Triangle-Mesh Simplification using Error Polyhedra

Mark Eastlick and Steve Maddock

Department of Computer Science
University of Sheffield
Portobello Road
Sheffield S1 4DP
<m.eastlick, s.maddock>@dcs.shef.ac.uk

Abstract
We present a novel and general framework for simplifying triangle-meshes based on *vertex decimation*, as opposed to the common *edge-collapse* operation. Our basic simplification metric is based on the evaluation of the volume of *error polyhedra*; such a measure is the geometric analogue to the L_1 norm between two functions. Unlike many other approaches, we perform triangulation in three-dimensions and utilise this extra information to increase simplification quality, dynamically making use of the principal curvatures of local biquadratic approximations of the mesh.

Keywords: mesh simplification, vertex decimation, 3D polygon triangulation, discrete differential geometry operators

1 Introduction

The triangle-mesh is the *de facto* representation for three-dimensional models on computers. The triangle is the simplest piecewise-linear tessellating geometric primitive. It naturally evolves in many processes that provide solutions to geometric problems. Algorithms based on triangles are often cleaner than similar examples for surfaces composed of other, more complex, polygons. Furthermore, any such surface subdivision can always be reduced to triangle-mesh form. This, as well as the fact that current graphics hardware caters most thoroughly for triangle-meshes, contributes to the triangle's ubiquity.

Despite its popularity, the triangle-mesh has various shortcomings when compared to other modelling representations, such as parametric patches. A primary limitation is that its piecewise linear nature produces large bandwidth requirements. A triangle-mesh can exhibit only C^0 continuity at best, and therefore curved surfaces must be densely sampled to ensure that all of their nuances are adequately approximated. This has the effect of creating a great many triangles for non-trivial objects. This volume of data is an important bottleneck in many computer graphics applications, and this weak-link can manifest itself as a bandwidth problem over various scales; from graphics device data buses up to permanent storage devices, and even networked systems. Alternatively, it can simply emerge as prolonged local processing of the mesh during, for example, rendering or editing. Mesh simplification provides the backbone for many of the methods that address these problems.

We will begin by reviewing some previous work on mesh simplification and certain related issues. We will then describe a general framework for simplifying meshes to an arbitrary number of vertices. Following this, we will describe the metric used during simplification, making use of the concept of *error polyhedra*. This will lead us to the three-dimensional, curvature-based, triangulation algorithm used during simplification. Finally, we will present and discuss the results of this work.

2 Related Work

There have been various paradigms proposed for the simplification of triangle-meshes. The

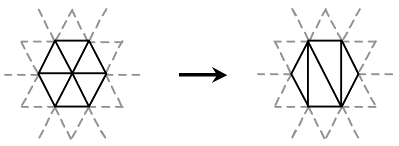

Figure 1 Example of a single vertex removal and retriangulation operation.

edge-collapse and related approaches have emerged as the most popular methods, and we will primarily concentrate on such approaches here. However, the interested reader is referred to [GAR99, EAS01] for a discussion of the full spectrum of techniques.

A popular framework for mesh simplification, introduced in [SCH92], has been termed *vertex-decimation*. This approach performs a sequence of *vertex removal and retriangulation* operations (fig. 1), removing vertices from the mesh and triangulating the resulting hole to maintain the topological type of the mesh.

The original technique operates by performing a number of passes over the mesh. On each pass, all vertices that pass a binary metric are removed. This metric is steadily relaxed until the required resolution is reached. Newer approaches have developed more sophisticated error metrics and triangulation algorithms, and many have adopted the greedy framework of incrementally applying a number of vertex removals operating on the vertex that currently minimises the chosen metric. Approaches based on incremental greedy vertex decimation do not usually optimise the location of the vertices participating in a vertex removal. This leads us to the property that the vertices of a vertex-decimated mesh are usually a subset of the original.

The current most popular methodology for mesh simplification operates by a sequence of iterative *vertex contractions* (fig. 2). The basic method performs a number of *edge-collapse* operations where the resultant vertex is located at either end or the mid-point of the collapsed edge, and the degenerate faces are removed.

The vertex placement strategy of constraining the location to either end of the collapsed edge can be viewed as a special case of the vertex

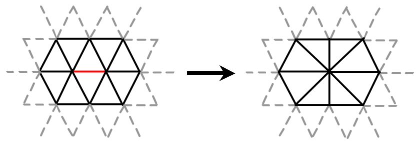

Figure 2 Example of a single vertex contraction operation.

decimation approach. This operation is called the *half edge-collapse*. However, a more powerful approach locates the vertex according to some optimisation criterion, which minimises the "error" of the operation [HOP96, RON96, GAR97, GAR98, LIN98]. This approach removes the subset constraint of vertex-decimated meshes, and can provide better quality simplifications, especially at low numbers of vertices. A further generalisation, introduced in [GAR97], allows vertices that do not share an edge to be merged, modifying the topology of the mesh to provide better quality simplifications especially at low numbers of vertices. Certain methods employ some global, as opposed to local, measure that retains information about the original model during simplification. Such methods also often provide guaranteed error bounds [RON96, HOP96, GAR97, GAR98]. The precursor to many of the current state-of-the-art methods is [RON96]. In this method, each vertex in the original mesh is associated with the set of planes of its incident triangles. During simplification, each child vertex that is a product of an edge-collapse is associated with the set of planes produced from the union of its parent vertex planes. The choice of whether a vertex should be removed is based on the distance to each of the planes in its set. This basic method requires a large amount of computational resources, as during simplification each vertex is associated with ever more planes. An efficient way of calculating the squared sum of these distances for a given vertex was presented in [GAR97], and is termed the quadric metric, and it has provided the backbone for most of the recent work on mesh simplification [GAR98, LIN98].

