Quality Control Tools for Interactive Rendering of 3D Triangle Meshes

Richard Southern Edwin Blake
Patrick Marais

CS00-27-01
November 12, 2001

Collaborative Visual Computing Laboratory
Department of Computer Science
University of Cape Town
Private Bag, RONDEBOSCH
7701 South Africa
e-mail: {rsouther,edwin,patrick}@cs.uct.ac.za

Abstract

In this dissertation we explore methods of quality control of untextured polygonal models. The tools presented build, evaluate and improve on the field of multiresolution analysis through decimation. We evaluate the quality of models generated through various simplification algorithms to develop efficient measures of image quality. We develop an application for selective, progressive and view-dependent refinement, suitable for browsing 3D models on the internet. Existing work in continuous level-of-detail is extended to allow for faster interpolation between LOD sequences and we present a new LOD control mechanism for maintaining a constant polygon count.

We present a generic framework generates multiresolution models through simplification. This allows for the comparison of surface compression methods under the same conditions, and to determine the performance of surface quality measures based on these results. These measures of surface quality are evaluated with both image and model based criteria. We find that the declining volume of a simplified object is a good method of predicting view-independent image quality. Using our generic framework, we extend two applications which can be used to improve rendering performance in a virtual environment.

We develop a new selective refinement application which refines only a desired region of the model, suitable for online model browsing. This method provides substantial space saving due to a more compact representation of the simplification hierarchy, and also provides optimisations for use with a client/server model. A novel method of defining smooth mappings between different resolution versions of a model (called continuous level-of-detail) is also defined. This technique greatly improves rendering performance of these models by employing commonly available programmable graphics hardware. We also present a method of controlling the number of polygons in large scenes, which is capable of predictively maintaining a constant frame rate by guaranteeing a polygon budget.

Chapter 1

Introduction

Surface meshes have become the medium of choice for representing objects, terrain and avatars in virtual environments. A surface mesh consists of a number of polygons stitched together to form an enclosing "skin" which represents the object. Issues such as the number of polygons drawn impact directly on the rendering time (and frame rate) of the scene. For this reason virtual environments (such as games) typically make use of simplified objects with low polygon counts.

Surfaces, such as those generated during the Michelangelo Project [LRG+00], can be simply too large to store or transmit, let alone render. The 3D surface of Michelangelo's David, intricately carved and over 3 meters tall, requires an astonishing 32 Gigabytes of storage, and consists of 2 billion polygons and 7000 colour images. A "virtual museum" consisting of exhibits of such works of art in a virtual environment would be of practical interest. Unfortunately, the storage and rendering requirements for such a scene makes it impossible on current graphics systems.

A commonly used method of controlling the frame rate of a rendered sequence is to use a level of detail (LOD) sequence. Each object in the scene is stored as a sequence of objects of reducing complexity. Interactivity is ensured by reducing or increasing the complexity of objects depending on their distance from the viewer, to ensure that fewer polygons are drawn in the scene. These changes can be visually distracting as models suddenly change to a lower or a higher level of complexity — a phenomenon known as *popping*.

At the heart of this problem is a fundamental trade-off between model quality and interactivity. Low polygon versions of models can distract users in a virtual environment, as they often consist of sharp edges and can appear unrealistic. Although recent advances in graphics hardware have seen significant improvements to rendering performance, polygon restrictions still apply to most applications due to the user requirement of interactivity.

Model quality and interactivity can be thought of as the two extremes on a sliding scale (like that of Figure 1.1). By making use of techniques of surface analysis (such as multiresolution analysis) we can define finer control over this trade-off. In this dissertation we develop measures and methods which can be used to gain better control of this sliding scale. Our main contributions are to tackle this problem from three directions:

- *Surface Simplification*: The number of polygons representing an object in a scene can be reduced. Using reduced polygon models is commonplace amongst most virtual environment applications. We evaluate simplification techniques based on the resultant image quality, to

Figure 1.1: Visual quality and interactivity on a sliding scale. As rendering times are generally dependent on the number of polygons in the scene, reducing the polygons used to represent an object results in an increase in the frame rate.

ensure that the visual distortion is minimised in compressed surfaces. Our finding is that image error can be closely and quickly approximated by measuring the volume of the model enclosed by surface. We discuss this further in Section 1.1 and Section 1.2.

- *Selective Refinement*: Only the region which is being viewed or selected is refined. This is useful for browsing surfaces which are too complex to render in their entirety. We adapt this technique to limited bandwidth and distributed systems such as the Internet. Our method of organising the mesh data offers significant space savings over other similar techniques. We introduce this in more detail in Section 1.3.

- *Continuous Level of Detail*: In order to remove visually distracting level of detail changes between polygon levels of detail, a mapping can be defined between them to allow smooth transitions. We present a hardware accelerated method of performing these transitions, and introduce a new technique for guaranteeing polygon limitations in complex scenes. This is further introduced in Section 1.4.

1.1 A Generic Simplification Platform

Many simplification techniques have been proposed based on the iterative removal of vertices. Essentially vertex removal is performed by applying simplification operations to localised portions of the mesh. Although fundamentally performing the same function, these techniques can differ in a number of ways:

- it may employ a distinct simplification operation to remove a vertex from the mesh;

- it must use an error metric to determine the ordering of these simplification operations;

- The position of the vertices in the affected region must be updated after simplification has taken place (this calculation is typically determined by the error metric used);

- it may present output configurations of the simplification results, such as the reversible progressive mesh format[Hop96];

- it may make use of optimisations which improve performance in terms of running time or memory overhead;

The most common simplification operation used is the edge collapse operation, which collapses a vertex pair into a single vertex. Most simplification techniques based on this operation differ in terms of the error metric used and a variety of memory-resident speed optimisations for error metric computation.

In Chapter 3 we present a simplification framework which is "memoryless", i.e. it does not make use of any memory resident simplification optimisations. It has already been shown that in some cases removing memory overhead does not reduce the quality of the simplified model. We show how several existing techniques can be converted into a memoryless form and integrated into the simplification framework. This has several implications:

Error metric comparison

Error metric comparisons based on quality are typically dealt with by using visual image comparisons and model deviation measurement computations performed on the models after simplification. Performance measures of simplification techniques are measured based on computation time, regardless of the excess hardware resources used to improve these results.

Comparisons of error metrics which are independent of algorithmic optimisations are not possible, since these optimisations make it impossible to use the same platform for different techniques. Due to the implementation specific nature of these simplification schemes, image comparison as a form of metric evaluation has only been performed visually on the simplified model produced.

We will show that a generic simplification framework permits simplification performance analysis without including memory-resident optimisations on the same platform. We also show that the quality of various error metrics can be compared using an image based comparison performed during simplification.

Adaptive Simplification

It may be desirable to change the metric used for simplification during the simplification process. An example of such an application would be one where an extremely dense model is simplified quickly to a simpler one with a rapid detail-reduction metric, after which a slower metric could be used, which is more sensitive to features or curvature.

We introduce a "batched" simplification ordering strategy, which allows the user to change the error metric adaptively during simplification. This process can also be automated, by determining the appropriate error metric based on the density of the points, or the relative curvature across the surface of the model.

Custom error metric design

Although general error metrics provide good results during simplification, they may not successfully deal with surface attributes such as normals, texture coordinates and colour values. Within the memoryless framework, new error metrics can quickly be devised and tested under the same conditions to determine which performs best with the model being simplified.

1.2 Assessing Error Metric Performance

In Section 1.1 above, we described how a memoryless simplification platform can be effectively used to compare error metrics in terms of performance analysis and image based comparisons. This gives us the opportunity not only to evaluate various error metrics in terms of their performance, but also allows us to draw conclusions about how surface simplification is evaluated.

It is difficult to define an effective measure of the deviation of a surface from the original model. Since models in three dimensions will be approximated on a screen in two dimensions, a two dimensional image-based comparison (from many viewing angles) would emulate how we would perceive error in the model. Unfortunately, an image based comparison measure has many parameters (the size of the image plays a large part in the magnitude of the error), and is difficult and slow to emulate on computers with no specialised rendering hardware.

A number of techniques are available to assess model quality (these are described in more detail in Section 2.5). Generally these model-based techniques are considerably easier to compute than image based measures, as the are independent of the graphics hardware. In Chapter 4 we evaluate a number of models during simplification with both image-based and model-based measurements. Our results show that the rate of decrease in model volume corresponds closely with our image-based error measures.

1.3 Selective Refinement and Transmission

Selective or view-dependent refinement is a method of adding detail to a model only in areas which are being viewed. This speeds up the rendering process, which is essential when browsing large amounts of data (for example terrain visualisation or medical imaging) or when bandwidth is restricted (in the case of web-based model browsing).

In Chapter 5 we present a number of extensions and improvements to existing view-dependent refinement algorithms and present a novel graph structure by which visibility is be determined. We also adapt this algorithm to the problem of progressive and selective transmission in an environment where the client is limited in its bandwidth and rendering capabilities.

We show that by exploiting coherence between client requests we are able to significantly limit the amount of information required to transmit to the client. We show that by maintaining a polygon budget the client is able to maintain the desired transmission and rendering performance.

1.4 Continuous Level-of-Detail Construction and Control

In large scenes consisting of many repeated models (such as a crowded street) it is not necessary to display all models at their highest resolution. A level-of-detail frame rate control mechanism reduces rendering complexity of distant objects by replacing them with a lower resolution version. This can give rise to unsightly "popping" between the discrete model representations.

"Continuous" level-of-detail provides a mapping for the vertices between the different levels of resolution. Unfortunately until recently real-time rendering hardware has been incapable of rendering these models at interactive frame rates. We introduce a novel structure in Chapter 6, the g-mesh. We describe its construction and use, and show how the performance can be greatly improved (over two hundred times faster) on existing graphics hardware.

We also present a reactive level-of-detail control mechanism for use with the g-mesh structure. This is facilitated by a global parameter which can be modified to increase or decrease the detail present in the scene. We show how the implementation can be modified to include simple animation.

1.5 Chapter Overview

In Chapter 2 we discuss some of the work which relates to ours in the fields of multiresolution analysis and surface compression, and define some of the terms which we will use within this dissertation. In Appendix A we include theoretical an implementation details of using wavelets for multiresolution analysis. We introduce our generic simplification platform in Chapter 3, and describe issues which need to be addressed in implementing a generic memoryless platform.

Models produced using different error metrics with our generic platform are then compared in Chapter 4 using visual and model-based measures. The results of these comparisons are analysed statistically to determine the significance. The results of the statistical analysis is included in Appendix B, the models used for the experiments are included in Appendix C and visual results are available in Appendix D.

We also use the output from the generic platform described in Chapter 3 to construct specific mesh formats for virtual environment applications. In Chapter 5, we use these meshes to build a hierarchy of surface information necessary to facilitate a client/server application for progressive and selective refinement. In Chapter 6 we describe a new method of constructing smooth geometric transformations between LOD models, and describe how these transformations can be performed in current graphics hardware. We also introduce a new greedy predictive LOD control algorithm which can guarantee the required polygon limitation.

Chapter 2

Background and Related Work

This Chapter serves as an introduction to the field of Multiresolution analysis, and describes work relating to the applications presented in this dissertation. We distinguish between multiresolution analysis schemes which depend on local or global connectivity, and define applications of each method.

After defining specific terminology necessary for describing mesh features, we focus on multiresolution methods depending on local connectivity. Techniques of generating multiresolution surfaces with local connectivity are described, as are error measures used for surface compression. We also discuss methods of surface evaluation, and previous work in the field of surface quality evaluation. We then detail related work in the fields of view-dependent and selective refinement as they relate to our work. Finally related work in the field of continuous level-of-detail is introduced.

2.1 What is Surface Compression?

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. Commonly in graphics hardware, surfaces are represented by a piecewise discrete approximation, consisting of interconnecting polygons (called a *mesh*). This can be thought of as a quilt, where each triangle is a patch (polygon) which is sewn (connected) to it's neighbours across the edges of the patches. Polygon meshes can be constructed out of any convex polygon, although the simplest and most common is the *triangle mesh*.

Certain information is required in order to represent a polygonal mesh:

1. *geometric information* represents the locations of vertices in a model,
2. *connectivity information* details how these vertices fit together (e.g. triangle strips or indexed face sets), and how these patches are constructed, and
3. *attribute information* such as normals, vertex colour or texture patch coordinates.

Lossless compression (i.e. techniques which compress a model without discarding any information) must make use of redundancies in these types of information. As an example, Huffman encoding could be used to compress the geometric information directly, or a surface could be reorganised into a sequence of triangle strips to reduce redundancies in storage of indexed triangle sequences.

Rossignac[Ros99] compresses mesh connectivity by eliminating redundancies in a triangle mesh representation, while Taubin *et al.* [TR98] apply an efficient connectivity ordering strategy to reduce connectivity information. Later techniques, such as Pajarola and Rossignac[PR00] and Alliez and Desburn[AD01b], are able to compress models losslessly to less than a half (in the case of [AD01a] a tenth) of its original raw data size. Extensions to these techniques [PR00, AD01a] allow these compressed models progressively. A comprehensive survey of direct information compression is available from Rossignac[Ros99].

Performing wavelet-based multiresolution analysis[SDS95] on surfaces allows vertices to be classified in terms of their contribution to the overall model. These vertices can then be removed based on their classification. This technique obviously incurs a certain measure of error to the representation of the model, but, since the error associated with each vertex is known, we have direct control over the error of the viewed model. This observation is the basis of multiresolution analysis of surfaces.

2.2 Multiresolution analysis

A common technique used in real-time rendering optimisation is to use a pre-created hierarchy of surface representations of a model at different resolutions[Cla76]. Objects which are less important to the scene (such as those which are obscured or far away) are replaced with lower resolution versions. This technique is commonly referred to as level-of-detail or LOD. It was first proposed by Clark[Cla76], and has been extended to include techniques for model selection based on cost-benefit heuristics[FS93, Mas99].

Obvious deficiencies with LOD representations of models are their increased storage, and objectionable "popping" caused when there is a sudden transition between two levels of detail. This is addressed by using "continuous" LOD representations [Gar99], such as geomorphs [Hop96] or smooth wavelet coefficient interpolations [SDS95]. The desired result is a model that encodes multiple levels of detail efficiently, while allowing for smooth transitions between representations. multiresolution analysis can produce such a representation.

Multiresolution methods refer to techniques which create and utilise various *hierarchical representations* of functions [SDS95]. The development of these methods for function decomposition was necessitated by deficiencies in traditional Fourier techniques, and make finite support and compact representations of functions possible. Multiresolution techniques have been applied to the fields of approximation theory, physics and signal and image processing[Mal89, Dau92], but only in the last decade to surfaces.

We define a mesh M^i to be a set of p indexed *faces* $\mathcal{F} = \{f_1, \ldots, f_p\}$, and a set of q indexed *vertices* $\mathcal{V} = \{v_1, \ldots, v_q\}$. We define the original model as $\hat{M} = M^n$. Multiresolution analysis of surfaces identifies a hierarchy of models $M^0 \subset M^1 \subset M^2 \subset \ldots \subset M^j \subset \ldots \subset M^n$, where j denotes the level of resolution of the original surface, and M^0 represents the *base mesh* or *control mesh*, at the coarsest level of detail. In this case, $M^i \subset M^{i+1}$ implies that all vertices in model M^n are also contained within the model M^{i+1}.

Locality of Connectivity

We distinguish between multiresolution techniques by their approach to constructing the hierarchy of multiresolution representations. An approach in which each new level of the analysis M^{i+1} is dependent on all geometric and connectivity information in the previous level M^i we call multiresolution analysis using *global connectivity*. Although wavelet analysis schemes have compact support, wavelet [Lou95] implementations are a common example of global connectivity, where there is a relatively small number of analysis levels (this is further described in Appendix A).

If the construction of level M^{i+1} is only dependent on a finite subset of the connectivity information stored within the previous model M^i, such as the presence of a single vertex, we refer to this as multiresolution analysis using *local connectivity*. The work of [Hop96] is an example of such a dependence, where the difference between M^{i+1} and M^i is that only a single vertex and two faces are inserted. We consider hierarchical decimation techniques as local connectivity multiresolution schemes.

Applications of Multiresolution Analysis

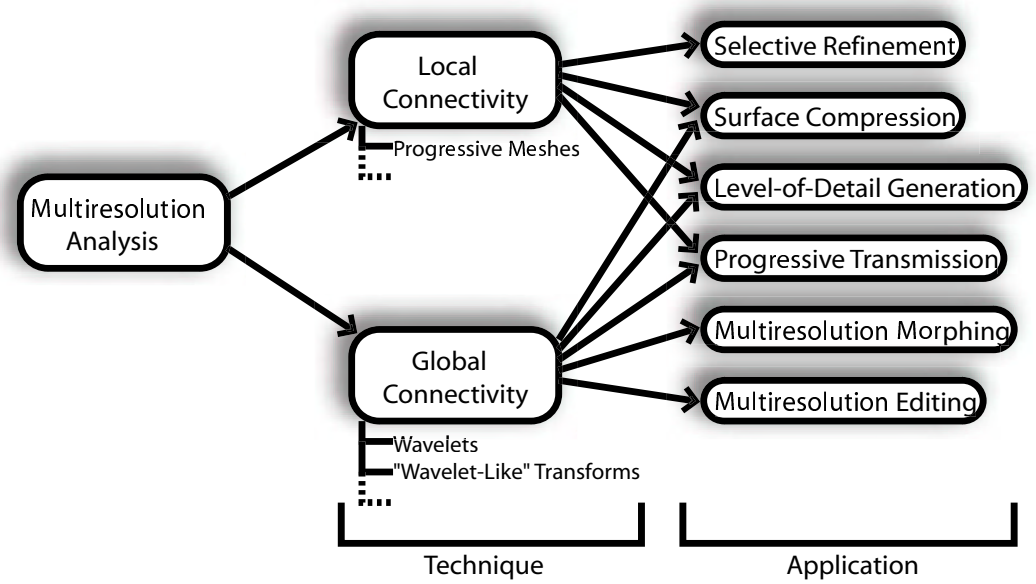


Figure 2.1: A summary of multiresolution analysis schemes and applications. This graph depicts a simplistic partitioning of the techniques by which multiresolution meshes can be built, and the applications that are available to each technique.

Several procedures have been developed for the creation of a multiresolution framework in three dimensional computer graphics, but the essential principle remains the same throughout — to represent a model (typically a triangle mesh) using a hierarchical decomposition at various levels of resolution. This has direct applications in the following fields (see Figure 2.1):

- *level-of-detail control* — a hierarchy of level of detail models can quickly be generated using multiresolution analysis and surface compression;
- *progressive transmission* — instead of transmitting the entire surface, a much smaller and

simpler version of the mesh could be transmitted, and afterwards a sequence of refinements. This greatly speeds up the task of browsing models online;

- *surface compression* — as mentioned before, models can be compressed by iteratively removing detail;
- *multiresolution editing* — models can be edited at various levels of resolution, allowing for greater control during the editing of a surface[ZSS99];
- *selective refinement* — only the selected region of the model, such as the region being viewed, needs to be refined, improving viewing and transmission interactivity[Hop97, XESV97];
- *multiresolution morphing* — models can be "morphed" or smoothly transformed from one shape to the other by using multiresolution techniques[LDSS99].

2.3 Multiresolution Analysis using Global Connectivity

A wavelet or "wavelet-like" transform is commonly used for generating multiresolution representations. Broadly speaking, a wavelet representation of a function is a coarse overall approximation together with detail functions at various scales. The original surface can be reconstructed using only the *base mesh* M^0 and by consecutively applying i sets of detail functions the model can be reconstructed to level M^i.

These coefficients are a measure of the "importance" of each vertex to the model, or rather the error incurred by it's removal. By reducing coefficients of the lowest magnitude to zero iteratively the vertices of least error are removed, and the model's detail is reduced. Wavelets lend themselves naturally to progressive transmission, where coefficients are transmitted in the order of their magnitude. For more details regarding the applications of wavelets of multiresolution analysis, the reader is referred to [SDS95]. In Appendix A we discuss some of the theoretical and implementation details of using wavelets for multiresolution analysis.

2.4 Multiresolution Analysis using Local Connectivity

Local connectivity compression schemes typically do so *progressively*. An atomic operation is an operation applied to a model which affects a small, finite subset of the models connectivity and geometric information. We refer to techniques which use these operations as "atomic" multiresolution analysis or *decimation*. Each level M^i only differs from it's predecessor M^{i-1} by the application of a single atomic operation.

The algorithm governing iterative model simplification is fundamentally the same throughout the techniques we describe here:

1. define candidates for the atomic operation,
2. sort the candidates on some criteria (typically some error which is incurred after a vertex removal),
3. perform the atomic operation, resulting in the removal of surface detail,

4. update the remaining candidates in the list, and

5. if there are still candidates, loop to step 3.

Decimation Techniques

A broad spectrum of atomic operations are presented in [SZL92, HDD+93, Hop96, PH97, COLR99, LT98, EM99], each of which can be used within a decimation framework. Two commonly used decimation techniques are shown in Figure 2.2. In this section, different frameworks for decimation will be outlined.

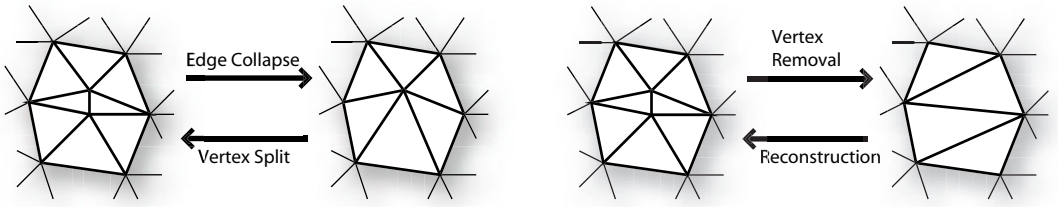


Figure 2.2: Two different atomic operations. In **(a)**, an edge collapse and corresponding vertex split operation are shown[Hop96]. The two circled vertices are removed or created in each case respectively. **(b)** shows an example of Delaunay triangulation, in this case six faces are reduced to four[Bon98].

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. While these properties may be desirable, they often have an associated cost. These techniques must make a number of trade-offs:

- *memory usage and speed* — storage of mesh attributes and simplification information can improve on the running time of the model simplification. As an example, Garland and Heckbert [GH97] store a quadric matrix at each vertex of the surrounding faces, while Hoppe [Hop96] makes use of the original model to determine the deviation of the current model;

- *model size and quality* — the resultant progressive mesh format can be compressed by using certain geometric attributes based on the position of the vertex. Hoppe [Hop96] uses a subset placement strategy, while Pajarola *et al.* [PR00] use a statistical model to predict the position of the vertex. Techniques using "optimal" vertex placement such as [GH97, Hop99, LT98] do not restrict the location of the vertex, and require more storage to reconstruct.

We require specific terminology to define vertices and faces which are used during atomic decimation. Since we reuse the vertex index of one of the vertices in the region, we refer to the vertex we *keep* as v_k, (see Figure 2.3) 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 position does not change during decimation. In Figure 2.3 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 2.3 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 a clockwise manner about it's focus point (v_k or v_l) from f_s or f_e respectively.

Vertex removal techniques differ fundamentally in three respects:

- *atomic operation* — refers to how the model is simplified, and how the faces and vertices of the mesh are affected after simplification;
- *vertex placement* — refers to how the geometry and connectivity is updated after an atomic operation has been performed;
- *error metrics* — refers to how the operations are ordered. The ordering is determined by the error incurred after the application of the atomic operation.

2.4.1 Atomic Operations for Decimation

We say that an atomic operation is *valid* if it can be performed given the current mesh configuration. In [Tur92, HDD+93] the validity of each operation is determined with respect the overall mesh through linear optimisation. Although [HDD+93] produces a good simplified representation of the original surface[CMS98], the procedure is extremely costly in terms of computational overhead, due to the dependence on the original model during decimation.

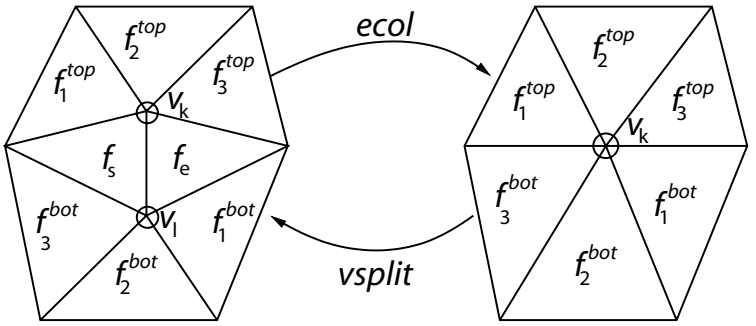


Figure 2.3: The edge collapse / vertex split. The vertex v_l and the edge between v_l and v_k are removed from the mesh after the edge collapse *ecol* has been applied. The inverse vertex split *vsplit* operation reintroduces these attributes into the mesh. The faces marked $f_i^{top}, i = 1 \ldots 3$ represent faces in the $\mathcal{TOP}$ fan domain, while those marked $f_i^{bot}, i = 1 \ldots 3$ are faces in the bottom fan domain. f_s and f_e refer to the *start* and *end* faces which will be removed respectively.

Hoppe [Hop96] realized that the work of [SZL92, HDD+93] could be modified so that the operations responsible for simplification could be applied inversely to recover the original surface. Progressive Meshes (PM), introduced by Hoppe in [Hop96], define a mesh representation where each simplified level M^j only differs from it's predecessor M^{j+1} by a single edge collapse operation $ecol_j$ (as shown in Figure 2.3). The original model can then be losslessly reconstructed from level M^j to M^{j+1} by using the inverse operation $vsplit_{j+1}$. Hence the multiresolution representation becomes

$$M^n \xrightarrow{ecol_{n-1}} M^{n-1} \xrightarrow{ecol_{n-2}} M^{n-2} \xrightarrow{ecol_{n-3}} \ldots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0$$

$$M^n \overset{vsplit_n}{\longleftarrow} M^{n-1} \overset{vsplit_{n-1}}{\longleftarrow} M^{n-2} \overset{vsplit_{n-2}}{\longleftarrow} \ldots \overset{vsplit_2}{\longleftarrow} M^1 \overset{vsplit_1}{\longleftarrow} M^0$$

and a Progressive Mesh is defined by $PM = \left\{M^0, \{vsplit_1, \ldots, vsplit_n\}\right\}$.

