Generic Memoryless Polygonal Simplification

Richard Southern Patrick Marais Edwin Blake

Collaborative Visual Computing Laboratory
University of Cape Town

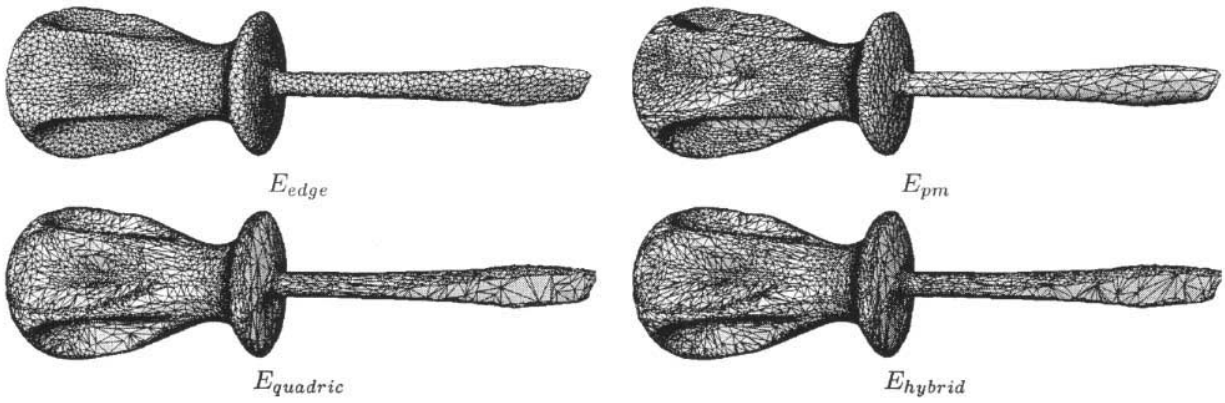

Figure 1: The screwdriver model is compressed from 54301 faces to 10000 faces with four different techniques.

Abstract

We present a new framework for generic and adaptive memoryless surface simplification. We show that many existing techniques of simplification based on the edge collapse / vertex split operations differ only in terms of memory-resident data used to improve running performance. By removing the need for this memory we are able to implement multiple simplification techniques on the same platform. Our generic platform can be used as a tool for the generation and evaluation of custom error metrics. We present two new error metrics designed using our generic framework. We present a novel batched ordering technique based on the generic simplification framework, which allows for adaptive simplification and automatic level-of-detail generation.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric Transformations; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Object Hierarchies; I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics Data Structures

Keywords: triangle mesh simplification, level of detail, error metric comparison

1 Introduction

We define a surface as an oriented 2-manifold, meaning a one-sided "skin" which is used to represent an object in a virtual environment. Surfaces such as those generated during the Michelangelo Project [11], can be simply too large to store, let alone transmit. A "virtual museum" consisting of on-line exhibits of such works of art would be of practical interest. Unfortunately, the storage and transmission time required for viewing such models is prohibitive.

Surfaces can be simplified (or compressed) by iteratively removing information from the model while still retaining surface connectivity and preserving attributes, such as topology, face orientation and volume. A large number of simplification techniques and error metrics have been presented based on "decimation" [15].

A common theme in these simplification techniques is the localization of the affected region during each iteration of the process — only faces and vertices within the domain of simplification need to be changed in the current mesh after a single operation. We consider only error measures and point placement strategies which are local in nature and depend only on the region within the edge collapse domain.

Error metrics for surface simplification based on the basic edge collapse operation are numerous and varied, but possess the same underlying algorithmic structure. The main distinction between these techniques is the additional memory used to improve performance. Comparing these different techniques under the same conditions is a difficult task.

We exploit consistencies between the various simplification strategies to define a generic framework. By defining a novel batched ordering technique we are able to adaptively switch between simplification techniques during simplification, and automatically produce a sequence of level of detail models. This framework has applications in error metric evaluation and adaptive compression. It allows a variety of output configurations including view-dependent refinement and continuous or discrete level-of-detail sequences. Our framework also provides a valuable test-bed for the

1. define candidates for the edge collapse operation,
2. sort the candidates on some criterion (typically some error which is incurred after a vertex removal),
3. perform the edge collapse operation, resulting in the removal of surface detail,
4. update the remaining candidates in the list, and
5. if there are still candidates, goto step 3.

Figure 2: A generic algorithm for simplification.

creation of custom error metrics, and we present two new error metrics designed within our generic platform.

2 Background

A number of authors [1, 2, 3, 4, 6, 8, 12] have contributed to the broad spectrum of simplification techniques. The general algorithm governing iterative model simplification is shown in Figure 2.

The above techniques make use of the edge collapse operations for simplification (see Figure 3), but differ fundamentally in three respects:

- *vertex placement* — refers to the way in which the geometry and connectivity is updated after an edge collapse has been performed;
- *error metrics* — refers to the ordering of operations. The ordering is determined by the error incurred after the application of a edge collapse;
- *memory overhead* — refers to additional resources used to speed up error metric calculation.

