RC 20440 (90191) (5/20/97 Updated/Revised)
Computer Science/Mathematics 56 pages

Research Report

Surface Simplification Inside a Tolerance Volume

André Guéziec

IBM Research Division
T.J. Watson Research Center
Yorktown Heights, NY 10598

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich · Austin

Surface Simplification Inside a Tolerance Volume

André Guéziec
IBM T.J. Watson Research Center
P.O. Box 704, Yorktown Heights, NY 10598
gueziec@watson.ibm.com

May 20, 1997

ABSTRACT:

We present a technique for simplifying a triangulated surface. Simplifying consists of approximating the surface with another surface of lower triangle count. Our algorithm can preserve the volume of a solid to within machine accuracy; it favors the creation of near-equilateral triangles. We develop novel methods for reporting and representing a bound to the approximation error between a simplified surface and the original, and respecting a variable tolerance across the surface.

A different positive error value is reported at each vertex. By linearly blending the error values in between vertices, we define a volume of space, called the *error volume*, as the union of balls of linearly varying radii. The error volume is built dynamically as the simplification progresses, on top of preexisting error volumes that it contains. We also build a *tolerance volume* to forbid simplification errors exceeding a local tolerance. The information necessary to compute error values is local to the star of a vertex; accordingly, the complexity of the algorithm is either linear or sub-quadratic in the original number of surface edges, depending on the variant.

We extend the mechanisms of error and tolerance volumes to preserve during simplification scalar and vector data associated with surface vertices; we present results on surface data from various domains; our method also handles surface normals.

Key-words : Surface Simplification, Error Volume, Tolerance Volume.

1. Introduction

Polygonal surfaces form one of the major representations of three dimensional geometric models. They are used in a wide range of domains, including Computer Graphics, Medical Imaging, Scientific Visualization, Rapid Prototyping, industrial CAD applications, video games. Processes creating such surfaces include, but are not limited to: reconstructing surfaces from scattered point data (such as range scanner data), tiling two dimensional contour data, building iso-surfaces, generating geometric glyphs, generating terrain data, using satellite mapping or various other methods.

Surface simplification deals with the approximation of a polygonal surface with another polygonal surface containing fewer polygons. We believe that this is important for both efficiency and necessity reasons. Firstly, algorithms producing polygonal surfaces from various data sets sometimes produce surfaces that, independent of their size, could be very closely approximated with significantly fewer polygons; iso-surface building algorithms, for instance, are known for producing a collection of small polygons. Recent algorithms for converting the output of a laser range scanner to a polygonal surface tend to create one surface vertex per scanned data point, resulting in a multitude of small polygons as well. Secondly, some surface data sets, whether oversampled or not, are too large to be processed with the computers and software currently available, or to be transmitted using available networking technology. This can occur for applications such as the fast visualization of scientific or CAD data, or for the transmission of polygonal surfaces across the Internet. In our opinion, the interest in surface simplification algorithms will grow in the future, as the demand for greater resolution and detail in geometric models appears to increase significantly as additional power in computing or networking becomes available. Also, new sensors provide increasing levels of accuracy and detail in images and geometric models.

Using the property that any polygon has a triangulation (See O'Rourke [1]), we restrict our algorithm to the case of a surface that has been triangulated, meaning that all polygons have been partitioned into triangles. For our purposes, a polygonal surface that has been triangulated will be referred to as *surface*. We give some background on surfaces in Section 3. As our algorithm does not require the surface to be connected, for simplicity we will assume that a given surface can have one or several connected components. The input to a simplification process will be called *original surface* and the result, *simplified surface*.

Our first goal (Goal I) is to produce a simplified surface such that, roughly stated, the maximum deviation between both original and simplified surfaces is bounded. Precisely, the simplified surface must be such that, the maximum distance between any point of the original surface and the simplified surface as well as the maximum distance between any point of the simplified surface and the original surface are no more than a user specified tolerance; recall that the distance from a point to a surface is the distance to the closest point on the surface. Equivalently Goal I consists in bounding what is usually called the Haussdorf distance between the two surfaces. The symmetry in Goal I is important, as illustrated in Fig. 1. Our computation for such symmetric distances is local and, we believe, practical; It is explained in Section 8 where we develop the notion of *error volume* for satisfying Goal I.

Note that we explicitely write "point" as opposed to "vertex", meaning that points along surface edges and inside surface triangles must equally respect this tolerance. Fig.2 shows an example using

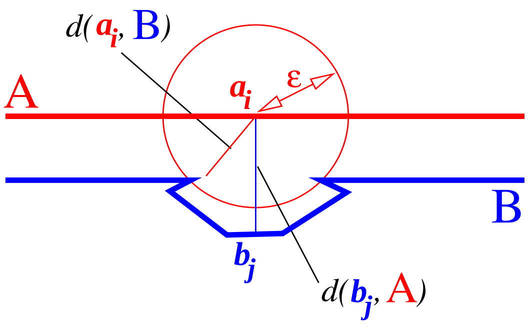

Figure 1: If the distance between the polygonal curves A and B were measured from A only, the maximum distance would not exceed ϵ, as for any point $\mathbf{a_i}$ of A, $d(\mathbf{a_i}, B) < \epsilon$. However, there exists $\mathbf{b_j}$ of B such that $d(\mathbf{b_j}, A) > \epsilon$.

a publicly available implementation of the algorithm of [2]; it is unclear how they handle points inside triangles. Fig.2 also indicates that a superficial visual inspection may not reveal that Goal I is not satisfied. Subsequently, the distance from a point of a surface (most often, of the simplified surface) to the other surface will be called *simplification error* or, when no ambiguity is possible, *error* at that point. The *maximum simplification error* can be measured on either the original or simplified surfaces; by default it is the maximum of both measurements.

For additional flexibility, we allow to specify different tolerances for different vertices. Between vertices, we define the tolerance as a linear combination of tolerances at vertices, thus defining a *tolerance volume* for the surface. By adjusting these tolerances, we can forbid self-intersections during simplification, and preserve contact properties, such as the tangency between two connected or disconnected surfaces. This flexibility adds a marginal cost to the algorithm, for significant benefits, as illustrated in Section 12.

Specifically, our second goal (Goal II) is to construct a simplified surface, such that at each point of the original surface, the distance to the closest point on the simplified surface will be less than the tolerance specified at that point. Also, we consider the union of balls centered at the original surface points with a radius equal to the tolerance at the point (the tolerance volume), and we require that it contain entirely the simplified surface. In particular, this will satisfy Goal I.

The present article develops a novel and practical solution for satisfying Goals I and II. We define the notions of error volume (for Goal I) and tolerance volume (for Goal II), which form the main contribution of this article. The error volume is visualized in Fig. 3. These notions were originally presented in our conference paper [3], that describes a simplification method using *edge collapses*, often called edge *contractions* in Graph Theory. The same mechanisms of error and tolerance volumes would apply equally to other methods, with some implementation differences. The present article provides the complete implementation details for [3], and also several important extensions. The first extension is concerned with the symmetry of Goals I and II, meaning that each point of the simplified surface must be within some distance bound to the original surface, as that part was missing from [3].

In Section 5.1 we describe the data structures used to compute efficiently the *star* (or neighborhood) of each vertex when requested, in the case where the surface has undergone simplification. The star

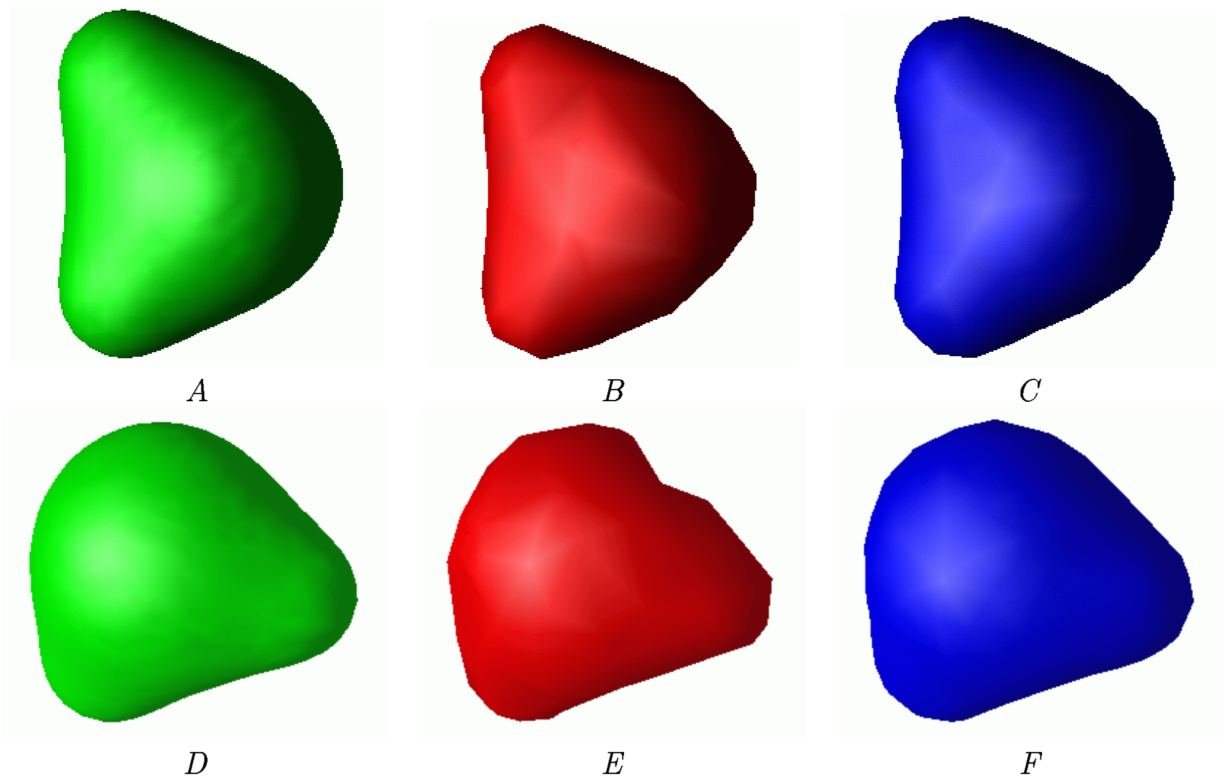

Figure 2: A: Iso-surface with 1,474 vertices. B: Surface simplification using a publicly available implementation of Schroeder's Triangle Decimation method; the 145 vertices are a subset of vertices in A. C: Surface simplification using our method, with 150 vertices. D, E, F: same surfaces viewed from a different angle; the same distance tolerance was used for B and C; it was not respected in B.

is defined in Section 3.

Another important idea in [3] is the preservation of volume for solids during simplification. To preserve the volume, we move the vertices from their original position. In Section 6, we describe a method for optimal placement of vertices, such that, if the original surface is the boundary of a solid, the final simplified surface, as well as every intermediate surface, will enclose the exact same volume, up to some floating point accuracy. We also describe validation tests. The volume preservation is an option of the algorithm. Another choice is to leave the vertices at their original location.

We then address the issue of creating triangles with a good aspect ratio. A practical measure for the aspect ratio and experiments are reported in Section 7.

The next extension, in Section 9, aims at preserving attributes specified at surface vertices, such as vertex red, green and blue colors, vertex normals, or scalar vertex data. We assume that color values and vertex data vary linearly on surface triangles. Error and tolerance volumes are defined in the same fashion as for positional error. For normals, a corrective term must be added to the error measured at the vertices to compensate for non-linearities.

Finally, the last extension is concerned with the simplification of surface boundaries, in Section 10.

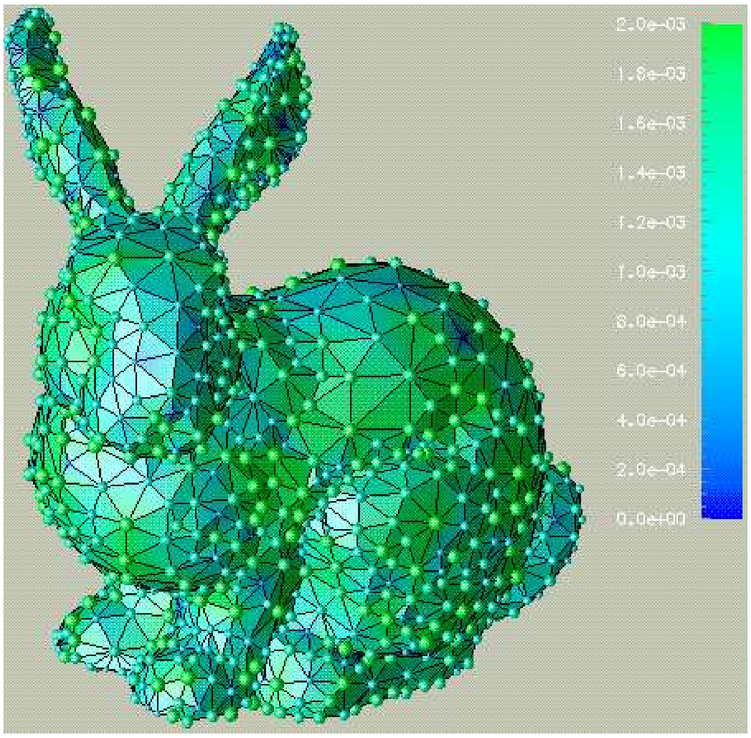

Figure 3: Error volume partially represented using color and spheres centered at surface vertices. The radius of each sphere equals the error value at the vertex, here bounded by .8 % of the bounding box diameter.

In the same spirit that guided us for volume preservation, we devised a method for maintaining boundaries of constant length. However, the volume preservation does not apply if the boundaries are modified.

In Section 12 we present results for surface models from Medical Imaging, Scientific Visualization as well as CAD models that are very different in nature, and represent a full spectrum of different sizes and characteristics. Some of these surfaces have colors and data values attached to the vertices.

2. Related Work

There are numerous references on surface simplification. We attempt to provide a faithful account of the previous work, focusing particularly on the early work as well as on the work most closely related to the present article.

Clark [4] introduces the notion of a scene tree whose nodes are objects available at various levels of details, associated with size information. Edges of the tree define transformations or relationships between more detailed or less detailed versions of objects. Clark does not discuss how the levels of detail could be automatically computed.

Schmitt and colleagues [5] and subsequently De Haemer and Zida [6] describe adaptive subdivision methods for spline surfaces [5] and polyhedra [6], with bounded approximation error.

Turk [7] uses point repulsion methods to position a new set of vertices on the surface. A mutual tessellation is formed using old and new vertices, and old vertices are progressively removed.

Schroeder and colleagues [2] sequentially remove vertices whose distance to an average plane or edge, determined using the positions of adjacent vertices, falls below a threshold. Then, they triangulate the gap left by omitting the vertex.

Rossignac and Borrel [8] decompose the bounding box of objects into parallelepipedic cells, and cluster vertices that are inside the same cell.

Hoppe and colleagues [9] develop an optimization algorithm that applies a series of three different mesh transformations until a minimum of energy is found.

Hinker and Hansen [10] merge quasi-coplanar faces and triangulate the perimeter of the resulting face.

Hamann [11] weights triangles using a local estimate of surface principal curvatures. Triangles are collapsed to a vertex in order of increasing weight.

The four following references are most similar to the present article. The simplification technique of Kalvin and Taylor [12] (first published in 1993) attempts to merge surface triangles to quasi-planar "Superfaces". Each original triangle is merged with one Superface, possibly limited to one single triangle. Each superface is guaranteed to lie within a given distance to the triangles that were merged into the Superface. However, because the algorithm does not give priorities to triangles before merging them, the algorithm does not produce economical simplifications for a given amount of processing time. In addition, the algorithm does not preserve the volume of a solid. Finally, the algorithm does not attempt to optimize the aspect ratios of triangles, as we do.

The simplification technique of Varshney [13] and subsequently of Cohen *et al.* [14] begins by constructing an envelope around the original surface, whose width is set to the maximum distance that is allowed between the simplified surface and the original surface. The envelope does not intersect itself. The surface vertices are then selected as candidates for removal and the hole left by the vertex removal is triangulated. The simplification algorithm ends when no more removal is possible. A subsequent process verifies that each new triangle is inside the envelope and that it does not intersect other surface triangles. This algorithm is computationally intensive and does not preserve the volume of a solid.

The simplification technique of Ronfard and Rossignac [15] (first published in 1994) places the edges of the surface in a priority queue, ordered by a key that measures the approximation error after the edge *collapses*, or contracts. Until the key exceeds a user specified value, the edge with the highest priority is collapsed. This algorithm is computationally intensive because each time an edge collapse is performed, the simplification error must be re-estimated for each edge that was adjacent to the collapsed edge. Moreover, the technique does not guarantee that the distance between a point of either the simplified or the original surface and the other surface will be less than a user-specified tolerance, but can only guarantee that a factor of this tolerance, depending on the surface geometry, will be respected. Also, the technique does not preserve the volume of a solid.

Guéziec [3] extends the edge collapse strategies of [15] to respect pre-specified distance tolerances and to preserve the volume of solids. The tolerance and error volumes that he defines can be used with vertex collapse [2, 16] or triangle collapse methods [11] as well. The present article is an archival description of [3] with full implementation details and extensions to attribute (or data) preservation at vertices.

In Computational Geometry, researchers have studied the problem of approximating a surface with an error bound, using the minimum number of triangles. They have exploited the analogy with the set-cover problem. The same goal is also pursued in [14]. Clarkson [17] studies convex polyhedra. Agarwal and Suri [18] attempt to prove that the minimal number of triangle approximation problem is NP-complete, when applied to the case of a height map. In terms of the number of triangles in the simplifications, our results compare with the results reported in [14].

DeRose *et al.* [19] and Eck *et al.* [20] develop a wavelet representation for surfaces. They first construct a base mesh. They subsequently subdivide the base mesh according to surface subdivision rules in order to approximate the original mesh as closely as possible. Schröder and Sweldens [21] and Gross *et al.* [22] have done related work. Certain and coworkers [23] extend the method for interactive visualization.

Among the work posterior to [3], Bajaj and Schikore [24] attach the tolerance to triangles instead of vertices, and address the issue of preserving multivariate data associated with the surface. They project a vertex star onto its average plane to compute errors: we believe that this cannot apply to all configurations of triangles. Klein *et al.* [25] attempt to obtain an error bounded simplification by managing a list of corresponding sets of triangles in the original and simplified surfaces and by computing the minimum of all possible triangle-triangle distances. We believe that our method is much more practical. Our method also handles a variable tolerance.

He and coworkers [26] scan-convert surfaces and smooth the resulting volume. Then, in the same spirit as our work with Hummel [27], they use efficient iso-surface building methods that gradually simplify the surface as it is being built.

Hoppe [28] presents a method for reversing the simplification process and accessing every intermediate simplification level.

Simplifications algorithm that are dependent of position or projection parameters have been proposed [29, 30, 28, 31]. We will not address this issue in the present paper.

In addition, there is a large body of litterature specialized in the simplification of terrains or height maps; For completeness, we cite [32, 33, 34, 35, 29]. Without modifications, our algorithm applies to terrains as well. Results are shown in Section 12.

To reduce transmission time and storage space, Deering [36] and Taubin and Rossignac [37] developed methods to encode the surface triangles and vertices in compressed form. These approaches do not reduce the number of triangles.

3. Surfaces

For our purposes, a *surface* $S(\{\mathbf{v_i}\}, \{t_j\})$ is defined with a set of vertices $\{\mathbf{v_i}\}$ and a set of triangles $\{t_j\}$. Each vertex has coordinates in $\mathbf{R^3}$. Each triangle is specified as a tuple of three vertex indices. The triangle is said to be *incident* to such vertices. There are two possible orderings of the three vertices modulo circular permutation, resulting in two orientations for a triangle. An *undirected surface edge*, or simply *edge* is defined as a pair of vertices $(\mathbf{v_1}, \mathbf{v_2})$ such that at least one triangle is incident to both of them. The triangle is said to be incident to the edge, and the edge incident to the vertices $\mathbf{v_1}$ and $\mathbf{v_2}$. $\mathbf{v_1}$ and $\mathbf{v_2}$ are also said to be *adjacent* vertices. Edges sharing a vertex and triangles sharing an edge are said to be adjacent edges and triangles. We assume that two triangles can only intersect at an edge or at a vertex of each of them.

Note that, with this last assumption, a surface is a particular case of a *simplicial complex*. A simplicial complex can be defined as follows: Firstly, a *n-simplex* is defined using an ordered tuple of $n+1$ linearly independent vertices $(\mathbf{v_0}, \ldots, \mathbf{v_n})$, as the set of all possible convex combinations of those vertices: a 0-simplex is a vertex, a 1-simplex a segment, a 2-simplex a triangle, a 3-simplex a tetrahedron. A *face* of a simplex is the simplex defined using a sub-tuple of the tuple of vertices defining the simplex. Secondly, a simplicial complex is a set of simplices, such that each simplex face is in the set and any two simplices intersect only at a face of each of them. Consult [38] for details. In our case, the simplices used are vertices, edges and triangles provided in the surface specification. If n_v is the number of vertices of the surface, n_e the number of edges and n_t the number of triangles, the Euler number is defined to be $n_v - n_e + n_t$.

We call the set of adjacent triangles that share a vertex $\mathbf{v}$ the *star* of $\mathbf{v}$: $\mathbf{v}^\star$. The number of triangles in $\mathbf{v}^\star$ is called the *valence* of the vertex $\mathbf{v}$, $v(\mathbf{v})$. To form the *link* of a vertex, $\ell(\mathbf{v})$ we first take the boundary of $\mathbf{v}^\star$, $\partial\mathbf{v}^\star$, and then remove from the boundary the edges incident to $\mathbf{v}$. The link is a polygonal curve made by linking up the remaining edges. See Figure 4. A *regular vertex* has a link formed of one simple polygonal curve; otherwise it is a *singular vertex*. If a regular vertex has a closed link, then it is *interior regular*, otherwise it is *boundary regular*. These cases are illustrated in Fig. 4.

A necessary and sufficient condition for a surface to be a *manifold* is that each vertex must be a regular vertex; otherwise it is a *non-manifold* surface. Recall that no two triangles can have a common intersection different from a common vertex or edge. By analogy with a smooth surface, the surface is a manifold if and only if the neighborhood of any vertex $\mathbf{v}$, defined to be $\mathbf{v}^\star$, can be continuously deformed to a disk or half-disk.

A manifold surface is *closed* if each edge is shared by exactly two triangles, or equivalently if each triangle has exactly three neighbors, each sharing a different edge of the triangle. Equivalently, a manifold surface is closed if it does not have any boundary vertices. In a manifold surface, a pair of adjacent triangles (t_1, t_2) share at most one edge, e. They have the same orientation if the two vertices of e listed in t_1 appear in opposite order in t_2. The surface is *orientable* if each triangle can be oriented such that any two adjacent triangles have the same orientation. We obtain an *oriented* surface by choosing one orientation among the possible two, for any one of the orientable surface triangles.

We call an edge incident to one single triangle *Type I boundary edge*. Its two vertices are then necessarily boundary vertices. The surface boundary is obtained by joining boundary vertices with Type I boundary edges, forming a series of closed loops; why the boundary must be closed is a result of Algebraic Topology; this is explained in [27] for instance. Edges incident to a single boundary vertex are called *Type II boundary edges*. Edges shared by two triangles, and incident to two boundary vertices are called *Type III boundary edges*. Other edges are called *interior edges*. Note that only Type I edges are included in the surface boundary.

For our algorithm, we assume that the surfaces are orientable and oriented manifolds, with or

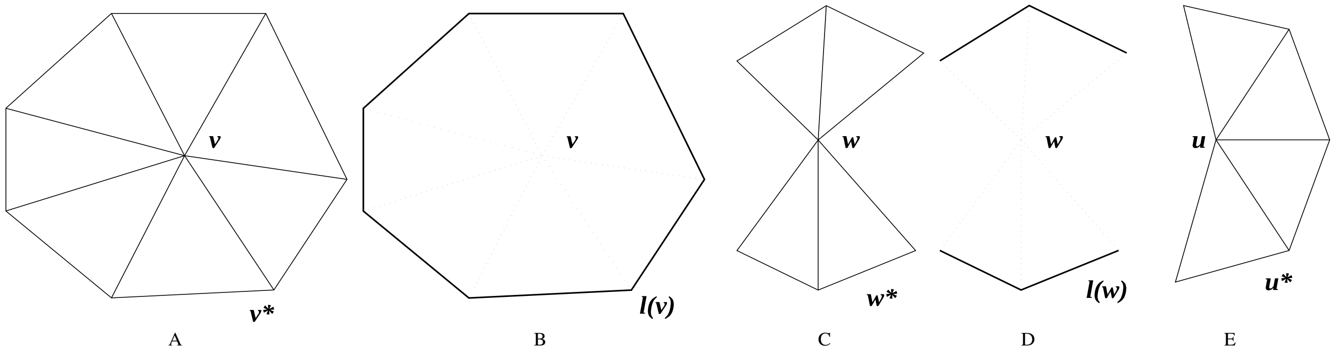

Figure 4: A: the star $\mathbf{v}^\star$ of a regular vertex $\mathbf{v}$ of valence seven. B: the link $\ell(\mathbf{v})$ of the regular vertex $\mathbf{v}$, composed of one simple closed polygonal curve. C: the star $\mathbf{w}^\star$ of a singular vertex $\mathbf{w}$ of valence four. D: the link $\ell(\mathbf{w})$ of $\mathbf{w}$, composed of two disconnected polygonal curves. E: the star $\mathbf{u}^\star$ of a boundary vertex of valence five.

without boundary. A closed and oriented manifold surface specifies the boundary of a collection of connected sets. Each connected set is a *solid*. In Section 6 we describe a method for preserving the volume of a solid during simplification.