Interestingly, a recent technique shows that a global error framework is not always required for high quality simplification [LIN98]. This goes somewhat against conventional wisdom, however the results are very convincing. This method is related to the quadric approach.

In general, a triangle-mesh based model will include extra attributes per vertex or face, such as its normal vector or colour. Recent methods for mesh simplification have included such information while performing simplification [COH98, HOP96, GAR98]. Certain methods de-couple the surface colour detail from the actual geometry, and store this data as images, which are then mapped back onto a simplified version of the mesh [COH98]. Pragmatic approaches such as these using image data produce considerably better simplifications at lower vertex counts and may decrease rendering time and storage requirements.

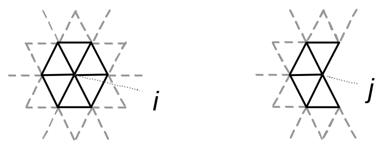

Figure 3 Example of 2-manifold vertex *i* and boundary vertex *j*.

3 Notation and Nomenclature

An *n-simplex* is the convex hull of *n*+1 affinely independent vertices, or 0-simplices. Therefore, an *n*-simplex has *n* dimensions and is composed of *n*+1 (*n*-1)-simplices. For example, a 1-simplex is an edge, which is composed of two 0-simplices, or vertices. In the same way, a 2-simplex is a triangle, which is composed of three 1-simplices, or edges.

A triangular mesh *M* is defined in terms of its constituent vertices and the connectivity

$$N(i) = \{ j \mid \{i, j\} \in K \} \tag{3.1}$$

relationships that exist between them. To this end, we denote a triangular mesh *M* as a pair (*P*, *K*).

$$S(i) = \{ s \mid i \in s, s \in K \} \tag{3.2}$$

The set *P* consists of points $p_i = (x_i, y_i, z_i) \in \mathbf{R}^3$, with $1 \le i \le N$, which correspond to the vertices of the mesh. The set *K* is an *abstract simplicial complex*, which is composed of a number of subsets of {1, …, N}, corresponding to the constituent 0-, 1-, and 2-simplices of the mesh. In other words, the vertices {i}, the edges {*i*, *j*}, and the triangles {*i*, *j*, *k*}. Using this complex *K*, we can construct the graph of the mesh, and with the aid of the vertices *P* we can embed the graph in real 3-space.

Two vertices {*i*} and {*j*} are neighbours if $\{i, j\} \in K$. The *1-ring neighbourhood N*(*i*) of a vertex *i* is the set of 0-simplices.

The *star S*(*i*) of a vertex *i* is defined as the set of its incident simplices.

We represent the simplices composing an arbitrary valid triangulation of a set of vertices *V* as T(*V*).

A *2-manifold* vertex *i* is one whose star S(*i*) is topologically equivalent, or *homeomorphic*, to a disc (fig. 3). A mesh is called *2-manifold* if each of its vertices has 2-manifold topology.

A *boundary* vertex *j* is one whose star S(*j*) is homeomorphic to a half-disc (fig. 3). A mesh is called *2-manifold with boundary* if each of its vertices has either 2-manifold or boundary local topology.

4 A Topology Preserving Mesh Simplification Framework

For the sake of brevity, we will primarily concern ourselves with *2-manifold with boundary* meshes. We define a *vertex removal* operator VR for a manifold triangle-mesh *M* and vertex *i* as:

$$\mathrm{VR}(M, i) = M' \;\middle|\; \begin{aligned} &P = P' \cup p_i, \\ &K \setminus K' = \mathrm{S}(i) \setminus \mathrm{N}(i), \\ &K' \setminus K = \mathrm{T}(N(i)) \setminus \mathrm{N}(i) \end{aligned} \tag{4.1}$$

The result of this operation is the mesh *M'*, based on mesh *M* with point p_i and simplices S(*i*) removed, and a number of new simplices T(N(*i*)) added to maintain the topological type. An application of the operator on a 2-manifold vertex *i* results in the removal of |N(*i*)| triangles and edges and the addition of |N(*i*)| – 2 triangles and |N(*i*)| – 3 edges. Similarly, for a boundary vertex *j*, an application of the operator results in the removal of |N(*j*)| – 1 triangles and |N(*j*)| edges, and the addition of |N(*i*)| – 2 triangles and |N(*i*)| – 3 edges. The operator is not defined for vertices whose local topology is not manifold with boundary. Figure 1 demonstrates the VR operator on a 2-manifold vertex *i*.

We can define an *error function*, or *metric*, EVR with a domain of all possible valid combinations of *M* and *i*, and a real number range according to the error, or change, between *M* and *M'* as a result of VR(*M*, *i*) as:

$$\mathrm{EVR}(M, i) \to \mathbf{R} \tag{4.2}$$

We can define another vertex removal operator, OVR, for mesh *M*, as:

$$\mathrm{OVR}(M) = M' = \mathrm{VR}(M, \arg\min_{\{i\} \in K} \mathrm{EVR}(M, i)) \tag{4.3}$$

The result of this operator for an *N* vertex mesh *M* is an *N* – 1 vertex approximation *M'* that minimises the error metric chosen for VR. For a given *M*, we can define a sequence of *n* meshes with increasing error produced by recursive applications of the OVR operator:

$$\begin{aligned} &\mathrm{OVR}(...\mathrm{OVR}(\mathrm{OVR}(M) \to M^1)) \to M^2)...) \\ &\quad = M^n \end{aligned} \tag{4.4}$$

This sequence produces a multiresolution representation, or *history*, of *M*, containing *n* different resolutions, ranging from the original high-resolution mesh down to a base low-resolution mesh at which point any further applications of the OVR operator will violate the mesh topology.

