mathcal{S}_1 \cup \mathcal{S}_2$. This allows the edge lengths in one surface to be constrained by the ideal lengths of the other. Second, the curvature of Λ may be greater than $\kappa_{\max}$ of either mesh. Since we want them to be represented accurately, we must include their curvature when computing ι. Finally, we would like the edge lengths of the remesh to blend into those of the unchanged portion of the input meshes. Since the edge lengths of the original mesh will not necessarily conform to the ideal lengths that the user specifies with ρ, we must create a transition. We do this by solving the following Laplacian system across the marked triangles where $i \in \Lambda, j \in \Omega$:

$$\Delta f = 0 \quad f_i = 0 \quad f_j = 1$$

This is a linear system that can be efficiently solved. The solution will give a smooth transition between the boundary constraints. The scalar field f is then used to blend between the user defined ideal step length and the lengths of the original edges at a given point.

The example shown in Figure 8 demonstrates our local remeshing after a CSG difference operation has been performed. While most modeling packages will introduce very poor triangles to join the two meshes, we are able to accurately remesh the intersection and blend the edge lengths into those of the input meshes. Only the portion shown in blue has been triangulated by our algorithm. The time required to perform this CSG operation, including finding the intersections, solving the linear system, computing the guidance field for both surfaces, then finally triangulating the local region, was less than 13 seconds.

4.2 Mixed-Mode Boolean Operations

Our generalized algorithm allows a new application, which we call mixed-mode boolean operations. For example, we can do CSG operations between a point-set surface $\mathcal{S}_1$ and a mesh $\mathcal{S}_2$. One way

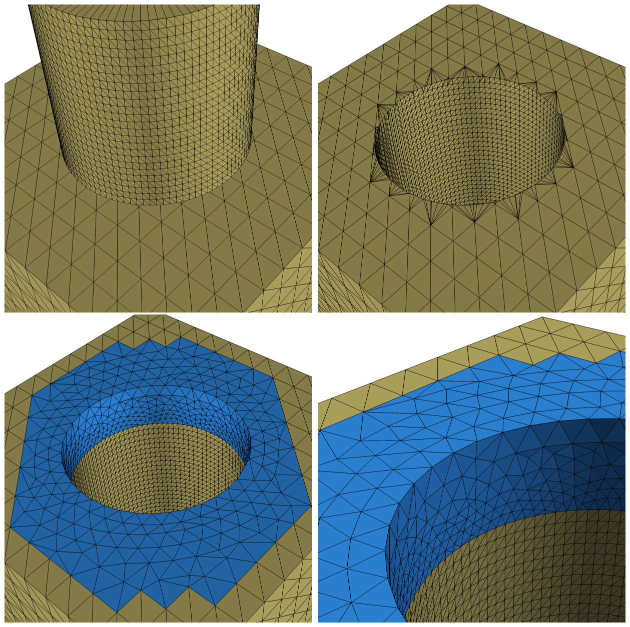


Figure 8: An example of a CSG difference operation. The output generated by Maya is shown in the upper-right corner, while the bottom row shows our operation. The algorithm only changes the portion shown in blue.

to do this would be to triangulate both of the surfaces, then use the procedure outlined above on the resulting meshes. This, however, is not the approach that we use since it introduces an additional error component in the result. Instead, we perform the operation directly on $\mathcal{S}_1$ and $\mathcal{S}_2$. Finding the intersection curves of two arbitrary surfaces is a much more difficult problem than when dealing solely with meshes. In this case, we require a small amount of user input to select a point close to the Λ. This point is then iteratively projected onto each surface until it converges to the intersection. The entire intersection curve is then traversed by moving parallel to both surfaces and reprojecting. This allows the discovery of the intersections of "black box" surfaces with a minimum of user input. If either of the surfaces is a triangle mesh, it will be locally remeshed as described above. Other surface types require that the entire surface be triangulated with initial fronts at the intersection curves.

We show an example of a mixed-mode union between a mesh and an MLS surface in Figure 1. The triangulations of the two surfaces clearly meet to form a watertight mesh, and the edge lengths of the locally remeshed area smoothly blend into the original part of the mesh. This example also demonstrates that our algorithm is oblivious to the genus of both the input surfaces and the output of the CSG operation.

