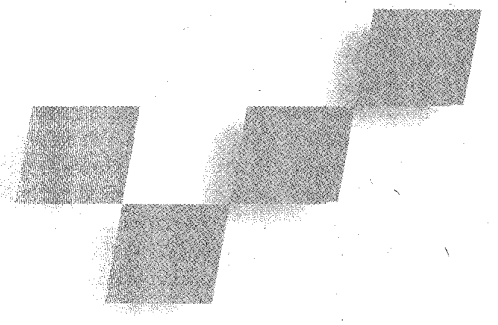
Superfaces: Polygonal Mesh Simplification with Bounded Error



Alan D. Kalvin
IBM T.J. Watson Research Center

Russell H. Taylor
Johns Hopkins University

This algorithm simplifies polyhedral meshes within prespecified tolerances based on a bounded approximation criterion. The vertices in the simplified mesh are a proper subset of the original vertices.

Planar polyhedra are used for geometric modeling of solid objects in a wide variety of applications. A major benefit of modeling with polyhedral meshes lies in the simplicity of the representation. Since planar polygons—and triangles in particular—are standard rendering primitives, they are especially useful in modeling objects for visualization.

A fundamental disadvantage, however, is that many planar faces are required to accurately describe complex shapes. This problem is frequently compounded by the algorithms used to construct the meshes. For example, almost all algorithms for creating polyhedral surfaces from data sampled on a regular 3D grid produce meshes with many small faces. This is because the grid spacing limits the maximum face size. With tiling algorithms that connect contours lying on a pair of adjacent 2D slices, the mesh faces can span no more than two slices.

Voxel-based algorithms construct isosurfaces by tessellating the 3D data into cubic or tetrahedral cells and computing where the isosurfaces pass through each cell. The faces produced by these algorithms are even more limited in size, since each face lies within a single cell (which is at most the size of a voxel). As a result, a typical mesh of a human skull produced from a 3D computed tomography (CT) study can contain from 250,000 to over 1 million triangles.

We present a general-purpose algorithm for simplifying polyhedral meshes by reducing the number of vertices, edges, and faces. This algorithm, called Superfaces, makes two major contributions to the research in this area:

- It uses a bounded approximation approach, which guarantees that a simplified mesh approximates the original mesh to within a prespecified tolerance. That is, every vertex v in the original mesh will lie within a user-specified distance ε of the simplified mesh.
- Its face-merging procedure is efficient and "greedy"—that is, it does not backtrack or undo any merging once completed. Thus, the algorithm is practical for simplifying very large meshes.

As the title of this article suggests, we are primarily concerned with creating geometrically accurate simplifications. We believe that it is far more useful and meaningful to consider "simplification quality" in terms of 3D approximation errors than it is to make subjective visual assessments based on 2D renderings of the simplifications. Of course, simplifying meshes in a way that yields nice-looking pictures is desirable, but it is not the driving issue here.

Related work

Schmitt et al.[1] used a top-down approach to simplify a regular rectangular mesh by refining a coarse approximating mesh of piecewise bicubic patches until it was within a given error bound of the original mesh. DeHaemer and Zyda[2] developed a variation of this method, using planar rectangular patches. Kalvin et al.[3] simplified isosurfaces generated from sample points on regular 3D volumes; the algorithm adaptively merges redundant coplanar polygon faces, preserving the shape of the original polyhedron.

Schroeder et al.[4] used a method called "triangle decimation," which reduces the number of faces in a triangular mesh by a specified percentage through iterative removal of vertices. Hamann[5] described a method that uses iterative removal of triangular faces ranked by vertex curvature estimates and shape. Turk[6] "re-tiled" polygonal surfaces by triangulating a new set of vertices that replaces the original one. Hoppe et al.[7] developed a mesh optimization algorithm that uses an energy-minimization scheme to simplify meshes. A method presented by Guéziec and Dean[8] simplified isosurface meshes derived from a tetrahedral tessellation of a 3D

0272-17-16/96/$5.00 © 1996 IEEE

grid; the algorithm successively removes edges, favoring removals in regions of low curvature.

Rossignac and Borrel developed a multiresolution approximation scheme that produces a series of approximations.[9] This scheme was developed for real-time visualization, and its goal was to preserve the appearance of 2D renderings of a 3D scene as the viewpoint varies. Unlike the other methods described in this section, this algorithm was not concerned with preserving the topology or geometric accuracy of the original mesh objects.

Hinker and Hansen[10] developed a simplification method called "geometric optimization." This algorithm was developed at about the same time as Superfaces.[11] Like Superfaces, it simplifies by first merging quasicoplanar faces and then triangulating the perimeters of these merged faces (the so-called superfaces). Despite the apparent similarity between these algorithms, they differ in many ways. For one thing, the geometric optimization algorithm solves a much simpler problem, since it simplifies without limiting approximation errors. Another major difference is that geometric optimization assumes that degenerate polygons are not created during the face-merging phase. It therefore does not check for them, as the Superfaces algorithm does. Since degenerate polygons can indeed occur, the geometric optimization algorithm can produce degenerate results.

Superfaces is also computationally more efficient. In both algorithms, the total running time is dominated by the initial face-merging step, which processes each of the N original polygon faces individually. For this step, Superfaces performs an $O(N)$ time greedy merge, compared with the $O(N \log N)$ process used by the geometric optimization algorithm. Since the need for mesh simplification tends to grow with the mesh size (number of polygons), this difference in computational efficiency is important in practical applications.

The Superfaces algorithm also appears to be far more effective in reducing the total polygon count. Currently, we cannot state this with complete certainty (since we have not yet compared the two algorithms directly using common data sets), but two arguments suggest its likely truth.

First, at each step of the face-merging phase, Superfaces controls the merging by considering an infinite number of possible "approximating planes" solutions (see below, "The approximating planes of a superface"). The merging of new faces into the current superface stops only when this solution set of approximating planes disappears. The geometric optimization algorithm controls the merging process by what is essentially a single approximating plane. This greatly limits the amount of face-merging possible, especially when this single plane is not well chosen.

Second, before triangulation of the superface perimeters, the Superfaces algorithm reduces the size of these perimeters by merging quasilinear edges (while preserving the error bound). In contrast, geometric optimization merges strictly colinear edges only. Since the number of triangles in the simplified mesh is essentially the same as the number of perimeter edges, Superfaces will produce a simplified mesh with a smaller number of triangles for a given set of merged faces.

Simplifying models of human anatomy

Although the Superfaces algorithm is domain-independent, one major motivation for its development is our interest in geometric modeling of human anatomy. In medicine, particularly in areas of patient therapy (for example, surgical planning, computer-assisted surgery, and radiotherapy treatment planning), it is critical to provide physicians with accurate models of the patient's anatomy. For simplified models to be accepted in clinical use, they must preserve some useful, quantifiable measure of fidelity to the original models.

Judging the quality of simplifications by visual inspection of 2D renderings of the simplified models is not sufficient. For this reason, clinicians are generally hesitant to base therapeutic treatment on models produced by existing simplification algorithms. The concerns about geometric inaccuracies usually outweigh the advantages of working with smaller data sets. The Superfaces algorithm explicitly addresses this concern and provides what we believe is a practical solution.

Preserving geometric accuracy

The important goal of ensuring that the reduced mesh is a good geometric approximation of the original mesh has been considered in a number of ways. Schroeder et al.[4] used a simplification criterion measure based on the distance of a vertex to a plane or edge. These distances were measured relative to intermediate mesh approximations, not the original mesh. For triangular meshes produced from bivariate functions, Hamann's method[5] computed a root-mean-square approximation error and used it as a stopping criterion for the simplification process. Turk[6] used a "point repulsion" method of uniformly distributing the set of simplified vertices over the original mesh to preserve the fidelity of the simplified mesh to the original.

Guéziec and Dean[8] focused on preserving accuracy in high-curvature regions. They reported results showing that their method produced more accurate meshes in these regions than were created by subsampling the initial 3D volume before isosurface generation.

