High Quality, Curvature Dependent Triangulation of Implicit Surfaces

Tasso Karkanis[1]

A. James Stewart[1,2]

[1] Dynamic Graphics Project
Department of Computer Science
University of Toronto

[2] iMAGIS–GRAVIR/IMAG
Grenoble, France

Abstract

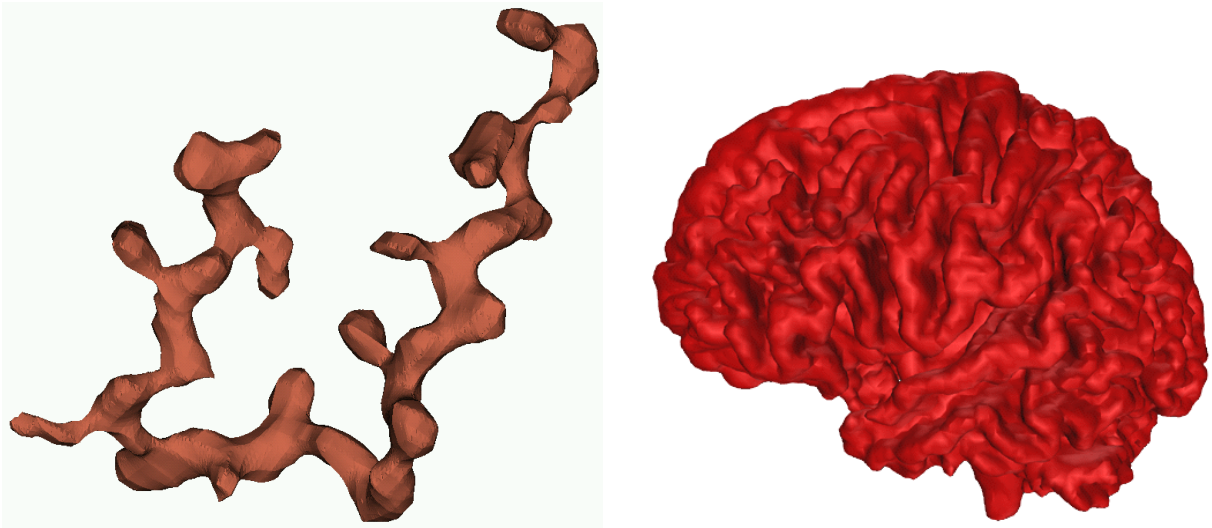
The paper describes an algorithm to generate a triangulation of an implicit surface. The generated triangles are close to equilateral and the triangle edge lengths vary with local surface curvature (the user may choose the ratio of edge length to surface curvature). The output of this algorithm is useful for applications that require high quality triangulations, such as medical imaging, molecular modeling, computer aided design, and finite element analysis.

Triangles are "grown" on the surface outward from a seed triangle. After the growing stops, the remaining gap in the triangulation is filled by a set of heuristics. Experiments show that the algorithm consistently builds high quality triangulations which compare favourably to those produced by cell–based and particle–based algorithms.

Key words: Implicit surface, adaptive triangulation

1 Introduction

Implicit surfaces appear in many applications including medical imaging, molecular modeling, computer aided design, computer graphics, and finite element analysis. An implicit surface is defined by a field function, $f(x, y, z)$, which assigns a scalar value to each point in space. The implicit surface consists of those points at which the field function takes on a specific value, c. In other words the surface is $\{(x, y, z) \mid f(x, y, z) = c\}$.



(a) surface of a snake venom molecule (b) surface of a human cortex

Figure 1: Examples of implicit surfaces. The blockiness of the molecule is due to the grid upon which the electron densities were sampled. Sincere thanks to Dr. Paul Bourke of the Swinburne University of Technology for the image of the human cortex, and to Dr. James Rini of the University of Toronto for the snake venom data.

In molecular modeling, for example, the electron density takes on a specific value at the Van der Waals surface of the molecule (e.g. [15]). This surface can give a good idea of the structure of an unknown molecule whose electron density has been sampled. In medical imaging, for example, a computed tomography (CT) scan produces a three dimensional array of sampled density values from a patient. Anatomical features are distinguished by their different densities and can be represented with implicit surfaces. In computer aided design, for example, an offset surface can provide a smooth blend between two components that meet at a sharp angle.

Despite their many advantages, implicit surfaces are difficult to render efficiently. Today's real–time graphics systems are heavily optimized for rendering triangles, so an implicit surface should be converted to a mesh of triangles before being rendered.

This paper describes an algorithm to polygonalize an implicit surface. The algorithm generates a mesh of close–to–equilateral triangles with sizes dependent upon the local surface curvature. The implicit surface is assumed to be connected and G^1 smooth. The algorithm requires an evaluator for the implicit function defined at all points in space, an evaluator for the function gradient defined at points near the surface, and a bounding box around the

surface.

The output of the algorithm is good for applications that require a "well behaved" triangulation, such as rendering systems and finite element PDE solvers. For rendering systems, curvature–dependent triangle sizing results in a accurate surface and silhouette (see the middle of the peanut in Figures 2(a) and 2(b)), while at the same time minimizing the number of triangles, resulting in faster rendering. For finite element methods, the near–equilateral triangles reduce the instability of the solution, and the curvature dependence keeps discretization error low while minimizing the number of triangle elements, leading to a faster solution.

2 Related Work

The new algorithm falls into the class of "continuation methods" which operate by incrementally extending a polygonization across the implicit surface. Following Bloomenthal [4], these methods are divided into "piecewise linear" and "predictor–corrector" classes.