The Progressive Meshes of Hoppe only consider a half-edge collapse, where v_k and v_l fuse to the original position of one of these vertices, or a collapse to the midpoint of these two points. The edge collapse operation supports a number of different vertex placement strategies. 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.*[PR00]. When applied to largely convex surfaces this vertex placement strategy 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[Hop96] 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.

- *Unconstrained placement*: The point position can be determined using optimisation. Unlike the above two techniques, the resultant point can lie anywhere. The criteria for optimisation varies from the distance from the nearby planes[GH98] to volume and triangle shape preservation[LT98, Hop99]. This technique requires the storage of both δv_k and δv_l, as the new vertex position is unconstrained.

Up to this point all multiresolution analysis techniques strictly preserve the topology of the original surface. Although often thought of as a desirable property, topologically complicated surfaces, or a collection of several surfaces may have a large M^0 representation. Popovic and Hoppe [PH97] and Garland and Heckbert[GH97] introduce a vertex unification operation. This allows the simplification of mesh topologies, such that surfaces of any topology can be simplified down to a single point.

Hoppe[Hop96] introduces the concept of *selective refinement*. Since atomic operations only refine a small area of the model, a hierarchy can be constructed of operation dependencies to refine only a small region of the model. This relationship has been utilised in the work of [Hop97, XESV97] to create a framework allowing for real-time regional refinement, suitable for view-dependent refinement. This is further discussed in Section 2.6.

In order to allow for the compression of models which are too large to store in main memory, Lindstrom [Lin00] introduces a technique to facilitate *out-of-core simplification*. The model is subdivided into separate segments, and each segment is processed individually. Large models

generated using laser scanning, such as those of the Michaelangelo project[LRG+00], are impossible to simplify progressively without out-of-core techniques.

2.4.2 Generic Simplification

Kobbelt *et al.* [KCS98] introduce a "generic" simplification algorithm. They divide simplification criteria into *distance measures* — which attempt to minimise 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[GH97] or explicitly[Hop96, Hop99]. 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.

2.4.3 Batched Operations

Gúeziec *et al.*[GTLH98] 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.

In Chapter 5 we introduce a new batching framework which 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 3.4), as the priority queue of simplification operations is empty. This becomes useful in Chapter 6 where it is important to restrict the number of levels-of-detail to the minimum.

2.4.4 Error Metrics for Decimation

Central to the resulting quality of models analysed by local decimation techniques is the error metric used to determine the locations of the vertices after each iteration. Hoppe *et al.* [HDD+93] use three error terms in order to ensure that the simplified model retains its shape to a certain degree — E_{dist} measures the distance of the introduced point from the original surface, E_{rep} is a user constraint on the level of simplification of the surface, and E_{spring} restrains the length of the resultant edges to remain proportional to each other.

Although producing very good results, the error metric defined in [HDD+93] requires user intervention (in the construction of E_{rep} as well as the coefficient of the E_{spring} term). The E_{dist} term requires additional memory overhead, due to the necessary storage of prior mesh information to determine the deviation from the original model.

Hoppe [Hop96] makes several significant changes to the metric proposed in [HDD+93]. The E_{rep} term is removed, and a new term E_{scalar} is added to efficiently deal with surface attributes, and handle surface discontinuities. The E_{dist} term now compares the error incurred by removing

the vertex only within the region local to the decimation. Hoppe also introduces a smooth geometric transformation of these operations called "geomorphs" (*geomorph*: *geo*metry *morph*) which smoothly interpolate vertices to their final positions.

Garland *et al.* [GH97] attempt to introduce an unconstrained placement policy for the new vertex by minimising 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 colour values and surface normals[GH98]. Results from this technique are impressive and extremely fast, despite the necessity of calculating a matrix inverse at each stage, and volume is better preserved than in [Hop96] above.

Lindstrom *et al.*[LT98] introduce the concept of *memoryless simplification*. Memoryless simplification does 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 optimising 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 optimisation.

Hoppe [Hop99] applies the principle of memoryless simplification and a volume post-process to improve upon the work of [GH98]. Hoppe[Hop99] also reduces the computational complexity required for the error metric, and improves scalar attribute preservation. Memoryless simplification is advantageous over techniques which use simplification "history", as it requires a smaller memory overhead. This can be significant when simplifying large models. Hoppe[Hop99] also finds that a memoryless version of the quadric error metric produces better results than one which uses the additional memory overhead.

Lindstrom *et al.*[LT00] present a novel method of simplification which uses a metric derived from the comparison of two-dimensional images of the surface to infer the importance of a vertex. It produces models on which the texture quality of the surface is well preserved, and, not surprisingly, silhouettes are highly detailed in compressed models. The technique is strongly dependent on both camera placement relative to the object and the lighting conditions. This is clear from the results in which regions which are hidden from camera view frustum are highly decimated. We similarly make use of images in order to determine the quality of simplified models in Chapter 4, and to assess the quality of existing simplification schemes.

2.5 Evaluation of Simplification Errors

In order to measure the accuracy of simplification schemes, models are commonly evaluated using geometric comparisons. For a geometric evaluation function K, evaluations can take place in either of the following manners:

- $K(\hat{M}, M^j)$, a comparison of the current simplified model with the original model, or
- $K(M^{j-1}, M^j)$, which is a comparison of the current simplified model with the previous simplified model.

Although a comparison with the original model $\hat{M}$ is desirable, it is sometimes unavailable due to memory or speed considerations. For example, Lindstrom and Turk[LT98] use a memoryless

simplification technique, which does not depend on the original model.

2.5.1 The Hausdorff Distance

For Medical Imaging and Industrial Design applications the accuracy of a surface approximation is directly dependent on the the spatial deviation of the simplified model from the original. In these professional applications it is essential that there is exact control of the error of the simplified surface. The maximal displacement of one surface from the other measures this deviation, and is called the *Hausdorff Distance.*

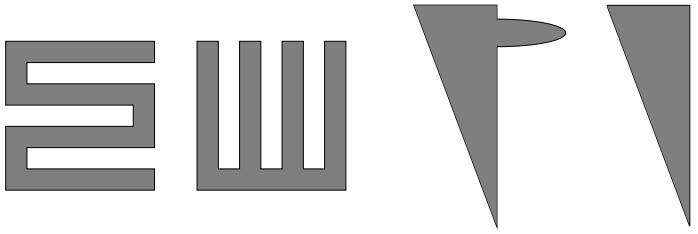
We define the Hausdorff distance (sometimes called the L^∞ norm or maximal difference) between two input meshes M^1 and M^2 as

$$K_{haus}(M^1, M^2) = \max\left(\mathbf{dev}(M^1, M^2), \mathbf{dev}(M^2, M^1)\right), \quad \text{where}$$

$$\mathbf{dev}(A, B) = \max_{p \in A}\left(d(p, B)\right)$$

measures the deviation of mesh A from mesh B, and $d(p, M)$ represents shortest distance between point p and the surface M. It measures the *worst case distance* that a point on one of the surfaces would have to travel to reach the other surface.

The Hausdorff distance provides a maximal geometric deviation between two shapes, and is view-independent. The Hausdorff always captures the worst case situation, where the ray cast from one mesh intersecting the other is orthogonal to the view direction. Although a exact measure of model locality for professional applications, it is not necessarily a good measure of shape similarity[Ros97]. Some examples of undesirable Hausdorff errors are depicted in Figure 2.4. The Hausdorff distance is also extremely slow to compute, as it is dependent on the number of triangles within both meshes. Klein *et al.*[KLS96] outline a method for controlling the Hausdorff error dur-



(a) (b)

Figure 2.4: The Hausdorff Distance. Two pathological cases where the Hausdorff distance may give misleading results. The two shapes in (a) are quite dissimilar, but the Hausdorff distance measure will give a small result due to the similar overall shape of the model. In (b) the bump will cause two otherwise identical models to have a large Hausdorff distance.

ing surface simplification, reducing the computational overhead of evaluating the resultant surface quality. Cignoni *et al.*[CMRS98] distinguish between positive and negative maximal error, as well as mean error (L^1) and mean squared (L^2) error in their surface comparison tool Metro. Using an efficient method of surface partitioning and optimisations they are able to approximate model error in tractable time. The Metro tool has been used to compare the compression of several compression algorithms[CMS98] but provides no statistical interpretations of the results.

2.5.2 Triangle Quality

In order to minimise lighting artifacts caused by per-vertex lighting (Gouraud shading) it is desirable to make triangle faces in the mesh as equilateral as possible. Figure 2.5 highlights errors incurred by triangle "slivers".

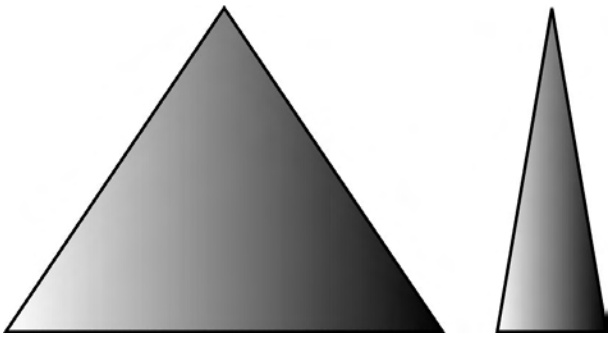


Figure 2.5: Triangle shape preservation. Per-vertex lighting is usually determined independently from the shape or size of the triangles constituting the vertex normal. This can cause unsatisfactory lighting artifacts, as the lighting value is interpolated across the surface with Gouraud shading.

Several techniques try to ensure that the resultant triangles are not degenerate. Hoppe *et al.*[HDD+93] regulate their optimisation problem by placing springs at rest at across each edge of the mesh. The tension of these springs penalises edge collapse operations which result in excessively long edges, but do not explicitly attempt to equalise triangle shape for each operation. Lindstrom and Turk[LT98] introduce a triangle equalisation term into their optimisation by determining the sum of squared lengths of edges incident on the new (optimally placed) vertex.

Frey and Borouchaki[FB97] measure the mean quality of the triangles within a face set $\mathcal{F}$ of mesh M^j as

$$K_{Tri}(M^j) = \frac{1}{|\mathcal{F}|} \sum_{f \in \mathcal{F}} Q_f, \quad \text{where}$$

$$Q_f = \frac{6}{\sqrt{3}} \frac{S_f}{p_k . h_k}$$

defines the quality or *aspect ratio* of a given triangle f. S_f is the area of face f, p_k is the half-perimeter of f and h_f is the longest edge of f. The term Q_f returns a value between 0 (flat) to 1 (equilateral triangle), while K_{Tri} is just the average of these values across all faces in the set $\mathcal{F}$. We derive a different triangle deviation measure in Chapter 4 based on the ratio of each of the edges of a face to the shortest edge in that face.

2.5.3 Volume

Several recent techniques[LT98, Hop99] use volume preservation as an approximation for the deviation of the current model from the original surface. Intuitively, the volume provides a less accurate form of error measure for model comparison, as shown in Figure 2.5.3. A measurement of the deviation of model volume is also provided in the surface comparison tool *Metro*[CMRS98]. We will, however, show in Chapter 4 that volume is a good measure of image deviation during simplification. This is an advantageous result, as volume is a quick metric to compute, and correlates significantly better with image deviation than the Hausdorff distance.

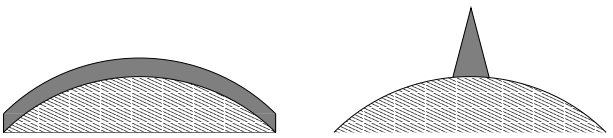


Figure 2.6: Hausdorff Distance vs Volume Difference. In this pathological case, the image on the left would yield a relatively low Hausdorff distance, while volume difference is relatively high, as a large portion of the model is slightly shifted. The image on the right yields a relatively high Hausdorff error while the volume difference might be low.

2.5.4 Visual Comparison

Watson *et al.* [WFM01] evaluate the quality of simplified surface meshes through user experiments. They compare the method of [RB93] and [GH97] using user evaluations by using naming times (the time taken to identify the object) and a simple rating of the objects quality. A large number of experiments yielded no significant results, although users did tend to prefer [GH97] to [RB93]. No conclusions could be drawn about correlations between error measures (such as *Metro*[CMRS98]) and the visual measurements.

There are a number of shortcomings of the visual experiments performed in [WFM01]. The distinction between "natural" and "man-made" models is a arbitrary one, since models are typically sculpted using the same modelling tools. To our knowledge, there is no evidence to show that we recognise these objects differently. Deciding on model quality based on only a single viewing direction may be misleading, although the number of tests required to evaluate all viewing directions is prohibitive. The evaluation of flat shaded models limits the application of the results attained in [WFM01] to flat shaded scenes, although very few of these still exist in Virtual Environments.

In our experiments (in Chapter 4) we evaluate multiple viewpoints of simplified models. Like Watson *et al.* we evaluate un-textured models, thereby determining the visual quality of the geometry of the model. We use smooth shading for our image evaluations, as most models will be viewed in this way.

2.5.5 Image Comparison

Lindstrom and Turk[Lin00] make use of image comparisons in order to determine what to simplify. Taking evenly spaced image captures about the model, they compare these images to the original surface using the L^2 image difference. Vertices are weighted according to this comparison, and areas of the surface which are obscured or hidden are heavily simplified.

It has been shown experimentally[DJL92] that for image compression the error incurred should be measured in the integral sense (L^1) rather than the mean-squared (L^2) sense. Intuitively a higher norm rates higher deviations with a more significant weighting. For this reason the commonly used L^2 error may produce misleading error values. To our knowledge, no one has yet used images to evaluate the quality of the surfaces resulting from simplification techniques. In Chapter 4 we make use of both the L^1 and L^2 metrics to compare image error to geometric error measures.

2.6 Selective Refinement

As in [XESV97, PR00] we define the *region of overlap* of an individual $ecol_i$ transformation as every edge originating from the base points of the region defined by $ecol_i$, as in Figure 2.7. We also define the *edge collapse domain* as the set of all faces affected by the edge collapse, i.e. $\mathcal{ECD} = \{\mathcal{TOP}, \mathcal{BOT}, f_s, f_e\}$.

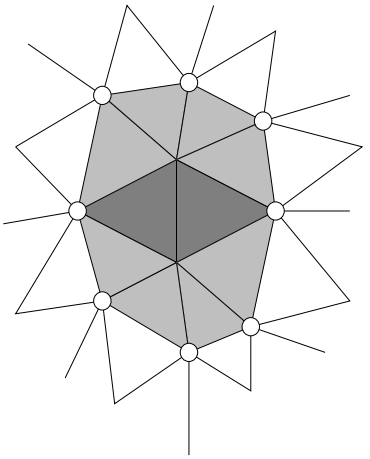


Figure 2.7: The region of overlap of an individual edge collapse operation. Circled vertices indicate the basepoints of the region. The two dark shaded triangles indicate faces which are to be removed, lighted shaded triangles indicate triangles in the face sets $\mathcal{TOP}$ and $\mathcal{BOT}$. The remaining edges are those originating from the base points of the region.

View dependent refinement schemes are distinguished by the way in which the operations are ordered, and the determination of which operations to apply. In [Hop97, TLG99] a *vertex hierarchy* is constructed as a tree where every split vertex v_k is represented by a node where the left child of that node is the inserted vertex v_l and the right child represents the split vertex at it's new position v'_k.

Xia *et al.*[XESV97] construct a *merge tree* during the simplification process, bottom up, by inserting the vertices present in the final mesh $\hat{M}$ as leaf nodes. A subset of edge collapse operations are applied to these vertices to produce a higher level of vertices in the tree. Luebke *et al.* [LE97] uses an octree of clusters of vertices and faces in order to collapse vertices together, allowing topology independent simplification.

Guéziec *et al.* [GTLH98] make use of a directed acyclic graph (DAG) to store partial ordering of edge collapse operations based on their dependencies. Floriani *et al.*[FMP98] use a DAG to order their multi-triangulation framework which allows for selective refinement in a client-server architecture.

The hierarchy of Xia *et al.*[XESV97] constructs the dependencies between the vertices in the merge tree on the premise that for edge collapse $ecol_i$, any $ecol_j$ collapsing edges in the *region of overlap* of $ecol_i$ are dependent on $ecol_i$. Like Hoppe [Hop97] we find this results in an unnecessarily deep tree, as refinements are seldom localised to the region selectively refined (especially in smaller models). To reduce the number of dependent faces, and hence the depth of the tree, we require only the presence of the faces neighbouring the face(s) being inserted/removed and the vertex v_k (according to Figure 2.3).

Spatial subdivision for selective refinement on arbitrary surfaces has been performed with octrees [LE97], bounding spheres [Hop97] and using subdivision (such as models provided by [LSS+98]). Atomic operations cannot be applied to subdivision surfaces without losing subdivision connectivity. While octrees provide rapid query times for spatially unrelated objects, a hierarchy of bounding spheres requires considerably fewer tests to be made when there are dependencies between objects. We use a bounding sphere hierarchy as it is also comparatively quick to construct.

To *et al.*[TLG99] provide a platform for progressive, selective (or view dependent) refinement by constructing a vertex hierarchy (similar to [Hop97, XESV97]). The hierarchy stored on the server also contains the triangle fans of the faces surrounding the vertices at various resolutions. These surrounding triangle fans are transmitted to the client. In the event of an overlap of triangle faces, the triangles of the *highest resolution* are chosen. This technique provides real-time adaptive and progressive view dependent refinement, and strictly refines only the triangles within the view frustum (in [XESV97, Hop97] vertex dependencies typically extend beyond the view frustum).

Because reconstruction is patchwork in nature, the client would not be able to make use of smooth transformations (geomorphs) between different levels of resolution. A change of the clients selection would also imply the previously refined area would remain refined. This may cause problems when browsing large models online on low-end workstations. The technique of To *et al.* provides a effective method of selective transmission, but is not practical for the online browsing of large model repositories (such as models from [LRG+00]) on low end machines. Our hierarchy is considerably smaller (±60% of the size) than the vertex hierarchy used by Hoppe[Hop97] and To *et al.*[TLG99]. We address poor rendering performance by allowing the user to restrict the number of polygons which are refined.

Floriani *et al.*[FMP98] present a framework for regional refinement of models based on multi-triangulations. Although presented for a different type of multiresolution representation, the algorithm used is very similar. They present a directed acyclic graph (or DAG) structure built on the multi-triangulation operations, and present a client-server architecture for performing selective transmission. Both the server and the client maintain a cache of transmitted operations to greatly reduce the transmission required to re-refine a previously refined region. Our work differs from that of Floriani *et al.* in that we introduce a method to determine visibility of an object in the view frustum. Our work was developed independently from that of Floriani *et al.*.

2.7 Smooth Geometric Transformations ("Geomorphs")

Surface blending can be defined as any technique which attempts to reduce the visual affect of transitions between discrete multi-resolution models. Techniques of surface blending can be divided into those which deal with smooth terrain transitions[LKR+96, TB94, COL96, FEKR90] and more general model-based blending techniques[Tur92, FS93, Lou95, SDS95, Hop96].

Funkhouser and Séquin [FS93] describe a technique to blend between two independent level of detail representations by using alpha blending to "phase-out" the old model and "phase-in" the new model. However, Ferguson *et al.* [FEKR90] claim that such blending techniques are visually distracting when used with adjacent level of detail models. Graphics toolkits such as IRIS Performer[RH94] and Renderman[Ups90] provide facilities for surface blending techniques by

making use of hardware alpha blending and interpolation between level of detail models.

Model blending techniques on terrain models make use of heuristics based on the special surface connectivity. Terrain models are typically represented by regularly spaced height-fields. Translating these height-fields into a triangle mesh results in a surface where every vertex (except those at the edges) has a valence[1] of exactly six.

Taylor and Barret [TB94] make use of the inherent two dimensional nature of height fields to construct a quad-tree representation of the model, and base their Triangulated Irregular Network (or TIN) on the cells of the quad-tree. Model blending is achieved by vertex interpolation between levels in each quad-tree cell.

In order to guarantee spatial and temporal continuity in the Delaunay triangulated terrain, Cohen-or and Levanoni [COL96] introduce a hierarchical Delaunay representation which allows for smooth transitions between different triangulations. The technique employed bears a resemblance to the method of Hoppe[Hop96], in that edges are collapsed iteratively in the re-triangulated region.

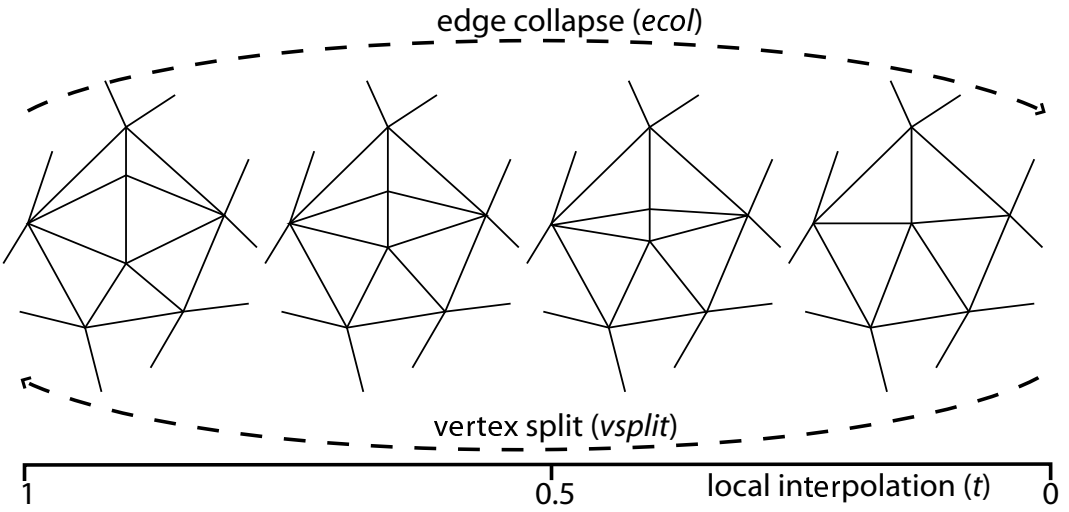


Figure 2.8: Interpolation across a vertex split / edge collapse operation. The value for t interpolates the positions of the vertices making up the edge being split or collapsed.

Lindstrom *et al.* [LKR+96] use the regular nature of these meshes to construct a hierarchical representation of the terrain model through subdivision. Simplification is defined as triangle fusing, where two adjacent triangles are merged into a single triangle by removing a shared vertex. These transitions can be performed smoothly between different levels of the hierarchy. Lindstrom *et al.* also define four types of continuity with respect to continuous level of detail across a terrain model:

1. *Geometric Continuity:* The function $z(x, y, t)$, where $x, y, t \in \mathbb{R}$, is continuous for every point in the mesh. In this case the parameter t refers to any scalar quantity used to morph the vertices into their new positions, and could represent distance, time or view parameters. This implies that every point in the mesh has *positional* C_0 continuity.

2. *Block Continuity:* For terrain models, neighbouring cells of the model must align so that the terrain model does not appear to have cracks in the surface. This differs from Point 1 in that

[1] The *valence* of a vertex is defined as the number of faces which include that vertex

a terrain model may consist of unconnected patches where points may be duplicated on the surface.

3. *Rendering Continuity:* The number of polygons sent through to the graphics pipeline is inherently discrete. However, Lindstrom *et al.* state that for this kind of continuity, *a sufficiently small change in view parameters results in the number of rendered polygons increasing or decreasing by at most one.*

4. *Polygon Density Continuity:* The number of polygons used to describe an area with respect to the view parameters is "continuous", i.e. the density of polygons over an area remains continuous according to a particular view point.

Hoppe generalises the problem of smooth model transitions between levels of detail to arbitrary surfaces for use with the progressive mesh structure. He defines a *geomorph* as a geometric transition between two level of detail representations, and defines the progressive mesh structure as a base mesh and a sequence of refinements necessary to reconstruct the original model. Progressive mesh refinements are in the form of a vertex split, which can be animated by interpolating the new vertex positions over time (see Figure 2.8). Multiple independent geomorphs can be performed at the same time, allowing for a large number of refinements to occur simultaneously. In order to determine which vertex splits can be performed simultaneously, the list of available vertex splits must be traversed to determine their validity.

Reddy [Red97] present an overview of factors upon which a LOD control system could be based, including the size, distance or velocity of the model, the viewing direction of the user and frame-rate control. Gobbetti and Bouvier [GB99] introduce a general LOD control mechanism for determining scene representations in time-critical systems. Their method, based on quadratic optimisation and the visual error measures of Reddy [Red97], provides a solution to LOD control in large scenes consisting of many multiresolution models.

Although the authors state that geomorphs can be implemented within their technique, they do not discuss the overhead of computing smooth geometric interpolations between frames of their scene. We show in Section 6.4 that for large scenes this computational overhead proves to be prohibitively costly. Their method of LOD selection could be used with the g-mesh structure as an alternative to the LOD control technique presented here.

2.8 Summary

In this chapter we have described the background and related work. The topic of multiresolution analysis was introduced, and techniques of producing multiresolution meshes are divided into two categories. Global connectivity describes a broad class of simplification techniques where each level of resolution depends on the connectivity and geometric information contained in the entire mesh (as is the case when using wavelets for multiresolution analysis).

