Polygonal Simplification:
An Overview

TR96-016
1996

Carl Erikson

Department of Computer Science
CB #3175, Sitterson Hall
UNC-Chapel Hill
Chapel Hill, NC 27599-3175

UNC is an Equal Opportunity/Affirmative Action Institution.

Abstract

Polygonal simplification is a very active research topic, demonstrated by the recent explosion of papers in the field. The goal of this paper is to give a foundation and overview of this rapidly growing research area. The paper provides a foundation by explaining what polygonal simplification is, describing why it is useful, and defining some terminology of the field. The bulk of the paper classifies and overviews various simplification algorithms. Finally, since polygonal simplification is by no means a solved problem, the paper discusses possible future work.

1. Who Should Read this Paper?

The intended reader of this paper is one who has an undergraduate background in computer graphics and desires an introduction and overview to the field of polygonal simplification. Other potential readers are people interested in scientific visualization techniques, real-time walk-through applications, real-time rendering methods, or other types of simplification. The reader should learn what polygonal simplification is, why it is used, how it is accomplished using various algorithms, and how these methods might be extended in the future. The paper focuses on the details of how existing algorithms attempt to solve the simplification problem.

2. Polygonal Simplification

2.1 What is Polygonal Simplification?

Polygonal simplification is the act of transforming a three-dimensional polygonal model into a simpler version. It reduces the number of polygons needed to represent a model while trying to retain a good approximation to the original shape and appearance.

2.2 Why is Polygonal Simplification Useful?

Polygons are the most popular drawing primitive in computer graphics. Specialized graphics hardware can render them very quickly, and polygons can represent any model approximately. Unfortunately, accurately representing a three-dimensional model often requires a large number of polygons. For example, a smooth looking sphere can easily require several thousand polygons.

Current high-end computer graphics systems can render approximately a million polygons per second. If an application requires a 30 frames per second update, then it can draw roughly 30,000 or fewer polygons in one frame. Although this number seems large, an accurate representation of one three-dimensional model can easily contain 30,000 polygons. Nowadays, a million polygon scene is not extraordinary, and the day of the first billion polygon model rapidly approaches. Computer graphics hardware will be hard pressed to keep up with this growth in model complexity. For every computer graphics system, there exists a model complex enough to bring its performance to a crawl.

Polygonal simplification attempts to narrow this discrepancy between model complexity and hardware performance. Suppose a graphics system or application renders a scene of a house in which a 10,000 polygon chair is distant from the viewer. Since the chair is far away, it spans only a few pixels in the final image. If the application renders all 10,000 polygons, then each vertex in the model transforms to and each polygon rasterizes to the same few pixels. In this case, the rendering bottleneck becomes the vertex processing instead of the polygon rasterization. Certainly, rendering all these polygons is excessive. Instead, suppose the system substitutes a 100 polygon simplified chair for the original model in the scene. Not only does the resulting image roughly look the same as the original, but the application achieves a 9,900 polygon performance gain. Instead of having one simplified representation of the chair, suppose there exist several gradations of simplified chairs. The system substitutes for the original an 8,000 polygon model when the viewer is a certain distance from the chair, a 5,000 polygon chair when the viewer is even further away, and a 100 polygon chair when the viewer is extremely distant. Ideally, polygonal simplification enables the rendering of the fewest number of polygons possible to represent an object while ensuring the same image quality as the original.

This paper emphasizes increased rendering performance, but polygonal simplification provides other benefits. A three-dimensional model requires a large amount of storage, but simplification reduces these requirements if a system permits model degradation. This storage reduction helps quicken the transmission of large models over networks. Simplification also increases the efficiency of computationally intensive problems such as radiosity calculation, ray tracing, and collision detection.

3. Terminology

This paper frequently uses the terms below.

3.1 Aspect Ratio

This paper defines the aspect ratio of a triangle as the ratio of the longest over the shortest side of the tightest rectangle that encloses the triangle. The tightest rectangle is the rectangle with smallest area. Long and skinny triangles have poor aspect ratios that are much greater than one while regularly shaped triangles have aspect ratios close to one. Having triangles with aspect ratios close to one is a desirable property since long and skinny triangles are more prone to rendering and numerical errors. For example, in Figure 3.1, the long and skinny triangle has a poor aspect ratio while the nearly equilateral triangle has a very small aspect ratio. Note that this paper does not claim that this is the standard or accepted definition of aspect ratio of a triangle.
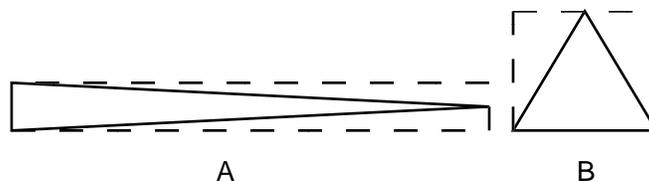

A   B

Figure 3.1: Triangle A has a poor aspect ratio of 10 while triangle B has an aspect ratio of 1.2.

3.2 Edge Swapping

If two triangles share an edge, then edge swapping is the act of removing the shared edge and adding an edge between the vertices of the triangles that were not adjacent to the shared edge. Algorithms use edge swapping to optimize the aspect ratios of triangles in polygonal models. Figure 3.2 is an example of edge swapping that results in better average aspect ratios.
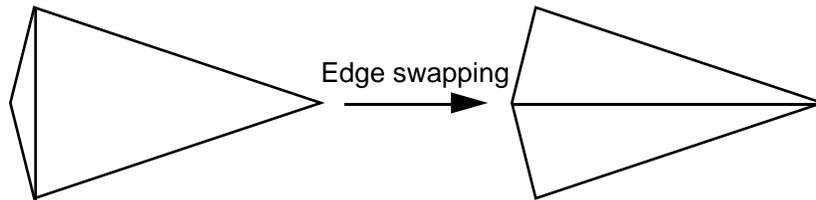

Figure 3.2: An example where edge swapping produces better average aspect ratios.

3.3 Levels of Detail

A level of detail is a single representation of a model. A greatly simplified model is a low level of detail. A marginally simplified model is a high level of detail. The original model is the highest level of detail. The levels of detail of a model are the model and all its simplified versions grouped together. When rendering an object, a graphics system substitutes different levels of detail for the original model according to how far away the viewer is. Figure 3.3 is an example of the levels of detail of a cat model.
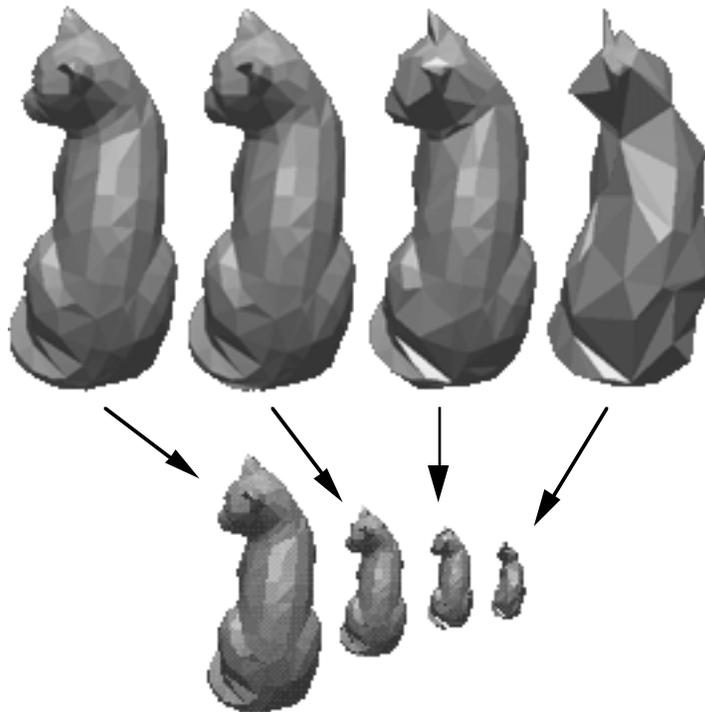

Figure 3.3: The top row of cats represent the levels of detail of a cat model. The bottom row of cats shows the distance at which an application might decide to draw each level of detail.

3.4 Local Curvature

The local curvature of a region of a model is an approximate measure of how fast the geometry changes at that area. A region of low curvature remains relatively stable, implying that the area is almost planar. A region of high curvature abruptly changes direction, meaning that the area contains a bump or spike. An algorithm using local curvature usually measures it at the vertices of a model. Figure 3.4 is an example of vertices with low and high local curvature.
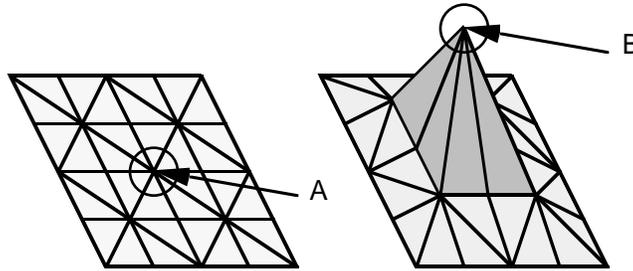

Figure 3.4: Vertex A has low local curvature while vertex B has relatively high local curvature.

3.5 Marching Cubes

William E. Lorensen and Harvey E. Cline created the Marching Cubes algorithm [11]. It creates a polygonal model out of a volumetric data set by substituting polygons for voxels on the surface of the model. The algorithm considers a voxel to be on the surface of a model when it and a neighboring voxel have density values that straddle a threshold value. Marching Cubes is very efficient but usually produces more than the necessary number of polygons needed to represent an object accurately. Figure 3.5 shows the execution of Marching Cubes on a simple volumetric data set.
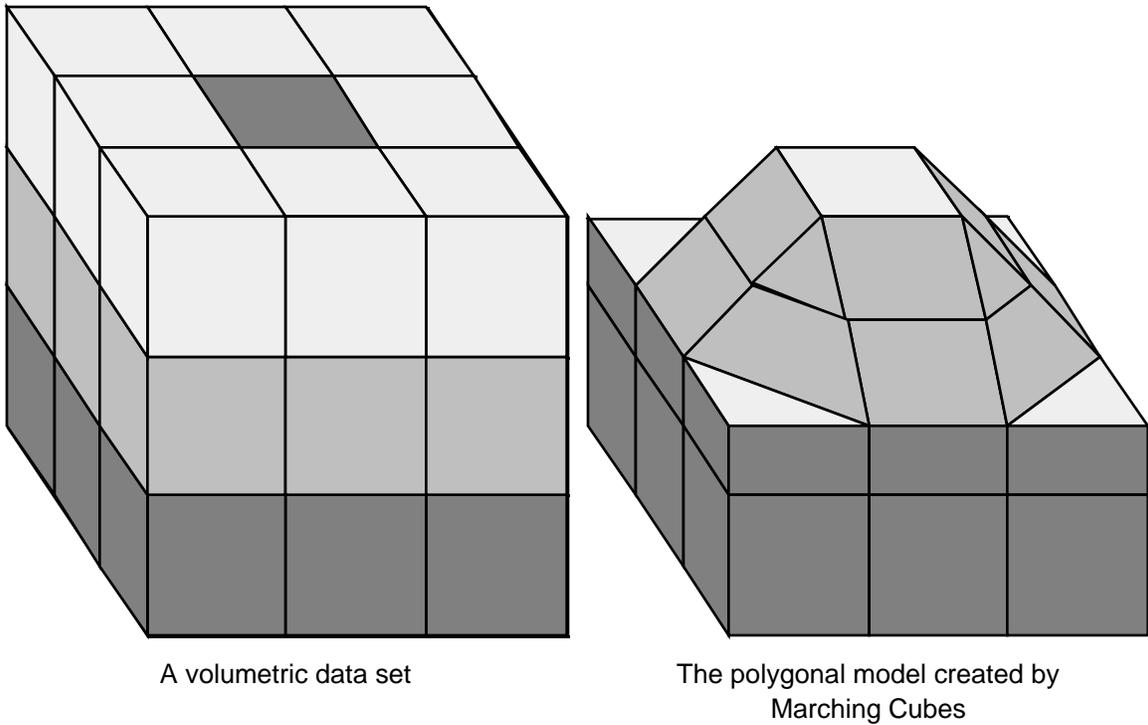
A volumetric data set   The polygonal model created by
                                                    Marching Cubes

Figure 3.5: Marching Cubes produces a polygonal model from a
volumetric data set.

