International Journal of Computational Geometry & Applications

SUCCESSIVE MAPPINGS: AN APPROACH TO POLYGONAL MESH SIMPLIFICATION WITH GUARANTEED ERROR BOUNDS

JONATHAN COHEN, * DINESH MANOCHA, † AND MARC OLANO ‡
Department of Computer Science
Sitterson Hall, CB 3175
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175, USA

ABSTRACT

We present the use of mapping functions to automatically generate levels of detail with known error bounds for polygonal models. We develop a piece-wise linear mapping function for each simplification operation and use this function to measure deviation of the new surface from both the previous level of detail and from the original surface. In addition, we use the mapping function to compute appropriate texture coordinates if the original model has texture coordinates at its vertices. Our overall algorithm uses edge collapse operations. We present rigorous procedures for the generation of local orthogonal projections to the plane as well as for the selection of a new vertex position resulting from the edge collapse operation. The algorithm computes guaranteed error bounds on surface deviation and produces an entire continuum of levels of detail with mappings between them. We demonstrate the effectiveness of our algorithm on several models: a Ford Bronco consisting of over 300 parts and 70,000 triangles, a textured lion model consisting of 49 parts and 86,000 triangles, a textured, wrinkled torus consisting of 79,000 triangles, a dragon model consisting of 871,000 triangles, a Buddha model consisting of 1,000,000 triangles, and an armadillo model consisting of 2,000,000 triangles.

Keywords: surface approximation, simplification, levels-of-detail, orthogonal projection, mapping, error bounds, linear programming

1. Introduction

Automatic generation of levels of detail for polygonal data sets has become a task of fundamental importance for real-time rendering of large polygonal environments on current graphics systems. Detailed models are obtained by a variety of methods, including range scanning of physical objects and modeling of new objects within CAD systems. In addition to surface geometry, these models often contain additional information such as normals, texture coordinates, color etc. As such

*Department of Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218-2694, USA, cohen@cs.jhu.edu

†Department of Computer Science Sitterson Hall, CB 3175 University of North Carolina at Chapel Hill Chapel Hill, North Carolina 27599-3175, USA, dm@cs.unc.edu

‡M/S 590, Silicon Graphics Incorporated, 2011 North Shoreline Boulevard, Mountain View, California, 94043, olano@engr.sgi.com

models become commonplace, many applications desire high quality simplifications, with error bounds of various types across the surface being simplified.

Most of the literature on simplification has focused purely on surface approximation. Many of these techniques give guaranteed error bounds on the deviation of the simplified surface from the original surface. Such bounds are useful for providing a measure of the screen-space deviation from the original surface. A few techniques have been proposed to preserve other attributes such as color or overall appearance. However, they are not able to give tight error bounds on these parameters. At times the errors accumulated in all these domains may cause visible artifacts, even though the surface deviation itself is properly constrained. We believe the most promising approach to measuring and bounding these attribute errors is to have a mapping between the original surface and the simplified surface. With such a mapping in hand, we are free to devise suitable methods for measuring and bounding each type of error.

Main Contribution: In this paper we present a new simplification algorithm, which computes a piece-wise linear mapping between the original surface and the simplified surface (portions of this work appear in [10] and extensions appear in [8]). The algorithm uses the edge collapse operation due to its simplicity, local control, and suitability for generating smooth transitions between levels of detail. We also present rigorous and complete algorithms for collapsing an edge to a vertex such that there are no local self-intersections and a one-to-one mapping is guaranteed. The algorithm keeps track of both incremental surface deviation from the current level of detail and total deviation from the original surface. The main features of our approach are:

1. **Successive Mapping:** This mapping between the levels of detail is a useful tool. We currently use the mapping in several ways: to measure the distance between the levels of detail before and after an edge collapse, to choose a location for the generated vertex that minimizes this distance, to accumulate an upper bound on the distance between the new level of detail and the original surface, and to map surface attributes to the simplified surface.

2. **Guaranteed Error Bounds:** Our approach can measure and minimize the incremental error for surface deviation (ultimately bounding the total surface deviation) and is extendible to other attributes. These error bounds give guarantees on the shape of the simplified object and screen-space deviation.

3. **Generality:** The algorithm for collapsing an edge into a vertex is rather general and does not restrict the vertex to lie on the original edge. Furthermore, portions of our approach can be easily combined with other algorithms, such as simplification envelopes[9].

4. **Surface Attributes:** Given an original surface with texture coordinates, our algorithm uses the successive mapping to compute appropriate texture coordinates for the simplified mesh. We have recently extended our approach to provide guarantees on the final shaded appearance of the simplified mesh

by maintaining colors and normals in texture and normal maps and bounding the deviation of texture coordinates[8]. Our approach can also be used to bound the error of any associated scalar fields[44].

5. **Continuum of Levels of Details:** The algorithm incrementally produces an entire spectrum of levels-of-details as opposed to a few discrete levels; the algorithm incrementally stores the error bounds for each level. Thus, the simplified model can be stored as a progressive mesh[28] if desired.

The algorithm has been successfully applied to a number of models. These models consist of hundreds of parts and millions of polygons, including a Ford Bronco with 300 parts, textured models of a lion and a wrinkled torus, and highly-tessellated scanned models such as a Buddha statue, a dragon, and a toy armadillo.

Organization: The rest of the paper is organized as follows. In Section 2, we survey related work on model simplification. We give an overview of our algorithm in Section 3. Section 4 discusses the creation of local mappings for the purpose of collapsing edges. Using these mappings, we minimize the incremental surface deviation error and bound the total deviation in Section 5. Section 6 describes how to compute new texture coordinates for the new mesh vertices. The implementation is discussed in Section 7 and its performance in Section 8. In Section 9 we compare our approach to some other algorithms, and we conclude with some directions for future research in Section 10. Appendix A provides more detail on the mathematical underpinings of our projection-based mapping algorithms.

2. Previous Work

Automatic simplification has been studied in both the computational geometry[1,3,7,12,39] and computer graphics literature[2,5,9,13,14,16,21,24,28,27,42,43,44,45,46,48,49,51] for several years. Several informative surveys are available on the subject[17,25].

It has been shown that computing the minimum-complexity simplification for a given error bound is NP-hard for both convex polytopes[12] and polyhedral terrains[1]. Thus, simplification algorithms have evolved around finding polynomial-time approximations that are close to optimal or employ efficient, greedy heuristics.

Some of the earlier work in computer graphics by Turk[49] and Schroeder[46] employs heuristics based on curvature to determine which parts of the surface to simplify to achieve a model with the desired polygon count. One interesting aspect of Turk's presentation is the description of the topological constraints on a vertex for its removal to preserve the local topology of a mesh. More recently, Dey et al.[15] provide a formal mathematical description of the topological constraints on an edge for its collapse to similarly preserve the local mesh topology.

Other early work includes that of Rossignac and Borrel[43], where vertices close to each other are clustered and a vertex is generated to represent them. This algorithm has been used in the *Brush* walkthrough system[45].

Hoppe et al.[27,28] posed the model simplification problem into a global optimization framework, minimizing the least-squares error from a set of point-samples on the original surface. Later, Hoppe extended this framework to handle other

scalar attributes, explicitly recognizing the distinction between smooth gradients and sharp discontinuities. He also introduced the progressive mesh[28], which is essentially a stored sequence of simplification operations, allowing quick construction of any desired level of detail along the continuum of simplifications. However, this algorithm provides no guaranteed error bounds. The process measures the distance from a set of points on the original surface to the resulting simplified surface, but not from the entire original surface to the simplified surface.

An efficient approach to measuring error as the distance between the simplified vertices and the planes of the original surface is presented in [42] and further refined in [19] to represent the error as a quadratic form. Although this error measure also does not bound the surface-to-surface distance from the original to the simplified model, it provides a fast metric to guide the simplification process. Lindstrom and Turk[36,37] have experimented with a purely local variant of this approach, incorporating volume preservation as well, demonstrating favorable results. The demonstration involves a post-simplification measure of the actual error in the simplified models. The error quadric approach has also been extended to measure the error of other attributes, such as vertex colors and normals.[20,29].

There is considerable literature on model simplification providing guaranteed surface-to-surface error bounds, which is an important component of this paper. Cohen and Varshney et al.[9,50] have used envelopes to preserve the model topology and obtain tight error bounds for a single simplification (within about 2 percent[6]), but they do not produce an entire spectrum of levels of detail. Klein[32,31] and Kobbelt et al. [33] measure a one-sided Hausdorff distance between the original and simplified surfaces. This measure can produce tighter bounds than the mapping-based measure we present, but the one-sided formulation does not provide a true guarantee of surface-to-surface distance. Guéziec[21,22] has presented an algorithm for computing local error bounds inside the simplification process by maintaining tolerance volumes. This approach optimizes the simplified vertices to preserve volume. However, the approach described does not generate a mapping between levels of detail. Bajaj and Schikore[2,44] have presented an algorithm for producing a mapping between approximations and measure the error of scalar fields across the surface based on vertex-removals. Some of the results presented in this paper extend this work non-trivially to the edge collapse operation. A detailed comparison with some of these approaches is presented in Section 9.

An elegant solution to the polygon simplification problem has been presented[14,16] in which arbitrary polygonal meshes are first subdivided into patches with *subdivision connectivity* and then multiresolution wavelet analysis is used over each patch. These methods preserve global topology, give error bounds on the simplified object and provide a mapping between levels of detail. They have been further extended[5] to handle colored meshes. However, the initial mesh is not contained in the level of detail hierarchy, but can only be recovered to within an ϵ-tolerance. In some cases this is undesirable. Furthermore, the wavelet based approach can be somewhat conservative and for a given error bound; algorithms based on vertex removal and edge collapses[9,28] have been *empirically* able to simplify more (in terms of reducing

the polygon count). This problem has recently been alleviated somewhat by Lee et al.[35]. Their approach allows the specification of feature constraints, such as sharp edges, before the simplification begins. These constraints affect the initial shape of the subdivision patches. Still, the simplification process is locked into a fairly rigid optimization path from this point onward.

The field of simplification has grown to be quite diverse, including research areas which are beyond the scope of this paper. In particular, several simplification algorithms allow dynamic, view-dependent simplification of objects or scenes during an interactive visualization session.[18,26,30,38,52] Recently, Guskov et al.[23] have even described simplification as a member of a suite of signal processing operations designed for operation on meshes.

3. Overview

Our simplification approach may be seen as a high-level algorithm which controls the simplification process with a lower-level cost function based on local mappings. Next we describe this high-level control algorithm and the idea of using local mappings for cost evaluation.

3.1. High-level Algorithm

At a broad level, our simplification algorithm is a generic greedy algorithm. Our simplification operation is the edge collapse. We initialize the algorithm by measuring the cost of all possible edge collapses, then we perform the edge collapses in order of increasing cost. The cost function represents local error bounds on surface deviation and other attributes. After performing each edge collapse, we locally re-compute the cost functions of all edges whose neighborhoods were affected by the collapse. This process continues until none of the remaining edges can be collapsed.

The output of our algorithm is the original model plus an ordered list of edge collapses and their associated cost functions. This *progressive mesh*[28] represents an entire *continuum* of levels of detail for the surface. Section 7.2 discusses how we use these levels of detail to render the model with the desired quality or speed-up.

3.2. Local Mappings

The edge collapse operation we perform to simplify the surface contracts an edge (the *collapsed edge*, e) to a single, new vertex (the *generated vertex*, v_{gen}). Most of the earlier algorithms position the generated vertex to one of the end vertices or mid-point of the collapse edge. These choices for the generated vertex position are reasonable heuristics, and may reduce storage overhead. However, these choices may not minimize the surface deviation or other attribute error bound and can result in a local self-intersection. We choose a vertex position in two dimensions to avoid local self-intersections and optimize in the third dimension to minimize incremental error. This optimization of the generated vertex position and measurement of the error are the keys to simplifying the surface without introducing significant error.