In this dissertation we develop the field of local connectivity, where each level of resolution depends only on a subset of the geometric information contained within the preceding level. We have described error metrics which are used within the field, and discuss methods by which they are evaluated. The background of two applications of multiresolution analysis, namely selective refinement and continuous level-of-detail control, is also described.

Chapter 3

Generic Memoryless Polygonal Simplification

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" [SZL92].

A common theme in these simplification techniques is the localisation 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 (defined in Section 2.4). 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 testbed for the creation of custom error metrics, and we present two new error metrics designed within our generic platform.

3.1 Method Overview

We present a 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 neighbours of each face must be stored (as in [Hop98]), in order to speed up traversal about faces of the mesh. Compression is initialised 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 3.4). 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.

3.2 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[Hop96] (Section 3.2.1) and Garland *et al.*[GH97] (Section 3.2.2) 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 3.2.3) is a simple measure which can be used to regularise a mesh in terms of triangle area, and a hybrid scheme (defined in Section 3.2.4) which preserves normal attributes and mesh volume.

3.2.1 Progressive Meshes (E_{pm})

Hoppe[Hop96] 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 colour, 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[Hop99] 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 [Hop99] 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

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

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

Garland and Heckbert[GH97] determine the position of v'_k by minimising 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 during 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 [Hop99] finds that a memoryless version of the quadric error metric produces better results than the memory resident equivalent.

3.2.3 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.

3.2.4 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:

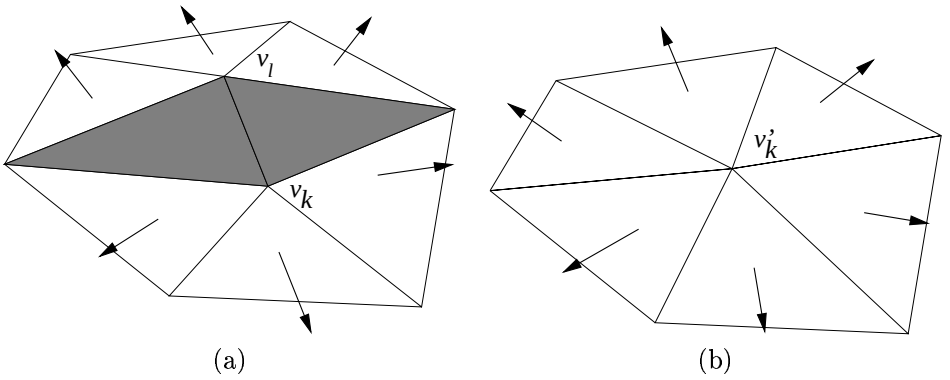


Figure 3.1: The derivation of E_{norm}. In (a) the normals of the original region **o** are shown, while in (b) the new normals **n** (i.e. after the edge collapse) are shown.

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

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 normalised 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.

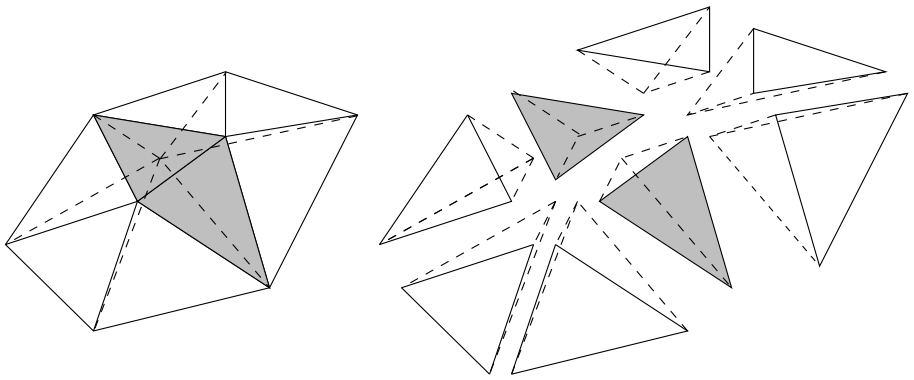


Figure 3.2: The simplification region is separated into its component tetrahedral volume elements. The old region is indicated in a solid line, the new region indicated with a dashed line and shaded faces are removed after simplification.

Lindstrom and Turk[LT98] perform local volume preservation by defining a signed volume component of their optimisation. They divide the local *edge collapse domain* into representative tetrahedrons (see Figure 3.2), 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

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

3.3 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 [Hop96]), 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 3.4.

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 2.3), 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:

- *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 neighbour of any of a face in the set $\mathcal{BOT}$. The consequence of not performing this test is shown in Figure 3.3.
- *Does removal of this edge result in face flipping or folding?* This problem is addressed in [Hop96, Hop99]. 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 3.4). It should be noted that the determination of the normals at each iteration of the process is an expensive time overhead.

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

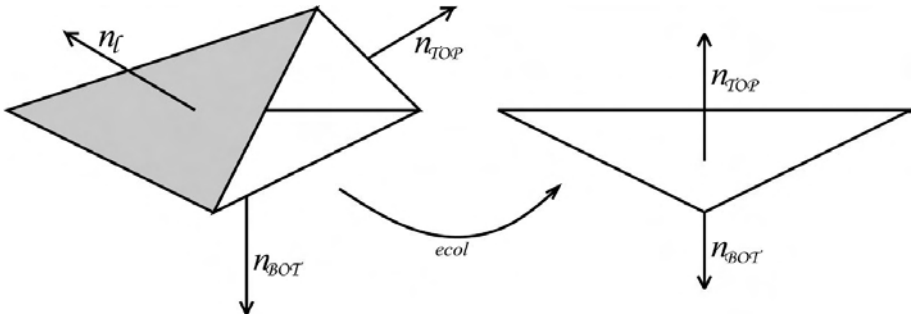


Figure 3.3: Detecting topology change. The removal of the shaded face results in a two sided face and hence a non-manifold mesh. n_l refers to a normal from the removed face, while $n_{\mathcal{TOP}}$ and $n_{\mathcal{BOT}}$ are the normals to the faces in the face sets $\mathcal{TOP}$ and $\mathcal{BOT}$ respectively.

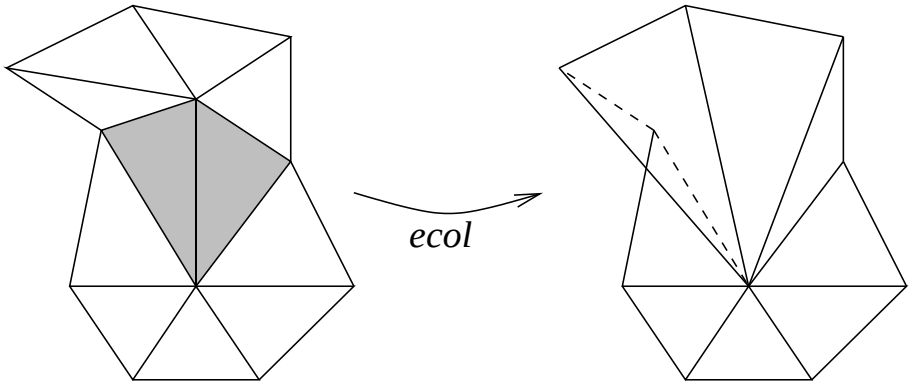


Figure 3.4: Detecting face flipping. Removal of the shaded faces results in a hidden or intersected face (indicated here with a dashed line). This can be avoided by testing the orientation of the faces in the $\mathcal{ECD}$ before and after the edge collapse operation.

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 3.4 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

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

- 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.

3.4 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:

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

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 3.9). 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[GH97] 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[Hop96] 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.

3.5 Implementation

The GeMS (or **Ge**neric **M**emoryless **S**implification) 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[Hop98]. Like Hoppe, we speed up face traversal by determining the neighbours 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 3.5, the basic algorithm for producing decimation meshes with linear dependence is shown. As described in Section 3.3, 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 3.3 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 3.6, the algorithm for the generation of batched decimation meshes is presented. Note that any *ecol* records representing edges within the *edge collapse domain* of *current_ecol* are erased (as described in Section 3.4).

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

```
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 3.5: An algorithm for generating decimation meshes with linear dependence in GeMS.

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.

Like [Hop96] (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 3.7). 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 Figure 4.3 (Chapter 4) 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 3.8 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 3.8) disregards areas of high curvature and detail features on the mesh.

In Figure 3.9 the batched hierarchy has been used to automatically generate a level of detail

```
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 3.6: 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.

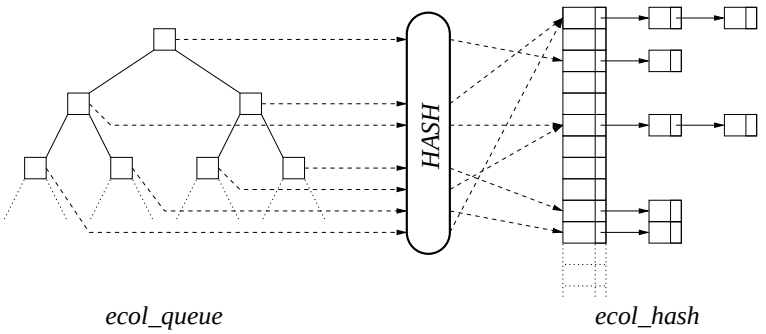


Figure 3.7: The hashed priority queue used within the GeMS framework.

hierarchy. Each consecutive level of detail has roughly half of the faces of the previous level, and are automatically generated during simplification.

3.6 Summary

We have presented a novel generic memoryless polygonal simplification framework for efficient compression and hierarchical decomposition of triangular surface meshes. We present four error metrics used within our generic framework, including memoryless versions of the method of Hoppe[Hop96] and Garland and Heckbert[GH97]. We also introduce two new error metrics, one based on the length of the edge being removed, and another based on the curvature and declining volume of the region being simplified.

We then describe important factors which arise when implementing a generic framework. Necessary tests to detect and prevent mesh faults, such as face flipping and two-sided faces are described. We also define the batched hierarchy, a method of structuring the output mesh with groups of independent operations, and explain how it can be used for automatic level of detail generation. We include two algorithms, one used to compute multiresolution meshes with linear dependence, and the other to generate meshes in the batched hierarchy format.

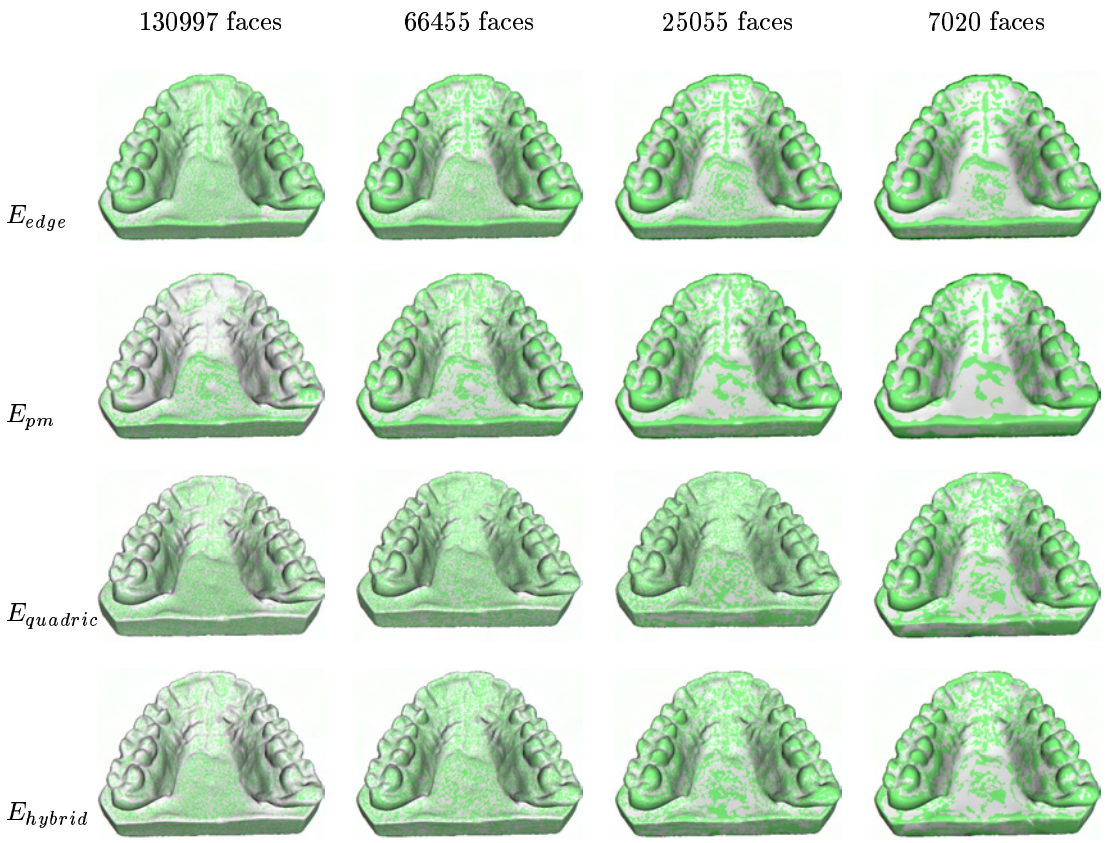


Figure 3.8: A generic platform allows the comparison of various techniques under the same conditions. Here we show the four error metrics discussed in Section 3.2 at various levels of simplification. In this visualisation the original model (233205 faces) has been alpha-blended over the simplified model 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.

Original Model, Level 0 (144898 faces)

Level 1 (73626 faces)

Level 2 (37830 faces)

Level 3 (17862 faces)

Level 4 (7348 faces)

Figure 3.9: 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.

Chapter 4

Evaluation of Memoryless Simplification

Essential for the overall quality of any simplified surface are the criteria for simplification. These criteria which are detailed in Chapter 2 are methods for vertex placement, and the error metrics used for simplification. Surface simplification techniques have been compared in the literature [HG97, CMS98], but these comparisons are either visual in nature (i.e. from a supplied picture) or derived from a common simplification measurement, such as running time or Hausdorff distance. No statistical conclusions have been drawn from these results. To date, and to our knowledge, no document has yet compared subset placement strategies for memoryless simplification or evaluated the performance of optimal against subset placement.

We present novel error metrics suitable for memoryless subset placement, and convert the metrics of Hoppe[Hop96] and Garland and Heckbert[GH98] to the same form. We also present a comparison of image-based metrics and model measurements to determine which model measurements correspond the best with image distortion and silhouette preservation.

In this chapter we use experimentation to attempt to resolve the following:

- What are the applications for the various subset placement techniques? Volume preservation is useful for precision in CAD/CAM and medical applications while preventing triangle degeneracy is important for textured models. We attempt to find the best subset placement technique for these criteria, as well as determining which technique most accurately preserves visual attributes.

- Is unconstrained vertex placement (using the optimal placement of Garland and Heckbert) significantly better than subset placement? It is commonly thought that surface quality can be better retained by "optimal" vertex placement (such as [GH97]). By allowing the vertex to be placed anywhere the storage required to reconstruct the surface (in the case of a progressive representation) is significantly greater than using a subset placement strategy. Hoppe[Hop96] chooses to place the vertex at either the midpoint or at either of the endpoints, requiring 2 bits of storage to indicate which, while Pajarola *et al.* require the vertex to be placed at the midpoint of the edge. We determine whether there is a significant visual improvement by increasing the domain of the subset and the number of bits required in order to reconstruct

the vertices position.

- Is their a better surface measure for determining the visual quality of an object? The Hausdorff distance is the most commonly used measure of surface quality, but it is difficult to determine (see Chapter 2). We propose that measuring surface volume is a better measure of visual degeneration of a compressed surface, and support this with experimental results.

In Section 4.1 we describe different strategies of vertex placement, while in Section 3.2 we describe the memoryless simplification techniques which we have tested. In Section 4.2 we define the evaluation criteria used to evaluate the error metrics used for simplification, and in Section 4.3 we state our hypotheses. In Section 4.4 we discuss the design of our experiment, including how data was acquired, and how it was analysed. In Section 4.5 we discuss the experimental results, which are available in Appendix B, and in Section 4.6 we discuss whether or not these results support our hypotheses. Finally we draw conclusions in Section 4.6.2.

4.1 Vertex Placement

The placement of a vertex is a major contributing factor to the storage required of a progressive mesh. Most commonly used placement techniques are to the midpoint of the collapsed edge[PR00] or a selection between the half-edge collapse and the midpoint[Hop96]. These techniques require 0 or 2 additional bits of storage respectively. We compare these with the simple 1-bit placement (or half-edge collapse). These different configurations are shown in Figure 4.1.

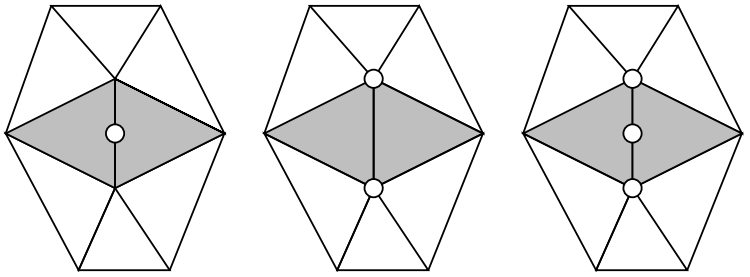


Figure 4.1: The subset placement techniques which were tested. The circles represent the possible positions of the collapsed vertex.

Unconstrained vertex placement (see Chapter 2) does not require the resultant vertex to lie on the edge, and relies on optimisation to place the point. However, in a progressive mesh format, storage of an unconstrained vertex would necessitate at least 24 additional bits (for three quantised floating point numbers) of storage per *vsplit* record in order to store the correction in the positions of vertices v_k and v_l.

4.2 Evaluation Criteria

We evaluate surfaces on several criteria We distinguish between error metrics based on *Model Criteria*, which include measures based on the surface geometry, and *Image-based Criteria*, which are measures based on the resulting image analysis.

4.2.1 Model Criteria

Surface Distance (K_{metro})

The Hausdorff distance (described in Chapter 2) is difficult and slow to determine. Due to the large number of evaluations which we will be making, we make use of the *Metro* package, which is able to quickly find the maximal (L^∞), mean of averages (L^1) and mean-squared (L^2) distance between two surfaces. We use the mean error between two surfaces to measure mesh distortion.

Enclosed Volume (K_{vol})

In modelling applications volume preservation can be essential to maintain an accurate compressed representation of the original surface. We derive our term K_{vol} from the Gaussian Divergence Formula,

$$K_{vol}(M) = \frac{1}{6}\sum_{i=1}^{|\mathcal{F}|}\left(v_1^i \times v_2^i\right)\cdot v_3^i,$$

where $|\mathcal{F}|$ represents the number of faces in mesh M and v_j^i represents the i_{th} vertex in face j of mesh M. We determine the volume of the mesh during the model simplification. In order to simplify these results we normalise them by dividing by the original surface volume. This yields a result within the range $[0\ldots1]$, and still accurately reflects the rate of decay of the volume during the simplification of the surface.

Sliver Ratio (K_{sliver})

We define our error measure K_{sliver} of mesh M to be a measurement of the deviation of all triangles in M from true equilateral triangles. We use the principle that the ratio of each edge in a equilateral triangle with each other edge is 1 to derive our error metric. We divide the sum of all edge lengths in the triangle by three times the minimum edge length in the triangle.

$$K_{sliver}(M) = \frac{1}{|\mathcal{F}|}\sum_{i=1}^{|\mathcal{F}|}\left(\frac{\sum_{j=1}^{3} e_j^i}{3\left(\min_{j=1}^{3} e_j^i\right)}\right)$$

where e_j^i is the length of the j_{th} edge of triangle i, i.e.

$$e_1^i = d(v_1^i, v_2^i),\ e_2^i = d(v_2^i, v_3^i),\ e_3^i = d(v_3^i, v_1^i).$$

We find that the value of K_{sliver} typically ranges within the region of $(1\ldots2)$, where a value greater than 2 indicates a poor model representation. We developed this evaluation criteria independently of Frey *et al.*[FB97], where they measure triangle degeneration by using the circumscribed circle (see Chapter 2). The method of Frey *et al.* produces a scaled value between $(0\ldots1)$. We find our measure runs very quickly, and the results are equivalent.

4.2.2 Image-based Criteria

Since the ultimate goal of computer graphics is to represent a virtual object to the viewer with the best quality, an image based evaluation is necessary.

Image Distortion (K_{L^1} and K_{L^2})

We evaluate both L^1 and L^2 image error by evaluating the pixel-wise image differences between a image of the original mesh I_1 and an image of the compressed mesh I_2. Like [LT00] we make use a number N_{view} of evenly spaced viewpoints about the object. We have chosen to use $N_{view} = 42$. We divide the resultant L^1 and L^2 error by the number of pixels in the image which constitute the model, and extract a single error value for this stage of the compression by averaging the results of all the N_{view} image comparisons.

Silhouette Deviation (K_{sil})

The occluding contours (or internal and external silhouettes) of an image give us a great deal of visual clues about the shape of an object[SGG+00]. Therefore a measurement of the accuracy of the silhouette of a rendered image of the object would indicate the accuracy of the technique. We measure the accuracy of the silhouette by determining the number of pixels in I_1 (defined above) which are not in I_2 and visa versa. We average this by the number of images being compared N_{view}.

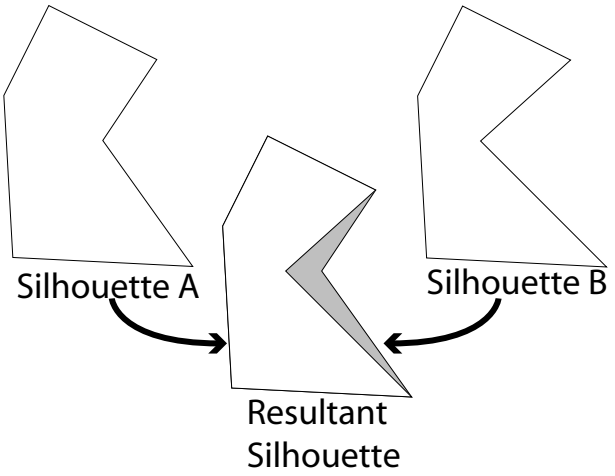


Figure 4.2: The determination of the silhouette deviation. The silhouette deviation between Silhouettes A and B is indicated by the shaded region.

4.3 Hypotheses

We formally define our hypotheses as follows:

1. Our first set of hypotheses is based on the comparison of memoryless subset placement techniques. This is tested in Experiment 1.
   (a) Simplification on the basis of unsigned volume (E_{uvol}) best preserves the volume of the model.
   (b) Simplification based on the length of the edge (E_{edge}) minimises the number of degenerate triangles (slivers).

(c) Our hybrid scheme (E_{hybrid}) preserves visual attributes significantly better than other techniques.

2. Our second hypothesis relates to the comparison of memoryless vertex placement techniques. This is tested in Experiment 2.

   - Unconstrained vertex placement offers no visual improvement over any subset placement techniques.

3. Our final hypothesis compares image-based measurements and model-based measurements. This is tested in Experiment 3.

   - Volume is as good as the Hausdorff distance as a predictive measure of the visual difference from the original model.

4.4 Experimental Design

Our experimental design is divided into two distinct sections. Initially we discuss the method in which the data was collected, and thereafter we outline our method of data analysis.

4.4.1 Data Acquisition

Our experimental results are derived during the simplification process. We define the number of models tested as $N_{model} = 13$, the number of tests performed on each model as $N_{test} = 10$ and the number of techniques being evaluated as $N_{tech} = 9$. For our image based measurements, we use a canvas of 512×512. This is twice the size of the image based compression platform used in by Lindstrom and Turk[LT00], as it affords us a high degree of accuracy when dealing with larger meshes. The techniques which we evaluated are outlined in Table 4.1. Note that the field **Test**

Name	**Test**	**Error Metric**	**Vertex Place**	**Where Defined**
E_{edge}	Technique	E_{edge}	0 bit	Section 3.2.3
E_{uvol}	Technique	E_{uvol}	2 bit	Section 3.2.4
$E_{quadric}$	Technique	$E_{quadric}$	2 bit	Section 3.2.2
E_{pm}	Technique	E_{pm}	2 bit	Section 3.2.1
E_{hybrid}	Technique	E_{hybrid}	2 bit	Section 3.2.4
0_bit	Placement	E_{hybrid}	0 bit	Section 4.1
1_bit	Placement	E_{hybrid}	1 bit	Section 4.1
2_bit	Placement	E_{hybrid}	2 bit	Section 4.1
optimal	Placement	$E_{quadric}$	optimal	Section 3.2.2

Table 4.1: A summary of the techniques which were evaluated.

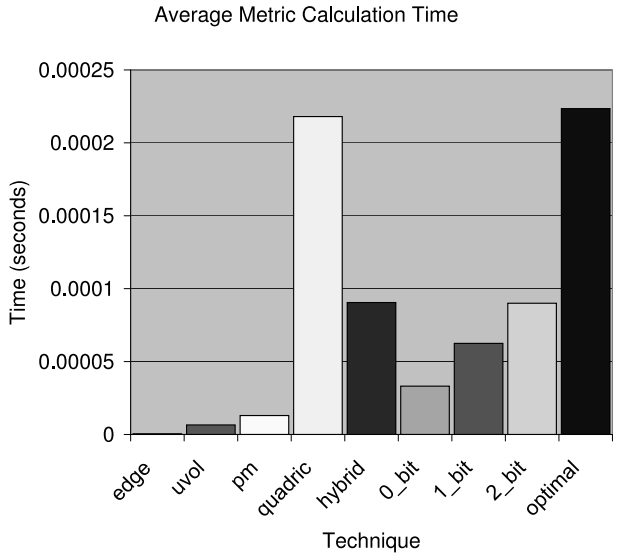
distinguishes between evaluations based on the error metric (Technique) or the vertex placement (Placement). Each model is simplified with all of the above techniques. Tests are performed on the model once the model has reach a fraction of its original size, defined by

$$\frac{N_{test} - m}{N_{test}},$$

where m represents the number of evaluations performed so far on the current model. At each testing stage, the evaluation criteria defined in Section 4.2 are applied to the current model. Note that the term K_{metro} is determined by an external program execution, so mesh files must be exported in a suitable mesh format at each step.