3.6 Topology

Topology, in the context of polygonal simplification, has two different meanings. Local
topology is the geometric structure surrounding a particular face or vertex of a
polygonal model. For example, a vertex and all its adjacent faces define a local topology.
When a simplification algorithm preserves local topology, it does not change the local
geometric structure and the genus, or number of holes, of the object. Global topology is
the geometric structure of the whole polygonal model. When a simplification algorithm
preserves global topology, it preserves local topology and does not create self-
intersections in the object. A self-intersection occurs in the model when two non-
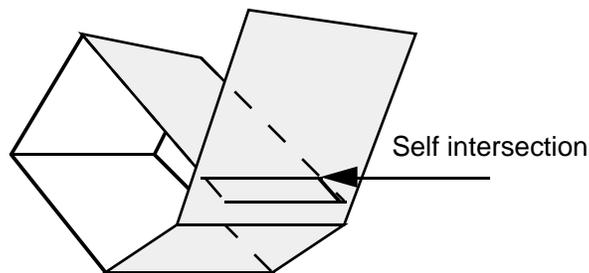adjacent faces intersect each other. Figure 3.6 demonstrates a self-intersection of a
model.

Self intersection

Figure 3.6: A self-intersection occurs in the model.

3.7 Triangulation

Triangulation is the act of decomposing a polygon into triangular polygons. Constrained triangulation is the same, except edges of the resulting triangulation cannot cross any edges specified as constraints. For example, if the edges that make up a hole inside a polygon are constraint edges, then constrained triangulation preserves the hole. Figure 3.7 is an example of triangulation and constrained triangulation.
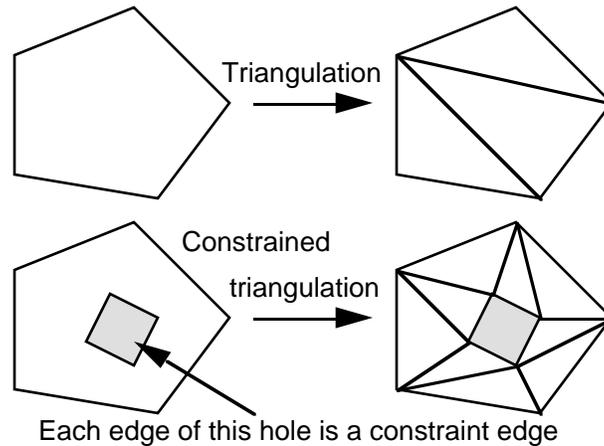

Figure 3.7: Examples of triangulation and constrained triangulation.

4. Overview of Polygonal Simplification Algorithms

This section presents several polygonal simplification algorithms, each having its own input criteria and advantages or disadvantages. A majority of the algorithms assume that the input model contains only triangular polygons. Because of this assumption and in order to make reading easier, this section replaces the words "triangular polygon" by the word "triangle."

This paper does not compare the performance of the algorithms for two reasons. First, testing the various algorithms requires obtaining working code. Second, evaluating the performance of a simplification algorithm involves using an error metric to determine the goodness of fit between the simplified and original models. Since a standard metric does not exist, comparing two algorithms is a subjective process. See the 5. Possible Future Work section for more details on error metrics.

4.1 Classification of Algorithms

Even though each algorithm is unique, this paper classifies them by the method they use to produce a simplified model. The three classifications are adaptive subdivision, geometry removal, and sampling. Note that even though the diagrams below show that the three techniques can produce the same simplified model, this event rarely happens in practice. Also, some algorithms combine elements of two classifications.

4.1.1 Adaptive Subdivision

An adaptive subdivision algorithm starts with a very simple base model and recursively subdivides it, adding more and more detail to local areas of the model at each step. Once the subdivided model approximates the original model to some user-specified degree, the algorithm stops. Figure 4.1 shows a subdivision algorithm simplifying a height field.
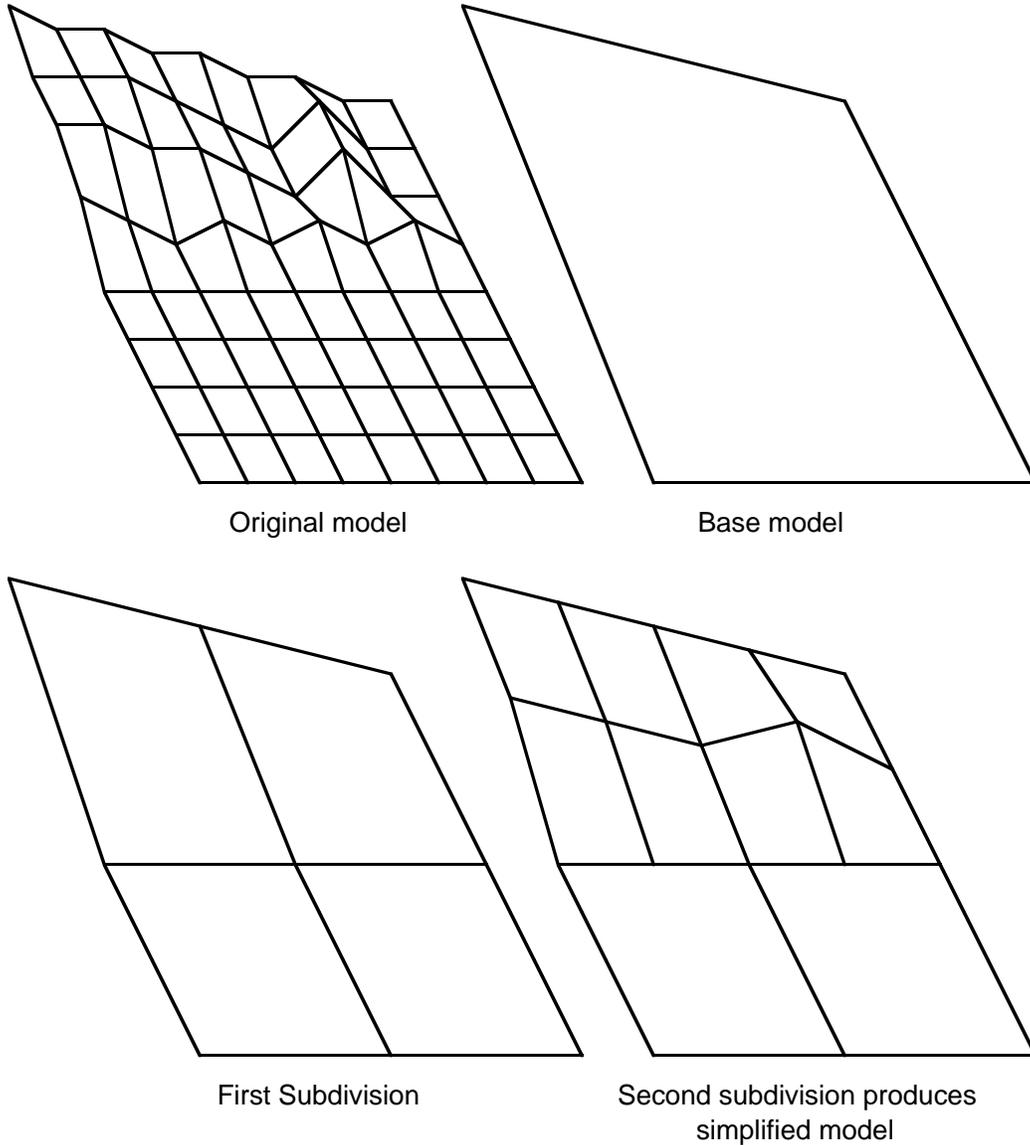

<div align="center">Original model   Base model</div>

<div align="center">First Subdivision   Second subdivision produces
simplified model</div>

Figure 4.1: Adaptive subdivision simplifies a height field.

4.1.2 Geometry Removal

A geometry removal algorithm starts with the original model and simplifies by repeatedly removing faces or vertices from the model. It removes geometry until the model can no longer be simplified and still meet some user-specified degree of approximation to the original. Most of these algorithms do not allow vertex and face removal operations that cause topology violations. Figure 4.2 shows a geometry removal algorithm simplifying a height field.
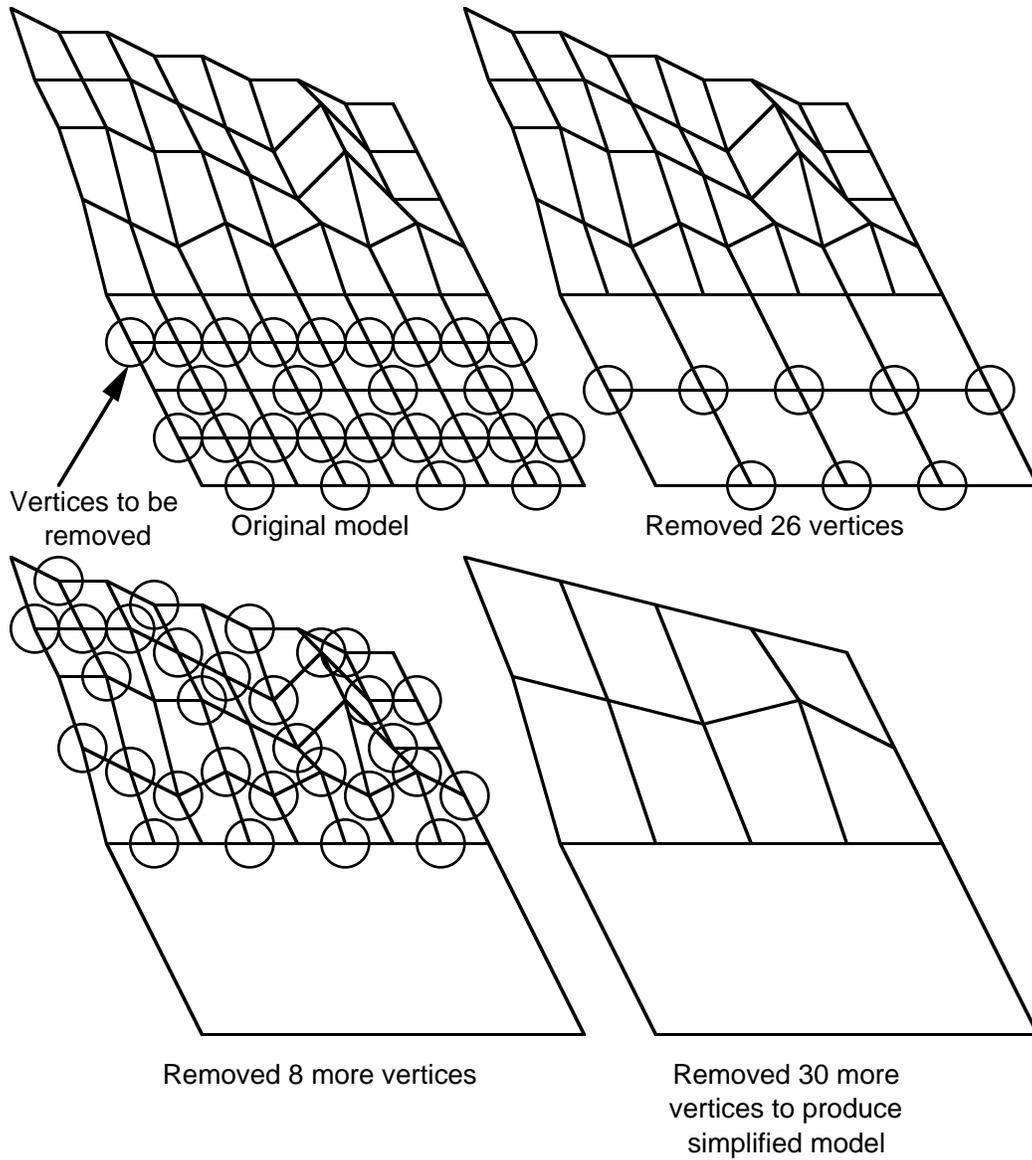
<div align="center">7</div>

Vertices to be removed   Original model   Removed 26 vertices

Removed 8 more vertices   Removed 30 more vertices to produce simplified model

Figure 4.2: Geometry removal simplifies a height field. The circled vertices are tagged for removal.

