Eurographics Symposium on Geometry Processing (2005)
M. Desbrun, H. Pottmann (Editors)

Triangulating Point Set Surfaces with Bounded Error

Carlos E. Scheidegger Shachar Fleishman Cláudio T. Silva

University of Utah

Abstract

We introduce an algorithm for constructing a high-quality triangulation directly from Point Set Surfaces. Our algorithm requires no intermediate representation and no post-processing of the output, and naturally handles noisy input data, typically in the form of a set of registered range scans. It creates a triangulation where triangle size respects the geometry of the surface rather than the sampling density of the range scans. Our technique does not require normal information, but still produces a consistent orientation of the triangles, assuming the sampled surface is an orientable two-manifold. Our work is based on using Moving Least-Squares (MLS) surfaces as the underlying representation. Our technique is a novel advancing front algorithm, that bounds the Hausdorff distance to within a user-specified limit. Specifically, we introduce a way of augmenting advancing front algorithms with global information, so that triangle size adapts gracefully even when there are large changes in surface curvature. Our results show that our technique generates high-quality triangulations where other techniques fail to reconstruct the correct surface due to irregular sampling on the point cloud, noise, registration artifacts, and underlying geometric features, such as regions with high curvature gradients.

1. Introduction

Point sets are now a popular modeling primitive in computer graphics. The initial need to model surfaces out of point sets came mostly from the emergence of affordable and accurate scanning devices able to generate dense point sets, which are initial representations of physical models [LPC*00]. Still, creating manifold surfaces out of point sets has shown to be useful in a variety of other applications, since point sets can be used a robust technique for representing surfaces. Among the many advantages of point sets are their generality: every shape can be represented by a set of points on its boundary, where the degree of accuracy typically depends only on the number of points.

Our motivation is that existing surface reconstruction techniques require a set of pre-processing and post-processing steps to handle noisy data and generate high-quality triangulations. The need for multiple steps makes it hard to maintain proper error bounds in the complete pipeline. For instance, for interpolatory schemes to create a good reconstruction, they require a preprocessing step that removes noise and makes sure each sample belongs to the surface [BMR*99]. If one wants an efficient, high-quality triangulation, potentially expensive post-processing steps such as remeshing and simplification become necessary. Figure 1 shows a number of different approaches to reconstructing a surface from a noisy point cloud. Specifically, we note that it is necessary to use pre-filtering (see insets (b), (c) and (d)), to have the triangulation accurately represent the surface. These use, however, an order of magnitude more triangles than we do. If further simplification is performed, detail is lost, as can be seen from the shrinkage in (e). Our technique (inset (f)) produces a mesh within a user-specified error-bound directly from the noisy data (see the triangulated Bunny in the figure).

In our work, we use the particularly powerful MLS approach for surface reconstruction [ABCO*01, Lev03]. One of the strengths of MLS surfaces is the intrinsic ability to handle noisy input. Furthermore, the MLS technique makes it simple to approximate intrinsic properties of the surface such as normal and curvature directly from a noisy point cloud. One downside of the MLS approach is that it does not directly lead to a mesh representation. Meshes are ubiquitous in digital geometry processing, and triangulated models are many times only intermediate steps in more complex systems [GSS99, LSS*98]. Reconstructing a triangle mesh from other surface representations is a fundamental tool.

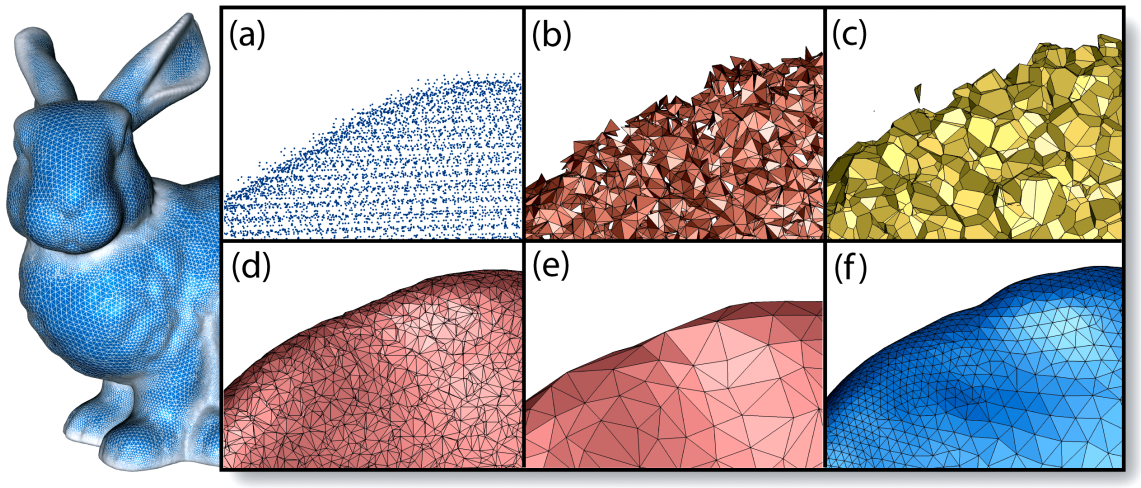

Figure 1: *Raw range scans contain significant noise, as can be seen in inset (a), a detail of the Stanford Bunny's left ear. Insets (b) and (c) show direct reconstructions using cocone and powercrust, respectively. Inset (d) shows a cocone reconstruction after cleaning up the point cloud using MLS smoothing. Inset (e) shows a QEM-based simplification of (d). Inset (f) shows our reconstruction, without any additional preprocessing or post-processing. The triangulations in (e) and (f) have the same triangle count over the whole model. Because the tip of the ear has high curvature, our technique samples it more densely than (e).*

In this paper, we propose a new technique for creating a triangulated mesh from a Point Set Surface (PSS) [ABCO*01]. This defines a large class of models that includes reconstructing a polygonal mesh from potentially noisy point clouds without normals. In our work, we have focused on producing a high-quality triangulation directly from point clouds that requires no pre-smoothing. Our work is inspired by the triangulation algorithm for implicit surfaces by Karkanis and Stewart [KS01]. Unlike interpolatory algorithms that always triangulate the input points, our technique generates a high-quality triangulation by sampling the surface at points appropriate to the local geometry, rather than necessarily at the input point set. The algorithm couples the machinery of MLS with the idea of advancing fronts. A key difference of our approach is the addition of a global guidance field to overcome the excessive locality of decisions in typical advancing front algorithms. Furthermore, since we do not resort to implicit surface definitions, we can also straightforwardly handle surface boundaries. Our main contributions are:

- a novel advancing-front algorithm for meshing point-set surfaces that works directly on the noisy point cloud data and produces a high-quality triangulation to within a user-specified error bound;
- the idea of a global guidance field to inform the algorithm about non-local surface features;
- a simple and robust implementation of the techniques described;
- experimental comparison to other techniques.