3.1. Self-Intersections

Again, one assumption is that no two triangles can have a common intersection different from a common vertex or edge. Otherwise, the surface would not be a simplicial complex, much less a manifold. If the triangles belong to the same surface component, such an intersection is sometimes called a *self-intersection*. In practice, it is often expensive to verify whether such intersections occur, especially if this verification is performed more than once. One possible algorithm would verify for each pair of triangles whether they intersect in their interior; it would take quadratic time in the number of triangles to complete this task; using data structures encoding geometric proximity, such as octtrees, can reduce the complexity. However, certain algorithms generate surfaces that are guaranteed to be simplicial complexes and even manifolds: for instance some iso-surface builders.

In practice, our algorithm requires solely that the link of each vertex be a simple polygonal curve; this requirement appears clearly in Section 5.1; self-intersection tests are not required; they would lengthen computations. Our strategy is to use the tolerance volume to avoid creating additional self-intersections in the resulting surface; the user must provide tolerance values at vertices that are sufficiently constraining: half the shortest distance d between two "portions" of the surface (an offset surface of width $d/2$ would self-intersect) must be provided as tolerance for the points separated by the distance d, as discussed in Appendix C.

We do guarantee that every vertex in the simplified surface is a regular vertex. We believe that this is important, as a growing number of methods rely on this property to compute useful characteristics of surfaces, such as surface curvatures [39], or geodesic distances and critical points [40], and to smooth surfaces [41]. These methods represent a limited sample of the various techniques that have been proposed.

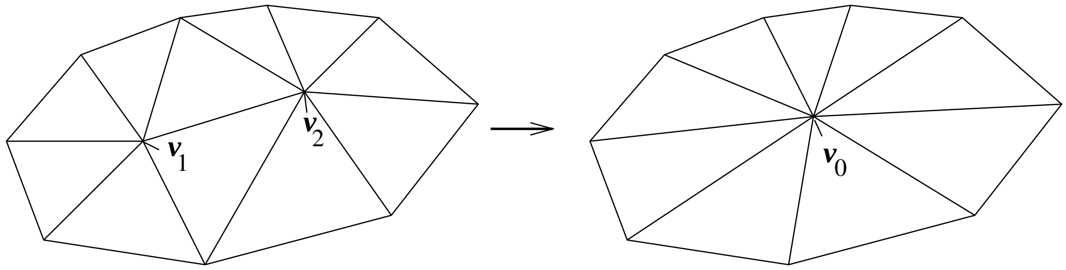

Figure 5: The edge collapse consists of replacing the edge $(\mathbf{v_1}, \mathbf{v_2})$ with the simplified vertex $\mathbf{v_0}$. As a result, the entire edge star $\star(\mathbf{v_1}, \mathbf{v_2})$ (Left) is modified to a vertex star $\star(\mathbf{v_0})$ (Right).

4. Description of the Algorithm

The original motivation of this work was the simplification of surfaces used in Scientific Visualization and Medical Imaging. The characteristics of these surfaces are that they are generally large –hundreds of thousands of triangles, and they are generally manifolds. Often, end-users may wish to take measurements on them, such as volumes, areas, distances, or may wish to register surfaces for comparisons. Another aspect of these surfaces is that they often are oversampled; this is particularly the case of the surfaces constructed as iso-surfaces in volumetric data. Our implementation choices might reflect these characteristics. However, our methods are largely domain independent, as illustrated in Section 12.

The algorithm builds upon the method of Ronfard and Rossignac [15], with a series of modifications for preserving volume and respecting distance tolerances. Ronfard and Rossignac, as well as Hoppe [28] and others, simplify a surface by applying a series of *edge collapses*. The principle of an edge collapse is to modify a surface edge to *collapse*, or contract to a given vertex, and to modify the surrounding edges and triangles to maintain the surface connectivity. A triangle collapse works similarly. Figure 5 shows the configuration for an edge collapse, an *edge star*, where $(\mathbf{v_1}, \mathbf{v_2})$ denotes the edge. Similarly to the vertex star, the star of $(\mathbf{v_1}, \mathbf{v_2})$, $(\mathbf{v_1}, \mathbf{v_2})^\star$, is the set of all triangles that meet either at $\mathbf{v_1}$ or $\mathbf{v_2}$. It is also the union of the stars of $\mathbf{v_1}$ and $\mathbf{v_2}$. If the edge is collapsed, then the vertices $\mathbf{v_1}$ and $\mathbf{v_2}$ are modified to terminate at $\mathbf{v_0}$, which we call a *simplified vertex*. If the edge is interior, the resulting surface has three fewer edges, two fewer triangles and one fewer vertex, which we call the *deleted* edges, triangles and vertex.

The method of [15] follows a greedy strategy. For each collapse, a score measuring the resulting approximation error is assigned to the edge. Edges are entered in a priority queue keyed with the score (priority queues are reviewed in [42]). The edge with the lowest score is simplified. New scores must be computed for the neighboring edges affected by the collapse, which are assigned new positions in the queue. The algorithm stops when the scores exceed a threshold or when sufficiently many edges have been collapsed. As mentioned in Section 2 other methods using triangle collapses or vertex removals could have suited our purpose. However, when collapsing an edge, the position of the *simplified vertex* must be specified. We specify a position such that a volume defined with the edge star will be preserved after simplification. This is described in Section 6.

Contrary to [15], we do not rely on a perfect ordering of the edges. Instead, we weight the edges using an inexpensive measure; we compute the length of the edge, and add to the length the sum of the error values corresponding to the edge endpoints; the exact significance of the error values is explained in Section 8. Accordingly, we develop a series of tests to determine whether the edge under current consideration can be safely collapsed. Our justification for using this simple weighting procedure is that in general, there are many more weights computed than edges tested. Each time an interior edge is collapsed, two neighboring edges are also collapsed and new weights must be computed for all the remaining neighboring edges. We may have to recompute the weight of a given edge several times before it becomes a candidate for simplification. We may also compute several times the weight of a given edge that will be collapsed when testing a neighbor.

For instance, when simplifying the Buddha model illustrated in Fig.24 we processed 500,383 surface edges, we tested 246,009 edges, collapsed 426,720 edges, rejected 103,769 edges and computed 1,378,771 weights; there were 5.6 times more weights computed than tests performed. The multiplicative factor is dependent upon the average number of edges adjacent to a given edge (which depends upon the Euler number), as well as upon the ratio of the number of edges rejected to the number of edges tested. Since in our algorithm, the weights are used solely for sorting purposes, we use a simple weight computation. However, the weighting process is independent of the rest of the algorithm; it can be modified for different applications.

We have implemented a variant of the algorithm that uses an accumulator rather than a priority queue. In the case of the accumulator, several edges with similar weights are contained in a given accumulator cell. The weights are positive real values; we map them to the interval $[0, 1]$ by applying the map $x \longmapsto 1/(1 + x^2)$, and we divide the interval in a number of regularly sized cells, whose number is a parameter of the algorithm. This version of the algorithm was marginally faster in some cases; its theoretical complexity is linear in the number of edges, as discussed in Section 11. Choosing an accumulator or a priority queue does not influence the rest of the algorithm.

The algorithm is summarized in Figure 6. In the sequel, the word “queue” stands for priority queue or accumulator, depending on the particular implementation of the method. In the case of the accumulator, the first edge is the edge appearing first in the first cell; it does not necessarily have the lowest weight. In the case of the priority queue, it is the edge with the lowest weight.

Until the queue becomes empty, the first edge is removed from the queue, and undergoes a series of tests that determine whether it can be safely collapsed. If the collapse is performed, adjacent edges are changed; generally, they will be longer. We retrieve the adjacent edges from the queue, compute their new weight, and reenter them in the queue. During this process, some edges that were previously tested and rejected may be reinstated in the queue. Consequently, the length of the queue, or the size of the accumulator, does not decrease monotonically.

We now describe the tests that enable to decide whether an edge is collapsible. Although they could be performed in any order, we attempt to perform the simpler tests first, so that the more expensive tests can be avoided if early tests invalidate them.

The first test consists of verifying that the valence of the simplified vertex is higher than the minimum of three for an interior vertex and of one for a boundary vertex. The valence must also be lower than a specified maximum value.

In agreement with the definition of a manifold surface, the second test verifies that the simplified vertex is regular. Since both $\mathbf{v_1}$ and $\mathbf{v_2}$ are regular, we simply determine whether there are duplicate vertices in the link of $(\mathbf{v_1}, \mathbf{v_2})$, $\ell(\mathbf{v_1}, \mathbf{v_2})$. Also, we forbid the collapse of Type III boundary edges; the resulting vertex would have a link with two disconnected components. The second test deals solely with surface connectivity information; the issue of surface intersections occurring with the new surface location after collapsing is not addressed here. However, if the original tolerances at vertices have been chosen suitably, such intersections can be detected when performing the forthcoming fifth test.

If the second test is passed, we then determine the position of a potential simplified vertex as described in Section 6. Once the potential simplified vertex has been positioned, the next three tests verify that the new triangulation is geometrically consistent with the current triangulation.

In the third test, we make sure that the orientations of the new triangles are consistent with the current orientations. Each new triangle is assigned a correspondent in the previous configuration, namely the triangle that shares the same link edge. We measure the scalar product of the triangle normal before and after the collapse, and record the minimum such scalar product. It must exceed the cosine of a maximum deviation angle between $[0, \pi/2]$ specified as a parameter of the algorithm; recall that the cosine is a strictly decreasing function of the angle between $]0, \pi[$, which is the span of angles that can be measured between 3-vectors. In addition to this third test, triangle normals are used for deriving the equation of planes supporting surface triangles (See Section 6). Consequently, we use an array for storing the normals of all triangles. When testing an edge, we compute new normals for the triangles in the edge star; we store them temporarily, and overwrite the previous normals if the collapse is performed.

Section 7 details the fourth test that limits the creation of thin triangles, especially if they are not necessary. We develop a measure of triangle *compactness*, and perform the collapse only if the minimum compactness in the edge star is not degraded in excess of a pre-specified factor after the edge star is contracted into a vertex star. We assume that if in a given region, thin triangles are necessary to model the geometry, the minimum compactness must already be low in edge stars of that region.

In the fifth test, we verify that the edge star and simplified vertex star are sufficiently close. This test is detailed in the Section 8. New error and tolerance values are computed for the simplified vertex star, and temporarily stored. The simplification cannot occur if the any of the new errors is larger than the minimum of the tolerance values in the edge star.

If attributes are available at the vertices, such as vertex red, green and blue colors, vertex normals or other real or vector data, additional error and tolerance values are computed for these attributes. Similarly, the errors cannot exceed the minimum of the tolerance values in the edge star. This computation is explained in Section 9.

If the fifth test is successfully passed, the edge is collapsed, and the new error and tolerance values are used.

5. Data Structures

We use an array of vertex coordinates, and an array of triangles specified as three indices to the vertex array. In addition, various arrays can accommodate floating point data or vectors associated

Until Queue/ Last Bucket Empty:

- Take edge with low(est) weight
- If edge can be safely collapsed
  1. If valence does not exceed maximum
  2. If simplified vertex is regular
  3. If triangle normal rotation is acceptable
  4. If triangle compactness is acceptable
  5. If error does not exceed tolerance

  □Change neighboring configuration

  □Remove all edges of the star from the queue

  □Reinstate new edges in the queue
- Else
  remove edge from queue

Figure 6: Simplification algorithm.

to the vertices, if such vertex attributes are available. The valence at vertices is stored in an array of integer values. Although the valence can be computed on demand, as shown in the Section 5.1, the implementation is simpler, and we believe, safer, if the valences are pre-stored and known at all times; our first test also requires the valence. The triangle normals are stored in an array of vectors, as they are frequently used by the algorithm. The triangle areas are stored in an array.

An array of edges is inferred from the array of triangles. The edge record is composed of four fields (See Fig.7A): the indices of the two vertices to which the edge is incident, with the lowest of the two indices coming first, and the indices of up to two triangles incident to the edge; the ordering of the triangles corresponds to their order of appearance when rotating counter-clockwise about the first vertex.

We use a data structure permitting constant, or almost constant access to edges given the indices of one or two endpoints. The solution we employ is a hash table keyed with either a code computed from two indices, in case we index with respect to the two endpoints simultaneously, or a code computed from one index, in case we index with respect to the two endpoints independently. If both types of indexing are required, we need two hash tables. The array of edges and associated hash table are constructed by traversing the array of triangles and filling in the triangle entries of the edge data structure as the triangles are encountered.

These structures, however, are static; as edges are collapsed, deleted edges, triangles and vertices stop to be referenced. Rather than eliminating those edges, triangles and vertices from the data structure, we maintain at each stage of the process arrays of vertex, triangle and edge *parents*, respectively for each vertex, triangle and edge of the surface. Initially, each element is its own parent: the parent array is a copy of the indices of the elements. When an edge is collapsed, the remaining vertex becomes the parent of the other vertex. In Fig. 5 $\mathbf{v_1}$ is chosen to be the remaining vertex; its coordinates are exchanged with the coordinates of $\mathbf{v_0}$. The parent of $\mathbf{v_2}$ is defined to be the index of

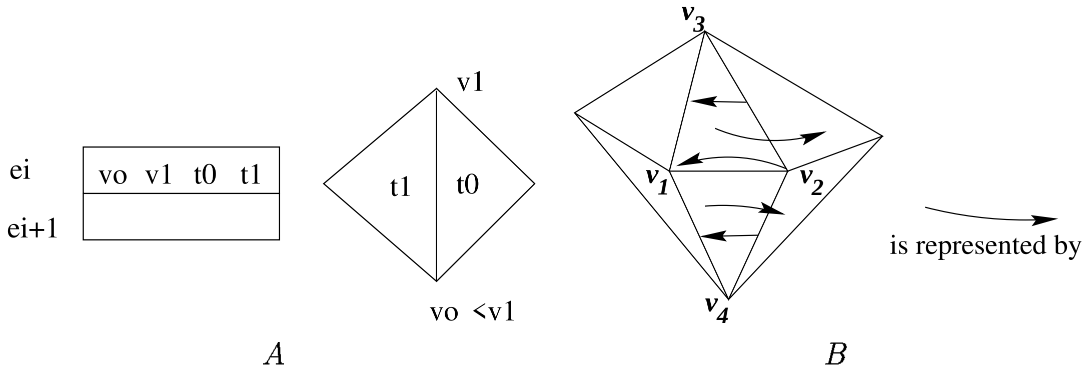

Figure 7: A: an edge refers to four indices. B: defining parents of surface elements during an edge collapse.

$\mathbf{v_1}$.

The three deleted edges and two deleted triangles are treated as shown in Fig. 7B. We note $\mathbf{v_3}$ and $\mathbf{v_4}$ the vertices adjacent to both $\mathbf{v_1}$ and $\mathbf{v_2}$. The edge $(\mathbf{v_1}, \mathbf{v_2})$ is now degenerate; it will not be referenced subsequently. The edge $(\mathbf{v_2}, \mathbf{v_3})$ is now identified with the edge $(\mathbf{v_1}, \mathbf{v_3})$ since $\mathbf{v_2}$ is identified with $\mathbf{v_1}$. Accordingly, we set the parent of $(\mathbf{v_2}, \mathbf{v_3})$ to be $(\mathbf{v_1}, \mathbf{v_3})$. Similarly, we set the parent of $(\mathbf{v_2}, \mathbf{v_4})$ to be $(\mathbf{v_1}, \mathbf{v_4})$. When identifying edges $(\mathbf{v_2}, \mathbf{v_3})$ and $(\mathbf{v_1}, \mathbf{v_3})$, we merge the triangle records associated with both edges; triangle records are illustrated in the Figure 7A; since the triangle $\Delta(\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3})$ is degenerate, we identify it with the non-degenerate triangle incident to $(\mathbf{v_2}, \mathbf{v_3})$, as shown in Fig. 7B. The parent of the triangle $\Delta(\mathbf{v_1}, \mathbf{v_4}, \mathbf{v_2})$ is chosen similarly.

The parent relationship induces a forest data structure for the surface elements. Each element is represented by the root of the tree to which it belongs. For instance, given an edge e, to determine its representative after a series of edge collapses in the surface, we retrieve its parent $p(e)$, and then its *grand* parent $p(p(e))$ and the next parent, continuing until the root r of the tree is encountered; the root is such that $p(r) = r$: the root is its own parent.

To speed up the algorithm, we store only the path compressed version of the parent hierarchy such that each element points directly to its associated root (See for instance [42]). The path compression occurs the first time the element is referenced after its root changed; this has a fixed cost: we simply change the parent to point to the new root, as all the intermediate parents have been bypassed earlier with path compression. Note that after path compression, every parent is a root. Surface elements (vertices, triangles, edges) that are carried to the next simplification level can be easily identified by their property of being their own parent; equivalently, they can be identified as being a root of a tree. We call them the *parent* elements.

In conclusion, we evolve from a static data structure, where the adjacency information is pre-stored in the edge record (Fig. 7A), and will not be modified, to a dynamic data structure, using three arrays of parents, one for the vertices, one for the edges and one for the triangles. The connectivity of the original surface is still available. The connectivity of the simplified surface can be obtained by substituting each element with its parent, as discussed in the next section.

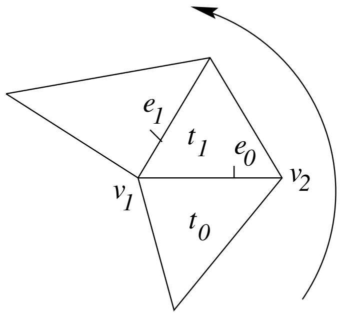

Figure 8: Building a vertex star by rotating counter-clockwise around an interior vertex.

5.1. Computing the Star of a Vertex

Exploiting the fact that in an oriented manifold surface, every vertex is regular, we develop a method for computing the star of every vertex, in a surface that has been simplified. Although after a series of simplifications, some vertices, edges and triangles are not referenced any more, we can still use the original arrays of edges, triangles and vertices, and especially the original connectivity information recorded in the edge data structure.

The star of a vertex is constructed when it is used, rather than stored all along the process. The method we use for computing the star of a simplified surface is novel. It does require a manifold surface. The approach that we advocate for simplifying non-manifolds is to operate the transformations on them that will produce a series of manifolds. Such a method is outlined in Hoffmann [38].

We first explain the method before simplification occurs, and then extend it to a simplified surface. Since every vertex is regular, in particular, a interior vertex is such that its link is homeomorphic to a circle. To compute the star of the vertex $\mathbf{v_1}$ in Fig. 8, we can consider a first triangle t_0 in the star, find two edges of t_0 incident to $\mathbf{v_1}$, and choose Edge e_0 that is visited second in counter-clockwise order from $\mathbf{v_1}$. Using the triangle record of Edge e_0, we determine the next triangle t_1 and then the next edge e_1. We loop until the first triangle t_0 is encountered.

The method is very similar for a boundary vertex. The difference between the previous method is that we first rotate clockwise until a boundary edge is encountered, and then rotate counter-clockwise until the second boundary edge is encountered. In each case we start with an edge, and we wish to compute its star; first triangles are taken from the edge triangle record.

If the surface has been simplified, we start with a parent edge $(\mathbf{v_1}, \mathbf{v_2})$. We replace the vertices by their parents $p(\mathbf{v_1})$ and $p(\mathbf{v_2})$. We consider a *first* triangle t_0, and replace it with its parent $p(t_0)$. Then we find the second edge e_0 of $p(t_0)$ incident to $p(\mathbf{v_1})$. To examine an incidence relation of an edge e to a vertex $\mathbf{v}$, we take the parents of the edge and vertex. We retrieve the original vertices to which the parent edge is incident to, from the edge vertices record, and take the parents of such vertices. We now compare these parent vertices to the parent vertex tested for incidence. If either parent vertex is the same, we decide that e is incident to $\mathbf{v}$. Hence we can determine Edge e_0. We proceed similarly as before making sure that the parents are used in place of the elements themselves. The justification of this procedure can be understood by examining closely Fig. 7B. Using the parents we can rotate around vertices $\mathbf{v_1}$, $\mathbf{v_3}$ and $\mathbf{v_4}$ without interruption after the edge collapse. The parents are used to

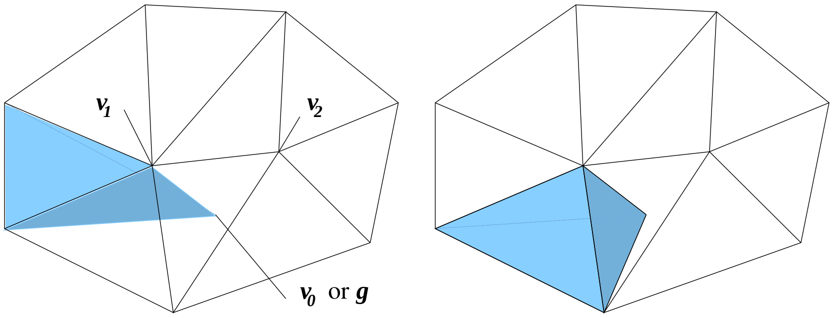

Figure 9: We compute a volume associated with the edge star by adding the volumes of individual tetrahedra.

close the gaps in the vertex stars left by the edge collapse.

6. Positioning the Simplified Vertex for Volume Preservation

In this section, we develop a method for positioning the simplified vertices. We limit the discussion to interior vertices. Boundary vertices will be discussed in Section 10.

Our criteria are to position the vertex as closely as possible to the planes of the edge star, and to ensure that the edge collapse results in no volume change. Specifically, a closed and oriented manifold surface encloses a volume that is obtained by adding the volumes of all tetrahedra spanned by any fixed origin and the oriented triangles of the surface. In the case of a manifold surface with boundary, we define a solid volume by arbitrarily closing the surface. To preserve this volume, one requirement will be to not alter the boundary. For instance, consider an iso-surface that is being built by dividing the volume data into sub-volumes, and by executing in parallel an iso-surface builder in each sub-volume. This scheme will construct a closed iso-surface as a collection of open surfaces sharing common boundaries. Our algorithm can simplify each open surface separately; by preserving each boundary, we guarantee that the volume of the composite simplified surface will exactly match the volume of the original iso-surface.

We now consider the edge $(\mathbf{v_1}, \mathbf{v_2})$; we associate a volume $V_{1,2}$ with the star $(\mathbf{v_1}, \mathbf{v_2})^\star$ of this edge as shown in Fig. 9 and as detailed subsequently; we associate a volume V_0 with the star $\mathbf{v_0}^\star$ of the potential simplified vertex $\mathbf{v_0}$; we determine a location for the potential simplified vertex $\mathbf{v_0}$ such that $V_{1,2} = V_0$; we then perform the geometric consistency tests as was described in Section 4.