4.1.3 Sampling

A sampling algorithm initially samples the geometry of the original model by either taking a certain number of random points from its surface, or by overlaying the model with a three-dimensional grid and sampling each box of the grid. The algorithm tries to create a simplified model that closely fits the sampled data. The user can control the amount of sampling by specifying the number of points to sample or the dimensions of the overlay grid. Figure 4.3 shows a sampling algorithm simplifying a height field.
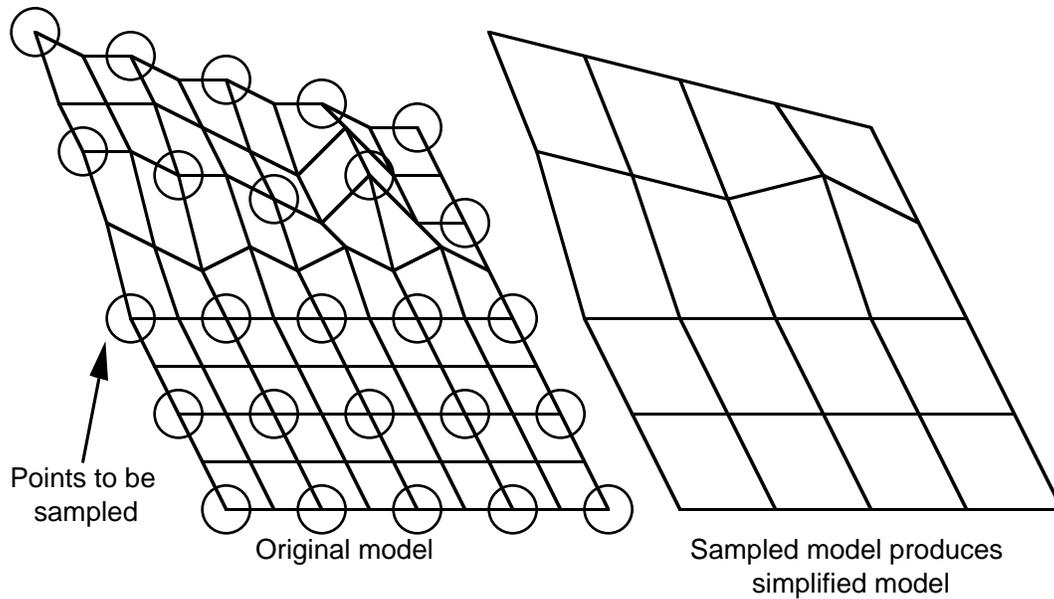
Figure 4.3: Sampling simplifies a height field. The circled
vertices are the samples that will make up the simplified model.

4.2 Topology Preserving Algorithms

A topology preserving algorithm preserves either local or global topology. Each
algorithm below preserves at least local topology. This section specifically notes
algorithms that also preserve global topology.

Many of the algorithms below expect a well-defined input model, also known as a mesh.
These algorithms reject any models containing degeneracies, such as coincident polygons
or edges adjacent to more than two faces.

4.2.1 Simplification of Objects Rendered by Polygonal Approximations

This algorithm, created by Michael J. DeHaemer, Jr. and Michael J. Zyda [2], uses the
technique of adaptive subdivision. It assumes the input model comes from three-
dimensional scanner output. A scanner samples a model by returning a height value for
each point along a regular grid of points. This method takes advantage of the regular
layout of its input and thus does not simplify arbitrary three-dimensional models.

The user specifies an error tolerance in terms of Euclidean distance. The algorithm
computes a simplified model that is within this error tolerance of the original. If the
error tolerance is high, this method greatly simplifies the model. If the error tolerance
is low, it marginally simplifies the model.

4.2.1.1 Algorithm Details

The process begins with one quadrilateral polygon representing the whole height map. If
this polygon satisfies the error tolerance, then the algorithm halts. Otherwise, the
method subdivides any polygon not within the error tolerance. It recursively applies
this subdivision rule until all polygons meet the error specification.

The algorithm can subdivide a polygon in several ways. It can split the polygon into two or four axis aligned polygons. If two, then the split occurs along the largest dimension of the polygon. The method produces axis aligned polygons to keep the regular grid structure of the data intact. It can choose the point of subdivision as the center point, the first interior point that is not within the error tolerance, the interior point of maximum error, or the interior point of maximum curvature. Figure 4.1 shows an example of this algorithm when it splits polygon into four axis aligned polygons and it chooses the center point as the point of subdivision.

The algorithm can recursively subdivide neighboring polygons a different number of times, creating gaps between them. When rendered from certain viewing angles, even small gaps become very noticeable. To avoid this problem, this method fills in a gap with a polygon. Figure 4.4 shows an example of using polygons to fill in gaps.

Original model

First Subdivision

Second subdivision produces
simplified model with gaps

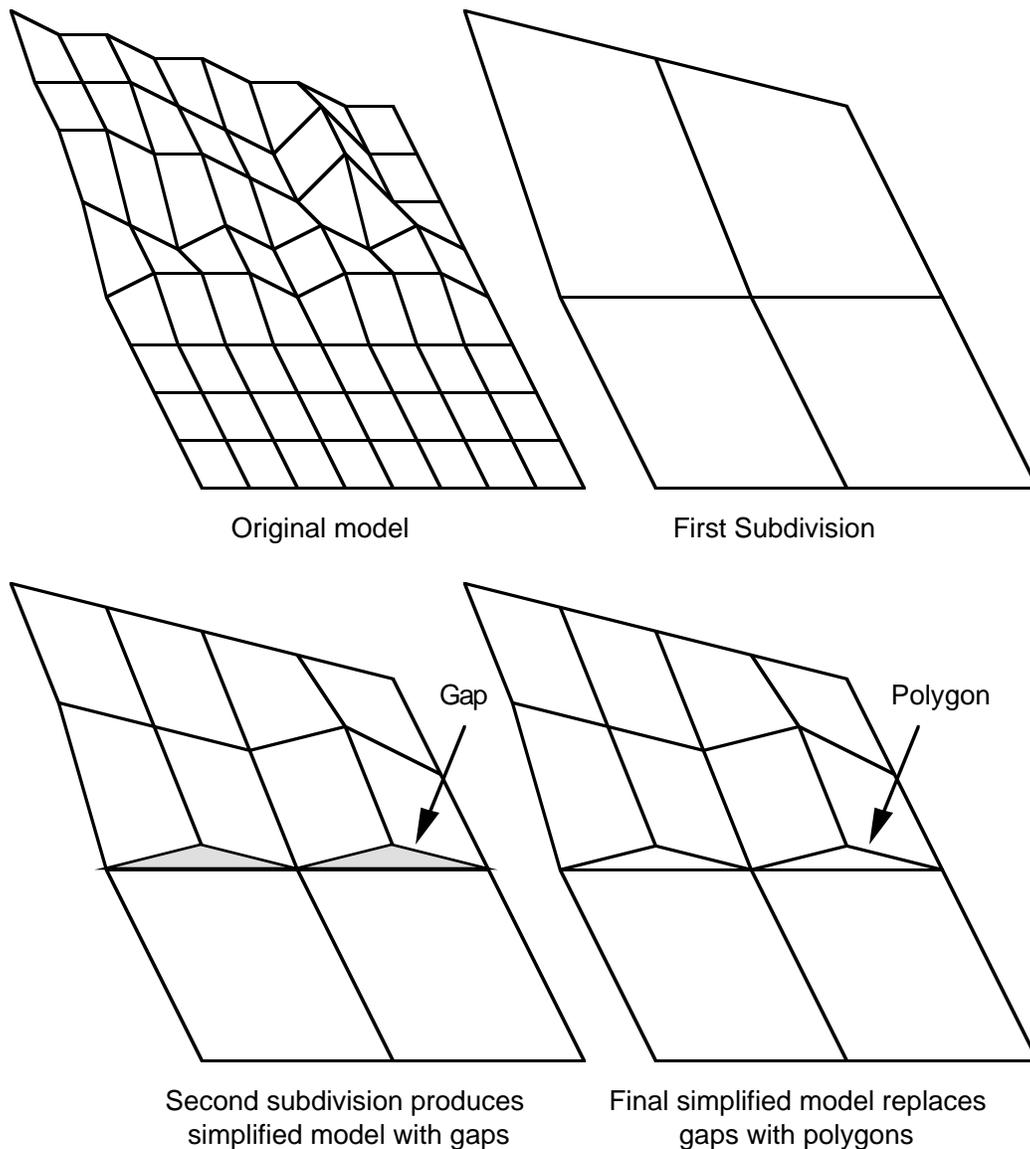Final simplified model replaces
gaps with polygons

Figure 4.4: Gaps can appear after subdividing neighboring
polygons a different number of times, but polygons can fill them
in to eliminate visible cracks during rendering.

4.2.2 Geometric Optimization

This geometry removal algorithm, created by Paul Hinker and Charles Hansen [7], excels at eliminating over-tessellated coplanar surfaces of polygons. Replacing numerous coplanar polygons by a few does not change the appearance of a model. Unfortunately, models with high curvature do not usually contain large areas of coplanar polygons.

The vertices in the simplified model are a subset of vertices in the original model. This feature allows normal, color, and texture information of the vertices to remain unchanged after the simplification process.

4.2.2.1 Algorithm Details

The method constructs nearly coplanar sets of polygons. It groups together neighboring polygons with roughly the same normal. Each group has a representative normal that can be calculated by averaging all of the normals of the polygons in the group. For each polygon, the algorithm compares the normal of the polygon and the representative normal of any neighboring group. If the two normals are within some angular tolerance, then it adds the polygon to the group and updates the representative normal. At the end of this process, the groups of polygons are the nearly coplanar sets.

Next, the algorithm creates segment lists. For each coplanar set, it adds all edges that make up any polygon in the set to a segment list. It sorts the segments by endpoint location which allows duplicate segments in the list to be removed efficiently. This removal eliminates all segments not on the boundary of the coplanar set.

For each coplanar set, the algorithm forms a boundary polygon. To trace a path around the boundary of a coplanar set, it links together segments that share endpoints in the segment list of the set. Care must be taken to recognize interior holes of the coplanar set.

Then, this method triangulates each boundary polygon. This operation requires constrained triangulation because there could exist holes and concavities in the coplanar set. The triangulation method is greedy and traverses the vertices of the boundary polygon in order, keeping any triangle that does not violate a constraint. Although this greedy method can produce triangles with poor aspect ratio, it is very efficient and in general produces decent results.