In practice, for a given mesh *M*, we initially construct a priority-queue containing every $\{i\} \in$

K, ordered by increasing EVR(*M*, *i*). Therefore removing the head vertex will give us OVR(*M*). After we perform an application of OVR(*M*), we need to recalculate the EVR(*M*, *i*) for all vertices whose incident triangles have changed as a result of the operation, and update their positions in the priority-queue. These vertices are simply the neighbourhood vertices *N*(*i*). This is an important reason for the efficiency of this approach to simplification.

5 Calculation of Error Metric using Error Polyhedra

The various discrete L_n norms and straightforward extensions to higher numbers of dimensions are commonly used as error metrics in other signal-processing disciplines, such as lossy sound, image, and video compression. A norm of the difference between the domain and range signals is calculated and used as a measure for the quality of the approximation given by the range signal. However, the use of such measures inherently relies on the signal being regularly sampled, be it in one, two, or three dimensions. We do not have such a luxury for the data we are considering here, and this is an important reason why mesh-processing techniques are often more challenging and are possibly relatively less evolved than their older siblings.

The L_n norms for a continuous signal *f* are defined as:

$$\|f\|_{L^n} = \left(\int |f(x)|^n dx\right)^{1/n} \qquad (5.3)$$

Therefore, we can see the L_n norm between two discrete signals *f* and *g* can be calculated using the following expression:

$$\|f - g\|_{L^n} = \left(\sum_x |f(x) - g(x)|^n\right)^{1/n} \qquad (5.2)$$

The L_∞ measure is a special case, and measures the magnitude of the maximum deviation between the two signals. This function is called the *Hausdorff error* and is often classed as the "strongest" metric. However, its exact computation is prohibitively complex for arbitrary pairs of non-trivial meshes. Therefore, it is desirable to find suitable alternatives for practical simplification algorithms.

A geometric analogue to the L_1 norm between a mesh *M* and the mesh produced by a vertex removal VR(*M*, *i*) can be seen to be the difference in volume of the meshes over the 1-ring neighbourhood of *i*. We call this closed region an *error polyhedron*, and we set the range of VR(*M*, *i*) to be the magnitude of this volume. Therefore, an application of the OVR operator results in the mesh with one fewer vertices that minimises the local change in volume.

We use the following fourth-order determinant for the volume of a 3-simplex as the basic building block for the calculation of error polyhedron volume. The simplex *s* is composed of vertices p_i, p_j, p_k, and p_l.

$$\mathrm{VOL}^3(s) = \frac{1}{6}\begin{vmatrix} x_i & y_i & z_i & 1 \\ x_j & y_j & z_j & 1 \\ x_k & y_k & z_k & 1 \\ x_l & y_l & z_l & 1 \end{vmatrix} \qquad (5.3)$$

The sign of $\mathrm{VOL}^3(s)$ is equal to the sign of the inner product of the vector from p_l to the barycentre of $\{p_i, p_j, p_k\}$ and the orientated normal vector of $\{p_i, p_j, p_k\}$. In other words, if the orientated normal vector of $\{p_i, p_j, p_k\}$ is pointing away from the half-space defined the plane of $\{p_i, p_j, p_k\}$ containing p_k then the sign of $\mathrm{VOL}^3(s)$ will be positively signed, otherwise, it will be negatively signed.

Using this property, we can generalise this formula to provide the volume of a closed simplicial complex or polyhedron. Given such a simplicial complex *K* and some arbitrary reference point *r*, we can classify its constituent 2-simplices as being part of the "upper" or "lower" surface of *K*, according to whether their orientated normal vectors face away from, or towards, *r*. The projection of a constituent 2-simplex onto *r* is a 3-simplex. Each 3-simplex of the upper surface will have positive volume, and each 3-simplex of the lower surface will have negative volume. We can view the complete projection of the upper and lower surfaces onto *r* as the projections of each of the simplices onto *r*. Notice that the projections of both the upper and lower surface onto the origin have identical cross section. Therefore, we can see that the space defined by the lower surface projection is a subset of that for the upper. The difference between the spaces of the two projections can be seen to be exactly the space of the error polyhedron. Therefore we can measure the volume of the polyhedron by taking the sum of the volume contributions from its upper and lower surface 2-simplex projections, which can be calculated using (5.3). Therefore, in the spirit of (5.2), we can define our error function EVR for a vertex *i* and the *n* 3-simplices defining its corresponding error polyhedron σ_i in the following way, where the *m*th simplex of σ_i is denoted by σ_i^m:

$$\begin{aligned}&\mathrm{EVR}(M, i) = \\ &\qquad \|M - M'\|_{L^1} = \sum_{m=1}^{n} \mathrm{VOL}^3(\sigma_i^{\ m})\end{aligned} \tag{5.4}$$

There are various methods for the computation of (5.4) which are significantly faster than the trivial separate evaluation of the *n* fourth-order determinants. We have developed an efficient alternative approach, full details of which can be found in [EAS01].