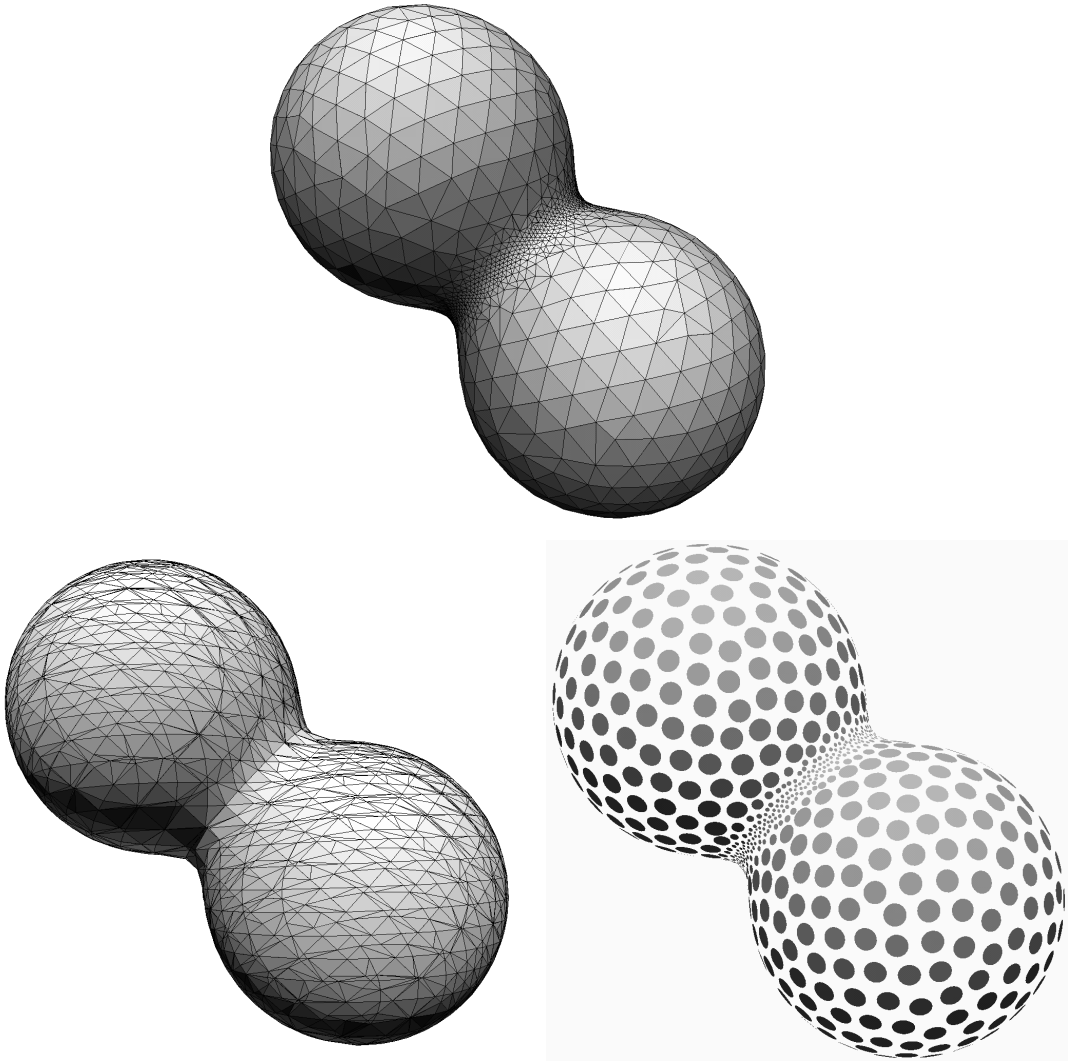
Piecewise linear continuation methods divide space into discrete cells (typically cubes or tetrahedra) and polygonize each cell individually [1, 3, 26]. A cell is considered for polygonization only if it is adjacent to another cell that already contains part of the surface. The polygonization of a cell is determined from a lookup table which is indexed by the signs of the implicit function at the cell's vertices [26]. Cells may be adaptively sized according to local surface features [3, 17].

These methods are very fast, but can result in polygonizations that contain high–aspect–ratio triangles and tiny polygons, such as shown in Figure 2(b). There are a number of post–processing methods that will eliminate these undesirable features (e.g. [7, 19, 22]).

Unlike these methods, the new algorithm does not require a separate post–processing step to produce a high–quality triangulation: When each triangle is created, it is sized according to the local surface curvature and is made to be close–to–equilateral, as shown in Figure 2(a).

Adaptive continuation methods [3, 17] devote considerable effort in joining polygons that lie in adjacent cells of different resolutions. The new algorithm avoids this difficulty by allowing the triangle sizes to vary continuously as the triangulation is extended.

The new algorithm is, however, slower than the piecewise linear methods described above. It spends considerable time computing local surface curvature in order to generate triangles of the appropriate size. (The curvature calculation requires many calls to the implicit function



(b) Cell–Based Algorithm (c) Particle–Based Algorithm

Figure 2: Shown are triangulations of a peanut produced by (a) the algorithm of this paper and (b) a cell–based algorithm [3]. The particle–based method (c) produces the same density distribution of vertices as the new algorithm.

evaluator, since we assume that the second derivative is not directly available.) This time is not spent with the piecewise linear methods, most of which only evaluate the implicit function at cell vertices.

Predictor–corrector continuation methods extend the polygonization by generating new vertices on the border of the current polygonization. These vertices initially lie in the tangent plane at the border (the predicted position) and are subsequently settled onto the implicit surface (the corrected position). New polygons are added to join the vertex to the current polygonization [18]. Alternatively, a disk centred on a boundary point may be created, projected onto the surface, and merged with the current triangulation [11, 12].

The new algorithm is a predictor–corrector method which successfully addresses the difficulties encountered with two dimensional surfaces. In particular, the new algorithm avoids overlapping triangles, which can occur if two separate "branches" of the polygonization converge as they are being extended. The new algorithm also fills in the narrow "gap" between adjacent branches with well–shaped triangles that are sized according to the local surface curvature.

Another approach [13] creates triangles of roughly uniform shape and size which are locally Delaunay[1]. This approach handles open two–manifolds, while the new algorithm assumes that the surface is closed. However, this approach does not size its triangles according to the local curvature, which may lead to rendering artifacts, particularly on the silhouette.

The new algorithm is most similar to an algorithm of Hartmann [10] which iteratively performs one of two operations: Either triangles are added to the current mesh boundary (like our growing phase) or the closest pair of non–adjacent vertices on the mesh boundary are joined (like our bridging operation). Both algorithms produce close–to–equilateral triangles, but Hartmann's algorithm does not appear to make triangle size dependent upon surface curvature. The new algorithm uses a large set of heuristics (Section 3.3) which help to produce better triangles in those areas where the triangle size changes rapidly.

Full–lattice methods assume that a full lattice of measured function values is available [14, 26]. These methods correctly treat surfaces of multiple connected components, while the new algorithm (and other continuation methods) require a "seed polygon" in each connected component.

Particle–based methods settle oriented particles to the implicit surface and maintain

[1]A triangle is **locally Delaunay** if its smallest circumscribing sphere does not contain any other point of the triangulation that has the same surface orientation.

them on the surface during editing operations [7, 21, 25, 8]. Using a set of particles requires that the point set be triangulated in a postprocess using, say, a local Delaunay triangulation or alpha shapes [9].

The triangulations produced by particle systems have the "Delaunay–like" property that the triangles are close to equilateral, and particle–based methods can be modified to adapt the particle density to local surface curvature. These methods have vertex distributions that are qualitatively similar to those of the new algorithm. However, the particle–based algorithm implemented for comparison (described in Section 4) took much longer to execute than the new algorithm, likely because every particle continues to be processed as long as at least one particle has not reached equilibrium.

The **pre–tessellation method** [6] considers implicit surfaces that are defined as a combination of certain primitives surfaces, each of which has a known surface tesselation. Each primitive's surface tesselation is terminated where it enters a blending area with another primitive, then the corresponding boundary curves of adjacent primitives are identified, and finally a mesh is constructed in the gap between corresponding curves.

Like the pre–tessellation method, the new algorithm builds a mesh over part of the surface and subsequently fills the gap which is left uncovered. However, the new algorithm handles arbitrary implicit functions, which makes the structure of the gap much more complicated. With the pre–tessellation method, the gap consists of relatively straight sections that separate pairs of primitives, and each section can be meshed independently. The new algorithm must handle general gaps, as shown in Figure 3. It is likely, however, that the pre–tessellation method is much faster for the surfaces that it handles.

Gap filling is a problem also studied in the field of computational geometry [2] and is somewhat related to the construction of meshes from range data [23].

The **shrinkwrap method** [5, 24] builds a surrounding polygonized mesh and incrementally shrinks this mesh onto the implicit surface: The mesh vertices are moved along the integral lines of the implicit function and the mesh is refined as necessary. The new algorithm polygonizes some surfaces that, we believe, cannot be polygonized with the shrinkwrap method (e.g. the "cube" in Figure 8).