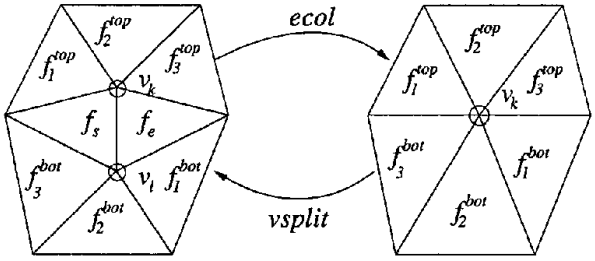

Figure 3: The edge collapse / vertex split. The vertex v_l and the edge between v_l and v_k are removed from the mesh after the edge collapse *ecol* has been applied. The inverse vertex split *vsplit* operation reintroduces these attributes into the mesh. The region depicted is called the *edge collapse domain*, and consists of all edges and faces originating from the central verticesv_k and v_l. All vertices in the region besides v_k and v_l are referred to the *base points* of a region.

We require specific terminology to define vertices and faces which are used during simplification. We denote a mesh consisting of a set of vertices and their connectivity as M^j, where j indicates the current level of resolution.

Since we reuse the vertex index of one of the vertices in the region of the edge collapse, we refer to the vertex we *keep* as v_k, (see Figure 3) while the vertex we *lose* is referred to as v_l. For consistency we orientate the figure so that v_k is above v_l.

We define the faces which are to be removed as the *start* face f_s and the *end* face f_e. We define the vertices $v_i^b, i = 1 \ldots t$ as the *base points* of the region, since their attributes do not change during decimation. In Figure 3 these are all vertices shown besides the vertices v_k and v_l.

The set of faces surrounding the vertex v_k, excluding the two removed faces f_s and f_e is referred to as the *top fan domain*, or $\mathcal{TOP}$ (in Figure 3 this would be defined as $\mathcal{TOP} = \{f_1^{top}, f_2^{top}, f_3^{top}\}$). Similarly the set of faces surrounding v_l is referred to as the *bottom fan domain*, or $\mathcal{BOT}$. Each set is constructed in an anti-clockwise manner about its focus point (v_k or v_l) from f_s or f_e respectively.

Hoppe[6] defines a multi-resolution hierarchy during the simplification process. Given a final mesh $\hat{M} = M^n$, a sequence of intermediate meshes M^j can be found by incrementally applying edge collapse operations to each consecutive level of resolution, i.e.

$$M^n \xrightarrow{ecol_{n-1}} M^{n-1} \xrightarrow{ecol_{n-2}} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0.$$

Each consecutive level of resolution only differs from the mesh preceding it by the presence of one or two faces. Hoppe also points out that this progressive format is invertible, and the original mesh can be reconstructed by applying the inverse vertex split operations. The progressive format of [6] is useful in view-dependent refinement, continuous level-of-detail generation and progressive transmission. A progressive mesh consists of a base mesh M^0 and the ordered sequence of n vertex split operations $\{vsplit_1, \ldots, vsplit_n\}$ necessary to restore the original mesh M^n.

Error Metrics for Simplification

Central to the resulting quality of models analyzed by local decimation techniques is the error metric used to determine the locations of the vertices after each iteration. There are effectively three approaches to determining the position of the kept vertex v_k after the edge collapse.

- *Fixed placement*: The simplest technique of vertex placement collapses the edge to only one fixed point, typically the midpoint of v_k and v_l. For reconstruction, only the correction δv_k needs to be stored, as $\delta v_k = -\delta v_l$. These three floating point values can be effectively Huffman encoded, and yield a high degree of compression, such as in Pajarola *et al.*[13]. When applied to largely convex surfaces this vertex placement will result in shrinkage of the overall volume of the surface.
- *Subset placement*: A simple modification on fixed placement is to determine the error of a number of candidate vertices and choose the point which offers the least error. This was first introduced by Hoppe[6] and produces reasonable results — the vertex is chosen from either a half-edge collapse (to either v_k or v_l) or the midpoint of the two. In this case, only δv_k and two additional bits needs to be stored with each vertex split operation to determine where the resultant vertex lies. As with fixed placement, subset placement results in volume shrinkage.
- *Optimal placement*: The point position can be determined using optimization. Unlike the above two techniques, the resultant point can lie anywhere on the model. The criteria for optimization varies from the distance from the nearby planes[4] to volume and triangle shape preservation[12, 8]. This technique requires the storage of both δv_k and δv_l, as the new vertex position is unconstrained.

Hoppe[6] associate an error term with each edge collapse operation by determining the distance of the points of the current surface with that of the original surface $\hat{M}$. Additional terms are added to this measure in order to ensure convergence of the simplification[9] and to preserve scalar attributes and discontinuities. The new position of the vertex v_k is then determined using a subset placement strategy.

Garland *et al.* [3] use an unconstrained vertex placement strategy by minimizing the distance of the new point from the surrounding faces of the region. Essentially this is solved as an inverse problem, resulting in an optimal position for the new vertex and an error associated with collapsing to this point. The same technique can be modified to accommodate attribute information such as vertex color values and surface normals[4].

Lindstrom *et al.* [12] introduce the concept of *memoryless simplification*. They do not store any edge collapse history during different stages of the decimation procedure, except for the priority queue required to order the atomic operations. The vertex position is found by constraining the point position in three near-orthogonal planes, and optimizing the point location by means of up to three constraints, such as triangle shape preservation, signed and unsigned volume preservation. The solution is found using quadric optimization.

Generic Simplification

Kobbelt *et al.* [10] introduce a "generic" simplification algorithm. They divide simplification criteria into *distance measures* — which attempt to minimize the deviation of the mesh after the application of a single simplifying operation — and *fairness criteria* — which ensure that the model is consistent (for example ensuring no triangle degeneracy).