2. Related work

Computational geometry approaches use Voronoi diagrams and Delaunay triangulations as building blocks for the reconstruction [ABK98, GKS00, KSO04]. Some examples of these are the Power Crust family [ACK01] and the cocone family [ACDL00] of algorithms. Typically, these algorithms treat every point in the input set as a vertex on the resulting mesh. Although these algorithms provably produce a triangulation under some assumptions, they necessarily interpolate at least a subset of the input points. As such, noisy point input will give undesired results, whatever the sampling density. In fact, the denser the sampling, the worse the problem will be. For further details, we point the interested reader to the excellent survey of Cazals and Giesen [CG04].

A popular way of meshing surfaces is to define them implicitly as the zero-set of a function in space. In this setting, most algorithms subdivide the space in a sufficiently fine way, so that by analyzing the signs of the functionals at a few places, it is possible to detect the presence or absence of a piece of surface. The technique of Curless and Levoy [CL96] is based on such a volumetric approach. The zero-set of the volume is then tessellated using Marching Cubes [LC87]. Even though recent developments such as Dual Contouring make it possible to reconstruct sharp features [JLSW02], most of the vertices lie on the intersections with the grid lines. The algorithms then output skinnier triangles than would otherwise be possible. Also, they cannot naturally adapt to a guidance field without post-processing.

There is extensive previous work on defining implicit functions from point clouds. Hoppe et al. [HDD*92] proposed one of the first ones, based on local estimations of the

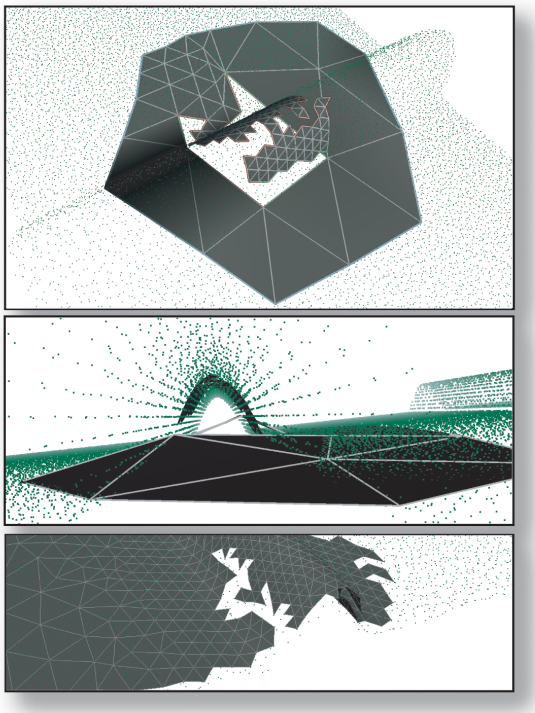

Figure 2: *Without the guidance field information, pieces of the fronts can meet one another with different triangle sizes, leading to arbitrarily bad triangles. Also, features may be missed entirely. The top and middle images are two different views of the same triangulation, created by removing the lookahead machinery from our implementation. The bottom image shows our implementation with the guidance field enabled – the fronts always meet one another in the correct resolution and no features are missed.*

distance field. We can use RBFs to solve the thin-plate spline interpolation problem [TO02]. These are slower than MPUs, local fittings joined together smoothly by weighted averaging [OBA*03]. These implicit reconstruction algorithms typically require an additional post-processing step to improve the quality of the triangulation. Most remeshing algorithms are based on reparameterization [FH04, SAG03], and as such, are expensive to compute. Moreover, it is not straightforward to have such algorithms work in massive models. Ideally, we would like our reconstruction algorithm to be able to output a good triangulation without additional requirements.

A set of efficient meshing algorithms have been based on *advancing fronts* (also called *surface tracking*). The main observation is that since a surface can be seen as a collection of localized features, each piece of the reconstruction (be it a triangle, a spline patch, etc.) should be decided locally. Composing these local decisions naturally leads to advancing front algorithms. Our work is based on the same principle. In fact, many successful algorithms use this concept

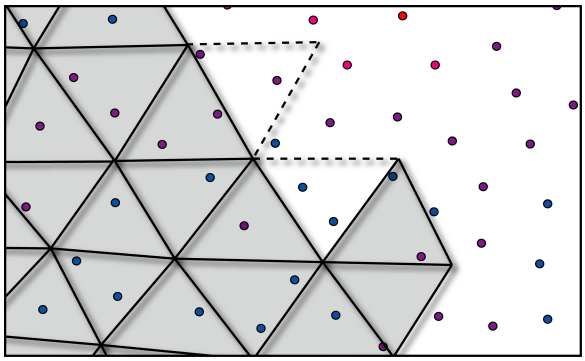

Figure 3: *The basic operations in front advancing algorithms. To grow a front, we either add a new vertex to the triangulation and* grow a triangle *or we use three adjacent front vertices to* cut an ear. *Front merging and splitting uses an existing front vertex for triangle growing, instead of a newly placed one.*

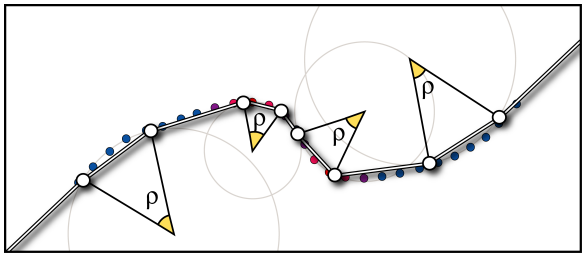

Figure 4: *In an optimal curvature-adaptive triangulation, each triangle edge subtends the same angle of the osculating circle. This angle is user-specified and is a natural way of controlling the approximation error.*