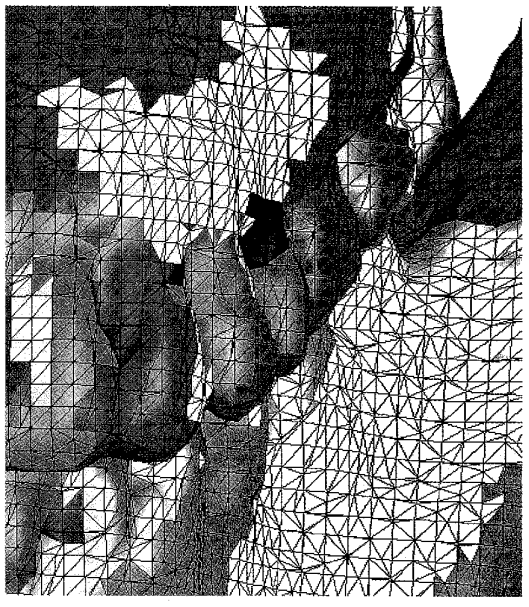
Hoppe et al.[7] defined the approximation error of a simplified mesh to be the sum of squared distances from the original vertices to the simplified mesh. They used this error measure as the "distance energy" term in the energy function that is minimized.

The only methods we are aware of that actually maintain bounds on the approximation errors introduced in the simplification process are the algorithms of Schmitt et al.[1] and DeHaemer and Zyda.[2] Both methods apply only to regular rectangular meshes.

An essential advantage of the Superfaces algorithm accrues from the following properties:

- it provides a provable bound on the approximation error, and
- it applies to any polyhedral mesh that is a valid manifold (the mesh need not be regular or closed, and its faces need not be triangles).

1 A polyhedral model of a skull partitioned into superfaces.



2 Detail of the skull model shown in Figure 1.

The Superfaces algorithm

In describing the Superfaces algorithm, we use the following terms: A *surface patch* is a set of connected polyhedral faces, and the *boundary* of a surface patch is the set of vertices lying on its perimeter. The boundary of each surface patch forms a nonplanar polygon, called a *superface*. We say that a superface subsumes the faces, vertices, and edges of the underlying surface patch. The faces that surround a surface patch (that is, those faces adjacent to a surface patch) are called *border faces* of the corresponding superface.

In simplifying a polyhedral mesh P_O, the Superfaces algorithm partitions the faces in P_O into a set of surface patches and approximates P_O by approximating each surface patch with a triangulation of its corresponding superface. The algorithm has the following key features:

1. The simplified mesh approximates the original mesh to within a given tolerance. That is, no vertex in the original mesh is more than a user-specified distance ε from the simplified mesh. Since the vertices of the simplified mesh come from the original mesh (see point 5 below), the converse is obviously true as well. That is, no vertex in the simplified mesh is more than ε from the original mesh.
2. The algorithm is very efficient, and it is a practical method for simplifying very large meshes, such as those derived from medical CT and MRI data (see sidebar on previous page).
3. The topological properties of the original mesh are preserved.
4. The method is domain-independent; it does not require any knowledge about the nature of the data to perform the simplification.
5. The vertices on the superface boundaries form a proper subset of the set of original vertices, so the algorithm is particularly suitable for building hierarchical representations of polyhedral meshes.

The Superfaces algorithm simplifies a mesh in three phases:

1. *Superface creation*. A face-merging procedure partitions the original faces into superface patches. Figure 1 shows a polyhedral object partitioned into superfaces, which are the colored patches surrounded by black perimeters. Figure 2 shows a close-up view of the skull in Figure 1 (near the teeth) with the original mesh shown in black to illustrate a typical relationship between the sizes of superfaces and the faces in the original mesh.
2. *Border straightening*. The borders of the superfaces are simplified by merging boundary edges. We call these merged edges *superedges*.
3. *Superface triangulation*. Triangulation points for the superfaces are defined. In this phase, a single superface can be decomposed into many superfaces, each with its own boundary and triangulation point.

Phase 1: Superface creation

Superface growing is based on a bottom-up, face-merging procedure. It is a "greedy" method—that is, it does not backtrack or undo any merging once done. A major reason for the algorithm's efficiency is that the face-merging runs in time linear in the number of faces.

The creation of a superface begins with the selection of an initial "seed" face that grows through a process of accretion. Border faces (that is, faces on the current superface boundary) are merged into the evolving superface if they satisfy the required merging criteria. A superface eventually stops growing when there are no more faces on its boundary that can be merged.

The "seed" faces for growing the superfaces are selected randomly from the set of (unmerged) faces in the

original mesh. The superface creation process ends when all the original faces have been merged.

The approximating planes of a superface. Associated with each superface F is a set E of feasible approximating planes. Every plane $p \in E$ satisfies the *bounded approximation constraint*, which stipulates that all vertices subsumed by F lie within a bounded distance of p, together with some other constraints that we will discuss below.

Also associated with each superface F is a "nominal" coordinate system $N = [R, \mathbf{v_0}]$, where R is a rotation matrix and $\mathbf{v_0}$ is the coordinate system origin in global coordinates. N is somewhat arbitrary, but chosen so that the outward facing normal of F is aligned approximately with R_z (the z-axis of N), and $\mathbf{v_0}$ is located somewhere "within" F. In the following discussion, we assume that all vertices and planar faces have been transformed into this coordinate system.

Choosing N in this way is convenient, since if we consider the "direction form" of the equation of a plane, $ax + by + z = d$, then E corresponds to the set of all points in (a, b, d) space that obey a set of constraints $C_i(a, b, d) \leq 0$. In this discussion, we use linear constraints of the form

$$C_i \cdot \mathbf{k} \leq c_i$$

so that E forms a polytope in (a, b, d)-space,

$$E = \{\mathbf{k} = [a, b, d]^T \mid C \cdot \mathbf{k} \leq c\}$$

A superface is grown by the successive merging of individual faces. The addition of each individual face f_b generates additional constraints $C_j \cdot \mathbf{k} \leq c_j$, which are used to "lop off" pieces of E. This greedy method stops when the addition of a face would cause E to become empty.

In general, the number of constraints will grow linearly with the number of faces being subsumed into superface F, and the polytope itself can become rather unwieldy for computation. Therefore we approximate the set E with a conservative ellipsoidal approximation

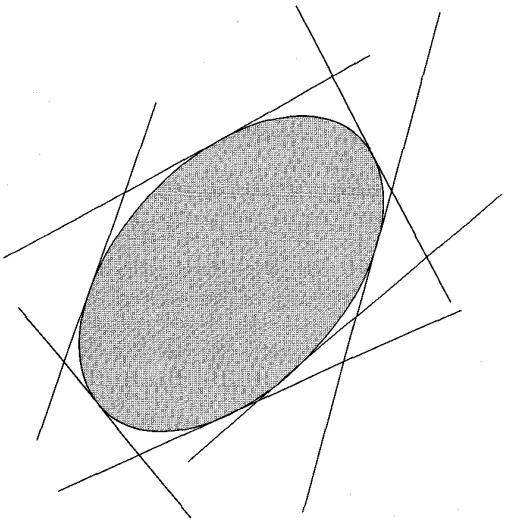
$$\overline{E} = \{\mathbf{k} = [a, b, d]^T \mid (\mathbf{k} - \mathbf{k_0})^T Q^T \operatorname{diag}(\beta) Q(\mathbf{k} - \mathbf{k_0}) \leq 1\}$$

where Q is orthogonal and all the β_i are positive real numbers (see Figure 3).