At the end of this process, the algorithm has reduced each nearly coplanar set in size. It replaces the original coplanar sets of polygons with the simplified sets. Figure 4.5 shows this algorithm simplifying a height field model. Note that it can simplify arbitrary three-dimensional models, not just height fields.
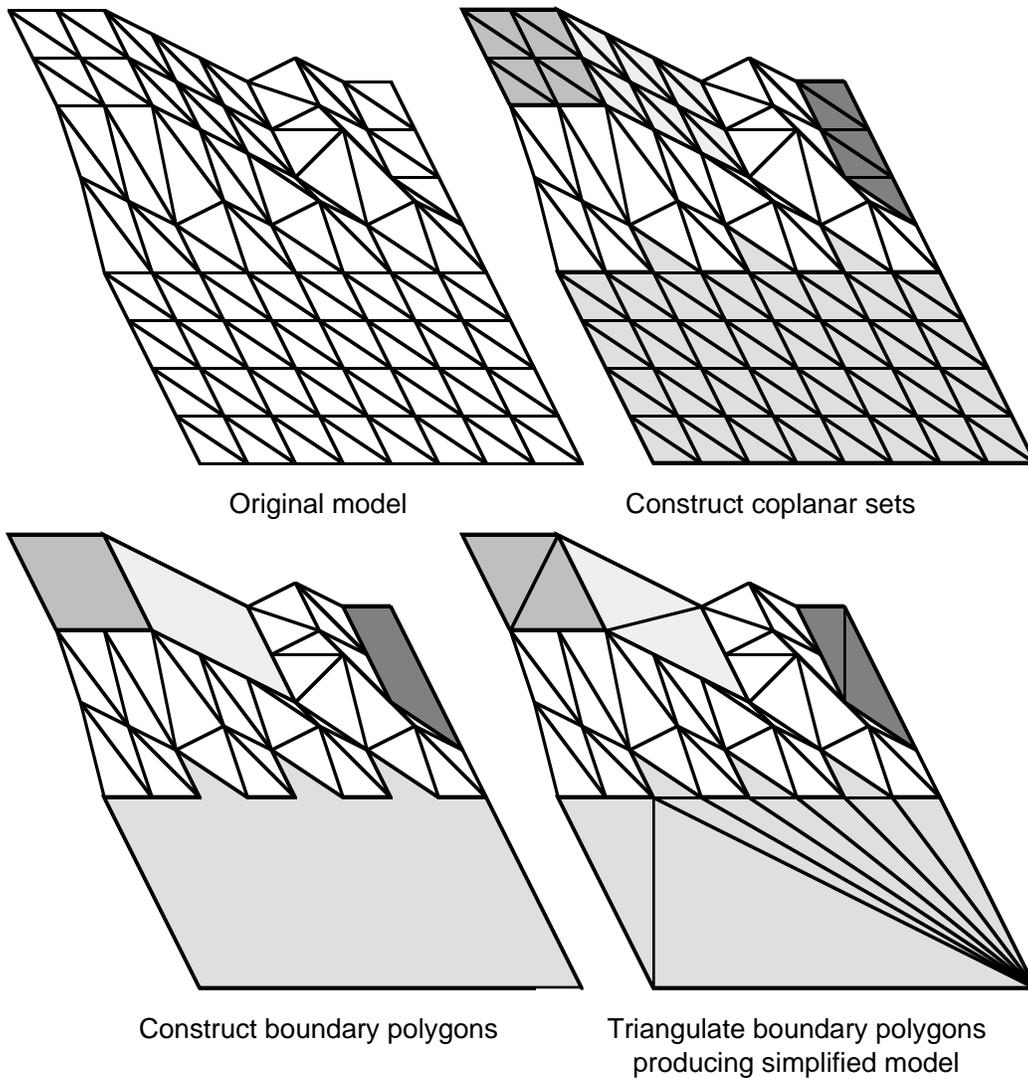
| Original model | Construct coplanar sets |

| Construct boundary polygons | Triangulate boundary polygons producing simplified model |

Figure 4.5: Geometric Optimization simplifies a height field.

4.2.3 A Data Reduction Scheme for Triangulated Surfaces

This geometry removal algorithm, created by Bernd Hamann [4], weights triangular polygons in the original model according to local curvature. It simplifies regions of high curvature very rarely and regions of low curvature often. Thus, it simplifies coplanar groups of polygons inherently.

The user specifies a percentage of triangles to be removed from the model. This feature allows the user to create several different levels of detail of the model, each with a specific number of polygons.

4.2.3.1 Algorithm Details

The method weights the triangles in the original model. For each vertex, it calculates an estimate to the local curvature using interior angles of adjacent faces. It determines the weight of a triangle by averaging the curvature estimates of each vertex of the triangle.

An iteration of the algorithm initially finds the triangle with lowest weight and removes it. If this deletion does not preserve local topology, then the algorithm does not perform the operation. This method uses several different classifications of the triangle and its neighboring triangles to detect violations of local topology.

Next, the algorithm replaces the triangle with a single point. It places the point so that it lies on a least squares approximation to the local surface. It calculates the surface approximation using the neighboring triangles of the deleted triangle.

Then, this method removes the neighbors of the deleted triangle and triangulates the resulting hole using the single point as the center vertex. Using edge swapping, it optimizes the aspect ratios of the triangles produced in the triangulation. It calculates the weights of the newly created triangles using the previously described technique. This calculation marks the end of one iteration.

The algorithm iterates until either it removes the specified percentage of triangles, or it detects that it cannot remove any more triangles because of local topology constraints. Figure 4.6 demonstrates one iteration of this algorithm.
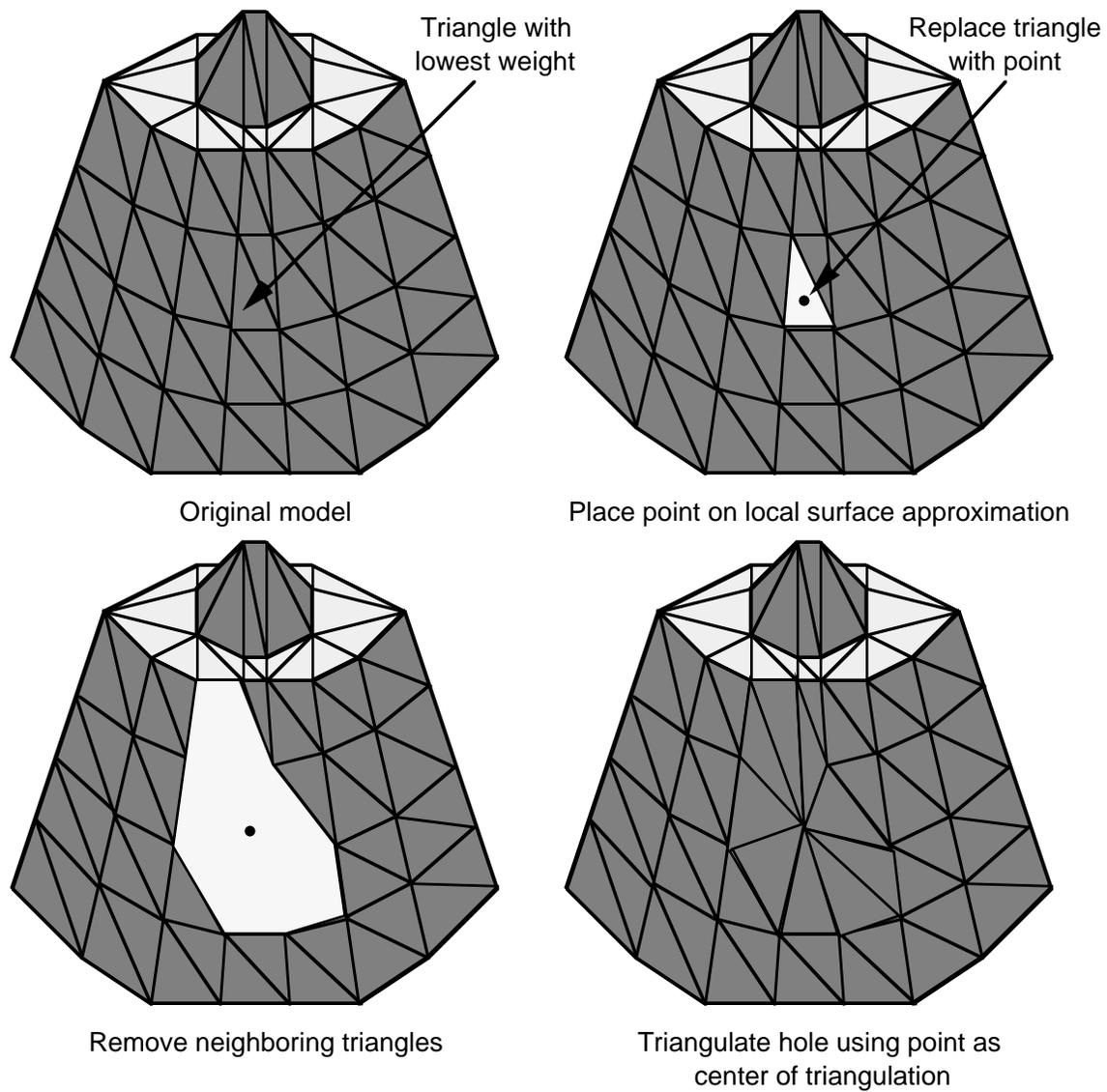
Figure 4.6: One iteration of A Data Reduction Scheme for Triangulated Surfaces. After each hole filling, the algorithm optimizes the triangulation using edge swapping.

4.2.4 Re-Tiling Polygonal Surfaces

This sampling and geometry removal algorithm, created by Greg Turk [16], performs better on models that represent curved surfaces rather than models containing sharp discontinuities. For example, this technique simplifies a rabbit well but a house model poorly.

The user specifies the number of vertices in the final output model. This feature allows creation of several levels of detail of the model, each with a specific number of vertices.

4.2.4.1 Algorithm   Details

The algorithm randomly distributes the user-specified number of vertices over the surface of the model. Since this distribution depends on the surface area of each triangle, it usually places more vertices in large triangles than small ones.

Next, the algorithm simulates repulsion forces between vertices over the surface of the model. If two vertices are close to each other across the surface, they repel each other. The algorithm constrains the movement of the vertices to the surface of the model so the repulsion forces more evenly distribute the vertices on the surface. This step refines the positions of the randomly distributed vertices.

Using a method called mutual tessellation, the algorithm reconstructs a new model from the distributed vertices. Using constraints, it triangulates each polygon of the original model along with any distributed vertices that lie inside the polygon. The resulting model contains both the original and distributed vertices. Then, the algorithm removes the original vertices one by one. It triangulates the hole that appears after each vertex removal. It does not remove a vertex if it cannot preserve the local topology. To decrease the number of triangles created with poor aspect ratio, the algorithm executes a local re-triangulation step before it removes any vertices. This mutual tessellation step produces the desired simplified model. Figure 4.7 shows how this algorithm might work on a simple model.