This testing process is a time consuming operation: Each of the N_{model} models must be simplified with N_{tech} techniques, and evaluated N_{test} times during the simplification process. Also, evaluation requires an image comparison from a N_{view} views. An increase in any of these terms drastically increases the running time of these technique evaluations of each model. Although we attempt to use models of a large variety of sizes and topological configurations, simplification time is dependent on the size of each of the tested models. For this reason, we chose not to evaluate excessively large models (300,000 faces+). Due to the restricted resolution of the image comparison window, differences between frames of large models would not be noticeable until vertices are sufficiently spaced. Note that only models which represent closed surfaces can be used when determining the error terms K_{L^1} and K_{L^2}, as an image comparison of frames containing holes may produce inaccurate results.

We also determine the running times of the techniques tested. Since each metric is implemented in the same platform (see Chapter 3) the overall time taken to perform the entire simplification is only dependent on this factor. We include these timing results in Figure 4.3.



Technique	Time (μs)
E_{edge}	0.4
E_{uvol}	6.6
E_{pm}	13.0
$E_{quadric}$	218.0
E_{hybrid}	90.5
0_bit	33.2
1_bit	62.5
2_bit	90.0
optimal	223.0

Figure 4.3: Average metric timings for the club model. This graph depicts the average time taken for each error metric to calculate the error associated with single edge collapse. These results were timed on an 500MHz AMD Athlon processor.

4.4.2 Analysis of Results

The above analysis yields a matrix of results, with the number of rows equal to the total number of tests, that is $N_{model} \times N_{tech} \times N_{test}$. Our tests with $N_{model} = 13$, $N_{tech} = 9$ and $N_{test} = 11$ yield over 1200 data points (ignoring null fields). The columns in this matrix correspond to the values determined by the six different comparison techniques, outlined in Section 4.2. Our statistical analysis of the results are discussed below.

Experiment 1

To compare the memoryless simplification techniques we have defined in Section 3.2, we exclude results which correspond to our placement experiments, leaving ± 700 data points. We analyse this data using a 1-way MANOVA, where the independent variable is the error metric used, and the dependent variables are the results of the evaluation criteria. The results are significant, with $p << .05$.

In order to evaluate these results, we perform a Scheffe Test on each of the evaluation criteria. The Scheffe Test produces similar results to the more popular Least-Squares Difference (LSD) Test, but is considered more conservative. We have included the resultant matrices of the Scheffe Tests in Appendix B. Entries which are emphasised indicate statistically relevant comparisons at $p < .05$.

Experiment 2

As in Experiment 1 above, in order to compare vertex placement techniques, we exclude results which correspond to out error metric evaluation experiments. This leaves us with roughly half the data. Again we analyse this in a 1-way MANOVA, with independent variables being the placement technique and the dependent variables are the evaluation criteria. The results are significant, with $p < .05$. We perform a Scheffe Test on each of the evaluation criteria, the results of which are included in Appendix B. Italicised entries indicate relevant comparisons at $p < .05$.

Experiment 3

We use all the data to determine what the correspondences between data collected using image based criteria and data determined directly from the surface. This is simply achieved by calculating a correlation matrix between the evaluation criteria of the collected data. We perform a correlation matrix on all the supplied data, as well as two pathological cases, those of optimal placement and 2 bit placement. We include these matrices in Appendix B; italicised results indicate a significant result at $p < .05$.

4.5 Experimental Results

4.5.1 Experiment 1: Comparison of memoryless subset placement techniques

We find that the error term E_{edge} (Mean = .0160414) is statistically worse in the K_{L^1} image sense than E_{uvol} (Mean = .0102896), while it is statistically worse (Mean = 5.222827) than E_{uvol} (Mean = 3.17403), E_{pm} (Mean = $3,270291$) and E_{hybrid} (Mean = 3.004087) in the K_{L^2} sense. E_{edge} is statistically worse (Mean = 167.4658) than all other techniques evaluated in terms of the silhouette measurement K_{sil}, but is statistically better than all other techniques in terms of the average triangle degeneracy, or K_{sliver} (Mean = 1.293184). E_{edge} is also found to be statistically worse than all other tested techniques in terms of the measurements K_{vol} (Mean = .9956310) and K_{metro} (Mean = .0007065).

4.5.2 Experiment 2: Comparison of vertex placement techniques

There was no statistical significance between the vertex placement techniques in terms of the image-based measurements K_{L^1}, K_{L^2} and K_{sil}. There was also no statistical difference in the evaluation criteria K_{metro}. The placement techniques *1_bit* (Mean = 1.520909) and *2_bit* (Mean = 1.501823) were found to be statistically worse than *0_bit* (Mean = 1.428884) in terms of the triangle degeneracy measure K_{sliver}. Unconstrained (optimal) vertex placement (Mean = .9993752) was found to be statistically better than *0_bit* (Mean = .9977720) placement in terms of volume preservation, or K_{vol}.

4.5.3 Experiment 3: Evaluation of image based and model based measurements

Including all measurements ($n = 1400$) yields a correlation matrix with all measures correlated (i.e. $p < .05$) except K_{metro} and K_{sliver}. Most notably, the term K_{vol} correlates better with the image based measures K_{L^1} ($r = -.76$ vs $r = 0.31$), K_{L^2} ($r = -.64$ vs $r = .33$) and K_{sil} ($r = -.89$ vs $r = .56$). Using Fishers r' comparison we find that for K_{L^1}, $z = 7.7553$, for K_{L^2}, $z = 4.7673$ and for K_{sil}, $z = 9.0572$. Each of these values has a $p < 10^{-7}$.

K_{vol} correlated better with image based measures than K_{metro} in the pathological case of unconstrained (optimal) vertex placement, but yields considerable lower r values in our tests (see Appendix B for these values. We include visual (colour) results of K_{metro} to measure the model quality in Figure 4.4. Fishers comparison yields that given the small sample size, no conclusions can be drawn. For K_{L^1}, $z = 0.6399$, for K_{L^2}, $z = 0.6799$ and for K_{sil}, $z = 0.3001$. These values are not significant, with $p > 0.05$. In the case of *2_bit* vertex placement the situation is different, with Fishers comparison yielding a $p < 10^{-8}$. In Figure 4.5 some results from the cranium model are displayed, showing the silhouette degradation for *0_bit* vertex placement. For K_{L^1}, $z = 8.4777$, for K_{L^2}, $z = 5.2745$ and for K_{sil}, $z = 10.7575$ for the small data set tested.

4.6 Discussion of Results

4.6.1 Our Hypothesis

1. (a) *Simplification on the basis of unsigned volume (E_{uvol}) best preserves the volume of the model.*

   This statement cannot be justified from the statistical evidence gathered. In fact, the results indicate that the volume degradation of all the methods are very similar, with the notable exception of E_{edge} (which has no implicit or explicit means of limiting volume loss).

   (b) *Simplification based on the length of the edge (E_{edge}) minimises the number of degenerate triangles (slivers).*

   The results we have gathered through experimentation support our claim. E_{edge} was found to have a statistically lower average of degenerate triangles than the other techniques tested.

   (c) *Our hybrid scheme (E_{hybrid}) preserves visual attributes significantly better than other*

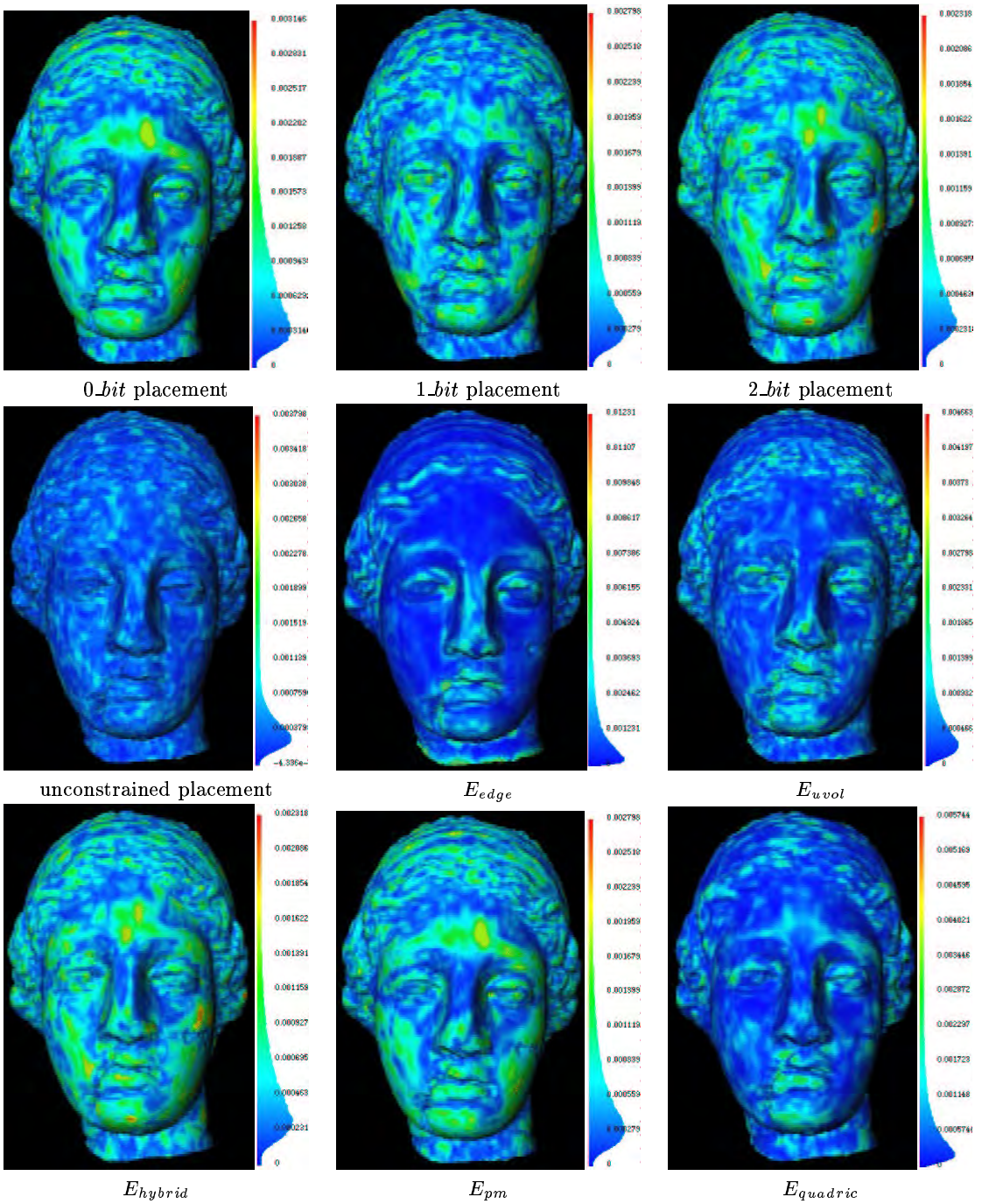


Figure 4.4: Visual Metro results. The results from 0_*bit*, 1_*bit*, 2_*bit* and unconstrained placement are from the placement experiments in Experiment 1. The remaining results were used in Experiment 2. All the above results were used in Experiment 3. The manner in which the Metro error is distributed is important to the overall model quality — note that in E_{hybrid} and unconstrained placement the error is distributed evenly about the mesh, while in E_{edge} the error is localised at the sharp corners of the model.

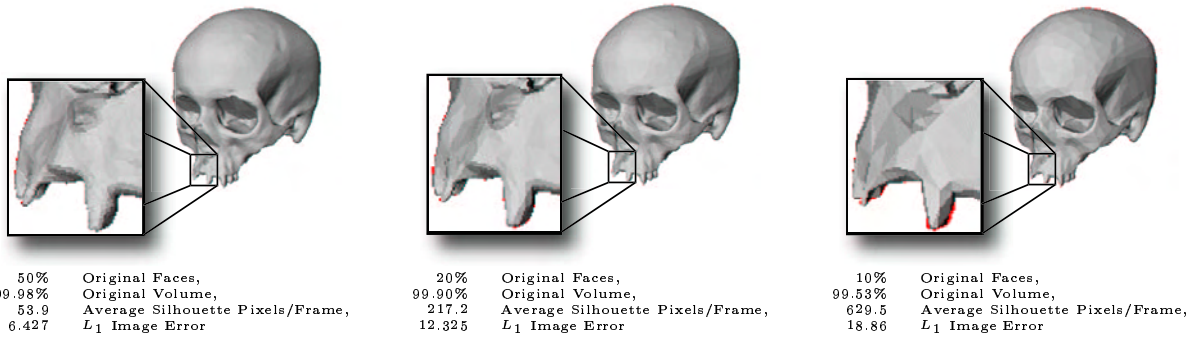


Figure 4.5: Volume degradation approximating image quality. In the images it is clear that the silhouette of the image degrades with respect to the original silhouette (shown in red) as the object is simplified. The average number of silhouette pixels per frame measures the deviation of the simplified object from the silhouette of the original. A number of experiments yields a correlation between volume and the L_1 image error of $-.76$, and between volume degradation and silhouette degradation there is a correlation of $-.89$.

techniques.

Only E_{edge} performed statistically worse in the image quality sense than our technique E_{hybrid}. This would indicate that all the remaining techniques tested have similar image quality.

2. *Unconstrained vertex placement offers no visual improvement over any subset placement techniques.*

   Our experimental results support this claim. Statistically the resultant image quality of optimal vertex placement is as high as any of the tested subset placement techniques.

3. *Volume is as good as the Hausdorff distance as a predictive measure of the visual difference from the original model.*

   From our statistical results, this claim is strongly supported. In fact, the volume correlated almost twice as well with the image measures K_{L^1} and K_{L^2} for our experiments, and significantly better for the silhouette measure K_{sil}.

It is clear from these experimental results that E_{edge} is offers a very specific tool for surface compression. As this error measure cannot limit the degradation of volume or preserve normal deviation, it is a relatively poor method to preserve visual details or model fidelity. It does, however, execute considerably quicker than the other techniques tested, and substantially reduces triangle degeneracy in the form of slivers. Like E_{vol}, E_{edge} could be incorporated into another error metric to reduce triangle degeneracy (like the E_{spring} term included in the method of Hoppe *et al.*[HDD+93]).

Our memoryless derivatives of Progressive Meshes[Hop96] E_{pm} and the quadric error metric of Garland and Heckbert[GH97] $E_{quadric}$ execute slower than the original versions, but provide results of the same quality. Our method E_{hybrid} performs comparably to these established techniques of surface compression (as is clear from our statistical results), but no statistical evidence can be provided from these experiments of distinguishing attributes of our technique.

The most significant results of our experiments are in the the comparison of subset placement with unconstrained vertex placement, and the effectiveness of surface measurements. Our finding

that subset placement is just as good as unconstrained vertex placement based on the measures we used implies that unconstrained vertex placement is unnecessary for preserving volume, image quality or triangle degeneracy. This is a very useful finding for surface simplification platforms, as subset placement strategies run quicker and the resultant progressive mesh would require less storage for reconstruction. Since collapsing to the midpoint results in the same quality as all the placement strategies tested, no additional storage is required to indicate during reconstruction which type of vertex removal was used.

We find from our experiments that the volume correlates considerably better with the image errors used than the results from the *Metro* tool, which is used to approximate the Hausdorff distance between the two surfaces. This is a useful result for evaluating the quality of a model during simplification as volume can be computed quickly to approximate the quality of the rendered image. Volume loss can be used as an effective threshold or termination criteria during simplification.

4.6.2 Summary

In this chapter we have presented evaluation criteria for evaluating simplified surfaces. The simplification techniques we evaluate differ on the basis of the position the vertex is collapsed to, and the error metric used. We present several criteria for evaluating simplification techniques based on image-space measurements instead of model-space measurements, specifically K_{L^1}, K_{L^2} and K_{sil}.

We have stated five hypotheses, which we have attempted to support with experimental results. We detail our experimental platform and method in which we capture the data. Using our measurement criteria we evaluated the various simplification techniques on thirteen un-textured surface models. The findings of our experiments have important implications in the field of surface simplification, and are summarised in Section 4.6. Visual results are included in Appendix D.

Chapter 5

Effective View-Dependent Transmission

Applications for browsing online model repositories are becoming more challenging to design. The improvement of laser scanning to sub-millimetre resolution has provided extremely high quality 3D representations of both medical data[Ack98] and works of art[LRG+00, LMWM01]. Often models of such high quality are too big for the client to store on their local machine, let alone to render, and for study purposes it is not sufficient to view a simplified version of the model.

In a selective or view-dependent transmission framework, only the details requested by a client are transmitted and refined. In this way the overhead of transmitting a level-of-detail sequence is avoided. Previous techniques of view dependent transmission [TLG99] have not made it possible for the client to avoid storing regions of the mesh that they are no longer looking at.

Several requirements must be addressed for view-dependent browsing of model repositories:

- **Interactive Model Browsing:** Clients require real-time interaction with the models which they are viewing. Possible problems could be limited rendering capacities of client machines, or bandwidth restrictions.

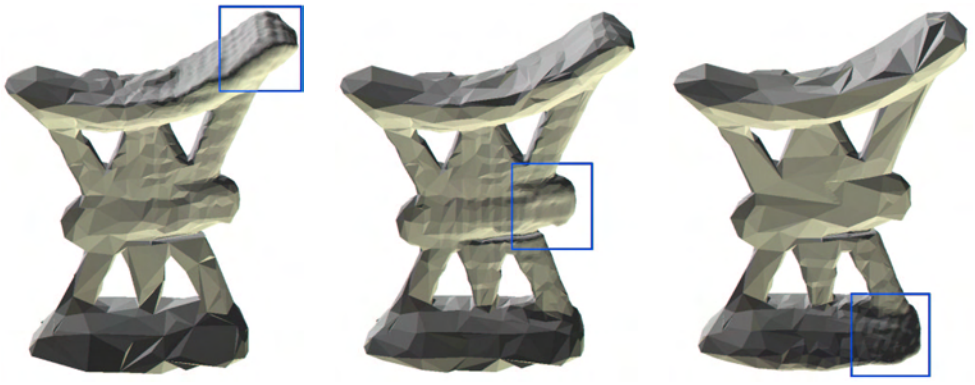


Figure 5.1: Selective refinement. The headrest is an object of African cultural heritage. In each frame the selected region is refined progressively. Selective Refinement applications are useful for the close study of complex models, such as art objects.

- **Dynamic Model Updates:** Changing the view point should result in immediate refinement of the region which has become visible.
- **Minimal Transmission:** Obviously to reduce the server load and the client wait time, the less transmitted the better.
- **Even Distribution of Refinements:** A client should progressively receive refinements spread evenly throughout the selected refinement region.

We present a stateless client for progressive view-dependent refinement which provides a solution to the problems stated above. Our client never receives the entire model, only a coarse representation and a sequence of refinements necessary to increase the detail of a selected region. We call this client *stateless* as it is completely dependent on the server to maintain the state of the refinements on the client model.

In Section 5.1 we describe a technique to perform view-dependent refinement, while it's conversion into a view-dependent transmission application is discussed in Section 5.2. In Section 5.3 we discuss the results of our work. Finally we conclude and offer suggestions for future work in Section 5.4. The implementation details included in this document are based largely upon the experimental results of Muller *et al.* [MPS+00].

5.1 Method Overview

We construct our view-dependent framework upon the commonly used [HDD+93, Hop96, XESV97, Hop97] atomic operations edge collapse (*ecol*) and vertex split (*vsplit*). We associate each operation with a spatial representation called a visibility sphere, used to determine whether an operation is visible and needs to be performed. These have attributes such as orientation, position, size and relative error (see Figure 5.2).

Initially we simplify the input mesh using progressive meshes. The visibility spheres for each atomic operation are defined during the simplification process. We construct these in a conservative fashion in order to minimise error of the transmitted representation. During the simplification process we also indicate which atomic operations each vertex split is dependent on. This dependency information is used to construct a directed acyclic graph of these vertex splits, where the root nodes correspond to the coarsest possible representation, while the terminal nodes correspond to the final mesh. The visibility spheres at each node of this graph are defined as the union of the visibility spheres bounding its children and the visibility sphere associated with the vertex split at that node.

Selective (or view dependent) refinement is accomplished using a breadth first search of the DAG defined above. If the visibility sphere of a parent node is found to be visible, then the nodes children can also be visible and are tested.

5.1.1 Visibility Spheres

Considering the region of the atomic operations as a measurable spatial entity is central to our technique. We use visibility spheres in conjunction with our vertex split hierarchy to provide a fast technique of determining which vertex split and edge collapse operations need to be applied to refine the selected region.

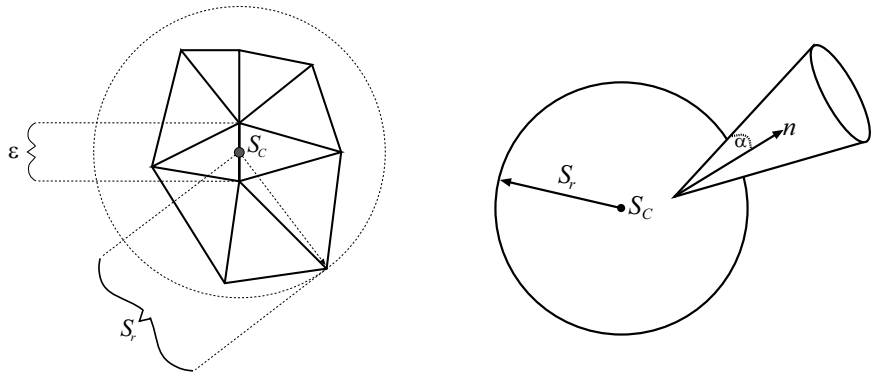
As in [XESV97, TLG99], we construct the visibility spheres during the surface simplification process, since during the simplification process the current state of the mesh (which vertices and faces are present) is known. This could be performed afterwards on any progressive mesh form (as with [Hop97]), but would require the reconstruction of the model.

A visibility sphere consists of three separate components:

- a bounding sphere $S = \{S_C, S_r\}$, with centre S_C and radius S_r, representing the position of one (or more) vertex split operations,
- a floating normal cone $N = \{\mathbf{n}, \alpha\}$, with axis $\mathbf{n}$ and extent angle α, bounding the normals and normal cones of one (or more) vertex split operations, and
- an error value ε representing the maximum distance between v_l and v_k of the bounded vertex split operations.

A vertex split operation not only inserts a vertex and one or two faces into a surface mesh, but also adjusts the position of a vertex within those faces and therefore their face normals. It is for this reason that we centre our visibility spheres about the centre of the region of the atomic operation (unlike [Hop97]), specifically the centroid of the base points in the region (we call this point S_C). The determination of the radius of the sphere S is shown in Figure 5.2.

The orientation of a visibility sphere is represented by a *floating* normal cone $\{\mathbf{n}, \alpha\}$ (first introduced by Shirman *et al.* [SAE93]). The axis of the normal cone $\mathbf{n}$ is the resultant of normals to the one or two faces removed as a result of applying the atomic operation, which is the orientation of the removed edge. Initially the extent α is zero. The error ε associated with an atomic operation is simply the distance between v_l (the vertex lost) and v_k (the vertex kept).



(a) (b)

Figure 5.2: The visibility sphere. (a) The determination of the attributes of the visibility sphere. S_r represents the radius of a sphere originating from the centroid of the base points of the region S_C. The error term ε indicates the distance between the two points which would be collapsed during the edge collapse operation. (b) A visibility sphere consists of the sphere (representing the atomic operation in space), and the cone of associated normals (for determining whether the sphere is forward facing in real-time).

A union operator $\cup$ can now be defined for visibility spheres. Given visibility spheres $\mathcal{A}$ and $\mathcal{B}$,

$$\mathcal{A} \cup \mathcal{B} = \{S_{\mathcal{A}} \cup S_{\mathcal{B}}, N_{\mathcal{A}} \cup N_{\mathcal{B}}, \max\{\varepsilon_{\mathcal{A}}, \varepsilon_{\mathcal{B}}\}\}$$

We say that a visibility sphere is *active* if it is visible. A visibility sphere is visible if and only if:

1. it is completely or partially within the view,
2. it is not orientated away, and
3. the noticeable change caused by applying the enclosed vertex split is larger than some tolerance (normally the screen pixel height). This is normally referred to as screen-space error.

Outside View

We determine the visibility of a sphere in the same way as Hoppe[Hop97]. Given a visibility sphere with centre $S_C = (S_C^x, S_C^y, S_C^z)$ and radius S_r, and a view frustum consisting of the four bounding planes $P_i = \{a_i, b_i, c_i, d_i\}, i = 1 \ldots 4$, the sphere is outside the view if

$$a_i S_C^x + b_i S_C^y + c_i S_C^z + d_i < -S_r, \text{ for any i} = 1 \ldots 4. \tag{5.1}$$

Orientated Away

The normal cone with axis $\mathbf{n}$ and extent α is tested against the eye vector $\mathbf{e}$. The visibility sphere is back facing if

$$\frac{\mathbf{e} \cdot \mathbf{n}}{||\mathbf{e}|| \, ||\mathbf{n}||} > \sin(\alpha). \tag{5.2}$$

Several recent documents [LE97, SGG+00] have stressed the importance of the silhouette for object recognition. Normally for visibility spheres bounding high-detail refinements, the normal cone angle α is close to (or equal to) zero. We introduce a minimum angle for the normal cone extent α in order to ensure that most silhouette curvature is restored.