This distinction is unnecessary, as many simplification techniques include a fairness component either implicitly[3] or explicitly[6, 8]. Rather than introduce a framework of generic simplification, they classify existing simplification metrics into these two criteria, and introduce a new error metric based on this classification.

Batched Operations

Gúeziec *et al.*[5] present a method of automatic level-of-detail partitioning. By defining the "level" of a particular vertex during simplification, they are able to apply refinement "batches" to create a level-of-detail model at a particular resolution. By using a graph of the refinements generated during simplification, they are able to selectively modify the level-of-detail of different areas of the model.

Our batched hierarchy does not permit the construction of a hierarchy for dynamic level-of-detail partitioning, but guarantees that refinement operations at a particular level are independent. We also have considerably fewer levels of detail, as we use the maximum number of independent simplification operations at each level. Our technique also permits changing the simplification technique after the completion of a batch (in Section 6), as the priority queue of simplification operations is empty.

3 Method Overview

We present a novel framework which permits the implementation of multiple atomic compression strategies and error metrics on the same platform. The underlying principle behind the generation of decimation meshes is the iterative application of edge collapse (*ecol*) operations. This process continues until some user specified stopping criteria is reached, or no further simplification is possible.

Before compression, the model must be converted to a structure where the neighbors of each face must be stored (as in [7]), in order to speed up traversal about faces of the mesh. Compression is initialized by inserting all valid edge collapse operations into a priority queue sorted on the error value associated with the operation. Multiple vertex placement techniques are accommodated by considering each technique as unconstrained. Compression due to vertex placement is resolved at output time.

Decimation takes place progressively, where the edge collapse with the smallest error is retrieved from the queue and applied to the mesh. All edge collapse operations within the *edge collapse domain* of that operation must either be updated or deleted (see Section 6). A batched hierarchy can be used to automatically generate level of detail sequences during simplification, and allows the error metric to be changed during surface simplification.

4 Memoryless Error Metrics

Error metrics are designed to assign a weighting to $ecol_i$ according to how much its application would affect the mesh. These weightings are used to order the operations in such a way that the compressed mesh appears as close as possible to the original model. Typically error metrics are constructed from a number of criteria.

In the following sections we show how two commonly used error metrics, that of Hoppe[6] (Section 4.1) and Garland *et al.*[3] (Section 4.2) can be converted to a memoryless version. We also introduce two novel error metrics designed within the generic framework in order to show the versatility of our technique. Edge length (defined in Section 4.3) is a simple measure which can be used to regularize a mesh in terms of triangle area, and a hybrid scheme (defined in Section 4.4) which preserves normal attributes and mesh volume.

4.1 Progressive Meshes (E_{pm})

Hoppe[6] defines four error terms for surface simplification. Only two of the terms, E_{dist} and E_{spring} are relevant to surface geometry. E_{scalar} is determined by scalar surface attributes, such as color, while E_{disc} is a term to allow the user to guide the simplification over regions of high curvature.

The E_{dist} term, which measures the distance of the current surface from the original surface, is difficult to replicate in a memoryless form. However, Hoppe[8] states that it is sufficient to consider only the current configuration of the model when determining error measures, as quality is not worsened (in the case of [8] it was found that quality was actually improved). We define

$$E^M_{dist}(v'_k) = \sum_{i \in \mathcal{P}} d^2(v'_k, \mathbf{p}_i), \qquad \mathcal{P} = \{\mathcal{TOP} \cup \mathcal{BOT}\}$$

where $\mathbf{p}_i$ is the plane representing triangle i in the surface. $d^2(v, \mathbf{p})$ is the distance of point v from from the plane $\mathbf{p}$. defined by

$$d^2(v, \mathbf{p}) = \left(\frac{\mathbf{n} \cdot \mathbf{q}}{||\mathbf{n} \cdot \mathbf{q}||} \right)^2,$$

where $\mathbf{n}$ represents a normal to the plane $\mathbf{p}$, and $\mathbf{q}$ is a vector from a point on the plane $\mathbf{p}$ to the point v.

The spring term, E_{spring} is independent of the original surface, and could be computed during each phase of the simplification. We define

$$E^M_{spring}(v'_k) = \sum_{v_b \in \mathcal{BASE}} \kappa ||v'_k - v_b||^2,$$

where the set $\mathcal{BASE}$ are the vertices in the base points of a region. κ is a scaling factor used to weight the importance of the E_{spring} component.

In our implementation, we simulate the error metric of Hoppe by using

$$E_{pm}\left(v'_k\right) = E^M_{dist}\left(v'_k\right) + E^M_{spring}\left(v'_k\right)$$

4.2 Quadric Error Metric ($E_{quadric}$)

Garland and Heckbert[3] determine the position of v'_k by minimizing the squared distance from the new point to the surrounding planes in the *edge collapse domain*, weighted by the area of each face. In order to accelerate the simplification process, the quadric matrix Q is stored for each vertex prior to simplification and updated during the process. The quadric matrix for each edge collapse is then the sum of the matrices representing the vertices v_k and v_l. This results in an unnecessary weighting of the quadric error towards the two removed faces f_e and f_s, as they would be represented in both matrices.

A memoryless translation of this technique would require the calculation of a quadric matrix Q for each vertex each step of the simplification. Although this slows the process, excess memory usage is eliminated, and the incorrect weighting caused by the edge collapse domain described above is eliminated. Hoppe [8] finds that a memoryless version of the quadric error metric produces better results than the memory resident equivalent.