6 A Framework for Computing Vertex Neighbourhood Triangulation

6.1 Approaches to Triangulation Computation

Many researchers have taken the approach of transforming the vertex neighbourhood into a plane; 2-dimensional triangulation is then performed in this domain, and the result is transformed back into 3-dimensional space to complete the operation. The justifications for this approach are two-fold. Firstly, manifold with boundary vertex neighbourhood is, topologically, a 2-dimensional entity, and secondly, the literature on 2-dimensional triangulation is relatively evolved, due, in the most part, to the requirements of finite-element (FE) analysis.

This approach has two main disadvantages. Firstly, the properties exhibited by the triangulation in 2-dimensions are either meaningless or do not usually hold when the triangulated vertex neighbourhood is projected back into 3-space. The degree of approximation of such properties in 3-space is related to the curvature of the mesh over the neighbourhood. The second limitation is that an extra dimension of information that could be used to derive a more appropriate triangulation is ignored, which leads to poor quality simplifications.

A solution to these problems can be found by performing triangulation in 3-dimensions [BAR98]. The first problem to be overcome for such an approach is how to define a valid triangulation. Clearly, a triangulation which leaves holes, flips triangle orientation, or otherwise alters the topology of the mesh is undesirable. Our solution to this problem is to use a simple linear-time validity test which is based on the notion of consistent triangle orientation. We attempt to find a point which is in each of the half planes defined by a triangle and its orientated normal vector for each triangle before and after the triangulation. If such a point is found, then we can be sure that we have not changed the orientation, and therefore the topology, of the part of the mesh involved in the operation. In practice, we use the area-weighted mean normal vector of the neighbourhood triangles before the triangulation operation to derive a ray from the focus vertex. This gives us a point "below" the surface designed to maximise the chances of success of this technique. We use a similar equation to (5.3) to check the orientation of a triangle relative to our reference point. We set p_i, p_j, and p_k, as the orientated vertices of the triangle in question, and p_l as the reference point. Therefore, a positive sign signifies a triangle with correct orientation. The limitation of this approach is that it is not guaranteed to find a valid reference point, if one does exist. However, we have found that this method works well in practice, and its efficiency is an attractive property.

6.2 Principal Curvature-based Triangulation

We employ a triangulation algorithm which acknowledges and rectifies the problems with the 2-dimensional transformation approach. To this end, an estimate of the level of local curvature of the mesh is made, and the triangulation criterion is adapted accordingly. For coplanar regions of the mesh, the algorithm reduces to an approximation of the minimum weight triangulation (MWT). However, the triangles in more curved areas are made to align with principal curvature direction approximations according to the relative level of curvature between those two directions. Recall that the principal curvature directions for a point on a surface are the orthogonal directions of least and most curvature at that point. Given the principal curvatures of a surface, many other curvature based measures, such as Gaussian and mean curvature, can be easily derived. A complete introduction to the theory of differential geometry is well beyond the scope of this paper, but the interested reader is referred to [DOC76].

Before presenting the triangulation algorithm itself, we will examine the steps needed to derive principal curvatures for triangle-meshes.

The typical equations from classical differential geometry used to calculate principal curvatures operate on parametric surfaces in $\mathbf{R}^3$ of type C^n, where $n > 1$. This presents two immediate problems for the calculation of such quantities for triangle-meshes. The first difficulty results from its piecewise linear nature, which can offer only C^0 type surfaces, and the second is that a mesh is not defined in terms of a parameterisation. A solution to both of these problems is to derive a parametric polynomial surface over the region of interest, which can then be analysed in the usual ways.

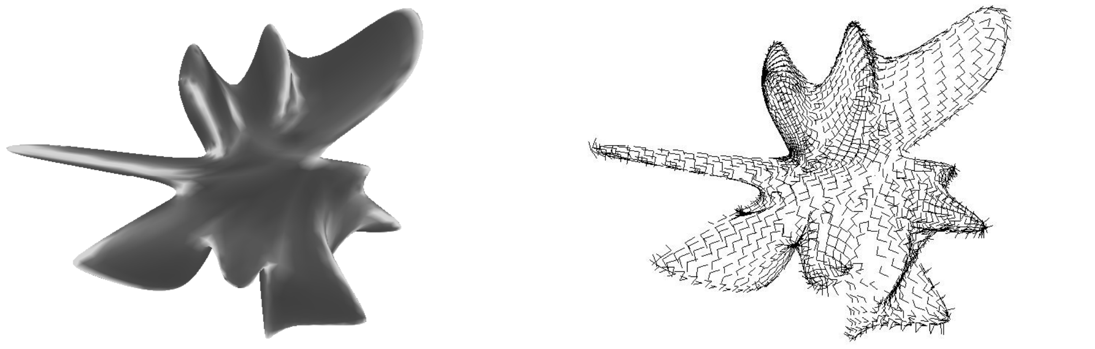
Figure 4 Example of mean curvature and normalised principal curvature direction operators.

We are interested in biquadratic polynomial surfaces of the following form:

$$f(u,v) = \lambda_1 u^2 + \lambda_2 v^2 + \lambda_3 u + \lambda_4 uv + \lambda_5 v \tag{6.1}$$

Using the (u, v) and $f(u, v)$ values, as well as orthogonal basis vectors e_1, e_2, and e_3, we can write (6.1) such that the range is a surface **x** in $\mathbf{R}^3$, as follows:

$$\mathbf{x}(u,v) = f(u,v)e_1 + ue_2 + ve_3 \tag{6.2}$$

This is the appropriate form of a surface for curvature analysis. To calculate the principal curvatures for a point $\mathbf{x}(u, v)$, we need the first and second order partial derivatives of **x** at (u, v). We will now examine the steps needed to derive the surface **x**, and hence its partial derivatives, for a vertex neighbourhood.