5 Discussion and Limitations

Approximation Error We can bound the error between $\mathcal{S}$ and the generated mesh. In fact, the Hausdorff error between the two surfaces can be bounded by

$$\xi = \left(1 - \sqrt{\frac{1+2\cos\rho}{3}}\right)\frac{1}{2\sin(\rho/2)} \quad (6)$$

$$\varepsilon(\rho) = e_{max}\xi, \quad (7)$$

where e_{max} is the largest edge in the generated triangulation [24]. We can use this bound to compute the maximum distance between any point $\mathbf{x} \in \mathcal{S}$ and the output mesh. Since we have constructed the guidance field such that $\mathbf{x}$ is never crossed by a triangle with edges longer than $\iota(\mathbf{x})$, we can extend Equation 7 to be defined over $\mathcal{S}$ with

$$\varepsilon(\rho,\mathbf{x}) = \iota(\mathbf{x})\xi. \quad (8)$$

We note that these bounds hold for sufficiently smooth approximations. On Figure 11 we show predicted error and measured error using Metro [8]. In practice, our algorithm tends to create meshes with maximum error half of the predicted theoretical bound. We note that when remeshing triangulated surfaces, because of non-smoothness, the theoretical bound on Hausdorff error might not hold for very small ρs. This happens because of the misalignment of a remeshed triangle across edges of the original triangles. We can circumvent most of these problems by identifying creases and triangulating *piecewise smooth* patches of the surface. We simply sample the crease to obey the guidance field, and a high-quality compatible match between the piecewise smooth patches can be generated.

Triangle Quality For every free triangle, we can give strong bounds on the aspect ratio of the generated triangle. The reason is that the Lipschitz property of the guidance field can be used to enforce that the edge lengths for these edges also satisfy a Lipschitz condition. For each of these vertices, the ratio of the shortest to longest incident edge is greater than or equal to the reduction factor. This automatically creates a smooth gradation between triangle sizes when going from high to low curvature areas. Furthermore, for each of the free triangles, the ratio of the longest to shortest edge will always be less than or equal to η. When η is close to one, all free triangles will be close to equilateral.

However, these guarantees do not exist for connection triangles. This is an inherent obstacle for advancing front algorithms, because an existing vertex needs to be used. Hence, the error bound for free triangles does not hold for connection triangles. In these areas, where the fronts are merged and split we use heuristics [16]. In practice, bad triangles rarely happen. See histograms shown in Figure 10. There, we show the ratio of incircle to circumcircle of a triangle (normalized to $[0,1]$) as a measure of triangle quality. Notice that the first half percentile of all remeshes have ratio 0.5 or better: 99.5% of the triangles have acceptable quality, and all of them have the median ratio within 3% of optimal. We also performed a series of edge flips on all meshes, to increase the minimum angles. While edge flips caused the overall worst remeshed triangle

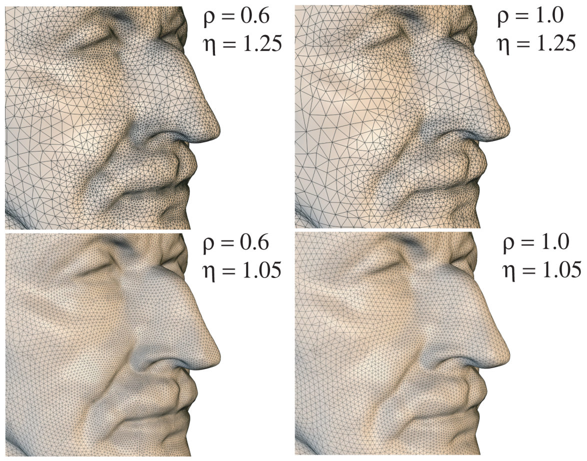


Figure 9: The effects of ρ and η on the resulting mesh. ρ controls the approximation accuracy: a bigger ρ will result in a coarser triangulation. η controls triangle grading: a larger η will result in more adaptive triangulations.

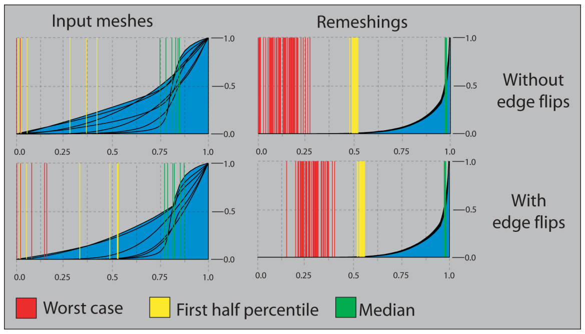


Figure 10: Cumulative histograms of incircle to circumcircle ratios. The histogram is normalized so that the best ratio is 1.0. The left column shows all the input meshes used in the paper, and the column in the right shows the results of all remeshes shown in this paper (80 in total).