4.3 Edge Length (E_{edge})

Probably the simplest criteria for determining the suitability of performing an edge collapse is the length of the edge being collapsed. Simplifying with only edge length as a criteria ensures that vertices at each stage of the decimation are evenly clustered. This is seldom a desirable property in mesh compression, since areas of high curvature are as simplified as areas without. However, edge length simplification can be successfully used to simplify dense models very quickly, and can be used to produce a mesh which has a regular point density — i.e. along the surface points are kept roughly the same distance apart.

We define our term E_{edge} as the squared distance between v_l and v_k. This term is independent of the new vertex position, so we collapse v_l and v_k to their midpoints. Although deceptively simple, a error metric based on edge length has a number of applications where degenerate triangles (called slivers) are highly undesirable. As will be shown in our results section, an error term based only on edge length although runs very quickly, and is suitable for the quick simplification of large models.

4.4 A Hybrid Scheme (E_{hybrid})

Surface normals are a natural measure of the curvature of a surface. Simplification algorithms commonly reduce detail in regions of low curvature, while retaining detail in areas of high curvature. We make use of a normal preservation term E_{norm} which is derived from the maximum deviation from the normals of the planes in $\mathcal{BOT}$ and $\mathcal{TOP}$ to the normals of the equivalent triangles in the new region. This can be written more formally as:

$$E_{norm}\left(v'_k\right) = \max_{i \in \mathcal{P}}\left(1 - \mathbf{o}_i \cdot \mathbf{n}_i(v'_k)\right), \qquad \mathcal{P} = \{\mathcal{TOP} \cup \mathcal{BOT}\}$$

where $\mathbf{o}_i$ returns the original normal of the i_{th} face, and $\mathbf{n}_i(v'_k)$ returns the normal of face i after all instances of v_k and v_l have been replaced by v'_k. E_{norm} produces a normalized term, with a value in the range $(0 \ldots 1)$ — a value closer to 0 implies a small normal deviation. Note that this term can also be used to test for face flipping during simplification. A value of E_{norm} greater than 1 would imply that the normals face in the opposite direction. The results of simplifying with only the E_{norm} shows that curvature is preserved to a significant degree. However, it produces intolerable results due to the high degree of volume loss and triangle slivers. We apply a local volume preservation term in order to reduce the resulting error.

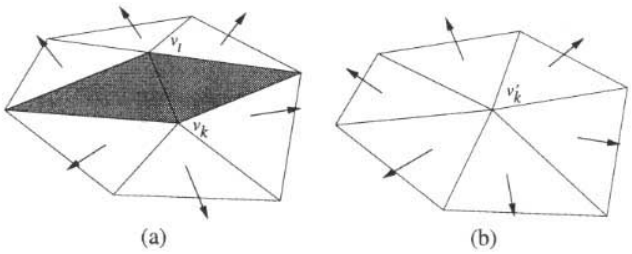

Figure 4: The derivation of E_{norm}. In (a) the normals of the original region $\mathbf{o}$ are shown, while in (b) the new normals $\mathbf{n}$ (i.e. after the edge collapse) are shown.

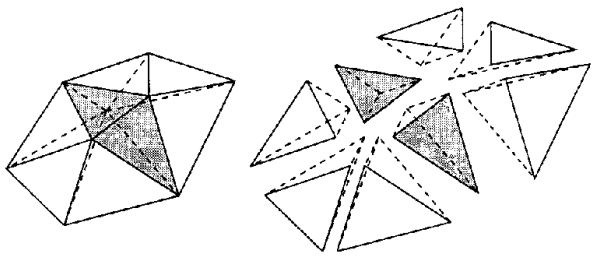

Figure 5: The simplification region is separated into its component tetrahedral volume elements. The old region is indicated in a solid line, the new region indicated with a dashed line and shaded faces are removed after simplification.

Lindstrom and Turk[12] perform local volume preservation by defining a signed volume component of their optimization. They divide the local *edge collapse domain* into representative tetrahedrons (see Figure 5), constructed from the three points of each triangle, and the new vertex. We use the unsigned volume to measure the deviation of our models. We define

$$E_{uvol} = \sum_{i \in \mathcal{P}} \text{Tet\_Vol}\left(v'_k, v^i_1, v^i_2, v^i_3\right), \qquad \mathcal{P} = \{\mathcal{TOP} \cup \mathcal{BOT}\}$$

where v^i_j indicates the j_{th} vertex of the i_{th} face, $\text{Tet\_Vol}\left(v', v^i_1, v^i_2, v^i_3\right)$ returns the volume of the tetrahedron formed by the four input points. We combine it with the term E_{norm} to yield

$$E_{hybrid}\left(v'_k\right) = E_{uvol}\left(v'_k\right) . E_{norm}\left(v'_k\right).$$

5 Surface Simplification

Error metrics are used to order edge collapse operations during simplification and vertex placement after each operation. In order to construct a progressive mesh, we must reduce the full resolution mesh to a base mesh M^0 and a sequence of vertex split operations necessary to retrieve the original model. We present two methods for producing an ordered sequence of vertex split operations:

- A simple *linear* hierarchy, where operations must be performed in the specified order (as in [6]), and

- a *batch* hierarchy, where independent operations are batched together — operations in the batch can be performed in any order, but no operation in a following batch can be performed until it's preceding batch has been completed.

The batched hierarchy is useful when determining valid candidates for geomorph operations. This will be further discussed in Section 6.