3 The Algorithm

The algorithm operates in two phases: In the **growing phase** a seed triangle, which forms the initial polygonization, is computed. The polygonization is extended by incrementally growing triangles from its edges. Each new triangle is sized according to the local curvature, and a triangle is *not* added if it would come too close to an already–existing triangle. At the end of this phase, the polygonization is a connected region with long, narrow gaps between its branches.

In the **filling phase**, the gap is subdivided into small pieces by finding "bridges" that cross the gap. These bridges are good edges in the final triangulation. They separate the gap into smaller, more manageable pieces. Each smaller piece is triangulated with a set of heuristics.

3.1 Preliminaries

Some definitions are given before the algorithm is discussed in detail:

Triangles are **sized** according to a user–defined parameter, ρ, which is the desired ratio of triangle edge length to local radius of curvature.

Two triangles are said to **overlap** if part of one, when projected onto the plane of the other, lies inside the other. For non–overlapping triangles, the **distance** between them is the minimum distance between a vertex of one and an edge of the other.

The **radius of curvature** at a point, x, is estimated by computing the radius of curvature of several geodesics that pass through x, and taking the minimum. Geodesics are assumed to locally lie in a plane through x and the surface normal at x, denoted n_x. To compute one such radius of curvature, the normal n_y at a point y close to x is first computed. Let θ be the angle between n_x and n_y, and let d be the distance between x and y. Then the radius of curvature is estimated as

$$R(x) = \frac{d}{2\sin(\frac{\theta}{2})}.$$

3.2 The Growing Phase

The bounding volume is first sampled until one point inside the surface and one point outside the surface are found. Repeated bisection then finds a point that lies on the surface. This point locates a **seed triangle** whose vertices lie on the implicit surface, and whose edge lengths are a fraction ρ of the local surface curvature.

The edges of the seed triangle are placed into an **active edge list**. Growing proceeds by iterating over this list and applying to each edge the two operations described below: "Isosceles Triangle Growing" and "Ear Cutting." The growing phase terminates when no operation can be applied to any edge in the list. Figure 3 shows some sample triangulations at the end of the growing phase.

Isosceles Triangle Growing

Given a candidate active edge, (u, v), a new point, p, is placed in the tangent plane of the active edge (to the side of the edge outside the current tiling) such that u, v, and p form an equilateral triangle. Point p is settled to the implicit surface. The vertices u, v, and p are used to estimate R, the radius of curvature. Then p is again placed in the tangent plane of the active edge, but now positioned such that the lengths of the new edges, (u, p) and (v, p), are ρR. Finally, p is again settled to the surface. This results in an almost–isosceles triangle with at least two edges whose lengths are appropriate for the local surface curvature.

Two tests are performed: The first test checks that each new edge makes an angle of at least 45 degrees with its neighbour in the old mesh. This ensures that later triangles will be close to equilateral.

The second test checks that the new triangle, T, does not approach existing triangles too closely. This ensures that the gap produced by the growing phase will not be too narrow to triangulate in the subsequent filling phase.

For the second test, all triangles within a radius r of T's centroid are enumerated (r is defined in the Appendix). This enumeration is performed efficiently by traversing an octtree [16] which stores all vertices of the current mesh. If any one of those triangles, say T', is closer to T than one–half the length of the longest edge in T and T', then the new triangle is rejected.

As a result of this second test, the gap is typically "just wide enough" to contain a triangle of the size appropriate to the local curvature. Note that the isosceles triangle growing operation need be attempted at most once on each active edge.

Ear Cutting

Ear cutting is applied when two adjacent active edges have an external angle smaller than 70 degrees. The three vertices on these two edges are used to define a new triangle. Note that, due to the second test above, it is not possible that a triangle from another branch

intrudes into the area between these three vertices. The value of 70 degrees was chosen to allow somewhat non–equilateral triangles. For angles larger than 70 degrees, the heuristics of the next section are more likely to produce a good local triangulation.

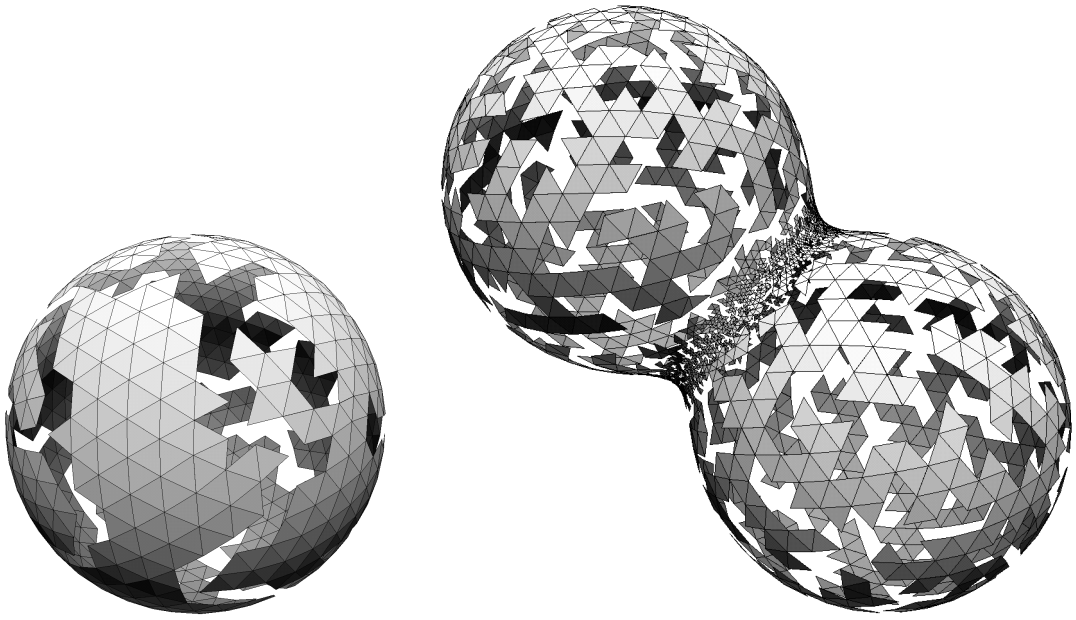


Figure 3: A sphere and peanut after the growing phase.

3.3 The Filling Phase

Upon termination of the growing phase, a connected polygonal gap remains to be triangulated. The filling phase starts by associating each vertex of the gap with its closest neighbour vertex. The closest neighbour relationship is carefully defined so as to make the line segment between a vertex and its closest neighbour a good segment in the triangulation.

Closest Neighbours and Bridges

For a vertex v on the gap, let N_v be the normal to the implicit surface at v, let v_1 be the vertex preceding v on the boundary of the gap, and let v_2 be the vertex succeeding v (vertices are ordered counter–clockwise around the gap, as seen from above the surface).

Every vertex v on the gap has two **associated planes**, P_v^1 and P_v^2. P_v^1 embeds v, v_1, and N_v, and is oriented such that the normal to the plane points into the gap. P_v^2 is defined similarly except that it embeds v_2 instead of v_1 (see Figure 4).