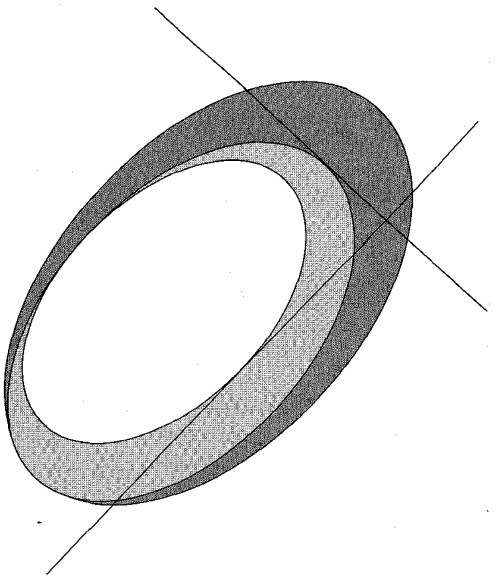
The fundamental growing step. The basic growing step is the expansion of the superface boundary through the merging of all acceptable border faces. A border face f_b is accepted if it satisfies a set of merging conditions (see below, "The merging rules"). Each condition is expressed in the form of a linearized constraint that defines a half-space in (a, b, d)-space:

$$H = \{\mathbf{k} \mid C_i \cdot \mathbf{k} \leq c_i\} \quad \textbf{(1)}$$

If $\overline{E} \cap H$ is empty, then f_b does not satisfy the merging conditions and is rejected. Otherwise, f_b is merged into the growing superface, and the ellipsoidal subset of feasible planes $\overline{E}$ is adjusted to satisfy Equation 1. This is shown schematically in Figure 4.



3 A polytope and its approximating ellipsoid.



4 Adjusting the set of feasible solutions.

While the ellipsoidal approximation of E can result in overly conservative solution sets, $\overline{E}$ can be adjusted in constant time. Therefore, the runtime face-merging procedure is linear in the number of polyhedral faces being merged.

Perimeter validity. To triangulate superfaces in phase 3 of the algorithm, we need to ensure that each superface is valid in the sense that projecting its perimeter into the nominal approximating plane yields a simple 2D polygon that does not self-intersect.

Checking perimeter validity is expensive because it involves the pairwise testing of all perimeter edges for intersection. We therefore use the following strategy to keep down the number of perimeter checks:

- Grow the superface to completion, ignoring the validity of the intermediate perimeters.

- If the final perimeter in not valid, "fix up" the superface by regrowing it as follows:
  - Regrow the superface, starting again from the original seed face.
  - Check perimeter validity after each iteration of perimeter expansion.
  - Stop growing as soon as an invalid perimeter is found.

This approach works well, since the merging rules inhibit the creation of invalid superfaces. Consequently, most superfaces need just one perimeter check, and only a small fraction of them require regrowing (in our experiments, at most 3 percent of the total number of superfaces).

The merging rules. The linear constraints that control superface growing are derived from a set of merging rules. In the current implementation of the Superfaces algorithm, we use three merging rules.

The planarity rule: All vertices on face f_b must be within a distance of $\varepsilon/2$ from each approximating plane p : $ax + by + z = d$.

That is, for each $v = (v_x, v_y, v_z) \in f_b$,

$$|(a, b, 1, -d) \cdot (v_x, v_y, v_z, 1)| \leq \varepsilon/2$$

This gives the pair of linear constraints on (a, b, d):

$$-\varepsilon/2 - v_z \leq av_x + bv_y - d \leq \varepsilon/2 - v_z$$

This rule is used to ensure that each vertex in the original mesh is within ε of the simplified mesh being created.

The face-axis rule: The orientation of face f_b must be similar to the orientation of each approximating plane $p : ax + by + z = d$. This rule is expressed by

$$\arccos\left(\frac{(a,b,1)\cdot(n_x,n_y,n_z)}{\sqrt{a^2+b^2+1}}\right) \leq \theta_{\max}$$

where (n_x, n_y, n_z) is the outward-facing unit normal of face f_b; $(a, b, 1)$ is the outward-facing normal of plane p : $ax + by + z = d$; and $\theta_{\max}$ is the maximum allowable angle between these two normals.

Since the choice of $\theta_{\max}$ is somewhat arbitrary, nothing is "lost" in approximating the above constraint by the linearization:

$$\begin{aligned} a \cdot n_x + b \cdot n_y &\geq (a^2 + b^2 + 1)\cos(\theta_{\max}) - n_z \\ &\geq \cos(\theta_{\max}) - n_z \end{aligned}$$

The no-foldover rule: Face f_b must not "fold over" or tuck under the superface. This condition is enforced by requiring that in the orthogonal projection into each approximating plane, the vertices of f_b lie outside the projected superface perimeter.

Let (v_x, v_y, v_z) denote the 3D coordinates of a vertex $\mathbf{v}$, and let $\mathbf{v}'$ be its orthogonal projection into plane $p : ax + by + z = d$. If $(\mathbf{u}, \mathbf{w})$ is the edge of f_b lying on the superface perimeter, and if $\mathbf{v}$ is any other vertex in f_b, then we wish to ensure that $\mathbf{v}'$ lies to the right of edge $(\mathbf{u}', \mathbf{w}')$. That is,

$$(u'_y - w'_y)v'_x + (w'_x - u'_x)v'_y + u'_x w'_y - w'_x u'_y \leq 0 \quad \textbf{(2)}$$

Let p_{nom} be the nominal approximating plane of f_b, and let $\hat{\mathbf{v}}$ be the projection of vertex $\mathbf{v}$ into p_{nom}, along $(a, b, 1)$ (the outward-facing normal of plane p). Then $\hat{\mathbf{v}} = (v_x - av_z, v_y - bv_z, 0)$. Since the angle between the outward-facing normals of p and p_{nom} is less than 90 degrees, $\mathbf{v}'$ lies to the right of edge $(\mathbf{u}', \mathbf{w}')$ if and only if $\hat{\mathbf{v}}$ lies to the right of edge $(\hat{\mathbf{u}}, \hat{\mathbf{w}})$. Hence, we can linearize the no-foldover constraint by substituting $\hat{\mathbf{u}}$, $\hat{\mathbf{v}}$, $\hat{\mathbf{w}}$ for $\mathbf{u}'$, $\mathbf{v}'$, $\mathbf{w}'$ in Equation 2 to give

$$\begin{aligned} &(u_y - w_y)v_x + (w_x - u_x)v_y + u_x w_y - w_x u_y \\ &\leq a[(w_z - u_z)v_y + (u_y - w_y)v_z + w_y u_z - u_y w_z] \\ &- b[(w_z - u_z)v_x + (u_x - w_x)v_z + w_x u_z - u_x w_z] \end{aligned}$$

Gerrymandering check. An additional constraint on superface growth can be enforced by a heuristic "gerrymandering " or "irregularity" check that prevents superfaces from becoming too long, thin, or grossly irregular. A simple estimate of the irregularity of a superface is the ratio p^2/Ω, where p is the length of the superface perimeter, and Ω is the area of the surface patch spanned by the superface. The threshold of acceptable irregularity is estimated as $[2(1 + h)]^2/h$, the ratio of perimeter squared to area of a $1 \times h$ rectangle.

We are especially interested in applying the Superfaces algorithm to simplify polyhedra that have been constructed from 3D data using the Alligator algorithm[3] or the Marching Cubes algorithm.[12] In these polyhedra, all the faces are of similar size, and so too are the edges. So the superface perimeter p and area Ω can be approximated by the number of perimeter edges and subsumed faces, respectively. Since both these values are calculated during the regular course of superface growth, the estimated superface irregularity can be calculated with just three additional floating-point multiplications.

Phase 2: Superface border straightening

Suppose for the moment that we think of surface area as a measure of "face size" and scalar length as a measure of "edge size." Then the merging of faces into superfaces in phase 1 produces a change of scale in face size without a corresponding change of scale in edge size: The edges on each superface perimeter are simply edges of the original mesh. In this second phase of the Superface algorithm, we make an appropriate change of scale in edge size, creating *superedges* by straightening the superface perimeters in two steps: maximal edge merging and edge splitting, as illustrated in Figure 5.

Maximal edge merging. The perimeter between each pair of adjacent superfaces is merged into a single superedge. That is, if F is a superface with perimeter $P = (v_1, v_2, \ldots, v_n)$, and F_j is a neighboring superface, then the segment $s_j = (v_{j1}, v_{j2}, \ldots, v_{jr})$ of maximal common boundary between F and F_j is replaced by the superedge $L_j = \overline{v_{j1} v_{jr}}$.