Vertex Removal

An edge collapse record, $ecol_i$, is a tuple consisting of $\{k, l, s, e, \epsilon\}$, where k, l, s and e are the indices of v_k, v_l, f_s and f_e respectively (as in Figure 3), and ϵ is the error incurred by the removal of v_l, f_s and f_e. The error term ϵ depends largely on the error technique used. An *ecol* record is generated from every valid edge within the original mesh M^n. These are sorted, and the $ecol_i$ with the smallest error ϵ is selected in turn. Each $ecol_i$ is then tested for validity according to the following criteria:

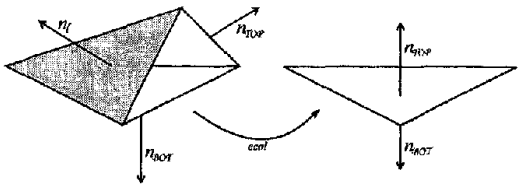

Figure 6: Topology Change. The removal of the shaded face results in a two sided face and hence a non-manifold mesh. n_l refers to a normal from the removed face, while $n_{\mathcal{TOP}}$ and $n_{\mathcal{BOT}}$ are the normals to the faces in the face sets $\mathcal{TOP}$ and $\mathcal{BOT}$ respectively.

- *Does removal of this edge result in a topological change?* A general heuristic to determine the presence of mesh folding is to determine whether any face in $\mathcal{TOP}$ is a neighbor of any of a face in the set $\mathcal{BOT}$. The consequence of not performing this test is shown in Figure 6.

- *Does removal of this edge result in face flipping or folding?* This problem is addressed in [6, 8]. We chose to perform a simple face orientation test - a rotation of a face more that $\pi/2$ radians implies the face would flip (as in Figure 7). It should be noted that the determination of the normals at each iteration of the process is an expensive time overhead.

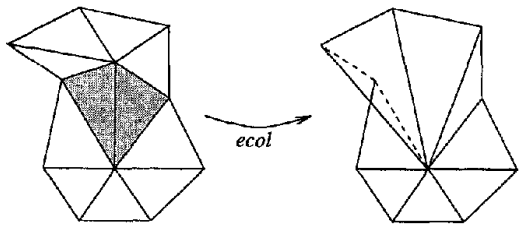

Figure 7: Face Flipping. Removal of the shaded faces results in a hidden or intersected face (indicated here with a dashed line). This can be avoided by testing the orientation of the faces in the $\mathcal{ECD}$ before and after the edge collapse operation.

Failure to comply with either of these two tests results in $ecol_i$ being deleted from the queue without being performed. These particular edge collapse operations may be reinserted into the queue at a later stage, since operations within the *edge collapse domain* of all edge collapse operations are reinserted into the queue.

If $ecol_i$ is considered valid it is applied to the current mesh. Once $ecol_i$ has been performed, a number of other $ecol_k$ records are updated:

- any $ecol_k$ containing f_s or f_e is erased,
- the error ϵ in any $ecol_k$ containing any of the faces in the $\mathcal{TOP}$ or $\mathcal{BOT}$ of $ecol_i$ are updated, since faces in the $\mathcal{TOP}$ and $\mathcal{BOT}$ of $ecol_k$ have been removed, and
- the error ϵ in any remaining $ecol_k$ in the *edge collapse domain* of $ecol_i$ is recalculated, since the orientation, shape and area of the faces in $\mathcal{TOP}$ and $\mathcal{BOT}$ of $ecol_i$ may have been altered.

All affected records $ecol_k$ are updated by erasing and reinserting them into the queue. In Section 6 this is modified to accommodate the batch hierarchy.

The inverse $(vsplit)_{i+1}$ operation can be deduced from $ecol_i$. A $(vsplit)$ tuple consists of

$\{k, l, s, e, top_0, top_n, bot_0, bot_m, \delta v_k, \delta v_l\}$, where :

- k, l, s and e are the indices of v_k, v_l, f_s and f_e respectively,
- top_0 and top_n are the indices of the first and last faces in $\mathcal{TOP}$,
- similarly bot_0 and bot_m are the indices of the first and last faces in $\mathcal{BOT}$,
- δv_k and δv_l represent the values which must be added to the current location of v_k in the current mesh to losslessly restore the positions of v_l and v_k. These can be efficiently encoded with Huffman or entropy encoding.

Once there are no longer any valid *ecol* records, the process is terminated - the resulting mesh represents the base mesh M^0. The original mesh M^n can be reconstructed by applying the sequence of transformations $\{vsplit_1, \ldots, vsplit_n\}$ in order.

6 Batched Operations

In order to ensure that operations in each batch are independent, edge collapse operations which are in the *edge collapse domain* of the operation being performed are **not** reinserted into the queue of valid operations. Once the queue is empty, a marker is written to the output file, and the edge collapse queue is rebuilt from the current version of the mesh. This process is continued until a batch process reaches completion without removing any vertices. The modified multi-resolution sequence becomes:

$$M^0 - \left\{ \begin{array}{c} B^0 \\ vsplit_1 \\ vsplit_2 \\ \vdots \\ vsplit_p \end{array} \right\} \rightarrow M^p - \left\{ \begin{array}{c} B^1 \\ vsplit_{p+1} \\ vsplit_{p+2} \\ \vdots \\ vsplit_{p+q} \end{array} \right\} \rightarrow \cdots \rightarrow M^n$$

where p and q represent the number of *vsplit* operations in batches B^0 and B^1 respectively.