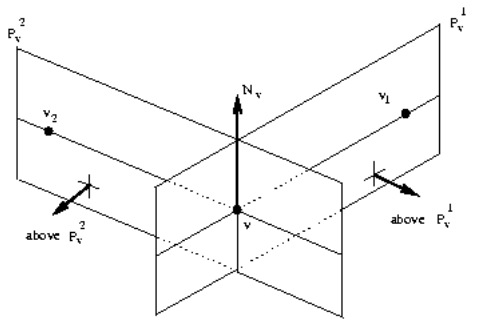


Figure 4: A concave vertex on the gap and its associated planes.

A vertex u is said to be **above** P_v^1 if it is contained in the half–space into which the normal of P_v^1 points, and is farther from P_v^1 than one–tenth the distance between v and v_1. The definition of "u is above P_v^2" is similar. The value of one–tenth was chosen to avoid extremely non–equilateral triangles, which would occur if, say, the triangle $v\ v_1\ w$ was formed with a vertex w that was closer to P_v^1 than one-tenth the distance between v and v_1.

A vertex v on the gap is **convex** if the interior angle (i.e. inside the gap polygon) that it makes with v_1 and v_2 is less than π and **concave** otherwise.

Every vertex v has an associated set of neighbours which consists of other vertices on the gap. If v is convex its **neighbours** are those vertices that are above P_v^1 and P_v^2, and if it is concave its neighbours are those that are above at least one of P_v^1 and P_v^2.

The **closest neighbour** v_c to a vertex v is the neighbour of v that has the shortest Euclidean distance to v. This relationship is denoted $v \rightarrow v_c$. Note that the closest neighbour may not exist, and that this relationship is not necessarily commutative.

A pair of vertices v_1 and v_2 is said to be **bridge** if $v_1 \rightarrow v_2$ and $v_2 \rightarrow v_1$. A bridge is denoted $v_1 \leftrightarrow v_2$. See Figure 5.

To compute the nearest neighbours, all gap vertices are inserted into an octtree. For each vertex v, a radius r is initialized to twice the distance between v and its predecessor. The octtree is queried for vertices within distance r of v. If this set contains neighbours, the nearest of these is selected. Otherwise, the radius is doubled and the procedure is repeated.

Filling the Gap

A **gap** is formally defined as a simple polygon (on the implicit surface) for which each vertex stores its nearest neighbour as defined above. An initial gap is created from the polygon produced by the growing phase. This initial gap is placed into an otherwise–empty **gap**

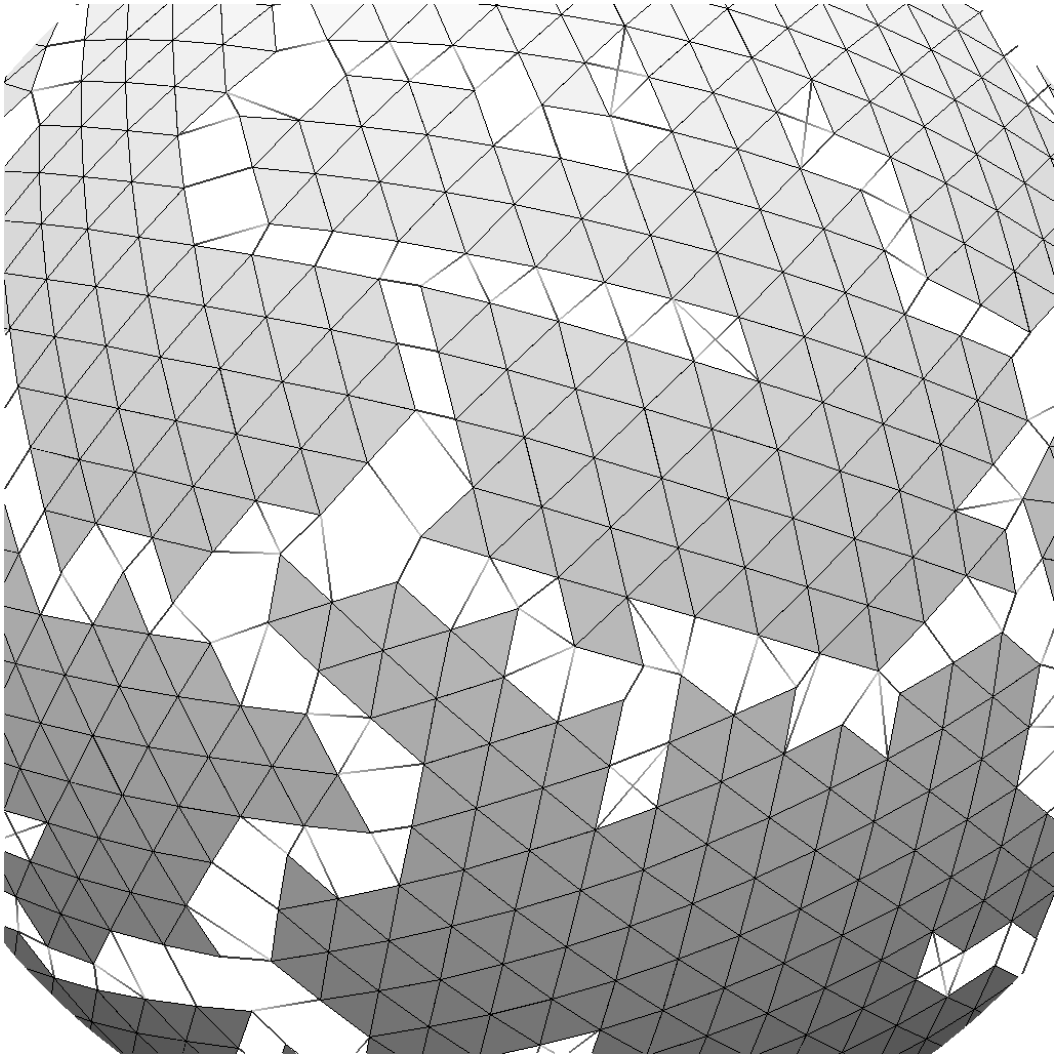


Figure 5: The gap with the closest neighbour relationships on a sphere. The thick uniformly coloured lines between two vertices u and v indicate $u \leftrightarrow v$. A dark line that fades to a lighter colour from a vertex u to a vertex v indicates $u \rightarrow v$.

queue.

The algorithm starts by removing the first gap from the queue. Heuristics are applied to add triangles to this gap: If a heuristic causes a gap to be split into several disjoint gaps, these gaps are placed at the tail of the queue. If the heuristic modifies the gap without filling it, the modified gap is returned to the head of the queue. Otherwise, the gap has been filled and the size of the queue decreases. (It is never that case that no heuristic applies.) The algorithm iterates until the queue becomes empty, at which point the surface is completely triangulated.

Note that when a gap is modified, some of its nearest neighbour relations are changed. It is relatively easy to identify these changes, so they will not be described below. For a particular gap, G, the following heuristics are attempted in the order presented. After one heuristic is successfully applied, G is either completely filled or is returned to the queue.

1. *Small Polygon Filling:* If G contains three vertices then one triangle is created. If G contains four vertices then two triangles are created. If the four vertices are convex, the diagonal between the two triangles can be chosen to maximize the minimum of the "qualities" of the two triangles, where the quality of a triangle is measured as the ratio of the smallest to largest edge lengths. In any case, G has been filled and is not returned to the queue.