Edge splitting. Maximal edge merging can produce an "oversimplified" mesh that is not within the required ε limit of all the vertices of the original mesh. To compensate for any possible oversimplification, the superedges are split. Typically, oversimplification occurs when a "tongue-like" region of one superface "shifts" to a neighboring superface. The result is that some of the vertices that shift from superface F_1 to superface F_2 are no longer within ε of either superface. This is because

- in the orthogonal projection into every one of the F_1 approximating planes, the vertices lie outside the perimeter of F_1, and
- the vertices are either (a) not within ε of any of the F_2 approximating planes or (b) outside the perimeter of F_2 in the orthogonal projection into every one of the F_2 approximating planes.

We use a standard polyline approximation method to split each superedge $L_j=\overline{v_{j1}v_{jr}}$. Let v_{jt} be that vertex in the segment $s_j = (v_{j1}, v_{j2}, \ldots, v_{jr})$ that is furthest from L_j. If the distance from v_{jt} to L_j is greater than some threshold $d_{\max}$, we recursively split the lines $L_1=\overline{v_{j1}v_{j2}}$ and $L_2=\overline{v_{jt}v_{jr}}$.

A sufficient condition for the bounded approximation criterion to hold after superedge splitting is that the splitting threshold $d_{\max}$ satisfies

$$d_{\max} \le \begin{cases} \varepsilon & 0\,\text{deg} \le \theta_{\max} \le 30\,\text{deg} \\ \dfrac{\varepsilon\cos(\theta_{\max})}{\sin(2\theta_{\max})} & 30\,\text{deg} \le \theta_{\max} \le 45\,\text{deg} \\ \varepsilon\cos(\theta_{max}) & 45\,\text{deg} \le \theta_{max} \le 90\,\text{deg} \end{cases}$$

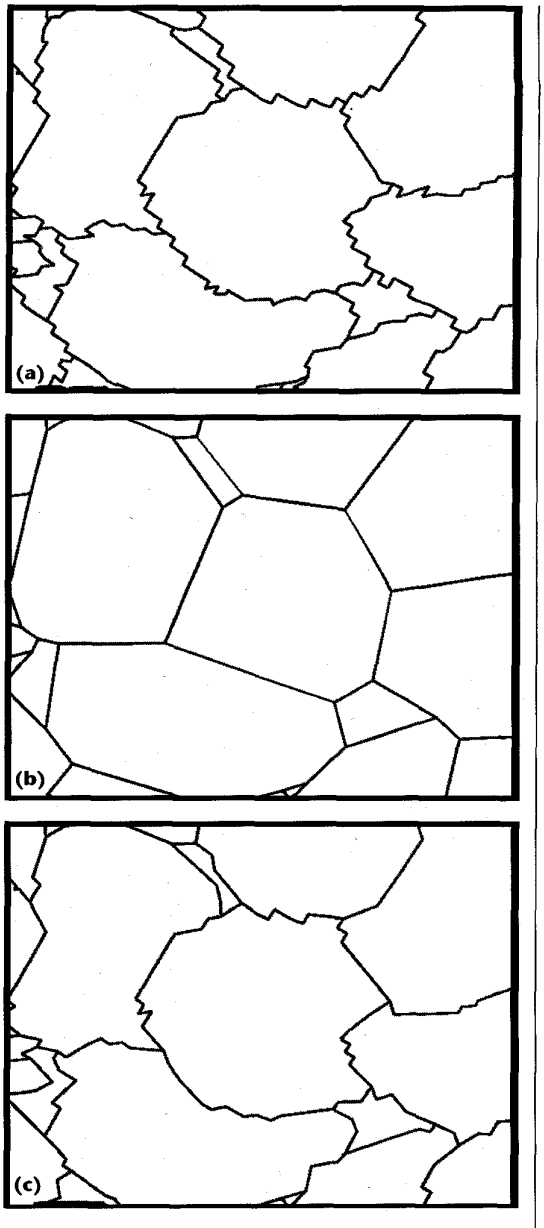
where $\theta_{\max}$ is maximum allowable angle used by the face-axis rule (see above, "The merging rules").

Alternative border-straightening method. The two-step method described above is a very conservative way to straighten borders, since it requires every vertex to be within ε of its *subsuming* superface, even though the bounded error condition is less restrictive and requires only that every vertex be within ε of *any* superface. The advantage of this approach is its efficiency. We do not have to explicitly identify exposed vertices that lie too far from the simplified mesh.

A more aggressive way to straighten borders is to split the initial superedge L between adjacent superfaces F_1 and F_2 only if necessary. That is, we subdivide only if there is at least one vertex subsumed by F_1 or F_2 that is further than ε from both these superfaces. This approach produces fewer superedges (and fewer triangles and therefore a better simplification) than the conservative method described above, but it is computationally more expensive (for comparisons, see "Experimental results").

Phase 3: Computing triangulation points

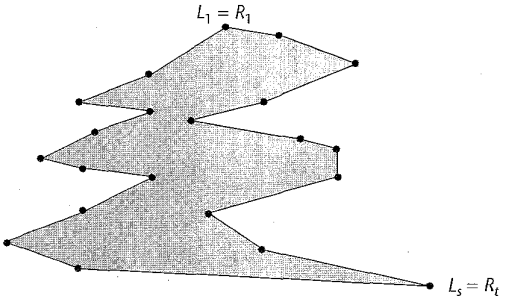
Recall that each superface is a polygon corresponding to the boundary of a surface patch in P_0, the mesh being simplified. We want to approximate P_O by approximating each surface patch by a triangulation of its subsuming superface. In this phase of the algorithm, we calculate triangulation points for the superfaces.



5 Superface borders (a) before straightening, (b) after edge merging, (c) after edge splitting.

To find a suitable triangulation point for a nonplanar superface f_b, we compute f'_b, the projection of f_b into its nominal approximating plane. We then search for a *star point* of planar polygon f'_b, that is, a point $\mathbf{v}$ inside f'_b that is visible from each of its vertices.

This method of triangulation reduces the 3D problem to a 2D problem and guarantees a superface triangulation in which no two triangles intersect. If a star point $\mathbf{v}$ is found, f'_b is a *star polygon*, and v (treated as a point in 3D lying on the approximating plane) is used as the triangulation point for superface f_b. If a star point is not found, then f'_b is decomposed into a set of star polygons $f'_{b1}, f'_{b2}, \ldots f'_{bk}$, with a corresponding decomposition of f_b into superfaces $f_{b1}, f_{b2}, \ldots, f_{bk}$.



6 Monotone polygon *P*.

Finding the star point of a polygon. Let f'_b be the projection of superface f_b into its approximating plane, and let $(\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}n)$ be a counterclockwise ordering of the vertices on the perimeter of f'_b.

A star point $\mathbf{v} = (x, y)$ of f'_b will lie to the left of each edge $(\mathbf{v}_i, \mathbf{v}_{i+1})$. So v will satisfy a set of constraints $C_i(\mathbf{v}_i, \mathbf{v}_{i+1})$, linear in (x, y), of the form

$$(y_{i+1} - y_i)x + (x_i - x_{i+1})y \leq x_i y_{i+1} - x_{i+1} y_i$$

The C_i's define a polytope $K(f_b)$, called the *kernel* of f'_b. The Superfaces algorithm computes a set of feasible star points $\overline{K} \subset K$, using the technique described earlier for approximating a polytope with a circumscribed ellipsoid (see "The approximating planes of a superface").

Since $\overline{K}$ is a conservative approximation of the kernel, it can be empty when K is not. The method might then fail to find any feasible star points for a true star polygon. In this case, the algorithm will decompose a star polygon into multiple star polygons.

The advantage of using this suboptimal approach for finding feasible star points is that it is both efficient, running in linear time, and trivial to implement, employing the same polytope approximation technique used for finding feasible approximating planes. An alternative method for finding feasible star points is the algorithm of Lee and Preparata.[12] Their algorithm is optimal and linear, but much more complicated to implement than the method described above.