Random distribution of points

Evenly distribute points using repulsion forces

Triangulate using distributed points

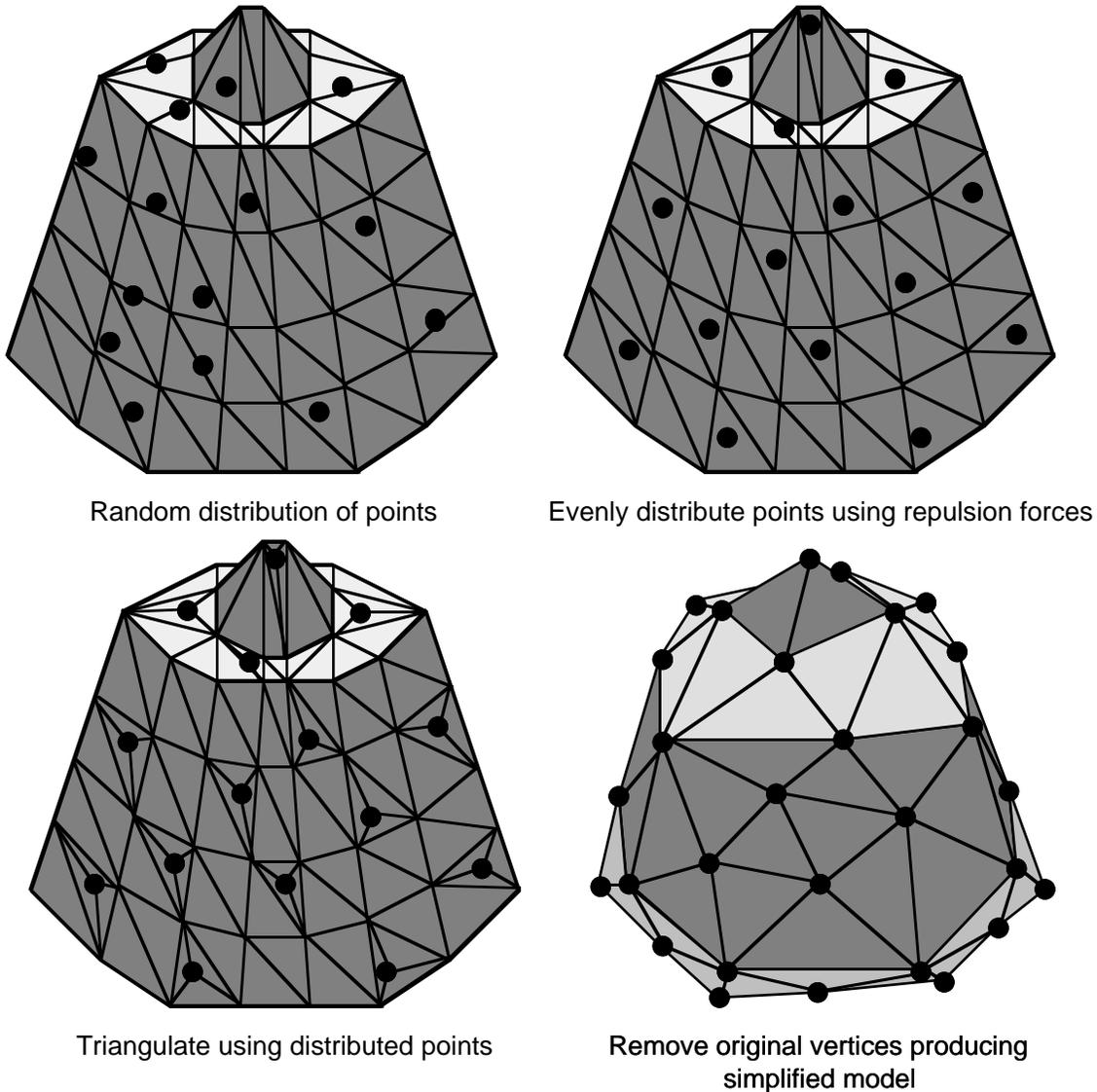Remove original vertices producing simplified model

Figure 4.7: Re-Tiling Polygonal Surfaces simplifies a simple model.

There are some possible enhancements to this algorithm. The first locates areas of high curvature and designates them as low repulsion regions. Thus, in the repulsion stage, more vertices move to and remain in areas of high curvature. The resulting model has more vertices in high curvature areas and better represents the original shape of the model. The second enhancement defines a correspondence between different levels of detail of the object. Creating this correspondence involves using mutual tessellation to define where the vertices of a model lie on the levels of detail above and below it. Once the algorithm calculates this correspondence, it can smoothly interpolate between the levels of detail.

4.2.5 Decimation of Triangle Meshes

This geometry removal algorithm, created by William J. Schroeder, Jonathan A. Zarge, and William E. Lorensen [15], performs well on models produced by the Marching Cubes

algorithm. Since Marching Cubes produces a large number of polygons, simplification is often a desirable post-processing tool.

The vertices of the simplified model are a subset of the original.

The user specifies a distance error term. The algorithm can remove a vertex only if it is within this distance of an approximation to its local topology. Unfortunately, this method calculates distance error using the current simplified model and the model of the previous iteration. To be more accurate, it should compare the current simplified model with the original.

4.2.5.1 Algorithm Details

An iteration of this algorithm initially picks a vertex in the model. Using information from adjacent faces, it characterizes the local vertex geometry and topology. It considers the vertex for removal only if this operation preserves local topology.

At the chosen vertex, the algorithm evaluates the decimation criteria. It calculates the distance from the vertex to the average plane of its adjacent vertices. If this distance is less than the user-specified error term, it removes the vertex.

The algorithm removes the vertex and a hole appears. It uses a technique called loop splitting to triangulate the hole. Loop splitting attempts to find the split across a polygon that produces triangles with optimal aspect ratios.

This algorithm iterates until there are no more vertices that meet the decimation criteria. Figure 4.8 demonstrates one iteration of the algorithm.
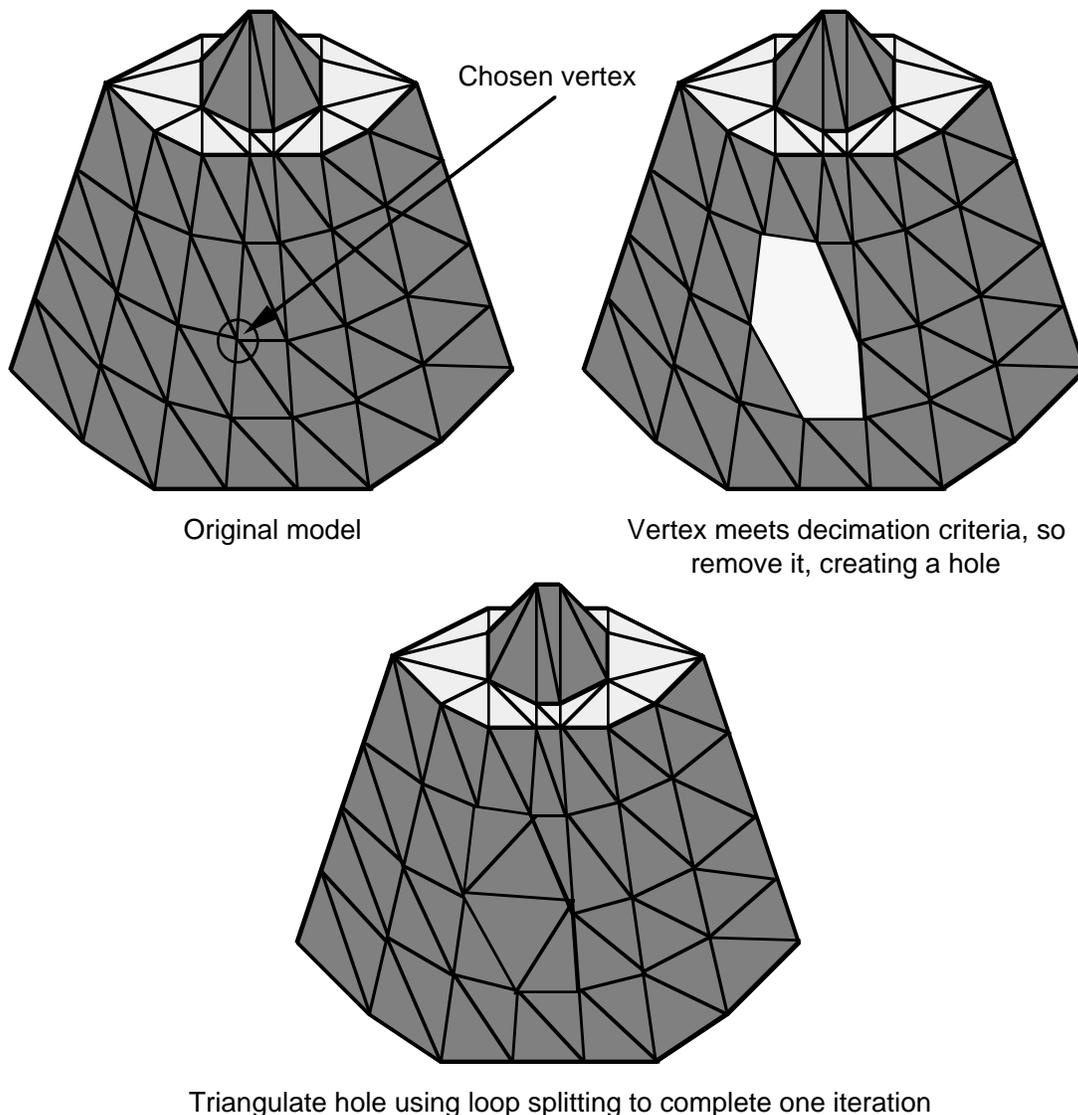
Chosen vertex

Original model

Vertex meets decimation criteria, so
remove it, creating a hole

Triangulate hole using loop splitting to complete one iteration

Figure 4.8: One iteration of Decimation of Triangle Meshes.

4.2.6 Mesh Optimization

The idea for this sampling algorithm, created by Hugues Hoppe, Tony DeRose, Tom
Duchamp, John McDonald, and Werner Stuetzle [8], originated from a surface
reconstruction algorithm by the same authors [9]. From a collection of points, a surface
reconstruction algorithm creates a three-dimensional mesh.

The algorithm simplifies a model by minimizing an energy function. Because of this
minimization, the method provides four visual benefits.   It uses many polygons to
represent areas of high curvature. It aligns long edges along directions of low curvature.
It adjusts the aspect ratios of triangles to match the local curvature. Finally, it places
edges and vertices near sharp features of the original model.

The user selects a parameter to control the degree of simplification performed on the
model.

4.2.6.1 Algorithm Details

The algorithm uses an energy function equal to the sum of three components: a distance term, a vertex term, and a spring term. The distance term is proportional to the sum of squared distances between the sample points and the mesh being created. It measures the closeness of fit between the original and simplified mesh. The vertex term is proportional to the number of vertices in the simplified mesh and therefore penalizes a mesh containing too many vertices. The spring term simulates the effect of placing springs on the edges of the mesh being created. As the algorithm iterates, a gradual change of the spring term guides the energy function to a local minimum.

The initial step randomly samples a specific number of points from the original mesh. It adds the original vertices of the mesh to this list of points.

An iteration of this algorithm executes two nested loops. The outer optimization loop minimizes the energy function using three operations: edge collapse, edge swap, and edge split. Figure 4.9 shows what these operations do. The loop performs one of these three operations randomly and checks for local or global topology violations. If it detects a violation, the operation is undone. The inner optimization loop minimizes the distance and spring term. It optimizes the positions of the vertices after a random operation in the outer loop completes. It does not add or remove vertices, so the vertex term remains the same.

Operation applied to this edge

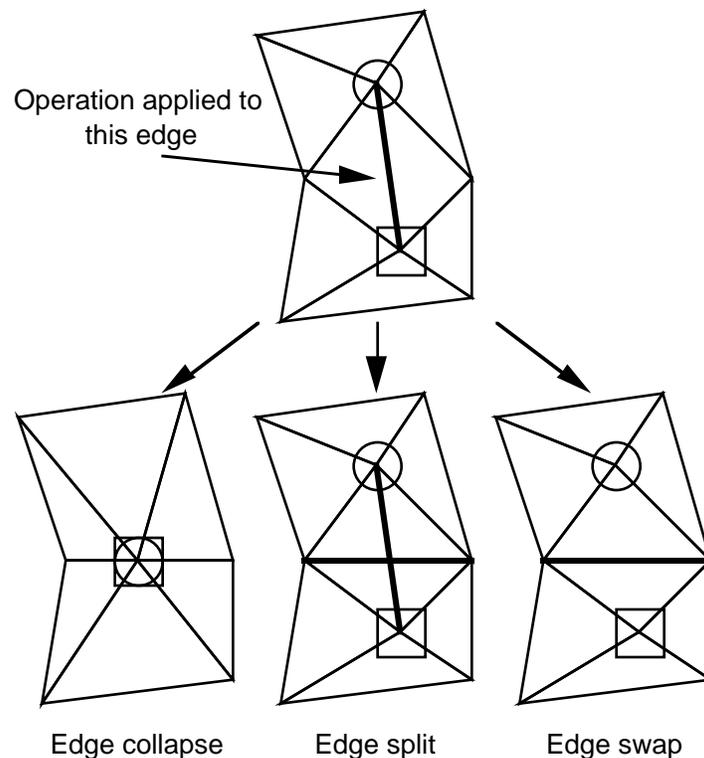Edge collapse   Edge split   Edge swap

Figure 4.9: The outer optimization loop applies these operations
randomly to edges.