[BMR*99, KS01]. Our algorithm is most similar to Karkanis and Stewart [KS01], where they propose an advancing front technique for meshing implicit surfaces. Their triangulation technique samples the surface adaptively with respect to the curvature, but it might miss features as shown in Figure 2. Our work extends their ideas by using a separate guidance field that allows us to *proactively* correct the triangle sizes that sample surface, resulting in guaranteed surface coverage. For the most part, we use curvature as the guidance field in our work, but any other measure of granularity could be allowed: the field is represented by the restriction of a scalar function in space. Additionally, we replace all the implicit surface machinery by Point Set Surfaces [ABCO*01], which is defined using a projection operation. Since the advancing front algorithm needs to "settle" vertices on the surface, the MLS projection is a natural candidate for such an operation. Furthermore, it allows us to reconstruct partial scans and surfaces with boundaries, since holes can be easily treated in the Point-Set Surfaces framework.

Our work is related to Cheng and Shi's [CS04], in the sense that they propose a technique for meshing a class of surfaces known as Molecular Skin Surfaces. We also propose an algorithm that meshes a certain, albeit more general, class of surfaces, within a given error bound. Our technique

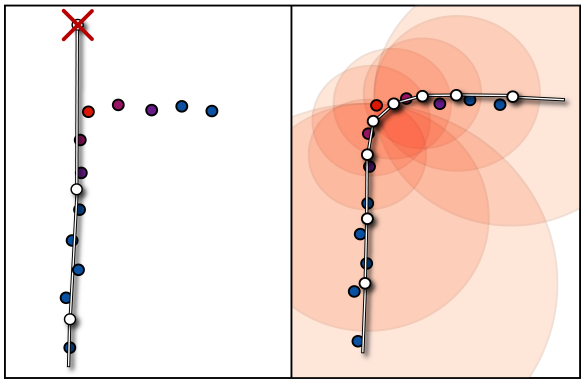

Figure 5: *Retroactive correction (left) may miss the surface, while proactive correction (right) guarantees feature coverage. The connected lines are the triangles, and the white points are the the triangulation vertices. The colored points are a subset of the input point cloud, where blue points indicate low curvature, and red ones indicate high curvature. Locally, the vertices on the left figure look adequate, but the vertex crossed out will be arbitrarily far away from the surface. By querying the curvature of the surface in the neighborhood (depicted by the red circles), the algorithm shrinks the triangles and properly covers the feature.*

is also reminiscent of Boissonnat and Oudot's surface sampling technique [BO03] in that their technique also uses a guidance field.

3. The triangulation algorithm

Our algorithm takes as input a set of points P without normals and produces a high quality triangle mesh. Starting with a seed triangle, its edges form *the front*, that is incrementally extended by two types of local operations. A *triangle growing* operation connects a new triangle to one edge of the front and adds a new vertex. An *ear-cut* operation creates a triangle from three adjacent vertices on the front (see Figure 3). Sometimes, a triangle growing step will add a new vertex that is too close to the current triangulation. This new vertex is then merged with an existing one, causing fronts to either merge or split, and changing the topology of the surface. The algorithm terminates when there are no more fronts to be processed.

The process of obtaining a point cloud from a real-life model through registration of a series of range scans is far from exact. For one, the scanner has limited precision and intrinsic noise. Not only that, but the noise for each sample is not independent: it is usually a function of the angle between the surface normal, the scan direction, and possibly of the surface material. Additionally, the registration process usually fails to exactly match the different scans. In general, the resulting point cloud does not approximate the original surface, even as the sampling density increases without bound. We use the moving least-squares (MLS) technique for specifying the underlying surface [ABCO*01, Lev03], and triangulate it to within a user-defined resolution. New vertices

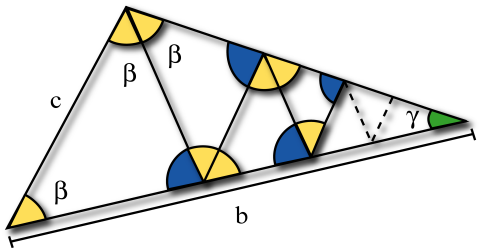

Figure 6: *The proactive triangle adaptation leads to a sequence of shrinking triangles. Assuming the first triangle is small enough sizes, the next one will also be. This means that b is an upper bound in the querying region size.*

are generated using a two-step algorithm. A vertex position is first estimated using a vertex prediction operator. The vertex is then projected on the MLS surface. The vertex prediction considers the current edge length and the maximum curvature in the triangle neighborhood to find a vertex position that forms a high-quality triangle and also guaranteedly covers the surface features.

3.1. The vertex prediction operator

Ideally, a triangulation should be adaptive: more triangles should be used in more featureful areas. We make precise this notion by considering curvature as a direct measure of our interest in the surface. In this sense, an optimal triangulation is one where each triangle subtends a constant solid angle of the *osculating sphere*. We consider the osculating sphere having a radius equal to the inverse of the maximum absolute curvature $\kappa = \max(|\kappa_1|, |\kappa_2|)$. We then let the user specify ρ, which is the angle of the osculating circle (a grand circle of the osculating sphere) a triangle edge should optimally subtend. The ideal edge length at a given point in the surface is then $\mathscr{L} = \rho/\kappa$. The approximation error is thus defined by ρ (see Figure 4).