A local frame for a vertex i and its neighbourhood N(i) is computed. The origin of the frame is the focus vertex. The vertical basis vector of the frame, e_1, is defined as the mean area-weighted normal vector of the neighbourhood triangles, and the two remaining basis vectors, e_2 and e_3, are derived using Gram-Schimdt orthogonalization. The vector e_1 defines the direction of elevation for the $f(u, v)$ range values, and the vectors e_2 and e_3 define iso-parametric directions for the domain values u and v respectively.

The vertices of the neighbourhood are transformed using an orthogonal projection onto the plane defined by the origin and the vectors e_2 and e_3. The (u, v) coordinates for these points can then be trivially computed. The distance that each point moves as a result of the projection gives us corresponding $f(u, v)$ elevation values.

A system of non-linear simultaneous equations is constructed using (6.1) and the parameter and range values of the projection, with the λ_i as unknown coefficients. Clearly this system can only be solved if there is a minimum of five neighbourhood vertices. However, the system of equations is usually over constrained, and therefore we derive a least-squares error fit for the λ_i using a variant of Newton's method.

Notice the similarity of (6.1) to a two-dimensional second-order Maclaurin series. We can express such a series as follows:

$$f(u,v) = f_u(0,0)u + f_v(0,0)v + \frac{1}{2}\left(f_{uu}(0,0)u^2 + 2f_{uv}(0,0)uv + f_{vv}(0,0)v^2\right) \tag{6.3}$$

Using (6.1) and (6.3), we can see that the following holds true:

$$\begin{aligned}\lambda_1 &= 2f_{uu}(0,0)\\ \lambda_2 &= 2f_{vv}(0,0)\\ \lambda_3 &= f_u(0,0)\\ \lambda_4 &= f_{uv}(0,0)\\ \lambda_5 &= f_v(0,0)\end{aligned} \tag{6.4}$$

Therefore the coefficients λ_i represent each of the partial derivatives of f(0,0) with respect to the parameters u and v. Using (6.2) we can see that the partial derivatives of **x**(0,0) with respect to u and v can be written as follows:

$$\begin{aligned}\mathbf{x}_u(0,0) &= f_u(0,0)e_1 + e_2 = \lambda_3 e_1 + e_2\\ \mathbf{x}_v(0,0) &= f_v(0,0)e_1 + e_3 = \lambda_5 e_1 + e_3\\ \mathbf{x}_{uu}(0,0) &= f_{uu}(0,0)e_1 = 1/2\,\lambda_1 e_1\\ \mathbf{x}_{uv}(0,0) &= f_{uv}(0,0)e_1 = \lambda_4 e_1\\ \mathbf{x}_{vv}(0,0) &= f_{vv}(0,0)e_1 = 1/2\,\lambda_2 e_1\end{aligned} \tag{6.5}$$

Recall that the focus vertex forms the origin of the neighbourhood frame, and therefore corresponds to the point **x**(0,0). The partial derivatives of **x** are all that is needed to compute principal curvatures over the surface, as we shall now see:

$$\kappa\left(\frac{\mathrm{d}v}{\mathrm{d}u}\right) = \kappa(\alpha) = \frac{e + 2f\alpha + g\alpha^2}{E + 2F\alpha + G\alpha^2} \qquad (6.6)$$

$$\begin{aligned} e &= \mathbf{x}_{uu}.\mathrm{N} \quad f = \mathbf{x}_{uv}.\mathrm{N} \quad g = \mathbf{x}_{vv}.\mathrm{N} \\ E &= \mathbf{x}_u.\mathbf{x}_u \quad F = \mathbf{x}_u.\mathbf{x}_v \quad G = \mathbf{x}_v.\mathbf{x}_v \end{aligned} \qquad (6.7)$$

$$\mathrm{N} = \frac{\mathbf{x}_u \wedge \mathbf{x}_v}{\left|\mathbf{x}_u \wedge \mathbf{x}_v\right|} \qquad (6.8)$$

This expression represents the normal curvature of the surface $\mathbf{x}$ in the parameter direction of α. The maxima and minima of $\kappa(\alpha)$ are characterised by $d\kappa / d\alpha = 0$. Inserting the partial derivatives from (6.5), we are left with a quadratic equation in α, which can be solved to find the directions of maximum and minimum normal curvature, α_1 and α_2 respectively, for the vertex in parameter space. These direction values can be inserted into (6.6) to compute principal curvatures κ_1 and κ_2. We can derive principal curvature directions for the vertex in $\mathbf{R}^3$ from the frame basis vectors e_2 and e_3 using (6.2).

Many useful differential geometry operators can be defined in terms of principal curvatures. Therefore, the above approach provides the foundation for a variety of (discrete) differential geometry operations [DOC76, DES00].

Figure 4 illustrates two such operators, showing approximations of vertex mean curvature and normalised principal curvature directions for a mesh. Note that the mesh on the left is not shaded, but pseudo-coloured according to mean curvature approximations.