We choose the centroid of the edge star $\mathbf{g}$ as a local origin. We translate the coordinates of all the vertices in $(\mathbf{v_1}, \mathbf{v_2})^\star$ by $-\mathbf{g}$. We define the volume of an edge or vertex star as the sum of $1/6$ of all 3-D determinants of vertices of tetrahedra spanned by the origin $\mathbf{g}$ and the oriented triangles belonging to the star:

$$V_{1,2} = \frac{1}{6} \sum_{t_i = \Delta(i_0, i_1, i_2) \in (\mathbf{v_1}, \mathbf{v_2})^\star} \begin{vmatrix} x_{i_0} & y_{i_0} & z_{i_0} & 1 \\ x_{i_1} & y_{i_1} & z_{i_1} & 1 \\ x_{i_2} & y_{i_2} & z_{i_2} & 1 \\ 0 & 0 & 0 & 1 \end{vmatrix} = \frac{1}{6} \sum_{(i_0, i_1, i_2)} \begin{vmatrix} x_{i_0} & y_{i_0} & z_{i_0} \\ x_{i_1} & y_{i_1} & z_{i_1} \\ x_{i_2} & y_{i_2} & z_{i_2} \end{vmatrix} \tag{6.1}$$

We note (x, y, z) the unknown coordinates of the simplified vertex $\mathbf{v_0}$. After collapsing the edge, the volume of the star of the simplified vertex V_0 is a sum of $1/6$ of determinants of tetrahedra spanned by $\mathbf{v_0}$, the origin $\mathbf{g}$ and vertices of the link $\ell(\mathbf{v_1}, \mathbf{v_2})$:

$$V_0 = \frac{1}{6} \sum_{e_j=(j_0,j_1)\in\ell(\mathbf{v_1},\mathbf{v_2})} \begin{vmatrix} x_{j_0} & y_{j_0} & z_{j_0} \\ x_{j_1} & y_{j_1} & z_{j_1} \\ x & y & z \end{vmatrix} \tag{6.2}$$

By equating $V_{1,2}$ and V_0, we obtain a linear equation in (x, y, z), namely:

$$6V_{1,2} = \sum_j (y_{j_0} z_{j_1} - z_{j_0} y_{j_1})x + \sum_j (z_{j_0} x_{j_1} - x_{j_0} z_{j_1})y + \sum_j (x_{j_0} y_{j_1} - y_{j_0} x_{j_1})z$$

which we rewrite as:

$$\alpha x + \beta y + \gamma z + \delta = 0 \tag{6.3}$$

This defines a plane P on which $\mathbf{v_0}$ must lie to preserve the volume of the star. An alternative solution for deriving Eqn. 6.3 is to replace $\mathbf{g}$ by $\mathbf{v_0}$ in the computation of $V_{1,2}$ (Eqn. 6.1) which must then be equal to zero. In that case $\mathbf{g}$ is not necessary, and we perform no translation.

For numerical reasons, using the determinants of Eqn. 6.1 is not the safest method for computing the volume of individual tetrahedra. We develop an alternative solution in Appendix A.

At this point we choose to change our coordinate system such that the vertical direction will be the same as the normal of P, $\mathbf{n}$. This only requires a Householder transformation S (symmetry) that will map either $(0, 0, 1)$ or $(0, 0, -1)$ to $\mathbf{n}$. In this new coordinate system, the requirement that $\mathbf{v_0}$ lie on P is simply expressed by the equation $z + \delta = 0$.

Some additional information is necessary in order to position $\mathbf{v_0}$. We choose to position the simplified vertex as closely as possible to the planes defined by the edge star. Specifically, we minimize the sum of squared distances between $\mathbf{v_0}$ and the planes defined by the triangles of $\star(\mathbf{v_1}, \mathbf{v_2})$. If $t_i = (i_0, i_1, i_2)$ is a triangle in the star $(\mathbf{v_1}, \mathbf{v_2})^\star$, we can write the equation of the plane P_i containing t_i using a determinant computation as before:

$$\begin{vmatrix} x_{i_0} & y_{i_0} & z_{i_0} & 1 \\ x_{i_1} & y_{i_1} & z_{i_1} & 1 \\ x_{i_2} & y_{i_2} & z_{i_2} & 1 \\ x & y & z & 1 \end{vmatrix} = 0$$

However, recall that triangle normals are used elsewhere in the simplification algorithm, for determining the variation in triangle orientations, and forbidding excessive variations. Instead of using a determinant, we use the normal a_i, b_i, c_i of the ith triangle in the star to quickly derive a normalized equation of its supporting plane P_i: $a_i x + b_i y + c_i z + d_i = 0$, with $(a_i^2 + b_i^2 + c_i^2 = 1)$. d_i can be derived by substituting any triangle vertex in the equation. For more stable results, we choose to substitute all three vertices and take the average value. The signed distance from a point (x, y, z) in space to the plane P_i is precisely $a_i x + b_i y + c_i z + d_i$.

model	Femur	Buddha	Deino
original # of triangles	180,854	333,586	44,954
simplified # of triangles	3,124	49,106	19,490
original volume	233,462.7455 mm^3	23,048,568.98 $pixel^3$	230,276.599 mm^3
vol. after simplification	233,462.7452 mm^3	23,048,569.03 $pixel^3$	230,276.600 mm^3

Table 1: Volumes before and after simplification for surface models of solids, or solids with a preserved boundary

Then we minimize the sum of squared distances to the planes P_i $\sum_i (a_i x + b_i y + c_i z + d_i)^2$ over (x, y, z), subject to the constraint $z + \delta = 0$. This preserves the volume exactly, to the extent of the numerical precision in deriving Eqn. 6.3.

We solve the following set of equations (6.4), using Normal Equations, which will give us the optimal intersection for a set of lines in two dimensions, using weighted least squares. Using a QR decomposition [43] is more stable but also slower. The solution of this system provides us with coordinates for $\mathbf{v_0}$ that we back-transform to the original coordinate system by applying successively the symmetry S and the translation of vector $\mathbf{g}$.

$$\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \\ a_2 & b_2 \\ \dots & \dots \\ a_n & b_n \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} c_0\delta - d_0 \\ c_1\delta - d_1 \\ c_2\delta - d_2 \\ \dots \\ c_n\delta - d_n \end{bmatrix} \tag{6.4}$$

Experiments have shown that the volume enclosed by the surfaces that the algorithm has processed has been preserved as accurately as the machine precision after simplification. Using the example of the Femur, illustrated in Fig.26, we measured a volume of 233,462.7455 mm^3 before simplification and of 233,462.7452 mm^3 after simplification: nine digits are correct. Other volumes are reported in Table 1. The Deino model, which is a surface representing the skull of a fossil hedgehog, is not shown. We identified two difficulties associated with the computation of such volumes, that we discuss in the Appendix A.

The volume preserving scheme for positioning $\mathbf{v_0}$ is an option of the algorithm. The default behavior is to identify $\mathbf{v_0}$ with $\mathbf{v_1}$ which implements a particular case of the vertex removal method [2]. Yet another solution is to minimize the sum of squared distances to the planes P_i supporting the triangles of the edge star, and to drop the constraint that $\mathbf{v_0}$ must lie on the plane P.

7. Fairness of the Triangulation

Edges collapse only if the operation does not degrade the highest *compactness* value for the triangles of $\star(\mathbf{v_1}, \mathbf{v_2})$ by more than a user-specified threshold. By analogy with the compactness of a closed curve, we define the compactness c of a triangle with the following formula:

$$c = \frac{4\sqrt{3}a}{l_0^2 + l_1^2 + l_2^2} \tag{7.1}$$

where a is the positive area of the triangle and (l_0, l_1, l_2) are the lengths of the three sides. This formula is related to the area to perimeter ratio, with the difference that taking the square of the perimeter is necessary to obtain a dimensionless measure; accordingly, we would take three square roots to compute l_0, l_1 and l_2; we would then take the square of the sum. Since the algorithm requires to compute such values often, we avoid square roots using Formula 7.1.

Formula 7.1 weights the area in such a way that a flat triangle will have a compactness value of zero and an equilateral triangle will have a compactness value of one. The actual criterion we use for controlling the compactness of the triangles during the simplification consists of comparing the minimum compactness $c_{1,2}$ of the triangles in $(\mathbf{v_1}, \mathbf{v_2})^\star$ with the minimum compactness c_0 observed in $\mathbf{v_0}^\star$, the new configuration. We define a minimum ratio $r_{\min}$ such that we will always have $r_{\min} c_{1,2} < c_0$. Thus, the minimum compactness cannot be degraded by more than a $r_{\min}$ factor. In order to favor the simplification process, $r_{\min}$ is generally chosen to be less than one. After successive simplifications, the compactness of certain triangles could tend towards zero. However, these flat triangles become in turn ideal candidates for simplification and we observe in practice that the compactness is improved for most triangles.

In Figure 10, we compare histograms of compactness values before and after simplification using the Femur example. We observe that after simplification 39% of the triangles have a compactness value larger than 0.9. In Fig. 11 we show the triangulation produced by our algorithm on the data set of the femur that was used for the measurements.

8. Error and Tolerance Volumes

We first explain the notions of error and tolerance volumes, omitting the details. Then, in Section 8.1 we give precise definitions of error and tolerance volumes, that apply indifferently to curves or surfaces. We give methods for the computation of error and tolerance values in Section 8.2.

Several attempts have been made to treat the simplification of surfaces in a fashion similar to the method of Schwartz and Sharir [44] and related methods for curves [45], where the simplified model is constrained to stay inside some *tubular neighborhood*, or volume, defined around the original model. The notion of tubular neighborhood defined in Differential Geometry [46] is very similar to the notion of *offset surface* in Solid Modeling [38], which consists of translating every surface point, by a predetermined amount, on both sides of the surface along the surface normal. Until recently [13], the methods of [44] or [45] were known to generalize poorly from curves to surfaces. The main reasons, we believe, are that the search space for the optimal model is much larger for surfaces than for curves, and the process of verifying that a surface is contained inside the tubular neighborhood is global.

We believe that it is more efficient to define the tubular neighborhood around the simplified model. As the simplification undergoes, we gradually construct an intermediate bounding volume, the *error volume*, based upon the simplified surface. The width of the error volume at vertices, or *error value* at vertices, is updated after each simplification operation, as explained subsequently. In the beginning, the error values are zero, unless inherited from a previous simplification, or resulting of uncertainty measurements. The error volume is an union of balls; as illustrated in Fig. 12 the final error volume contains all intermediate error volumes; one additional requirement is that each ball intersect the original surface. The ball radii, or error values, provide a bound to the simplification error measured

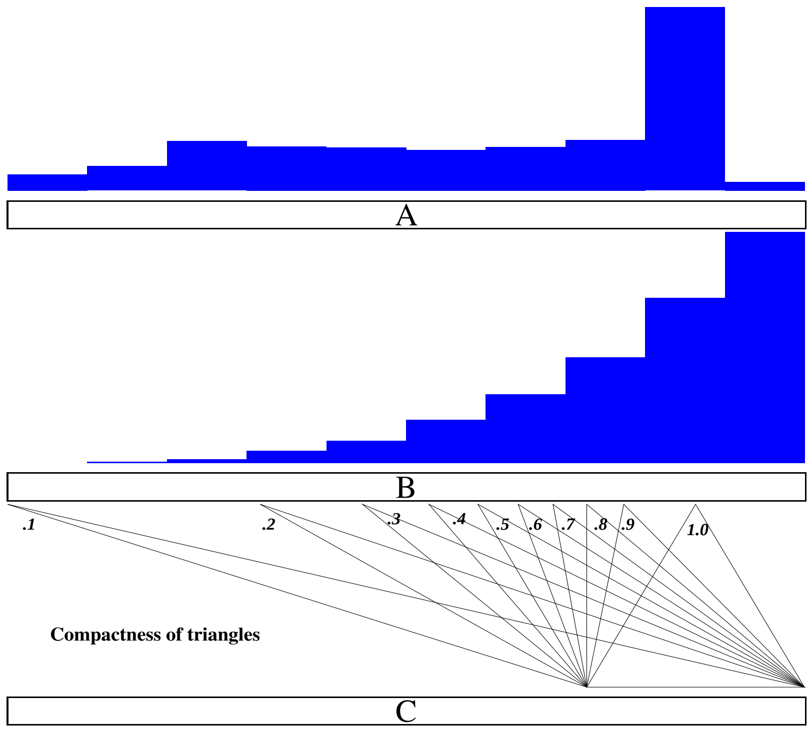

Figure 10: A: histogram of compactness values before simplification for the Femur example. B: histogram of compactness values after simplification. C: example of triangles of compactness values of .1 to 1 in .1 increments, corresponding to boundaries between the histogram bins. A flat triangle has a compactness of zero.

on the simplified surface. To satisfy Goal I, we perform edge collapses and grow the error volume until its width reaches a maximum tolerance, or until further simplifications are forbidden for failure to satisfy topological or geometrical criteria as previously explained in Section 4.

Goal II however, cannot be satisfied with the error volume only. Accordingly, as the simplification undergoes, we maintain in parallel a *tolerance volume*, that defines a neighborhood around the original surface that must contain the simplified surface, as well as each intermediate tolerance volume. The tolerance volume is also a union of balls; each ball intersects the simplified surface. In the beginning, tolerance values are specified at the surface vertices by the user. Various methods can be used for defining tolerance values that vary across the surface. One straightforward method is to define a function of $\mathbf{R}^3 \longrightarrow \mathbf{R}$ and to sample this function at surface vertices. For the Buddha model (Fig.24), we have used distances to spheres. For iso-surfaces extracted from CAT-scan, the tolerance could be dependent on the slice number and uniform for a given slice.

After each edge collapse, we could update tolerance values at the new vertices such that the newly defined tolerance volume would be contained inside the original tolerance volume, as shown in Fig. 12. We would then forbid a simplification resulting in negative tolerances. Since the volumes are computed conservatively, we could even simplify further until both the tolerances are negative and the errors exceed a maximum value. Section 8.2 describes methods suitable for such computations. However, we have implemented a simpler solution; we abandon the requirement that a tolerance

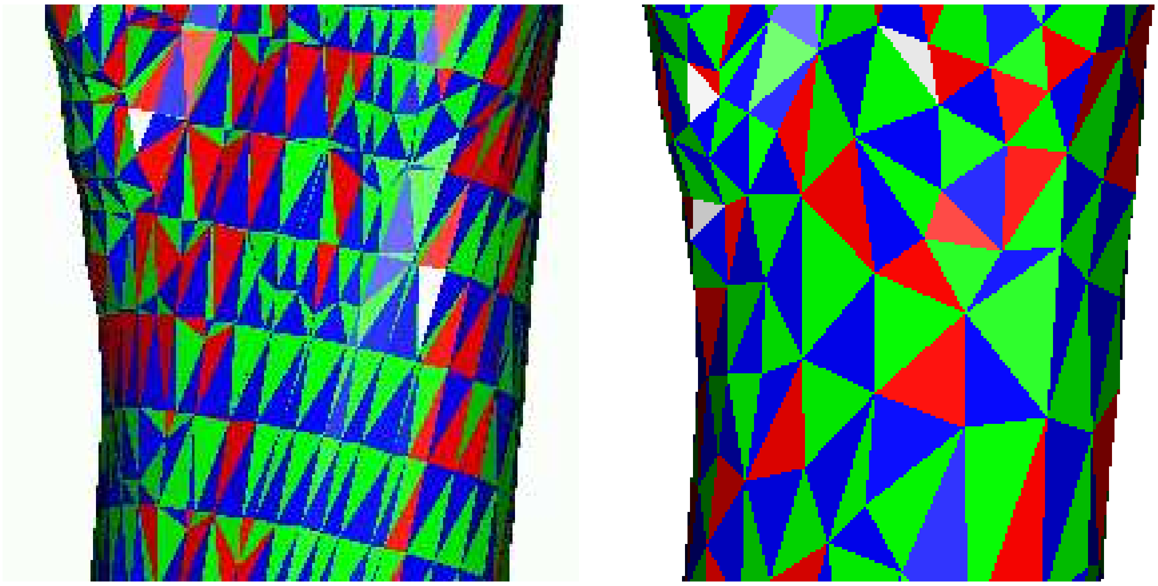

Figure 11: A visual inspection of these triangles extracted from the shaft of the Femur model shows that facets are more regular in the simplified femur model produced by the our algorithm (Right) than they are in the original output of an iso-surface algorithm (Left). In particular, most “sliver” (very narrow) triangles have been removed. Histograms of triangle compactness presented in Fig.10 confirm this observation.

volume must contain subsequent tolerance volumes; for each edge collapse test we assign to the potential simplified vertex $\mathbf{v_0}$, the minimum of the tolerances at vertices in the edge star, τ_0. The collapse is forbidden if any error value in the vertex star exceeds the tolerance value at $\mathbf{v_0}$. If the edge is actually collapsed, the tolerance τ_0 is assigned to the simplified vertex.

Using this simple method for propagating tolerances, we can satisfy Goal II: Consider a point p inside a triangle t_p of the original surface. Suppose that a tolerance τ is assigned to p, as a linear combination of the three tolerances at the vertices of t_p. By construction, in Section 8.2, we map p to a point q on the simplified surface. q is inside a parent triangle t_q. At least one vertex of t_q has a tolerance attached that is less than the minimum tolerance at the vertices of t_p. Using our fifth test, the error ϵ attached to q must then be less than the minimum tolerance in t_p. Finally, p is within some distance ϵ to a point q of the simplified surface, with $\epsilon < \tau$: this is what Goal II prescribes.

Figure 13 illustrates an application of this method to enforcing the tangency of two surfaces after simplification: at the contact point, a tolerance value of zero is used.

8.1. Definitions

Both error and tolerance volumes are very simple objects; they are obtained by sweeping a ball across the surface; the radius of this ball is specified at vertices; between vertices, we use linear interpolation, implemented using barycentric coordinates, to determine the radius.

Here, a triangle t is a triple of vertices $\Delta(\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2})$. Another notation for a triangle is to use the

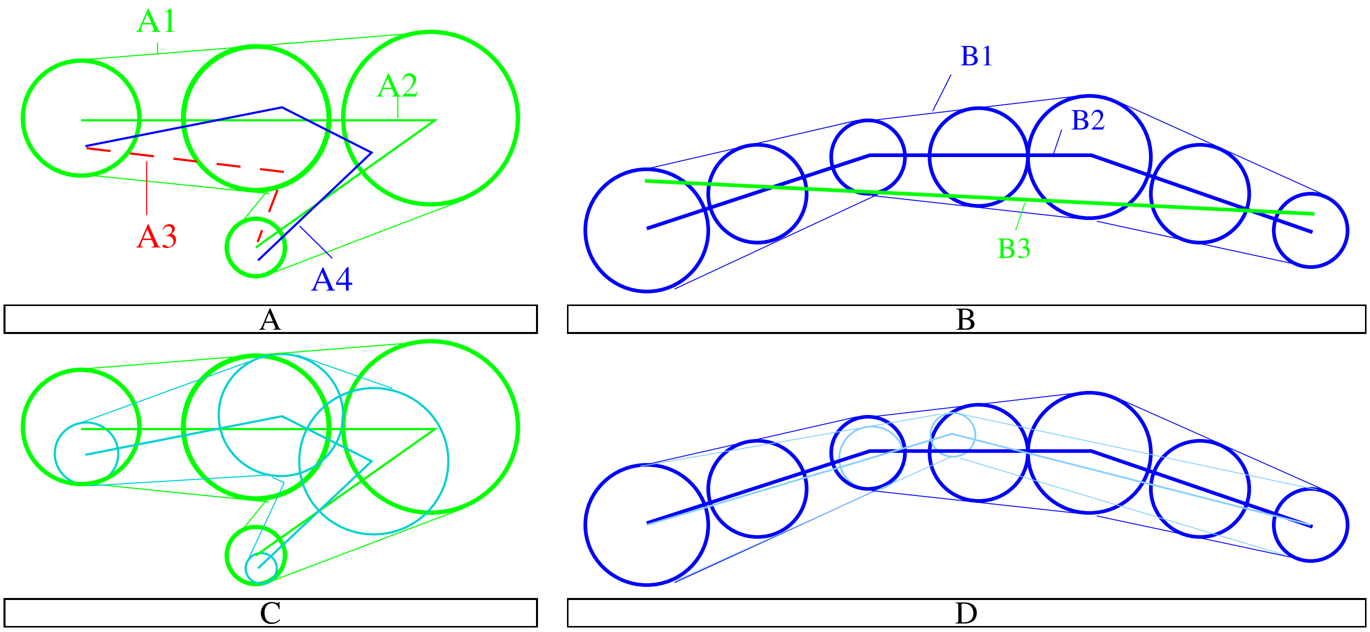

Figure 12: A: Error volume, in green (A1), centered on the simplified surface (A2). The original surface (here, curve) is not only contained in the error volume, in dashed red (A3), but also intersects all the spheres, in blue (A4). B: Tolerance volume, in blue (B1), centered on the original surface (B2). The simplified surface (B3) is contained inside the tolerance volume and intersects all the spheres. C: The final error volume contains all the intermediate error volumes. D: The initial tolerance volume contains all the intermediate tolerance volumes. A simplification operation is rejected if the width of the resulting tolerance volume is negative.

tensor $\bar{\mathbf{v}}$ of coordinates of all three vertices. We first recall that for each vertex $\mathbf{v}$ lying on the plane of the triangle, the *barycentric coordinates* are the three values $(\alpha_0, \alpha_1, \alpha_2)$ such that:

$$\begin{cases} \sum \alpha_i & = 1 \\ \sum \alpha_i \mathbf{v_i} & = \mathbf{v} \end{cases}$$

If we employ a tensor notation for the vertices, whereby $\bar{\mathbf{v}}$ designates the matrix whose three columns are $(\mathbf{v_0}, \mathbf{v_1}, \mathbf{v_2})$ and for the barycentric coordinates $\alpha = (\alpha_0, \alpha_1, \alpha_2)^T$, the last expression becomes: $\bar{\mathbf{v}} \cdot \alpha = \mathbf{v}$.

Definition 1. *An* edge tube $N((\mathbf{v_1}, \mathbf{v_2}), (\epsilon_1, \epsilon_2))$ *is the union of all balls*

$$B(\lambda \mathbf{v_1} + (1-\lambda) v_2, \lambda \epsilon_1 + (1-\lambda)\epsilon_2),\ 0 \leq \lambda \leq 1,$$

where λ and $1-\lambda$ are the barycentric coordinates of the center of the ball with respect to the vertices $\mathbf{v_1}$ and $\mathbf{v_2}$, and $\epsilon_1, \epsilon_2 \geq 0$.

The "N" notation for a tube recalls the similarity with a tubular neighborhood. The edge tube is illustrated in Fig. 14. Error and tolerance volumes of a curve are obtained by taking the union of edge tubes.

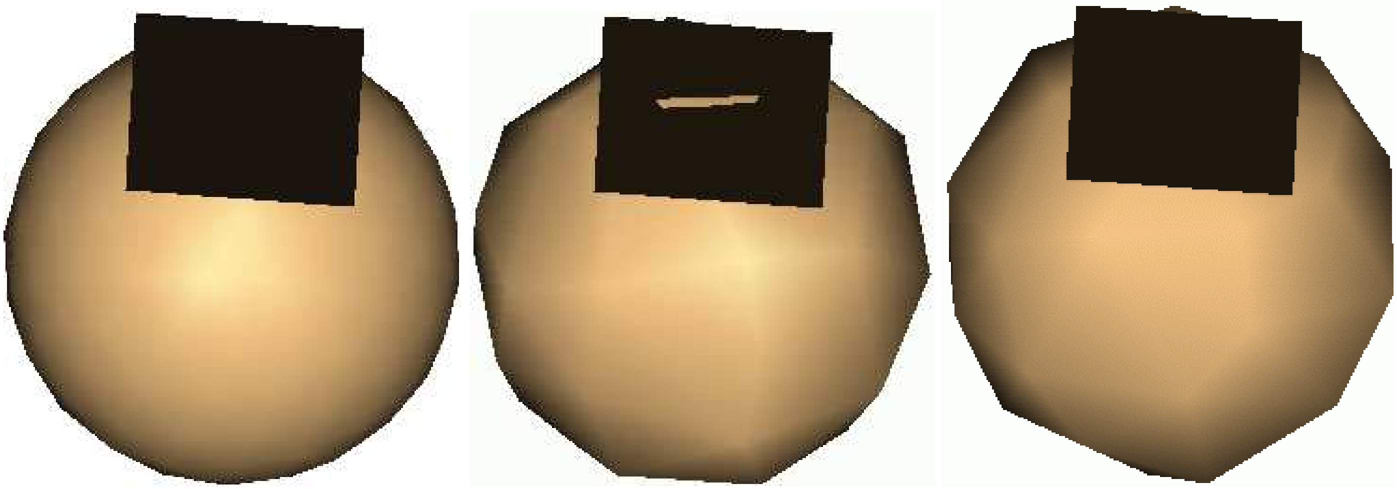

Figure 13: Using the tolerance volume to maintain tangency of two surfaces after simplification. From Left to Right, original sphere model (320 triangles) represented with tangent plane, simplified sphere model (82 triangles) with a tolerance equal to 10 % of the sphere diameter; the tangency condition is violated. Simplified sphere model (88 triangles) with the same global tolerance, but with a tolerance of 0 at the contact point.