After each iteration, the algorithm gradually decreases the spring term to guide the optimization process to a local minimum. It halts after a fixed number of iterations.

4.2.7 Superfaces: Polygonal Mesh Simplification with Bounded Error

This adaptive subdivision algorithm, created by Alan D. Kalvin and Russell H. Taylor [10], uses a greedy method for its time-critical section. Thus, it runs very quickly compared to other methods that guarantee error tolerance bounds relative to the original mesh.

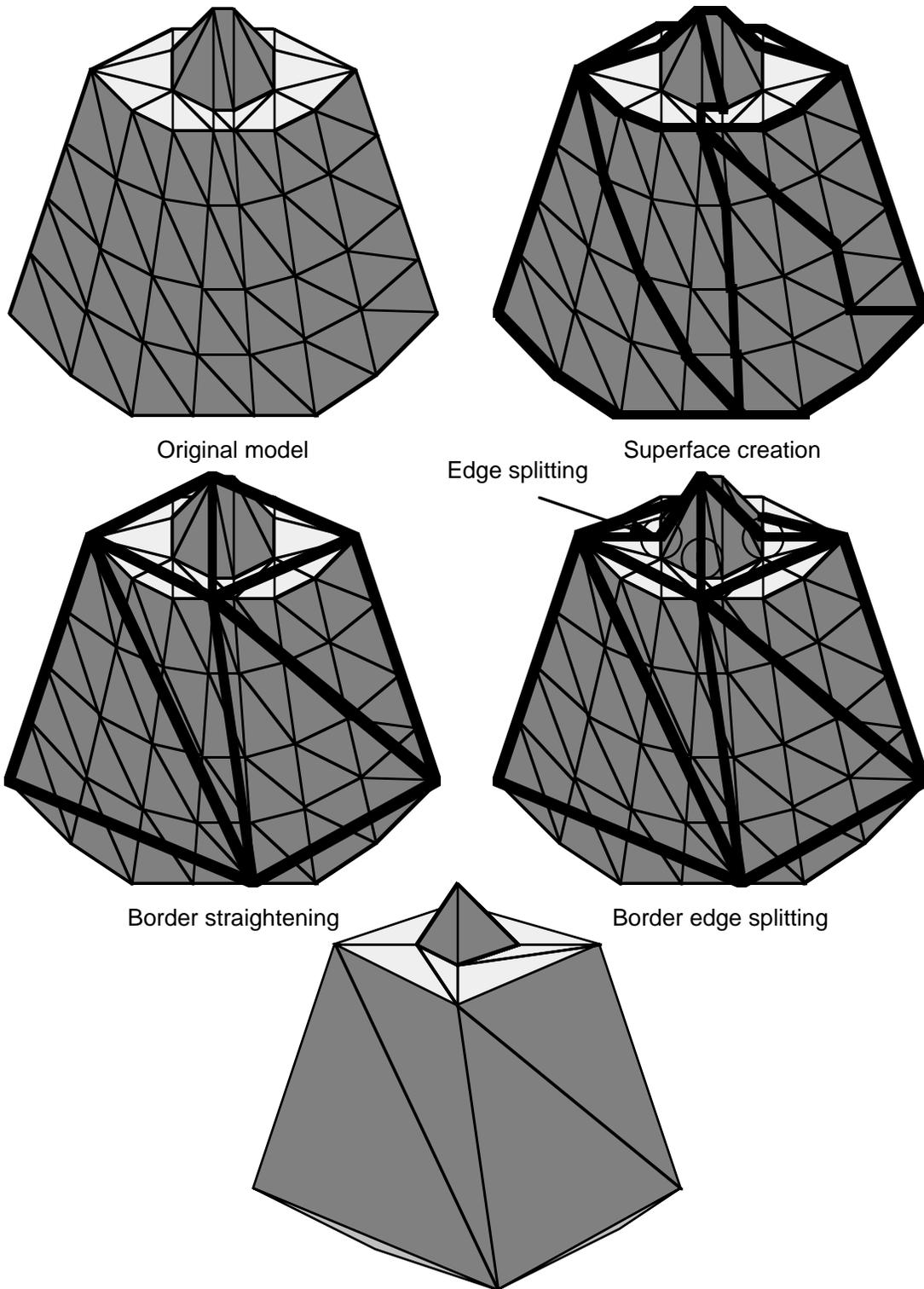The vertices of the simplified model are a subset of the original.

The user specifies a distance error bound. The algorithm guarantees that every vertex of the original mesh is within this distance of the simplified mesh and vice versa.

4.2.7.1 Algorithm Details

The algorithm consists of three main steps: superface creation, border straightening, and superface triangulation. Superface creation selects a seed face as its initial superface. The algorithm attempts to grow adjacent faces onto this superface. A candidate face is adjacent to one of the faces in the superface. If a candidate face meets topology, aspect ratio, and error bound requirements, then the algorithm adds it to the superface. When the superface can no longer grow, the method creates a new superface using another seed face. This process continues until all faces are members of superfaces. Creation of superface holes, or superfaces within other superfaces, is a problem yet to be solved by this algorithm.

Border straightening straightens the border edges of the superfaces created in the first step. Since superfaces are a collection of faces, their borders are most likely jagged. A superface is adjacent to another superface if a face in one of the superfaces is adjacent to a face in the other. If a superface is adjacent to another, the algorithm straightens the border between the two by connecting the starting and ending points of the border with a straight edge. If this edge violates the error bound, the process recursively splits the border edge in half until the requirement is met.

Superface triangulation triangulates every superface. The algorithm tries to find a point in the interior of the superface that meets specific triangulation criteria. If it cannot find this triangulation point, it recursively subdivides the superface until triangulation points are found for each subdivision. It finishes by triangulating the superfaces using their respective triangulation points as their center vertices. Figure 4.10 shows the execution of this algorithm on a simple model.

Original model

Superface creation

Edge splitting

Border straightening

Border edge splitting

Superface triangulation producing simplified model

Figure 4.10: The Superfaces algorithm simplifies a model.

4.2.8 Hierarchical Geometric Approximations

This geometry removal algorithm, created by Amitabh Varshney [17], generates two offset surfaces for an input mesh. It creates one surface on the outside of the mesh and another on the inside. It uses these offset surfaces to guarantee the preservation of global topology.

The vertices of the simplified model are a subset of the original.

The user specifies a distance error tolerance. The algorithm uses this value to guarantee the simplified model is within this distance from the original and vice versa.

4.2.8.1 Algorithm Details

The initial step constructs the inside and outside offset surfaces of the model. It creates the outer offset by extending the position of each vertex by its normal scaled by the error tolerance. The algorithm creates the inner offset similarly except each normal is scaled by the negative of the error tolerance. Each triangle together with its offset extensions forms a fundamental prism. Using these prisms, the algorithm detects self-intersections of the outer and inner offset surfaces. If there exist areas of self-intersection of the offset surfaces, this method scales the offset surfaces around these regions until the intersections disappear. Thus, the error tolerance around certain portions of a model may be very small in comparison to the specified error tolerance.

Next, the algorithm computes candidate triangles using the list of vertices inside the offset surfaces. A candidate triangle is one that lies completely within the offset surfaces and its vertices are a subset of the original model. Using the offset surfaces as occluders, the algorithm calculates vertex to vertex and edge to edge visibility. It finds a candidate triangle when every edge of the triangle is visible to its opposite vertex.

For each candidate triangle, the algorithm creates an association with the vertices of the original model. It associates a vertex with a candidate triangle if the triangle covers the vertex. A triangle covers a vertex if the projection of the vertex onto the plane of the triangle lies within the triangle.

The author shows that the problem of picking a set of candidate triangles that creates a complete topologically preserving mesh is NP-complete. Therefore, the algorithm uses a practical greedy approach to pick which candidate triangles are included in the simplified model. At each iteration, it maintains a complete mesh. An iteration involves choosing an association between a collection of vertices and a candidate triangle that maximizes the number of vertices. If introducing this candidate triangle in the model produces a topological or triangulation violation, it discards the triangle. Otherwise, it adds the candidate triangle and removes the associated vertices from the simplified model. This process creates holes that are subsequently triangulated. This greedy approach produces nearly optimal results. Figure 4.11 demonstrates the execution of this algorithm on a simple model.

Original model

Construct offset surfaces, only outside surface is visible in this diagram

Greedily choose the triangle that stays within the offset surfaces and has the maximum number of associated vertices

Hole created

Greedily pick candidate triangles and remove their associated vertices, note that any resulting holes must be filled

After several iterations of picking candidate triangles, the algorithm produces a simplified model with triangles that are contained within the offset surfaces
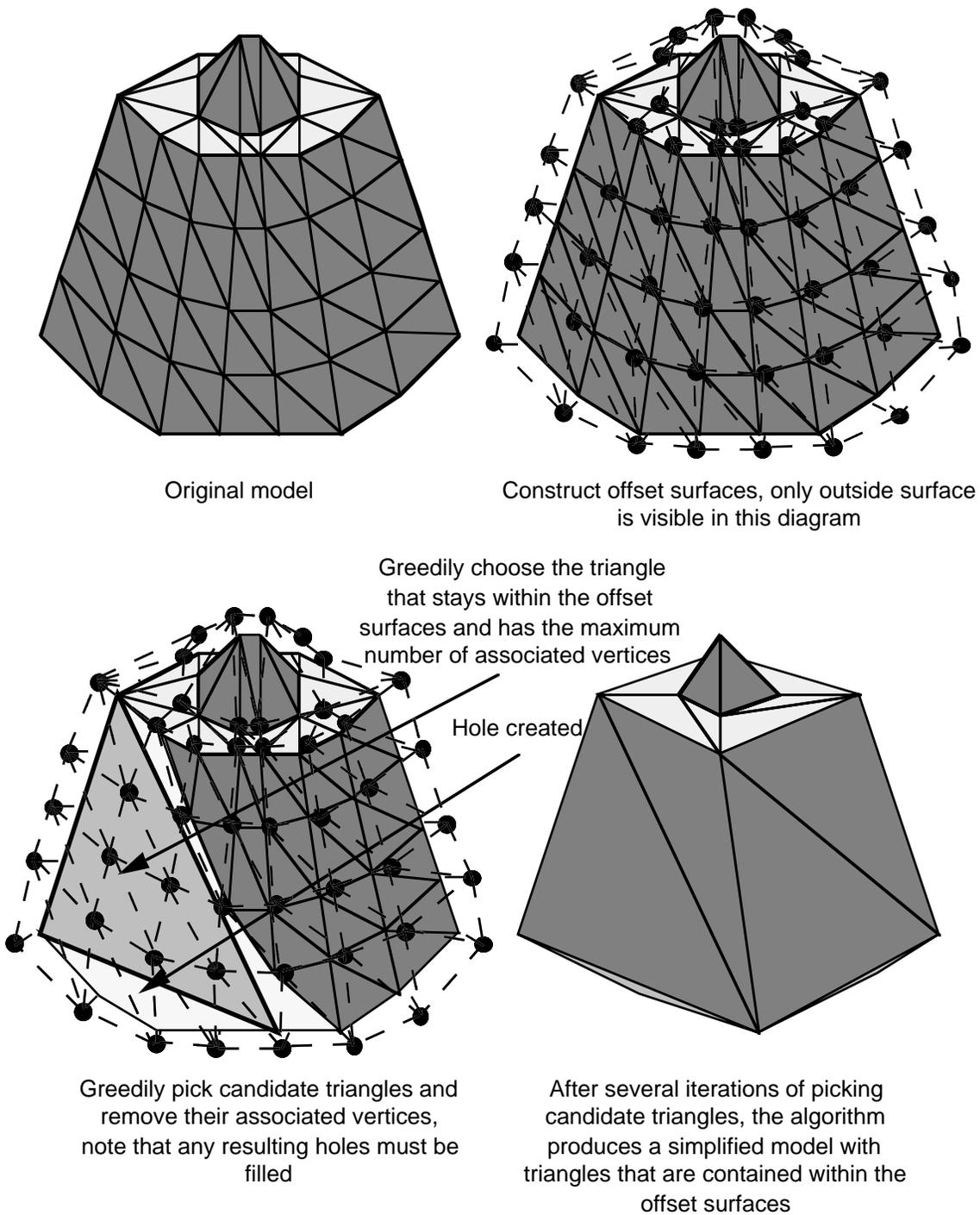
Figure 4.11: The execution of Hierarchical Geometric Approximations on a simple model.