6.3 A Greedy Triangulation Algorithm for Three-dimensional Polygons

We use a simple greedy approach during triangulation, which incrementally adds the current ear edge (fig. 5) of lowest weight to derive a polygon with one fewer vertices until all that remains is a triangle. We define an ear edge as any orientated edge which partitions an *n* vertex polygon into a triangle of the correct orientation and an *n*-1 vertex polygon (fig. 5). In other words, a candidate edge spanning consecutive vertex triples is deemed to be an ear edge, if and only if the triangle produced is consistently orientated with the chosen reference point using the test from section 6.1. In (fig. 5) the reference point is anywhere below the plane of the page, assuming counter-clockwise orientation.

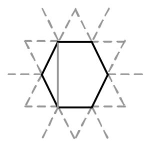
Figure 5 Example of an ear edge.

The initialisation stage consists of filling a priority-queue with the *n* candidate ear edges, ordered according to increasing weight. An iteration of the algorithm itself produces a diagonal that is part of the final triangulation, and may add up to two new, and will remove from between one and three, ear edge candidates from the queue. The diagonal is chosen as the current minimum weight ear edge, and is therefore the head priority-queue edge. The edge is then removed from the queue, and added to the triangulation. The addition of such a diagonal renders the ear edges incident on the vertex of the new triangle that is not part of the current ear edge as invalid. Any such edges are removed from the queue to ensure the resulting triangulation has the appropriate topology. The removal of the current minimum weight ear edge also allows up to two new candidate edges to be considered in subsequent iterations. For some current *n* vertex polygon, these edges represent the ear edges of the new, *n*-1 vertex, polygon that are not contained within its own set of ear edges. As the number of neighbourhood vertices is bounded from above, this approach leads to a reasonable $O(n \log n)$ triangulation algorithm when using a standard heap-based priority-queue.

Figure 6 shows the various edges involved in an iteration of the triangulation algorithm, where the current ear edge is solid grey, the corresponding invalidated edges are dotted lighter grey, and the new candidate ear edges are darker dashed grey.

During triangulation of the neighbourhood of vertex *k*, we give each candidate edge between vertex p_i and p_j a weight according to the following equation W:

$$\mathrm{W}(\overrightarrow{p_i p_j}) = \kappa_d^k \left| \overrightarrow{p_i p_j} \times \hat{d}_{\kappa_2^k} \right| + (1 - \kappa_d^k) \left| \overrightarrow{p_i p_j} \right| \qquad (6.9)$$

$$\kappa_d^k = \begin{cases} \dfrac{\kappa_1^k - \kappa_2^k}{\kappa_{\mathrm{MAX}}} & \kappa_1^k - \kappa_2^k \le \kappa_{\mathrm{MAX}} \\ 1.0 & \text{else} \end{cases} \qquad (6.10)$$

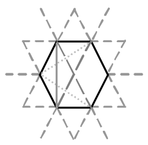
Figure 6 Edges involved in an iteration of the triangulation algorithm.

$$\hat{d}_{\kappa_2^k} = \frac{d_{\kappa_2^k}}{\left| d_{\kappa_2^k} \right|} \qquad (6.11)$$

As shown by (6.9) and (6.10), we apply a convex combination of a curvature-based edge metric and a traditional MWT-style edge metric according to a function of the principal curvatures of the neighbourhood of vertex *k*.

Equation 6.10 demonstrates the dynamic aspect of the algorithm. The κ_1^k and κ_2^k correspond to the initial principal curvatures of vertex *k*, as calculated using the method described in section 6.2. The values κ_d^k represent the relative level of curvature between the two principal curvature directions of the vertex *k*. The value κ_{MAX} represents the sensitivity of the curvature adaptation, with lower values representing higher sensitivity. We have found that $\kappa_{MAX} = 1.0$ works well in practice.

Note that the principal curvatures κ^k are not recalculated as the mesh is simplified. We have found that the initial approximation is invariably of a higher quality than those derived from a simplified mesh. This also has an attractive performance side-effect, as the relatively expensive curvature calculations are only performed as a pre-processing stage.

7 Results

We have applied our approach to many different meshes, and we have included examples of some of these in figures 7-11. We have included meshes produced using a triangulation algorithm based on the traditional MWT metric to facilitate comparison with the curvature-based approach.

The original cow mesh is composed of 2903 vertices, and the 50%, 75%, and 88% reductions consist of 1453, 729, and 374 vertices respectively. The original triceratops mesh is composed of 2832 vertices, and the 88% reductions consist of 354 vertices.

Figures 8 and 9 demonstrate the full method on the cow mesh with normal and wireframe renderings respectively. These can be compared with figures 10 and 11 to illustrate the improvements gained. Notice that the triangles composing the legs of the animal have aligned themselves with the direction of least curvature in figures 8 and 9, but have become almost equilateral in figures 10 and 11. Moreover, see that the horns of the 88% curvature-based reduction are comparable to the horns of the 75% MWT-based reduction. Also, notice that the triangulations of the main body of the cow are similar. This illustrates the dynamic nature of the curvature-based triangulation algorithm which gracefully reduces to an approximation of the MWT in areas of zero curvature.

The L_1 norm related nature of the error polyhedron metric is well illustrated in figure 7, and the level to which the curvature-based algorithm overcomes these shortcomings is also demonstrated in figure 7. See that the horns of the mesh produced by the MWT-based algorithm have completely disappeared, where as the horns of the curvature-based approach stand proud.

The meshes were produced on a 600MHz Pentium III, reaching reduction rates of over 6000 and 5000 triangles per second for the MWT and curvature based approaches respectively. Rates significantly higher than these could be achieved using a fully optimised algorithm.