2. *Subdivision on Bridges:* All bridges of G are collected. A bridge $u \leftrightarrow v$ is then discarded if vertices u and v are not separated on the boundary of the gap polygon by at least two other vertices. The remaining bridges subdivide the original gap into smaller gaps, which are placed at the tail of the queue. If a bridge $u \leftrightarrow v$ is longer than 150% of the optimal local edge length, the bridge is split by adding a vertex at the midpoint of u and v, and settling this point to the surface. The value of 150% was chosen because it is halfway between having one edge of the correct length (100%) and two edges of the correct length (200%).

3. *X Filling:* An ***X*–sequence** (see Figure 6) is a sequence $S = \{v_1, v_2, v_3, v_4\}$ of adjacent vertices with the properties (1) $v_2 \rightarrow v_4$, (2) $v_3 \rightarrow v_1$, and (3) neither v_2 nor v_3 is the closest neighbour of any vertex not in S.

   If G contains an X–sequence, $\{v_1, v_2, v_3, v_4\}$, then the distance between v_1 and v_4 is checked against the local optimal edge length. If the distance between the vertices is

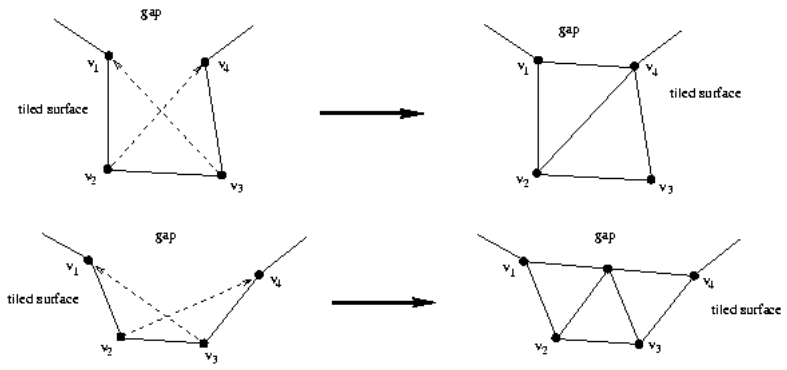


Figure 6: Gap X's being filled with and without the addition of another vertex. The nearest neighbour relationships are indicated with dashed lines.

more than 150% of optimal, a vertex v is generated on the midpoint between v_1 and v_4. The X is then filled with either two or three triangles.

4. *Ear Filling:* An **ear** is a sequence $\{v_1, v_2, v_3\}$ of adjacent vertices such that $v_1 \leftrightarrow v_3$ and no vertex of G has v_2 as its closest neighbour. If G contains an ear, $\{v_1, v_2, v_3\}$, then the distance between v_1 and v_3 is split if it is more than 150% of the optimal local edge length. The ear is then filled with one or two triangles.

5. *Convex Polygon Filling:* If every vertex of G is convex, a new vertex is placed at the average of the vertices of G and is settled to the surface. A fan of triangles is created around the new vertex. G has been filled and is not returned to the queue.

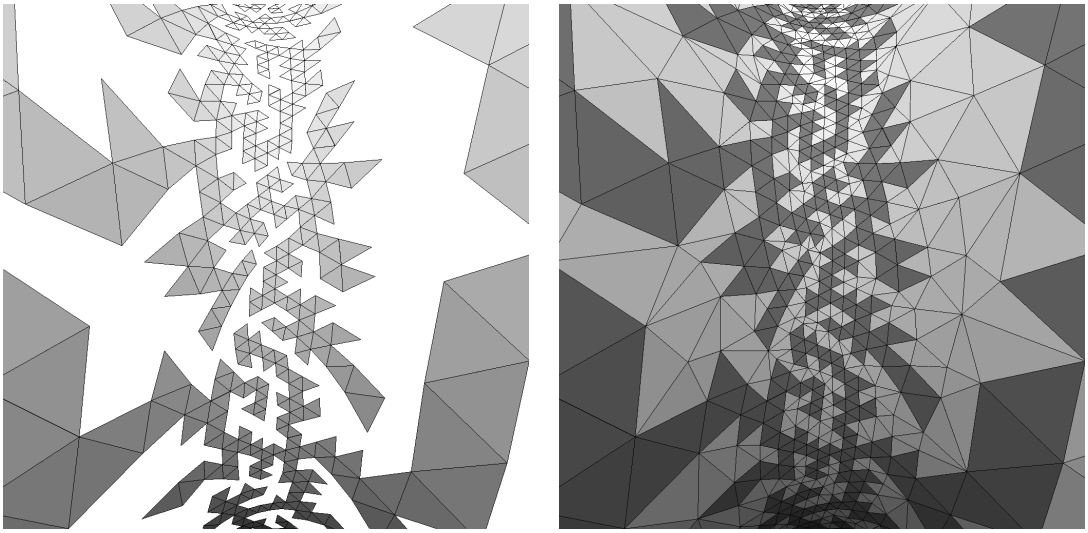
6. *Relaxed Ear Filling:* A **relaxed ear**, $\{v_1, v_2, v_3\}$, has the same properties as an ear except that either $v_1 \rightarrow v_3$ or $v_3 \rightarrow v_1$ is acceptable (whereas an "ear" has $v_1 \leftrightarrow v_3$). The relaxed ear filling procedure is otherwise the same as ear filling.

7. *Concave Vertex Bisection:* If all of the preceding heuristics fail, G must necessarily contain at least one concave vertex. The vertex v_1 that has the largest interior angle is selected from G, and a corresponding vertex v_2 in the gap is found such that the line segment (v_1, v_2) most closely bisects the interior angle at v_1. G is split into two components along this line segment. If the distance between these vertices is more than 150% of the optimal local edge length the edge is split by settling the midpoint to the surface. The two new gaps are placed at the tail of the queue.

After the gap queue becomes empty, each edge which separates a pair of adjacent triangles is

"flipped" (so that it joins the other two vertices of the two adjacent triangles) if this results in a better local triangulation. In order to avoid a cascade of flips, each edge is considered only once. Typically, between zero and one percent of edges are flipped. Figure 7 shows a triangulation after the filling phase.

An final improvement could be made by applying Laplacian smoothing (although we didn't implement this): Each vertex is moved to the geometric centre of its adjacent vertices if and only if this results in an improvement to the local triangulation.

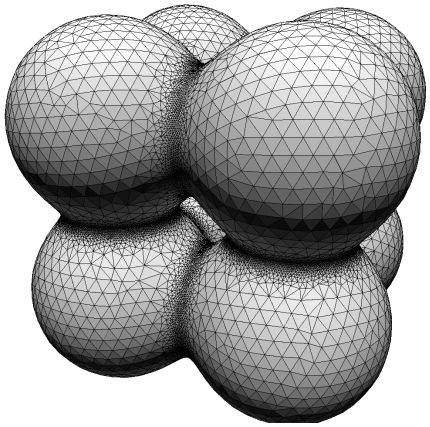


(a) after growing phase (b) after filling phase

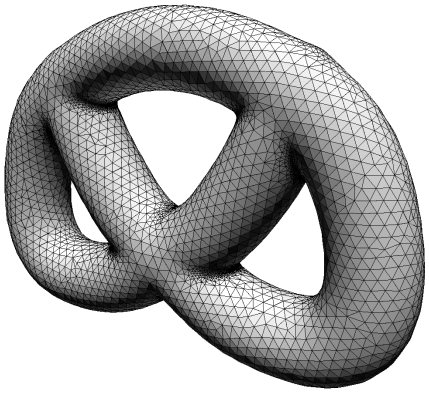
Figure 7: A closeup of the peanut middle after each phase. In (a), no more triangles can be extended into the gap without encroaching too closely to an already–existing triangle. In (b), the light triangles were produced by the filling phase. Note that one triangle pair on the right boundary of the gap has changed due to a diagonal flip which reduces the average aspect ratio.