Decomposing a polygon into star polygons. If a simple, planar r-sided polygon has no holes, it can be decomposed (partitioned) into at most $\lfloor r/3 \rfloor$ star polygons in $O(r \log r)$ time,[13] and it can be decomposed into a minimum number of star polygons in $O(r^5 k^2 \log r)$ time[13] (where k is the number of reflex vertices).

However, neither of these methods can handle polygons with holes, and so neither is suitable for decomposing 2D projections of superfaces. Since decomposing a polygon with holes into a minimum number of star polygons is not practical (it is NP-hard), we have developed an $O(r^2)$ algorithm that will decompose a polygon, possibly having holes, into a small (rather than minimum) number of star polygons.

This algorithm does the decomposition in two phases, first decomposing the polygon into monotone polygons, then decomposing each monotone polygon into star polygons.

From polygons with holes to monotone polygons. Let $C = (v_{j1}, v_{j2}, \ldots, v_{jr})$ be a segment or chain of polygon $P = (v_1, v_2, \ldots, v_n)$. C is said to be monotone with respect to a line L if the projections of $v_{j1}, v_{j2}, \ldots, v_{jr}$ onto L have the same ordering as in the chain. In other words, any line orthogonal to L will intersect C in at most one point. A polygon is monotone if it can be partitioned into two chains that are monotone with respect to the same line (see Figure 6).

A polygon vertex v_i is *reflex* if its interior angle is larger than 180 degrees. A reflex vertex v_i is called an *interior cusp* if the y-coordinates of its adjacent vertices, v_{i-1} and v_{i+1}, are either both larger or both smaller than the y-coordinate of v_i. Garey et al.[13] showed that a polygon with no interior cusps is monotone (with respect to the y-axis).

Lee and Preparata developed an algorithm that decomposes a polygon without holes into monotone polygons by using a "plane sweep" technique to remove all interior cusps.[13] We have extended this algorithm to handle polygons with holes and use it in the first step of our algorithm for decomposing polygons with holes into star polygons.

From monotone polygons to star polygons. After partitioning a polygon P (that may have holes) into monotone polygons $P_1, P_2, \ldots, P_m$, we decompose each P_i into star polygons. (Note that in this step we don't need to worry about holes, since a monotone polygon cannot have any.) The decomposition is done without introducing new vertices (also called Steiner points).

It is possible, of course, to adapt an algorithm for triangulating monotone polygons so that it produces a star decomposition by eliminating some of the internal diagonals created for the triangulation. This is the approach of Avis and Toussaint.[13] The difficulty for us with this strategy is that it does not necessarily lead to decompositions with small numbers of star polygons. For example, a straightforward modification of the $O(r \log r)$ triangulation algorithm of Garey et al.[13] (and one that preserves the algorithm's computational efficiency) does not allow much control over the number of star polygons that will be produced.

Therefore we developed a new decomposition algorithm that specifically attempts to keep the number of star polygons small. To describe our star decomposition algorithm, suppose polygon P is a monotone with respect to the y-axis and has a left chain $(L_1, L_2, \ldots, L_s)$ and a right chain $(R_1, R_2, \ldots, R_t)$ as shown in Figure 6. The algorithm visits the vertices of P in order of descending height (that is, y-coordinate), creating all the star partitions in one top-to-bottom scan of P.

Starting at the top of P, the algorithm searches for the first two reflex vertices. If P has no reflex vertices, it is convex and any vertex can be chosen as the star point. If P has exactly one reflex vertex, then this vertex is a star point.

Otherwise, let A and B be the first and second reflex vertices found. The basic step in the algorithm is to create an internal diagonal from vertex B to some vertex C that lies below B and on the opposite chain. This diagonal partitions P into

- a star polygon $P_{star} = (C, \ldots, A, \ldots, B)$ with star point A, and
- a truncated monotone polygon $P_{trunc} = (C, B, \ldots, L_s)$.

We repeat this process, lopping off a star polygon from the monotone polygon, until the bottommost vertex in P is reached, at which stage we are done.

For simplicity, we describe the process of finding the vertex C used to define the internal diagonal $[B, C]$ for the case when vertex B is on the left chain. For B on the right chain, the method is the same with the terms "left" and "right" interchanged. The method is described by the pseudocode shown in Figure 7, using the following notation:

- L_i and R_j are the most recently visited vertices on the left and right chains, respectively.
- (v, w) represents the directed line segment from vertex v to vertex w.

To find C, we descend polygon P, maintaining two bounding line segments, (S_1, S_2) and (R_{j-1}, A). We stop at the first R_j, the current vertex on the right chain, that does not lie between the bounding lines segments. R_{j-1} becomes vertex C, and diagonal $[B, C]$ is created (see Figure 8). The line segment (S_1, S_2) ensures that no edges below B on the left chain will intersect diagonal $[B, C]$, and the line segment (R_{j-1}, A) ensures that all vertices on the right chain up to and including vertex C will be visible from A.

If C is not visible from B—that is, if C is not on the right of (B, A)—then $[B, R_{j-1}]$ is not an internal diagonal (see Figure 9). This can occur only for termination on the condition that R_j is not on the right of (S_1, S_2). The method for handling this case is given in Figure 10 (next page).

Computational complexity

As noted earlier, the face-merging procedure in phase 1 runs in $O(n)$ time, where n is the number of faces in the original mesh. However, the complete algorithm does not necessarily run in $O(n)$ time. All other steps in the algorithm (perimeter validity checking, border straightening, and superface triangulation) require more than linear running time in the size of the superface perimeters.

Nevertheless, from a practical standpoint, the number of original faces (n) is typically much greater than the sizes of the superface perimeters. Therefore, the crucial issue—and the reason the Superfaces algorithm is efficient—is that the initial, face-merging step is linear.

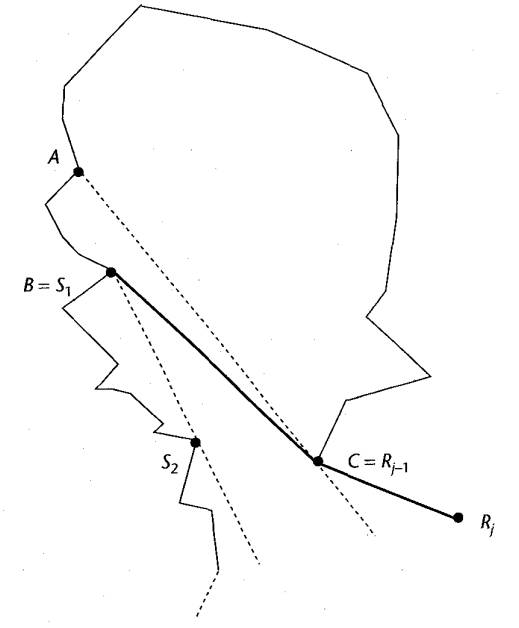
Experimental results

We now discuss some results of using the Superface algorithm, presenting tables of error measures as well as rendered images.