Screen Space Error

Refinements to the model need not be performed if the deviation caused by the insertion of the new vertex is smaller than the pixel size (approximated by τ), i.e. if the projected distance of the edge inserted after a vertex split is smaller than the error tolerance τ. This is achieved by approximating the orientation of the inserted edge by the bounding normal cone $\{\mathbf{n}, \alpha\}$, and the maximum length ε of the enclosed edges. Given the eye vector $\mathbf{e}$, we define the angle between it and the normal cone axis as λ, and we define r as the length of the vector formed by projecting $(S_C - p)$ onto the eye vector $\mathbf{e}$.

We approximate two screen space errors (in Equation 5.3), as the farthest extents of the normal cone,

$$\varepsilon_1' = \frac{\varepsilon}{r}\cos(\lambda + \alpha) \text{ and } \varepsilon_2' = \frac{\varepsilon}{\mathrm{r}}\cos(\lambda - \alpha) \tag{5.3}$$

If $\max\{\varepsilon_1', \varepsilon_2'\} < \tau$ then no vertex split operations enclosed within the tested visibility sphere would be noticeable when applied.

5.1.2 The Vertex Split Hierarchy

A vertex split operation cannot be performed until it is *valid*, i.e. the necessary faces and vertices are present in the current model. In order to ensure that refinements are ordered in such a way that only valid vertex split operations are permitted, a hierarchy of these dependencies must be constructed.

Although analogous to the vertex hierarchy of Hoppe [Hop97, TLG99], constructing the hierarchy of dependency information from the atomic operations rather than the vertices themselves has various advantages. We find a vertex split hierarchy is smaller, in both the number of nodes as well as the total size.

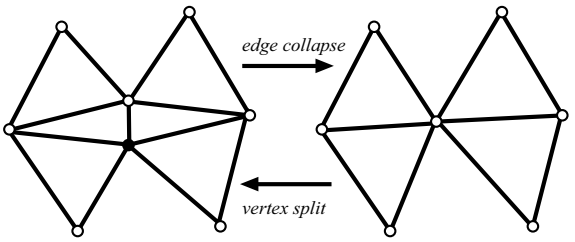


Figure 5.3: Dependent vertices for a vertex split. The dependent vertices for a vertex split are shown by the white filled points. The black filled point indicates the vertex which is lost after the edge collapse has been applied (earlier defined as v_l).

Each node of the vertex split hierarchy contains a single atomic operation, and a visibility sphere bounding it and all of its children. We determine dependencies between the vertex split operations during the simplification process. We identify the dependencies between atomic operations in a similar way to Hoppe, where only the vertices within the faces neighbouring the one or two inserted faces (Figure 2.3) need be present before a particular vertex split can be performed (see Figure 5.3). Each vertex split operation can therefore have up to seven parent vertex split operations — each parent introduces one of the required vertices into the model. This is translated into a Directed Acyclic Graph (or DAG) where each node corresponds to a vertex split operation (or it's inverse, edge collapse), and a nodes parents correspond to the vertex split operations introducing one of the required vertices into the mesh. Each node has at most seven parents, but can have any number of children. The root nodes of the graph correspond to the vertex splits which are applied to the coarsest representation of the model, while performing vertex split operations at terminal nodes results in the reconstruction of the original model $\hat{M}$.

The spatial representation of each vertex split within the DAG is defined by its associated visibility sphere (shown in Figure 5.4). The determination of the visibility sphere associated with a vertex split $vsplit^p$ is described as follows. Given

- a parent node $vsplit^p$ with representative visibility sphere vs^p,
- n children of $vsplit^p$, $vsplit^c_i, i = 1 \ldots n$, and
- the visibility spheres associated with the children $vs^c_i, i = 1 \ldots n$,

the visibility sphere associated with $vsplit^p$ is then

$$vs^p \cup vs^c_1 \cup \ldots \cup vs^c_n.$$

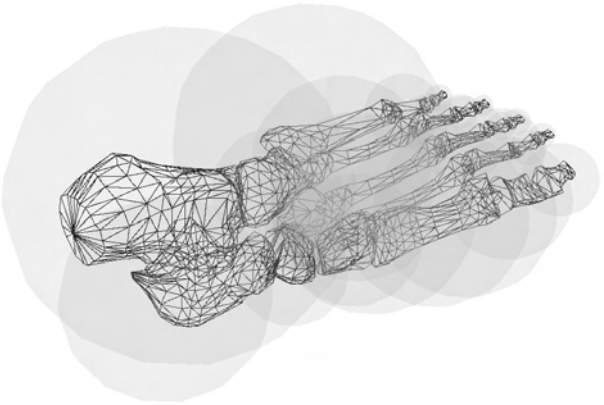


Figure 5.4: Bounding spheres. These spheres represent the spheres bounding the coarsest level of refinement associated with the root nodes of the vertex split hierarchy.

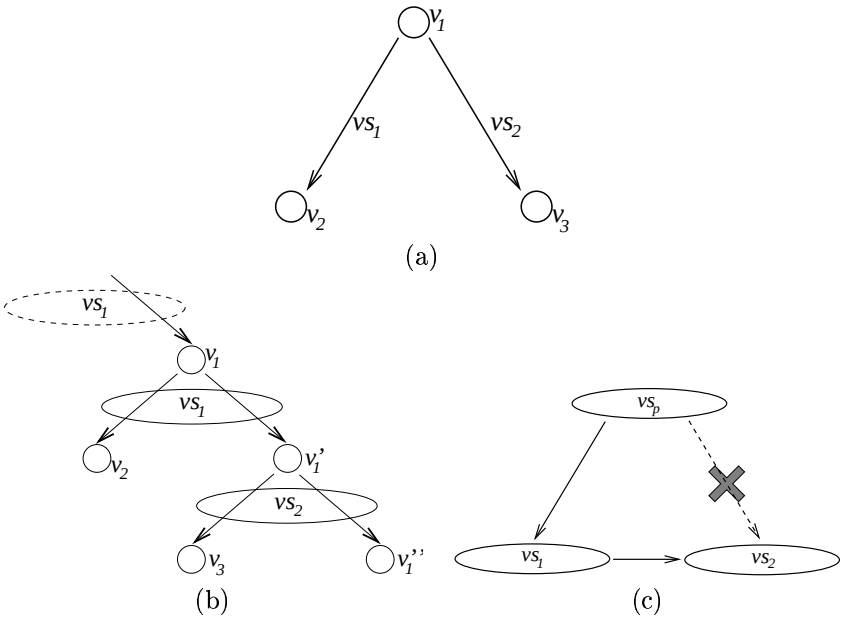


Figure 5.5: A comparison of a vertex hierarchy and a vertex split hierarchy. The initial problem is shown in (a), the vertex hierarchy solution in (b) and the vertex split hierarchy solution in (c). In (c) the removed dependency is indicated by a dashed arrow.

This dependency structure results in a problem, shown in Figure 5.5. The actual spatial problem is shown in Figure 5.5(a) — a single vertex (v_1) is split twice into v_2 and v_3, by vertex splits vs_1 and vs_2 respectively. The order in which these operations is performed is important. Although the faces and vertices necessary for vs_2 to be performed may be present in the mesh before vs_1 has been performed, the indices may be present in the wrong faces. In order to avoid this problem, the vertex hierarchy (of Hoppe [Hop97]) (in Figure 5.5(b)) propagates the vertex v_1 into v_1', thus preventing vs_2 from occurring before vs_1. In a vertex split hierarchy (in Figure 5.5(c)), this requirement is addressed by simply making the node vs_1 a parent of vs_2. Note that both vs_1 and vs_2 will have a common ancestor (in this case vs_p) since they both originate from the vertex v_1.

5.1.3 Selective Refinement

The vertex split hierarchy defined in Section 5.1.2 provides an elegant correspondence between the visibility spheres defined in Section 5.1.1 and the graph of atomic operations. We will now describe how to use these structures to perform selective refinement.

Given a refinement region defined by a view frustum, we initialise our tree traversal by placing the root nodes of the graph in a queue. We then test the first element of the queue. If the visibility sphere associated with that node is *active* (i.e. it is visible), the node is marked to be refined, and the nodes children are appended to the end of the queue. If the visibility sphere is not *active*, it implies that none of that nodes children will be active.

5.2 View Dependent Transmission

In Sections 5.1.1 and 5.1.2 we have described a framework to perform selective refinement at real-time on progressive meshes. We now present a view-dependent transmission technique and address the problems inherent with browsing large model repositories.

Typically the client would request a model from a repository, and after a base mesh has been sent, a refinement request is initiated by the client transmitting the view frustum. The server determines what needs to be transmitted using the method described in Section 5.1.3. The server initially transmits the edge collapse operations to un-refine nodes which are no longer visible and thereafter sends the vertex split operations to refine what has entered the view.

Polygon Restrictions

In order to ensure interactive model browsing and relatively constant frame-rates, we introduce a technique of polygon transmission restrictions (similar to those of [TLG99, LE97]). The client informs the server of a polygon restriction during the initialisation of the transmission. The server restricts the number of atomic operations which are sent to the client such that the client does not exceed this polygon limitation. Because refinements are also transmitted in the order in which they are present in the hierarchy, (i.e. top down) there is an even distribution of refinements within the viewed region (as is clear from Figure 5.8).

The client could decide to alter her view during transmission. The server must be able to stop the current transfer and update the transmission list immediately as the client changes her mind.

Coherence

Using model coherence within a view-dependent refinement technique is not new [Hop97, LE97], but the merits of maintaining a stateless client have not yet been explored. There is typically a high degree of frame-to-frame coherence between the view changes of the client, especially when viewing specific regions of the model. By transmitting edge collapse operations necessary to un-refine what has left the view and then vertex splits only of the regions which have entered the view, we are naturally exploiting this property. It is clear from Figure 5.6 that the polygon count of a remotely viewed model can be kept relatively constant when applying coherence.

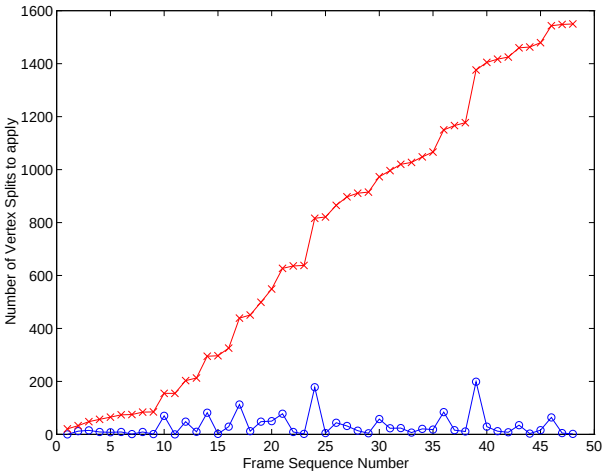


Figure 5.6: The benefit of coherence. The view frustum is moved across a surface in a preset path. Crossed points indicates transmission with refinement (vertex splits) only, while circled points indicate transmission of both vertex split and edge collapse operations taking advantage of coherence between refined regions. With no polygon limitations in place it is clear that un-refining sections of the mesh drastically reduces the graphics load on machines with poor rendering capabilities.

Client Cache

In order to further improve client interactivity with the model, we observe that only vertex split operations which have already been transmitted can be collapsed. We create a cache on the client side containing recently applied vertex split operations. We maintain the same cache on the server side. Instead of transmitting a vertex split or edge collapse to the client, it is sufficient to transmit only the index number of the operation stored in the cache.

5.3 Results

Results are based on the implementation described in Muller *et al.*[MPS+00]. The table below compares an unoptimised vertex hierarchy implementation with visibility spheres, with a similarly unoptimised vertex split hierarchy. Both a vertex hierarchy and a vertex split hierarchy are quick (normally $O(\log n)$ to traverse, but since the vertex list must be traversed every time in order to transmit the necessary refinements, both run in $O(n)$. It is clear that the vertex split hierarchy occupies roughly 70% of the disk space of the vertex hierarchy.

	Vertex Hierarchy		**Vertex Split Hierarchy**	
Mesh	# nodes	size (kb)	# nodes	size (kb)
bones	4143	99	1989	63
bunny	70,395	1548	34,448	1102
headrest	288,826	6354	144,189	4614

We find also that the cache maintained by both the client and the server reduces the number of edge collapse operations which must be transmitted to between 5% and 10% of the original number depending on the degree of coherence between frames, and the size of the cache. We used a cache

of 5% of the total number of vertex split operations in the original progressive mesh model.

The results of progressive and selective transmission are clearly shown in Figures 5.1, 5.7 and 5.8. In Figure 5.8 the region surrounding the eye of the bunny is refined with various polygon limitations. In Figure 5.7 and 5.1 three selectively refined regions are shown in each case.

5.4 Summary

In this chapter we have described a technique to facilitate selective and progressive refinement, which we adapted to the application of view-dependent transmission. We introduced the concept of a "visibility sphere", which is a convenient method of organising visibility information. We have presented a novel technique in which the normal cones associated with each sphere can be used to determine whether an atomic operation is noticeable in screen space.

The vertex split hierarchy is also introduced, which provides a more compact ordering of refinement operations. The construction of the hierarchy is discussed, along with problems that might be encountered in the hierarchy construction. We also show a number of optimisations which can be applied to the framework, such as a client cache, which greatly reduce the number of operations which must be transmitted.

Our view dependent refinement framework provides interactive, progressive and selective transmission of polygonal meshes in large model repositories. The stateless client system described gives the client a great deal of flexibility with respect to interactivity and control of the transmission process.

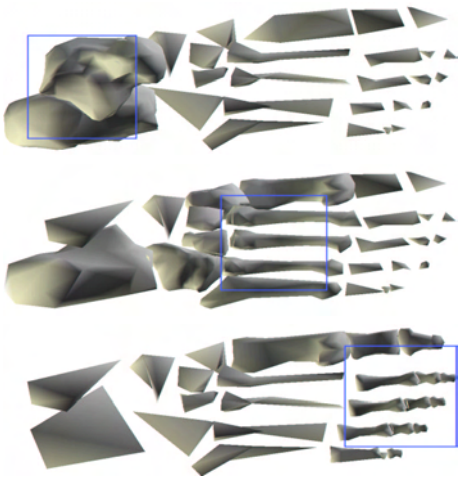
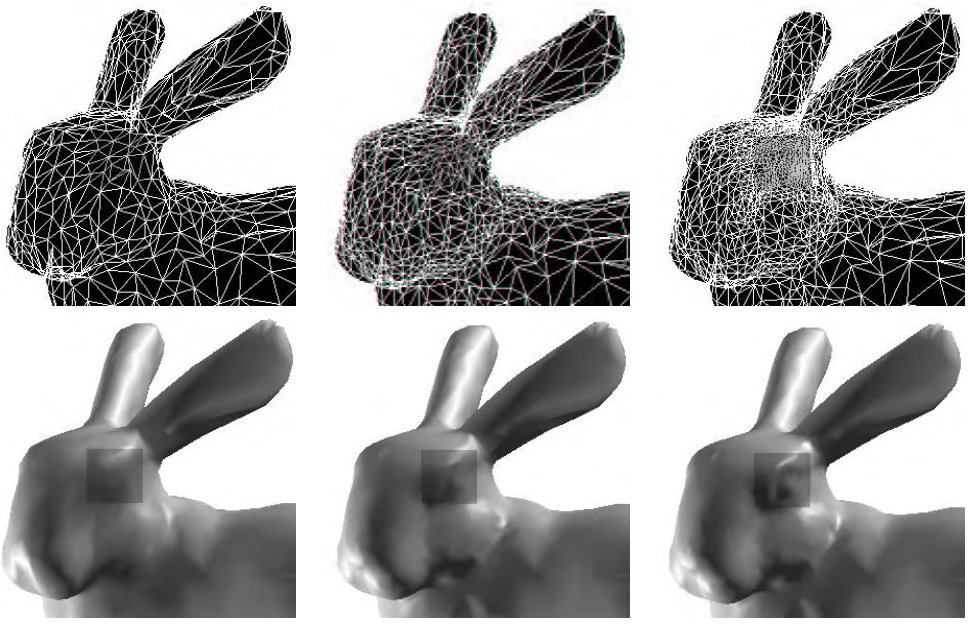


Figure 5.7: An example of selective refinement. Separate sections of the model are refined in each frame.



1000 polygons 1500 polygons 2000 polygons

Figure 5.8: An example of progressive refinement. The bunny's eye is progressively refined by setting the polygon limit to 1000, 1500 and 2000 polygons respectively.

Chapter 6

Creation and Control of Continuous Level-of-Detail

Level of detail blending through geometric morphing (or *geomorphing*[Hop96]) is effective at hiding discontinuities in level of detail (LOD) sequences (this artefact is often called "popping"). Geomorphing is an interpolated transition between LOD models, and are designed to ensure that model transitions are not noticeable. Although widely employed in terrain visualisation, geomorphing techniques are typically difficult to apply to objects in a scene — most applications using objects are forced to use low polygon count models, or discrete LOD to optimise rendering performance. There are several reasons for this:

- *Performance:* The number of interpolations per frame of a geomorph sequence directly impacts on rendering performance, as they must be performed on the CPU. Unfortunately for real-time applications (such as games) model quality is considerably less important than interactivity. We make use of current graphics hardware to ensure that there is little or no CPU overhead incurred by performing interpolations between LOD models.

- *Animation:* Although morphs between different vertex positions of the geomorph are clearly defined through interpolation, the combination of geomorphing and animation, such as matrix palette skinning or wires has not yet been explored. Our interpolation technique can be combined with the deformation technique of Singh and Fiume[SF98] or bones[LKM01] to enable time-dependent animation which is independent of the LOD switches.

- *Continuous LOD Control:* Continuous LOD presents unique challenges when attempting to maintain a constant frame rate. Switching between LOD models may result in sudden model discontinuities, which is exactly what geomorphing is designed to avoid. We present a novel greedy algorithm for dynamic LOD adjustment based on predicted image degradation, to produce a reactive frame-rate control mechanism.

- *Construction Complexity:* The geomorphing format described by Hoppe[Hop96] based in the progressive mesh is difficult to construct. Constructing the geomorph sequence from the progressive mesh format requires that each vertex split operation be tested for validity to determine which operations can be applied independently. This requires the construction

of the progressive mesh, and a post-process is necessary to build the geomorph structure. We address this problem by introducing a new geomorph structure (the g-mesh) which is constructed during simplification.

We present a novel mesh representation — the geomorph (or g) mesh — which greatly simplifies the generation of a continuous level of detail for arbitrary meshes. We describe the construction and application of the g-mesh, and describe how it's design allows for considerable acceleration on current graphics hardware. As the g-mesh takes only one parameter to control it's current continuous level of detail, a level of detail control system is easy to define. Using current hardware it is easy to incorporate animation techniques based on vertex transformations, such as bones[LKM01] and wires[SF98].

6.1 Continuous Level of Detail

We define a technique of continuous level of detail for surfaces of arbitrary topology. Lindstrom *et al.* [LKR+96] defines four types of continuity used in continuous LOD techniques for terrain models (described in Section 2.7). Points 3 and 4 are dependent on the viewing parameters, defining distribution and polygon density within the view frustum. These concerns do not apply our technique, which does not refine models based on the view frustum. We find that there are three types of continuity associated with multiresolution surface models:

- *Visual Continuity:* This form of continuity is the goal of all LOD blending techniques — to hide changes in the model geometry from the user. It is clear from our experience that geomorphing reduces popping to a point where discrete LOD switches are entirely unnoticeable — except possibly by a change in frame rate. Note that we only address visual discontinuities caused by geometric artefacts, and not those caused by texture.

- *Geometric Continuity:* In order to prevent surface cracks in [LKR+96] block continuity is defined in addition to geometric continuity. For an arbitrary model, i.e. one where the topology is not constrained, there are no strict boundaries to construct these blocks. However, the definition of geometric continuity encompasses these concerns. We define geometric continuity in terms of the insertion and removal of vertices in the model: *a vertex can only be inserted or removed from the surface if it is at the position of a vertex which is already present in the surface.* If we regard removed vertices as always present in the mesh (while it may be sharing its position with another vertex), this describes C_0 positional continuity. This form of geometric continuity guarantees that for a small enough change in time or distance parameters, no popping effect is noticeable.

- *Frame rate Continuity:* Slater and Steed[SS00] state that any disruption can cause a break in the feeling of presence. These can take the form of popping between LOD models (defined above as *visual discontinuities*), or a sudden change in frame rate, resulting in reduced interactivity or *lag*[Red97]. Frame rate continuity is similar to points 3 and 4 described in Section 2.7, as this is achieved by constraining the number of polygons in the scene.

Unfortunately, *frame rate continuity* and *visual continuity* may not be reconcilable (for example, when a large amount of geometry becomes visible in a new frame). In order to guarantee a

constant frame rate a predictive LOD scheme must be used (such as in [Mas99]), but this cannot constrain the **rate** at which the models change. If the transitions are performed too quickly, visual discontinuities (in the form of popping) may result.

We call situations where frame rate continuity and visual continuity cannot be maintained between frames as *volatile* scenes. A region of volatility in a scene is a sequence of frames where volatility occurs. In Figure 6.10 we show the two scenes in our experiment — the volatile scene has a region of volatility in the frames surrounding the point when the camera points down at a single dolphin, after which many dolphins are introduced into the view. In Section 6.3 we discuss the issue of volatility, show how the rate of transitions can be controlled to reduce popping.

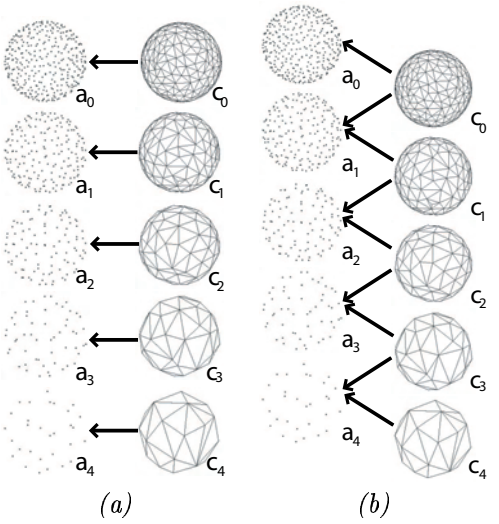


Figure 6.1: The g-mesh structure. A standard discrete LOD model sequence is shown in *(a)*, while in *(b)* we present the g-mesh structure. A g-mesh is a hierarchy of attribute (a_i) (such as vertex positions and colour) and connectivity (c_i) (the manner in which the attributes are connected) information. In *(a)* each c_i applies only to a specific attribute set a_i. Each connectivity set in *(b)* links together the two attribute sets a_i and a_{i+1}. A local interpolation parameter (defined in Section 6.2.2 as $g_i(T)$) determines how these two attribute sets are combined.

6.2 Defining the g-mesh

We introduce the geomorph mesh (or g-mesh) as a structure which greatly simplifies defining transitions between LOD models. While an ordinary LOD sequence only defines a set of unrelated models, a g-mesh is a sequence of attributes which share connectivity sets. This allows smooth transitions between these attribute sets, and hence LOD models, to be easily defined. We use the superscript to specify individual components of a set, while the subscript denotes the set number.

A g-mesh is a hierarchical level of detail sequence, defining a sequence of $n+1$ attribute information arrays $a_i, i = 0, \ldots, n$, and a sequence of $n+1$ connectivity information sets $c_i, i = 0, \ldots, n$ defining the shared connectivity information for attribute arrays a_i and a_{i+1}. This differs from a standard LOD sequence, where connectivity set c_i would only refer to attribute set a_i (see Figure 6.1).

Connectivity set c_0 has the same number of polygons as the original model at its full resolution, while connectivity set c_n is the final or simplest level of detail, which only refers to attribute array

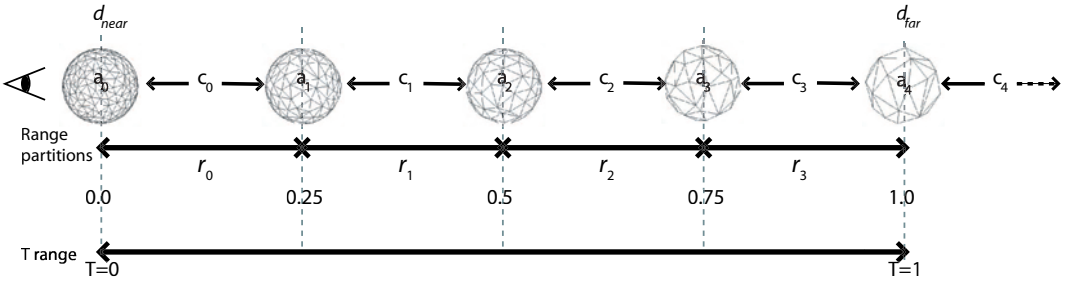


Figure 6.2: An even LOD partition. This demonstrates a LOD control mechanism, where the parameter T maps into a model representation defined by a connectivity set c_i. The function $g_i(T)$ maps the global interpolation parameter into a local interpolation parameter (see Section 6.2.2), which is used to interpolated between attribute sets a_i and $a_{(i+1)}$. r_i defines the range partitions of each LOD model in the sequence. In this even LOD partition $r_i, i = 0, \ldots, 3$ are equal in size. For $T > 1$ the least complex model representation is used, made up of $c_{(n-1)}$ and $a_{(n-1)}$ (in this case, $n = 5$).

a_n. The g-mesh is defined as

$$g = [\{a_0, \ldots, a_n\}, \{c_0, \ldots, c_n\}, n].$$

An attribute array a_i is an array of vertex attribute information. Each entry $a_i^j, j = 1 \ldots m$ is a vector of attributes including position, texture coordinates, colour and normals, where m is the number of vertices in the array a_i. The connectivity information set c_i contains index information for the drawing of the model primitives, such as vertex indices for triangle strips.