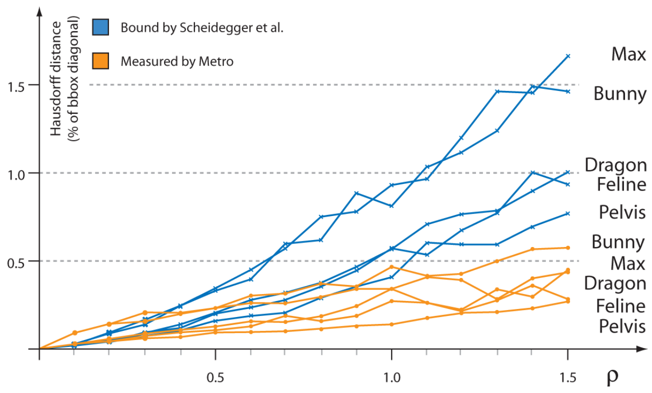


Figure 11: Predicted error and measured error, in percent of the bounding box diagonal.

(among the 30 million generated throughout) to have aspect ratio greater than 1/8, it did not improve the worst case triangles of the original meshes. Finally, edge flips did not significantly change the overall histogram of our meshes, indicating that our connectivity choices were adequate most of the time.

Performance Our algorithm is fast and requires a small amount of RAM. The bottleneck is in the construction and evaluation of the guidance field, which takes approximately half of the execution time. The running time of our remesher is typically measured in seconds or minutes. See Table 1 for detailed timings for a number of meshes of varying sizes (up to around 2 million triangles). The new mesh is streamed to disk as it is created, so the memory footprint is only dependent on the size of the input surface and the number of edges in the advancing fronts.

User Control The coarseness and quality of the mesh are entirely controlled by two parameters: $\rho \in (0, 2\pi/3]$ and $\eta \in (1,2)$. ρ directly controls the coarseness of the triangles by scaling the ideal edge length at each point. η is a compromise between adaptability of the triangle size to the surface, and quality of the triangle shape. A small η will not allow the triangle size to change quickly, resulting in nearly equilateral triangles, but sizing them much smaller than required for the local curvature. Using a larger η allows more adaptive triangles, but results in more triangles with poor aspect ratios. We have found setting $\eta = 1.2$ to perform very well, and we typically set $0.2 \leq \rho \leq 1.5$, depending on the desired coarseness of the resulting mesh.

Comparison to Scheidegger et al. Though the algorithm of Scheidegger et al. [24] served as a stepping stone to our algorithm, there are several significant differences between them. The accumulation of these differences results in an algorithm that produces higher quality triangles, can easily be applied to many more meshing scenarios, and is faster and more robust.

One of the most important differences is in the construction and evaluation of the guidance field. Though they mention a Lipschitz condition on the edge lengths, their guidance field construction does not enforce it. We provide a principled way to create the guidance field, which can be proven to be Lipschitz with constant $(1-\eta^{-1})$, regardless of the ideal edge lengths ι. Our procedure for evaluating the guidance field is only a function of the point location that it is being evaluated at, rather than the point location and the incident edge lengths of our output mesh. This additionally increases the robustness of our algorithm because it prevents suboptimal edge lengths (from connected triangles) from cascading throughout the triangulation. Our evaluation procedure is also more efficient since it only searches in the smallest radius necessary to determine the correct result.

Another significant difference is in the robustness of our front intersection tests. We use the bound on the Hausdorff distance to ensure that the fronts will not cross each other. This results in a much more reliable meshing tool that is not as sensitive to the input.

Our algorithm is more widely applicable than theirs. We treat $\mathcal{S}$ in a completely abstract way which allows us to plug any surface definition into our system. We can also robustly and accurately mesh surfaces with boundaries and sharp features.