A basic curvature-adaptive technique is to locally determine the ideal edge size near the expanding triangles and have the expanding triangles be closer to the ideal edge sizes, either shrinking or expanding them. This works well for simple surfaces with no abrupt changes in curvature. These simpler surfaces allow an algorithm to be *retroactively corrective*: the front adapts only *after* "seeing" changes in surface curvature. This assumption is central to Karkanis and Stewart's algorithm: by only allowing changes in triangle size when curvature is effectively measured, there is no guarantee that features will be captured, as illustrated in Figure 5 (left). In fact, since the curvature is basically the derivative of the normal (assuming arc-length parameterization), we can think of the normal as being determined by the integral of the curvature. It follows directly that for retroactively corrective methods to work, the integral of the curvature for the "next" triangle must be at most a certain fraction of the integral of the curvature of the "current" triangle. Thus one must either

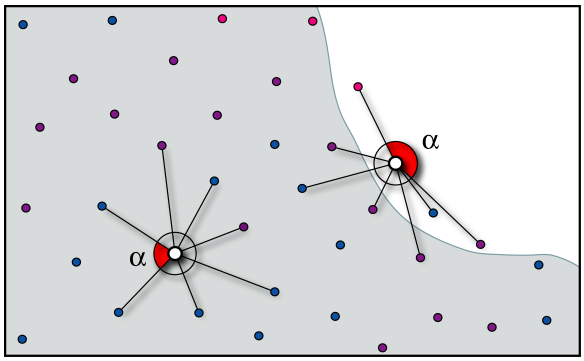

Figure 7: *The largest projected angle measures closeness to a boundary.*

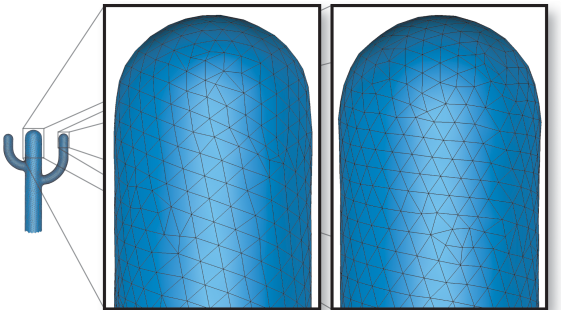

Figure 8: *Optimal curvature-adaptive sampling on a synthetic cactus. $\rho = \pi/16$. Notice that the main trunk and the branch triangles have different sizes, but they are in the same proportion to the branch radius, which is precisely the curvature radius for the regions.*

choose an extremely small ρ or make assumptions about the surfaces. For arbitrary surfaces such as MLS surfaces, we want to guarantee that the integral of the curvature under each triangle is bounded.

We introduce a general and simple way of augmenting advancing front algorithms with global information. We achieve this by introducing a *guidance field*. This field represents the ideal edge size for a triangle in the reconstructed surface at each point in space. The guidance field is decoupled both from the surface to be triangulated and from the mesh reconstruction: it is simply a scalar function defined on the embedding space of the triangulated surface. In our work, we use $\mathscr{L}(x,y,z)$ as the guidance field. We sample this function at every point in the point set by projecting it on the MLS surface and determining the curvature from the local bivariate polynomials. The function is then extended to R^3 by having the ideal edge length at any point in space be the ideal edge length at the closest point in the point set.

As observed before, in regular front advancing algorithms the triangle sizes on the front at a given state may look adequate, but a feature that requires finer resolution is beyond the front *horizon* (the "one triangle thick" area around a front) as shown in Figure 5(a).

To adapt the triangulation to features that are to eventually appear in the horizon, a new triangle may be made slightly smaller than ideal. Now imagine that at some point (possibly far away from the front horizon), there is a feature of unbounded curvature. In other words, the ideal size of the triangle in that point approaches zero. We show that front advancing, informed by a guidance field, can always adapt to these distant features in the surface. Not only that, but it is possible to do so by querying the field inside a *finite* region, no matter how small the feature is or how large front triangles are.

We always grow triangles with isosceles triangles, and so the change in the triangle edge lengths is achieved by changing the base angle β, the only free parameter in the new triangle. The main geometric insight is shown in Figure 6: the edges of an infinite sequence of shrinking isosceles triangles form a geometric series, the sum of whose terms is finite. Additionally, we can see that by only assuming that the current triangle size is appropriate, the following triangle will also be, since in the limit the triangle sizes go to zero as they get close to the point of infinite detail. This means that the querying radius needs only be as large as b. Instead of computing the sum directly, we use simple triangle geometry as follows:

$$\frac{\sin(2\beta)}{b} = \frac{\sin(\gamma)}{c} = \frac{\sin(3\beta)}{c}, \tag{1}$$

and thus

$$b = \frac{\sin 2\beta}{\sin 3\beta} \cdot c = \eta \cdot c. \tag{2}$$

There are several consequences of this simple geometric result. The guidance field needs to be checked only inside a sphere of radius $\eta \cdot c$. This is enough for the triangulation to adapt to arbitrarily small features. Also, the radius is proportional to the triangle size, meeting our expectations that the algorithm should be more "conservative" when using larger triangles. Still, the search radius is surprisingly small. This can be seen by taking η to be a function of β. Even though we have $\lim_{\beta \to 60^\circ} \eta = \infty$, η decreases very quickly. For example, $\beta = 55^\circ$ gives $\eta \approx 3.63$. If we were to only slightly change β to 50°, we would have $\eta \approx 1.96$, almost halving the querying radius. This allows trading off triangle quality for efficiency, while still guaranteeing feature coverage. One can see this querying of the curvature field as a way to have the triangle sizes enforce an analog to a Lipschitz condition along arcs of the surface, with β then controlling the corresponding Lipschitz number. Given an adequately sized edge in which to grow a new triangle, the following procedure returns a point on the MLS surface in a way that guarantees feature coverage and triangle quality:

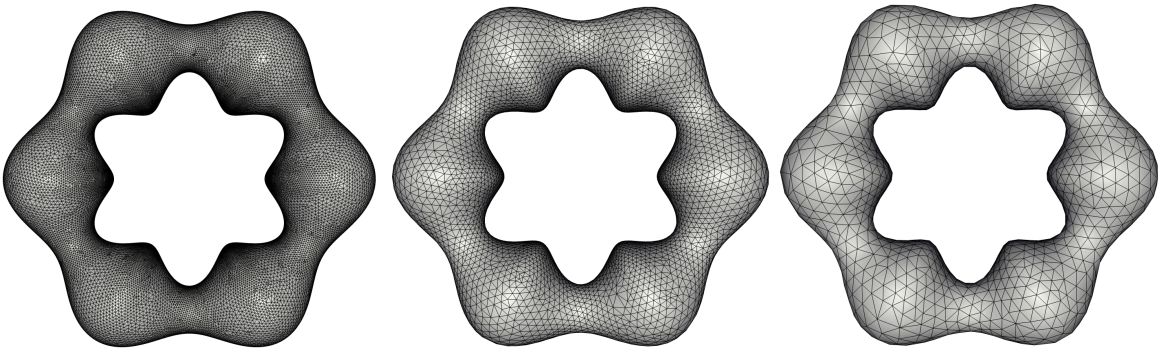

Figure 9: *The ρ parameter controls the quality of the reconstruction: a sequence of wavy torus triangulations using different ρ values (from left to right, $\rho = \pi/30, \pi/10, \pi/4$)*