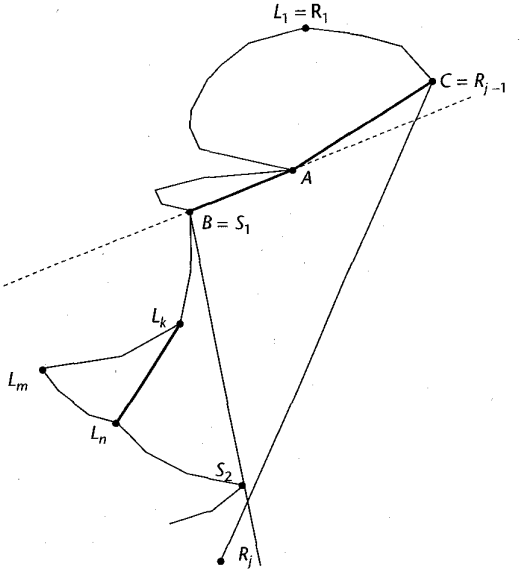
As stated previously, one of our primary interests in simplifying polyhedral meshes is to obtain results that preserve geometric accuracy rather than results that simply look good. Bearing this in mind, together with the fact that there is no single "correct" way to shade surfaces, we have not used the same shading technique for all renderings (even though this approach might

```
(S1, S2) ← (B, A) ;
terminate ← FALSE;
while (terminate = FALSE) do
    w ← next vertex in descending order of height;
    if w is on the left chain then
        Li ← w;
        if Li lies to the right of (S1, S2) then (S1, S2) ← (Li, B);
    else
    Rj ← w;
        if Rj is not on the left of (Rj–1, A) then
            terminate ← TRUE;
        else if Rj is not on the right of (S1, S2) then
            terminate ← TRUE;
end while
C ← Rj–1;
Create internal diagonal [B, C];
Make star polygon (C, ... , A, ... , B) with star point A;
```

7 When $[B, C]$ is an internal diagonal, $(C, \ldots, A, \ldots, B)$ is a star polygon with star point A.



8 Creating star polygon $(C, \ldots, A, \ldots, B)$.



9 The case where $[B, C]$ is not an internal diagonal.

```
Lm ← first nonreflex vertex below B;
Ln ← first reflex vertex below Lm;
Lk ← first vertex in chain (B, ... , Lm) visible from Ln;
Create internal diagonal [Ln, Lk];
Make star polygon (Lk, ... , Lm , , ... , Ln) with star point Ln;
if [A, C] is not an edge in P then
    Create internal diagonal [A, C];
    Make star polygon (C, ... , A) with star point A;
if [B, A] is not an edge in P then
    Create internal diagonal [B, A];
```

10 Creating star polygons when [*B, C*] is not an internal diagonal.

11 Original skull model (349,792 triangles).

12 Simplified skull (a) mesh and (b) color-coded approximation errors in pixel units: ε = 0.5 (36.60 percent of original triangles).

13 Simplified skull (a) mesh and (b) color-coded approximation errors in pixel units—with aggressive border straightening: ε = 0.5 (15.58 percent of original triangles).

make direct visual comparisons a bit easier). Rather, we applied Gouraud shading to those meshes that have been only slightly simplified, and we used flat shading for the others. For the flat shading, the normal used in rendering each triangle is a weighted sum of the triangle's surface normal and the nominal normal of the superface to which the triangle belongs.

Human skull

We applied the Superfaces algorithm on a polyhedral mesh of the human skull that contains 349,792 triangular faces and 174,834 vertices.

The mesh was constructed by the Alligator surface construction algorithm[3] from a 170 × 170 × 173 3D volume. This volume was produced by subsampling the slices from a CT scan of a life-size plastic replica of a human skull consisting of 173 slices of 512 × 512 pixels. The subsampling was done to handle current memory limitations associated with the implementation of the winged-edge polyhedral modeling system used in this work. Alternatively, we could have retained the original 512 × 512 slices and constructed a partial skull from a subvolume (for example, quadrant or octant) of the CT scan. However, this would have led to polyhedral meshes with large artificial planar regions along the subvolume boundaries, which would skew the results of the simplification. The dimensions of the bounding box (width × depth × height) of the original skull are 109 × 177 × 168 pixel^3 = 156.5 × 254.1 × 241.2mm^3, and the unit of distance used for ε is one pixel.

Figure 11 shows the original skull mesh, and Figure 12 shows the simplified mesh that approximates the original with a maximum error bound of ε = 0.5, using conservative border straightening. Figure 12a shows the simplified mesh itself, and Figure 12b shows a color-coded illustration of the corresponding approximation errors (in pixel units). Figure 13 shows the results for simplification with ε = 0.5, using aggressive border straightening.

Figures 14 and 15 show the simplified meshes that approximate the original with a maximum error bound of ε = 4.0, with conservative and aggressive border straightening. Here the simplifications are produced using flat shading.

Table 1 shows results for a range of values for the approximation bound ε and a fixed θ_{max} = 45 degrees (see above, "The merging rules"). Each row in Table 1 shows

- the mean approximation error,
- the maximum approximation error,

- the number of triangles in the simplified mesh,
- the percentage of triangles remaining after simplification, and
- simplification running time on an RS/6000 model 550 uniprocessor workstation.

Note that for all values of ε the average approximation error is approximately an order of magnitude below the maximum allowable error.

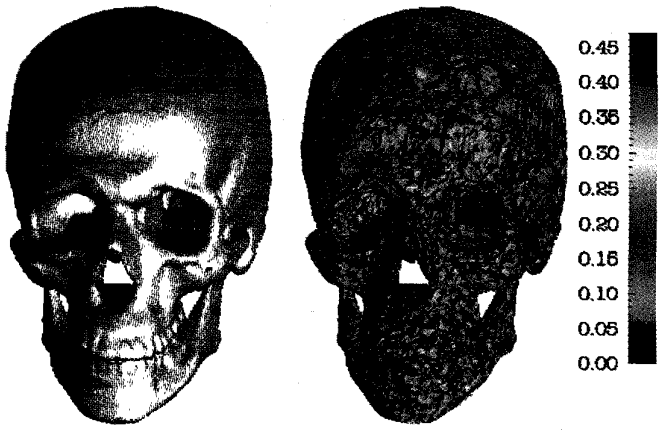
Table 2 shows the improvements achievable using the more aggressive (and expensive) border straightening described in the discussion of phase 2. Recall that this approach consists of (a) straightening borders as much as possible by creating a single superedge L between each adjacent superface pair (F_1, F_2)—maximum border straightening—and then (b) subdividing each L only if necessary. That is, we subdivide only if there is at least one vertex subsumed by F_1 or F_2 that lies beyond the ε limit from the simplified mesh.

Table 2 shows the results of the first stage of this approach—maximum border straightening. Here, setting $\theta_{max} = 60$ degrees produced the best results. The two additional columns in this table are

- the number of vertices in the original mesh that lie beyond e from the simplified mesh, and
- the number of superfaces that must have their borders adjusted over the total number of superfaces.

These results are very encouraging and show two things:

- Aggressive border straightening improves simplification signifi-



14 Simplified skull (a) mesh and (b) color-coded approximation errors in pixel units: ε = 4.0 (6.91 percent of original triangles).

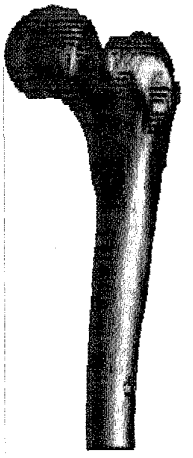
15 Simplified skull (a) mesh and (b) color-coded approximation errors in pixel units—with aggressive border straightening: ε = 4.0 (2.55 percent of original triangles).

Table 1. Results of simplifying the skull mesh of 349,792 triangles.

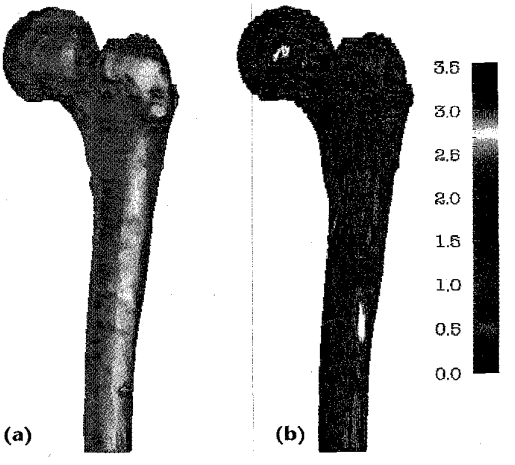
Error Bound ε	Approximation Error Mean	Approximation Error Max.	Triangles Count	Triangles Percent of Original	Running Time (m:ss)
0.5	0. 0544	0.4723	128,040	36.60	9:52
1.0	0. 1289	0.9231	78,002	22.30	8:03
1.5	0. 2017	1.4387	50,442	14.42	7:34
2.0	0. 2559	1.8690	37,438	10.70	6:41
3.0	0. 3088	2.6119	28,388	8.12	6:26
4.0	0. 3358	2.7684	24,170	6.91	6:00

Table 2. Results of simplifying the skull mesh of 349,792 triangles—with aggressive border straightening.