Although any of the *vsplit* operations within B^0 can be applied at any time, every operation within B^0 must be completed before any within B^1 can be performed. Note that the batch sizes increase as the model resolution increases, where the largest batch is applied to reach the final mesh M^n. The independence of the *vsplit* operations in each batch increase the set of possible mesh configurations, compared to the standard linear hierarchy.

Level Of Detail Generation

The application of every refinement operation in a batch results in a view-independent refinement of the entire object. After each batch has been performed during simplification an intermediate model can be saved, providing an incremental and automatic level-of-detail sequence. Typically the number of faces in a level of detail model M^i has half the number of faces in M^{i+1} (as in Figure 12). It should be noted that although the difference in the number of faces between models can be *reduced* by increasing the span of the edge collapse domain, it cannot be *increased* with this algorithm.

Adaptive Simplification

Batching also allows the metric to be changed during the simplification, and allows for a form of adaptive simplification. Garland and Heckbert[3] use a subset placement strategy when optimal placement positions v_k in an unsuitable position. This can occur in regions of high curvature and irregular triangle size. This is an example of *adaptive simplification*, where alternative vertex placement strategies are employed.

This definition can be extended to include adaptive alteration of the error metric during *batches* of simplification. Obviously the error values which are used to sort the items in the queue cannot be adaptively changed if there are existing items in the queue with differing error methods, since the error ϵ is differently scaled. The error metric can be changed when the queue is emptied after a batch has been completed.

An example of adaptive simplification would be to start simplification of a large model with E_{edge} due to it's quick running time, and switch to another error metric when the preservation of detail becomes important. In this way unnoticeable detail is removed quickly, and feature preserving metrics can be applied when the mesh less complex. This type of technique would be useful in dense models such as those derived from laser scanning.

Hoppe[6] finds that the shape preservation term E_{spring} is most applicable during the start of the simplification, and diminishes in importance later. By decreasing the coefficient of E_{spring}, κ as our batch number increases, we can adaptively scale the importance of this term. This is only possible once the queue of edge collapse operations is empty, otherwise inserted error terms may be improperly scaled.

7 Implementation

The GeMS (or Generic Memoryless Simplification) tool was written in C++, and produces quick, high quality progressive models suitable for compression and progressive transmission. Storage of the mesh M^n and its subsequent levels is facilitated by a mesh class, similar to that used by Hoppe[7]. Like Hoppe, we speed up face traversal by determining the neighbors of each face before decimation begins. This process is traditionally slow ($O(n^2)$), but can be improved with the use of heuristics.

In Figure 8, the basic algorithm for producing decimation meshes with linear dependence is shown. As described in Section 5, the function *generate_ecol_queue()* passes over the model M^n, creates *ecol* records from each valid edge, and inserts them into a heap. The function ecol_queue.*min()* returns the first element in ecol_queue and deletes it from the queue. The function *valid()* performs the tests described in Section 5 to determine the validity of an edge collapse. The functions ecol_queue.*update(k)* and ecol_queue.*delete(k)* update or delete the record $ecol_k$ in the heap respectively. *current_ecol.ecd* refers to the edge collapse domain surrounding *current_ecol.ecd*.

In Figure 9, the algorithm for the generation of batched decimation meshes is presented. Note that any *ecol* records representing

```
proc linear_hierarchy()
begin main
    open outfile
    m = 0
    ecol_queue = generate_ecol_queue(M^n)
    while not ecol_queue.empty()
      current_ecol = ecol_queue.mindelete()
      if valid(current_ecol) then
        m++
        M^(n-m) = apply_ecol(current_ecol, M^(n-m+1))
        foreach ecol_k in current_ecol.ecd
            ecol_queue.delete(k)
            ecol_queue.update(k)
        end foreach
        write vsplit_i to outfile
      end if
    end while
    write M^(n-m) to outfile
    close outfile
end main
```

Figure 8: An algorithm for generating decimation meshes with linear dependence in GeMS.

```
proc batched_hierarchy()
begin main
    open outfile
    m = L = count = 0
    ecol_queue = generate_ecol_queue(M^n)
    loop
      if ecol_queue.empty() then
        if count==0 then break
        else
          count = 0
          write marker to outfile
          ecol_queue = generate_ecol_queue(M^(n-m))
        end if
      end if
      current_ecol = ecol_queue.mindelete()
      if valid(current_ecol) then
        m++
        count++
        M^(n-m) = apply_ecol(current_ecol, M^(n-m+1))
        foreach ecol_k in current_ecol.ecd
            ecol_queue.delete(k)
        end foreach
        current_vsplit = inverse(current_ecol)
        write current_vsplit to outfile
      end if
    end loop
    write M^(n-m) to outfile
    close outfile
end main
```

Figure 9: An algorithm for generating the batched hierarchy. Note that it is similar in most respects to the linear hierarchy, except that *ecol* operations in the edge collapse domain (*ecd*) are not updated, but removed. Once the queue has completed processing the entire *ecol_queue* is rebuilt.

edges within the *edge collapse domain* of *current_ecol* are erased (as described in Section 6).

The framework allows the user to choose a simplification algorithm from a number of atomic simplification schemes, so that the output is tailored to their specifications. The framework allows also for a number of termination criteria, such as restricting the number of faces or vertices, the size of the file, or the total error incurred.