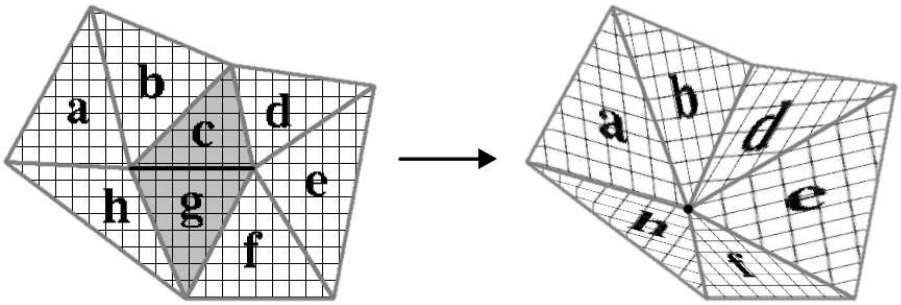

Figure 1: The natural mapping primarily maps triangles to triangles. The two grey triangles map to edges, and the collapsed edge maps to the generated vertex

For each edge collapse, we consider only the neighborhood of the surface that is modified by the operation (i.e. those faces, edges and vertices adjacent to the collapsed edge). There is a *natural mapping* between the neighborhood of the collapsed edge and the neighborhood of the generated vertex (see Figure 1). Most of the triangles incident to the collapsed edge are stretched into corresponding triangles incident to the generated vertex. However, the two triangles that share the collapsed edge are themselves collapsed to edges. These natural correspondences are one form of mapping.

This natural mapping has two weaknesses.

1. The degeneracy of the triangles mapping to edges prevents us from mapping points of the simplified surface back to unique points on the original surface. This also implies that if we have any sort of attribute field across the surface, a portion of it disappears as a result of the operation.

2. The error implied by this mapping may be larger than necessary.

We measure the surface deviation error of the edge collapse operation as the distances between corresponding points of our mapping. Using the natural mapping, the maximum distance between any pair of corresponding points is defined as:

$$E = \max(distance(v_1, v_{gen}), distance(v_2, v_{gen})), \tag{1}$$

where v_1 and v_2 are the vertices of e.

If we place the generated vertex at the midpoint of the collapsed edge, this distance error will be half the length of the edge. If we place the vertex at any other location, the error will be even greater.

We can create mappings that are free of degeneracies and often imply less error than the natural mapping. For simplicity, and to guarantee no local self-intersections, we perform our mappings using orthogonal projections of our local neighborhood to the plane. Because they are applied one after another as we simplify the mesh, we refer to them as *successive mappings*.

4. Successive Mapping

In this section we present an algorithm to compute the mappings we use to compute error bounds and to guide the simplification process. We present efficient and complete algorithms for computing a planar projection, finding a generated vertex in the plane, and creating a mapping in the plane. These algorithms employ well-known techniques from computational geometry and are efficient in practice. The correctness of these algorithms is proven in Appendix A.

4.1. Computing a Planar Projection

Given a set of triangles in 3D, we present an efficient algorithm to compute a planar projection which is *fold-free*. Such a fold-free projection contains no pair of edge-adjacent triangles which overlap in the plane. This fold-free characteristic is a necessary, but not sufficient, condition for a projection to provide a *one-to-one mapping* between the set of triangles and a portion of the plane. In practice, most fold-free projections provide such a one-to-one mapping. We later perform an additional test to verify that our fold-free projection is indeed one-to-one (see Section 4.3).

The projection we seek should be one-to-one to guarantee that the operations we perform in the plane are meaningful. For example, suppose we project a connected set of triangles onto a plane and then re-triangulate the polygon described by their boundary. The resulting set of triangles will contain no self-intersections, so long as the projection is one-to-one. Many other simplification algorithms, such as those by Turk[49], and Schroeder[46] also use such projections for vertex removal. However, they simply choose a likely direction, such as the average of the normal vectors of the triangles of interest. To test the validity of the resulting projection, these earlier algorithms project all the triangles onto the plane and check for self-intersections. This process can be relatively expensive and does not provide a robust method for finding a one-to-one projecting plane.

We improve on earlier brute-force approaches in two ways. First, we present a simple, linear-time algorithm for testing the validity of a given direction, ensuring that it produces a fold-free projection. Second, we present a slightly more complex, but still expected linear-time, algorithm which will find a valid direction if one exists, or report that no such direction exists for the given set of triangles. We defer until Section 4.3 a final, linear-time test to guarantee that our fold-free projection provides a one-to-one mapping.

4.1.1. Validity Test for Planar Projection

In this section, we briefly describe the algorithm which determines whether a given set of triangles has a fold-free planar projection. (Note that this fold-free projection may not be one-to-one in rare cases - see Appendix A for a more detailed discussion). Assume that we can calculate a consistent set of normal vectors for the set of triangles in question (if we cannot, the local surface is non-orientable and cannot be mapped onto a plane in a one-to-one fashion). If the angle between a given

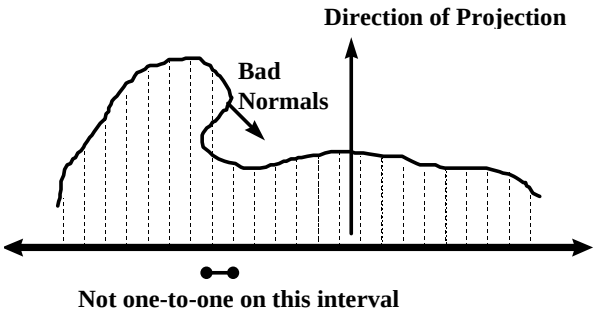

Figure 2: A 2D example of an invalid projection due to folding.

direction of projection and the normal vector of each of the triangles is less than 90^o, then the direction of projection is valid and defines a fold-free mapping from the 3D triangles to a set of triangles in the plane of projection (any plane perpendicular to the direction of projection). Note that for a given direction of projection and a given set of triangles, this test involves only a single dot product and a sign test for each triangle in the set. The correctness of this test is demonstrated in Appendix A.

To develop some intuition, we examine a 2D version of our problem, shown in Figure 2. We would like to determine if the projection of the curve onto the line is fold-free. Without loss of generality, assume the direction of projection is the y-axis. Each point on the curve projects to its x-coordinate on the line. If we traverse the curve from its left-most endpoint, we will project onto a previously projected location if and only if we reverse our direction along the x-axis. This can only occur when the y-component of the curve's normal vector goes from a positive value to a negative value. This is equivalent to our statement that the invalid normal will be more than 90^o from the direction of projection.

4.1.2. Finding a valid direction

The validity test in the previous section provides a quick method of testing the validity of a likely direction as a fold-free projection (such as the average normal of the local triangles). Unfortunately, the wider the spread of the normal vectors of our set of triangles, the less likely we are to find a valid direction by using any sort of heuristic. It is possible, in fact, to compute the set of all valid directions of projection for a given set of triangles. However, to achieve greater efficiency and to reduce the complexity of the software system, we choose to find only a single valid direction, which is typically all we require.

The *Gaussian sphere*[4] is the unit sphere on which each point corresponds to a unit normal vector with the same coordinates. Given a triangle, we define a plane through the origin with the same normal as the triangle. For a direction of projection to be valid with respect to this triangle, its point on the Gaussian sphere must lie on the correct side of this plane (i.e. within the correct hemisphere). If we consider two triangles simultaneously (shown in 2D in Figure 3) the direction of projection must

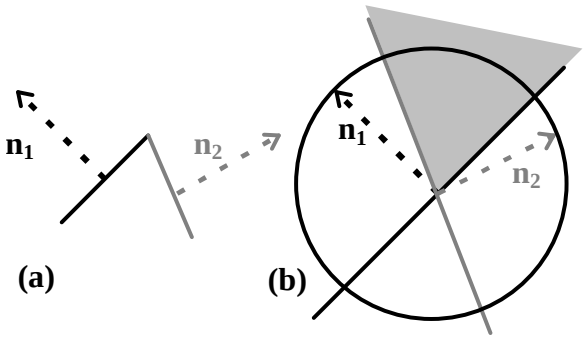

Figure 3: A 2D example of the valid projection space. (a) Two line segments and their normals. (b) The 2D Gaussian circle, the planes corresponding to each segment, and the space of valid projection directions (shaded in grey).

lie on the correct side of each of the two planes determined by the normal vectors of the triangles. This is equivalent to saying that the valid directions lie within the intersection of half-spaces defined by these two planes. Thus, the valid directions of projection for a set of N triangles lie within the intersection of N half-spaces.

This intersection of half-spaces forms a convex polyhedron. This polyhedron is a cone, with its apex at the origin and an unbounded base (shown as a shaded, triangular region in Figure 3). We can force this polyhedron to be bounded by adding more half-spaces (we use the six faces of a cube containing the origin). By finding a point on the interior of this cone and normalizing its coordinates, we shall construct a unit vector in a valid direction of projection.

Rather than explicitly calculating the boundary of the cone, we simply find a few corners (vertices) and average them to find a point that is strictly inside. By construction, the origin is definitely such a corner, so we just need to find three more *unique* (and linearly independent) corners to calculate an interior point. We can find each of these corners by solving a 3D *linear programming* problem (described below). Linear programming allows us to find a point that maximizes a linear objective function subject to a collection of linear constraints[34]. The equations of the half-spaces serve as our linear constraints. We maximize in the direction of a vector to find the corner of our cone that lies the farthest in that direction.

As stated above, the origin is our first corner. To find the second corner, we try maximizing in the positive-x direction. If the resulting point is the origin, we instead maximize in the negative-x direction. To find the third corner, we maximize in a direction orthogonal to the line containing the first two corners. If the resulting point is one of the first two corners, we maximize in the opposite direction. Finally, we maximize in a direction orthogonal to the plane containing the first three corners. Once again, we may need to maximize in the opposite direction instead. Note that it is possible to reduce the worst-case number of optimizations from six to four by using the triangle normals to guide the selection of optimization vectors.

We used Seidel's linear time randomized algorithm[47] to solve each linear pro-

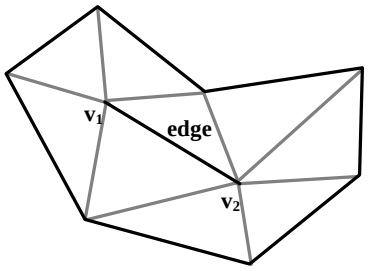

Figure 4: The neighborhood of an edge as projected into 2D

gramming problem. A public domain implementation of this algorithm by Hohmeyer is available. It is very fast in practice.

4.2. Placing the Vertex in the Plane

In the previous section, we presented an algorithm to compute a valid plane. The edge collapse, which we use as our simplification operation, merges the two vertices of a particular edge into a single vertex. The topology of the resulting mesh is completely determined, but we are free to choose the position of the vertex, which will determine the geometry of the resulting mesh.

When we project the triangles neighboring the given edge onto a valid plane of projection, we get a triangulated polygon with two interior vertices, as shown in Figure 4. The edge collapse will reduce this edge to a single vertex. There will be edges connecting this generated vertex to each of the vertices of the polygon. We would like the set of triangles around the generated vertex to have a one-to-one mapping with our chosen plane of projection, and thus to have a one-to- one mapping with the original edge neighborhood as well.

In this section, we present linear time algorithms both to test a candidate vertex position for validity, and to find a valid vertex position, if one exists.

4.2.1. Validity test for Vertex Position

The edge collapse operation leaves the boundary of the polygon in the plane unchanged. For the neighborhood of the generated vertex to have a one-to-one mapping with the plane, its edges must lie entirely within the polygon, ensuring that no edge crossings occur.

This 2D visibility problem has been well-studied in the computational geometry literature[40]. The generated vertex must have an unobstructed line of sight to each of the surrounding polygon vertices (unlike the vertex shown in Figure 5(a)). This condition holds if and only if the generated vertex lies within the polygon's *kernel*, shown in Figure 5(b). This kernel is the intersection of inward-facing half-planes defined by the polygon's edges.