6 Conclusions and Future Work

We have presented a novel algorithm for meshing a large class of surfaces. Our technique can be used for remeshing a complete surface, or selectively triangulating a portion of it. We exploit this to perform high quality CSG operations and mixed-mode CSG. The algorithm is fast, robust, simple to implement, and produces very high quality meshes.

We believe that stronger properties of the algorithm can be shown by using different ιs, such as the popular local feature size [23]. The same strategy may be used to preserve other features of the surface. We would also like to study the necessary sampling conditions for the guidance field to correctly bound ι over the entire surface rather than only at a region around $\tilde{s}$. The algorithm is currently being extended to generate compatible contact surfaces for finite element simulations of the mechanics of hip dysplasia. We believe that these are all an exciting directions for future work.

References

[1] Anders Adamson and Marc Alexa. Approximating and intersecting surfaces from points. In *Symposium on Geometry Processing*, 2003.

[2] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization 2001*, pages 21–28, 2001.

[3] P. Alliez, G. Ucelli, C. Gotsman, and M. Attene. Recent advances in remeshing of surfaces. In *State-of-the-art report of the AIM@SHAPE EU network*. Springer, 2005.

[4] Pierre Alliez, David Cohen-Steiner, Mariette Yvinec, and Mathieu Desbrun. Variational tetrahedral meshing. *ACM Trans. Graph.*, 24(3):617–625, 2005.

[5] Nina Amenta and Yong Joo Kil. Defining point-set surfaces. *ACM Trans. Graph. (Proceedings of ACM SIGGRAPH 2004)*, 23(3):264–270, 2004.

[6] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[7] J. D. Boissonnat and S. Oudot. Provably good surface sampling and approximation. In *Symposium on Geometry Processing*, pages 9–18, 2003.

[8] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.

[9] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *SIGGRAPH 1999*, pages 317–324, 1999.

[10] M. do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, Englewood Cliffs, New Jersey, 1976.

[11] S. Fleishman, D. Cohen-Or, and C. Silva. Robust moving least squares fitting with sharp features. *ACM Transactions on Graphics*, 2005.

[12] R. V. Garimella and B. K. Swartz. Curvature estimation for unstructured triangulations of surfaces. Technical Report LA-UR-03-8240, Los Alamos National Laboratory, November 2003.

[13] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *SIGGRAPH 99*, pages 325–334, 1999.

[14] Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. In *SIGGRAPH 2000*, pages 95–102, 2000.

[15] Klaus Hildebrandt, Konrad Polthier, and Max Wardetzky. Smooth feature lines on surface meshes. In *Eurographics Symposium on Geometry Processing*, pages 85–90, 2005.

[16] T. Karkanis and A.J. Stewart. Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications*, 21(2):60–69, 2001.

[17] Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *SIGGRAPH 98*, pages 95–104, 1998.

[18] Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *ACM SIGGRAPH 2000*, pages 131–144, 2000.

[19] W. Lorensen and H. Cline. Marching cubes: A high resolution 3d surface reconstruction algorithm. In *SIGGRAPH 1987*, pages 163–169, 1987.

[20] Y. Matsumoto. *Introduction to Morse Theory*. American Mathematical Society, 1997.

[21] Mark Meyer, Mathieu Desbrun, Schröder, and Alan H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. *VisMath*, 2002.

[22] Gregory M. Nielson and Bernd Hamann. The asymptotic decider: resolving the ambiguity in marching cubes. In *IEEE Visualization 1991*, pages 83–91, 1991.

[23] Jim Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. In *Symposium on Discrete Algorithms*, 1993.

[24] Carlos E. Scheidegger, Shachar Fleishman, and Claudio T. Silva. Triangulating point set surfaces with bounded error. In *Symposium on Geometry Processing*, pages 63–72, 2005.

[25] Olga Sorkine. State-of-the-art report: Laplacian mesh processing. In *Proceedings of Eurographics*, 2005.

[26] Olga Sorkine, Yaron Lipman, Daniel Cohen-Or, Marc Alexa, Christian Rössl, and Hans-Peter Seidel. Laplacian surface editing. In *Symposium on Geometry processing*, pages 179–188, 2004.

[27] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popovic. Mesh-based inverse kinematics. *ACM Trans. Graph.*, 24(3):488–495, 2005.

[28] Vitaly Surazhsky and Craig Gotsman. Explicit surface remeshing. In *Symposium on Geometry Processing*, pages 20–30, 2003.

[29] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(3):644–651, 2004.