VERTEX-PREDICT(EDGE e, FIELD F)

1 $c \leftarrow ||e||$
2 $b \leftarrow c \cdot \eta$
3 $mp \leftarrow$ MIDPOINT(e)
4 $i \leftarrow$ FIELD-MIN-IN-SPHERE(F, *Sphere*(mp, b))
5 Let t be a triangle with edge lengths c, i, i
6 Clamp i so that base angle of $t \in [60-\beta, 60+\beta]$
7 Let p be a point lying on the plane of the adjacent triangle over *edge*, so that the vertices of e and p form a triangle t
8 **return** MLS-PROJECT(p)

3.2. The advancing front algorithm

The seeding of the triangulation happens at a random spot, picked by taking a random point of the point cloud and projecting in on the MLS surface. There is no previous triangle from which we are growing a new one, so we have a free parameter for the triangle edge length. We then force an equilateral triangle and look for an edge size appropriate for the region. We achieve that by interpreting the curvature query at a point as a function $f : R \rightarrow R$. We then use bracketing and interval bisection to find the best initial size for the triangle. The best triangle will be the one whose query returns its own size, that is, a fixpoint of f. This forms the inductive basis of the inductive argument described above.

From the initial triangle, we advance the front through *ear cutting* and *triangle growing*. These operations do not change the topology of the surface. Eventually, a front may meet with itself, causing a *front split* and the creation of a topological hole over the surface. When we have more than one front, two of the fronts may meet with one another, causing a *front merge* and the creation of a topological handle on the surface. If a front has only three vertices, we close it with a triangle, closing a hole. We call splits, merges and front closures *topological events*. When there are no more fronts to be advanced, the algorithm terminates. Pseudocode for the main loop follows:

TRIANGULATE(FIELD F)

1 *fronts* $\leftarrow$ FIRST-FRONT()
2 **while** |*front-set*| > 0
3 **do** *current* $\leftarrow$ first[*fronts*]
4 **if** |*current*| $= 3$
5 **then** CLOSE-FRONT(*current*)
6 *fronts* $-=$ *current*
7 **continue**
8 $e \leftarrow$ BEST-EDGE(*current*)
9 **if** CAN-CUT-EAR(e)
10 **then** CUT-EAR(e)
11 **continue**
12 $p \leftarrow$ VERTEX-PREDICT(e, F)
13 **if** not TRIANGLE-TOO-CLOSE(e, p)
14 **then** GROW-TRIANGLE(e)
15 **else** *front* $\leftarrow$ CLOSEST-FRONT(e, p)
16 **if** *front* $=$ *current*
17 **then** *fronts* $+=$ SPLIT(*current-front*)
18 **else** MERGE(*current*, *front*)
19 *fronts* $-=$ *front*

CAN-CUT-EAR() returns true if it is possible to form a good triangle by connecting the edge e to any of the adjacent edges. We allow ear cuts to happen when all the angles in the resulting triangle are less than 70 degrees. If this is not the case, then triangle growing is performed. TRIANGLE-TOO-CLOSE() checks to see whether the added point is closer than allowed to the existing triangulation. We define this distance as half the ideal edge length at the point p. If it is the case, then a topological event will necessarily occur.

We note that front edges that are created through CUT-EAR and GROW-TRIANGLE are always of adequate size. Edges created by topological events, on the other hand, might not be. This happens because SPLIT and MERGE do not choose the position of the vertices in the operations: they use vertices that are already present in the triangulation. The resulting configuration is one where some edges in the front are closer to their ideal edge length than others.

We solve this by keeping each front as a data structure that

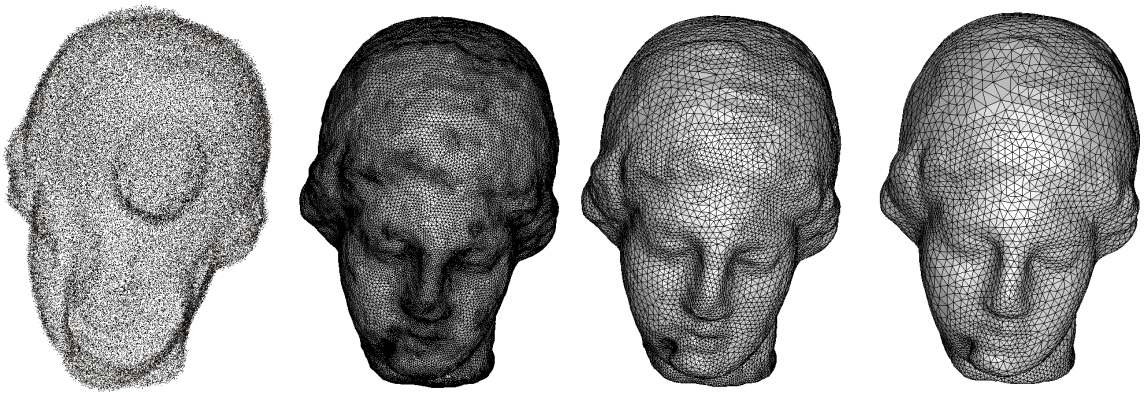

Figure 10: *Triangulations of the Igea model, perturbed with uniform noise equal to 2% of the bounding box diagonal. Even with significant noise, the technique successfully reconstructs the model.*

combines doubly linked lists and a priority queue. Through the linked list interface we traverse the front efficiently, and through the priority queue we extract the best edge at each step of the main loop. The best edge is one whose ratio of length to ideal length is closest to one. To further avoid bad triangles, we introduce the notion of *deferred* edges. Every edge whose expansion would introduce a topological event can potentially introduce a bad triangle. So every time a regular edge would create a topological event, we defer this event by lowering the edge priority below all regular ones. This allows good edges to try and cover the surface in the best way possible.