Given a candidate position for the generated vertex in 2D, we test its validity by plugging it into the implicit-form equation of each of the lines containing the

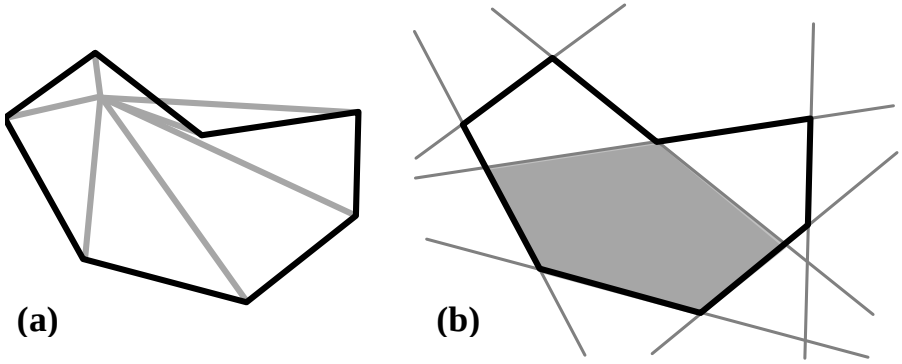

Figure 5: (a) An invalid 2D vertex position. (b) The kernel of a polygon is the set of valid positions for a single, interior vertex to be placed. It is the intersection of a set of inward half-spaces.

polygon's edges. If the position is on the interior with respect to each line, the position is valid; otherwise it is invalid.

4.2.2. Finding a Valid Position

The validity test described above is useful if we wish to test out a likely candidate for the generated vertex position, such as the midpoint of the edge being collapsed. If such a heuristic choice succeeds, we can avoid the work necessary to compute a valid position directly.

Given the kernel definition for valid points, it is straightforward to find a valid vertex position using 2D linear programming. Each of the lines provides one of the constraints for the linear programming problem. Using the same methods as in Section 4.1.2, we can find a point in the kernel with no more than four calls to the linear programming routine. The first and second corners are found by maximizing in the positive- and negative-x directions. The final corner is found using a vector orthogonal to the first two corners.

4.3. Guaranteeing a One-to-One Projection

While rare in practice, it is possible in theory for us to find both a fold-free projection and a vertex position within the planar polygon's kernel, yet still have a projection which is not one-to-one. Figure A.5 shows an example of such a projection.

As proved in Appendix A, we can verify that both the neighborhoods of the generated vertex and the collapsed edge have one-to-one projections with the plane with a simple, linear-time test. Given our edge, e, its two vertices, v_1 and v_2, and the generated vertex, v_{gen}, these projections are one-to-one if and only if the orientations of the triangles surrounding the v_{gen} are consistent and the triangles surrounding v_1, v_2, and v_{gen} each cover angular ranges in the plane which sum to 2π.

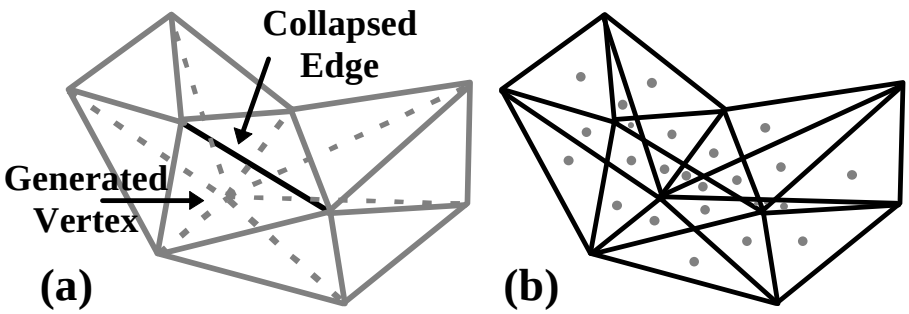

Figure 6: (a) Edge neighborhood and generated vertex neighborhood superimposed. (b) A mapping in the plane, composed of 25 polygonal cells (each cell contains a dot). Each cell maps between a pair of planar elements in 3D.

We can verify the orientations of v_{gen}'s triangles by performing a single cross product for each triangle. If the signed areas of all the triangles have the same sign, they are consistently oriented, and the projections are one-to-one. We verify the angular sums of triangles surrounding v_1, v_2, and v_{gen} using a vector normalization, dot product, and arccos operation for each triangle to compute its angular range. Each floating point sum will be within some small tolerance of an integer multiple of 2π, with 1 being the valid multiplier.

4.4. Creating a Mapping in the Plane

After mapping the edge neighborhood to a valid plane and choosing a valid position for the generated vertex, we define a mapping between the edge neighborhood and the generated vertex neighborhood. We shall map to each other the pairs of 3D points which project to identical points on the plane. These correspondences are shown in Figure 6(a) by superimposing these two sets of triangles in the plane.

We can represent the mapping by a set of map cells, shown in Figure 6(b). Each cell is a convex polygon in the plane and maps a piece of a triangle from the edge neighborhood to a similar piece of a triangle from the generated vertex neighborhood. The mapping represented by each cell is linear.

The vertices of the polygonal cells fall into four categories: vertices of the overall polygon in the plane, vertices of the collapsed edge, the generated vertex itself, and edge-edge intersection points. We already know the locations of the first three categories of cell vertices, but we must calculate the edge-edge intersection points explicitly. Each such point is the intersection of an edge adjacent to the collapsed edge with an edge adjacent to the generated vertex. The number of such points can be quadratic (in the worst case) in the number of neighborhood edges. If we choose to construct the actual cells, we may do so by sorting the intersection points along each neighborhood edge and then walking the boundary of each cell. However, this is not necessary for computing the surface deviation.

5. Measuring Surface Deviation Error

Up to this point, we have projected the collapsed edge neighborhood onto a plane, collapsed the edge to the generated vertex in this plane, and computed a mapping in the plane between these two local meshes. The generated vertex has not yet been placed in 3D. We will choose its 3D position to minimize the incremental error in surface deviation.

Given the overlay in the plane of the collapsed edge neighborhood, $\mathcal{M}_{i-1}$, and the generated vertex neighborhood, $\mathcal{M}_i$, we define the *incremental surface deviation* between them:

$$E_{i,i-1}(x) = \|F_i^{-1}(x) - F_{i-1}^{-1}(x)\| \tag{2}$$

The function, $F_i : \mathcal{M}_i \rightarrow \mathcal{P}$, maps points on the 3D mesh, $\mathcal{M}_i$, to points, x, in the plane. $F_{i-1} : \mathcal{M}_{i-1} \rightarrow \mathcal{P}$ acts similarly for the points on $\mathcal{M}_{i-1}$. $E_{i,i-1}$ measures the distance between the pair of 3D points corresponding to each point, x, in the planar overlay.

Within each of our polygonal mapping cells in the plane, the incremental deviation function is linear, so the maximum incremental deviation for each cell occurs at one of its boundary points. Thus, we bound the incremental deviation using only the deviation at the cell vertices, V:

$$E_{i,i-1}(\mathcal{P}) = \max_{x \in \mathcal{P}} E_{i,i-1}(x) = \max_{v_k \in V} E_{i,i-1}(v_k) \tag{3}$$

5.1. Distance Functions of the Cell Vertices

To preserve our one-to-one mapping, it is necessary that all the points of the generated vertex neighborhood, including the generated vertex itself, project back into 3D along the direction of projection (the normal to the plane of projection). This restricts the 3D position of the generated vertex to the line parallel to the direction of projection and passing through the generated vertex's 2D position in the plane. We choose the vertex's position along this line such that it minimizes the incremental surface deviation.

We parameterize the position of the generated vertex along its line of projection by a single parameter, t. As t varies, the distance between the corresponding cell vertices in 3D varies linearly. Notice that these distances will always be along the direction of projection, because the distance between corresponding cell vertices is zero in the other two dimensions (those of the plane of projection). The distance function for each cell vertex, v_k, has the form (see Figure 7):

$$E_{i,i-1}(v_k) = |m_k t + b_k|, \tag{4}$$

where m_k and b_k are the slope and y-intercept of the signed distance function for v_k as t varies.

5.2. Minimizing the Incremental Surface Deviation

Given the distance function, we would like to choose the parameter t that minimizes the maximum distance between any pair of mapped points. This point is

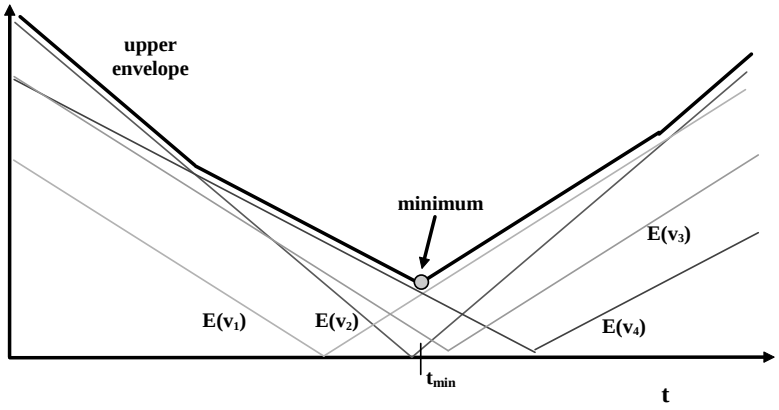

Figure 7: We parameterize the position of the generated vertex along the direction of projection by t. The incremental surface deviation at each cell vertex varies linearly with t, so the minimum of maximum deviations over all the cell vertices occurs at t_{min}, the value of t at the minimum of the *upper envelope*.

the minimum of the so-called *upper envelope*, shown in Figure 7. For a set of k functions, we define the upper envelope function as follows:

$$U(t) = \{f_i(t) \mid f_i(t) \ \geq \ f_j(t) \ \forall \ i,j \quad 1 \leq i,j \leq k; \quad i \neq j\}. \tag{5}$$

For linear functions with no boundary conditions, this function is convex. We convert the distance function for each cell vertex to two linear functions, then use linear programming to find the t value at which the minimum occurs. We use this value of t to calculate the 3D position for the generated vertex which minimizes the maximum incremental surface deviation.

5.3. Bounding Total Surface Deviation

While it is straightforward to measure the incremental surface deviation and choose the position of the generated vertex to minimize it, this is not the error we eventually store with the edge collapse. To know how much error the simplification process has created, we need to measure the *total surface deviation* of the mesh $\mathcal{M}_i$:

$$S_i(X) = E_{i,0}(F_i(X)) = \|X - F_0^{-1}(F_i(X))\| \tag{6}$$

Unfortunately, our projection formulation of the mapping functions provides only F_{i-1}^{-1} and F_i^{-1} when we are performing edge collapse i; it is more difficult to construct F_0^{-1}, and the complexity of this mapping is at least as complex as the original surface.

We approximate $E_{i,0}$ by using a set of axis-aligned boxes (other possible choices for these approximation volumes include triangle-aligned prisms and spheres). This provides a convenient representation of a bound on $S_i(X)$ that we can update from one simplified mesh to the next without having to refer to the original mesh. Each triangle, $\triangle_k$, in $\mathcal{M}_i$, has its own axis-aligned box, $b_{i,k}$ such that at every point on

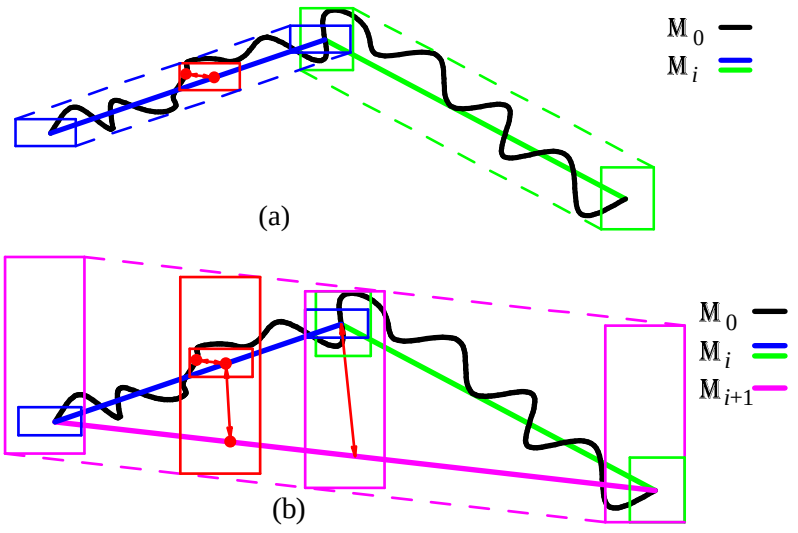

Figure 8: 2D illustration of the box approximation to total surface deviation. (a) A curve has been simplified to two segments, each with an associated box to bound the deviation. (b) As we simplify one more step, the approximation is propagated to the newly created segment.

the triangle, the Minkowski sum of the 3D point with the box gives a region that contains the corresponding point on the original surface.

$$\forall X \in \triangle_k, F_0^{-1}(F_i(X)) \in X \oplus b_{i,k} \tag{7}$$

Figure 8(a) shows an original surface (curve) and a simplification of it, consisting of two thick lines. Each line has an associated box. As the box slides over the line it is applied to each point along the way; the corresponding point on the original mesh is contained within the translated box. One such correspondence is shown halfway along the left line.