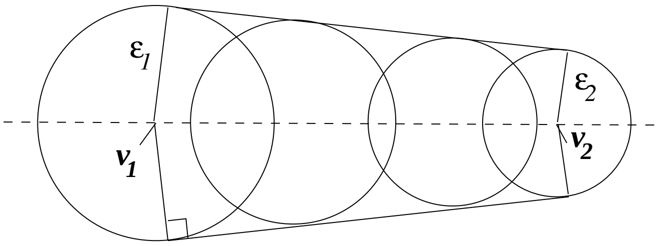

Figure 14: An edge tube

Definition 2. *A* triangle tube $N(\bar{\mathbf{v}}, \epsilon)$ *is the union of all balls:*

$$B(\bar{\mathbf{v}} \cdot \alpha, \epsilon \cdot \alpha), \sum_i \alpha_i = 1, \alpha_i \geq 0,$$

whose centers are linear combinations of the triangle vertices and whose radii are linear combinations of the ϵ values at the vertices, using barycentric coordinates α. (We suppose $\epsilon_i \geq 0$.)

The triangle tube is also the convex union of three edge tubes and two prisms (Fig. 15). The prisms p_h and p_l are comprised within the plane of the triangle P and the planes P_h and P_l tangent to the three balls $B_i(\mathbf{v_i}, \epsilon_i)$ such that the three balls are below P_h and above P_l, when P is a horizontal plane. The prism p_h is shown shaded in Fig. 15. The angle between P and P_h, i.e., the angle between the normals of the planes $\mathbf{n}$ and $\mathbf{n_h}$, is noted θ. The distance from each $\mathbf{v_i}$ to the planes P_h and P_l is $\epsilon_i / \cos\theta$. If the triangle tube is relatively flat, meaning that the three error values are small compared with the three edge lengths, we can approximate the triangle tube volume V as the sum of the volumes

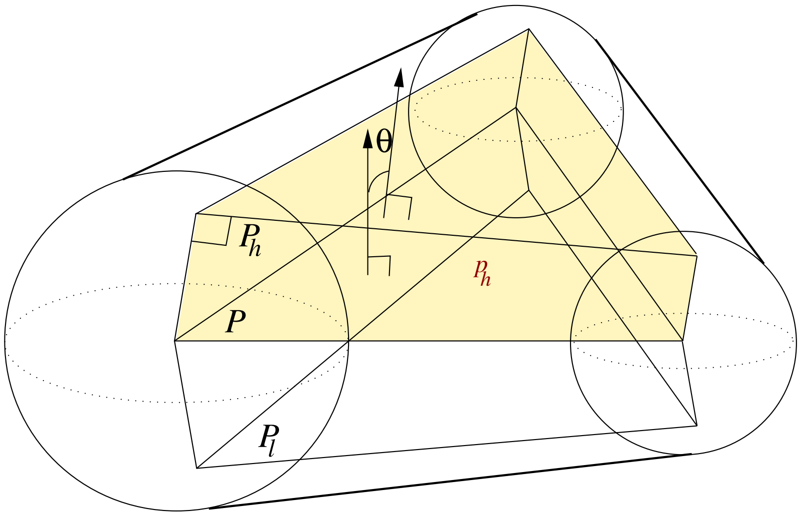

Figure 15: A triangle tube.

of the two prisms p_h and p_l:

$$V \simeq a\frac{\epsilon_0 + \epsilon_1 + \epsilon_2}{3\cos\theta} \tag{8.1}$$

where a is the triangle area and θ is the aforementioned angle.

Definition 3. *The* error volume E *associated with a surface* $S(\{\mathbf{v_i}\}, \{t_j\})$*, and a set* $\{\epsilon_i\}$ *of error values, with one error value for each vertex, is the union of all triangle tubes* $N(\bar{\mathbf{v}}_\mathbf{j}, \epsilon_\mathbf{j})$ *of the surface, where* $\epsilon_\mathbf{j}$ *is a triple of error values associated with the triangle* $\bar{\mathbf{v}}_\mathbf{j}$*.*

$$E(S, \{\epsilon_i\}) = \bigcup_j N(\bar{\mathbf{v}}_\mathbf{j}, \epsilon_\mathbf{j}).$$

The tolerance volume is defined similarly, using tolerance values $\{\tau_i\}$ instead of error values.

8.2. Computation of Error and Tolerance Values

The tolerance values are computed as follows: to satisfy Goal I, whereby the tolerance is the same everywhere on the surface, it is not necessary to compute explicitely the tolerance volume; to satisfy Goal II, as explained earlier, we assign the minimum tolerance value in the edge star $(\mathbf{v_1}, \mathbf{v_2})^\star$ to the simplified vertex $\mathbf{v_0}$. We next explain how the error values are computed.

We determine new error values $\{\delta_i\}$ at the vertices of the simplified star $\mathbf{v_0}^\star$ such that the old error volume $E((\mathbf{v_1}, \mathbf{v_2})^\star, \{\epsilon_i\})$ will be nested inside the new error volume $E(\mathbf{v_0}^\star, \{\delta_i\})$. We also choose these new error values such that all balls associated with the new error volume contain a ball of the error volume $E((\mathbf{v_1}, \mathbf{v_2})^\star, \{\epsilon_i\})$: each ball must then necessarily intersect the original surface. We call such an error volume a *valid* volume. Clearly, there is no unique valid error volume. Supposing that $E(\mathbf{v_0}^\star, \{\mu_i\})$ is a valid error volume, $E(\mathbf{v_0}^\star, \{\nu_i\})$, with $\nu_i \geq \mu_i$ is also valid: there is a one to one mapping between each ball associated with the error volumes, and each ball of the second volume

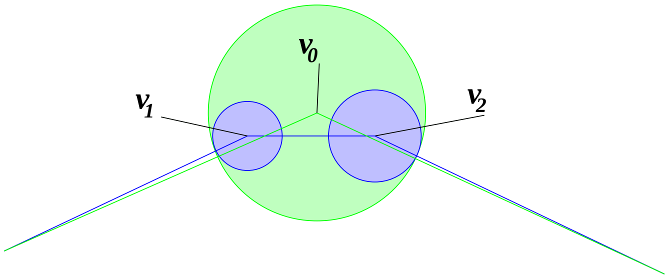

Figure 16: Simplest method (Contraction Method) for computing error values.

contains the corresponding ball from the first volume. However, we can define the notion of an error volume of minimum size.

Accordingly, we proceed in two steps. In the first step, we collect constraints on the unknown error values $\{\delta_i\}$ that a valid error volume must satisfy. In the second step, we practically compute an error volume that satisfies the constraints; and we propose different interpretations for an error volume of minimum size.

Before, we mention one simple strategy for choosing error values, which we originally used: error values at the link $\ell(\mathbf{v_0})$ are copied from the link $\ell(\mathbf{v_1}, \mathbf{v2})$. δ_0 is chosen such that the Ball $B(\mathbf{v_0}, \delta_0)$ centered at $\mathbf{v_0}$ and of radius δ_0 contains both balls $B(\mathbf{v_1}, \delta_1)$ and $B(\mathbf{v_2}, \delta_2)$, as shown in Fig. 16: clearly, we have defined a valid error volume; this is a particular case of the methods that follow, whereby the particular map from $(\mathbf{v_1}, \mathbf{v_2})^\star$ to $\mathbf{v_0}^\star$ maps both $\mathbf{v_1}$ and $\mathbf{v_2}$ to $\mathbf{v_0}$. As this map results directly from an edge contraction (when considering the graph of surface edges), without additional computations relative to adjacent edges or triangles, we call this method the Contraction Method. Experimental results obtained with the Contraction Method are reported in [3].

8.2.1. Constraints on the Error Values. To guarantee that each point of the simplified surface is within some distance bound to the original and vice versa, we compute either two maps between simplified and original, and original and simplified surfaces (Projection Method) or one bijective map (Subdivision Method). We provide implementations for both solutions. Both types of maps are piecewise linear. The bijective map is obtained using a method related to Kent and coworkers [47]. For each vertex between linear portions of such maps, we collect one constraint on the new error values. We next detail of to compute the maps between an edge star $(\mathbf{v_1}, \mathbf{v_2})^\star$ and the resulting simplified vertex star $\mathbf{v_0}^\star$ after edge collapse. By applying these maps in succession, we can map points of the original surface to the final simplified surface and vice versa.

Throughout the constraint collection process, we need to compute the barycentric coordinates of a point with respect to a triangle, as well as the height of the point below or above the plane of the triangle. We perform this computation as follows:

We consider a triangle t_j with a normal vector $\mathbf{n_j}$ and supported by a plane P_j. Given the three vertices of the triangle $t_j = \Delta(j_0, j_1, j_2) = \bar{\mathbf{v}}_\mathbf{j}$, the barycentric coordinates $\alpha = (\alpha_0, \alpha_1, \alpha_2)^T$ of the vertex $\mathbf{v}$ of coordinates (x, y, z) with respect to the vertices (j_0, j_1, j_2) and its height h above the plane

P_j satisfy the following equation:

$$\mathbf{v} = \bar{\mathbf{v}}_\mathbf{j} \cdot \alpha + \mathbf{n_j} h \tag{8.2}$$

After permutation, we assume that (j_0, j_1) is the shortest edge of the triangle: as observed in Appendix A, this performs better numerically. In order to determine the barycentric coordinates, we solve the following system using Normal Equations:

$$\begin{bmatrix} x_{j_1} - x_{j_0} & x_{j_2} - x_{j_0} \\ y_{j_1} - y_{j_0} & y_{j_2} - y_{j_0} \\ z_{j_1} - z_{j_0} & z_{j_2} - z_{j_0} \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} x - x_{j_0} \\ y - y_{j_0} \\ z - z_{j_0} \end{bmatrix} \tag{8.3}$$

α_0 is set to $1 - \alpha_1 - \alpha_2$. h is obtained by substituting α into Eqn. 8.2: $h = (\mathbf{v} - \bar{\mathbf{v}}_\mathbf{j} \cdot \alpha) \cdot \mathbf{n_j}$.

Subdivision Method. The Subdivision Method consists of constructing a graph G that is imbedded in $(\mathbf{v_1}, \mathbf{v_2})^\star$ together with a graph H that is imbedded in $\mathbf{v_0}^\star$, such that there is a graph isomorphism p_1, i.e., a one to one map, between G and H. p_1 is one to one because G and H have corresponding vertices and corresponding edges. After identification of corresponding vertices and edges, we identify corresponding faces by looping around edges: since each graph is imbedded, there is a unique collection of faces for each graph. In the beginning G contains the m vertices $\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}, \ldots, \mathbf{v_r}, \ldots, \mathbf{v_m}$ and the $m+1$ edges of $(\mathbf{v_1}, \mathbf{v_2})^\star$, and H contains the $m-1$ vertices $\mathbf{v_0}, \mathbf{v_3}, \ldots, \mathbf{v_r}, \ldots, \mathbf{v_m}$ and $m-2$ edges of $\mathbf{v_0}^\star$ (link edges are not counted). The notations used for vertices and triangles of $\mathbf{v_0}^\star$ and $(\mathbf{v_1}, \mathbf{v_2})^\star$ are specified in Fig. 17.

The Subdivision Method begins by assigning three arbitrary vertex-triangle correspondences, and adding one vertex and three edges to G and two vertices and six edges to H. The process is completed by assigning correspondences between edges of $(\mathbf{v_1}, \mathbf{v_2})^\star$ and $\mathbf{v_0}^\star$; it may be necessary to subdivide edges and add additional corresponding edges; accordingly, we determine *bridges* when computing the shortest distance between edges; bridges are precisely defined below. To prove that the method works, we explain how to build G and H. In practice, we collect constraints, and neither G nor H are explicitely built.

Step I. First, we determine the closest point $\mathbf{w_0}$ to the vertex $\mathbf{v_0}$ on the triangles t_0 and t_1. The closest point from a vertex to a triangle, as well as the distance from the vertex to the triangle are obtained by computing the barycentric coordinates and height of the vertex with respect to the triangle as set forth above. If all three barycentric coordinates are positive, then the closest point is the point defined with the barycentric coordinates and the distance is the absolute value of the height. If one barycentric coordinate is negative, then the closest point on the triangle edge associated with the two positive barycentric coordinates is determined; finally, if two barycentric coordinates are negative, the closest point is the vertex associated with the positive barycentric coordinate.

We associate with the couple of points $(\mathbf{v_0}, \mathbf{w_0})$ the distance d_0 from $\mathbf{v_0}$ to $\mathbf{w_0}$ and the barycentric coordinates $\alpha_\mathbf{0}$ of $\mathbf{w_0}$ with respect to the triangle t_j on which $\mathbf{w_0}$ lies. A constraint associated with $(\mathbf{v_0}, \mathbf{w_0})$ is derived as follows:

$$\delta_0 \geq d_0 + \epsilon_\mathbf{j} \cdot \alpha_\mathbf{0}, \tag{8.4}$$

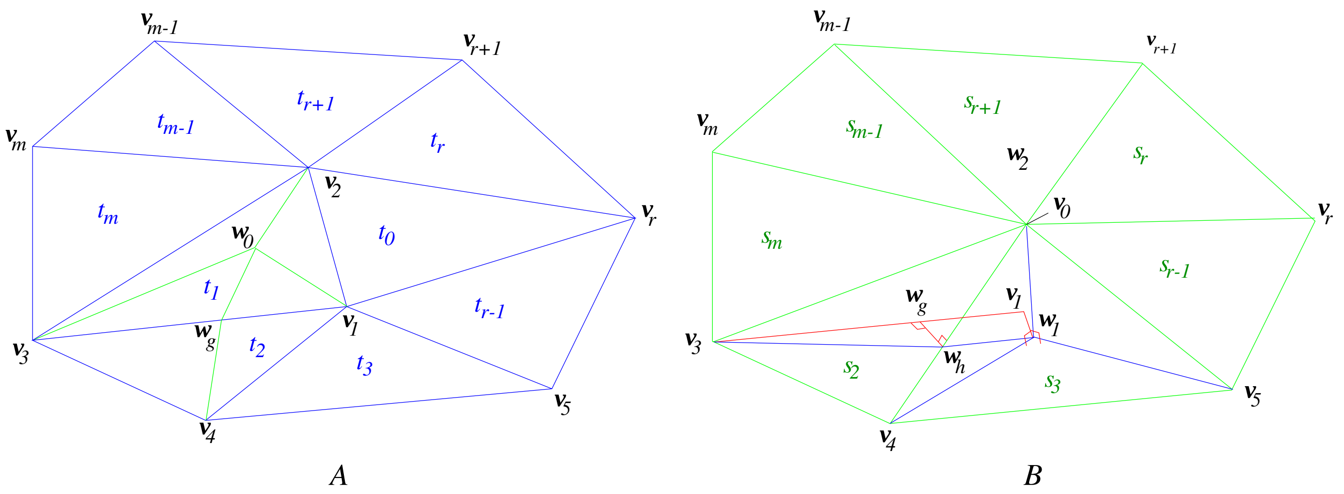

Figure 17: Notations for the Subdivision Method. A: graph G. B: graph H

where $\epsilon_\mathbf{j}$ is the triple of error values associated with t_j.

The vertex $\mathbf{w_0}$ as well as the three edges between $\mathbf{w_0}$ and the vertices of t_j (in the configuration of Fig. 17, such vertices are $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$) are inserted in the graph G.

Step II. Then, we determine the closest point to the vertex $\mathbf{v_1}$ on the triangles $s_2, s_3, \ldots, s_{r-1}$. The position of this point is obtained by projecting $\mathbf{v_1}$ on each triangle of the sequence $s_2, s_3, \ldots, s_{r-1}$ and by retaining the projected point that is closest to $\mathbf{v_1}$. This point is denoted $\mathbf{w_1}$. The couple of points $(\mathbf{v_1}, \mathbf{w_1})$ is associated with the distance d_1 from $\mathbf{v_1}$ to $\mathbf{w_1}$ and with the barycentric coordinates $\alpha_\mathbf{1}$ of $\mathbf{w_1}$ with respect to the triangle s_j on which $\mathbf{w_1}$ lies. For instance, in Fig. 17, these are vertices $\mathbf{v_0}$, $\mathbf{v_4}$ and $\mathbf{v_5}$. The constraint associated with $(\mathbf{v_1}, \mathbf{w_1})$ is:

$$\alpha_{10}\delta_0 + \alpha_{11}\delta_4 + \alpha_{12}\delta_5 \geq d_1 + \epsilon_1, \tag{8.5}$$

whereby ϵ_1 is the error associated with $\mathbf{v_1}$. The vertex $\mathbf{w_1}$ as well as the three edges $(\mathbf{v_0}, \mathbf{w_1})$, $(\mathbf{v_4}, \mathbf{w_1})$ and $(\mathbf{v_5}, \mathbf{w_1})$ are inserted in the graph H.

Step III. Then, we determine the closest point $\mathbf{w_2}$ to the vertex $\mathbf{v_2}$ on the triangles $s_r, \ldots, s_m$. The constraint associated with $(\mathbf{v_2}, \mathbf{w_2})$ is processed similarly as the constraint associated with $(\mathbf{v_1}, \mathbf{w_1})$. From then on G and H each have $m+1$ vertices and $m+4$ edges.

Step IV. For each edge e of the sequence $(\mathbf{v_0}, \mathbf{v_3}), (\mathbf{v_0}, \mathbf{v_4}), \ldots, (\mathbf{v_0}, \mathbf{v_r})$, we take in turn each edge f of the sequence $(\mathbf{v_1}, \mathbf{v_2}), (\mathbf{v_1}, \mathbf{v_3}), (\mathbf{v_1}, \mathbf{v_4}), \ldots, (\mathbf{v_1}, \mathbf{v_r})$. We skip an edge pair (e, f) sharing one endpoint: for instance the pair $((\mathbf{v_0}, \mathbf{v_3}), (\mathbf{v_1}, \mathbf{v_3}))$ is skipped. For each remaining edge pair (e, f), the shortest distance between the edges d is measured. For instance for the edge pair $((\mathbf{v_0}, \mathbf{v_4}), (\mathbf{v_1}, \mathbf{v_3}))$ the shortest distance occurs for a point $\mathbf{w_h}$ on $(\mathbf{v_0}, \mathbf{v_4})$ and a point $\mathbf{w_g}$ on $(\mathbf{v_1}, \mathbf{v_3})$. If the segment $(\mathbf{w_h}, \mathbf{w_g})$ is perpendicular to both $(\mathbf{v_0}, \mathbf{v_4})$ and $(\mathbf{v_1}, \mathbf{v_3})$ (this is not necessarily the case when either $\mathbf{w_h}$ or $\mathbf{w_g}$ is an endpoint of its supporting edge) such pair of points $(\mathbf{w_h}, \mathbf{w_g})$ is called a *bridge*. For each bridge, we compute λ and μ as explained shortly. λ and μ satisfy: $\mathbf{w_h} = \lambda \mathbf{v_0} + (1-\lambda)\mathbf{v_4}$ and $\mathbf{w_g} = \mu \mathbf{v_1} + (1-\mu)\mathbf{v_3}$. We then derive the constraint:

$$\lambda \delta_0 + (1-\lambda)\delta_4 \geq d + \mu \epsilon_1 + (1-\mu)\epsilon_3 \tag{8.6}$$

The following changes occur in the graph G: the vertex $\mathbf{w_g}$ is inserted in G; accordingly, the edge $(\mathbf{v_1}, \mathbf{v_3})$ is subdivided into $(\mathbf{v_1}, \mathbf{w_g})$ and $(\mathbf{w_g}, \mathbf{v_3})$. The two edges $(\mathbf{w_0}, \mathbf{w_g})$ and $(\mathbf{w_g}, \mathbf{v_4})$ are inserted in G. For subsequent subdivisions, $\mathbf{w_g}$ will play the role of $\mathbf{v_3}$.

Similarly, the following changes occur in the graph H: the vertex $\mathbf{w_h}$ is inserted in H; the edge $(\mathbf{v_0}, \mathbf{v_4})$ is subdivided into $(\mathbf{v_0}, \mathbf{w_h})$ and $(\mathbf{w_h}, \mathbf{v_4})$. The two edges $(\mathbf{v_3}, \mathbf{w_h})$ and $(\mathbf{w_h}, \mathbf{w_1})$ are inserted in H. For subsequent subdivisions, $\mathbf{w_h}$ will play the role of $\mathbf{v_0}$.

If there is no bridge, we simply add $(\mathbf{w_0}, \mathbf{v_4})$ to G and $(\mathbf{v_3}, \mathbf{w_1})$ to H. In all cases, the resulting number of vertices and edges are the same in G and H.

Step V. We re-apply Step IV to edge pairs of the sequences $(\mathbf{v_0}, \mathbf{v_3}), (\mathbf{v_0}, \mathbf{v_m}), (\mathbf{v_0}, \mathbf{v_{m-1}}), \ldots, (\mathbf{v_0}, \mathbf{v_{r+1}}), (\mathbf{v_0}, \mathbf{v_r})$ and $(\mathbf{v_2}, \mathbf{v_1}), (\mathbf{v_2}, \mathbf{v_3}), (\mathbf{v_2}, \mathbf{v_m}), (\mathbf{v_2}, \mathbf{v_{m-1}}), \ldots, (\mathbf{v_2}, \mathbf{v_{r+1}}), (\mathbf{v_2}, \mathbf{v_r})$.

Bridges. Finding the shortest point-pair between two segments $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{c}, \mathbf{d})$ requires finding a pair of barycentric coordinates (λ, μ): λ for $(\mathbf{a}, \mathbf{b})$ and μ for $(\mathbf{c}, \mathbf{d})$. This is, in fact, a one-dimensional problem: our solution begins by translating, rotating and scaling $(\mathbf{a}, \mathbf{b})$ such that $\mathbf{a}$ is mapped to the origin $(0,0,0)$ and $\mathbf{b}$ is mapped to $(0,0,1)$, as shown in Fig. 18. The rotation and scaling is implemented using a Householder transform followed with scaling, as in Section 6. In the Subdivision Method, a given edge $(\mathbf{a}, \mathbf{b})$ is compared with several edges $(\mathbf{c}, \mathbf{d}), (\mathbf{c}, \mathbf{e}), \ldots, (\mathbf{c}, \mathbf{r})$. We take advantage of such repetitions by performing the coordinate transformation only once for all vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \ldots, \mathbf{r}$. In the new coordinate system, we note (c_x, c_y, c_z) and (d_x, d_y, d_z) the coordinates of $\mathbf{c}$ and $\mathbf{d}$. We also note $\mathbf{v} = \mathbf{c} - \mathbf{d}$. Clearly, there is no bridge if $c_z, d_z < 0$ or $c_z, d_z > 1$. Otherwise, if the segments are parallel ($v_x = v_y = 0$), we can easily choose one bridge for which $0 <= \lambda, \mu <= 1$.

We next suppose that the segments are not parallel. We parameterize the segment $(\mathbf{c}, \mathbf{d})$ using μ: $x = d_x + \mu v_x, y = d_y + \mu v_y$. The closest point pair between the lines supporting the segments $(\mathbf{a}, \mathbf{b})$ and $(\mathbf{c}, \mathbf{d})$ corresponds to the minimum of $d^2 = x^2 + y^2$, which occurs for $\mu = -(v_x d_x + v_y d_y)/(v_x^2 + v_y^2)$. if $0 \leq \mu \leq 1$ we compute the corresponding $z = d_z + \mu v_z$ and we determine that there is a bridge if $0 \leq (\lambda = 1 - z) \leq 1$; we then compute the shortest distance $d = \sqrt{d_x^2 + d_y^2 - \mu^2(v_x^2 + v_y^2)}$.

Projection Method. The Projection Method requires more tests than the Subdivision Method. However, the maps produced are entirely dependent of the geometry. The Projection Method was partially presented in [3]. It comprises two steps. In Step I, we build a map p_2 from each triangle of $(\mathbf{v_1}, \mathbf{v_2})^\star$ to a connected region of $\mathbf{v_0}^\star$. In step II, we build a map p_3 from each triangle in $\mathbf{v_0}^\star$ to a connected region of $(\mathbf{v_1}, \mathbf{v_2})^\star$. We leave the details of the Projection Method to Appendix B.

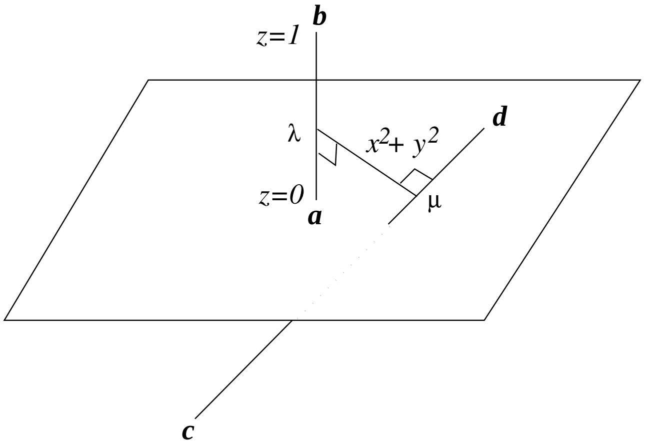

Figure 18: Determination of bridges.