8 Conclusions

We have presented a novel error metric for use during mesh simplification based on error polyhedra. The metric is both accurate and efficient, and its efficacy is well illustrated in the figures. However, its inherent similarity to the standard L_1 norm means that it suffers from identical problems, but also enjoys similar benefits. These benefits manifest themselves through fast evaluation, but when combined with a triangulation algorithm which approximates the MWT, the error polyhedra metric has a tendency to remove high frequency detail earlier than a L_∞ based metric would. As shown in the figures, this shortcoming is reduced when a curvature-based triangulation algorithm is used.

We have presented a novel approach to neighbourhood triangulation during simplification. The method is based on the dynamic use of discrete differential operators and utilises the extra dimension that most other approaches ignore. We have shown that coercing the triangulation edges to align with the direction of least curvature improves simplification quality, especially at low numbers of vertices. This method does not significantly increase simplification time and is not metric specific, and therefore it could act as an effective improvement to other approaches to simplification.

The combination of error polyhedra and the dynamic triangulation algorithm provides a fast and accurate approach which demonstrates well that methods based on vertex decimation can provide a basis for efficient and effective mesh simplification.

Acknowledgements

This work has been sponsored by the EPSRC and Kazoo3D plc (www.kazoo3d.com).

We also thank Alan Watt, Michael Meredith, and James Edge for helpful discussions during the production of this report. James Robinson has, as ever, provided much inspiration.

Finally, we are indebted to *Avalon – The Public Domain 3D Mesh Repository* for supplying the meshes used in this work.

Bibliography

Barequet, G. and Dickerson, M., and Eppstein, D (1998). On Triangulating Three-Dimensional Polygons, *Computational Geometry Theory & Applications* (10), 155-170.

Cohen, J., Olano, M., and Manocha, D. (1998). Appearance-Preserving Simplification, *Computer Graphics* Proceedings, Annual Conference Series, 1998, ACM SIGGRAPH, 115-122.

Desbrun, M., Meyer, M., and Schröder, P. (2000). Differential-Geometry Operators in nD, *Multi-Resolution Modelling Group*, California Institute of Technology, pre-print.

Do Carmo, P. (1976). *Differential Geometry of Curves and Surfaces*, Prentice-Hall, Inc., Upper Saddle River, NJ.

Eastlick, M. and Maddock, S. (2001). Triangle Mesh Simplification using Error Polyhedra, *Department of Computer Science* Technical Report, University of Sheffield.

Garland, M. (1999). Multiresolution Modeling: Survey & Future Opportunities, *State of the Art Report* (STAR), EuroGraphics '99.

Garland, M. and Heckbert, P. (1997). Surface Simplification using Quadric Error Metrics, *Computer Graphics* Proceedings, Annual Conference Series 1997, ACM SIGGRAPH, 209-216.

Garland, M. and Heckbert, P. (1998). Simplifying Surfaces with Colour and Texture using Quadric Error Metrics, Proceedings of *IEEE Visualization '98,* 263-269.

Garland, M. and Heckbert, P. (1999). Optimal Triangulation and Simplifying Surfaces with Colour and Texture using Quadric Error Metrics, Proceedings of *IEEE Visualization '98*, 263-269.

Hoppe, H. (1996). Progressive Meshes, C*omputer Graphics* Proceedings, Annual Conference Series 1996, ACM SIGGRAPH, pp. 99-108. 1996.

Lindstrom, P. and Turk, G. (1998). Fast and Memory Efficient Polygonal Simplification, *Georgia Institute of Technology* Technical Report GIT-GVU-98-11.

Ronfard, R. and Rossignac, J. Full-Range Approximation of Triangulated Polyhedra, *Computer Graphics Forum (Proceedings of Eurographics '96)*, 15(3).

Schroeder, W. J. (1992). Decimation of Triangle-meshes, *Computer Graphics* Proceedings, Annual Conference Series 1992, ACM SIGGRAPH, 65-70.

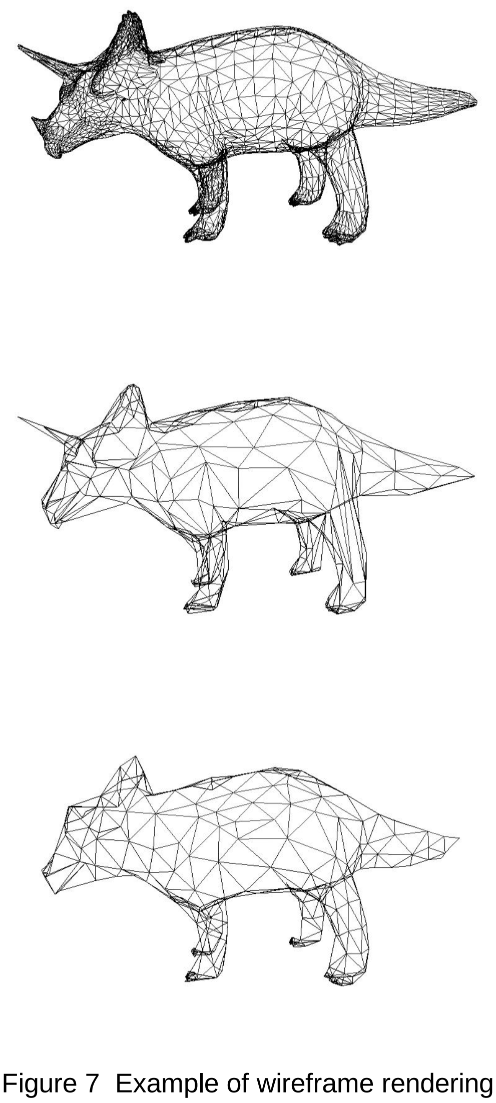
Figure 7 Example of wireframe rendering of original triceratops mesh, 88% curvature-based reduction, and 88% MWT-based reduction.