From (6) and (7), we produce $\tilde{S}_i(X)$, a bound on the total surface deviation, $S_i(X)$. This is the surface deviation error reported with each edge collapse.

$$\tilde{S}_i(X) = \max_{X' \in X \oplus b_{i,k}} \|X - X'\| \geq S_i(X) \tag{8}$$

$\tilde{S}_i(X)$ is the distance from X to the farthest corner of the box at X. This will always bound the distance from X to $F_0^{-1}(F_i(X))$. The maximum deviation over an edge collapse neighborhood is the maximum $\tilde{S}_i(X)$ for any cell vertex.

The boxes, $b_{i,k}$, are the only information we keep about the position of the original mesh as we simplify. We create a new set of boxes, $b_{i+1,k}$, for mesh $\mathcal{M}_{i+1}$ using an incremental computation (described in Figure 9). Figure 8(b) shows the propagation from $\mathcal{M}_i$ to $\mathcal{M}_{i+1}$. The two lines from Figure 8(a) have now been

PropagateError():
foreach cell vertex, v
    foreach triangle, $\triangle_{i-1}$, in $\mathcal{M}_{i-1}$ touching v
        foreach triangle, $\triangle_i$, in $\mathcal{M}_i$ touching v
            PropagateBox(v, $\triangle_{i-1}$, $\triangle_i$)
PropagateBox(v, $\triangle_{i-1}$, $\triangle_i$):
$X_{i-1} = F_{i-1}^{-1}(v), X_i = F_i^{-1}(v)$
Expand $\triangle_i$'s box so that $\triangle_i$'s box applied at X_i contains
    $\triangle_{i-1}$'s box applied at X_{i-1}

Figure 9: Pseudo-code to propagate the total deviation from mesh $\mathcal{M}_{i-1}$ to $\mathcal{M}_i$.

simplified to a single line. The new box, $b_{i+1,0}$, is constant as it slides across the new line. The size and offset is chosen so that, at every point of application, $b_{i+1,0}$ contains the box $b_{i,0}$ or $b_{i,1}$, as appropriate.

If X is a point on $\mathcal{M}_i$ in triangle k, and Y is the corresponding point on $\mathcal{M}_{i+1}$, the containment property of (7) holds:

$$F_0^{-1}(F_{i+1}(Y)) \in X \oplus b_{i,k} \subseteq Y \oplus b_{i+1,k'} \tag{9}$$

For example, all three dots in Figure 8(b) correspond to each other. The dot on original surface, $\mathcal{M}_0$ is contained in a small box, $X \oplus b_{i,0}$, which is contained in the larger box, $Y \oplus b_{i+1,0}$.

Because each mapping cell in the overlay between $\mathcal{M}_i$ and $\mathcal{M}_{i+1}$ is linear, we compute the sizes of the boxes, $b_{i+1,k'}$, by considering only the box correspondences at cell vertices. In Figure 8(b), there are three places we must consider. If $b_{i+1,0}$ contains $b_{i,0}$ and/or $b_{i,1}$ at all three places, it will contain them everywhere.

Together, the propagation rules, which are simple to implement, and the box-based approximation to the total surface deviation, provide the tools we need to efficiently provide a surface deviation for the simplification process.

5.4. Accommodating Bordered Surfaces

Bordered surface are those containing edges adjacent to only a single triangle, as opposed to two triangles. Such surfaces are quite common in practice. Borders create some complications for the creation of a mapping in the plane. The problem is that the total shape of the neighborhood projected into the plane changes as a result of the edge collapse.

Bajaj and Schikore[2], who employ a vertex-removal approach, deal with this problem by mapping the removed vertex to a length-parameterized position along the border. We employ this technique for the edge-collapse operation by a simple extension. In their case, a single vertex maps to a point on an edge. In ours, three vertices map to points on a chain of edges.

6. Computing Texture Coordinates

The use of texture maps has become common over the last several years, as the hardware support for texture mapping has increased. Texture maps provide visual richness to computer-rendered models without adding more polygons to the scene.

Texture mapping requires 2D *texture coordinates* at every vertex of the model. These coordinates provide a parameterization of the texture map over the surface. Surfaces with complex geometric structure may be decomposed into polygonal patches, each with its own parameterization. Our system can simplify surfaces composed of a connected network of such polygonal patches, treating the patch boundaries as common borders which are simplified consistently to avoid cracks.

As we collapse an edge, we must compute texture coordinates for the generated vertex. These coordinates should reflect the original parameterization of the texture over the surface. We use linear interpolation to find texture coordinates for the corresponding point on the old surface, and assign these coordinates to the generated vertex.

This approach works well in many cases, as demonstrated in Section 8. However, there can still be some sliding of the texture across the surface. We have recently extended our mapping approach to also measure and bound the deviation of the texture coordinates[8]. In this approach, the texture coordinates produce a new set of pointwise correspondences between simplifications, and the deviation measured using these correspondences measures the deviation of the texture. This extension allows us to make guarantees about the complete appearance of the simplified meshes, measuring not only the surface deviation (seen at the silhouettes), but the texture coordinate deviation of all the interior pixels.

As we add more error measures to our system, it becomes necessary to decide how to weight these errors to determine the overall cost of an edge collapse. Each type of error at an edge mandates a particular viewing distance based on a user-specified screen-space tolerance (e.g. number of allowable pixels of surface or texture deviation). We conservatively choose the farthest of these. At run-time, the user can still adjust an overall screen-space tolerance, but the relationships between the types of error are fixed at the time of the simplification pre-process.

7. System Implementation

We divide our software system into two major components: the simplification pre-process, which performs the automatic simplification described previously in this article, and the interactive visualization application, which employs the resulting levels of detail to perform high-speed, high-quality rendering.

7.1. Simplification Pre-Process

All the algorithms described in this paper have been implemented and applied to various models. While the simplification process itself is only a pre-process with respect to the graphics application, we would still like it to be as efficient as possible. The most time-consuming part of our implementation is the re-computation of edge costs as the surface is simplified, as described in Section 3.1. To reduce this

Model	Method	# Evals	# Collapses	#E/#C	CPU Time
Bunny	complete	1,372,122	34,819	39.4	5:01
	lazy	436,817	34,819	12.5	1:56
Torus	complete	1,494,625	39,982	37.4	5:27
	lazy	589,839	39,987	14.8	2:44

Table 1: Effect of lazy cost evaluation on simplification speed. The lazy method reduces the number of edge cost evaluations performed per edge collapse operation performed, speeding up the simplification process. Time is in minutes:seconds on a 195 MHz MIPS R10000 processor.

computation time, we allow our approach to be slightly less greedy by performing a *lazy evaluation* of edge costs as the simplification proceeds.

Rather than recompute all the local edge costs after a collapse, we simply set a *dirty flag* for these edges. When we pick the next edge to collapse off the priority queue, we check to see if the edge is dirty. If so, we re-compute it's cost, place it back in the queue, and pick again. We repeat this until the lowest cost edge in the queue is clean. This clean edge has a lower cost than the known costs of all the other edges, be they clean or dirty. If the recent edge collapses cause an edge's cost to increase significantly, we will find out about it before actually choosing to collapse it. The potentially negative effect is that if the cost of a dirty edge has decreased, we may not find out about it immediately, so we will not collapse the edge until later in the simplification process.

This lazy evaluation of edge costs significantly speeds up the algorithm without much effect on the error growth of the progressive mesh. Table 1 shows the number of edge cost evaluations and running times for simplifications of the bunny and torus models with the complete and lazy evaluation schemes. Figures 10 and 11 show the effect of lazy evaluation on error growth for these models. The lazy evaluation has a minimal effect on error. In fact in some cases, the error of the simplification using the lazy evaluation is actually smaller. This is not surprising, because a strictly greedy choice of edge collapses does not guarantee optimal error growth.

Given that the lazy evaluation is so successful at speeding up the simplification process with little impact on the error growth, we still have room to be more aggressive in speeding up the process. For instance, it may be possible to include a *cost estimation* method in our prioritization scheme. If we have a way to quickly estimate the cost of an edge collapse, we can use these estimates in our prioritization. Of course, we must still record the guaranteed error bound when we finally perform a collapse operation. If our guaranteed bound is too far off from our initial estimate, we may choose to put the edge back on the queue, prioritized by its true cost.

7.2. Interactive Visualization Application

More important than the speed of the simplification itself is the speed at which our graphics application runs. The simplification algorithm outputs a list of edge collapses and associated error bounds. While it is possible to use this output to

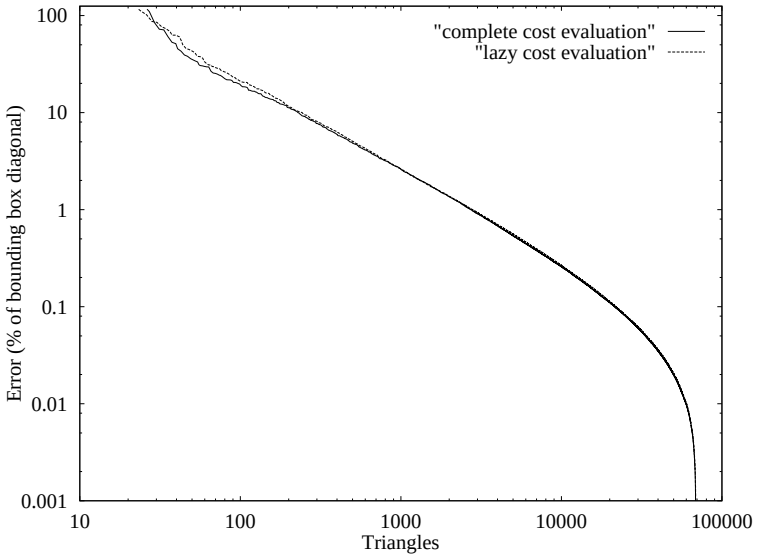

Figure 10: Error growth for simplification of the bunny model.

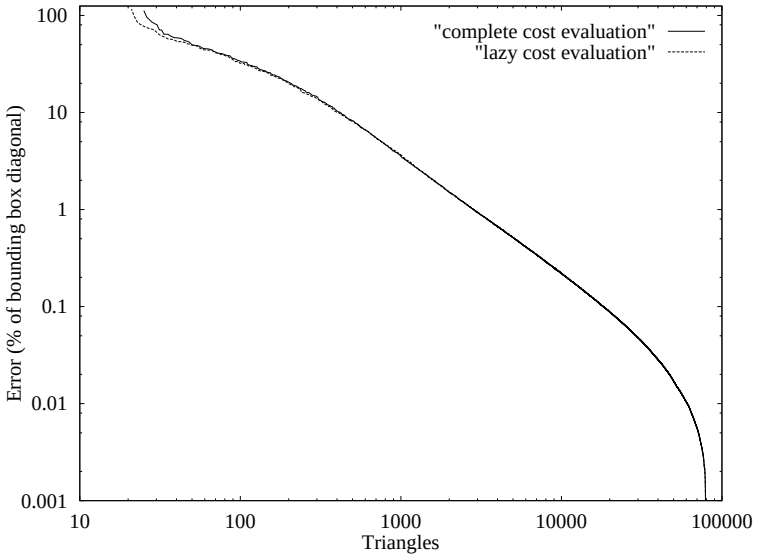

Figure 11: Error growth for simplification of the wrinkled torus model.

create view-dependent simplifications on the fly in the visualization application (as described by Hoppe[26]), such a system is fairly complex, requiring computational resources to adapt the simplifications and *immediate-mode* rendering of the final triangles.