For both Subdivision and Projection Methods. the time necessary for collecting constraints is proportional to the product of the number of triangles in $(\mathbf{v_1}, \mathbf{v_2})^\star$ with the number of triangles in $\mathbf{v_0}^\star$; it is quadratic in the number of triangles in $(\mathbf{v_1}, \mathbf{v_2})^\star$.

8.2.2. Optimum Error Volume. As potential error values are computed each time an edge is tested for collapsing, we attempt to use an inexpensive computation.

A first objective (Objective A) is to minimize the volume actually enclosed by $E(S, \{\delta_i\})$. A practical measure of the volume of a triangle tube is provided in Eqn. 8.1, that holds if the triangle tube is flat. When adding the contributions of all the triangle in $\mathbf{v_0}^\star$, we further simplify the objective by assuming that the angle θ is close to zero, thus ignoring the $\cos\theta$ factor. We obtain a volume V:

$$V = \frac{1}{3}\sum_i \delta_i a(\mathbf{v_i}^\star),$$

where the summation is performed over surface vertices, and where $a(\mathbf{v_i}^\star)$ is the area of a vertex star. Accordingly, a reasonably simple objective is to minimize the sum of the δ_i values in $\mathbf{v_0}^\star$, or, if the areas $a(\mathbf{v_i}^\star)$ are easily available, to minimize the sum of δ_i values weighted by $a(\mathbf{v_i}^\star)$. In both cases, we must solve a linear programming problem. We have implemented the minimization of the sum of δ_i values using the IBM Optimization Subroutine Library (OSL), which is commercially available for a broad range of operating systems and programming environments.

A second objective (Objective B) is to solve a one dimensional linear programming problem: all the new error values δ_i at the link $\ell(\mathbf{v_0})$ are copied from the ϵ_i at the link of $\ell(\mathbf{v_1}, \mathbf{v2})$. In addition we choose the minimum value of δ_0 that satisfies all constraints. Note that some constraints are intractable using Objective B, namely constraints for which the weight of δ_0 is zero. If such constraints occur, and if they are not implicitly satisfied, the potential δ_0 value is set to infinity (or a large value), which will result in forbidding the collapse.

The maximum amount of triangle reduction reduction given the same set of tolerances was obtained by pursuing Objective B, using constraints collected with the Subdivision Method. This is also the least computationally expensive combination of methods, excluding the simple Contraction Method.

9. Preserving Vertex Attributes

We aim at preserving one or more attributes associated with each vertex of the original surface. Attributes may represent vertex red, green and blue colors, vertex normals, or other scalar or vector data specified at surface vertices, such as texture coordinates. We assume that color values and vertex data vary linearly on surface triangles. The principle of the method is to use the same maps p_1 or p_2 and p_3 that were used in Section 8 for computing the error volume for the surface position.

We measure discrepancies at the vertices of this map. For normals, a corrective term must be added to the error measured at the vertices to compensate for the fact that they do not vary linearly. Error and tolerance volumes for attributes are defined in the same fashion as for positional error.

9.1. Scalar Attributes

Assuming that a scalar attribute is linearly interpolated between vertices, the difference between the attribute associated with the original surface and the attribute associated with the simplified surface will be bounded by a maximum value γ.

We first assume that the attribute is a real number. $c_1, c_2, \ldots, c_m$ are associated with vertices $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_m}$ of an edge star before simplification. There is an attribute error volume associated with the attribute: at each vertex a positive attribute error value ϕ_i is specified. It indicates that the true attribute value at the vertex is between $c_i - \phi_i$ and $c_i + \phi_i$. We determine new attribute values $d_0, d_3, \ldots, ..d_m$ at the vertices $\mathbf{v_0}, \mathbf{v_3}, \mathbf{v_4}, \ldots, \mathbf{v_m}$ of the vertex star after simplification, as well as new error values $\psi_0, \psi_3, \ldots, \psi_m$. The principle underlying the method is very similar with what was set forth above: constraints are collected on the d_i values. Similarly to Objective B above, the attribute values as well as the attribute error values are copied at the link of the edge star:

$$d_3 = c_3, d_4 = c_4, \ldots, d_m = c_m$$
$$\psi_3 = \phi_3, \psi_4 = \phi_4, \ldots, \psi_m = \phi_m$$

To determine the potential attribute value d_0 and the potential attribute error ψ_0, we rewrite the constraints that were encountered when computing the positional error volume in Section 8 using the following four rules: (1) each h_i value is replaced with zero; (2) each ϵ_j value is replaced with $c_j + \phi_j$; (3) each δ_k value is replaced with $d_k + \psi_k$; (4) each "$\geq$" symbol is replaced with a "$=$" symbol. As with Objective B, the sum $d_k + \psi_k$ is isolated from the constraints. Each such obtained constraint is numbered using a counter i and for each i, we determine λ_i and w_i such that the constraint i can be rewritten as:

$$\lambda_i(d_0 + \psi_0) = w_i,$$

where $0 < \lambda \leq 1$ (recall that when $\lambda = 0$ edge collapses using Objective B are not performed). We then propose three different schemes for computing d_0:

Copy Attribute Value. Set $d_0 = c_1$: This method was used to obtain the results in Fig. 31.

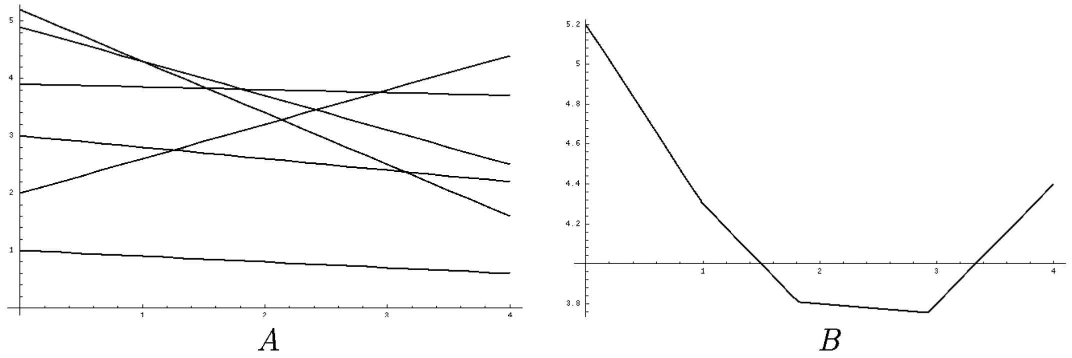

Figure 19: Minimize Maximum Discrepancy. A: data discrepancies as a function of x, new data value. B: the maximum of data discrepancies is a convex function.

Interpolate Attributes Values. Retrieve the correspondent of $\mathbf{v_0}, \mathbf{w_0}$ on the old edge star $(\mathbf{v_1}, \mathbf{v_2})^\star$, as determined in Section 8; use the barycentric coordinates of $\mathbf{w_0}$ to interpolate the attributes values associated with vertices of the old edge star.

Minimize Maximum Discrepancy. Determine the value x that minimizes

$$\max_i |\lambda_i x - w_i|$$

This function is convex, as illustrated in Fig. 19. The minimum, and the solution x can be obtained by solving a one dimensional linear programming problem. Using a method such as in [48], this can be done in time proportional to $O(n)$, where n is the number of constraints. Starting with a point on the graph of the function, the steepest descent method can take time proportional to $O(n^2)$ in the worst case.

The maximum discrepancy, $\max_i |\lambda_i d_0 - w_i|$, is obtained for a particular constraint i_1. The following expression is then evaluated to give ψ_0, which should not exceed γ.

$$\psi_0 = |\lambda_{i_1} d_0 - w_{i_1}|/\lambda_{i_1}$$

This algorithm can be easily extended by allowing to vary γ across the surface, similarly with the positional error tolerance.

9.2. Vector Attributes

Complex numbers and vector attributes are treated as follows. We note $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}$ and $\mathbf{d_0}$ the attribute vectors associated with the vertices. We copy or interpolate the attribute vector $\mathbf{d_0}$ as in the previous section. We compute again the maximum discrepancy $\max_i \|\lambda_i \mathbf{d_0} - \mathbf{w_i}\|_2$ Using a Euclidean distance. The resulting ψ_0 should not exceed γ.

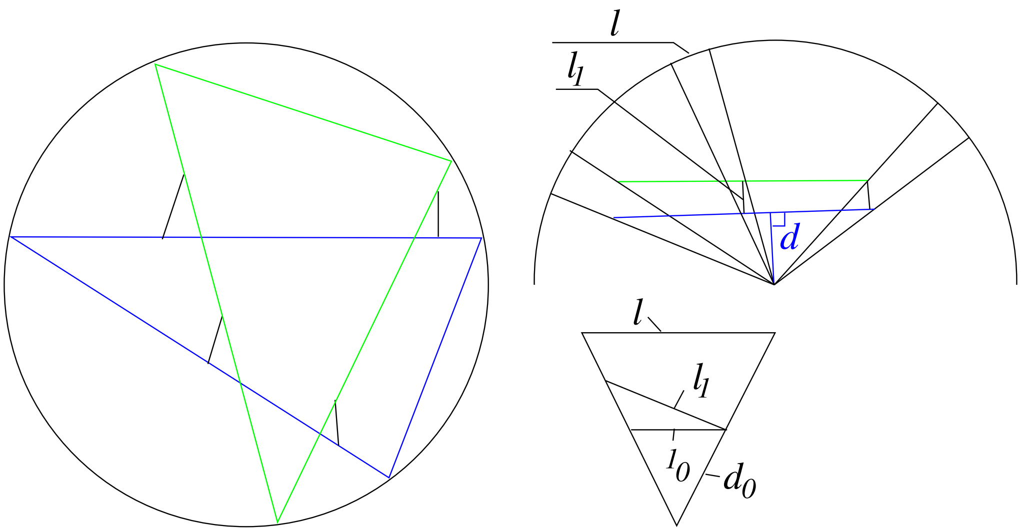

Figure 20: Triangles are mapped to the Gaussian sphere. The maximum discrepancy l between normals after re-normalization is bounded with $\frac{l_1}{d}$

9.3. Normals

Normals do not vary linearly on a surface triangle. Consequently, we cannot assume that the minimum and maximum differences between normals on the simplified and original surfaces occur at the vertices where the constraints were collected.

We consider the normal as an attribute vector, and we linearly interpolate the normal coordinates. We do not renormalize them, but we rather bound the maximum discrepancy after normalization given the discrepancy observed on the interpolated values. We map the triangles to the Gaussian Sphere, as shown in Fig. 20. We measure discrepancies as before with attribute vectors. We divide discrepancies with d, which is the minimum distance to any of the triangles in both stars to the origin of the Gaussian sphere. This distance is very easy to compute: the minimum distance from the origin to a triangle, whose three vertices are located on a sphere centered at the origin, is measured at the triangle centroid. This provides us with a conservative estimate for the maximum discrepancy, allowing us to apply the concepts of error and tolerance volumes to normals as well.

10. Surface Boundary Simplification

We discuss the simplification of Type I and Type II boundary edges. Type III boundary edges are not simplified; simplifying a Type III boundary edge would merge two boundaries and create a singular vertex on the resulting boundary. When simplifying boundary edges, the volume preserving property breaks down. However, for vertices sufficiently distant from the boundary, the vertex positioning method of Section 6 does prevent shrinkage.

The series of tests that are applied to the boundary edges for determining collapsibility are the same as for interior edges, except for the following differences. The minimum valence is one for a simplified vertex resulting from a Type I or II boundary edge collapse. The fifth test (maximum error

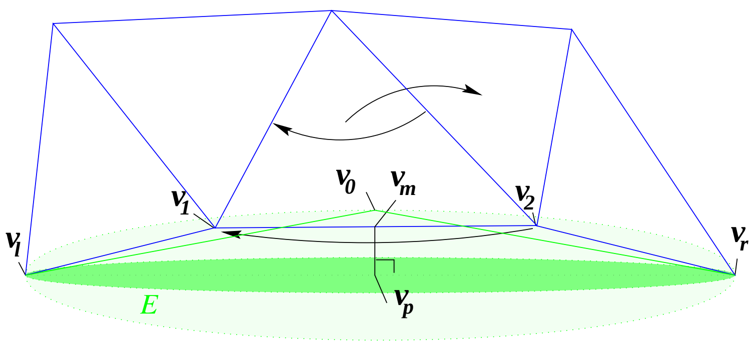

Figure 21: Collapse of a Type I boundary edge. Arrows indicate vertex, triangle and edge representatives after collapse.

lower than tolerance) must be adapted to the particular topology of Types I and II boundary edge neighborhoods. It would be tedious to specify completely these tests here. We leave to the reader the task of performing the necessary modifications with respect to Section 8.2. Also, the computation of the stars is different for boundary vertices, as was explained in Section 5.1.

In the same spirit that guided us for volume preservation, we wish to preserve the length of the boundary, and we position the simplified vertex accordingly.

10.1. Type I Boundary Edges

We consider the Type I boundary edge $(\mathbf{v_1}, \mathbf{v_2})$. We assume that the configuration is that of Fig. 21, after permutation of the vertices $\mathbf{v_1}$ and $\mathbf{v_2}$ if necessary. To preserve the boundary length, the simplified vertex must be located on an ellipsoid E; the focal points of the ellipsoid E are $\mathbf{v_r}$ and $\mathbf{v_l}$; $\mathbf{v_l}$ denotes the leftmost vertex adjacent to $\mathbf{v_1}$ in $(\mathbf{v_1}, \mathbf{v_2})^\star$, as shown in Fig. 21; similarly, $\mathbf{v_r}$ is the rightmost vertex adjacent to $\mathbf{v_2}$ in $(\mathbf{v_1}, \mathbf{v_2})^\star$; we note c half the distance between $\mathbf{v_r}$ and $\mathbf{v_l}$. We compute the cord length of the polygonal arc $(\mathbf{v_l}, \mathbf{v_1}, \mathbf{v_2}, \mathbf{v_r})$; we note a half this length. E is completely defined with $\mathbf{v_r}$, $\mathbf{v_l}$ and a, half the long axis length.

Knowing E, one solution would be to choose a point on E minimizing a measure of distance to the polygonal arc $(\mathbf{v_l}, \mathbf{v_1}, \mathbf{v_2}, \mathbf{v_r})$, using an optimization method. We present a simpler solution for positioning $\mathbf{v_0}$ on E. We take the middle $\mathbf{v_m}$ of $\mathbf{v_1}$ and $\mathbf{v_2}$. Then, we determine whether a and c are different; otherwise, we use $\mathbf{v_m}$ for the simplified vertex $\mathbf{v_0}$, since $a = c$ is equivalent to the vertices $\mathbf{v_l}$, $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_r}$ being aligned.

If it is determined that $a > c$, we find the closest point $\mathbf{v_p}$ to $\mathbf{v_m}$ on the segment $(\mathbf{v_r}, \mathbf{v_l})$ using known methods. To obtain $\mathbf{v_0}$, we project $\mathbf{v_p}$ along the direction $\overrightarrow{\mathbf{v_p}\mathbf{v_m}}$ onto the ellipsoid E. See Figs. 21 and 22.

Specifically, we consider the coordinate system formed with the normalized vectors $\overrightarrow{\mathbf{v_l}\mathbf{v_r}}$ and $\overrightarrow{\mathbf{v_p}\mathbf{v_m}}$, centered in the middle of $(\mathbf{v_r}, \mathbf{v_l})$; this coordinate system is not necessarily orthogonal. In this coordinate system, we denote by x and y the coordinates of $\mathbf{v_m}$. y is simply the distance between $\mathbf{v_p}$ and $\mathbf{v_m}$. See Fig. 22.

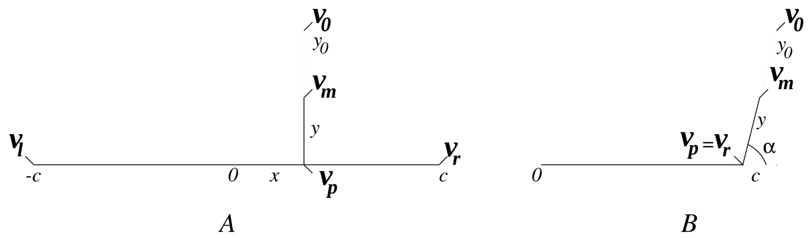

Figure 22: Type I boundary edge: positioning the simplified vertex $\mathbf{v_0}$. A: $\mathbf{v_p}$ is different from $\mathbf{v_r}$ and $\mathbf{v_l}$. B: $\mathbf{v_p}$ is identified with $\mathbf{v_r}$.

If $y = 0$ the ellipsoid is still non-degenerate, since we ruled out the possibility that $a = c$, but $\mathbf{v_m}$ is accidentally on the segment $(\mathbf{v_r}, \mathbf{v_l})$. In this case, we replace $\mathbf{v_m}$ with $\mathbf{v_1}$ or $\mathbf{v_2}$ and proceed. Both $\mathbf{v_1}$ and $\mathbf{v_2}$ cannot lie inside the segment $(\mathbf{v_r}, \mathbf{v_l})$ as this would violate the condition $a \neq c$. In the sequel, we suppose $y > 0$.

If $\mathbf{v_p}$ is different from $\mathbf{v_r}$ and $\mathbf{v_l}$ (i.e., if the closest point on the segment is inside the segment), we compute y_0 such that $\frac{x^2}{a^2} + \frac{y_0^2}{b^2} = 1$ with $b^2 = a^2 - c^2$. We take for y_0 the positive square root of $(1 - \frac{x^2}{a^2})(a^2 - c^2)$. Finally, $\mathbf{v_0}$ is given by the formula:

$$\mathbf{v_0} = \mathbf{v_p} + \frac{y_0}{y}\,\overrightarrow{\mathbf{v_p v_m}} \tag{10.1}$$

If $\mathbf{v_p}$ is identified with $\mathbf{v_r}$ or $\mathbf{v_l}$, we determine the angle α between $\overrightarrow{\mathbf{v_l v_r}}$ and $\overrightarrow{\mathbf{v_p v_m}}$. y_0 is then solution of the following second degree equation:

$$\frac{(y_0 \cos\alpha + c)^2}{a^2} + \frac{y_0^2 \sin^2\alpha}{a^2 - c^2} = 1,$$

for which there is only one positive solution. We then apply Eqn. 10.1 to determine v_0.

10.2. Type II Boundary Edges

Type II boundary edges (incident to one single boundary vertex) present no particular difficulties. Let $(\mathbf{v_1}, \mathbf{v_2})$ be the Type II boundary edge under testing, and suppose that $\mathbf{v_1}$ is the boundary vertex among the two vertices. Since we wish to preserve the boundary length, we slide vertex $\mathbf{v_2}$ along the edge $(\mathbf{v_1}, \mathbf{v_2})$ until it is identified with $\mathbf{v_1}$ which becomes the location of the simplified vertex. The next tests are essentially the same as in section 8.2 with the exception that only one half of an interior edge star is tested.

11. Computational Complexity

We prove that the computational complexity of the method is linear in the number of edges when using an accumulator and sub-quadratic in the number of edges when using a priority queue. When testing an edge for potential collapse, the time necessary to complete the tests is bounded by a factor of the number of edges that are adjacent to that edge. Generally this number, the edge valence, varies little during simplification, which explains the little attention it was given by other researchers. An exception is found on surfaces with a Euler number significantly different from zero, or on surfaces where extreme simplification ratios are observed, e.g., one hundred or more. These infrequent cases justify to enforce an upper limit $k \geq 3$ to the edge valence, as k plays a critical role in determining the worst case complexity of the method.

A preprocessing step consists of building the accumulator or the priority queue for storing surface edges. It turns out that either operations can be done in time linear in the number of edges. For a priority queue, this is explained in [42]. Subsequently, three types of operations are applied to the accumulator or the priority queue: (1) removing the first element (from the queue, or the first element from the first accumulator cell), (2) adding an element, and (3) deleting an element. We next study both the accumulator and priority queue implementations. One difficulty in the analysis is that after an edge collapses, adjacent edges that have been rejected before may be reentered in the queue; accordingly, the queue length does not decrease monotonically.

11.1. Method Using a Priority Queue

The following discussion applies to the simplification of interior edges. It can easily be generalized to boundary edges. The priority queue is implemented using a binary heap. If after testing, the collapse is not performed, the time necessary to delete the element leading the priority queue is bounded by the base two logarithm of the queue length, since the heap condition must be restored. The initial queue length is equal to the number of surface edges n_e. If the collapse is performed, three edges are removed from the queue, at most $k-3$ are modified, and at most $k-3$ must be reentered in the queue. The number of operations required to insert or remove an element from a priority queue is proportional to $\log_2 n_q$ where n_q is the number of edges enqueued.

A batch of $k+1$ edge collapse tests is called a *test batch*. We next prove a bound on the queue length after each batch, that decreases with the batch number.

Suppose that n test batches have been performed, and suppose that n_s is the number of actual simplifications (edge collapses) that have occurred. The queue length n_q is at most:

$$\begin{aligned} n_q &\leq n_e - 3n_s + kn_s - (n(k+1) - n_s) \\ &= n_e + n_s(k-2) - n(k+1). \end{aligned}$$

If $n_s \geq n$ then there cannot be more than $n_e - 3n$ edges in the queue, since more than $3n$ edges have effectively disappeared and cannot be reinstated. If $n_s < n$ the final length is n_q is at most:

$$\begin{aligned} n_q &\leq n_e + n_s(k-2) - n(k+1) \\ &< n_e + n(k-2) - n(k+1) = n_e - 3n \end{aligned}$$

We have just proven that after n test batches, the length of the queue decreases at least by $3n$. This allows to draw a complexity relation: For simplicity, we introduce m, the integer part of $n_e/3$.

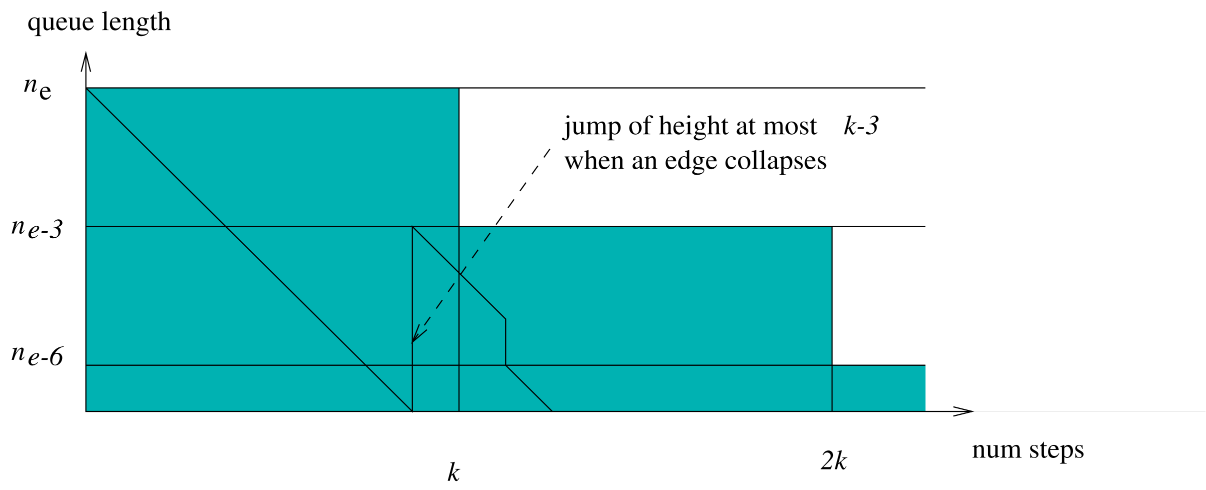

Figure 23: Edge queue length versus number of collapse tests performed. The shaded region shows a bound to the maximum queue length.

The discussion in Section 8 indicates that, independent of queue maintenance, the total cost of testing whether an edge can be simplified is performed in time proportional to k^2 or less. Hence, a number $T_1(m)$ for the computational complexity is given by, using the Stirling formula:

$$\begin{aligned} T_1(m) &= k^2(k+1)(\log_2 m + \log_2(m-1) + \cdots + \log_2 1)(1+o(1)) \\ &= k^3 \log_2(m!)(1+o(1)) \\ &= k^3 \log_2(\exp^{-m-1}(m+1)^{m+\frac{1}{2}}\sqrt{2\pi})(1+o(1)) \\ &= O(k^3 m \log_2 m) = O(k^3 n_e \log_2 n_e) = T_1(n_e) \end{aligned}$$

The complexity is thus sub-quadratic in the initial number of surface edges of the queue.