6.2.1 LOD Partitioning

We define $T \in [0, 1]$, the global interpolation factor for all transitions. This defines the range over which model transitions are defined. Interpolation starts from the user defined near distance d_{near}, and ends at the far distance d_{far} for the scene. If the model is currently at a distance d from the viewer, our parameter T is simply

$$T = \frac{d - d_{far}}{d_{near} - d_{far}}.$$

T now defines the global LOD control parameter. $T > 1$ implies that the object is past the maximum distance for LOD transition, and the lowest level of detail is visible. A value of $T < 0$ implies that the object is at its maximum resolution, or is behind the viewer and must be culled.

In order to determine which level of detail model to use — i.e. the value i of c_i — we partition the space between d_{near} and d_{far} into n segments, r_i, $i = 0, \ldots, (n-1)$, where

$$\sum_{i=0}^{n-1} r_i = 1.$$

In Figure 6.2 we demonstrate an even partition of T, where $r_0 = r_1 = \ldots = r_{(n-1)}$. This form of

partition would give each level of detail an even "slice" of the LOD partition.

6.2.2 Model Interpolation

We have shown above how the parameter T determines which connectivity set c_i is currently being used. We now need to define a function which determines the current g-mesh attributes such that geometric continuity is preserved. For this we use a basic linear interpolation function for each component of the mesh.

We define a function $F^j(T)$ which determines the current position of attribute a^j. We define R_i as the T value at which the range r_i begins:

$$R_i = \sum_{k=0}^{i} r_i.$$

We define the function $g_i(T)$ as a mapping function between the global transition parameter T to the local transition parameter for connectivity set i, that is:

$$g_i(T) = \frac{R_{(i+1)} - T}{R_{(i+1)} - R_i}$$

The function $g_i(T)$ is equivalent to the parameter t defined in Figure 2.8. Note that if T is not within the range $[R_i, R_{(i+1)}]$ the value for $g_i(T)$ will be outside of the range $[0, 1]$. We now determine the value of a particular attribute j within an attribute set i using the function f_i^j, defined as

$$f_i^j(T) = \begin{cases} (1 - g_i(T))a_i^j + g_i(T)a_{i+1}^j \text{ if } \mathrm{R_i} < \mathrm{T} \leq \mathrm{R_{(i+1)}}, \text{ or} \\ \mathbf{0} \text{ otherwise.} \end{cases}$$

It is easy to show that the sum of the functions

$$F^j(T) = \sum_{i=0}^{(n-1)} f_i^j(T)$$

is C_0 continuous (and hence geometrically continuous) on the interval $T \in (0, 1]$.

Note that according to this notation, interpolations will take place even if the connectivity information for a particular LOD c_i would not require that attribute. This would results in unnecessary interpolations in later attribute sets after an attribute has been collapsed to its final position. In practice, the need for these redundant interpolations is removed by using the *vertex program*, introduced in Section 6.4.

6.2.3 g-mesh Construction

We construct our g-mesh during the surface simplification process. We make use of the batched hierarchy, defined in Chapter 3. The mesh structure is initialised by writing the unsimplified attribute and connectivity information into the g-mesh as a_0 and c_0 respectively. After each individual batch of independent edge collapse operations has been completed, a new attribute an connectivity set is written to the mesh. Once a particular termination criteria has been met, the final attribute and connectivity sets are written to the g-mesh, along with the number of levels of

detail n.

At present, all attribute sets within a g-mesh must be of the same size, as re-indexing the attributes in an attribute set would cause the connectivity information to index the incorrect vertices during interpolation. Storage of the g-mesh can be made more efficient by storing only the change in each of the attribute sets.

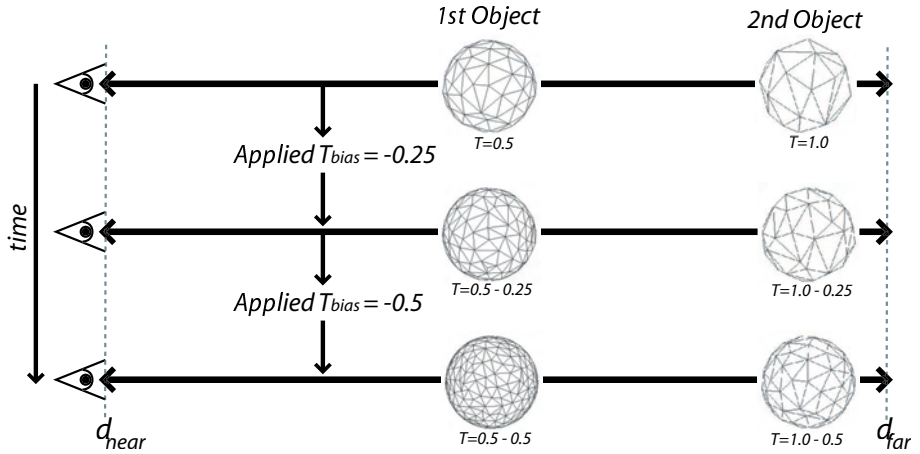


Figure 6.3: Applying a T_{bias}. By gradually incrementing the value of the T_{bias}, we can reduce visual discontinuities caused by changing the parameter T. In this example, we stabilise the frame rate by improving the overall quality of the scene. This is achieved by subtracting from the T_{bias} in each frame, gradually the overall detail in the scene.

6.3 Continuous LOD Control

A continuous LOD algorithm presents a unique challenge to frame rate control. Funkhouser and Séquin[FS93] state that the only method by which a constant frame rate can be maintained is by *prediction*. A predictive scheme attempts to maintain the desired frame rate by computing the scene complexity before rendering and adjusting the model complexity accordingly. In the case of continuous LOD sequences an unguided predictive scheme may result in "popping" when there is a sudden change in the complexity of the scene (as it would when a new object is inserted) — these cases are called *regions of volatility*, which occur when frame rate continuity and visual continuity are mutually exclusive.

We define a greedy global predictive algorithm for continuous LOD control. Note that this technique does not depend on the g-mesh structure, and can be used with any LOD model structure. The interpolation parameter T provides an effective global control over the LOD of all models in a scene. In scenes containing several objects, a bias can be added or subtracted from the global interpolation parameter T during rendering to increase or reduce the complexity of all the models in the scene. We call this bias T_{bias} (this is explained in Figure 6.3). We call the change which is applied to T_{bias} in each frame as T_{change}.

6.3.1 Determining T_{bias}

In order to ensure that the least possible correction is made to ensure that a frame rate (or in this case polygon count) is maintained, we define a new global LOD control algorithm for determining the optimal value for the parameter T_{bias}. Our goal is to restrict the number of polygons in the scene to a desired value p, with a permitted tolerance of ϵ.

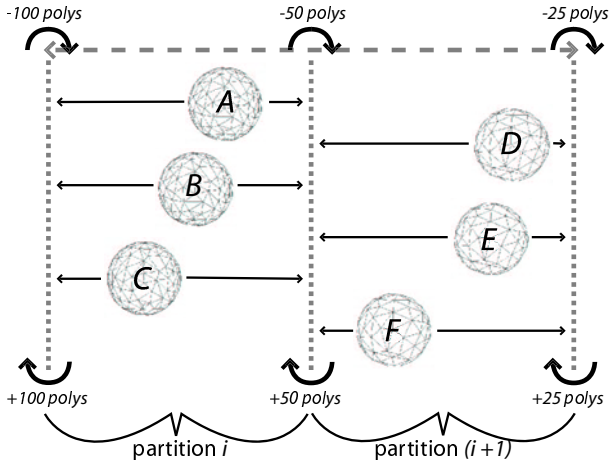


Figure 6.4: A continuous level of detail control system. In this diagram the six objects are scattered across two consecutive LOD partitions. If the number of polygons in the scene needs to be reduced by 50 polygons, we firstly construct LOD control structures for each object, and sort them in increasing order on the value of the parameter D_{next}. The ordering of the objects in the array is now $\{D, A, E, B, C, F\}$. Shifting object D across the LOD partition causes a polygon change of only 25, but shifting A across the LOD partition causes an accumulated polygon change of 75 $(25 + 50)$, which is sufficient to balance the number of polygons in the scene. The adjustment to T_{bias} is then the T_{change} necessary to make the object A cross over the LOD partition.

Before we draw each frame, we determine the number of polygons within the scene q, and determine the required polygon correction $k = p - q$. For each object in the scene we define a control structure consisting of $\{D_{prev}, D_{next}, P_{prev}, P_{next}\}$. D_{prev} and D_{next} represent the minimum T_{bias} necessary to change to the previous connectivity level or next connectivity level respectively. Similarly P_{prev} and P_{next} is the number of polygons that would be removed from or added to the scene after changing to the previous or next connectivity level respectively.

We define an algorithm for level of detail control as follows:

1. set current $T_{change} = 0$
2. count the number of polygons in the scene q
3. build an array of LOD control structures — each entry is an object in the scene
4. sort the array on either the D_{next} or D_{prev}, depending on whether $k = p - q$ is negative or positive respectively
5. set $i = 0$
   (a) if $k > 0$, reduce q by the P_{next} of the i_{th} element of the array, else if $k < 0$, increase q by the P_{prev} of the i_{th} element of the array
   (b) if the sign of k has not been changed (i.e. the required polygon number has not yet been reached) and i is less than the array size then increment i and goto 5a
   (c) add D_{next} or subtract D_{prev} of the i_{th} element from T_{change}, depending on whether $p - q$ is negative or positive respectively
6. count the number of polygons in the scene q
7. if $|p - q| > \epsilon$, goto 3
8. $T_{bias} = T_{bias} + T_{change}$

An example of using this algorithm for determining the value of T_{bias} is shown in Figure 6.4.

Note that in Figure 6.4, the correction overshoots the desired correction by 30 polygons. If there is no change in the following frame it would be undesirable for the LOD correction scheme to "counter-correct" so few polygons, as it would result in an unnecessary oscillation about the desired polygon number (which it may be impossible to reach). For this reason we permit a region of tolerance about the desired polygon number, ϵ. The maximum number of polygons which can be introduced or removed from a single object is the number of faces in the highest resolution mesh c_0 subtracted from the number of faces in c_1, assuming that every object has a unique value for D_{prev} and D_{next}. We use a tolerance of $\epsilon = (|c_0| - |c_1|)$.

6.3.2 Maintaining Visual Continuity

Although the predictive LOD correction scheme described above guarantees continuity in terms of the frame rate, it cannot guarantee visual continuity. Popping during model interpolations between frames can be reduced by restricting the amount by which the T_{bias} changes (called T_{change}) to ensure that the visual difference is minimised. Unfortunately surfaces do not degrade linearly as they geomorph to the lower resolution versions, as interpolated edges become longer as the level of detail increases.

In order to add some form of control to this T_{change} based on the image quality of the model, it would be necessary to either change the T_{change} dynamically according to which LOD partition the object is in, or shift the LOD partitions so that T_{change} can remain constant. As we would like a global T_{bias} value which is applied to all models in the scene, adjusting the T_{change} for each

model is not possible, so we adjust the individual partitions of each object.

6.3.3 LOD Partitioning based on Image Quality

In Chapter 4 it has been shown that the decay of the volume of a simplified model correlates significantly (-0.75) with the decrease in the measured image quality, based on the L_1 norm, and correlates significantly (-0.89) with the decline in the quality of the objects silhouette (see Figure 4.5). Therefore we use the decline in the volume of the model to construct LOD partitions which approximate the image quality of a model.

The volume of the object is measured after each batch has been completed during simplification. This volume is divided by the original volume of the object to yield a normalised volume measurement $V_i, i = 0 \ldots n$ for each batch n ($V_0 = 1$, as it represents the volume of the unsimplified model). We use these volume measurements to determine the ranges $r_i = V_i - V_{i+1}, i = 0 \ldots (n-1)$. An example of shifting the volume partition is shown in Figure 6.5, while an example of a shifted LOD sequence based on volume degradation is shown in Figure 6.6.

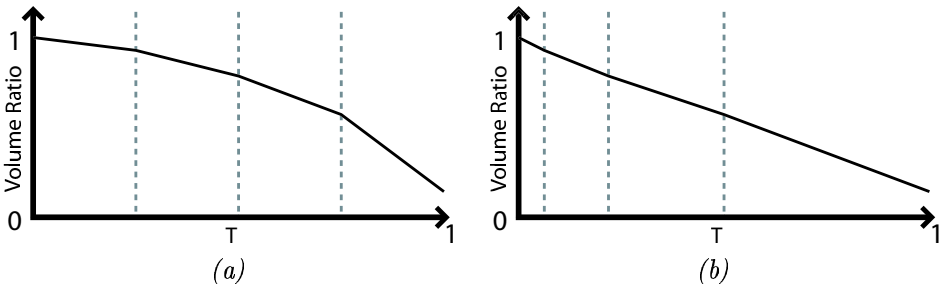


Figure 6.5: Shifting the LOD partition. As the surface is simplified, the volume of the enclosed object shrinks exponentially (a). In (b), we adjust the LOD partitions (depicted by the vertical dashed lines) to make ensure that the volume shrinkage is close to linear in T.

Generally this form of partitioning of the LOD models causes the ranges of the partition to be increasing as the detail increases, i.e. $r_0 \leq r_1 \leq \ldots \leq r_{n-1}$. This implies that the largest portion of the geomorph sequence $T \in [0, 1]$ is spent within the lowest level of detail partition, ensuring smoother transitions at the level when the model in its lowest resolution.

Note that objects in the scene may have user defined partitions $r_i, i = 0 \ldots n-1$. This can be used to weight some objects in the scene with more importance (and more detail) than others in the scene, and can effectively guide the focus of the viewer to objects of higher detail.

6.3.4 Deciding on a minimum T_{change}

Several authors [FS93, Red97] have noticed that viewers are more tolerant of lower detail and visual discontinuities in situations when the model is moving at a high velocity, or when the viewers focus is shifted elsewhere. In these situations, it is permissible to use a larger (or unconstrained) value for T_{change} for these models between frames, as these changes would be unnoticeable.

Fixing the value of T_{change} can have side effects in scenes with a great deal of *volatility*, when the LOD control system described in Section 6.3 cannot keep up with the changes taking place. This kind of volatility is often experienced when exposing large amounts of previously occluded geometry (e.g. through a portal).

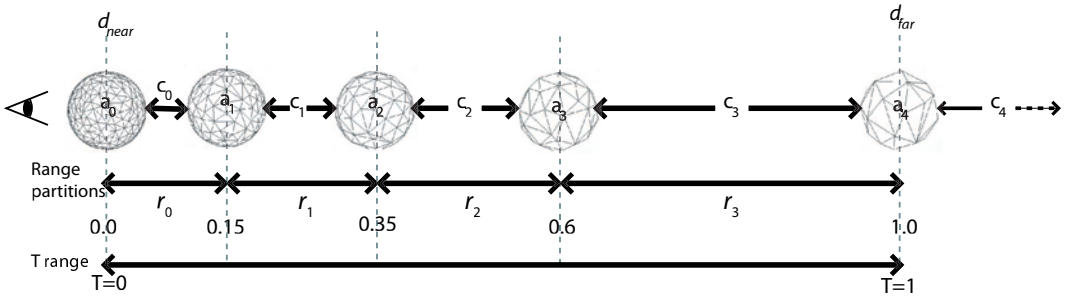


Figure 6.6: A shifted LOD partition based on approximated image degradation. The definitions for $r_i, i = 0 \ldots 3$, T and $a_i, c_i, i = 0 \ldots 4$ are the same as in Figure 6.2. The partitions r_i are built by approximating image degradation with volume degradation.

We find it difficult to notice *visual discontinuity* when moving through a scene, but interactivity becomes essential. In order to ensure that we do not lose control of the frame rate, we allow the T_{change} to remain unconstrained in circumstances where large amounts of new geometry could be exposed between frames (for example, while the user is moving through the scene). An unconstrained frame rate guarantees *frame rate continuity*. While the viewer is stationary, we restrain T_{change} to minimise visual discontinuities.

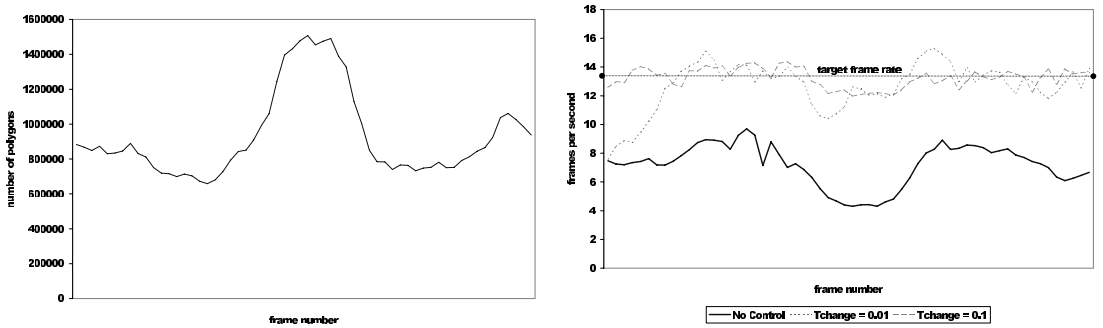
In our experiments we have evaluated a small T_{change} parameter of 0.01, and a large parameter of 0.1 to determine how well the LOD control mechanism recovers from frame rate fluctuations with each parameter (see Figure 6.7). In Figure 6.8 we show that a higher maximum value for T_{change} allows for more drastic visual changes to be permitted in order to stabilise the frame rate. This parameter could more accurately be determined on a per model basis by using visual experiments such as the *just noticeable difference* test [Web78]. A visual measurement would more accurately measure a perceived deviation in model quality, but this is an area for future research.

6.4 Implementation

In order to construct the g-mesh file we make use of the Generic Memoryless Polygonal Framework and the batched hierarchy described in Chapter 3. Each attribute set a_i and connectivity set c_i is written to the g-mesh file at the start of the process and after the completion of each batch. This process is run until some stopping criteria has been met, such as a face or vertex limitation. We have not experimented with different error metrics for this technique, but used the hybrid scheme E_{hybrid} (defined in Chapter 3 due to its good preservation of normal and volume attributes.

The visual platform implementation was written in *C++* with the GL Utility Toolkit (glut). A number of extensions were employed to ensure optimal frame rates. In order to improve the performance of the many mesh interpolations required during the generation of the desired model, we make use of the *vertex program*[LKM01] extension available for the NVidia GeForce 3. A graphic overview of our vertex program is shown in Figure 6.9. The SIMD architecture of vertex programmable hardware permits parallel computation of vertex interpolations as part of the rendering pipeline.

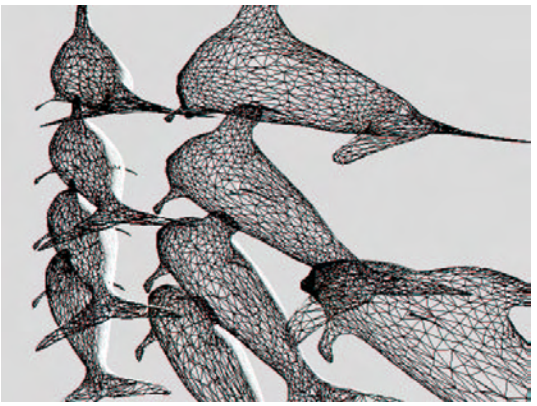
We used a derivative of the technique of Singh and Fiume[SF98] to perform animation. The



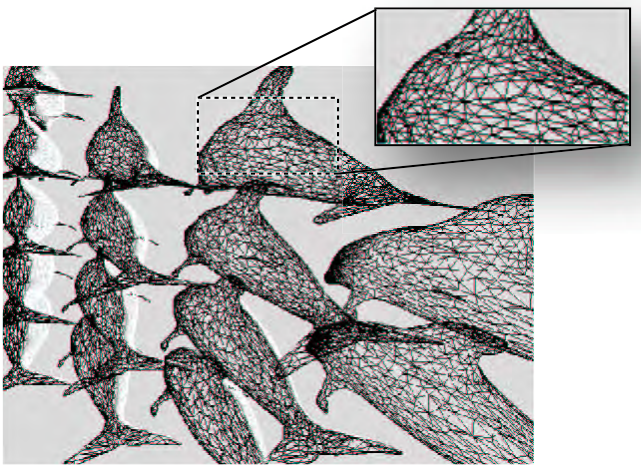
(a) non-volatile scene *(b) non-volatile scene*

(c) volatile scene *(d) volatile scene*

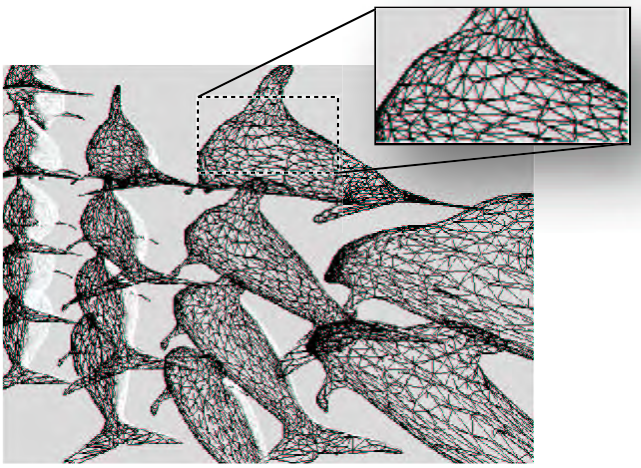
Figure 6.7: Controlling the number of polygons in the scene. Each graph depicts the measured number of polygons or frames per second over 120 frames of our predefined path. In *(a)* and *(c)* the polygon counts for uncontrolled non-volatile and volatile scenes are shown respectively. Note that in *(c)* the number of polygons in the scene drops suddenly at the point where the camera looks down (as in Figure 6.10 *(b)*). In *(b)* a T_{change} of 0.01 is capable of maintaining the frame rate close to the desired level. It is clear than in *(d)* a T_{change} value of 0.1 is better at handling frame rate discontinuities than the smaller value. The peak results from there being insufficient dolphins in the scene to allow for the required number of polygons. Note that this can easily be corrected by simply waiting until the desired time per frame has been reached.



(a) $T_{change} = 0.0$ (previous frame)



(b) $T_{change} = 0.01$



(c) $T_{change} = 0.1$

Figure 6.8: T_{change} and visual discontinuities in a volatile scene. *(a)* depicts the contents of the frame preceding both *(b)* and *(c)*. Note that the number of dolphins in frames *(b)* and *(c)* are nearly twice that of frame *(a)*. In order to quickly correct the frame rate, a higher value for T_{change} (in *(c)*) produces a coarser mesh representation than in *(b)*.

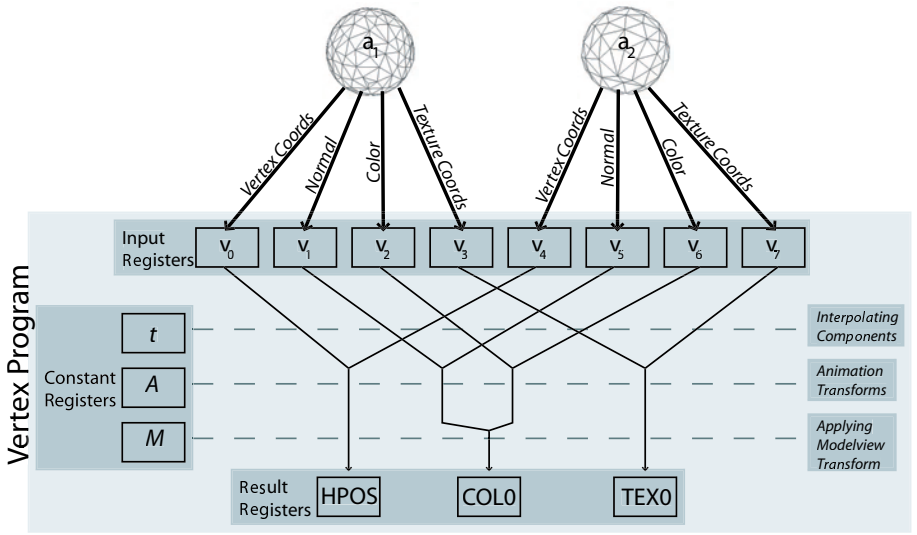


Figure 6.9: Our Vertex Program. The two attribute sets a_1 and a_2 are passed through input registers $v_i, i = 0 \ldots 7$. The first stage combines these attributes with the t parameter, stored in a constant register. After the points in the mesh are determined, an animation routine can be applied (such as bones or wires). The transformed points are then transformed into homogenous coordinates and clipped using the modelview matrix M and the world view projection matrix. The results are written to the output registers HPOS (homogenous position), COL0 (colour) and TEX0 (texture coordinates).

constant registers are used to store the wire parameters and an interpolation parameter (all animation constants are referred to in Figure 6.9 as A). This can be used to make our dolphins tail flap in the scene.

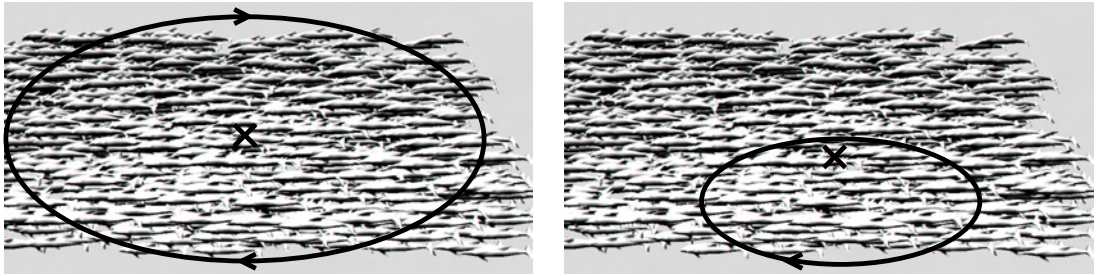
An important concern is the use of memory for repeated models. As model attribute and connectivity data must be accessed quickly it is stored in video memory. So as not to repeat this data we instantiate only one data object, and create model objects which store their own state and point into the data stored on the graphics card.