Our application is written to be simple and efficient. We first sample the progressive mesh to generate a static set of levels of detail. These are chosen to have triangle counts which decrease by a factor of two from level to level. This limits the total memory usage to twice the size of the input model.

We next load these levels of detail into our visualization application, which store them as display lists (often referred to as *retained mode*). On machines with high-performance graphics acceleration, such display lists are retained in a cache on the accelerator and do not need to be sent by the CPU over a bus to the accelerator every frame. On an SGI Onyx with InfiniteReality graphics, we have seen a speedup of 2-3 times, just due to the use of display lists.

Our interactive application is written on top of SGI's Iris Performer library[41], which provides a software pipeline designed to achieve high graphics performance. The geometry of our model, which may be composed of many individual objects at several levels of detail, is stored in a scene graph. One of the scene graph structures, the LODNode, is used to store the levels of detail of an object. This LODNode also stores a list of switching distances, which indicate at what viewing distance each level of detail should be used (the viewing distance is the 3D distance from the eye point to the center of the object's bounding sphere). We compute these switching distances based on the 3D surface deviation error we have measured for each level of detail (using the total surface deviation, field of view, and screen resolution). The bounding sphere radius is added to the computed distances to account for Performer's measuring of the distance to the sphere center (rather than the closest point on the sphere).

The rendering of the levels of detail in this system involves minimal overhead. When a frame is rendered, the viewing distance for each object is computed and this distance is compared to the list of switching distances to determine which level of detail to render.

The application allows the user to set a 2D error tolerance, which is used to scale the switching distances. When the error tolerance is set to 1.0, the 3D error for the rendered levels of detail will project to no more than a single pixel on the screen. Setting it to 2.0 allows two pixels of error, etc. This screen-space surface deviation amounts to the number of pixels the objects' silhouettes may be off from a rendering of the original level of detail.

8. Results

We have applied our simplification algorithm to several distinct objects: a bunny rabbit, a wrinkled torus, a lion, a Ford Bronco, a dragon, a Buddha statue, and an armadillo, composed of a total of 393 parts (each simplified independently). Table 2 shows the total input complexity of each of these objects as well as the time needed to generate a progressive mesh representation. All simplifications were performed

Model	Parts	Orig. Triangles	CPU Time (Min:Sec)
Bunny	1	69,451	1:56
Torus	1	79,202	2:44
Lion	49	86,844	1:56
Bronco	339	74,308	1:29
Dragon	1	871,306	18:37
Buddha	1	1,087,474	23:56
Armadillo	1	1,999,404	42:27

Table 2: Simplifications performed. CPU time indicates time to generate a progressive mesh of edge collapses until no more simplification is possible.

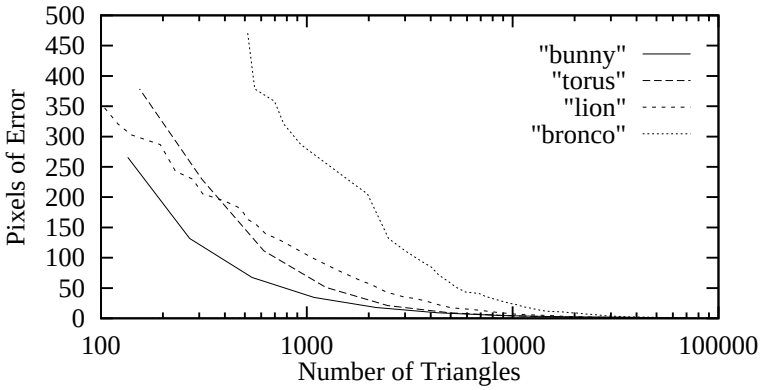

Figure 12: Complexity vs. screen-space error for four models

on an SGI workstation running a MIPS R12000 process.

Figure 12 graphs the complexity of each object vs. the number of pixels of screen-space error for a particular viewpoint. Each set of data was measured with the object centered in the foreground of a 1000x1000-pixel viewport, with a 45^o field-of-view, like the Bronco in Plates 2 and 3. This was the easiest way for us to measure the screen-space error, because the lion and bronco models each have multiple parts that independently switch levels of detail. Conveniently, this function of complexity vs. error at a fixed distance is proportional to the function of complexity vs. viewing distance with a fixed error. The latter is typically the function of interest.

Plate 1 shows the typical way of viewing levels of detail – with a fixed error bound and levels of detail changing as a function of distance. Plates 2 and 3 show close-ups of the Bronco model at full and reduced resolution.

Plates 4 and 5 show the application of our algorithm to the texture-mapped wrinkled torus and lion models. If you know how to free-fuse stereo image pairs, you can fuse the tori or any of the adjacent pairs of textured lion. Because the tori are rendered at an appropriate distance for switching between the two levels of detail, the images are nearly indistinguishable, and fuse to a sharp, clear image. The lions, however, are not rendered at their appropriate viewing distances, so certain discrepancies will appear as fuzzy areas. Each of the lion's 49 parts is

individually colored in the wire-frame rendering to indicate which of its levels of detail is currently being rendered.

8.1. Applications of the Projection Algorithm

We have also applied the technique of finding a one-to-one planar projection to the simplification envelopes algorithm[9]. The simplification envelopes method requires the calculation of a vertex normal at each vertex that may be used as a direction to offset the vertex. The criterion for being able to move a vertex without creating a local self-intersection is the same as the criterion for being able to project to a plane. The algorithm presented by Cohen, Varshney, et al.[9] used a heuristic based on averaging the face normals.

By applying the projection algorithm based on linear programming (presented in Section 4.1) to the computation of the offset directions, we were able to perform more drastic simplifications. The simplification envelopes method could previously only reduce the bunny model to about 500 triangles, without resulting in any self-intersections. Using the new approach, the algorithm can reduce the bunny to 129 triangles, with no local self-intersections. Because we found valid offset directions where previous heuristics failed, the envelopes were displaced more, allowing more room for simplification between the envelopes.

8.2. Video Demonstration

We have produced a video demonstrating the capabilities of the algorithm and smooth switching between different levels-of-details for different models (IJCGA does not publish a video proceedings, but the video appears as [11]. It shows the speed-up in the frame rate for eight circling Bronco models (about a factor of six) with almost no degradation in image quality (the error tolerance was 6 pixels of deviation in screen space). The video also highlights the performance on simplifying textured models, showing smooth switching between levels of detail. The texture coordinates were computed using the algorithm in Section 6.

9. Comparison to Previous Work

While concrete comparisons are difficult to make without careful implementations of all the related approaches readily available, we compare some of the features of our algorithm with those of a few others. The efficient and complete algorithms for computing the planar projection and placing the generated vertex after edge collapse should improve the performance of many earlier algorithms that use vertex removals or edge collapses.

We have directly compared our implementation with that of the simplification envelopes approach[9]. We generated levels of detail of the Stanford bunny model (70,000 triangles) using the simplification envelopes method, then generated levels of detail with the same number of triangles using the successive mapping approach. Visually, the models were comparable. The error bounds for the simplification envelopes method were smaller by about a factor of two for a given number of

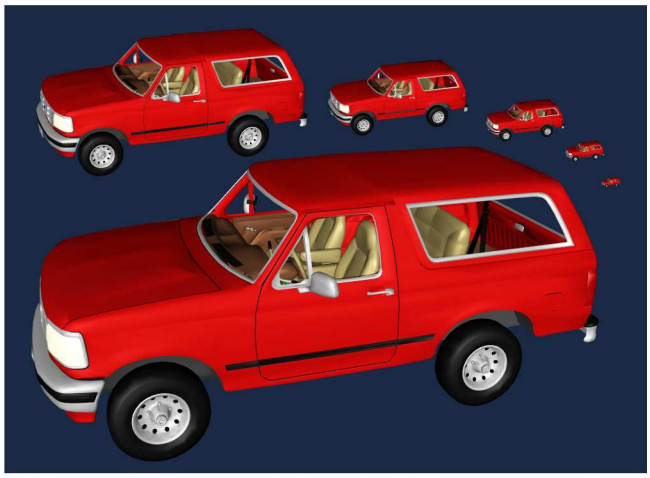

Plate 1: 6 views of the Ford Bronco model, all at 2 pixels of error (0.17 mm)

Triangle counts:	41,855	12,939
	27,970	8,385
	20,922	4,766

Plate 2: Bronco at full resolution 74,000 triangles

Plate 3: 26 pixels of error (4.4 mm) 9,000 triangles

Plate 4: Wrinkled torus model at a transitional distance for 1 pixel of error (0.085 mm)

39,600 triangles **19,800 triangles**

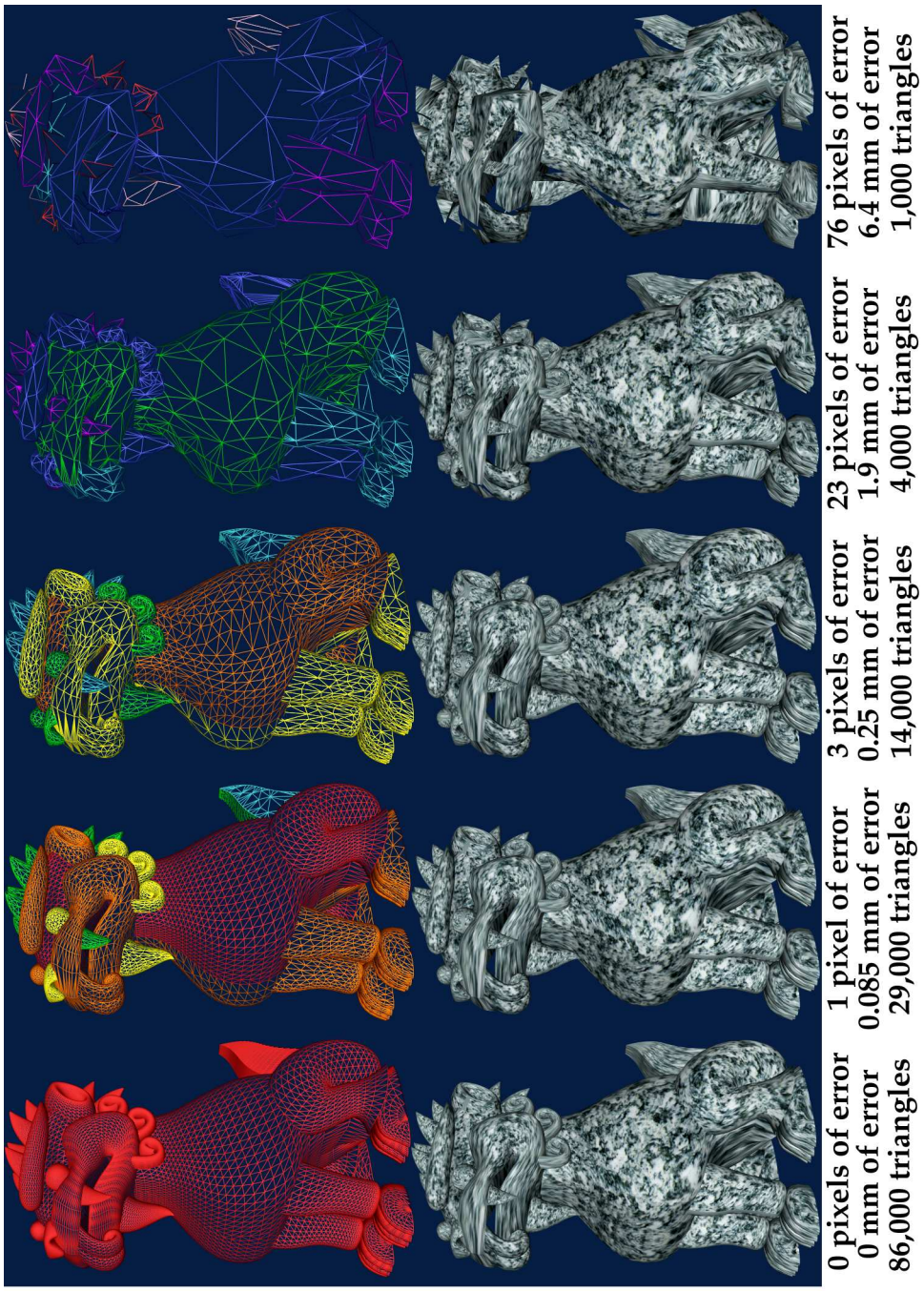

Plate 5: 6 levels of detail for the lion (colors indicate levels of detail of individual parts)

triangles, because the error bounds for the two methods measure different things. Simplification envelopes only bounds the surface deviation in the direction normal to the original surface, while the mapping approach prevents the surface from sliding around as well. Also, simplification envelopes created local creases in the bunnies, resulting in some shading artifacts. The successive mapping approach discourages such creases by its use of planar projections. At the same time, the performance of the simplification envelopes approach (in terms complexity vs. error) has been improved by our new projection algorithm.