There are possible enhancements to this algorithm. If it allows the error tolerance to vary across the surface of the model, the user can specify specific regions to be simplified greatly or marginally. Also, it can preserve sharp edges in the model by adding local occluders to the visibility calculations during candidate triangle creation. These occluders guarantee that candidate triangles cannot cover any vertices that are part of a sharp edge, preventing the algorithm from removing these edges.

4.2.9 Multiresolution Analysis of Arbitrary Meshes

In previous work, Michael Lounsbery, Tony DeRose, and Joe Warren used wavelet theory to create a multiresolution representation of a mesh [12]. Building on top of this work, the adaptive subdivision algorithm, created by Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle [3], uses wavelets to compactly store multiple levels of detail within one model. Smoothly interpolating between these levels of detail simply requires adding or subtracting wavelet coefficients. Editing the model at different levels of details is another benefit of using a wavelet representation.

If the user specifies a distance error tolerance, the algorithm uses an appropriate number of wavelet coefficients to guarantee this error bound.

4.2.9.1 Algorithm Details

The algorithm tries to approximate the original mesh by one within the error tolerance and with the property of "subdivision connectivity." A complicated mesh derived from a simple base mesh by recursive subdivision exhibits subdivision connectivity. For this algorithm, replacing a triangle with four triangles is the only valid subdivision operation.

There are three steps to the algorithm: partitioning, parametrization, and resampling. Partitioning creates a base mesh that represents the lowest level of detail for the object. The algorithm grows a base mesh face from a single triangle of the original model. It adds to the base face all neighboring triangles that are closest to it by geodesic distance. When all base faces stop growing, they form Voronoi-like regions. The algorithm checks whether a Delaunay-like triangulation of a base face is valid. It corrects triangles with poor aspect ratios by subdividing base faces. In the end, each base face corresponds to one triangle in the lowest level of detail model. Ideally, this lowest level contains few triangles.

Next, the algorithm constructs a parametrization of the local surface for each base face. It creates a parametrization that is continuous at the boundaries of the base faces. Thus, the parametrization is globally continuous. This step tries to find a parametrization that varies little across the surface of a face, because then the wavelet representation of the surface requires very few coefficients.

Resampling recursively subdivides the base faces of the mesh. At each subdivision step, the algorithm splits a triangle into four triangles. The algorithm approximates the original mesh by perturbing the vertices of the subdivided faces according to the parametrization of each base face. It recursively subdivides the simplified mesh until the error tolerance is met. Thus, the resulting model has the property of subdivision connectivity. Figure 4.12 demonstrates this algorithm simplifying a model.
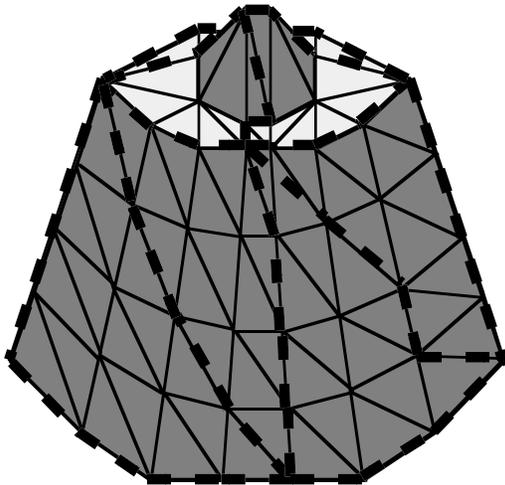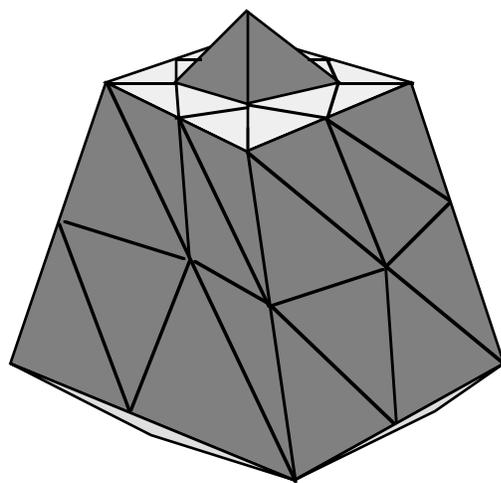
Original model showing outline
of base faces
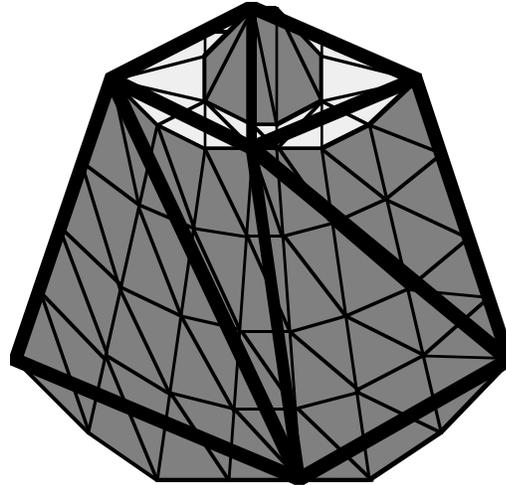
Partitioning
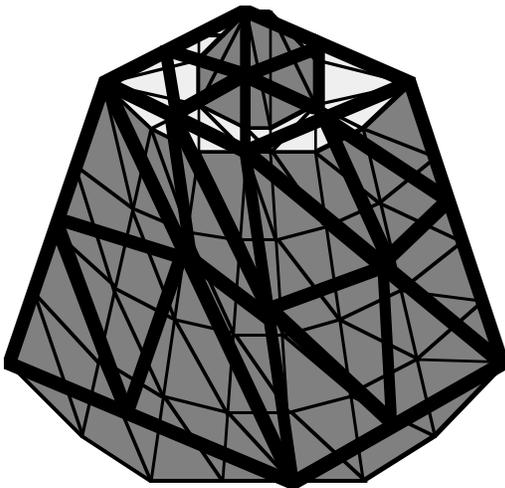
Resampling

At resampled points, use parametrization to
approximate mesh, producing simplified model

Figure 4.12: Multiresolution Analysis of Arbitrary Meshes
simplifies a model.

4.3 Topology Simplifying Algorithms

Topology simplifying algorithms do not guarantee the preservation of local or global topology of a model. Thus, holes in a model can suddenly disappear from one level of detail to the next. Because these algorithms have fewer constraints to satisfy, they can simplify models to a greater degree than topology preserving algorithms. Usually, an application uses these algorithms on objects or scenes when a desired frame rate is essential and degradation of visual quality is acceptable. Also, systems use topological simplification when models do not simplify otherwise. For example, a topology preserving algorithm has problems simplifying a model of a slice of swiss cheese. It cannot remove the holes in the swiss cheese without violating topology constraints.

However, a topology simplifying algorithm can fill in the holes of the cheese, thereby simplifying both the geometry and topology a great deal.

4.3.1 Multi-resolution 3D Approximations for Rendering Complex Scenes

This sampling algorithm, created by Jarek R. Rossignac and Paul Borrel [14], works on any input model, no matter how degenerate it may be. Thus, it is an attractive option to use on large scenes of "polygon soup," a collection of polygons that do not constitute a valid mesh.

The implementation of the algorithm is very efficient and runs faster than a topologically preserving algorithm. However, because it does not preserve topology nor guarantee error bounds, the models it produces are not as visually pleasing.

4.3.1.1 Algorithm Details

The algorithm works in four main steps: grading, clustering, synthesis, and elimination. The initial step grades each vertex in the original model. It assigns each vertex a weight based upon two perceptually important factors. It considers vertices important that have a high probability of being on the silhouette of an object from an arbitrary viewing angle. The algorithm calculates this factor using the inverse of the maximum angle between any pair of edges adjacent to the vertex. Thus, a vertex at the end of a spike has a high silhouette weighting. The algorithm deems vertices important that bound large faces of the original model. It calculates this factor using the maximum length of all edges adjacent to the vertex. This method calculates a final weighting for each vertex using a linear combination of these two factors.

The algorithm triangulates each face of the original model and then performs clustering that breaks up the bounding box of the model into uniform subboxes. In other words, clustering lays a three-dimensional grid of boxes over the model. The algorithm adds each vertex to the vertex list of exactly one subbox, namely the one that contains the vertex. This method greatly simplifies the model if the subboxes are large and minimally simplifies if the subboxes are small.

Synthesis collapses all the vertices contained in a subbox to the most important vertex in the subbox. The most important vertex is the vertex with highest weight. This step produces some triangles that degenerate to points or edges.

Elimination removes these degeneracies from the model using a simple test of the vertices of each of the triangles. If two or more vertices in a triangle collapse to the same vertex during synthesis, the algorithm eliminates the triangle. To complete the simplified model, it recalculates the normals of the resulting triangles. Figure 4.13 shows this algorithm simplifying a model.

Most important vertex in a subbox
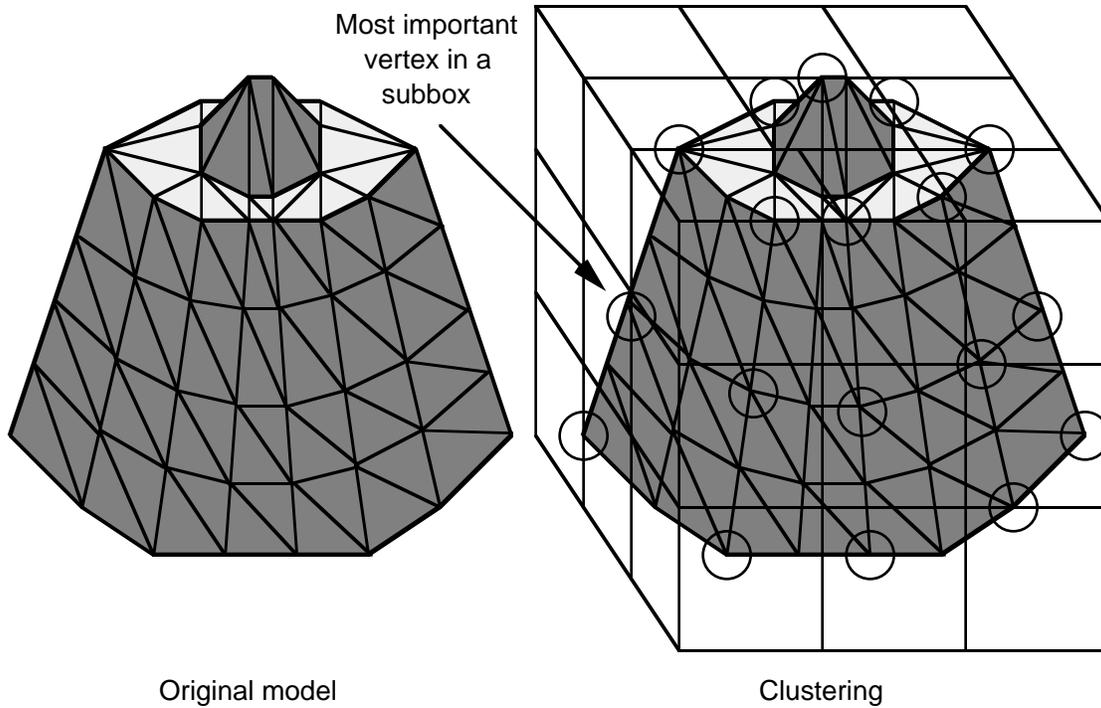
Original model

Clustering

Synthesis and elimination producing simplified model

Figure 4.13: The algorithm performs grading to weight vertices, clustering to determine the most important vertices in each grid box, synthesis to collapse the model, and elimination to remove degeneracies created by synthesis.