4 Experimental Results

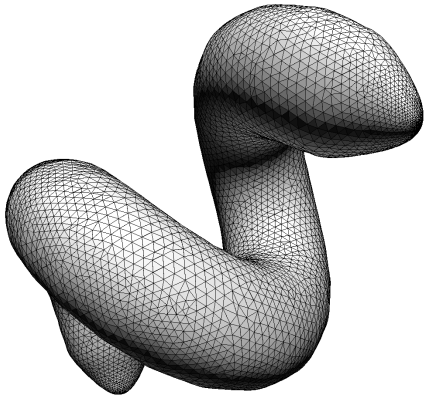
Several meshes produced by this algorithm are shown in Figure 8. Two other polygonizers were implemented for comparison: Bloomenthal's cell–based polygonizer [3] and Witkin and Heckbert's particle–based modeller [25], which was modified to make particle density proportional to the local surface curvature. The particle–based method is discussed in more depth below.



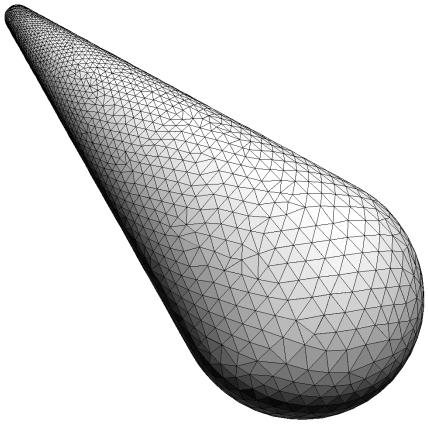
(a) Cube



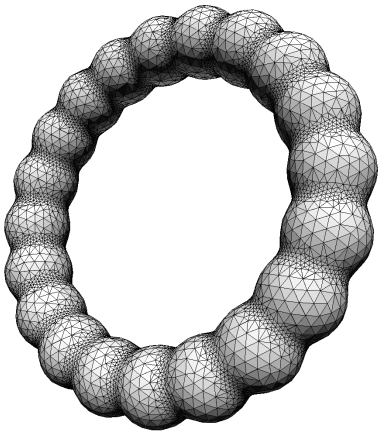
(b) Pretzel



(c) Spiral



(d) Cone



(e) Torus

Figure 8: Meshes produced by the new algorithm

Triangle Quality

The new algorithm consistently produces close–to–equilateral triangles, as shown in Figure 9. The particle–based polygonizer produced somewhat less–equilateral triangles, and the cell–based polygonizer produced fairly poor triangles.

The new algorithm also produces edges whose edge–length to surface–curvature ratios are tightly clustered around the desired ratio set by the user, as shown in Figure 10. The particle–based polygonizer had a qualitatively similar distribution of ratios.

Execution Times

Execution times are reported in Table 1, which shows: a point is settled to the surface about 5.6 times per triangle of the final mesh; between 4.8 and 9.0 implicit function evaluations are required for each settling; and between 27 and 50 function evaluations are required for each triangle of the final mesh.

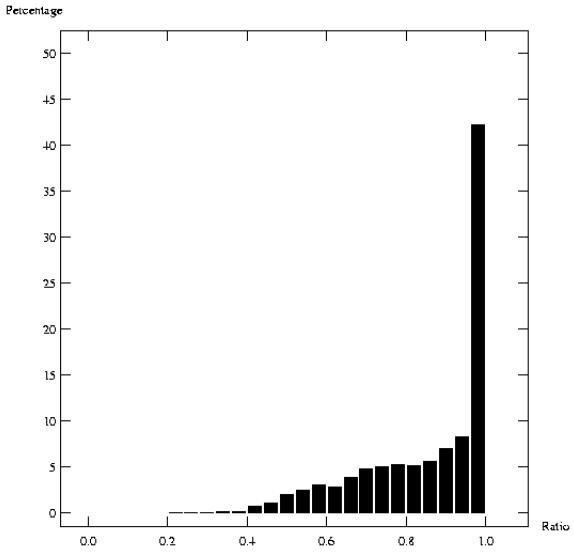
The new algorithm was typically twenty times slower than the cell–based algorithm. But it was typically five times faster than the curvature–dependent particle–based algorithm. The particle–based algorithm must process every particle with each iteration, while the new algorithm simply creates a new triangle or triangles, which thereafter remain fixed.

The new algorithm is easily fast enough for any application that builds the polygonization in an offline, preprocessing step. The large number of implicit function evaluations per triangle is principally due to curvature evaluations (70 % of all implicit function evaluations are used for this). While we could have evaluated the curvature directly for some models, this is not possible for general implicit functions represented as "black boxes," so we chose not to do so.

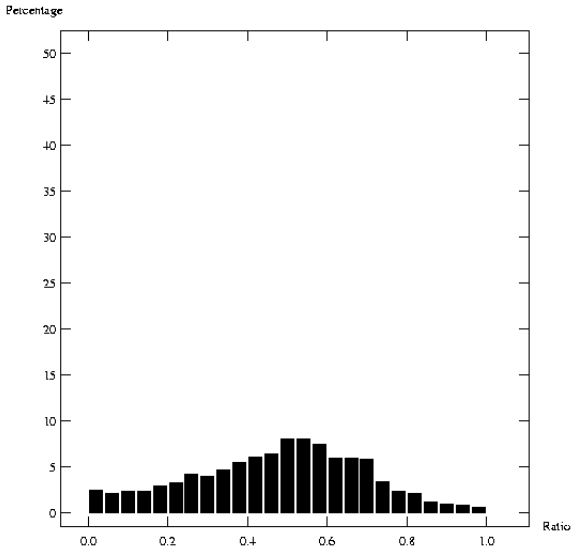
Failure Modes

It was found that accurate curvature measurement is critical to the triangle growing phase. If the curvature is not measured accurately, small triangles may be placed close to large ones and this situation will be handled badly by the filling phase, which implicity expects similar triangles sizes on opposite sides of a narrow gap.