Hoppe's progressive mesh[28] implementation is more complete than ours in its handling of colors, textures, and discontinuities. However, this technique provides no guaranteed error bounds, so there is no simple way to automatically choose switching distances that guarantee some visual quality.

The multi-resolution analysis approach to simplification[14,16,35] does, in fact, provide strict error bounds as well as a mapping between surfaces. However, the requirements of its subdivision topology and the coarse granularity of its simplification operation do not provide the local control of the edge collapse. The earlier approaches[14,16] do not deal well with sharp edges. Hoppe[28] had previously compared his progressive meshes with the multi-resolution analysis meshes. For a given number of triangles, his progressive meshes provide much higher visual quality. However, recent advances[35] have improved the quality of the multi-resolution analysis meshes by allowing the specification of constraints (e.g. along sharp edges). Like our algorithm, their approach uses a sequence of local planar mappings to compute error bounds during a simplification process. They use a conformal mapping to the plane rather than a projection as employed by our algorithm. Their conformal map always exists and optimizes the preservation of angles and areas.

Guéziec's tolerance volume approach[21,22] also uses edge collapses with local error bounds. Whereas the boxes used by the successive mapping approach are maintained in a global object space, Guéziec's error volume is defined using spheres centered at the simplified vertices. One possible disadvantage of this approach is that the error volume may grow as the simplified surface fluctuates closer to and farther away from the original surface. This is due to the fact that the newer spheres must always contain the older spheres. The boxes used by our successive mapping approach are not centered on the surface and do not grow as a result of such fluctuations. However, his approach has the advantage that the locations of the new vertices are truly optimized in 3D rather than in 1D. This could result in tighter bounds, but this cannot be determined without a side-by-side comparison of results. Also, the tolerance volume approach does not generate mappings between the surfaces for use with other attributes. It may be possible to incorporate a mapping procedure into this approach, but it would probably not be an inherent part of the optimization procedure. Thus some of the optimized vertices may not have bijective mappings.

We have made several significant improvements over the simplification algorithm presented by Bajaj and Schikore[2,44]. First, we have replaced their projection heuristic with a robust algorithm for finding a valid direction of projection. Second, we

have generalized their approach to handle more complex operations, such as the edge collapse. Finally, we have presented an error propagation algorithm which correctly bounds the error in the surface deviation. Their approach represented error as infinite slabs surrounding each triangle. Because there is no information about the extent of these slabs, it is impossible to correctly propagate the error from a slab with one orientation to a new slab with a different orientation.

10. Future Work

There are several apparent areas for future work. These involve improvements in running time, space requirements, tightness of error bounds, generality, and appearance-preservation.

As described in Section 7.1, there is great potential for speed improvements using cost estimation in place of many of the guaranteed cost computations. If the estimates are often close to the true costs, it should be possible to keep the error growth down while providing significant speed improvements along with guaranteed error bounds on the final results.

We have focused on vertex placements that are more general than most current edge collapse algorithms. However, these general-position vertices have greater storage requirements than more restricted placements, such as the end- or mid-points of an edge. It may be useful to use the general position vertices only when they produce sufficiently smaller error bounds than any of the restricted positions.

There are cases where the projection onto a plane produces mappings with unnecessarily large error. We only optimize surface position in the direction orthogonal to the plane of projection. It would be useful to generate and optimize mappings directly in 3D to produce better simplifications, with tighter error bounds. Such mappings would also be capable of measuring simplification error for edge neighborhoods that have no one-to-one projection to the plane. However, it seems that an approach which optimizes mappings on the surface may be significantly slower than our current approach.

Our system currently handles non-manifold topologies by breaking them into independent surfaces, which does not maintain connectivity between the components. In our appearance-preserving system[8], however, we simplify a network of connected patches by enforcing connectivity constraints on the patch boundaries. Handling non-manifold regions in this way, preserving their connectivity, should provide higher visual fidelity for large screen-space tolerances. This sort of generality in input models is desirable for dealing with real-word data, such as those produced by CAD applications.

Finally, the true preservation of appearance with significant reduction in model complexity is one of the major goals of simplification. We have pursued this goal with an approach employing texture and normal maps to store the material color and surface curvature properties of the input model[8]. This is the first work to provide guaranteed bounds on the final, shaded appearance of simplified objects. However, it may also be desirable to preserve these important appearance attributes in a per-vertex representation. Thus we need better measures of how changes in

the interpolated colors or normals across a surface affect the final rendered images. Such measures must take into account the effect of an object's distance from the eye point, and allow increased simplification as an object gets farther away, while guaranteeing some measure of final appearance.

Acknowledgments

We would like to thank Stanford Computer Graphics Laboratory for the bunny, dragon, and Buddha models, Stefan Gottschalk for the wrinkled torus model, Lifeng Wang and Xing Xing Computer for the lion model from the Yuan Ming Garden, Division and Viewpoint for the Ford Bronco model, and Venkat Krishnamurthy, Marc Levoy, and Peter Schröder for the armadillo model. Thanks to Michael Hohmeyer for the linear programming library. We would also like to thank Jeff Erikson, Carl Mueller, and the UNC Walkthrough Group. This work was supported in part by an Alfred P. Sloan Foundation Fellowship, ARO Contract DAAH04-96-1-0257, NSF Grant CCR-9319957, NSF Grant CCR-9625217, ONR Young Investigator Award, Intel, DARPA Contract DABT63-93-C-0048, NSF/ARPA Center for Computer Graphics and Scientific Visualization, and NIH/National Center for Research Resources Award 2 P41RR02170-13 on Interactive Graphics for Molecular Studies and Microscopy.

References

1. Pankaj K. Agarwal and Subhash Suri. Surface approximation and geometric partitions. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, pages 24–33, 1994.
2. C. Bajaj and D. Schikore. Error-bounded reduction of triangle meshes with multivariate data. *SPIE*, 2656:34–45, 1996.
3. H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 293–302, 1994.
4. M.P. Do Carmo. *Differential Geometry of Curves and Surfaces.* Prentice Hall, 1976.
5. A. Certain, J. Popovic, T. Derose, T. Duchamp, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *Proc. of ACM Siggraph*, pages 91–98, 1996.
6. Paolo Cignoni, C. Rocchini, and Roberto Scopigno. Metro: Measuring error on simplified surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998. ISSN 1067-7055.
7. Kenneth L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes Comput. Sci.*, pages 246–252. Springer-Verlag, 1993.
8. J. Cohen, M. Olano, and D. Manocha. Appearance preserving simplification. In *Proc. of ACM SIGGRAPH*, pages 115–122, 1998.
9. J. Cohen, A. Varshney, D. Manocha, and G. Turk et al. Simplification envelopes. In *Proc. of ACM Siggraph'96*, pages 119–128, 1996.
10. Jonathan Cohen, Dinesh Manocha, and Marc Olano. Simplifying polygonal models using successive mappings. *IEEE Visualization '97*, pages 395–402, November 1997.

ISBN 0-58113-011-2.

11. Jonathan Cohen, Dinesh Manocha, and Marc Olano. Simplifying polygonal models using successive mappings. *Video Proceedings of IEEE Visualization '97*, November 1997.
12. G. Das and D. Joseph. The complexity of minimum convex nested polyhedra. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 296–301, 1990.
13. M. J. DeHaemer, Jr. and M. J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers & Graphics*, 15(2):175–184, 1991.
14. T. Derose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topology type. Technical Report TR 93-10-05, Department of Computer Science, University of Washington, 1993.
15. T. K. Dey, H. Edelsbrunner, S. Guha, and D. Nekhayev. Topology preserving edge contraction. Technical Report RGI-Tech-98-018, Raindrop Geomagic, 1999. http://www.cis.ohio-state.edu/ tamaldey/paper/simplify/paper.ps.gz.
16. M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proc. of ACM Siggraph*, pages 173–182, 1995.
17. Carl Erikson. Polygonal simplification: An overview. Technical Report TR96-016, Department of Computer Science, University of North Carolina at Chapel Hill, 1996.
18. Leila De Floriani, Paola Magillo, and Enrico Puppo. Building and traversing a surface at variable resolution. In Roni Yagel and Hans Hagen, editors, *IEEE Visualization 97*, pages 103–110. IEEE, November 1997.
19. Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. *Proceedings of SIGGRAPH 97*, pages 209–216, August 1997. ISBN 0-89791-896-7. Held in Los Angeles, California.
20. Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. *IEEE Visualization '98*, pages 263–270, October 1998. ISBN 0-8186-9176-X.
21. A. Gueziec. Surface simplification with variable tolerance. In *Second Annual Intl. Symp. on Medical Robotics and Computer Assisted Surgery (MRCAS '95)*, pages 132–139, November 1995.
22. André Guéziec. Locally toleranced surface simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):168–189, April - June 1999. ISSN 1077-2626.
23. Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. *Proceedings of SIGGRAPH 99*, pages 325–334, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.
24. P. Heckbert and M. Garland. Multiresolution modeling for fast rendering. *Proceedings of Graphics Interface '94*, pages 43–50, May 1994.
25. Paul S. Heckbert and Michael Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH 97 Course Notes*, 1997.
26. H. Hoppe. View dependent refinement of progressive meshes. In *SIGGRAPH 97 Conference Proceedings*, pages 189–198. ACM SIGGRAPH, 1997.
27. H. Hoppe, T. Derose, T. Duchamp, J. Mcdonald, and W. Stuetzle. Mesh optimization. In *Proc. of ACM Siggraph*, pages 19–26, 1993.
28. Hugues Hoppe. Progressive meshes. In *SIGGRAPH 96 Conference Proceedings*, pages 99–108. ACM SIGGRAPH, Addison Wesley, August 1996.

29. Hugues H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. *IEEE Visualization '99*, pages 59–66, October 1999. ISBN 0-7803-5897-X. Held in San Francisco, California.

30. Reinhard Klein. Multiresolution representations for surface meshes. Technical report, Wilhelm Schickard Institut für Informatik, Universität Tübingen, 1997.

31. Reinhard Klein and Jörg Krämer. Building multiresolution models for fast interactive visualization. In *SCCG*, 1997.

32. Reinhard Klein, Gunther Liebich, and Wolfgang Straßer. Mesh reduction with error control. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

33. Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. *Graphics Interface '98*, pages 43–50, June 1998. ISBN 0-9695338-6-1.

34. B. Kolman and R. Beck. *Elementary Linear Programming with Applications*. Academic Press, New York, 1980.

35. Aaron W. F. Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 95–104. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

36. Peter Lindstrom and Greg Turk. Evaluation of memoryless simplification. *IEEE Transactions on Visualization and Computer Graphics*, 5(2):98–115, April - June 1999. ISSN 1077-2626.

37. Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. *IEEE Visualization '98*, pages 279–286, October 1998. ISBN 0-8186-9176-X.

38. David Luebke and Carl Erikson. View-dependent simplification of arbitrary polygonal environments. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 199–208. ACM SIGGRAPH, Addison Wesley, August 1997. ISBN 0-89791-896-7.

39. J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 296–306, 1992.

40. J. O'Rourke. *Computational Geometry in C*. Cambridge University Press, 1994.

41. John Rohlf and James Helman. IRIS performer: A high performance multiprocessing toolkit for real–Time 3D graphics. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 381–395. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

42. R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):67–76, 462, Aug. 1996. Proc. Eurographics '96.

43. J. Rossignac and P. Borrel. Multi-resolution 3D approximations for rendering. In *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, June–July 1993.

44. D. Schikore and C. Bajaj. Decimation of 2d scalar data with error control. Technical report, Computer Science Report CSD-TR-95-004, Purdue University, 1995.

45. B. Schneider, P. Borrel, J. Menon, J. Mittleman, and J. Rossignac. Brush as a walkthrough system for architectural models. In *Fifth Eurographics Workshop on Rendering*, pages 389–399, July 1994.

46. W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. In *Proc. of ACM Siggraph*, pages 65–70, 1992.

47. R. Seidel. Linear programming and convex hulls made easy. In *Proc. 6th Ann.*

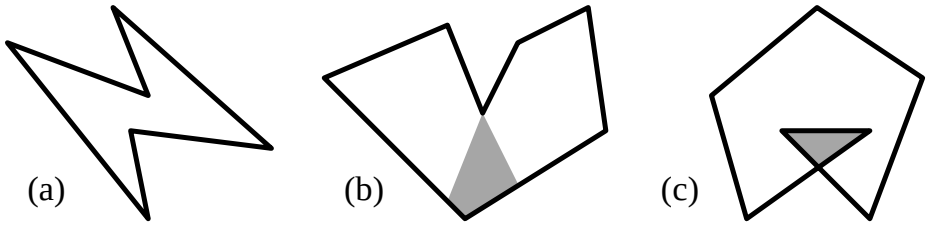

Figure A.1: Polygons in the plane. (a) A simple polygon (with an empty kernel). (b) A star-shaped polygon with its kernel shaded. (c) A non-simple polygon with its kernel shaded.

ACM Conf. on Computational Geometry, pages 211–215, Berkeley, California, 1990.

48. D. C. Taylor and W. A. Barrett. An algorithm for continuous resolution polygonalizations of a discrete surface. In *Proc. Graphics Interface '94*, pages 33–42, Banff, Canada, May 1994.
49. G. Turk. Re-tiling polygonal surfaces. In *Proc. of ACM Siggraph*, pages 55–64, 1992.
50. A. Varshney. *Hierarchical Geometric Approximations*. PhD thesis, University of N. Carolina, 1994.
51. J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, June 1997.
52. Julie C. Xia and Amitabh Varshney. Dynamic view-dependent simplification for polygonal models. In *IEEE Visualization '96*. IEEE, October 1996. ISBN 0-89791-864-9.

Appendix A: Projection Theorems

The simplification algorithm we have presented depends on our ability to efficiently compute orthogonal projections which provide one-to-one mappings between small portions of triangle meshes. With this in mind, we now present the mathematical properties of the mapping used in designing the projection algorithm.

Definition A.1 *A* simple *polygon is a planar polygon in which edges only intersect at their two endpoints (vertices) and each vertex is adjacent to exactly two edges (see Figure A.1(a)).*

Definition A.2 *The* kernel *of a simple polygon is the intersection of the inward-facing half-spaces bounded by its edges (see Figure A.1(b)). For a non-simple polygon (see Figure A.1(c)), the kernel is the intersection of a consistently-oriented set of half-spaces bounded by its edges (i.e. if we traverse the edges in a topological order, the half-spaces must be either all to our right or all to our left).*

Definition A.3 *A* star-shaped *polygon is a simple polygon with a non-empty kernel (see Figure A.1(b)).*

By construction, any point in the kernel of a star-shaped polygon has an unobstructed line of sight to the polygon's entire boundary.

Definition A.4 *A* complete vertex neighborhood, N_v, *is a set of triangles which forms a complete cycle around a vertex, v.*

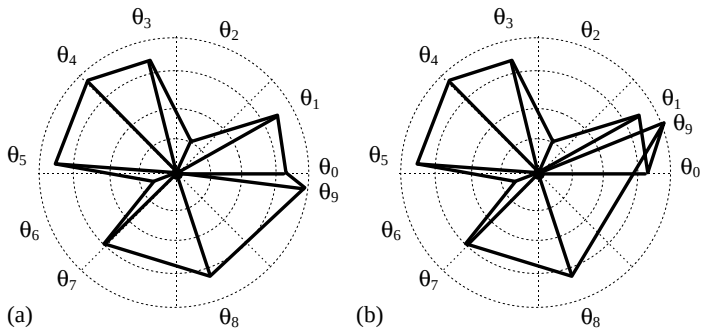

Figure A.2: Projections of a vertex neighborhood, visualized in polar coordinates. (a) No angular intervals overlap, so the boundary is star-shaped, and the projection is a one-to-one mapping. (b) Several angular intervals overlap, so the boundary is not star-shaped, and the projection is not one-to-one.

The triangles of N_v are ordered: $\triangle_0, \triangle_1, ..., \triangle_{n-1}, \triangle_0$. Each pair of consecutive triangles in this ordering, $(\triangle_i, \triangle_{i+1})$, is adjacent, sharing a single edge, e_i; one of the vertices of e_i is v.

Definition A.5 *The* angle space *of an orthogonal projection of a complete vertex neighborhood, N_v, is the θ-coordinate space, $[0, 2\pi]$, constructed by converting the projected neighborhood to polar coordinates, (r, θ), with v at the origin (see Figure A.2(a)).*

Definition A.6 *The* angular interval *covered by the orthogonal projection of triangle, $\triangle_i$, from a complete vertex neighborhood, N_v, is the interval $[\theta_i, \theta_{(i+1) \bmod n}]$, where θ_i is the theta-coordinate of edge e_i.*

Definition A.7 *The angle space of an orthogonal projection of a complete vertex neighborhood is* multiply-covered *if each angle, $\theta \in [0, 2\pi]$, is covered by the projections of at least two triangles from N_v. It is k-*covered *if each angle is covered the projections of exactly k such triangles. A k-covered angle space is* exactly multiply-covered *if $k > 1$.*

Lemma A.1 *The orthogonal projection of a complete vertex neighborhood, N_v, onto the plane, $\mathcal{P}$, provides a one-to-one mapping between N_v and a polygonal subset of $\mathcal{P}$ iff the angular intervals of the projected triangles of N_v do not overlap.*

Proof. Consider the projection of N_v in polar coordinates, with v at the origin, and e_0 at $\theta = 0$ (see Figure A.2). Each triangle, $\triangle_i$, spans an angular interval in θ, bounded by e_i on one side and $e_{(i+1) \bmod n}$ on the other. If the intervals of the triangles do not overlap , then the triangles cannot overlap, and the projection must be one-to-one. If the intervals do overlap, the triangles themselves must overlap (near the origin, which they both contain), and the projection cannot be one-to-one (see Figure A.2(b)). □

Corollary A.1 *The orthogonal projection of a complete vertex neighborhood, N_v, onto the plane, $\mathcal{P}$, provides a one-to-one mapping between N_v and a polygonal subset of $\mathcal{P}$ iff the angle space of the projection of N_v is 1-covered.*

Proof. Lemma A.1 shows that for a one-to-one mapping, the angle space cannot be multiply-covered. Because the triangles of N_v form a complete cycle around v,

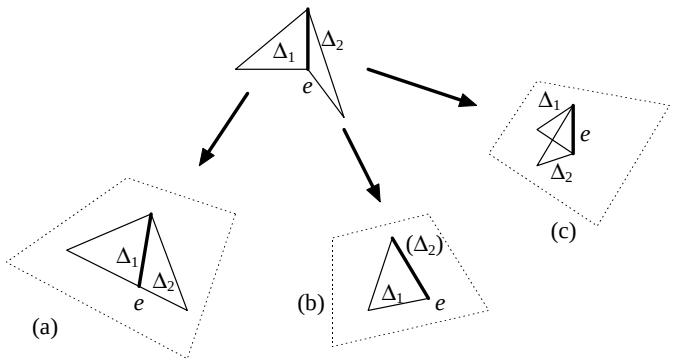

Figure A.3: Three projections of a pair of edge-adjacent triangles. (a) The projected edge is not a fold, because the normals of both triangles are within 90^o of the direction of projection. (b) The projected edge is a degenerate fold, because the normal of $\triangle_2$ is perpendicular to the direction of projection. (c) The projected edge is a fold because the normal of $\triangle_2$ is more than 90^o from the direction of projection.

the angle space must be fully covered. Thus, each angle must be covered exactly once. □

Lemma A.2 *The orthogonal projection of N_v onto the plane, $\mathcal{P}$, provides a one-to-one mapping between N_v and a polygonal subset of $\mathcal{P}$ iff the projection of N_v's boundary forms a star-shaped polygon in $\mathcal{P}$, with v in its kernel.*

Proof. If the projection provides a one-to-one mapping, the angular intervals of the triangles do not overlap, and the boundary forms a simple polygon, with the origin in the interior. The entire boundary of the polygon is visible from the origin. This is by definition a star-shaped polygon, with the origin, v, in its kernel. In the case where one or more interval pairs overlap, portions of the boundary are typically occluded from the origin's point of view. Thus v cannot be in the kernel of a star-shaped polygon. Note that if the angle space is exactly multiply-covered, and the boundaries of these coverings are totally coincident, the entire boundary also seems to be visible from the origin. However, such a polygon is not technically simple, thus the projection of N_v is not technically star-shaped. □

Definition A.8 *A* fold *in an orthogonal projection of a triangle mesh is an edge with two adjacent triangle whose projections lie to the same side of the projected edge. A* degenerate fold *is an edge with at least one triangle with a degenerate projection, lying entirely on the projected edge.*

Lemma A.3 *An orthogonal projection of a consistently-oriented triangle mesh is* fold-free *iff the triangle normals either all lie less than 90^o or all lie greater than 90^o from a vector in the direction of projection.*

Proof. We are given that the triangle mesh is orientable, with consistently oriented triangles and consistent normal vectors. The orientation of a projected triangle depends only on the relationship of its normal vector to the direction of projection (see Figure A.3). When these two vectors are less than 90^o apart, the projected triangle will have one orientation, while if they are greater than 90^o apart, the projected triangle will have the opposite orientation. At exactly 90^o, the projected triangle degenerates into line segment.

At a fold, the two triangles adjacent to the folded edge have opposite orientation in the plane, while at a non-folded edge, they have the same orientation. If all the triangle normals lie within the same hemisphere, either less than or greater than 90^o from the direction of projection, all the projected triangles will be consistently oriented, implying that none of the edges are folded.