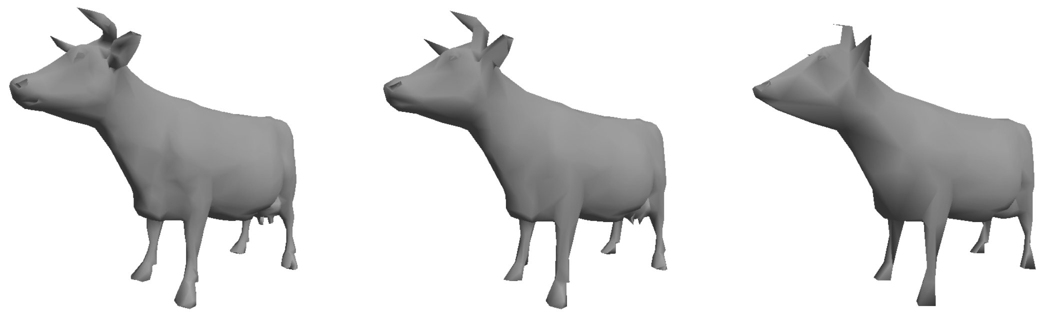
Figure 8 Standard rendering of 50%, 75%, and 88% curvature-based reduction of cow mesh.

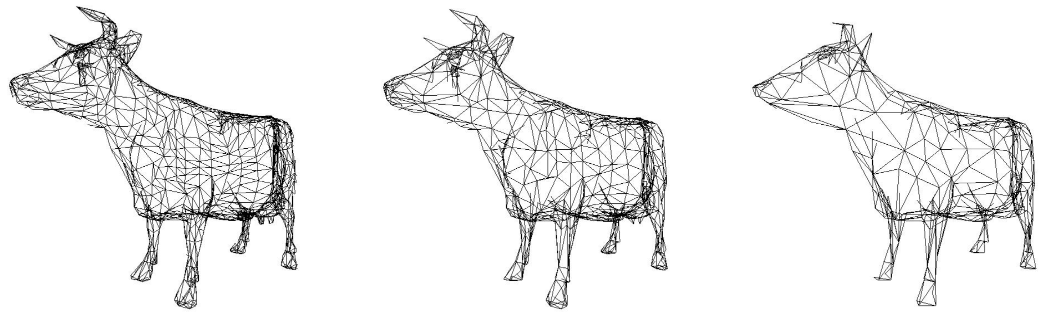
Figure 9 Wireframe rendering of 50%, 75%, and 88% curvature-based reduction of cow mesh.

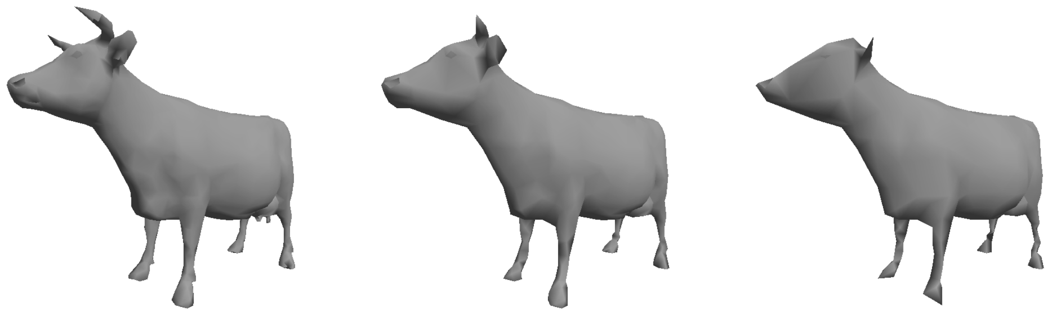
Figure 10 Standard rendering of 50%, 75%, and 88% MWT-based reduction of cow mesh.

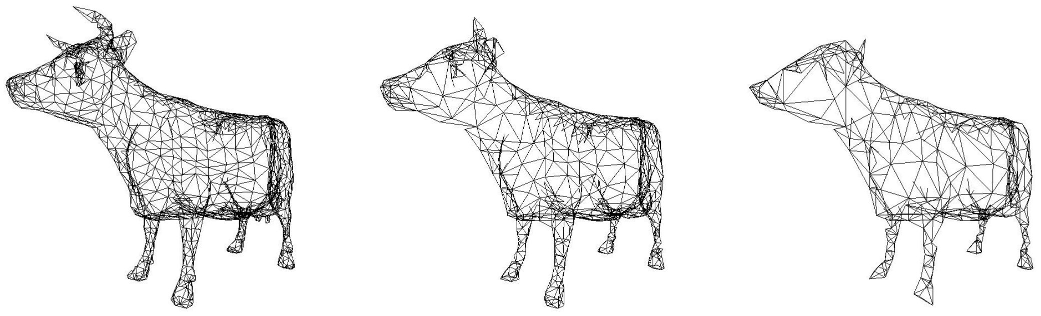
Figure 11 Wireframe rendering of 50%, 75%, and 88% MWT-based reduction of cow mesh.