11.2. Method Using an Accumulator

Removing the first element in a cell requires a constant number of operations; the same applies to adding an element to a cell. In our implementation using linked lists, deleting a particular element from a cell takes at most time proportional to the maximum number of edges e_c in a cell. However, by adjusting the number of cells, e_c can be assumed to be a constant. Or if necessary, we can implement direct indexing in the cells, resulting in a constant cost for deleting. In this case the complexity T_2 is given by:

$$T_2(n_e) = O(ck^3 n_e) = O(k^3 n_e)$$

Since n_v the number of vertices, n_t the number of triangles, and n_e are linearly related with the Euler number, unless this number is significantly higher or lower than zero, the complexity estimates can be applied to triangles and vertices as well.

Although we found no reference to the following observation in previous work, a k^2 or higher factor potentially applies to many simplification methods that locally alter a triangulated surface and produce a new triangulated surface. This is particularly true if enqueuing and testing is used as in our algorithm.

12. Experimental Results

Unless otherwise specified, all timings reported are seconds of CPU time, measured on a IBM RS6000 Model 58K workstation; the simplification error is expressed as a percentage of the bounding box diameter. The bounding box diameter is obtained by determining the smallest box parallel to the x, y and z axes that encloses the model, and by measuring the length of its diagonal. Results were obtained using the volume preservation of Section 6, the Subdivision Method of Section 8 and Objective B, of minimizing the error value at the simplified vertex. Results of the Projection Method combined with Objective A were reported in the technical report referenced as[49].

12.1. Buddha Model

Starting with a volume of sampled densities representing data collected from a statuette using a range scanner, we first build an iso-surface containing 2.7 M (millions of) triangles. In a first step, we reduce it to 334 K (thousands of) triangles and 167 K vertices, with a maximum error of approximately 0.5 pixels. The result is shown in Fig. 24.

A subsequent simplification step, with a maximum tolerance of 2 pixels, brings the count of triangles down to 46.1 K, with 23.0 K vertices. This simplification takes 310 seconds CPU. A different simplification, starting with the same 334 K triangles model, with a maximum tolerance of 2 pixels as well, brings the count of triangles to 49.1 K with 45.5 K vertices, in 313 seconds CPU. A lower tolerance is applied in the facial region. We use 6 spheres to specify the tolerance from 0.1 to 0.5 pixels from the sphere centers to sphere boundaries. The sphere centers were determined by selecting points on a computer display, and by determining the corresponding surface points. The realism of the resulting model is significantly improved, with a marginal penalty in CPU time and number of triangles.

12.2. Lamp Model

We next consider the CAD model of a lamp consisting of 3,058 vertices and 5,052 triangles, visualized in Fig. 25A. In Fig. 25B, we simplify this model down to 528 vertices and 686 triangles in 3.8 seconds CPU. This simplification does not preserve volume: the simplified vertex $\mathbf{v_0}$ is placed at the location of $\mathbf{v_1}$. The volume preserving simplification, in Fig. 25C, reduces the model to 1,219 vertices and 2,035 triangles in 4.9 seconds. We use a maximum tolerance of 0.34 for both simplifications, corresponding to 1% of the bounding box diameter of 34.1. The volume preservation requires more triangles because of the large angular variations of triangle normals in the model.

In Fig. 25D, we show a simplification that respects a distance between original and simplified surface, not between simplified and original (not satisfying Goal I).

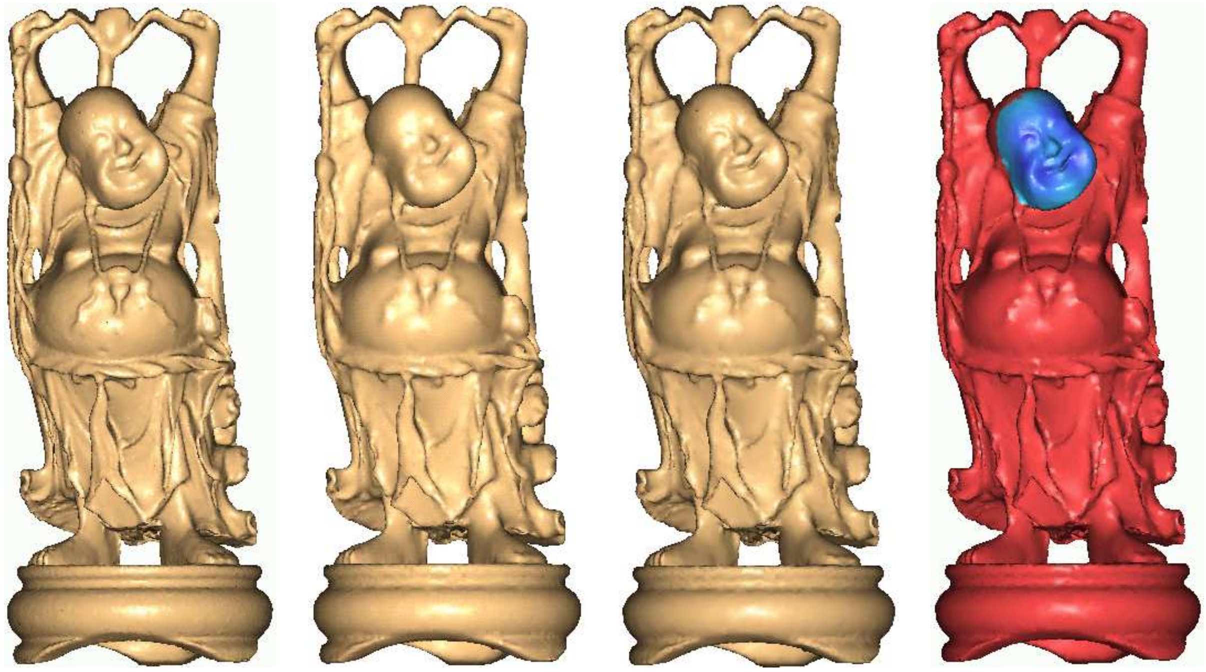

Figure 24: A: Buddha model of 334 K triangles. B: simplification with 46 K triangles using a uniform tolerance. C: simplification with 49 K triangles using a variable tolerance. D: coloring of the tolerance volume for the surface of C, with increasing values from blue to red in a Rainbow colormap.

12.3. Femur Model

We consider the model of a human femur, shown in Fig 26, with 180.8 K triangles and 90.4 K vertices, defined as an iso-surface of CAT scan data. The Femur model has a bounding box of 90.1 x 75.7 x 225.6 mm^3 and a bounding box diameter of 254.4 mm. We simplify the Femur to 26.8 K triangles and 13.4 K vertices with a tolerance of 0.5 mm (0.2 %) in 171.5 seconds, and store the resulting error values at the vertices. Starting with this simplified model and associated error volume, we simplify to 9,592 triangles and 4,794 vertices with a tolerance of 1 mm (0.39 %) in 28.5 seconds. Starting with this simplified model and associated error volume, we simplify to 4,618 triangles and 2,307 vertices with a tolerance of 1.6 mm, or 4 pixels of the original CAT scan (0.63 %), in 10.6 seconds. Finally, starting with this simplified model and associated error, we simplify to 3,124 triangles and 1,560 vertices with a tolerance of 2 mm (0.78 %) in 5.4 seconds.

We visualize the models using flat shading in Figs. 26A–J. The error volumes associated with the models are represented by coloring each vertex according to its error value using the colormap of Fig. 26B, and by using Gouraud shading for interpolating colors inside triangles. Both a model and the associated error volume are necessary to perform the next simplification.

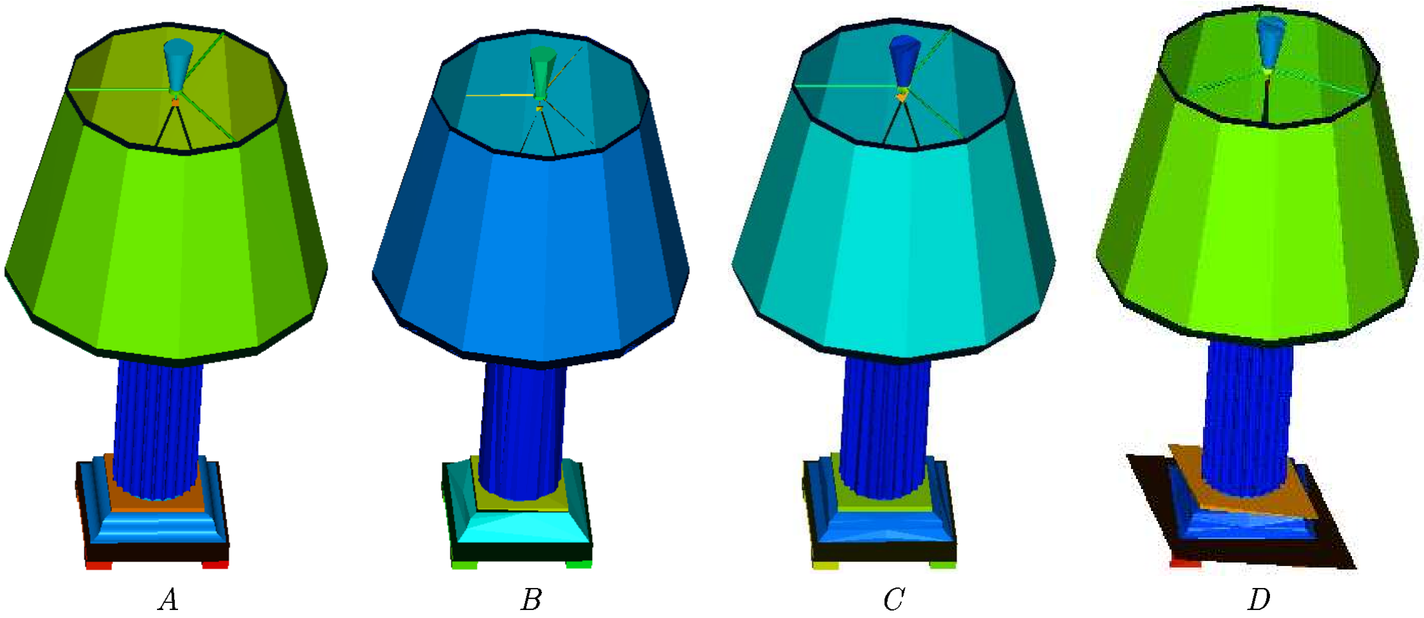

Figure 25: A: Lamp model with 5,052 triangles, flat shaded; Rainbow colors are assigned to surface components by order of vertex count. B: Simplified model (686 triangles) with maximum error of 1% of the bounding box diameter. B: Simplified model (2,035 triangles) with volume preservation. D: Result of a method that does not bound the distance from simplified to original.

12.4. Arteries Model

Starting with a model representing a network of arteries extracted from a MRI scan, with 56.5 K triangles and 28.5 K vertices, and with a bounding box diameter of 127.2 mm, we apply a first simplification step that reduces the triangle count to 5,633 and the vertex count to 2,846. This simplification is performed in 58 seconds, and corresponds to an error tolerance of 1 mm, which is 0.8% of the bounding box diameter. Then, starting with the simplified model, we simplify it down to 2,571 triangles and 1,303 vertices in 6.5 seconds with an error tolerance of 1 mm again, corresponding to a tolerance of 2 mm (1.6%) with respect to the original.

12.5. Jet Engine Noise Model

A height field is constructed by simulation of the noise level generated by a jet engine: we simplify 660K triangles down to 6.7 K in 583 seconds with a maximum error of approximately 4% of the surface largest dimension. The result is shown in Fig. 28.

12.6. Humidity Model

We simplify a terrain model of the Atlanta area with associated vertex colors representing relative humidity values. The original model, visualized in Fig. 29, contains 309 K triangles and 155 K

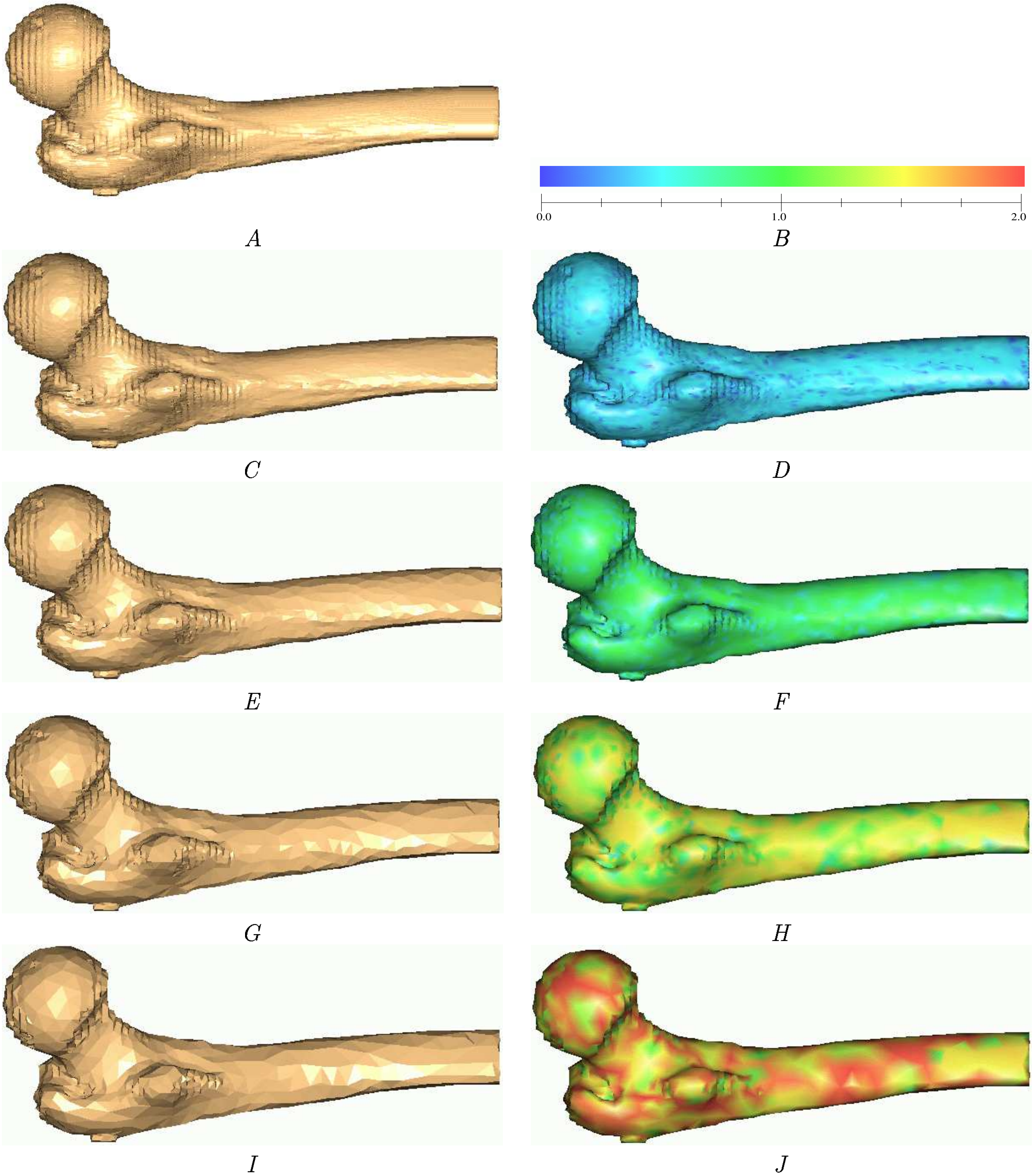

Figure 26: A: Femur Model (181 K triangles). B: error volume colormap. C: simplification with 0.5 mm tolerance (26.8 K triangles) D: error volume of C. E, F: simplification with 1.0 mm tolerance (9,592 triangles). G, H: 1.6 mm tolerance (4,618 triangles). I, J: 2 mm tolerance (3,124 triangles).

A B C

Figure 27: A: Carotid Arteries (57 K triangles). B: Simplification (5.6 K triangles) with a maximum error of 0.8%. C: Superimposition of A and B.

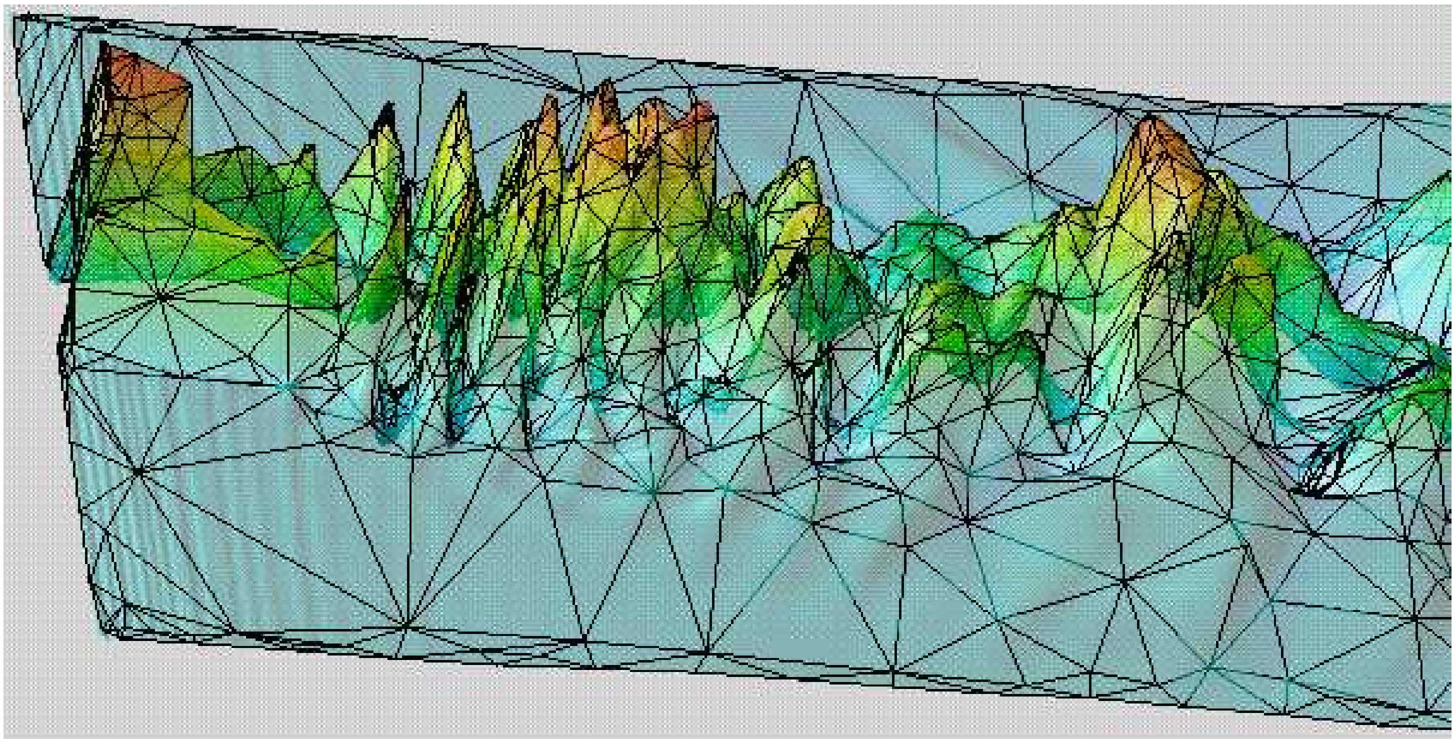

Figure 28: Engine Noise level Data: simplified from 660 K triangles down to 6.7 K, with an error tolerance of 4% of the surface largest dimension.

vertices. We process the model in 361 seconds, resulting in 88.5 K triangles and 44.6 K vertices. This simplification, shown in Fig. 29 does not attempt to preserve the original colors. A simplification implementing the color preservation of Section 9 operates in 428 seconds, resulting in 92.4 K triangles and 46.6 K vertices: we show the result in Fig. 31. In RGB space, we bound the maximum discrepancy with 0.4, wherein $0 \leq R, G, B \leq 1$. The colors of the bottom right of the terrain are more faithfully preserved than in Fig. 30: In this area where the surface (here the Atlantic ocean) is flat, the colors vary more rapidly than the geometry, and the method of Section 8 fails to preserve them.

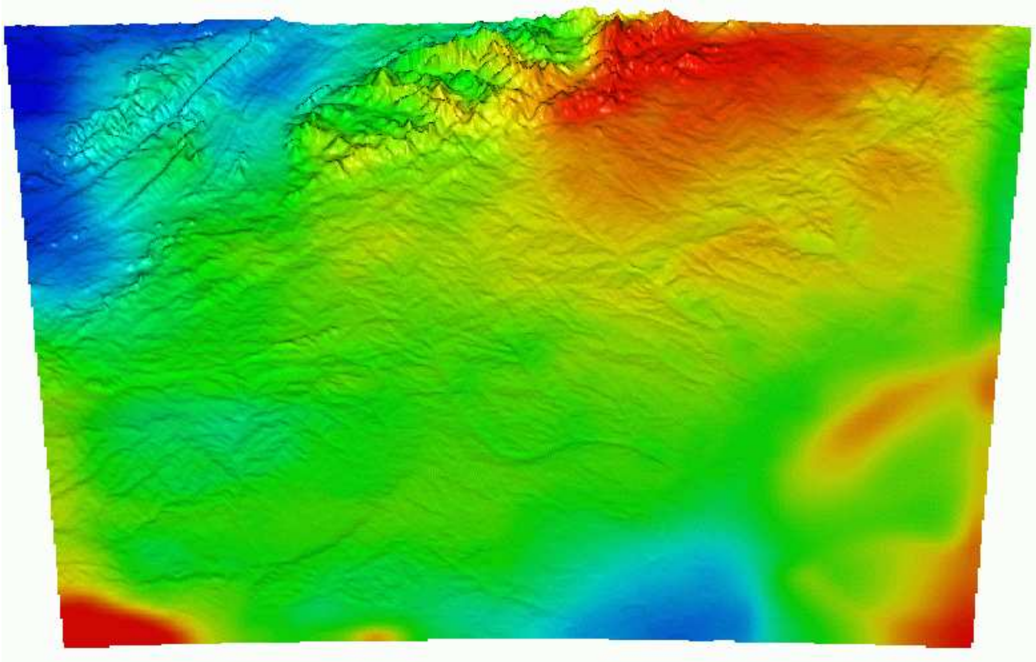

Figure 29: Terrain model of the region of Atlanta with humidity values color coded using a Rainbow colormap (309 K triangles).

12.7. Bunny Model

We consider the Bunny model consisting of 69,473 triangles and 34,835 vertices, with a bounding box of 0.1557 x 0.1543 x 0.1207, corresponding to a bounding box diameter of 0.2502. We simplify the Bunny with a tolerance of 1 / 32 % of the bounding box diameter ($7.82\ 10^{-5}$), and save the resulting error volume. Starting with the simplified model, we perform a subsequent simplification with a tolerance of 1 / 16 % of the bounding box diameter. We apply a succession of simplifications, finishing with a tolerance of 2 % ($5.00\ 10^{-3}$). Timings and resulting numbers of triangles are reported in Table 2. When comparing with the results reported in [14] for the same models, assuming that the same tolerances are applied, we observe that the method of [14] reduces the models slightly more, for a fivefold to tenfold increase in processing time. Note that our method preserves volume. This may be why more triangles are necessary in our simplifications.

12.8. Phone Model

The Phone model contains 165,963 triangles and 83,044 vertices and has a bounding box of 0.1745 x 0.06852 x 0.1597 resulting in a bounding box diameter of 0.2462. We perform successive simplifications on the Phone model as detailed in Table 2. Resulting surfaces are shown in Fig. 33. The observations made for the Bunny model apply to the Phone model as well.