We keep the triangulation spatially coherent by allowing the algorithm to work only on a given neighborhood at a time. All edges that are outside this working space are marked as *frozen* and ignored until the neighborhood is fully triangulated.

3.3. Point sets with boundaries

The final feature of our algorithm is the ability to reconstruct surfaces with boundaries. To do so, we define a notion of *anisotropy* of a neighborhood in a point-set. Similarly to Linsen's method [Lin01], we observe that near the boundaries of a scan, neighborhoods of a given point tend not to distribute evenly around the point that will be projected. We define the anisotropy of a neighborhood based on a best-fitting local plane. We take all lines from neighborhood points to the reference point and project them on the local plane. Sorting these lines radially gives one angle for each pair of consecutive projected lines. The closer the reference point is to a boundary, the larger the largest of the angles will be, as illustrated by Figure 7.

The procedure NEW-POINT is then updated to check for points whose largest angle is above a certain threshold (we use 150°). If a point to be expanded is outside the point set, the corresponding front edge is marked as a boundary edge. When all edges in a front are marked as boundary, that whole

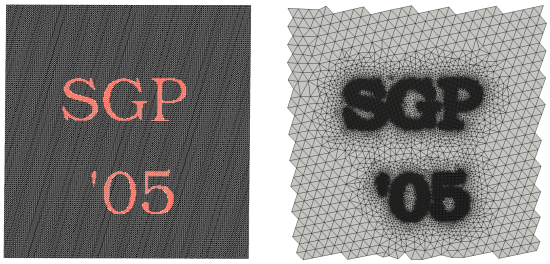

Figure 11: *Arbitrary guidance fields can be used to add additional triangles where necessary on the resulting triangulation. The algorithm is otherwise unchanged.*

front is reported as a boundary of the point-set. Finally, we must consider boundaries as interesting features: if only surface curvature is considered, holes in very flat areas may be skipped over. Our solution is to include anisotropy information in the guidance field: if the angles are too large, we just set the ideal edge length to the radius of curvature of the boundary curve. This is an example of the generality of our informed approach: we only add the boundary information to the guidance field, and the rest of the algorithm remains unchanged.

4. Results

We implemented the algorithm described above and all MLS machinery in C++, on a Red Hat Linux system with a Pentium 4 processor running at 2.8 GHz and 1.5 GB of main memory. Our technique requires little memory: besides acceleration structures such as a kd-tree and an octree, the only data residing on memory are the points, the guidance field, the partial triangulation and the current fronts. During our testing, our implementation never used more than 200MB of main memory.

We report here typical results of our algorithm. The first example shows the optimal curvature-adaptive sampling on a synthetic cactus dataset. The two insets in Figure 8 show

Input	ρ	$\|P\|$	Time Prep.	Time Run	Mem (MB)	Output ktri.
Cactus	$\pi/16$	3.3k	2.7s	63.6s	5	21
	$\pi/8$			20.3s	5	7
	$\pi/4$			9.4s	5	3
Torus	$\pi/16$	30k	44.8s	80.8s	13	24
	$\pi/8$			32.5s	13	8
	$\pi/4$			22.5s	13	3
Igea	$\pi/16$	276k	403s	892s	29	91
	$\pi/8$			357s	29	32
	$\pi/4$			100s	29	11
Bunny	$\pi/16$	760k	90m	90m	160	172

Table 1: *Summary of results of our algorithm.*

that the triangulation took almost exactly the same number of triangles to complete the main trunk and the branches, even though they have different diameters. Our second example illustrates how ρ can be effectively used to control the approximation error of the reconstruction. We use a synthetic undulating torus, in which the different curvature features are very clear (see Figure 9).

We ran experimental comparisons with two widely-known surface reconstruction codes: powercrust and cocone. The reconstructions were run directly from the registered Stanford Bunny point cloud. There are two main issues with interpolatory reconstructions: they assume no noise and they have no notion of triangle quality. As can be seen in Figure 1, this has significant impact on the reconstructed models. Figure 10 shows several triangulations of the Igea model, with noise equal to 2% of the bounding box diagonal artificially added. Our technique handles noise gracefully and doesn't significantly degrade the original model.

In this work, we use curvature information in the guidance field. Still, nothing in our algorithm precludes using different data. The user might want to specify additional detail on certain parts of the mesh if, for example, some numerical simulation will be performed in the triangles. We illustrate this possibility in Figure 11, where a completely arbitrary guidance field is specified. Note that the field is quite discontinuous, but the algorithm gracefully adapts ahead of time so that triangle quality is kept.

5. Discussion

The only two user-specified parameters are ρ and τ. The smoothing factor τ is related to the MLS projection procedure, and should be directly proportional to the amount of smoothing or denoising to be applied. This factor τ relates the MLS h value to the neighborhood radius (see Appendix A). The remaining parameter ρ specifies the quality of the reconstruction in terms of the subtended angle of the osculating circle. Then, the Hausdorff distance ε between triangle mesh and the surface patch under it can be shown to be at most $r(1-\sqrt{1+8\cos\rho}/3)$, where r is the curvature radius. This provides a way of computing a upper bound on the Hausdorff error for the entire surface. Appendix B contains the proof.

Instead of writing the error in terms of a user-specified ρ, we can let the user fix the maximum allowed Hausdorff error ε, and make ρ vary as a function of ε and r: $\rho = \cos^{-1}(9/8(1-\varepsilon/r)^2 - 1)$. Our algorithm will then reconstruct the surface with guaranteed quality.

Another feature of our algorithm is that it is eminently local. Only a small neighborhood of points is necessary to perform the MLS projection, and only a small number of triangles are necessary to check against possible self-intersections. This means that domain subdivision is a very attractive alternative for parallelization. A recent avenue of research is the use of *streaming* models, where only a small, bounded portion of the entire model is stored in memory at any given time. It seems possible to adapt our triangulation technique to use streaming techniques, so that huge meshes can be handled by appropriate out-of-core extensions.