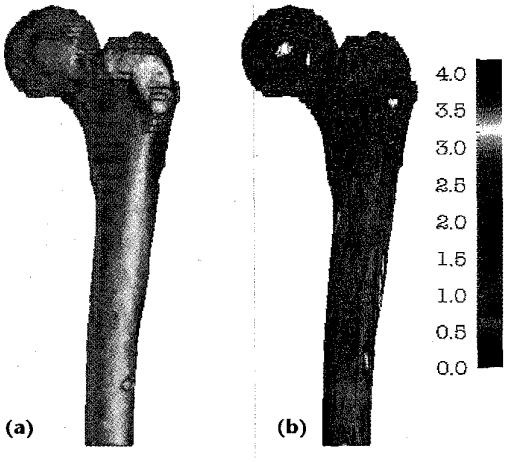
Error Bound ε	Approximation Error Mean	Approximation Error Max.	Triangles Count	Triangles Percent of Original	Running Time (m:ss)	Vertices above ε limit	Superfaces to adjust/ total superfaces
0.5	0.0947	1.4240	53,790	15.38	8:49	31	18/14,403
1.0	0.2187	1.3973	23,704	6.78	7:12	36	9/ 5,626
1.5	0.3402	2.3713	15,470	4.42	6:28	56	7/ 3,606
2.0	0.4523	5.8117	11,994	3.43	6:20	184	5/ 2,738
3.0	0.5984	3.3584	9,820	2.81	6:03	19	1/ 2,299
4.0	0.6714	3.6544	8,934	2.55	5:52	0	–



16 Original femur model (179,916 triangles).



17 Simplified femur (a) mesh and (b) color-coded approximation errors in pixel units: $\varepsilon = 4.0$ (12.14 percent of original triangles).



18 Simplified femur (a) mesh and (b) color-coded approximation errors in pixel units—with aggressive border straightening: $\varepsilon = 4.0$ (4.41 percent of original triangles).

cantly. For the skull we see improvements by factors between 2.38 to 3.29. While this improvement will drop somewhat after the necessary superface borders are subdivided, the results should not change appreciably because so few subdivisions are needed.

- Very few of the vertices lie beyond the ε limit. In the worst case ($\varepsilon = 0.5$), just 27 vertices out of 174,834 lie beyond the limit; and for $\varepsilon = 2.5$ and 4.0, no vertices are out of range. Therefore, only a small fraction (in the worst case, less than a fifth of one percent) of the superfaces require border adjustments to make the simplified mesh satisfy the ε-tolerance criterion.

These results show the significant improvements possible from using the more aggressive border-straightening approach and suggest that most of the extra expense comes from detecting out-of-range vertices. Very little additional work (or possibly none) is required to correct the oversimplification of the mesh and to get the vertices back within range.

Human femur

Figure 16 shows a polyhedral model of a human femur consisting of 179,916 triangles obtained from a CT scan. The dimensions of its bounding box are $231 \times 197 \times 585$ pixels^3 = $90.2 \times 76.8 \times 228.6$ mm^3, and as with the skull, ε is given in pixel units.

Figure 17 shows the simplified mesh and corresponding error map obtained by simplifying with $\varepsilon = 4.0$ using conservative boundary straightening. Figure 18 shows the results of simplifying with $\varepsilon = 4.0$ using aggressive boundary straightening at $\theta_{max} = 60$ degrees.

Tables 3 and 4 summarize the results of simplifying over a range of ε, with $\theta_{max} = 45$ degrees used for conservative boundary straightening and $\theta_{max} = 60$ degrees for aggressive boundary straightening.

Topographic map of the earth

Figure 19 shows a map of topographic data of the surface of the earth exhibiting a range of heights from –6,819.7 meters below sea level to 5,487.4 meters above sea level. The mesh was produced by triangulation of a 360×180 rectangular mesh. (Note that the vertical resolution of the original data is at best 1 meter, and the horizontal resolution is at best 5 minutes of latitude and longitude, assuming that the earth is flat—a claim now disputed in some quarters. In reality, the resolution is much lower in some parts of the earth.)

Figure 20 shows the result of applying the Superface algorithm with $\varepsilon = 32$ meters. The approximation consists of 53.2 percent of the original 128,522 triangles, and it has mean and maximum errors of 1.27 and 26.83 meters. Running time was 3 minutes, 56 seconds.

With this data set, aggressive border straightening was not as useful as in the previous examples. With $\varepsilon = 32$ meters, the aggressive straightening gives a mesh with 32.25 percent of the original triangles, and while the mean error of 4.63 meters is still quite reasonable, the maximum error jumps to 387.79 meters.

Comparing simplification algorithms

Table 5 gives a general idea of the range of running times for different algorithms and, in particular, how these algorithms compare in simplifying meshes of similar sizes to the some of the meshes we have simplified with the Superfaces algorithm.

The obvious caveat here is to avoid overinterpreting these results. Making meaningful comparisons is not a simple task, first because there is no consistency in the

Table 3. Results of simplifying the femur mesh of 179,916 triangles.

Error Bound ε	Approximation Error Mean	Approximation Error Max.	Triangles Count	Triangles Percent of Original	Running Time (m:ss)
0.5	0.0378	0.4826	97,010	53.92	4:54
1.0	0.1060	0.8821	66,318	36.86	3:31
1.5	0.1708	1.3265	46,748	25.98	3:10
2.0	0.2263	1.7860	36,018	20.02	2:54
2.5	0.2745	2.2530	30,766	17.10	2:45
3.0	0.3196	2.7049	26,832	14.91	2:36
4.0	0.3921	3.5481	21,840	12.14	2:28

Table 4. Results of simplifying the femur mesh of 179,916 triangles—with aggressive border straightening.

Error Bound ε	Approximation Error Mean	Approximation Error Max.	Triangles Count	Triangles Percent of Original	Running Time (m:ss)	Vertices above ε limit	Superfaces to adjust/ total superfaces
0.5	0.0733	0.9509	56,294	31.29	4:22	12	10/ 18,792
1.0	0.1797	2.1814	28,216	15.68	3:07	17	8/ 7,599
1.5	0.2778	1.6823	19,348	10.75	2:46	2	2/ 4,908
2.0	0.4000	2.3609	12,762	7.09	2:33	42	4/ 3,016
3.0	0.5516	3.7760	9,046	5.03	2:21	59	2/ 2,152
4.0	0.6797	4.1972	7,942	4.41	2:17	4	1/ 1,885

hardware or in the test data sets used in published experiments. Moreover, the algorithms do not all solve the same problem. For example, Table 5 shows the triangle decimation, mesh optimization, and multiresolution approximation algorithms all having faster running times than the Superface algorithm for an approximately 300,000-triangle model, but the first two methods do not guarantee an error bound and the third one does not preserve topology.

Nevertheless, we have compiled these summary results to give a rough idea of the relative speed of different algorithms. Efficiency was an important design factor in developing the Superfaces algorithm, and we believe we have succeeded in producing a fast algorithm.

Mesh simplification and rapid prototyping

Rapid prototyping systems are used to fabricate physical 3D parts from computer-based geometric models. This "3D printing" process has many of the same problems in handling large meshes as are encountered with standard graphics rendering and 2D printing: Large meshes take a long time to process, and very large meshes cannot be processed at all.

The Rapid Prototyping System (RPS) recently developed at IBM Research can currently process models with up to about 50,000 faces. We have used the Superfaces algorithm to facilitate the fabrication of much larger models. Figure 21 shows a photograph of a part produced by RPS from a simplified model of the skull described in our experimental results.