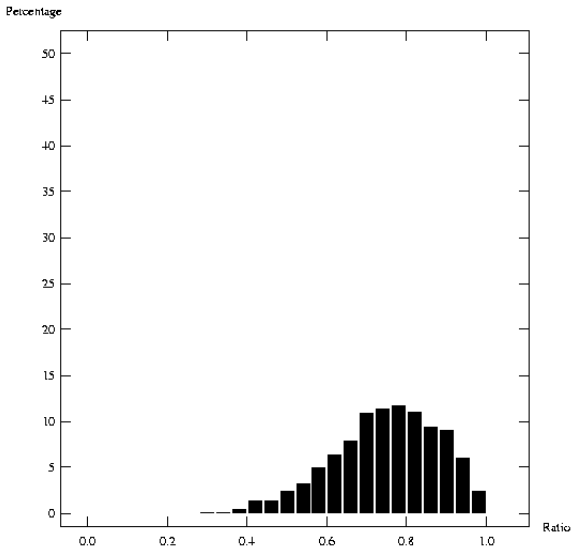
The same problem occurs if the rate of change of curvature is high or if the surface is not G^1. In this case there are almost–flat areas adjacent to highly curved areas and the growing phase produces large triangles that are close to small triangles. Again, the filling phase has



(a) Peanut with new algorithm

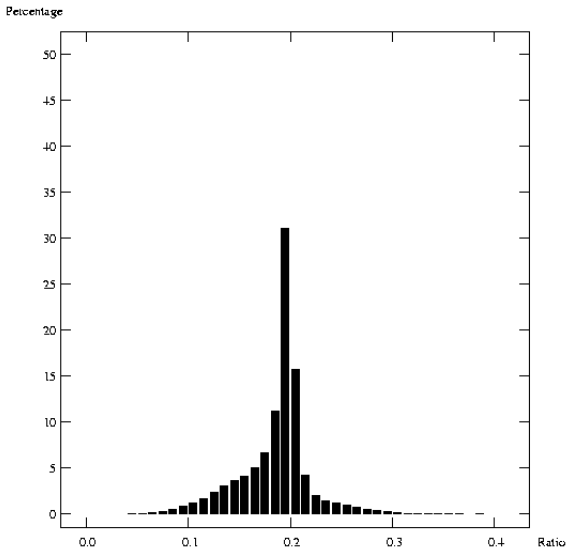


(c) Peanut with cell–based algorithm

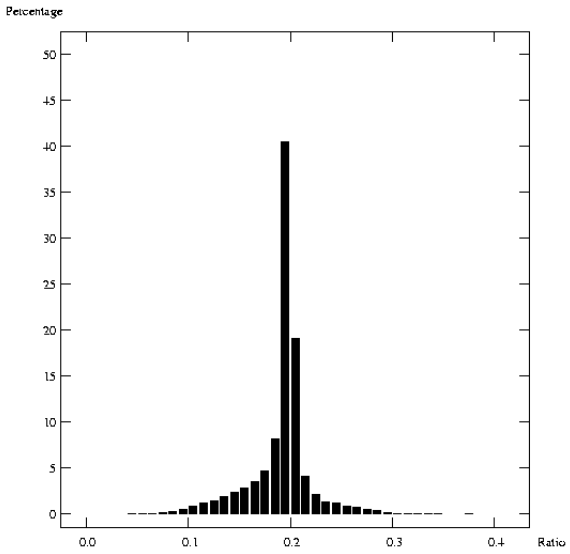


(b) Peanut with particle–based algorithm

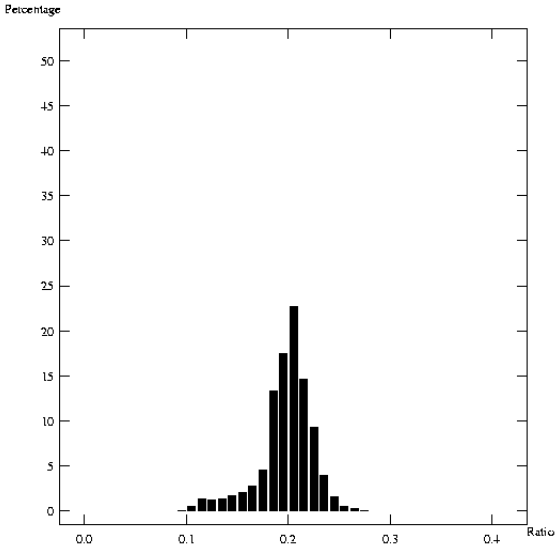
Figure 9: The distributions of the ratio of shortest edge length to longest edge length of each triangle, for the peanut model. Distributions for the other objects were qualitatively similar. A ratio of 1.0 corresponds to an equilateral triangle.



(a) Cube with new algorithm



(b) Pretzel with new algorithm



(c) Peanut with particle–based algorithm

Figure 10: The distribution of actual edge–length to surface–curvature ratios, with a desired ratio of $\rho = 0.2$ set by the user. Distributions for the other objects were qualitatively similar.

Table 1: Execution results on a 500 MHz Pentium PC. These tests used a smaller setting of ρ than that of Figures 2 and 8.

Model	Time (sec)	Triangles	Settlings per triangle	Func evals per settling	Func evals per triangle
peanut	10	6,546	5.7	6.2	35.3
spiral	102	18,072	5.6	9.0	50.3
cube	190	56,292	5.6	4.8	27.1
pretzel	587	40,924	5.7	5.8	33.3

difficuly in this situation. The pretzel and peanut were particularly stressful tests since they have areas in which the curvature changes very rapidly (see Figure 7(a) for example).

One possible fix (which we haven't yet implemented) is the following: This situation is detected when the nearest neighbour of a vertex, v, is adjacent to a triangle of much different size than a triangle adjacent to v. In this case, subdivide each large triangle into four smaller triangles by adding a vertex at the midpoint of each triangle edge and joining the new vertices (care must be taken to repair any 'T' vertices that occur). Next, attempt to grow triangles from the new, subdivided edges that border the gap. Finally, continue with the gap filling phase, which may involve further subdivisions of large triangles.

If the peanut model is changed so that it looks more like two spheres connected by a thin strand, the area on one sphere from which the strand emanates can be triangulated improperly during the triangle growing phase. When this happens, a triangle is created which covers this area, so the strand and the other sphere are never reached. Note that cell– and particle–based tilers that follow the surface instead of sampling all space can have the same problems (but a topology–guaranteeing polygonizer [20] would not).

Details of the Particle–Based Algorithm

For comparison, the Witkin–Heckbert particle–based modeller [25] was implemented. This modeller centres a Gaussian repulsion field around each particle and particles are moved according to the forces exerted between them and neighbouring particles. A spatial grid was used to efficiently determine the neighbouring particles. Particles are automatically created in sparsely populated areas and are automatically deleted in densely populated areas. Once

the particles achieve equilibrium (i.e. the maximum particle velocity falls below a threshold), a triangulation can be performed. We built the triangulation by computing alpha shapes at various resolutions and stitching together the resulting triangle meshes. We did not count this step in the execution times of the particle–based method.

The modeller was modified to make the standard deviation of the Gaussian field around a particle proportional to the local radius of curvature of the surface. Despite this, particles in dense areas would repel particles in sparse areas, simply because there are *more* particles in the dense areas pushing on the particles in the sparse areas. This resulted in particles migrating from high curvature areas into adjacent low curvature areas and more particles continually being created in the high curvature areas to replace those that left. To prevent this migration, we gave more weight to the forces exterted by particles with larger Gaussian repulsion fields (i.e. those particle in the sparser areas). Specifically, the force exerted by particle j upon particle i was weighted by the apparent angle, as seen from i, of a disk centred at j with radius equal to the standard deviation of j's Gaussian repulsion field.

The output of the particle–based modeller might have been improved by adding a Laplacian smoothing step after the particles had reached equilibrium. Also, Paul Heckbert has suggested using an elliptical repulsion field having the axes of the ellipse aligned with the directions of minimum and maximum curvature.