For convenience we denote by 1% Model the surface obtained with a tolerance of 1% of the bounding box diameter. In Fig. 34 we compare a histogram of distances from vertices of the original

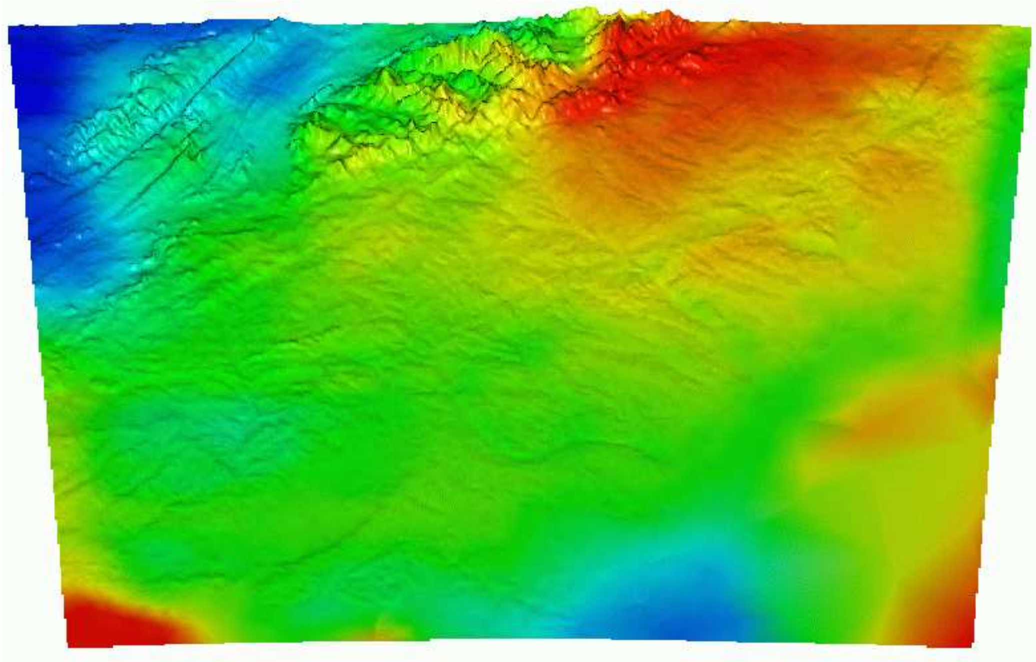

Figure 30: Simplified terrain without color preservation (89 K triangles).

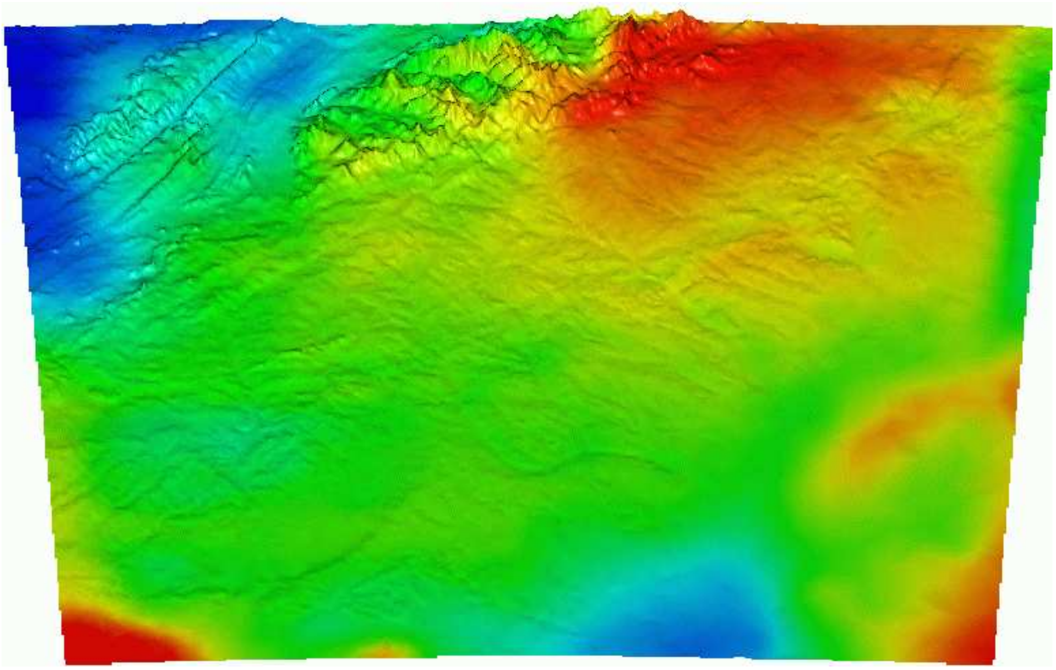

Figure 31: Simplified terrain with color preservation (92 K triangles).

model to the 1% Model to a histogram of error values of the 1% Model. We observe that the highest errors are twice as large as the largest distances: t is a partial account of all the distances that could be

Bunny				Phone		
Tolerance	Number of Triangles	Percentage of Original	CPU time	Number of Triangles	Percentage of Original	CPU time
0	69,473	100%	0	165,963	100%	0
1 / 64				95,839	58%	209
1 / 32	42,894	62%	85	52,584	32%	121
1 / 16	26,528	38%	53	24,102	14.5%	64
1 / 8	14,938	21%	32	10,412	6,3%	28
1 / 4	7,829	11%	17	4,618	2.78%	12
.4	4,835	6.9%	9.2	2,587	1.56%	5.6
.5	3,780	5.4%	5.8	1,943	1.17%	3.2
.6	3,070	4.4%	4.5	1,479	0.89%	2.3
.7	2,541	3.6%	3.6	1,198	0.72%	1.7
.8	2,159	3.1%	3.0	986	0.59%	1.5
1	1,676	2.4%	2.5	743	0.44%	1.36
1.5	1,050	1.5%	1.9			
2	766	1.1%	1.1			

Table 2: Bunny and Phone models: subsequent simplifications with increasing tolerances. The number of triangles, reduction factor and CPU time in seconds are reported.

computed; the distances from points (not only vertices) of the simplified surface to the original surface are missing. Also, for the Phone model, each vertex of the 1% Model is the product of an average of 220 edge collapses. In other words, each final error value was recomputed an average of 220 times. In these conditions, we believe that our error estimates are very economical.

13. Conclusion

We have described the error and tolerance volumes, and applied them to the simplification of triangulated surfaces using edge collapses. The linear to sub-quadratic computational complexity of the method justifies its application to a large class of surfaces. Using the Subdivision Method for computing error values, and avoiding redundant computations of triangle normals, areas and tetrahedra volumes as explained in various sections of this article, surfaces containing half a million triangles and more are processed in a few minutes on a Unix workstation.

Error and tolerance volumes can be easily adapted to other simplification algorithms, for instance using vertex collapses, using the Subdivision Method of Section 8; in fact, the simplification of a Type II boundary edges of Section 10 is a particular case of vertex collapse. One advantage of the edge collapse is the freedom in positioning the simplified vertex, which allows volume preservation, or boundary length preservation. One disadvantage may be a higher cost of computing the error values.

We have described three methods for computing the error volume. We have found that the Subdivision Method, used with the objective of minimizing the error value at the simplified vertex, produced the best speed and reduction factor combination. The tolerance volume can be updated using a simple rule for propagating the lowest value; alternatively, it can be updated using constraint collection and minimization, the constraints being collected at the same location used for the error volume, and using the same barycentric coordinates.

Using the error volume, we can compute new simplifications of the resulting surface, while still measuring error bounds with respect to the original surface. At no additional cost, we can report a bound to the simplification error corresponding to a particular vertex or region, as opposed to a global error bound. One single floating point number per surface vertex is necessary. In particular, if a region was not simplified at all, it will be assigned an error value of zero as opposed to the same error bound as elsewhere.

In Section 9 the simplification of attributes attached to surface vertices is a very general method, that applies whenever linear interpolation is used between surface vertices. We have extended this method to vertex normals, by providing a bound to the maximum discrepancy.

The current method maintains the surface topology. This is a requirement for some applications in Medical Imaging and Scientific Visualization (consider the Arteries data set, formed with a series of thin tubes). For some Computer Graphics applications, this can be considered a limitation. One alternative is to change the topology and to use textures or other techniques to simulate the original topology. A new definition of the error volume will be necessary, whereby a single sphere may intersect several original surface patches, or no patch at all.

The method operates on a manifold surface, tolerating some self-intersections, and returns a manifold surface. In a subsequent publication, we will explain how non-manifold input can be processed.

As noted by other researchers, the forest data structure built during simplification gives access to all intermediate simplification levels, in addition to the final simplified surface. Methods for exploiting this information will be described in a subsequent paper.

14. Acknowledgements

I wish to thank all persons who exchanged ideas with me on this topic and provided extremely useful feedback: Alan Kalvin, Robert Hummel, Francis Lazarus, Jay Sbrollini, Russel Taylor, Frank Tip, Richard Hammer, and especially, Gabriel Taubin. The Bunny and Phone models were provided by Mark Levoy, the Buddha by Brian Curless, and the Atlanta Humidity by Lloyd Treinish.

A. Computing the Volume of a Solid

We compute the volume enclosed by a solid by choosing an origin, and by adding the contribution of the volumes of individual tetrahedra using the same formula as Eqn. 6.1, where the summation is performed over all surface triangles; in practice, the origin of the coordinate system was used. We have identified two difficulties associated with this computation. Firstly, we must add a multitude of small numbers to form a large number: the Femur contains 180,854 triangles; once the sum uses all the allotted precision digits, subsequent increments are incorrectly added. One solution is to perform the addition in higher precision that is necessary for the result. For instance, the results previously quoted were obtained using double precision for summing. Another solution that we implemented, is to split the summation in the following binary fashion: We place the individual tetrahedra volume values in an array; we sum consecutive values in pairs and place the sums in a new array of half the size of the original array (in practice, we overwrite the array); if the number of values is odd, the last value is added to the last sum; we repeat the operation on the newly formed array, until the array is of size one, whereby the first element is the final sum. Using the Femur data, a straightforward sum of tetrahedra volumes in simple precision gives 233,462.8593 mm^3, whereas the binary summation gives 233,462.7500 mm^3, which is closer to the result of double precision computation.

Secondly, we wish to accurately evaluate the volume of an individual tetrahedron. With an example, we show the shortcomings of computing determinants of 3x3 matrices as in Eqn. 6.1 using a straightforward approach, and we develop a more stable method. The straightforward approach (Method I) for computing a determinant of $\mathrm{det}(\mathbf{u}, \mathbf{v}, \mathbf{w})$ is to compute the cross product of $\mathbf{u}$ and $\mathbf{v}$ and take the scalar product of the result with $\mathbf{w}$.

Methods II and III consist of orthogonalizing the 3x3 matrix; we perform a QR decomposition of the matrix and we compute the determinant of the resulting upper triangular matrix R. Note that the same QR decomposition is used throughout the algorithm to compute the normal of a triangle; we store the triangle normal as well as the triangle area for subsequent uses by the algorithm. We consider the oriented vector basis $(\mathbf{x_1}, \mathbf{x_2})$ formed with two consecutive oriented sides of the triangle defined with the 3x3 matrix. (A tetrahedron is implicitely defined using the origin $(0, 0, 0)$ as a fourth vertex.) Method II uses the two longest triangle sides for the basis $(\mathbf{x_1}, \mathbf{x_2})$. Method III uses both the shortest and the longest sides. Both Methods II and III continue by performing a QR decomposition of $(\mathbf{x_1}, \mathbf{x_2})$ and by returning the determinant of the resulting 2x2 triangular matrix, which is twice the triangle area. The normalized cross product of $(\mathbf{x_1}, \mathbf{x_2})$, or triangle normal, is also returned for subsequent use. We multiply the triangle area with a third of the scalar product between the first triangle vertex and the triangle normal, and obtain the tetrahedron volume, which is one sixth of the matrix determinant. Consider the following determinant $\Delta(a)$:

$$\Delta(a) = 10^{-a} = \begin{vmatrix} 0 & 10^{a} & 10^{-a} \\ 10^{-a} & 0 & 10^{-a} \\ 10^{a} & 10^{a} & 10^{a} \end{vmatrix}$$

For various values of a we compare in Table 3 the values of the determinant obtained with the three methods. In each case, the computation was performed in double precision. Method III provides the exact result for values of a below 16, clearly outperforming the other two methods on this input. One interesting lesson is that Method II, though sophisticated, performs worse than the simple Method I.

B. Projection Method for Error Volume Computation

In Step I, we consider in turn each triangle t_j of $(\mathbf{v_1}, \mathbf{v_2})^\star$. The triangle t_j has a tensor of vertex coordinates $\bar{\mathbf{v}}_\mathbf{j}$, error values $\epsilon_\mathbf{j}$ as well as a normal $\mathbf{n_j}$ and it is supported by a plane P_j of known equation. We then project orthogonally on P_j all the vertices of $\mathbf{v_0}^\star$, recording their barycentric coordinates with respect to t_j and their positive or negative height above or below the plane P_j. We

value of a	6	7	8	9	15	16
Method I	1.000008e-06	1.005828e-07	1.490116e-08	0.0	0.0	0.0
Method II	1.000089e-06	1.021405e-07	NaN	NaN	NaN	NaN
Method III	1.000000e-06	1.000000e-07	1.000000e-08	1.0e-09	1.0e-15	0.0

Table 3: Values of the determinant $\Delta(a)$ computed using Methods I, II and III for several values of a. The symbols NaN mean that a floating point exception occurred.

call π this projection. The barycentric coordinates and heights are computed as set forth above. In practice, we solve the linear system in Eqn. 8.3 simultaneously for all vertices of $\mathbf{v_0}^\star$ using multiple right hand sides.

For each vertex $\mathbf{v_i}$ of $\mathbf{v_0}^\star$ we call $\mathbf{w_i}$ its projection on P_j using π. Accordingly, the projection of the simplified vertex is denoted $\mathbf{w_0}$. We note $\mathbf{w_0}^\star$ the projection of the star, and $\ell(\mathbf{w_0})$ the projection of the link of $\mathbf{v_0}$. We have identified three types of constraints that can be collected: Edge Constraints, Corner Constraints, and Vertex Constraints. The three types of constraints are illustrated in Fig. 35.

Edge Constraint. For each edge of $\mathbf{w_0}^\star$, we determine if there is a change of sign of any barycentric coordinate between the two vertices of the edge. If there is a change of sign, the point $\mathbf{w_h}$ wherein the barycentric coordinate is zero is determined. This point is called a *potential edge crossing*. For explanatory purposes, consider the case illustrated in Fig.35, where the edge tested is $(\mathbf{w_0}, \mathbf{w_3})$. Since $\mathbf{w_3}$ is a vertex of the triangle $t_j = t_2$, after permutation, its associated vector of barycentric coordinates $\alpha_\mathbf{3}$ is $\alpha_\mathbf{3} = (1, 0, 0)^T$. The first barycentric coordinate changes sign between $\mathbf{w_0}$ and $\mathbf{w_3}$. We note $\alpha_\mathbf{0}$ the barycentric coordinates associated with $\mathbf{w_0}$; we have: $\alpha_{00} < 0, \alpha_{30} = 1$. We introduce λ defined as:

$$\lambda = -\frac{\alpha_{30}}{\alpha_{30} - \alpha_{00}}$$

Next, the barycentric coordinates of $\mathbf{v_c}$, denoted $\alpha_\mathbf{c}$ are computed as follows:

$$\alpha_\mathbf{c} = \lambda \alpha_\mathbf{0} + (1 - \lambda)\alpha_\mathbf{3}$$

If all three coordinates of $\alpha_\mathbf{c}$ are greater than or equal to zero the potential crossing $\mathbf{v_c}$ is determined to be a edge crossing, meaning that $\mathbf{v_c}$ projects on the boundary of t_j (it can't project inside since one barycentric coordinate is equal to zero). A crossing determines an Edge Constraint as follows: The coordinates of $\mathbf{v_c}$ are computed using: $\mathbf{v_c} = \lambda \mathbf{v_0} + (1 - \lambda)\mathbf{v_3}$. Then, the height h_c of $\mathbf{v_c}$ above or below the plane P_j is computed as: $h_c = \lambda h_0 + (1 - \lambda) h_3$. The error ϵ_c at $\mathbf{v_c}$ is defined to be: $\epsilon_c = \epsilon_\mathbf{j} \cdot \alpha_\mathbf{c}$. And finally a constraint on the new error values $\{\delta_i\}$ is determined as follows:

$$\lambda \delta_0 + (1 - \lambda)\delta_3 \geq |h_c| + \epsilon_c \tag{B.1}$$

For a given edge, there can be several crossings, as illustrated by $\mathbf{v_{c_2}}$ and $\mathbf{v_{c_3}}$ in Fig.35. However, there is at most one crossing per sign change in a particular barycentric coordinate.

We determine whether all edges for which a crossing was identified can be circularly ordered without gaps. If this condition is satisfied, then the process continues; otherwise, the edge is marked as not collapsible and the testing process terminates.

Corner Constraint. A Corner constraint is generated when we identify that one corner of t_j corresponding to either $\mathbf{v_1}$ or $\mathbf{v_2}$ (not belonging to $\ell(\mathbf{v_1}, \mathbf{v_2})$) is inside one of the triangles of $\mathbf{w_0}^\star$.

To determine a constraint for the corner $\mathbf{v_1}$, we loop serially through the triangles of $\mathbf{w_0}^\star$ and identify the first triangle containing $\mathbf{v_1}$. Verifying that a triangle q of $\mathbf{w_0}^\star$ contains $\mathbf{v_1}$ is done as follows. After permutation of the barycentric coordinates, the last two coordinates of $\mathbf{v_1}$ are (0,0). We note $((x_0, y_0), (x_1, y_1), (x_2, y_2))$ the last two barycentric coordinates of the vertices of s. For a given point $\mathbf{u}$ of barycentric coordinates (x, y), it is inside q if and only if the areas of the three triangles joining $\mathbf{u}$ and the oriented edges of q are positive. Using the determinant formulae for triangle areas of Section 6, we establish that the three following equations must be satisfied:

$$\begin{cases} x_1 y_2 - y_1 x_2 \geq 0 \\ x_2 y_0 - y_2 x_0 \geq 0 \\ x_0 y_1 - y_0 x_1 \geq 0 \end{cases}$$

If q_k contains $\mathbf{v_1}$, we compute the closest point $\mathbf{v}$ from $\mathbf{v_1}$ on the triangle s_k of $\mathbf{v_0}\star$, such that s_k is the inverse image of q_k by the projection π. We record the barycentric coordinates β of $\mathbf{v}$ with respect to s_k as well as the distance d from $\mathbf{v_1}$ to $\mathbf{v}$. The associated Corner Constraint is:

$$\delta_{\mathbf{k}} \cdot \beta \geq d + \epsilon_1, \tag{B.2}$$

where ϵ_1 is the error associated with $\mathbf{v_1}$ and $\delta_{\mathbf{k}}$ are the new errors associated with the triangle s_k. We require that both $\mathbf{v_1}$ and $\mathbf{v_2}$ have at least one associated Corner Constraints: we use the constraint corresponding to the lowest value of d and discard other constraints.

Vertex Constraint. A Vertex Constraint is collected when $\mathbf{w_0}$ is inside the triangle t_j; this occurs if the three barycentric coordinates $\alpha_{\mathbf{0}}$ of $\mathbf{w_0}$ are positive. The corresponding constraint is:

$$\delta_0 \geq |h_0| + \epsilon_{\mathbf{j}} \cdot \alpha_{\mathbf{0}}, \tag{B.3}$$

where $\epsilon_{\mathbf{j}}$ represent the errors associated with t_j and h_0 the height of $\mathbf{w_0}$ above P_j.

The Projection method requires that the triangle t_j be contained in $\mathbf{w_0}^\star$. We are in fact more restrictive, by requiring that $t_j \subset \ell(\mathbf{w_0})$, and by testing sufficient conditions: none of the projected link edges $\ell(\mathbf{w_0})$ should intersect t_j except at the edge or vertex common to both. This is tested while collecting Edge Constraints. Then, either one of the three situations applies: t_j is either entirely inside $\ell(\mathbf{w_0})$ (Case A), or t_j contains $\ell(\mathbf{w_0})$ (Case B), or t_j is entirely outside (Case C: t_j and $\ell(\mathbf{w_0})$ simply touch at a point or vertex). See Fig. 36. To verify whether Case A applies for each t_j, we simply prove that $\mathbf{v_1}$ or $\mathbf{v_2}$ project inside $\ell(\mathbf{w_0})$: as when treating a Corner Constraint, this is performed using barycentric coordinates. We permute the barycentric coordinates such that $\mathbf{v_1}$ has coordinates $(0, 0)$; we omit the third coordinate which is redundant. We choose the vertex $\mathbf{w_i}$ of $\ell(\mathbf{w_0})$ whose distance to the origin, expressed in the two barycentric coordinates, is the smallest. We then consider the next vertex on the loop $\ell(\mathbf{w_0})$, denoted $\mathbf{w_{i+1}}$ for convenience. Then, in barycentric coordinates, we take the cross product: $\overrightarrow{\mathbf{w_i w_{i+1}}} \times \overrightarrow{\mathbf{w_i v_1}} = \overrightarrow{\mathbf{w_i w_{i+1}}} \times \overrightarrow{-\mathbf{w_i}}$ which is positive in Case A.

We have already verified that all edges of $\mathbf{w_0}^\star$ for which constrains were collected are incident to a connected set of triangles T. In addition, we require that the triangles of $\mathbf{w_0}^\star$ where Corner

Constraints were collected be adjacent to T or belong to T. This avoids mapping the triangle t_j to disconnected regions of $\mathbf{w_0}^\star$.

In Step II of the Projection Method, we exchange the roles of the edge star $(\mathbf{v_1}, \mathbf{v_2})^\star$ and vertex star $\mathbf{v_0}^\star$ to build the map p_2: We take in turn each triangle s_k of $\mathbf{v_0}^\star$ and we project the vertices of $(\mathbf{v_1}, \mathbf{v_2})^\star$ onto it. We record the same three types of constraints as before. The algorithm that we first implemented ignored Step II. However, although it seems reasonable to assume that each point of $\mathbf{v_0}^\star$ has an inverse image by p_2 after each triangle of $(\mathbf{v_1}, \mathbf{v_2})^\star$ we can construct configurations of $\mathbf{v_0}^\star$ and $(\mathbf{v_1}, \mathbf{v_2})^\star$ for which this is not true.

C. Confining Self-Intersections

We introduce two techniques to avoid or limit self-intersections of the simplified surface. The first technique specifies suitable tolerance values at the vertices of the original surface S: we assign to each surface vertex $\mathbf{v}$ a tolerance τ such that $B(\mathbf{v}, 2\tau) \bigcap S$ is an immersion of a disk (ignoring the vertex coordinates, the triangles of $B(\mathbf{v}, 2\tau) \bigcap S$ have the topology of a disk, or a single closed boundary); we write $B(\mathbf{v}, 2\tau) \bigcap S \sim B_2$. This is a difficult task, as it corresponds to building an offset surface. We propose the following algorithm, but more efficient methods are possible: for each vertex $\mathbf{v}$, set τ originally to the global simplification tolerance τ_0; compute the distance d to each non-incident triangle t. Suppose this distance verifies $d/2 < \tau$ (or less than the tolerances associated to either vertex of t); if the triangles of $B(\mathbf{v}, d) \bigcap S$ do not have the topology of a disk, we assign the tolerance $d/2$ to $\mathbf{v}$ and to the vertices of t. To verify whether $B(\mathbf{v}, d) \bigcap S \sim B_2$, if τ and the average edge length in S have the same order of magnitude, we can spiral around $\mathbf{v}$ or use Dijsktra's algorithm to collect surface triangles closer than d.

Suppose that there is a self-intersection in the simplified surface Σ; then we can find coincident vertices $\mathbf{w_1}$ and $\mathbf{w_2}$ belonging to different triangles of Σ. $\mathbf{w_1}$ has a correspondent $\mathbf{v_1}$ on the original surface S at a distance less than $\tau_1/2$, and $\mathbf{w_2}$ has a correspondent $\mathbf{v_2}$ on S at a distance less than $\tau_2/2$, with for instance $\tau_1 > \tau_2$; $B(\mathbf{v_1}, \tau_1)$ contains $\mathbf{v_2}$; by construction, $B(\mathbf{v_1}, \tau_1) \bigcap S \sim B_2$; because the present algorithm respects tolerances, $B(\mathbf{v_1}, \tau_1 + \tau_0) \bigcap \Sigma \sim B_2$. We say that the self-intersection is *confined* in $B(\mathbf{v_1}, \tau_1 + \tau_0)$ (see Fig. 37). If we know that the edges of Σ are all larger than $\tau_1 + \tau_0$, there cannot be any self-intersection; otherwise, adjacent triangles (or triangles connected by a few edges) must intersect, which may be forbidden using the next technique.

The second technique consists of forbidding self-intersections within the star of the simplified vertex. We take in turn each triangle t of the star, and we determine whether any link edge e intersects that triangle; we do not test the edges whose endpoints are shared by t; we compute the equation of the plane P supporting the triangle t, and determine whether the edge e intersects the plane P, by evaluating the equation of P at the endpoints $\mathbf{v_1}$ and $\mathbf{v_2}$ of e. If e intersects P, we record the barycentric coordinates λ and $1 - \lambda$ of the intersection with respect to the edge endpoints. We also determine the barycentric coordinates α_1 and α_2 of the edge endpoints with respect to the triangle t, using Eqn. 8.3. We decide that the edge intersects the triangle if and only if $\lambda\alpha_1 + (1 - \lambda)\alpha_2 \geq 0$.

References.