If the normals do not all lie in one of these two hemispheres, the projected triangles may be divided into three groups according to their orientations in the plane (one group is for degenerate projections). Because the triangle mesh is fully connected, there must exist some edge which is adjacent to two triangles from different groups; this edge is a fold (or degenerate fold). □

Lemma A.4 *The orthogonal projection of N_v onto $\mathcal{P}$ provides a one-to-one mapping iff the projection is fold-free and its angle space is not exactly multiply-covered.*

Proof. Again, consider the projection of N_v in polar coordinates. When a fold occurs, the angular intervals of these triangles overlap. Thus a projection with a fold does not provide a one-to-one mapping. On the other hand, if the projection is fold-free, every edge around v has its triangles laid out to either side. Because the final triangle of N_v connects to the initial triangle, this fold-free projection provides a k-covering of the angle space. If $k = 1$, the projection provides a one-to-one-mapping (from Corollary A.1). If $k > 1$, the projection is exactly multiply-covered, implying that angular intervals overlap, and the projection does not provide a one-to-one mapping. □

Lemma A.5 *The orthogonal projection of N_v onto $\mathcal{P}$ provides a one-to-one mapping iff the projected triangles are consistently oriented and the angle-space of the projection is not exactly multiply-covered.*

Proof. We must show that the consistent orientation criterion is equivalent to the fold-free criterion of Lemma A.4. The projection of each of the edges, $e_0 ... e_n$, is either a fold or not a fold. The two triangles adjacent to each non-folded edge are consistently oriented, while those adjacent to each folded edge are inconsistently oriented (or degenerate). If none of the edges are folded, all adjacent pairs of triangles are consistently oriented, implying that all of N_v is consistently oriented. If any of the edges are folded, N_v is not consistently oriented. □

Theorem A.1 *The following statements about the orthogonal projection of a complete vertex neighborhood, N_v, onto the plane, $\mathcal{P}$, are equivalent:*

- *The projection provides a one-to-one mapping between N_v and a polygonal subset of $\mathcal{P}$.*
- *The angular intervals of the projected triangles of N_v do not overlap.*
- *The angle space of the projection of N_v is 1-covered.*
- *The projection of N_v's boundary forms a star-shaped polygon in $\mathcal{P}$, with the vertex, v, in its kernel.*
- *The normals of the triangles of N_v all lie within the same hemisphere about the line of projection and the angle space of the projection is not exactly multiply-covered.*
- *The projection of N_v is fold-free and its angle space is not exactly multiply-covered.*
- *The projected triangles of N_v are consistently oriented in $\mathcal{P}$ and the angle space of the projection is not exactly multiply-covered.*

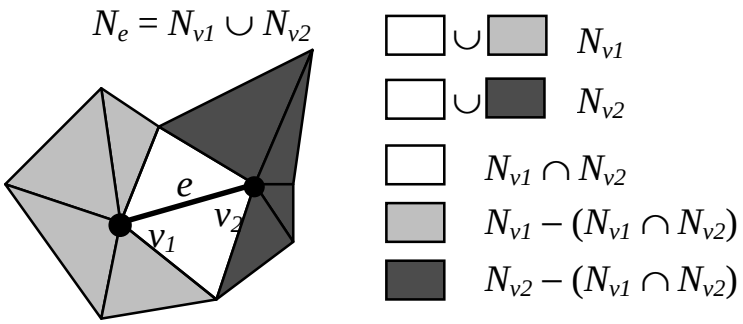

Figure A.4: The edge neighborhood is the union of two vertex neighborhoods. If we remove the two triangles of their intersection, we get two independent polygons in the plane.

Proof. This equivalence list is a direct consequence of Lemmas A.1, A.2, A.3, A.4, and A.5 and Corollary A.1. □

Definition A.9 *A* complete edge neighborhood, N_e, *is a set of triangles which forms a complete cycle around an edge, e (see Figure A.4).*

If v_1 and v_2 are the vertices of e, we can also write:

$$N_e = N_{v_1} \cup N_{v_2} \tag{A.1}$$

Lemma A.6 *Given an edge, e, and its vertices, v_1 and v_2, the orthogonal projection of N_e onto the plane, $\mathcal{P}$, is fold-free iff the projections of N_{v_1} and N_{v_2} onto $\mathcal{P}$ are fold-free.*

Proof. The set of triangle edges in N_e is the union of the edges from N_{v_1} and N_{v_2}. If neither N_{v_1} nor N_{v_2} contains a folded edge, then N_e cannot contain a folded edge. Similarly, if either N_{v_1} or N_{v_2} contains a folded edge, N_e will contain that folded edge as well. Also note that the projections of N_{v_1} and N_{v_2} must have the same orientation, because they have two triangles and one interior edge (e) in common. □

Lemma A.7 *The orthogonal projection of N_e onto $\mathcal{P}$ provides a one-to-one mapping between N_e and a polygonal subset of $\mathcal{P}$ iff the projections of its vertices, v_1 and v_2, provide one-to-one mappings between their neighborhoods and the plane, and the projection of the boundary of N_e is a simple polygon in $\mathcal{P}$.*

Proof. The projection provides a one-to-one mapping between N_{v_1} and a star-shaped subset of $\mathcal{P}$, and between N_{v_2} and a star-shaped subset of $\mathcal{P}$. The only way for N_e to not have a one-to-one mapping with a polygon in the plane is if the projections of N_{v_1} and N_{v_2} overlap, covering some points in the plane more than once.

Let N'_{v_1} and N'_{v_2} be the neighborhoods N_{v_1} and N_{v_2} with the two common triangles removed, as shown in Figure A.4:

$$N'_{v_1} = N_{v_1} - (N_{v_1} \cap N_{v_2}); N'_{v_2} = N_{v_2} - (N_{v_1} \cap N_{v_2}); \tag{A.2}$$

The projections of N'_{v_1} and N'_{v_2} are two polygons in $\mathcal{P}$. If the projections of N_{v_1} and N_{v_2} are each one-to-one, and these two polygons do not overlap, then the

projection of N_e is one-to-one. If the two polygons do overlap, the projection is not one-to-one, because multiple points on N_e are projecting to the same point in $\mathcal{P}$. Given that the projections of N_{v_1} and N_{v_2} are fold-free, the only way for the two polygons to overlap is for their boundaries to intersect. This intersection implies that the projection of N_e is a non-simple polygon.

So we have shown that if the projections of N_{v_1} and N_{v_2} provide one-to-one mappings with polygons in $\mathcal{P}$ and the projection of N_e's boundary is a simple polygon in $\mathcal{P}$, then the projection provides a one-to-one mapping between N_e and this simple polygon in $\mathcal{P}$. Also, if the projection covers a non-simple polygon, there can be no one-to-one mapping. □

Theorem A.2 *The orthogonal projection of N_e onto $\mathcal{P}$ provides a one-to-one mapping between N_e and a polygonal subset of $\mathcal{P}$ iff the projection of N_e is fold-free, the projections of the neighborhoods of its vertices, v_1 and v_2, are not exactly multiply covered, and the projection of its boundary is a simple polygon in $\mathcal{P}$.*

Proof. Given Lemma A.7, we only need to show that the projections of N_{v_1} and N_{v_2} provide one-to-one mappings iff the projection of N_e is fold-free, and the projections of N_{v_1} and N_{v_2} are not exactly multiply-covered. This is a direct consequence of Lemmas A.6 and A.4. □

Definition A.10 *An* edge collapse *operation applied to edge e, with vertices v_1 and v_2, merges v_1 and v_2 into a single, generated vertex, v_{gen}. In the process, any triangles adjacent to e become degenerate and are deleted.*

Lemma A.8 *Given an edge, e, which is collapsed to a vertex, v_{gen}, an orthogonal projection of N_e is a simple polygon iff the same orthogonal projection of $N_{v_{gen}}$ is a simple polygon.*

Proof. The collapse of e to v_{gen} does not affect the vertices on the boundary of N_e, so N_e and $N_{v_{gen}}$ have the same boundary. Thus the projection of the boundary of N_e is simple iff the projection of the boundary of $N_{v_{gen}}$ is simple. □

Lemma A.9 *A planar polygon with a non-empty kernel is simple iff it is star-shaped.*

Proof. A star-shaped polygon is defined as a simple polygon with a non-empty kernel. Thus if a polygon with a non-empty kernel is simple, it is star-shaped by definition. If a polygon with a non-empty kernel is not simple, it cannot be star-shaped. □

Lemma A.10 *Given an edge, e, which is collapsed to a vertex, v_{gen} inside the kernel of e, an orthogonal projection of N_e is simple iff the same projection of $N_{v_{gen}}$ is star-shaped.*

Proof. Recall from Lemma A.8 that N_e and $N_{v_{gen}}$ have the same projected boundary. We have been given that this projected boundary is a planar polygon with a non-empty kernel. From Lemma A.9, we know that this polygon is simple iff it is star-shaped. Thus the projection of the boundary of N_e is a simple polygon iff the projection of the boundary of $N_{v_{gen}}$ is a star-shaped polygon. □

Theorem A.3 *Given an edge, e, which is collapsed to a vertex, v_{gen} in the kernel of e, an orthogonal projection of N_e onto $\mathcal{P}$ provides a one-to-one mapping between N_e and a polygonal subset of $\mathcal{P}$ iff the projection of N_e is fold-free and the projected triangles of $N_{v_{gen}}$ are consistently oriented and do not multiply-cover the angle space.*

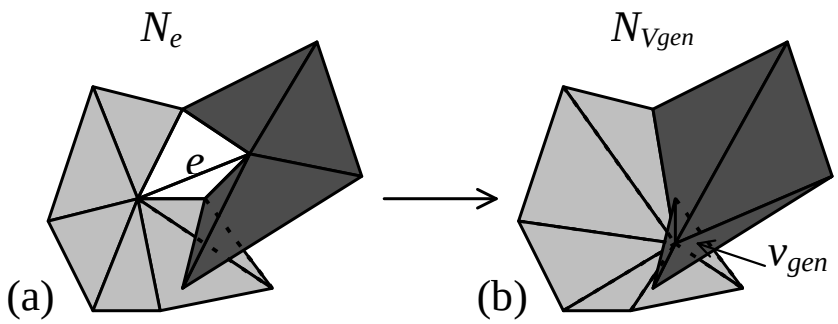

Figure A.5: A fold-free projection of an edge neighborhood, N_e, which is not one-to-one. (a) The projection of N_e has a non-empty kernel. (b) The projection of $N_{v_{gen}}$ has a 2-covered angle space. This can be detected by noting that the sum of the angular intervals of the triangles of $N_{v_{gen}}$ sum to 4π.

Proof. Theorem A.2 shows that the projection of N_e is one-to-one iff it is fold-free and simple. Lemma A.10 shows that it is simple iff the projection of $N_{v_{gen}}$ is star-shaped. Theorem A.1 shows that the projection of $N_{v_{gen}}$ is star-shaped iff its projected triangles are consistently oriented and do not multiply-cover the angle space. □

Figure A.5 shows an example of a fold-free edge projection that is not one-to-one and collapses to a multiply-covered vertex neighborhood.

Edge Collapse in the Plane

Theorems A.1 and A.3 lead us to an efficient algorithm for performing an edge collapse operation in the plane.

First, we find a fold-free projection for the edge, e. We can use linear programming with the normals of N_e as constraints to find a direction that guarantees such a fold-free projection, if one exists for this edge. We do not yet know if the projection is one-to-one, but rather than check to see if the projection forms a simple polygon, we defer this test until later.

Second, we find a point inside the kernel of the projection of N_e. Again, we can use linear programming to find such a point, if one exists for this projection.

Third, we collapse e's vertices, v_1 and v_2 to this point, v_{gen}, in the kernel. We do not know yet if the overall polygon is star-shaped, because even a non-simple polygon may have a non-empty kernel. If the polygon is not simple, neither the projection of N_e nor the projection of $N_{v_{gen}}$ provides a one-to-one mapping with a polygon in $\mathcal{P}$.

Finally, we verify that projections of N_{v_1}, N_{v_2}, and $N_{v_{gen}}$ are all one-to-one. For N_{v_1} and N_{v_2}, we verify that they are not exactly multiply-covered by adding up the spans of the angular intervals of their triangles. These spans should sum to 2π (within some floating point tolerance). For $N_{v_{gen}}$, we check not only the sum of the angular spans, but also the orientations of the projected triangles. If the spans sum to 2π and the orientations are consistent, $N_{v_{gen}}$ has a one-to-one mapping, and its boundary is star-shaped. These are all simple, $O(n)$ tests, with

small constant factors. They guarantee that we have a one-to-one mapping between N_e and the plane, and also between $N_{v_{gen}}$ and the plane; this also provides a one-to-one mapping between N_e and $N_{v_{gen}}$.

All the steps of the preceding algorithm run in $O(n)$ time (though we will later find $O(n^2)$ edge-edge intersections, which we will use in the error calculation and 3D vertex placement). This algorithm for performing an edge-collapse in the plane is described in more detail in Section 4.