Experimental Design and Results

We design a test platform based on the implementation described. We make use of an un-textured dolphin model for our experiments. The g-file for the dolphin has 6 levels ($n = 6$) and varies from 10000 to 200 faces. We initialise our scene with 1000 dolphins, each of which is translated to be within the region within which model interpolation is defined (i.e. between d_{near} and d_{far}). Objects outside of the view frustum are culled.

Two paths are tested for our evaluation. The first (Figure 6.10*(a)*) describes a non-volatile scene, as there are relatively small fluctuations in the number of polygons in the scene. Our LOD control system maintains a steady frame rate by restricting the number of polygons in the scene exactly within the tolerance $\epsilon = 5000$. The second (Figure 6.10*(b)*) describes a volatile scene, where frame rate fluctuations cannot be contained (see Figure 6.7), as there are insufficient polygons in the scene to achieve the required frame rate.

The use of programmable hardware is essential for interactive display of the g-mesh. Our experiments with the non-volatile path (in Figure 6.10(a)) show a speed up from 71.42 *seconds*



(a) non-volatile scene *(b) volatile scene*

Figure 6.10: Two test scenes. Our test scenes consist of a thousand dolphins, with $|c_0| = 10000$ and $|c_6| = 200$ polygons. Our camera moves through the scene along the described path, always pointing at the 'X'. In *(a)* a simple circular path about the scene is described. In this case, there is usually a small change in the polygon counts of consecutive frames. In *(b)* the camera path travels through the centre of the scene, at which point the camera points straight down. The frames surrounding and including this point make up our *region of volatility* (this is shown more clearly in Figure 6.7(c)).

per frame in the case of software interpolation, to between 12 and 14 *frames per second* on an Athlon 500 Mhz processor using an ASUS v8200 GeForce 3. This is attributed to the large number of interpolations which is required within each frame of the sequence (for 1000 dolphins at there highest level of detail, each frame would require the interpolation of 10 million attributes). These computations are significantly faster when performed using programmable hardware.

Although volume is a good measure of geometric degradation, we find that texture sliding is a major visual distraction. The large changes that occur to the model when close to the camera position causes the textured on the surface to slide considerably. The work of Sander *et al.*[SSGH01] would significantly reduce this artefact, and could easily be adapted during model simplification. This is, however, an area for future work.

6.5 Summary

In this chapter we have defined a new mesh structure, the g-mesh, which can be constructed during mesh simplification. The g-mesh guarantees geometric C_0 continuity between frames due to the nature of the model interpolation between attributes. Interpolation between attribute sets of the g-mesh is facilitated by using programmable hardware, which substantially accelerates the rendering time and relieves the CPU from unnecessary computations.

We define a greedy predictive LOD control mechanism, based on the global LOD control parameter T. A bias is added or subtracted from the T value of all models in the scene in order to reach the desired polygon number within each frame. In volatile scenes however, a predictive scheme may results in visual discontinuities when the bias added between frames causes objects in the scene to change several LOD levels between frames. For this reason we constrain the maximum change to bias between frames. We have tested our LOD control system with two different scenes, and show the performance of our LOD correction scheme with two different parameters for the maximum change to the bias between frames.

Chapter 7

Conclusion and Future Work

In this dissertation we have made several extensions and improvements to the field of multiresolution meshes with local connectivity. Using the method of Hoppe [Hop96] as a basis, we develop a generic framework for mesh decimation (in Chapter 3). This generic framework allows us to evaluate the error metrics of existing and new simplification schemes under the same conditions. We use the results of these evaluations and the generic platform to explore two applications of multiresolution analysis — view-dependent refinement and smooth geometric transformations between model representations.

The tools which we present in this dissertation offer better control over the tradeoff between model quality and interactivity. The image error of a simplified model can be quickly approximated with a simple volume measure. Our selective refinement platform guarantees a fixed triangle budget, which can guarantee interactivity on low-end graphics platforms. A method of continuous LOD control is presented which guarantees a desired polygon count within each frame. Each of these applications of multiresolution analysis can be used to gain tighter control over the quality and interactivity of the rendered scene.

We offer a number of contributions to the field of multiresolution analysis. Our novel generic framework for simplification provides the basis for adaptive and batched simplification of surface models, which is exploited by our new structure for facilitating smooth model transitions using programmable graphics hardware. Our new view dependent refinement framework is more compact than existing methods, and provides a solution to the problem of selective refinement over limited bandwidth mediums such as the Internet. Our evaluation of error metrics for surface simplification yield a significant correlation between image based degradation and the declining volume of the model. This heuristic proves invaluable in applications where model quality must be estimated at run-time, and is used with our level of detail control system in order to determine the length of each LOD partition.

7.1 A Generic Memoryless Polygonal Simplification Platform

In Chapter 3 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 these memory resident optimisations may improve performance, a memoryless derivative requires less storage and often produces better results. We show how two existing error metrics, that of Hoppe[Hop96] and Garland and Heckbert[GH97], 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.

7.2 Evaluation of Memoryless Simplification

In Chapter 3 a number of error metrics suitable for surface simplification using subset vertex placement are presented. These include E_{edge}, E_{uvol} and E_{hybrid}. In Chapter 4 we evaluate each of the error metrics introduced, including our memoryless implementations of the commonly used progressive meshes [Hop96] (E_{pm}) and quadric error metrics [GH97] ($E_{quadric}$).

We present several criteria for evaluating the quality of models generated through simplification, including image based metrics (K_{L_1}, K_{L_2} and K_{sil}) and model based measurements, such as model volume and surface distortion (measured with the *Metro* package). Using these evaluation criteria we analysed the various simplification techniques on a large number of un-textured surface models. The results of these experiments are summarised in Section 4.6. These results have several implications for the field of surface simplification:

- E_{edge} is an effective error metric for applications requiring a quick method for simplifying large un-textured scenes (such as terrain models). Because of the speed (an average of $4.3 \times 10^{-7}s$ for each calculation in our implementation) and the prevention of degenerate triangles (according to Section 4.5) E_{edge} is a good technique of approximating large surfaces by simpler versions.

- There is no visual improvement (according to our image-based criteria) in using an unconstrained (or optimal) vertex placement technique over subset placement. Subset placement is quicker to compute (2 bit subset placement with the error measure E_{uvol} takes only $6.56 \times 10^{-6}s$, while optimal vertex placement $0.000223s$ for each error calculation), and in the case of a progressive representation is significantly cheaper to store (as shown in [PR00]).

- While the Hausdorff distance is employed as a standard technique of evaluating the quality of a compressed model in *model space*, it is not necessarily a good measure of the *visual quality* of the model. We have shown that a simple metric based on the volume of the compressed surface corresponds better with our visual measures. A volume measure is significantly quicker to calculate than the relatively cumbersome and slow Hausdorff calculation, and several techniques already use a volume measure as a simplification metric.

7.3 View Dependent Refinement

In Chapter 5 we have described a technique to facilitate selective and progressive refinement based on the output from the application developed in Chapter 3. We have presented several enhancements over existing techniques, such as those presented in [Hop98, XESV97, TLG99].

We present the concept of a "visibility sphere", which is a convenient method of organising visibility information. It encapsulates the three components necessary to approximate the visibility of a single atomic operation, including information necessary to determine if the operation is within the view, is back-facing or does not contribute anything to the image.

The vertex split hierarchy is also introduced, which provides a more compact ordering of refinement operations than the standard vertex hierarchy. Our operations are inserted into a directed acyclic graph, eliminating the need for the repeated nodes required in the tree structure used in a vertex hierarchy. This more efficient format affords an average of a 30% saving in the memory required to store the structure.

We adapt our view dependent refinement technique to a client/server framework, and discuss a number of optimisations, such as the client cache, which can greatly reduce the number of operations which must be transmitted to a client. The stateless client system described gives the client a great deal of flexibility with respect to interactivity and control of the transmission process.

7.4 Creation and Control of Continuous Level of Detail

In Chapter 6 we have defined three types of continuity in level of detail sequences. Visual discontinuity, which is caused by popping between different levels of detail, can be minimised with the use of geometric continuity. Unfortunately visual continuity and frame rate continuity cannot be guaranteed in volatile scenes.

The g-mesh structure offers a comprehensive solution to the problem of continuous level of detail in highly populated scenes We have defined this structure, and described its constructed using a batched hierarchy during simplification. Every point location of the resulting mesh structure has C_0 (positional) continuity, which is an important attribute when ensuring the minimisation of visual discontinuities.

We describe a level of detail control mechanism based on the global control parameter T, whereby the number of polygons in the scene can be accurately controlled by means of a single value used to correct the current polygon number. By restraining this this correction and shifting the level of detail partitions defined for each object, we are able to place an upper limit on visual discontinuities caused by sudden level of detail changes.

We have also shown that programmable hardware is essential for real-time rendering of large continuous level of detail scenes. Programmable hardware is also useful when mixing level of detail interpolations with other forms of animation, such as wires or bones. We show that our level of detail control implementation is capable of ensuring the desired polygon count.

7.5 Future Work

There are several areas of future research which stem from our work. At present, our simplification 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 neighbours) 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[PH97] or Garland and Heckbert[GH97] to allow unconnected vertices to be contracted during model simplification.

Our system for selective and progressive transmission could easily be extended to permit texture information transmission for meshes with local connectivity. The technique of Certain *et al.*[CPD+96] transmits view independent textures and progressively, but the topic of view-dependent texture transmission has, to our knowledge, not yet been addressed. The implications of including operations other than the vertex split in our operation dependent hierarchy, such as the vertex merge operation[PH97], has yet to be explored.

Minimising visual discontinuities based on the degradation of volume of the enclosed surface has been shown to work only with un-textured models — we find that texture sliding is a substantial artefact in our continuous level of detail system. The work of Sander *et al.* [SSGH01] provides a method for minimising texture deviation, which could be used to derive better texture coordinates during simplification, but a method for determining visually better LOD partitions based on textured models is an open problem.

We have shown that attributes of the g-mesh are C_0 continuous over the range of interpolation. Higher orders of continuity could quite easily be constructed based on these positions. Although we have great difficulty perceiving visual discontinuities in un-textured models with C_0 continuity, higher levels of continuity may introduce further improvements in reducing popping artefacts. A derivative of the g-mesh structure could also be used to define morphing between a sequence of different models with no distinct one-to-one mapping. This technique would require models with topological equivalence, and a method would be needed to find point correspondences between these models.

Appendix A

Wavelets for Multiresolution Analysis

The application of Wavelets to the Multiresolution Analysis of surfaces provides an elegant, mathematically rigorous framework for the implementation of subdivision surfaces. This appendix serves to give the reader a better understanding of the complexities of using Wavelets as a Multiresolution testbed. We discuss a method similar to Lounsbery [Lou95], and describe its implementation.

A.1 Background

A wavelet or "wavelet-like" transform is commonly used for generating multi-resolution representations. Broadly speaking, a wavelet representation of a function is a coarse overall approximation together with detail functions at various scales. The original surface can be reconstructed using only the *base mesh* M^0 and by consecutively applying i sets of detail functions the model can be reconstructed to level M^i.

These coefficients are a measure of the "importance" of each vertex to the model, or rather the error incurred by it's removal. By reducing coefficients of the lowest magnitude to zero iteratively the vertices of least error are removed, and the model's detail is reduced. Wavelets lend themselves naturally to progressive transmission, where coefficients are transmitted in the order of their magnitude. For more details regarding the applications of wavelets of multi-resolution analysis, the reader is referred to [SDS95].

"Wavelet-like" techniques [BG98, Bon98, COLR99] do not make use of the traditional filter-based approach of surface analysis, choosing to construct an ad-hoc subdivision of the surface at each level of analysis. These techniques require their own filters (or use a simple Haar filter, as in [BG98, Bon98]), and more adaptive filters cannot be applied to the particular subdivision technique. In [Lou95] different wavelet filters can be constructed depending on the desired surface properties.

In [Lou95, SDS95, SS95, CPD+96, GSS99, KSS00, GVSS00, PSS01] subdivision schemes are used in order to differentiate clearly between distinct levels of resolution. In [SS95] the analysis is based only on surfaces which are topologically equivalent to a sphere, while in [Lou95, SDS95, CPD+96] wavelets have been extended to apply to surfaces of any arbitrary topology. By making

use of a subdivision scheme (such as [CC78, Doo78, Loo87]) a hierarchy of levels of resolution are clearly defined, since M^{i+1} is M^i after each edge has been subdivided. [CPD+96] extended this work to allow for colour and texture information to be utilised.

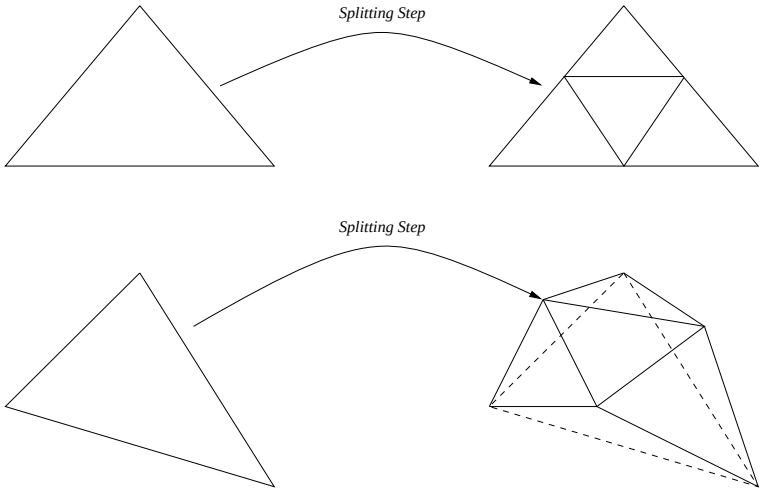


Figure A.1: Quadrisection. New vertices are inserted into the mesh at each triangles midpoint. In this way, four new faces replace the original triangle. This is a form of uniform surface subdivision.

The above schemes can only be applied to meshes which are "semi-regular". A regular mesh is one in which every vertex has exactly six edges originating from it, except at surface discontinuities. The number of edges originating from a vertex is referred to as the *valence* of a vertex. An example of a regular mesh would be a terrain height field, where the surface would be a rectangular grid of evenly spaced points. Each vertex in a semi-regular mesh is constrained to have a valence of six, except vertices which are present on the base mesh M^0. This allows the model to have folding and more complicated topology. Meshes of this nature are typically generated with quadrisection, shown in Figure A.1. Khodakovsky *et al.* [KSS00] show that extremely effective surface compression is possible using semi-regular wavelets.

In [GVSS00] *Normal Meshes* are introduced. This model representation makes use of the fact that in meshes with subdivision connectivity M^{i+1} has new vertices introduced at the midpoints of the edges of M^i — allowing each new vertex to be stored as a single float value representing the displacement of the new vertex in the direction of the face. Normal meshes can be thought of as an efficient compressed format for meshes by eliminating redundant connectivity information.

The work of Praun *et al.* [PSS01] show how semi-regular meshes which share a similar base mesh parameterisation can be consistently parameterised onto the same base mesh. Such a parameterisation is useful for a number of applications, ranging from transfer of texture coordinates to principal component analysis and mesh morphing.

Semi-regular meshes feature very seldom in practice. Mesh regularisation techniques [EDD+95, LSS+98] (which incur error), and techniques to convert volumes directly to semi-regular meshes [WDSB00] are the conventional source of meshes with this property.

In order to address the deficiencies of wavelet schemes based on semi-regular subdivision, [GSS99, DGSS99] offer a framework based on a graph of irregular connectivity . In [GSS99] a Burt-Adelson pyramid is used to define the subdivision hierarchy, and permits the construction of a hierarchy of resolutions.

A.2 Theory

As in Lounsbery et. al [Lou95], the technique implemented makes use of a semi-regular multi-resolution framework with biorthogonal surface wavelets. The underlying theory of wavelets and multi-resolution analysis will not be detailed here. For more information regarding terminology and theory the reader is referred to [Mal89, Dau92].

The underlying algorithm is relatively simple: Given a base-mesh / final-mesh pair (M^0, M^n respectively), the algorithm generates n hierarchical levels of resolution such that $M^0 \subset M^1 \subset \cdots \subset M^{j-1} \subset M^n$. Note that in order to generate mesh M^j each triangle $t \in M^{j-1}$ is split into 4 by introducing new points at the midpoints of each of the edges of t (as in Figure A.1).

A.2.1 Refinable Basis Functions

We define $\phi_i^j(\mathbf{x})$ to be the ith scaling function at resolution j, while $\mathbf{x}$ represents a point over the domain. $\mathbf{\Phi^j(x)}$ is hence defined as the matrix consisting of the i functions $\phi_i^j(\mathbf{x})$. From previous work [Lou95] these functions have been proven to be refinable, and hence $\phi_i^j(\mathbf{x})$ can be written as a linear combination of the functions $\phi_i^{j+1}(\mathbf{x})$.

We now write the matrix $\mathbf{\Phi^j x}$) as

$$\mathbf{\Phi^j(x)} = \left[\mathbf{O^j(x)} \;\; \mathbf{N^j(x)}\right],$$

where $\mathbf{O^j(x)}$ consists of the scaling functions $\phi_i^{j+1}(\mathbf{x})$ associated with the old vertices of M^j, while $\mathbf{N^j(x)}$ refers to the scaling functions associated with vertices added to the last mesh.

The refinability of the scaling functions allows us to define a matrix $\mathbf{P^j}$ such that

$$\mathbf{\Phi^j(x)} = \mathbf{\Phi^{j+1}(x)P^j}. \tag{A.1}$$

A.2.2 Wavelet Construction

The biorthogonal surface wavelet construction scheme of [SDS95] uses lifting to construct wavelets which are "nearly orthogonal" to the scaling functions. The strategy employed is to construct "lazy wavelets" $\mathbf{\Psi_{lazy}^{j-1}(x)}$ consisting of the scaling functions associated with the midpoints of the edges of M^{j-1}.

These wavelets are far from orthogonal to the scaling functions. To make them "more orthogonal" we subtract a linear combination of nearby coarse scaling functions, and define the improved wavelet as:

$$\psi_i^{j-1}(\mathbf{x}) = \phi_{\mathbf{m,i}}^{\mathbf{j}}(\mathbf{x}) - \sum_{\mathbf{k}} \left(\mathbf{s_{k,i}^j}\phi_{\mathbf{k}}^{\mathbf{j-1}}(\mathbf{x})\right) \tag{A.2}$$

where k is restricted to a few values corresponding to the vertices in M^{j-1} in the neighbourhood of $\phi_{m,i}^j(\mathbf{x})$. To find the values of $s_{k,i}^j$ we take the inner product of each of the terms in A.2 with the scaling function $\phi_{i'}^{j-1}(\mathbf{x})$ for all i' such that the support of $\psi_i^{j-1}(\mathbf{x})$ overlaps with $\phi_{i'}^{j-1}(\mathbf{x})$, and then set $\left\langle \psi_i^{j-1} | \phi_{i'}^{j-1} \right\rangle = 0$. This results in

$$\sum_k \left(s_{k,i}^j \left\langle \phi_k^{j-1} | \phi_{i'}^{j-1} \right\rangle\right) = \left\langle \phi_{m,i}^j | \phi_{i'}^{j-1} \right\rangle. \tag{A.3}$$

It is convenient to write the inner product of $\langle f|g\rangle$ in matrix form. Due to the bilinearity of the inner product:

$$\langle f|g\rangle = \mathbf{g^T I^j f}$$

where $\mathbf{f}$ and $\mathbf{g}$ are column matrices consisting of the coefficients of f and g respectively, and $\mathbf{I^j}$ is the square matrix whose i, i'-th entry is $(\mathbf{I^j})_{\mathbf{i,i'}} = \left\langle \phi^{\mathbf{j}}_{\mathbf{i}} | \phi^{\mathbf{j}}_{\mathbf{i'}} \right\rangle$.

It can be shown using equation A.1 that the following recurrence relation exists:

$$\mathbf{I^j} = (\mathbf{P^j})^{\mathbf{T}} \mathbf{I^{j+1}} \mathbf{P^j}. \tag{A.4}$$

We write the equation of A.3 in matrix form, and using the recurrence relationship described in A.4 we derive

$$\left[\left\langle \mathbf{\Phi^j} | \mathbf{\Phi^j} \right\rangle\right] \mathbf{S^j} = (\mathbf{P^j})^{\mathbf{T}} \left[\left\langle \mathbf{\Phi^{j+1}} | \mathbf{N^{j+1}} \right\rangle\right] \tag{A.5}$$

where $\left[\left\langle \mathbf{\Phi^j} | \mathbf{\Phi^j} \right\rangle\right]$ is simply $\mathbf{I^j}$, $\mathbf{S^j}$ is the matrix of the coefficients $s^j_{k,i}$ and $\left[\left\langle \mathbf{\Phi^{j+1}} | \mathbf{N^{j+1}} \right\rangle\right]$ is the submatrix of $\mathbf{I^{j+1}}$ that consists only of the columns that correspond to the members of N^{j+1}.

A.2.3 A Filterbank algorithm

Let $\psi^j_i(\mathbf{x})$ denote the ith locally supported wavelet approximation, and let $\Psi^j(\mathbf{x})$ be the row matrix of these functions. Define the analysis and synthesis filters by

$$[\mathbf{\Phi^j(x)} \;\; \mathbf{\Psi^j(x)}] = \mathbf{\Phi^{j+1}(x)} [\mathbf{P^j} \;\; \mathbf{Q^j}], \tag{A.6}$$

and

$$\begin{bmatrix} \mathbf{A^j} \\ \mathbf{B^j} \end{bmatrix} = [\mathbf{P^j} \;\; \mathbf{Q^j}]^{-1} \tag{A.7}$$

respectively.

For lazy wavelet construction, we simply define

$$[\mathbf{P^j_{lazy}} \;\; \mathbf{Q^j_{lazy}}] = \left[\begin{array}{cccc|cccc} 1 & 0 & \cdots & 0 & & & & \\ 0 & \ddots & \ddots & \vdots & & & & \\ \vdots & \ddots & \ddots & 0 & & & \mathbf{0} & \\ 0 & \cdots & 0 & 1 & & & & \\ \hline & & & & 1 & 0 & \cdots & 0 \\ & & \mathbf{P^j_m} & & 0 & \ddots & \ddots & \vdots \\ & & & & \vdots & \ddots & \ddots & 0 \\ & & & & 0 & \cdots & 0 & 1 \end{array}\right] \tag{A.8}$$

where $\mathbf{P^j_m}$ is merely the matrix of connectivity information of the new vertices at step j. In order to construct what is referred to in [SDS95] as "k-disk" wavelets the above matrices are modified by the matrix $\mathbf{S^j}$ defined in equation A.5 in the following fashion:

$$[\mathbf{P^j_{kd}} | \mathbf{Q^j_{kd}}] \;\; = \;\; [\mathbf{P^j_{lazy}} | \mathbf{Q^j_{lazy}} - \mathbf{P^j_{lazy}} \mathbf{S^j}] \tag{A.9}$$

$$\left[\begin{array}{c} \mathbf{A^j_{kd}} \\ \hline \mathbf{B^j_{kd}} \end{array}\right] = \left[\begin{array}{c} \mathbf{A^j_{lazy}} + \mathbf{S^j B^j_{lazy}} \\ \hline \mathbf{B^j_{lazy}} \end{array}\right] \tag{A.10}$$

Let $\mathbf{V^j}$ denote the column vector of vertices of M^j, and $\mathbf{W^j}$ denote the corresponding matrix of wavelet coefficients. **Analysis** can be defined by

$$\mathbf{V^j} = \mathbf{A^j V^{j+1}} \tag{A.11}$$

$$\mathbf{W^j} = \mathbf{B^j V^{j+1}} \tag{A.12}$$

while **synthesis** is defined by

$$\mathbf{V^{j+1}} = \mathbf{P^j V^j} + \mathbf{Q^j W^j}. \tag{A.13}$$

A.3 Implementation

For the implementation of this wavelet scheme, it was decided to stay as close to the theoretical basis as possible, using optimisations such as sparse matrix structures and numerical solutions where possible. It can be divided into different sections.

A.3.1 Inner Product Matrix Calculation

Thanks to equation A.5 it is not necessary to calculate the coefficients of the matrix $\mathbf{I^{j,j+1}}$, (i.e. $[\langle \mathbf{\Phi^j} | \mathbf{\Phi^{j+1}} \rangle]$) - it is sufficient to calculate only the inner product of the function with itself. The calculation of the matrix depends on the extent of the *k-disk* required - in this implementation a 1-disk wavelet will be described. *Figure A.2* shows the two dimensional inner product calculation of a 1-disk wavelet.

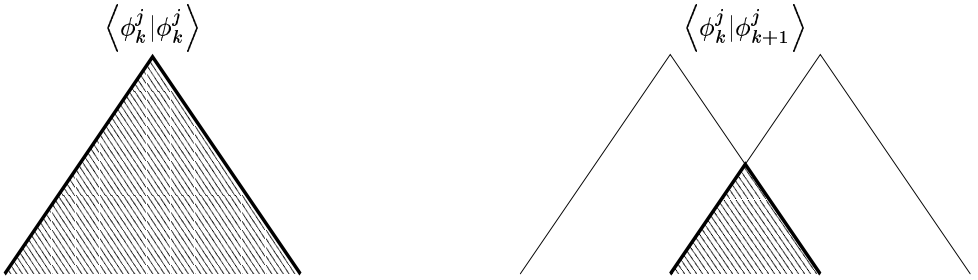


Figure A.2: The inner product in 2 dimensions. The shaded region represents the overlap and hence represents the shape of the region created by the product of the two functions.

When translated into three dimensions, the problem is harder to visualise, as it does not actually correspond to logical three dimensional structures. However, the two cases of overlap of the functions are shown in Figure A.4. Only a single wedge is depicted of the hat function in each case. Other than at vertices on the base mesh, there would be six such wedges surrounding each vertex - vertices on the base mesh must have fewer wedges in order to define structure. The values of the areas in each case can easily be precomputed, and numerical methods were used to generate a good approximation to the solution.