[1] J. O'Rourke. *Computational Geometry in C.* Cambridge University Press, 1994.

[2] W. Schroeder, J. Zarge, and W.E. Lorensen. Decimation of triangular meshes. *Computer Graphics (Proc. Siggraph)*, 26(2):65–70, July 1992.

[3] A. Guéziec. Surface simplification with variable tolerance. In *Second Annual International Symposium on Medical Robotics and Computer Assisted Surgery*, pages 132–139, Baltimore, MD, November 1995.

[4] J.H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[5] F. Schmitt, B. Barsky, and W.-H. Du. An adaptive subdivision method for surface-fitting from sampled data. *Computer Graphics (Proc. Siggraph)*, 20(4):179–188, August 1986.

[6] M. DeHaemer and M. Zida. Simplification of objects rendered by polygonal approximations. *Computer & Graphics*, 15(2):175–184, 1991.

[7] G. Turk. Retiling polygonal surfaces. *Computer Graphics (Proc. Siggraph)*, 26(2):55–64, July 1992.

[8] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering. In B. Falcidieno and T.L. Kunii, editors, *Modeling in Computer Graphics*, pages 455–465. Springer-Verlag, 1993.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *Siggraph*, pages 19–25, Anaheim, July 1993. ACM.

[10] P. Hinker and C. Hansen. Geometric optimization. In *Visualization 93*, pages 189–195. IEEE, 1993.

[11] B. Hamann. A data reduction scheme for triangulated surfaces. *Computer Aided Geometric Design*, 11(2):197–214, 1994.

[12] A.D. Kalvin and R.H. Taylor. Superfaces: Polygonal mesh simplificaton with bounded error. *Computer Graphics and Applications*, 16(3):64–77, May 1996.

[13] A. Varshney. *Hierarchical Geometric Approximations*. PhD thesis, University of North Carolina at Chapel Hill, 1994.

[14] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *Siggraph*, pages 119–128. ACM, Addison Wesley, August 1996.

[15] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *Computer Graphics Forum*, 15(3):C67–C76, 1996. Proc. Eurographics'96.

[16] R. Crawfis and W. Schroeder. Advanced techniques for scientific visualization, polygon reduction techniques. Course Notes, ACM Siggraph 95, August 1995.

[17] K.L. Clarkson. Algorithms for polytope covering and approximation. In *3rd Workshop on Algorithms and Data Structures*, volume 709 of *Lecture Notes in Computer Science*, pages 246–252. Springer Verlag, 1993.

[18] P.K. Agarwal and S. Suri. Surface approximation and geometric partitions. In *Proceedings of the Fifth Symposium on Discrete Algorithms*, pages 24–33, Arlington, VA, January 1994.

[19] T. DeRose, M. Lounsbery, and J. Warren. Multiresolution analysis of surfaces of arbitrary topological type. Technical Report 93-10-05, University of Washington, 1993.

[20] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsberry, and W. Stuetzle. Multiresolution analysis of arbitrary meshes. In *Siggraph*, pages 173–182, Los Angeles, August 1995. ACM.

[21] P. Schröder and W. Sweldens. Spherical wavelets. In *Siggraph*, pages 161–172, Los Angeles, August 1995. ACM.

[22] M.H. Gross, O.G. Staadt, and R. Gatti. Efficient triangular surface approximations using wavelets and quadtree data structures. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):130–143, June 1996.

[23] A. Certain, J. Popovic, T. DeRose, D. Salesin, and W. Stuetzle. Interactive multiresolution surface viewing. In *Siggraph*, pages 91–98, New Orleans, August 1996.

[24] C. Bajaj and D. Schikore. Error-bounded reduction of triangle meshes with multivariate data. In Georges G. Grinstein and Robert F. Erbacher, editors, *Visual Data Exploration and Analysis III*, volume 2656, pages 34–45. SPIE, 1996.

[25] R. Klein, G. Liebich, and W. Strasser. Mesh reduction with error control. In Yagel and Nielson, editors, *Visualization'96*, pages 311–318. IEEE, October 1996.

[26] T. He, L. Hong, A. Varshney, and S. Wang. Controlled topology simplification. *IEEE Transactions on Visualization and Computer Graphics*, 2(2):171–184, June 1996.

[27] A. Guéziec and R. Hummel. Exploiting triangulated surface extration using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):328–342, December 1995.

[28] H. Hoppe. Progressive meshes. In *Siggraph*, pages 99–108, New Orleans, August 1996. ACM.

[29] P. Lindstrom, D. Koller, W. Ribarsky, L.F. Hodges, N. Faust, and G.A. Turner. Real-time, continuous level of detail rendering of height fields. In *Siggraph*, pages 109–118, New Orleans, August 1996. ACM.

[30] D. Luebke. Hierarchical structures for dynamic polygonal simplification. Technical Report TR96-006, University of North Carolina at Chapel Hill, January 1996.

[31] J.C. Xia and A. Varshney. Dynamic view-dependent simplification for polygonal models. In Yagel and Nielson, editors, *Visualization 96*, pages 327–334. IEEE, October 1996.

[32] L. De Floriani, B. Falcidieno, and C. Pienovi. Delaunay-based representation of surfaces defined over arbitrarily shaped domains. *Computer Vision, Graphics, and Image Processing*, 32:127–140, 1985.

[33] R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *Computer Graphics (Proc. Siggraph)*, 13(2):199–207, 1979.

[34] M. de Berg and K.T.G. Dobringt. On levels of detail in terrains. Technical Report UU-CS-1995-12, Utrecht University, April 1995.

[35] M. Garland and P. Heckbert. Fast polygonal approximation of terrains and height fields. Technical Report CMU-CS-95-181, Carnegie Mellon University, September 1995.

[36] M. Deering. Geometric compression. In *Siggraph*, pages 13–20, Los Angeles, August 1995. ACM.

[37] G. Taubin and J. Rossignac. Geometry compression through topological surgery. Technical Report RC-20340, IBM T.J. Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598, January 1996.

[38] C.M. Hoffmann. *Geometric and Solid Modeling: An Introduction*. Morgan Kaufmann, San Mateo, California, 1989.

[39] B. Hamann. Curvature approximation for triangulated surfaces. In G. Farin, H. Hagen, and H. Noltemeier, editors, *Geometric Modelling*, Computing Suppl. 8, pages 139–153. Springer-

Verlag, New York, 1993.

[40] F. Lazarus and A. Verroust. Décomposition cylindrique de polyèdre et courbe squelette. *Revue de CFAO et d'Informatique Graphique*, 11(4):363–377, 1996.

[41] G. Taubin. A signal processing approach to fair surface design. In *Siggraph*, pages 351–358, Los Angeles, August 1995. ACM.

[42] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. Mac Graw Hill, 1989.

[43] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1993.

[44] J. T. Schwartz and M. Sharir. Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves. *International Journal of Robotics Research*, 6(2):29–44, 1987.

[45] M.T. Goodrich. Efficient piecewise-linear function approximation using the uniform metric. *Discrete Comput Geom*, 14:445–462, 1995.

[46] M.P. do Carmo. *Differential Geometry of Curves and Surface*. Prentice Hall, Englewood Cliffs, New Jersey, 1976.

[47] J.R. Kent, W.E. Carlson, and R.E. Parent. Shape transformation for polyhedral objects. *Computer Graphics (Proc. Siggraph)*, 26(2):47–54, July 1992.

[48] N. Megiddo. Linear-time algorithms for linear programming in $\mathbf{R}^3$ and related problems. In *23rd. Symp. on the Foundations of Computer Science*, pages 329–338. IEEE, 1982.

[49] A. Guéziec. Surface simplification inside a tolerance volume. Technical Report RC 20440, IBM T.J. Watson Research Center, Yorktown Heights, New York, March 1996.

[50] A. Gourdon. Simplification of irregular surface meshes in 3d medical images. In *Computer Vision, Virtual Reality and Robotics in Medicine*, number 905 in Lecture Notes in Computer Science, pages 413–419, Nice, April 1995.

[51] H. Buchegger, R. Sainitzer, and D. Schmalstieg. *LODESTAR: Level of Detail Generator for VRML 1.0*. Technical University Wien, 1996.

[52] M-E. Algorri and F. Schmitt. Mesh simplification. *Computer Graphics Forum*, 15(3):C77–C86, 1996. Proc. Eurographics'96.

[53] F. Aubert and D. Bechmann. Deformations d'objets a volume constant. Technical report, Université Louis Pasteur, Strasbourg, 1995.

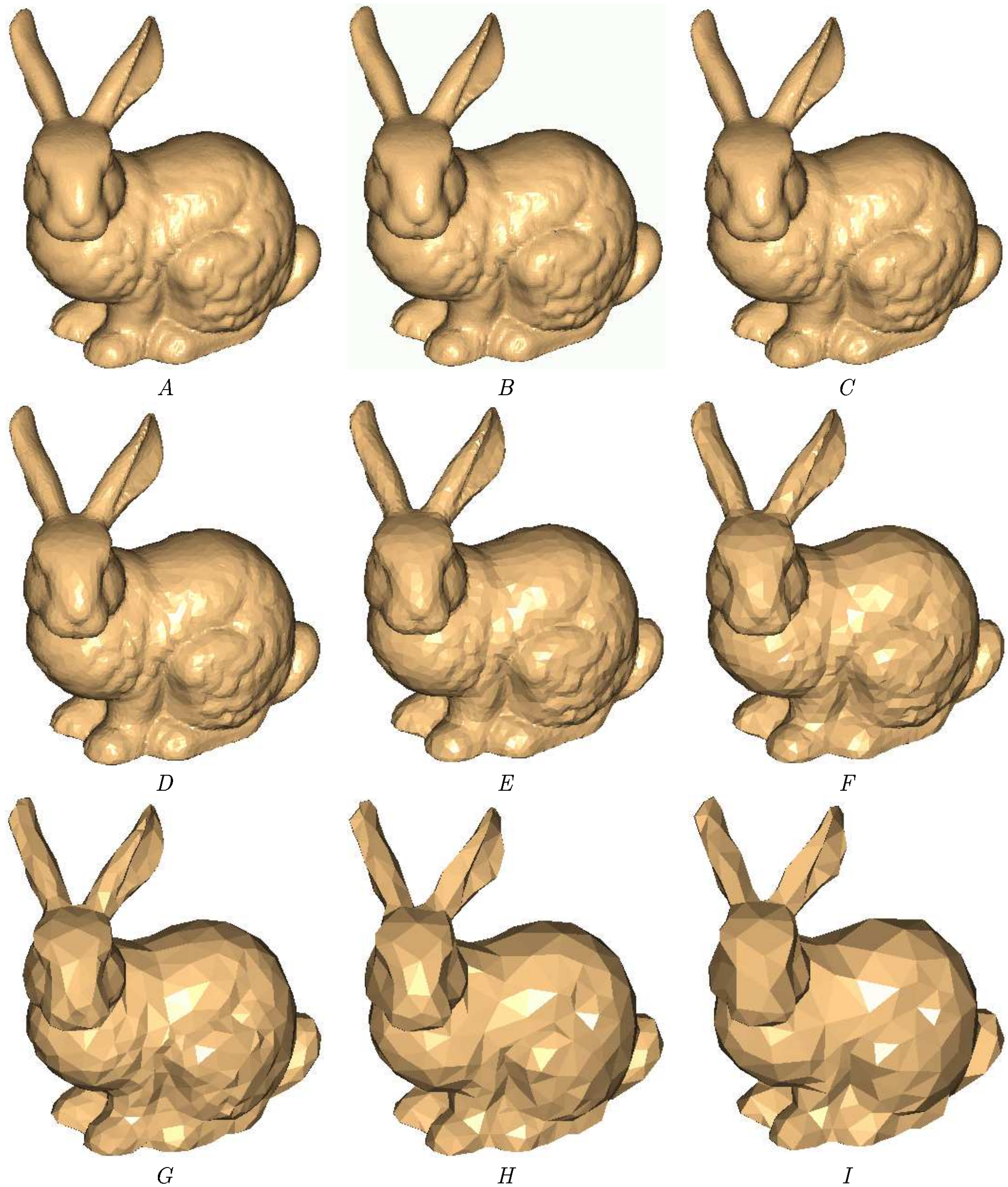

Figure 32: Successive simplifications of the Bunny model, flat shaded. A: original. B: tolerance of 1/32% of bounding box diameter. C: tolerance of 1/16%. D: 1/8%. E: 1/4%. F: 0.5%. G: 1%. H: 1.5%. I: 2%.

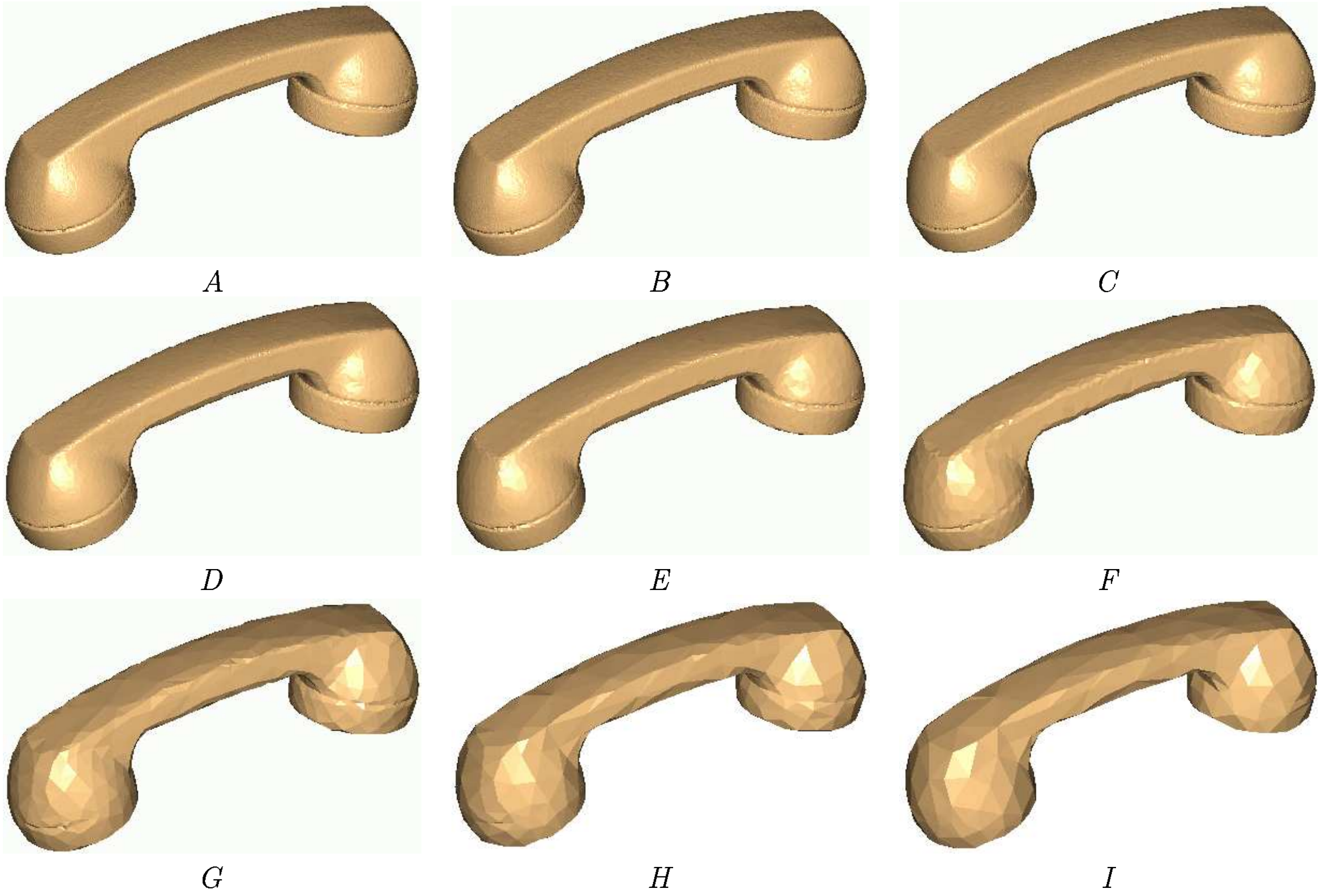

Figure 33: Successive simplifications of the Phone model, flat shaded. A: original. B: tolerance of 1/64% (96 K triangles). C tolerance of 1/32% (53 K triangles). D: 1/16% (24 K triangles). E: 1/8% (10.4 K triangles). F: 1/4% (4,618 triangles). G: 0.5% (1,943 triangles). H: 0.8% (986 triangles). I: 1% (743 triangles).

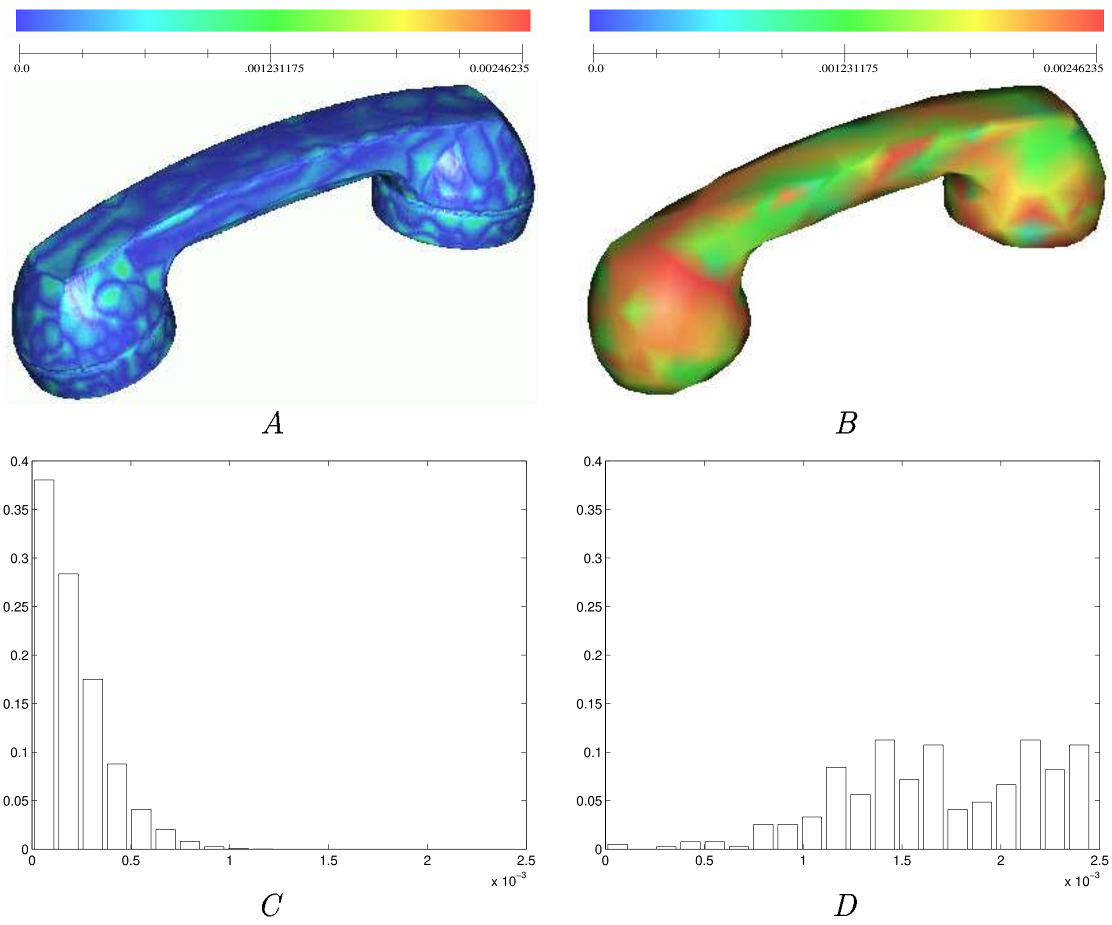

Figure 34: A: visualization of distances from each point of the original Phone to the 1% Model, with the colormap used for A and B. B: visualization of the error volume of the 1% Model. C: histogram of distances measured from points of original Phone to 1% Model. B: histogram of error values on the 1% Model.

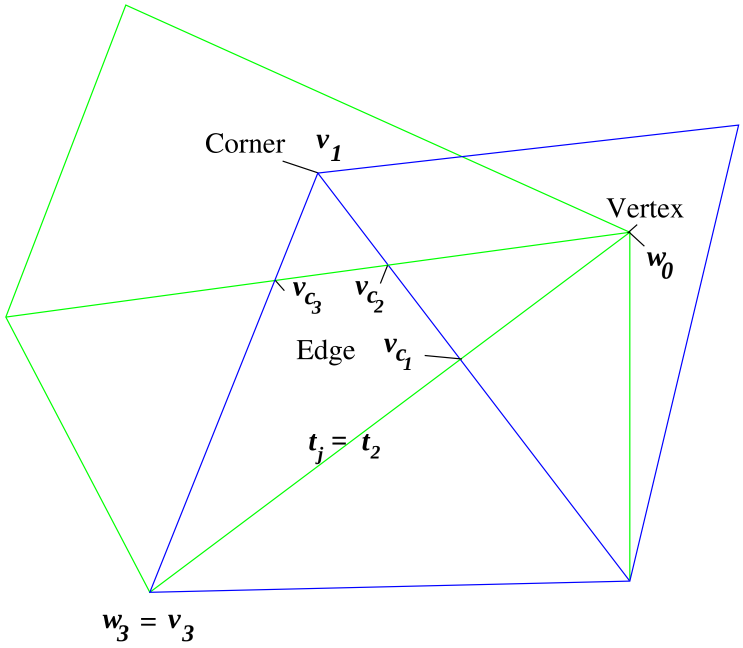

Figure 35: Types of Constraints generated with the Projection Method.

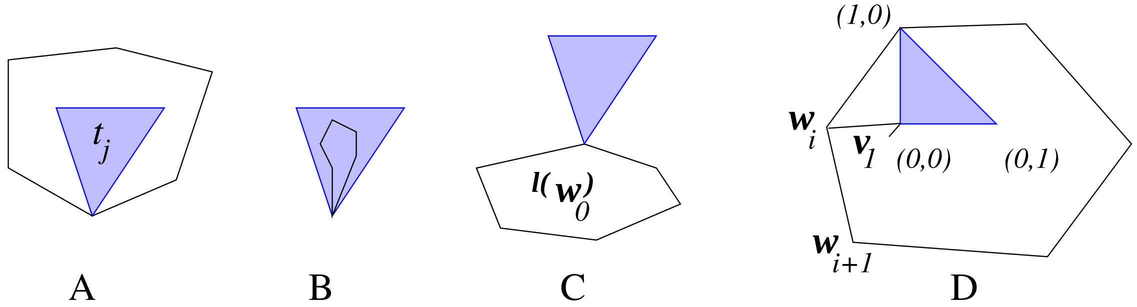

Figure 36: Projection Method: A. The triangle t_j is entirely inside $\ell(\mathbf{w_0})$. B: t_j contains $\ell(\mathbf{w_0})$. C: t_j is entirely outside. D: test establishing whether Case A applies.

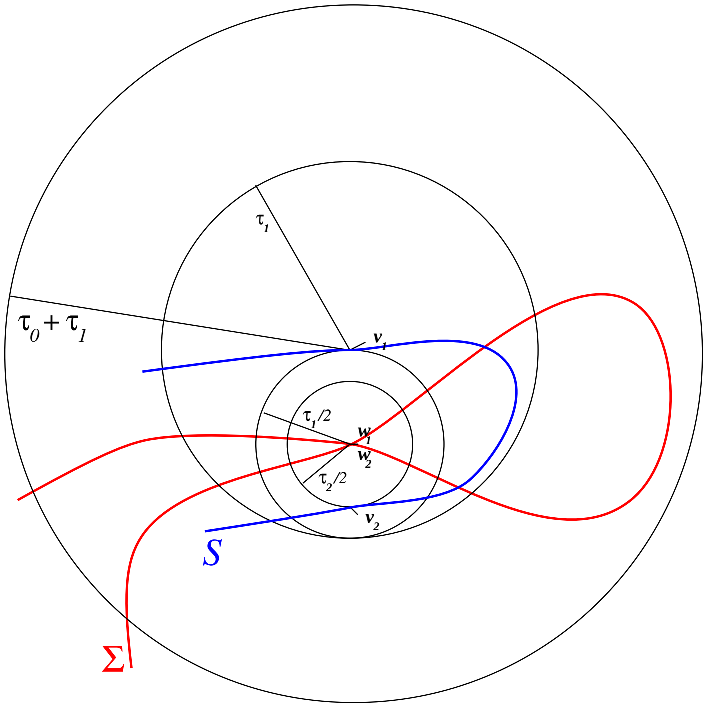

Figure 37: Confined Self-Intersection.