5 Conclusions and Future Work

The new implicit surface polygonizer produces close–to–equilateral, curvature–dependent triangulations of G^1 surfaces. It is well suited to applications for which the quality of triangulation is important such as renderers and finite element solvers. However, the new polygonizer is substantially slower than the classical cell–based polygonizer and is not suited for interactive applications such as surface editing.

The algorithm should be extended to include non–manifold and non–G^1 surfaces: Sharp edges on the surface would have to be identified and triangles would have to be generated around these features. These triangles would be joined to the rest of the mesh in the gap filling phase. The sharp edges would be identified in the growing phase by detecting high surface curvature and by following the line of maximum curvature along the surface.

The algorithm can be made much faster for surfaces for which the curvature can be computed analytically. This is often the case, especially for surfaces constructed from a

limited set of primitives. In these cases, the algorithm would directly evaluate the curvature rather than use its slow numerical computation.

Finally, the gap filling technique might be used in other applications, such as building models from laser range data: When making a model of an object, several data sets are required in order to cover the object from all sides. These data sets are triangulated and then stitched together. The gap filling technique could be used to perform the stitching.

Acknowledgements

The authors wish to thank Paul Bourke of the Swinburne University of Technology for the image of the human cortex and Dr. James Rini of the University of Toronto for the snake venom data. We also wish to thank the anonymous reviewers for their many helpful suggestions.

Appendix

Settling a Point

Many places in the algorithm use a procedure to settle a point v onto the implicit surface, assuming that v is already near the surface. To do so, a point v' on the other side of the surface is determined. Repeated bisection of the interval between v and v' yields a point on the surface. To determine v', the vector

$$\widehat{n} = -\frac{f(v) \bigtriangledown f(v)}{|| f(v) \bigtriangledown f(v) ||},$$

which points from v to the surface, is computed. Then f is sampled at increasing distances from v in the direction of $\widehat{n}$ until a point v' is found such that the sign of $f(v')$ is opposite that of $f(v)$.

Computing Radius r

The radius r within which triangles are checked for the "Isosceles Triangle Growing" operation (Section 3.2) is chosen conservatively. Figure 11 shows the situation that generates the maximum possible value of r: Let ℓ and ℓ' be the lengths of the longest edges in triangles

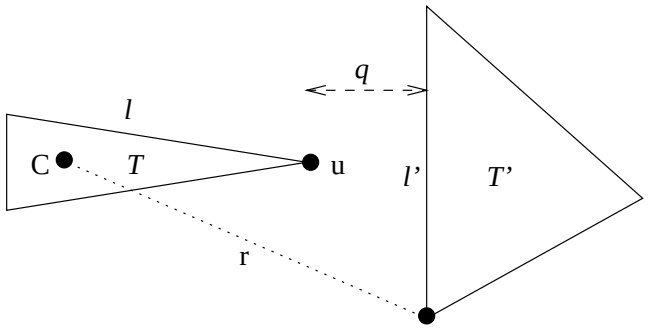


Figure 11: The geometry that produces the highest ratio of proximity radius r to inter–triangle proximity q. T is a narrow triangle with axis perpendicular to the closest and longest edge of T'.

T and T', respectively, and let q be the distance between T and T'. If T' is too close to T, then

$$r^2 < \left(\frac{2}{3}\ell + q\right)^2 + \left(\frac{1}{2}\ell'\right)^2.$$

Since ℓ' is unknown, the length of the longest edge of any triangle ever added to the mesh is maintained by the algorithm, and is used in the place of ℓ' when calculating a maximum r. This makes the test very conservative: Many triangles are needlessly checked in areas of high curvature, since nearby triangles are small compared to the largest triangle in the mesh.

References

[1] Eugene L. Allgower and Phillip H. Schmidt. An algorithm for piecewise-linear approximation of an implicitly defined manifold. *SIAM Journal on Numerical Analysis*, 22(2):322–346, April 1985.

[2] Gill Barequet and Micha Sharir. Filling gaps in the boundary of a polyhedron. *Computer–Aided Geometric Design*, 12(2):207–229, March 1995.

[3] Jules Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5:341–355, 1988.

[4] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaugman, 1997.

[5] Andrea Bottino, Wim Nuij, and Kees van Overveld. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Implicit Surfaces*, pages 53–72, 1997.

[6] B. Crespin, P. Guitton, and C. Schlick. Efficient and accurate tessellation of implicit sweeps. In *Constructive Solid Geometry*, April 1998.

[7] Luiz Henrique de Figueiredo, Jonas de Miranda Gomes, Demetri Terzopoulos, and Luiz Velho. Physically based methods for polygonization of implicit surfaces. In *Graphics Interface*, pages 250–257, 1992.

[8] Mathieu Desbrun, Nicolas Tsingos, and Marie-Paule Gascuel. Adaptive sampling of implicit surfaces for interactive modelling and animation. *Computer Graphics Forum*, 15(5):319–325, 1996.

[9] Herbert Edelsbrunner and Ernst Mücke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1):43–72, January 1994.

[10] Erich Hartmann. A marching method for the triangulation of surfaces. *The Visual Computer*, 14(3):95–108, 1998.

[11] Michael Henderson. Computing implicitly defined surfaces: Two parameter continuation. Technical Report RC 18777, IBM T. J. Watson Research Center, March 1993.

[12] Michael Henderson. Multiple parameter continuation: Computing implicitly defined k-manifolds - draft. Technical report, IBM T. J. Watson Research Center, February 2000. www.research.ibm.com/people/h/henderson/MultiParm/Continuation.html.

[13] A. Hilton, A. J. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: Range image fusion for complex object modelling. In *International Conference on Image Processing*, 1996.

[14] William Lorensen and Harvey Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics (SIGGRAPH)*, 21(4):163–169, 1987.

[15] Nelson Max. Computer representation of molecular surfaces. *IEEE Computer Graphics and Applications*, 3(7):21–29, August 1983.

[16] Donald Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.

[17] Heinrich Muller and Michael Stark. Adaptive generation of surfaces in volume data. *The Visual Computer*, 9(4):182–199, 1993.

[18] Werner Rheinboldt. On the computation of multi–dimensional solution manifolds of parameterized equations. *Numerische Mathematik*, 53:165–182, 1988.

[19] William Schroeder, Jonathan Zarge, and William Lorensen. Decimation of triangle meshes. *Computer Graphics (SIGGRAPH)*, 26(2):65–70, 1992.

[20] Barton T. Stander and John C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. *Computer Graphics (SIGGRAPH)*, 31(2):279–286, 1997.

[21] Richard Szeliski and David Tonnesen. Surface modelling with oriented particle systems. *Computer Graphics (SIGGRAPH)*, 26(2):185–194, 1992.

[22] Greg Turk. Re–tiling polygonal surfaces. *Computer Graphics (SIGGRAPH)*, 26(2):55–64, 1992.

[23] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. *Computer Graphics (SIGGRAPH)*, 28(2):311–318, 1994.

[24] C. W. A. M. van Overveld and B. Wywill. Shrinkwrap: an adaptive algorithm for polygonizing an implicit surface. Technical Report 93/514/19, University of Calgary, March 1993.

[25] Andrew P. Witkin and Paul S. Heckbert. Using particles to sample and control implicit surfaces. *Computer Graphics (SIGGRAPH)*, 28(2):269–277, 1994.

[26] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2(4):227–234, 1986.