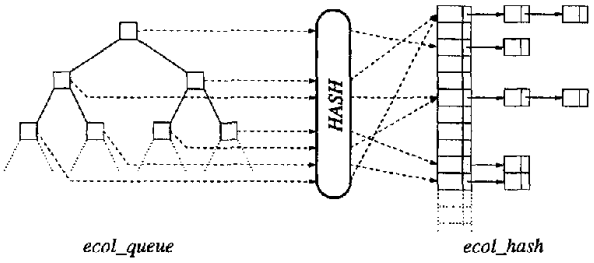

Figure 10: The hashed priority queue used within the GeMS framework.

Like [6] (amongst others) GeMS uses a priority queue (as a heap) of edge collapse records for quick sorting and extraction of the smallest element. The heap is sorted on the error ϵ of each edge collapse operation.

Central to speed considerations during decimation is the access time of the priority queue *ecol_queue*. This was implemented as a *hashed priority queue*. The hash table is built upon the indices of the start and end faces, f_s and f_e, and must be capable of rapidly determining whether an *ecol* is already present. For this reason, we modify the hashing technique to represent a linked list of all *ecol* references to which the hash function refers, so that we can determine whether the record is present (shown in Figure 10). This hash table stores the index within *ecol_queue* which contains f_s and f_e. Since the heap is constantly changing in size, this index value must be updated regularly.

Results

In Table 1 timing results are shown of the various error metrics implemented within our memoryless framework. It is clear from these results that simplification times differ greatly depending on the techniques used. The resulting model quality of these error metrics is shown in Figure 11 and Figure 1 for comparison purposes.

It is clear from these results that the technique E_{edge} executes very quickly in comparison to the alternative techniques, but the resultant model quality (shown in the top left figure of Figure 11) disregards areas of high curvature and detail features on the mesh.

In Figure 12 the batched hierarchy has been used to automatically generate a level of detail hierarchy. Each consecutive level of detail has roughly half of the faces of the previous level, and are automatically generated during simplification.

Metric	Time	Placement
E_{edge}	0.43 μs	Fixed
E_{pm}	13.0 μs	Subset
E_{hybrid}	90.5 μs	Subset
$E_{quadric}$	218.0 μs	Unconstrained

Table 1: The times shown represent the average time taken to compute a single error value, and are independent of the model being simplified. These values were generated on an Athlon 500Mhz processor.

8 Conclusion and Future Work

We have presented a novel generic memoryless polygonal simplification framework for efficient compression and hierarchical decomposition of triangular surface meshes. Necessary tests to prevent mesh faults and a method of indicating operation dependencies have been described.

Memoryless simplification offers an alternative to traditional approaches which use a simplification "history". Although running slower than the memory intensive equivalent, it requires less storage and often produces better results. We show how two existing error metrics, that of Hoppe[6] and Garland and Heckbert[3], can be converted to a memoryless derivative. We also present two new error metrics, E_{edge} and E_{hybrid} which have been implemented within our framework.

Our novel batched ordering technique presents an alternative to linear mesh reconstruction. We show that memoryless error metrics and a batched ordering technique permit adaptive simplification, where different simplification techniques can be used while simplifying a single model, and automatic level of detail generation.

At present, the scheme does not change the models topology — the topology of the surface will be preserved during simplification. Similarly, breaks in the surface (i.e. faces with no neighbors) are also preserved. Unfortunately, a simple hole-filling algorithm cannot guarantee satisfactory results, since tiling non-convex holes can produce flipped and hidden faces. In the future, we would like to extend the work of Popovic and Hoppe[14] or Garland and Heckbert[3] to allow unconnected vertices to be contracted during model simplification.

Our framework is an ideal platform for the comparison of current error metrics and techniques of vertex placement, as well as a convenient interface to design custom error metrics. We intend to analyze the results of the different techniques using our platform.

Acknowledgments

We would like to thank Cyberware™ for their support of this field of research by making their *teeth*, *screwdriver* and *female1* models available to us. We would also like to extend our thanks the members of the Collaborative Visual Computing Laboratory, for contributing to the inspiring and creative energy of our research group.

References

[1] D. Cohen-Or, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. In *IEEE Visualisation*, pages 67–72, San Fransisco, 1999.

[2] C. Erikson and D. Manocha. Gaps: General and automatic polygonal simplification. In *Symposium of Interactive 3D Graphics*, pages 79–88, 1999.

[3] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 209 – 216, 1997.

[4] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualisation*, pages 263 – 270, 1998.

[5] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *ACM VRML*, pages 25–31, 1998.

[6] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 99 – 108, 1996.

[7] H. Hoppe. Efficient implementation of progressive meshes. Technical Report MSR-TR-98-92, Microsoft Research, January 1998.

[8] H. Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualisation*, pages 59–66, 1999.

[9] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimizatio. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 19 – 26, 1993.

[10] L. Kobbelt, S. Campagna, and H.-P. Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, 1998.

[11] M. Levoy, S. Rusinkiewicz, M. Ginzton, J. Ginsberg, K. Pulli, D. Koller, S. Anderson, J. Shade, B. Cirless, L. Pereira, J. Davis, and D. Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 131–144, 2000.

[12] P. Lindstrom and G. Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualisation*, pages 279–286, 1998.

[13] R. Pajarola and J. Rossignac. Compressed progressive meshes. Technical Report GIT-GVU-99-05, Georgia Institute of Technology, 1999.

[14] J. Popović and H. Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 59–66, 1997.

[15] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Proceedings of SIGGRAPH (Computer Graphics)*, pages 65–70, 1992.