Future work

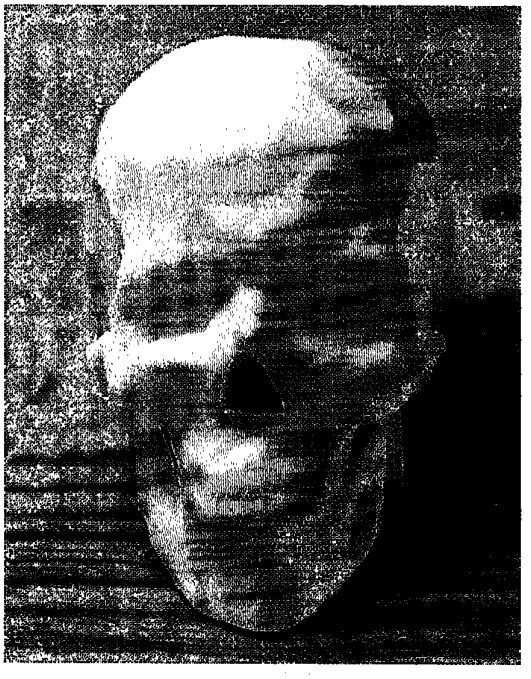
One limitation of the current version of the Superfaces algorithm (as well as of all the other simplification methods discussed earlier) is that the resulting

19 Map of topographic data of the earth.

20 Simplified map of topographic data of the earth: ε = 32.0 meters.

Table 5. Running times of different mesh simplification algorithms.

Algorithm Hardware Configuration	Triangles Original Mesh	Reduced Mesh	Running Times (seconds)
Superfaces	30,876	2,038	27
IBM RS/6000 (model 550)	179,916	21,840	148
	349,792	24,170	360
Triangle decimation[12]	38,394	17,799	8
SGI Onyx Reality Engine	186,630	71,485	43
2-processor model	334,643	84,342	90
	1,049,476	29,507	322
Geometric optimization[10]	315,812	295,636	96
(hardware not specified)	1,019,373	642,204	538
Mesh optimization[7]	3,832	432	600
DEC Alpha	18,272	1,348	2,820
Multiresolution approximation[9] IBM RS/6000 (model 560)	349,792	N/A	80



21 3D hard copy of a simplified skull (9,820 triangles).

simplified mesh can self-intersect when the original mesh does not—that is, two triangles from different superfaces might intersect. We still need to extend the algorithm to detect and avoid intersections.

We are currently working on improving the "gerrymandering" strategy to produce more regularly shaped superfaces and to eliminate "island" superfaces that are completely surrounded by a single neighboring superface.

Our experimental results show that significant improvements are possible by using the more aggressive strategy for border straightening in phase 2. The extra expense of this approach comes from

- identifying those vertices in the original mesh that are not within ε-tolerance of the simplified mesh, and
- splitting the edges of the superfaces that subsume these vertices.

The experimental results indicate that very few vertices lie beyond the allowable limit and that very little edge splitting will be required. We plan to complete implementation of the aggressive straightening method soon.

We also plan to extend the algorithm to other types of simplifications. For example, we can produce a simplified mesh that always lies inside (or outside) the original model simply by eliminating one of the two planarity rule constraints. More generally, we can use the algorithm to approximate different kinds of geometric models by changing the merging constraints. For example, models with curved surfaces can be simplified by replacing the planarity rule with a rule that produces superfaces approximating higher order surface patches.

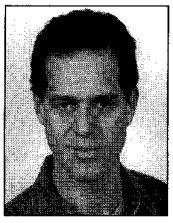
Another possible extension applies the Superface algorithm to the problem of mesh smoothing. By using higher order approximating surfaces, rather than approximating planes, we can produce a simplified mesh that is a curved approximation of the original mesh. The original mesh can then be smoothed, with ε-tolerance, by projecting each of its vertices onto the nominal approximating surface of its subsuming superface.

Since the vertices of a simplified mesh are a subset of the vertices of the original one, we have a natural way of using the Superfaces algorithm to construct a multiresolution representation of nested simplifications with graded degrees of detail at each level in the hierarchy. ■

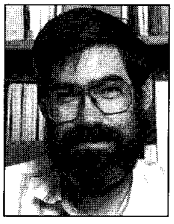
References

1. F.J.M. Schmitt, B. Barsky, and W.-H. Du, "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data," *Computer Graphics* (Proc. Siggraph), Vol. 20, No. 4, 1986, pp. 179-188.
2. M.J. DeHaemer, Jr. and M.J. Zyda, "Simplification of Objects Rendered by Polygonal Approximations," *Computer Graphics*, Vol. 15, No. 2, 1991, pp. 175-184.
3. A.D. Kalvin et al., "Constructing Topologically Connected Surfaces for the Comprehensive Analysis of 3D Medical Structures," in *Medical Imaging V: Image Processing*, SPIE Proc. Conf. 1445, SPIE, Bellingham, Wash., 1991, pp. 247-258.
4. W.J. Schroeder, J.A. Zarge, and W.E. Lorensen, "Decimation of Triangle Meshes," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 65-70.
5. B. Hamann, "A Data Reduction Scheme for Triangulated Surfaces," *Computer Aided Geometric Design*, Vol. 11, No. 2, Apr. 1994, pp. 197-214.
6. G. Turk, "Re-tiling of Polygonal Surfaces," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 55-64.
7. H. Hoppe et al., "Mesh Optimization," *Computer Graphics* (Proc. Siggraph), Vol. 17, No. 3, Aug. 1983, pp. 19-25.
8. A. Guéziec and D. Dean, "The Wrapper Algorithm: A Surface Optimization Algorithm That Preserves Highly Curved Areas," in *Visualization in Biomedical Computing 94*, SPIE,

Bellingham, Wash., 1994, pp. 631-642.
9. J. Rossignac and P. Borrel, "Multi-resolution 3D Approximations for Rendering Complex Scenes," in *Modeling in Computer Graphics*, B. Falcidieno and T. L. Kunii, eds., Springer-Verlag, Berlin, 1993, pp. 455-465.
10. P. Hinker and C. Hansen, "Geometric Optimization," *Proc. Visualization 93*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 189-195.
11. A. D. Kalvin and R. H. Taylor, "Superfaces: Polyhedron Approximation with Bounded Error," Research Report RC 19135, I.B.M. Thomas J. Watson Research Center, Yorktown Heights, New York, Apr. 1993.
12. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics* (Proc. Sigggraph), Vol. 21, No. 3, July 1987, pp. 311-317.
13. J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, 1987.
14. W.E. Lorensen, "Marching Through the Visible Man," on the World Wide Web at http://www.ge.com/crd/ivl/vm/vm.html.



Alan Kalvin *is a research staff member in the Computer-Assisted Surgery Group at the IBM T.J. Watson Research Center. His research interests include medical imaging, geometric modeling, computer-assisted anthropology and archaeology, and computer graphics. Kalvin received a BSc degree from the University of the Witwatersrand, South Africa, a BSc Honors degree from the University of Cape Town, South Africa, and MS and PhD degrees in computer science from the Courant Institute of Mathematical Sciences, New York University, in 1975, 1976, 1985, and 1991, respectively.*



Russell H. Taylor *became a professor of computer science at Johns Hopkins University in September 1995. From 1976 to 1995, he was a research staff member and research manager at IBM T.J. Watson Research Center. His research interests include robot systems, programming languages, model-based planning, and (most recently) the use of imaging, model-based planning, and robotic systems to augment human performance in surgical procedures. Taylor received a BES degree from Johns Hopkins University in 1970 and a PhD in computer science from Stanford in 1976. He is editor emeritus of the IEEE Transactions on Robotics and Automation, a Fellow of the IEEE, and a member of various honorary societies, panels, program committees, and advisory boards.*

Readers may contact Kalvin at IBM T.J. Watson Research Center, PO Box 704 Yorktown Heights, NY 10598, e-mail kalvin@watson.ibm.com and Taylor at Computer Science Dept., Johns Hopkins University, New Engineering Bldg. 224, 3400 N. Charles St., Baltimore, MD 21218, e-mail rht@cs.jhu.edu.