4.3.2 Voxel Based Object Simplification

This sampling algorithm, created by Taosong He, Lichan Hong, Arie Kaufman, Amitabh Varshney, and Sidney Wang [5], uses a signal processing approach that has a robust theoretical foundation. Thus, it eliminates high frequency detail of a model but has trouble with sharp discontinuities.

This algorithm simplifies geometry and topology in a controlled fashion. It assumes that the input model is a closed volume, meaning that the object has a distinct inside and outside.

4.3.2.1 Algorithm Details

The first step converts the object into a voxel representation. It places a three-dimensional voxel grid over the model. For each voxel, the algorithm samples the polygons of the model that lie within the voxel. Then, this method uses a low pass filter on these samples to produce a density value. This value determines whether a voxel is considered to be in the interior or exterior of the object.

Using Marching Cubes, the algorithm reconstructs a simplified polygonal model from the constructed volumetric representation. Since Marching Cubes creates redundant polygons, this method executes a topology preserving algorithm as a post-processing step. Figure 4.14 demonstrates this algorithm on a simple model.

Original model

Place three dimensional voxel
grid over model

Determine volumetric density values

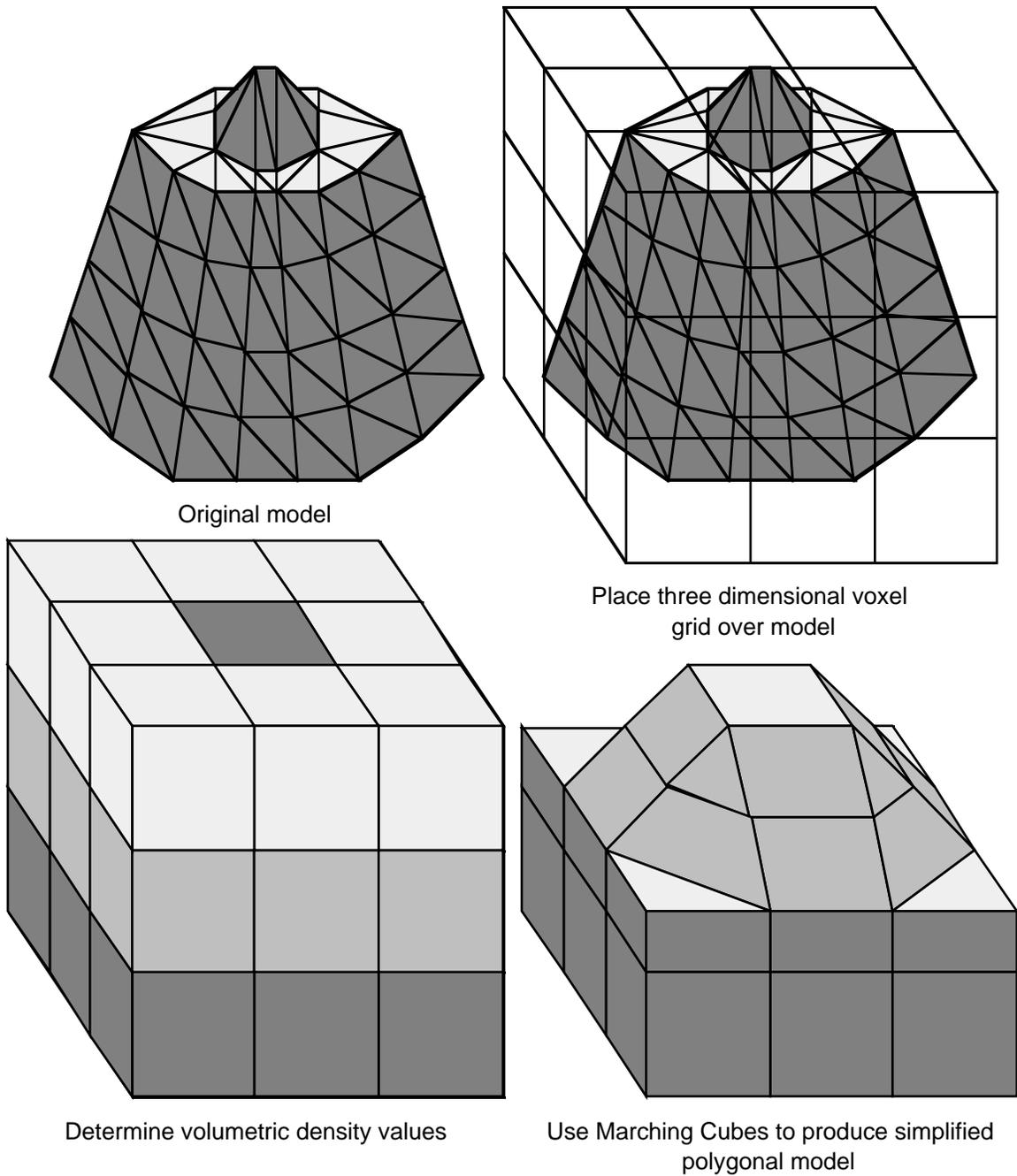Use Marching Cubes to produce simplified
polygonal model

Figure 4.14: Voxel Based Object Simplification simplifies a model.

If the voxels in the voxel grid are large, the algorithm greatly simplifies the input model. If the voxels are small, the algorithm minimally simplifies the model.

5. Possible Future Work

Polygonal simplification is a field of active research. This section presents several questions and ideas about possible future work.

5.1 Color and Texture

This paper presents several algorithms that all concentrate on simplification of geometry and topology of an object. Other factors affect the way an object appears besides its physical shape. How do color and texture affect the way an object looks? Can an algorithm use this extra information to produce better visual results?

5.2 Error Metrics

An error metric is a function that returns a single number that encapsulates the amount of difference between two polygonal models. A small value implies the models are very similar while a large value implies the models are very different. Can a simplification algorithm use an error metric to guide itself to better solutions? Varshney [17] attempts to address this problem by using an error metric based solely on distance. Is it possible to create a reasonably accurate metric that considers distance, color, and texture?

5.3 Help from other Fields

Several fields use the idea of simplification. Computational fluid dynamics has a history of producing efficient and useful mesh generation algorithms that create meshes from sample points. Generating the simplest and yet most useful mesh possible is a goal of these algorithms [13]. Radiosity and ray tracing algorithms both use simplification techniques to speed up visibility calculations. Collision detection uses simplification to avoid unnecessary computations. Can examining these fields produce new methods applicable to polygonal simplification?

5.4 Popping Minimization

Applications use discrete levels of detail to represent objects in a scene. When a level of detail changes, a noticeable visual pop sometimes occurs near the object. Since the application swaps discrete models, the human eye detects this discontinuity. One solution renders two levels of detail and alpha blends the two images. Unfortunately, this method requires rendering two transparent objects, a fairly expensive operation. Eck et al. [3] present a way of combining the levels of detail of an object into one representation. Using this technique, is it possible to switch between levels of detail continuously and in real time by adding or subtracting wavelet coefficients? Are there other methods which can minimize the popping effect?

5.5 Semantic Driven Simplification

Jason Weber and Joseph Penn use the semantics of a tree to guide the rendering of forests [18]. Their application renders a tree very far away as a single pixel. It renders a tree fairly far away as a line. As the tree gets closer, the application renders the tree with more and more graphics primitives specific to the type of tree. Using this method, it can render very realistic forests in almost real time. Can an application use semantic driven simplification for other objects?

5.6 Whole Scene Simplification

Most algorithms take a single object as input and produce a simplified version of it as output.   Can a simplification algorithm produces better results by examining the whole scene rather than individual objects of the scene? Is there some way to generalize simplification to large environments that contain numerous objects?

6. Conclusion

The recent explosion of papers dealing with polygonal simplification has provided a foundation for the field. However, as the 5. Possible Future Work section demonstrates, the simplification problem is far from being solved. Simplification techniques of today and the future must continue to help alleviate the problems associated with rendering complex models. Hopefully, this paper has provided the reader with some basic knowledge and motivation to explore the field.

7. Acknowledgments

I thank my family, Chris DiMattia, Cindy Hong, David Luebke, Cynthia Pettit, Greg Turk, and Stephen Weiss for proof-reading this paper. I thank the UNC Simplification group consisting of Rui Bastos, Jon Cohen, Merlin Hughes, Rob Katz, David Luebke, Dinesh Manocha, Greg Turk, and Hansong Zhang for sharing knowledge on this subject. The cat model in Figure 3.3 comes from the University of Washington.

References

[1] Clark, James H. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547-554, October 1976.

[2] DeHaemer, Jr., Michael J. and Michael J. Zyda. Simplification of Objects Rendered by Polygonal Approximations. *Computer & Graphics*, 15(2):175-184, 1991.

[3] Eck, Matthias, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution Analysis of Arbitrary Meshes. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 173-182, August 1995.

[4] Hamann, Bernd. A Data Reduction Scheme for Triangulated Surfaces. *Computer Aided Geometric Design*, 11(2):197-214, April 1994.

[5] He, Taosong, Lichan Hong, Arie Kaufman, Amitabh Varshney, and Sidney Wang. Voxel Based Object Simplification. *Visualization '95 Proceedings*, 296-303, 1995.

[6] Heckbert, Paul S. and Michael Garland. Multiresolution Modeling for Fast Rendering. *Proceedings of Graphics Interface '94*, 43-50, May 1994.

[7] Hinker, Paul and Charles Hansen. Geometric Optimization. *Proceedings of Visualization*, 189-195, October 1993.

[8] Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh Optimization. *Computer Graphics (SIGGRAPH '93 Proceedings)*, 19-26, August 1993.

[9] Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface Reconstruction from Unorganized Points. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):71-78, July 1992.

[10] Kalvin, Alan D. and Russell H. Taylor. Superfaces: Polygonal Mesh Simplification with Bounded Error. Technical Report RC 19808 (#87702), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10958, 1994.

[11] Lorensen, William E. and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(3):163-169, July 1987.

[12] Lounsbery, Michael, Tony DeRose, and Joe Warren. Multiresolution Analysis for Surfaces of Arbitrary Topological Type. Technical Report 93-10-05b, Department of Computer Science and Engineering, University of Washington, January 1994.

[13] Mavriplis, D. J. Unstructured Mesh Generation and Adaptivity. Technical Report, Institute for Computer Applications in Science and Engineering, MS 132C, NASA Langley Research Center, Hampton, VA 23681-0001, 1995.

[14] Rossignac, Jarek R. and Paul Borrel. Multi-resolution 3D Approximations for Rendering Complex Scenes. Technical Report RC 17697 (#77951), IBM Research Division, T. J. Watson Research Center, Yorktown Heights, NY 10958, 1992.

[15] Schroeder, William J., Jonathan A. Zarge, and William E. Lorensen. Decimation of Triangle Meshes. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):65-70, July 1992.

[16] Turk, Greg. Re-Tiling Polygonal Surfaces. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):55-64, July 1992.

[17] Varshney, Amitabh. Hierarchical Geometric Approximations. PhD Thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1994.

[18] Weber, Jason and Joseph Penn. Creation and Rendering of Realistic Trees. *Computer Graphics (SIGGRAPH '95 Proceedings)*, 119-128, August 1995.