5.1. Limitations

Our current implementation naively computes the curvature field by computing an MLS projection for every input point. Typically, half the time is spent with the MLS projections at this precomputation step. If we could determine cheaper to compute, conservative bounds on the curvature, we could forego this expensive precomputation. Our approach relies heavily on MLS surfaces. MLS surfaces provide easy access to extensive curvature information, and thus provide most of the properties in our algorithm. At the same time, the MLS formulation generates C^∞ surfaces, and this comes at a price. For example, our algorithm cannot handle sharp features, because they do not actually exist on MLS surfaces. A recent MLS surface definition [FCOS05] accounts for sharp features, but since the curvature is then only piecewise continuous, our algorithm would have to be changed. Specifically, the vertex prediction operator would have to change to account for unsmoothness in the surface. One interesting alternative would be to seed the triangulation with fronts that started on the edges, and to proceed inward, avoiding large changes in the code. Another limitation is that we do not make use of the full curvature tensor. This may result, for coarser triangulations, in edges misaligned with the principal curvatures, and visual artifacts. Additionally, there are very few theoretical results on MLS surfaces that are usable in practice. And thus it should be pointed out that our guidance field samples the surface densely, however, there is no guarantee that the sampled curvatures contain the local critical values, and so the curvature field may not be strictly an upper bound on the curvature. Since we are interested in densely sampled point sets, we can expect the field to be very close to it.

6. Summary

We presented a novel algorithm to directly triangulate Point Set Surfaces. Our algorithm produces high-quality triangulations with bounded error, even for noisy input data, and is capable of generating meshes with user-specifiable approximation errors.

We would like to explore the informed front triangulation technique introduced here in other contexts. Storing the full curvature tensor is also promising for anisotropic meshing. Finally, an important extension of this algorithm would be the proper treatment of sharp edges. This would increase significantly the types of surfaces that can be triangulated by our technique.

Acknowledgments

The bunny model was created from the range scans available at the Stanford 3D Scanning Repository. The Igea head scan is courtesy of CyberWare. We thank Tamal Dey for the cocone implementation, and Nina Amenta for the Power Crust implementation. We thank Steve Callahan for help with the illustrations and proofreading, and Louis Bavoil for help with surface simplification. The relation of our querying technique to Lipschitz continuity was pointed out by Herbert Edelsbrunner. This work was partially supported by the DOE under the VIEWS program and the MICS office, the National Science Foundation under grants CCF-0401498, EIA-0323604, and OISE-0405402, and a University of Utah Seed Grant.

References

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *IEEE Visualization 2001* (2001), pp. 21–28. 1, 2, 3, 4

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi-based surface reconstruction algorithm. *Proceedings of SIGGRAPH 98* (1998), 415–422. 2

[ACDL00] AMENTA N., CHOI S., DEY T., LEEKHA N.: A simple algorithm for homeomorphic surface reconstruction. In *16th ACM Symposium on Computational Geometry* (2000), pp. 213–222. 2

[ACK01] AMENTA N., CHOI S., KOLLURI R.: The power crust. In *Sixth ACM Symposium on Solid Modeling and Applications* (2001), pp. 249–260. 2

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 349–359. 1, 3

[BO03] BOISONNAT J.-D., OUDOT S.: Provably good surface sampling and approximation. In *Proceedings of Symposium on Geometry Processing 2003* (2003), pp. 9–19. 4

[CG04] CAZALS F., GIESEN J.: *Delaunay Triangulation based Surface Reconstruction: Ideas and Algorithms*. Tech. Rep. 5393, INRIA, 2004. 2

[CL96] CURLESS B., LEVOY M.: A volumetric method for building complex models from range images. In *SIGGRAPH 1996* (1996), pp. 303–312. 2

[CS04] CHENG H.-L., SHI X.: Guaranteed quality triangulation of molecular skin surfaces. In *Proceedings of IEEE Visualization 2004* (2004), pp. 481–488. 3

[FCOS05] FLEISHMAN S., COHEN-OR D., SILVA C.: Robust moving least squares fitting with sharp features. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2005)* (2005). 8

[FH04] FLOATER M., HORMANN K.: *Advances in Multiresolution for Geometric Modelling*. Springer-Verlag, 2004, ch. Surface Parameterization: a Tutorial and Survey, pp. 157–186. 3

[GKS00] GOPI M., KRISHNAN S., SILVA C. T.: Surface reconstruction based on lower dimensional localized delaunay triangulation. *Computer Graphics Forum 19*, 3 (2000). 2

[GSS99] GUSKOV I., SWELDENS W., SCHRÖDER P.: Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99* (1999), 325–334. 1

[HDD*92] HOPPE H., DEROSE T., DUCHAMP T., MCDONALD J., STUETZLE W.: Surface reconstruction from unorganized points. *Computer Graphics 26*, 2 (1992), 71–78. 2

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of hermite data. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2002) 21*, 3 (2002), 339–346. 2

[KS01] KARKANIS T., STEWART A.: Curvature-dependent triangulation of implicit surfaces. *IEEE Computer Graphics and Applications 21*, 2 (2001), 60–69. 2, 3

[KSO04] KOLLURI R., SHEWCHUK J. R., O'BRIEN J. F.: Spectral surface reconstruction from noisy point clouds. In *Symposium on Geometry Processing* (2004), ACM Press, pp. 11–21. 2

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high resolution 3d surface reconstruction algorithm. In *SIGGRAPH 1987* (1987), pp. 163–169. 2

[Lev03] LEVIN D.: Mesh-independent surface interpolation. In *Geometric Modeling for Scientific Visualization*, Brunnett G., Hamann B.,, Mueller H., (Eds.). Spinger-Verlag, 2003. 1, 4, 10

[Lin01] LINSEN L.: *Point Cloud Representation*. Tech. Rep. 3, Fakultaet fuer Informatik, Universitaet Karlsruhe, 2001. 7

[LPC*00] LEVOY M., PULLI K., CURLESS B., RUSINKIEWICZ S., KOLLER D., PEREIRA L., GINZTON M., ANDERSON S., DAVIS J., GINSBERG J., SHADE J., FULK D.: The Digital Michelangelo Project: 3D Scanning of Large Statues. *Proceedings of SIGGRAPH 2000* (2000), 131–144. 1

[LSS*98] LEE A., SWELDENS W., SCHRÖDER P., COWSAR L., DOBKIN D.: Maps: Multiresolution adaptive parameterization of surfaces. *Proceedings of SIGGRAPH 98* (1998), 95–104. 1