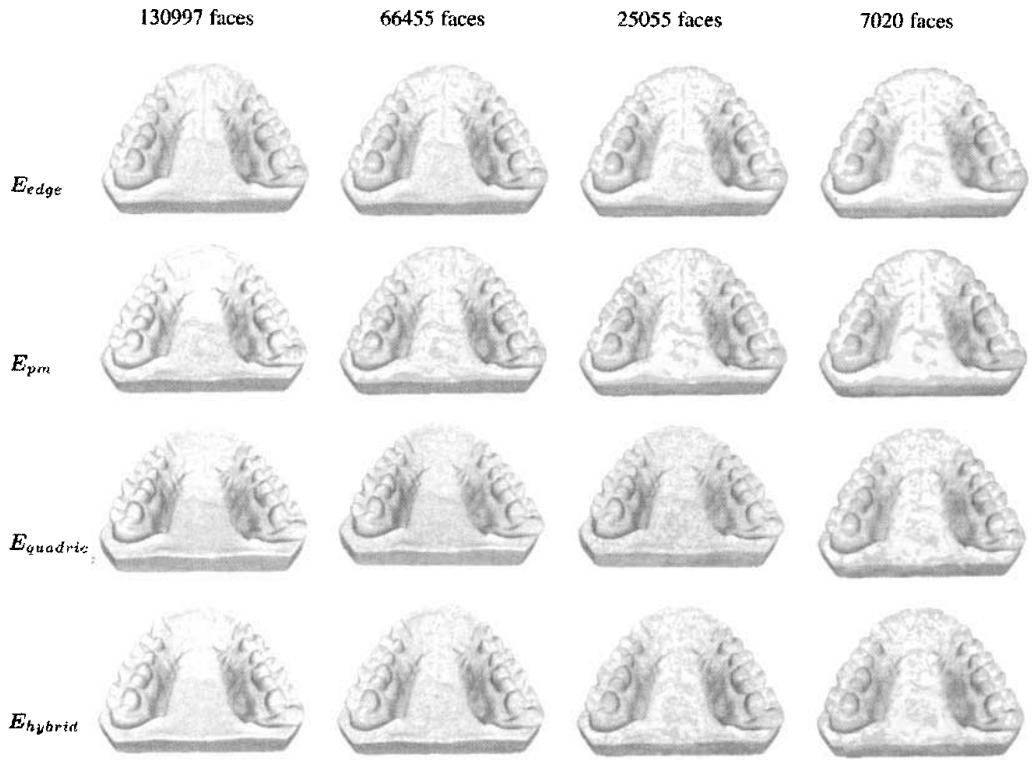

Figure 11: A generic platform allows the comparison of various techniques under the same conditions. Here we show the four error metrics discussed in Section 4 at various levels of simplification. In this visualization the original model (233205 faces) has been alpha-blended over the simplified model in green. Simplified areas which differ from the original model in terms of volume shrinkage are clearly discernible. Note that volume shrinkage occurs over areas of differing curvature depending on the error metric used.

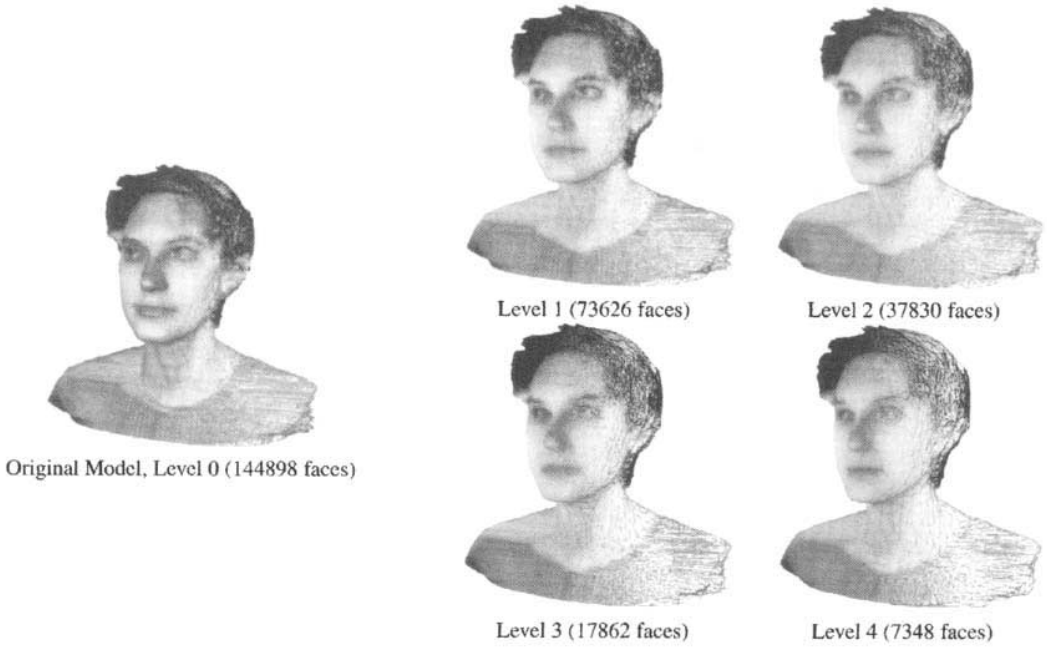

Figure 12: Batched ordering defines a natural level of detail hierarchy, where the model is compressed to half it's original size after each batch has been applied.