[OBA*03] OHTAKE Y., BELYAEV A., ALEXA M., TURK G., SEIDEL H.-P.: Multi-level partition of unity implicits. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22*, 3 (2003), 463–470. 3

[PKKG03] PAULY M., KEISER R., KOBBELT L. P., GROSS M.: Shape modeling with point-sampled geometry. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2003) 22*, 3 (2003), 641–650. 10

[SAG03] SURAZHSKY V., ALLIEZ P., GOTSMAN C.: Isotropic remeshing of surfaces: A local parametrization approach. In *Proceedings of International Meshing Roundtable 2003* (2003). 3

[TO02] TURK G., O'BRIEN J. F.: Modelling with implicit surfaces that interpolate. *ACM Transactions on Graphics 21*, 4 (2002), 855–873. 3

Appendix A: The MLS projection

For the sake of completeness, we briefly review the MLS projection [Lev03]. The main idea is to define a projection procedure that takes a point $\mathbf{r}$ in 3D space and project it to the surface $\mathbf{r}' = \mathscr{P}(\mathbf{r})$. The surface is then defined as the set of points that project to themselves. This is achieved by first fitting a reference plane $H = (\mathbf{q}, \mathbf{n})$ to the neighborhood of $\mathbf{r}$; $\mathbf{q}$ is a point on the plane and $\mathbf{n}$ is the (unit length) normal to the plane. The reference plane is defined as one that minimizes the following energy function:

$$\min_q \sum_i \langle \mathbf{n}, \mathbf{p}_i - \mathbf{q} \rangle^2 \theta(||\mathbf{p}_i - \mathbf{q}||), \tag{3}$$

where $\mathbf{p}_i$ is the ith neighbor of $\mathbf{r}$, $\mathbf{n} = \frac{\mathbf{r}-\mathbf{q}}{||\mathbf{r}-\mathbf{q}||}$ and $\theta(\cdot)$ is a smooth monotonically decreasing function. Equation 3 is minimized using gradient descent. The second step of the projection procedure is to locally fit a bivariate polynomial of low degree g over H. Let (u_i, v_i, w_i) be the coordinates of $\mathbf{p}_i$ in a coordinate system that is defined by H, then g is the polynomial that minimizes the following weighted least squares error:

$$\sum_i (g(u_i, v_i) - w_i)^2 \theta(||\mathbf{p}_i - \mathbf{q}||). \tag{4}$$

The projection is then defined as $\mathscr{P}(r) = \mathbf{q} + g(0,0)\mathbf{n}$.

The weight function has a scale parameter h, that determines the amount of smoothing that is applied to the data. We determine the local feature size of each point and thus set h as a function of the local feature size as in Pauly et al. [PKKG03]. That is the local feature size of a point $L(\mathbf{x})$ is defined as the radius of the k nearest neighbors to $\mathbf{x}$. The scale is then defined as $h(\mathbf{x}) = \tau \cdot weighted\,average(L(Nbhd(\mathbf{x})))$, where τ is the only parameter that is defined by the user which determines the amount of smoothing or denoising to apply.

In Equation 3 we look for a local minimum on the surface that is closest to $\mathbf{r}$. In order to robustly find the expected local minimum, we must find a good initial value for the optimization, i.e. we need to define $\mathbf{q}_0$. Our heuristic for finding such an initial value has two terms, one for points that are close to the surface $\mathbf{q}_{near}$ and another that is for far points $\mathbf{q}_{far}$. Let $\mathbf{c}_0$ be the centroid of the neighborhood of $\mathbf{r}$ and let $\mathbf{n}_0$ be the normalized eigenvector that corresponds to the smallest eigenvalue of the covariance matrix of the neighborhood of $\mathbf{r}$, then $\mathbf{q}_{near}$ is set to the projection of $\mathbf{r}$ on the plane that is defined by $\mathbf{c}_0$ and $\mathbf{n}_0$, i.e. $\mathbf{q}_{near} = \mathbf{r} - \langle \mathbf{n}_0, \mathbf{r} - \mathbf{c}_0 \rangle \mathbf{n}_0$. We set $\mathbf{q}_{far} = \mathbf{c}_0$, and finally $\mathbf{q}_0 = \alpha \cdot \mathbf{q}_{near} + (1-\alpha) \cdot \mathbf{q}_{far}$, where α is set to one for $\mathbf{r} = \mathbf{c}_0$ and zero when $||\mathbf{r} - \mathbf{c}_0|| = L\mathbf{c}_0$.

Appendix B: Bounded Hausdorff Distance

Theorem B.1 Let $e_{\max}$ be the length of the largest edge of the triangulation. Then the Hausdorff distance between the MLS surface and the triangulated mesh is bounded from above by $e_{\max} \frac{3-\sqrt{1+8\cos\rho}}{3\sqrt{2-2\cos\rho}}$.

Proof The Hausdorff distance between two surfaces $A, B \subset R^3$ is defined as $\varepsilon(A,B) = \max_{a \in A}(\min_{b \in B} ||a-b||)$. Given a triangle T on the mesh reconstruction and the patch of surface "over" the triangle, we first locally approximate the MLS as a sphere S whose radius is the radius of curvature of the patch. This is valid as long as ρ is small. Then, since the sphere touches the triangle at the vertices, the points that give the maximum distance to the MLS surface (and the Hausdorff distance $\varepsilon(S,T)$, assuming S is restricted to the portion "over" the triangle) are the barycenter of the triangle and its closest point on the osculating sphere S. We know ρ gives the angle subtended by the edges on the grand circles of S. If S has radius r, so do the grand circles. Then, the edges of the triangle have length at most $2r\sin(\rho/2)$, and the distance from T to the center of S is given by $r - \varepsilon(S,T)$. The barycenter of T, one of its vertices and the center of S form a right triangle with sides $r - \varepsilon(S,T)$, $4/3r\sin(\rho/2)$ and hypotenuse r. Solving this gives $\varepsilon(S,T) = r \cdot (1 - \sqrt{(1+8\cos\rho)}/3)$.
The Hausdorff distance for each patch is proportional to the radius of the osculating sphere. From that, it immediately follows that the Hausdorff error for the entire surface is given by the sphere-triangle Hausdorff distance of the largest sphere. Substituting r for the corresponding largest triangle edge length and simplifying gives the expression stated in the theorem. □