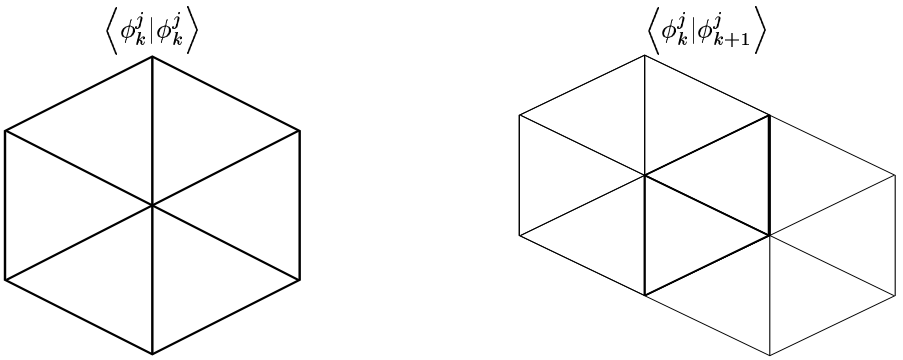


Figure A.3: The two possible overlap cases of the 1-disk wavelet inner product. The highlighted region indicates the overlap of the two functions

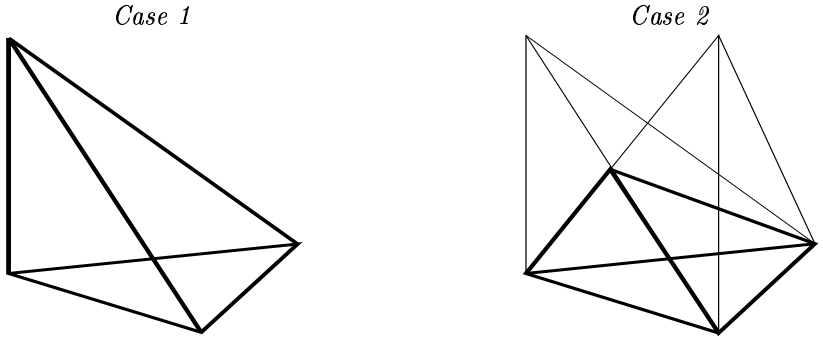


Figure A.4: Two cases of overlap for individual segments of a 1-disk wavelet. *Case 1* indicates an overlap of the same function, and hence the region of the product of the two functions is represented by the tetrahedron. *Case 2* represents a case when ϕ_k^j overlaps with ϕ_{k+1}^j. Each of these cases represents a single triangle as shown in Figure A.3.

A.3.2 Analysis / Synthesis Matrix Calculation

Initially it is necessary to generate the matrices $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{j}}$, $\mathbf{Q}_{\mathbf{lazy}}^{\mathbf{j}}$, $\mathbf{A}_{\mathbf{lazy}}^{\mathbf{j}}$ and $\mathbf{B}_{\mathbf{lazy}}^{\mathbf{j}}$ for the desired level j from the base mesh M^0. Each of these matrices is sparse, so a sparse matrix structure was used to allow for efficient storage. These matrices are defined by Equation A.8 and Equation A.7. $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{1}}$ is generated from the connectivity information of M^0, whereafter it is possible to generate the remaining $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{j}}$ using only the connectivity information stored in $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{j-1}}$.

The connectivity information stored in $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{j}}$ is then used, along with the constants generated above in section A.3.1 to generate the sparse matrices $\mathbf{I}^{\mathbf{j}}$. It is necessary to find $\mathbf{I}^{\mathbf{n+1}}$ for equation A.5, so $\mathbf{P}_{\mathbf{lazy}}^{\mathbf{n+1}}$ must be found. Once $\mathbf{I}^{\mathbf{n+1}}$ has been found the remaining $\mathbf{I}^{\mathbf{j}}$, $\mathbf{j = 0 \ldots n}$ are found using the recurrence relation in Equation A.4. The matrix $\mathbf{S}^{\mathbf{j}}$ is then determined by the formula (derived from equation A.5),

$$\mathbf{S}^{\mathbf{j}} = (\mathbf{I}^{\mathbf{j}})^{-\mathbf{1}} \mathbf{P}^{\mathbf{j}} \mathbf{N}^{\mathbf{j+1}}$$

where $\mathbf{N}^{\mathbf{j+1}}$ is the submatrix of $\mathbf{I}^{\mathbf{j+1}}$ where the columns correspond to the new vertices.

Note that at this step the inverse of the matrix $\mathbf{I}^{\mathbf{j}}$ must be found - a potentially expensive operation to perform at each level j. A numerical solution was generated using Gauss-Seidel iteration. The solution is not sparse, making it extremely expensive to store. This problem can be tackled in two ways, as is described in [Lou95], but we chose merely to truncate the coefficients

which were within a certain tolerance, restricting the support to be local.

The matrices $\mathbf{S^j}$, $\mathbf{j = 0 \ldots n}$ are then used to calculate the matrices $\mathbf{P^j_{kd}}$, $\mathbf{Q^j_{kd}}$, $\mathbf{A^j_{kd}}$ and $\mathbf{B^j_{kd}}$ by Equations A.9 and Equation A.10.

A.3.3 Multi-resolution Analysis

Given a base mesh / final mesh pair M^0, M^n and the value n, the matrices $\mathbf{P^j_{kd}}$, $\mathbf{Q^j_{kd}}$, $\mathbf{A^j_{kd}}$ and $\mathbf{B^j_{kd}}$ are calculated as above. The matrices $\mathbf{V^j}$ and $\mathbf{W^j}$ are calculated according to equations A.11 and A.12.

Once all the $\mathbf{W^j}$ have been found, the $\mathbf{V^j}$ can be discarded - $\mathbf{V^0}$ is simply the vertex information of the base mesh M^0, and the remaining levels $\mathbf{V^j}$ are determined using the synthesis Equation A.13.

The l_2 error incurred by each vertex is represented by the value of the detail coefficient associated with each vertex. By setting individual elements in the filter bank $\mathbf{W^j}$ to zero the vertex is removed from the reconstructed model. Compression is possible using simple stopping criteria:

- a global error tolerance ϵ could be defined indicating the sum of the detail coefficients which can be set to zero or
- the model could be compressed to a specified number of faces or vertices.

A.4 Implications

Wavelets for multi-resolution analysis provide a useful tool for the generation of multi-resolution models, progressive refinement and compression [SDS95] in a mathematically rigorous fashion. However, such structured solutions require equally structured input.

The calculation of matrices is a slow, mathematically intensive procedure, requiring a number of matrices to be preprocessed or processed at run-time (depending on whether time or space is to be preserved respectively). Although the computationally intensive matrix operations, such as inversion and multiplication, can be addressed in a number of ways (in this case the inversion was solved numerically, while the multiplication is a linear operation due to the implementation of a sparse matrix structure) there is a large amount of resource overhead at each level. At the very least, the matrices $\mathbf{V^j}$, $\mathbf{W^j}$, $\mathbf{P^j_{kd}}$ and $\mathbf{Q^j_{kd}}$ must be easily accessible at each level $j = 0 \ldots n$.

Subdivision connectivity (the restriction requiring all vertices to have a valence of six, except those on the base mesh) occurs very seldom in practice (except perhaps in models generated by hand). To address this issue techniques [LSS+98, EDD+95] can guarantee that models have this property. Re-meshing a surface incurs error into the new model approximation (see Chapter 2).

Of importance is the preservation of particular detail features on the surface. The effectiveness of describing a surface by a wavelet scheme could be determined by the magnitude of the wavelet coefficients in the resulting filter bank $\mathbf{W^j}$. A wavelet scheme may not efficiently describe the entire model - certain regions of the filter may have small detail coefficients, while others may have large values (a spike on a flat surface is an example). This problem could be addressed in a number of ways:

- Several wavelet schemes could be used to analyse the same model - the scheme with the smallest coefficients could be used as the best representation of the surface. Given the limited

number of wavelet construction strategies applicable to surfaces, there would unfortunately be a limited search space. This technique would probably not be feasible in practice, given the large amount of additional computation required.

- An adaptive, or non-linear wavelet scheme could be designed to segment the surface into separate regions, each of which could be best represented by different wavelets. This technique would incur new computational overhead due to the surface segmentation. We are not aware of any "adaptive" wavelet construction schemes at present.

Appendix B

Statistical Results

This appendix contains all statistical results generated from the data gathered in the experiments described in Chapter 4. All values in boldface indicate statistically relevant values at $p < .05$.

B.1 Experiment 1: Evaluation of Memoryless Simplification Metrics

K_{L^1}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	.0160414	.0102896	.0113115	.0140666	.0109296
E_{edge}		**0.031876959**	0.12749286	0.869128644	0.079249494
E_{uvol}	**0.031876959**		0.987373769	0.333604455	0.997921288
E_{pm}	0.12749286	0.987373769		0.655655026	0.999728918
$E_{quadric}$	0.869128644	0.333604455	0.655655026		0.531428695
E_{hybrid}	0.079249494	0.997921288	0.999728918	0.531428695	

Table B.1: Scheffe test of K_{L^1}.

K_{L^2}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	5.222827	3.174043	3.270291	4.298470	3.004087
E_{edge}		**0.000125428**	**0.000325529**	0.314223439	**2.06E-05**
E_{uvol}	**0.000125428**		0.999672353	0.135180801	0.996928275
E_{pm}	**0.000325529**	0.999672353		0.209154412	0.982888758
$E_{quadric}$	0.314223439	0.135180801	0.209154412		0.054494862
E_{hybrid}	**2.06E-05**	0.996928275	0.982888758	0.054494862	

Table B.2: Scheffe test of K_{L^2}.

K_{sil}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	167.4658	39.26492	37.96596	61.62542	36.83920
E_{edge}		**4.28874E-07**	**3.08148E-07**	**7.4201E-05**	**2.30699E-07**
E_{uvol}	**4.28874E-07**		0.999998271	0.89488399	0.999979258
E_{pm}	**3.08148E-07**	0.999998271		0.873620272	0.999999046
$E_{quadric}$	**7.4201E-05**	0.89488399	0.873620272		0.853416562
E_{hybrid}	**2.30699E-07**	0.999979258	0.999999046	0.853416562	

Table B.3: Scheffe test of K_{sil}.

K_{sliver}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	1.293184	1.497499	1.484808	1.512611	1.501823
E_{edge}		**4.09642E-25**	**3.06219E-22**	**1.04972E-28**	**4.00902E-26**
E_{uvol}	**4.09642E-25**		0.972815454	0.949008822	0.999575198
E_{pm}	**3.06219E-22**	0.972815454		0.657250404	0.922967434
$E_{quadric}$	**1.04972E-28**	0.949008822	0.657250404		0.985141158
E_{hybrid}	**4.00902E-26**	0.999575198	0.922967434	0.985141158	

Table B.4: Scheffe test of K_{sliver}.

K_{vol}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	.9956310	.9988992	.9985449	.9980779	.9984400
E_{edge}		**7.29815E-05**	**0.000685578**	**0.008446292**	**0.001260261**
E_{uvol}	**7.29815E-05**		0.990505159	0.81723851	0.97486788
E_{pm}	**0.000685578**	0.990505159		0.973248243	0.99992007
$E_{quadric}$	**0.008446292**	0.81723851	0.973248243		0.989677489
E_{hybrid}	**0.001260261**	0.97486788	0.99992007	0.989677489	

Table B.5: Scheffe test of K_{vol}.

K_{metro}	E_{edge}	E_{uvol}	E_{pm}	$E_{quadric}$	E_{hybrid}
Mean	.0007065	.0002565	.0001889	.0003158	.0001855
E_{edge}		**0.000977563**	**6.62617E-05**	**0.007281112**	**5.73799E-05**
E_{uvol}	**0.000977563**		0.980428398	0.988104641	0.976591587
E_{pm}	**6.62617E-05**	0.980428398		0.828010619	0.999999881
$E_{quadric}$	**0.007281112**	0.988104641	0.828010619		0.813828588
E_{hybrid}	**5.73799E-05**	0.976591587	0.999999881	0.813828588	

Table B.6: Scheffe test of K_{metro}.

B.2 Experiment 2: Optimal Placement vs Subset Placement

K_{L^1}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	.0104090	.0110861	.0109296	.0110307
0_*bit*		0.981387079	0.991356432	0.9854756
1_*bit*	0.981387079		0.999758422	0.999989212
2_*bit*	0.991356432	0.999758422		0.999934852
optimal	0.9854756	0.999989212	0.999934852	

Table B.7: Scheffe test of K_{L^1}.

K_{L^2}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	2.848524	3.035858	3.004093	3.090453
0_*bit*		0.966515839	0.98034966	0.931547105
1_*bit*	0.966515839		0.999823689	0.99910897
2_*bit*	0.98034966	0.999823689		0.996508896
optimal	0.931547105	0.99910897	0.996508896	

Table B.8: Scheffe test of K_{L^2}.

K_{sil}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	61.79460	33.30919	36.83920	29.17824
0_*bit*		0.155906081	0.259727508	0.077066347
1_*bit*	0.155906081		0.994054914	0.990557194
2_*bit*	0.259727508	0.994054914		0.944402397
optimal	0.077066347	0.990557194	0.944402397	

Table B.9: Scheffe test of K_{sil}.

K_{sliver}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	1.428884	1.520909	1.501823	1.480909
0_*bit*		**2.82727E-05**	**0.0017898**	0.05270198
1_*bit*	**2.82727E-05**		0.79107672	0.206741035
2_*bit*	**0.0017898**	0.79107672		0.740771949
optimal	0.05270198	0.206741035	0.740771949	

Table B.10: Scheffe test of K_{sliver}.

K_{vol}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	.9977720	.9985763	.9984400	.9993752
0_*bit*		0.394977659	0.560846627	**0.008318375**
1_*bit*	0.394977659		0.993490338	0.401114076
2_*bit*	0.560846627	0.993490338		0.258933127
optimal	**0.008318375**	0.401114076	0.258933127	

Table B.11: Scheffe test of K_{vol}.

B.3 Experiment 3: Image Space Measurements vs Model Space Measurements

K_{metro}	0_*bit*	1_*bit*	2_*bit*	optimal
Mean	.0003113	.0001785	.0001855	.0001290
0_*bit*		0.329014152	0.378934056	0.091423728
1_*bit*	0.329014152		0.999747455	0.923752666
2_*bit*	0.378934056	0.999747455		0.891075432
optimal	0.091423728	0.923752666	0.891075432	

Table B.12: Scheffe test of K_{metro}.

	K_{L^1}	K_{L^2}	K_{sil}	K_{sliver}	K_{vol}	K_{metro}
K_{L^1}	1	**0.923334249**	**0.737278283**	**0.419325856**	**-0.764617075**	**0.319471487**
K_{L^2}	**0.923334249**	1	**0.704498964**	**0.377429309**	**-0.647060296**	**0.333703121**
K_{sil}	**0.737278283**	**0.704498964**	1	**0.174642873**	**-0.893368893**	**0.566995367**
K_{sliver}	**0.419325856**	**0.377429309**	**0.174642873**	1	**-0.236658579**	0.156981695
K_{vol}	**-0.764617075**	**-0.647060296**	**-0.893368893**	**-0.236658579**	1	**-0.480305828**
K_{metro}	**0.319471487**	**0.333703121**	**0.566995367**	0.156981695	**-0.480305828**	1

Table B.13: Matrix of Correlations of all data (1400 data points).

	K_{L^1}	K_{L^2}	K_{sil}	K_{sliver}	K_{vol}	K_{metro}
K_{L^1}	1	**0.943714645**	**0.852700901**	**0.458925679**	**-0.459163303**	**0.3069908**
K_{L^2}	**0.943714645**	1	**0.804027355**	**0.433928295**	**-0.464069664**	**0.310586453**
K_{sil}	**0.852700901**	**0.804027355**	1	**0.308889452**	**-0.438759019**	**0.368452094**
K_{sliver}	**0.458925679**	**0.433928295**	**0.308889452**	1	-0.146136935	**0.296620384**
K_{vol}	**-0.459163303**	**-0.464069664**	**-0.438759019**	-0.146136935	1	-0.169713727
K_{metro}	**0.3069908**	**0.310586453**	**0.368452094**	**0.296620384**	-0.169713727	1

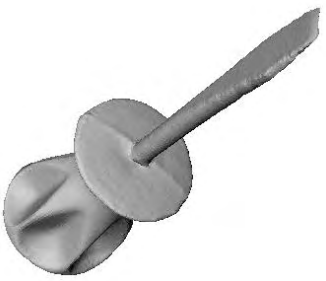
Table B.14: Matrix of Correlations of optimal placement (142 data points).

	K_{L^1}	K_{L^2}	K_{sil}	K_{sliver}	K_{vol}	K_{metro}
K_{L^1}	1	**0.948758781**	**0.826192601**	**0.594536846**	**-0.817728917**	**0.379442881**
K_{L^2}	**0.948758781**	1	**0.771174652**	**0.582368998**	**-0.698739705**	**0.373742517**
K_{sil}	**0.826192601**	**0.771174652**	1	**0.367248164**	**-0.893135404**	**0.458527565**
K_{sliver}	**0.594536846**	**0.582368998**	**0.367248164**	1	**-0.373720696**	**0.304425906**
K_{vol}	**-0.817728917**	**-0.698739705**	**-0.893135404**	**-0.373720696**	1	**-0.404602033**
K_{metro}	**0.379442881**	**0.373742517**	**0.458527565**	**0.304425906**	**-0.404602033**	1

Table B.15: Matrix of Correlations of 2-bit subset placement (142 data points).

Appendix C

Models

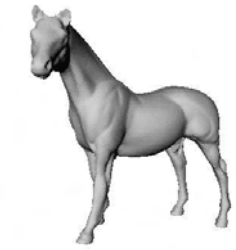


Model	Screwdriver
# Vertices	27153
# Faces	54301
Initial K_{sliver}	1.353365

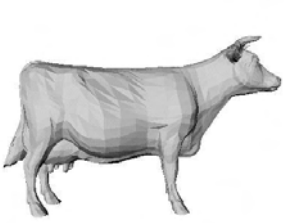
Model	Dinosaur
# Vertices	56194
# Faces	112384
Initial K_{sliver}	1.308651

Model	Club
# Vertices	52447
# Faces	104889
Initial K_{sliver}	1.445086

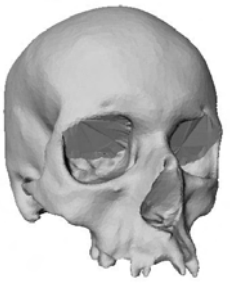
Model	Rockerarm
# Vertices	40178
# Faces	80355
Initial K_{sliver}	1.312711



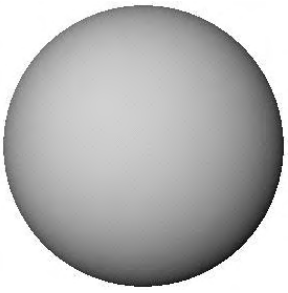
Model	Horse
# Vertices	48486
# Faces	96967
Initial K_{sliver}	1.40353



Model	Cow
# Vertices	2904
# Faces	5804
Initial K_{sliver}	1.40353



Model	Cranium
# Vertices	28981
# Faces	57966
Initial K_{sliver}	1.433173



Model	Sphere
# Vertices	25002
# Faces	50000
Initial K_{sliver}	1.052858

Model	Santa
# Vertices	75782
# Faces	151559
Initial K_{sliver}	1.247697

Model	Balljoint
# Vertices	34268
# Faces	68531
Initial K_{sliver}	1.474914

Model	Terrasque
# Vertices	22718
# Faces	45432
Initial K_{sliver}	1.72025

Model	Hip
# Vertices	25002
# Faces	50000
Initial K_{sliver}	1.496918

Model	Venus
# Vertices	50002
# Faces	100000
Initial K_{sliver}	1.397413

Appendix D

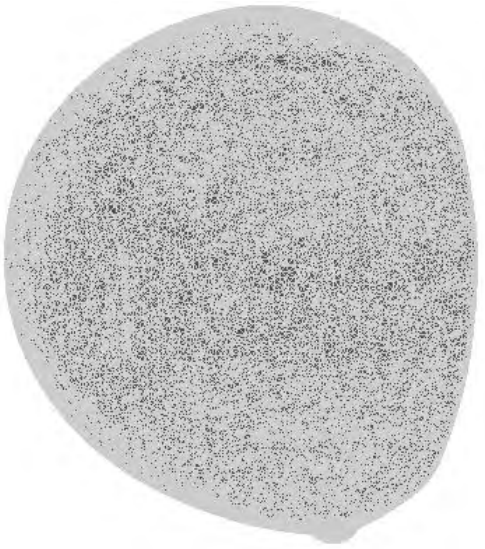
Compression Technique Comparison

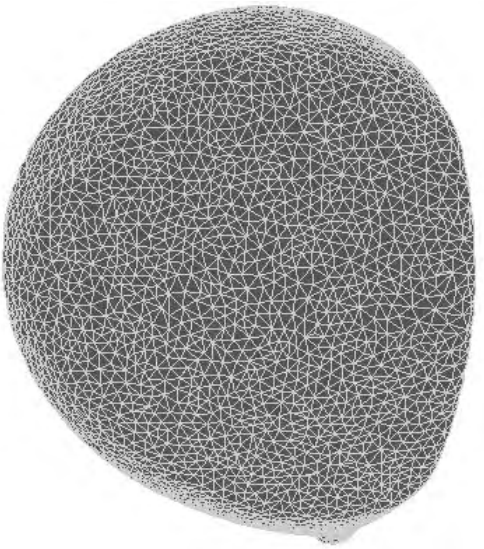
In the following pages we display visual results of the experiments detailed in Chapter 4, including both technique comparison results and placement comparison results. We show the highly detailed underside of the golf-club model in each case. All models are compressed to one tenth of the original model size (from 104889 faces to 10488 faces), and displayed using both flat shading and wireframe, to give an idea of the relative complexity of the model.

The lettering is still clearly legible using the techniques of E_{uvol}, E_{hybrid} and E_{pm}, while only partially legible using $E_{quadric}$. It is clear from the results of 0_bit, 1_bit, 2_bit and optimal placement that there is very little visual difference between the resulting model quality.

Original Model

E_{edge}





Original Model

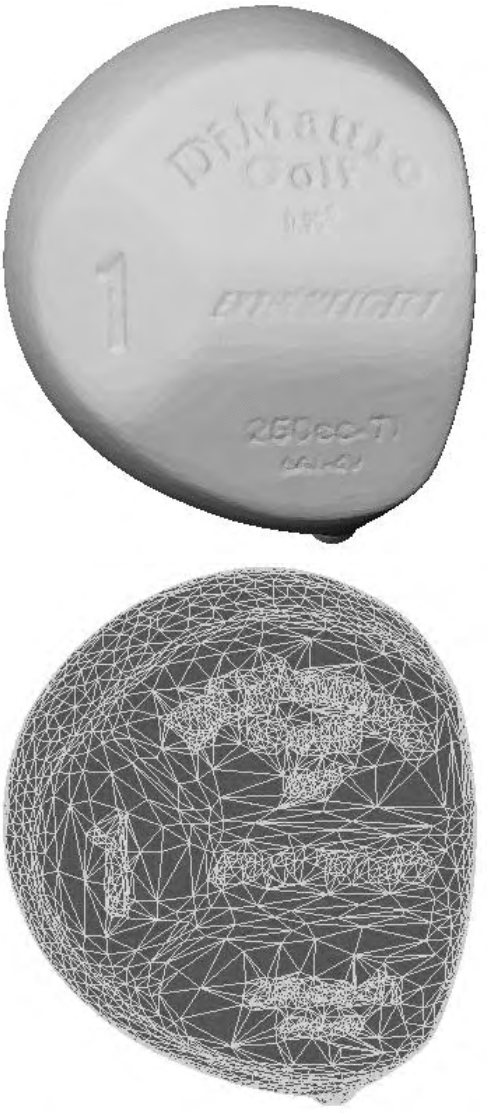
E_{uvol}

Original Model E_{pm}

Original Model

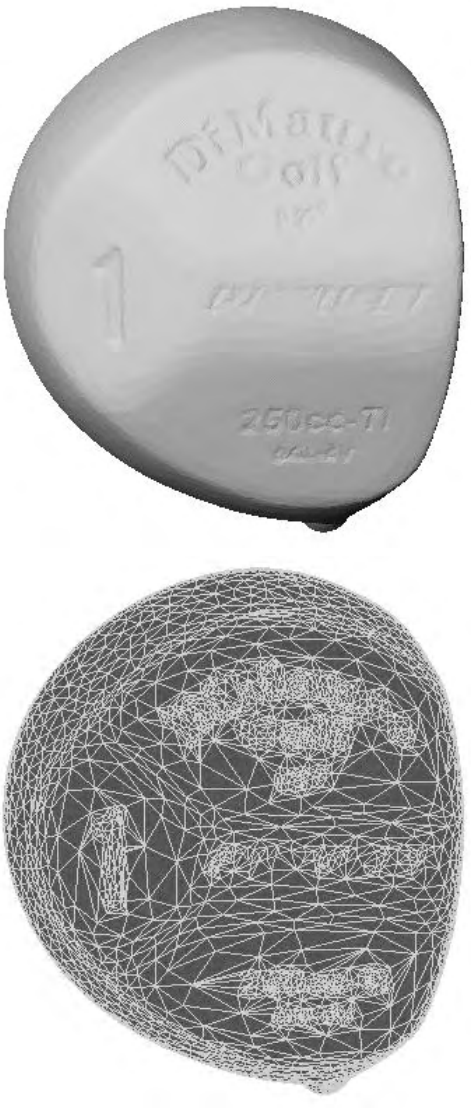
$E_{quadric}$

Original Model



E_{hybrid}

Original Model

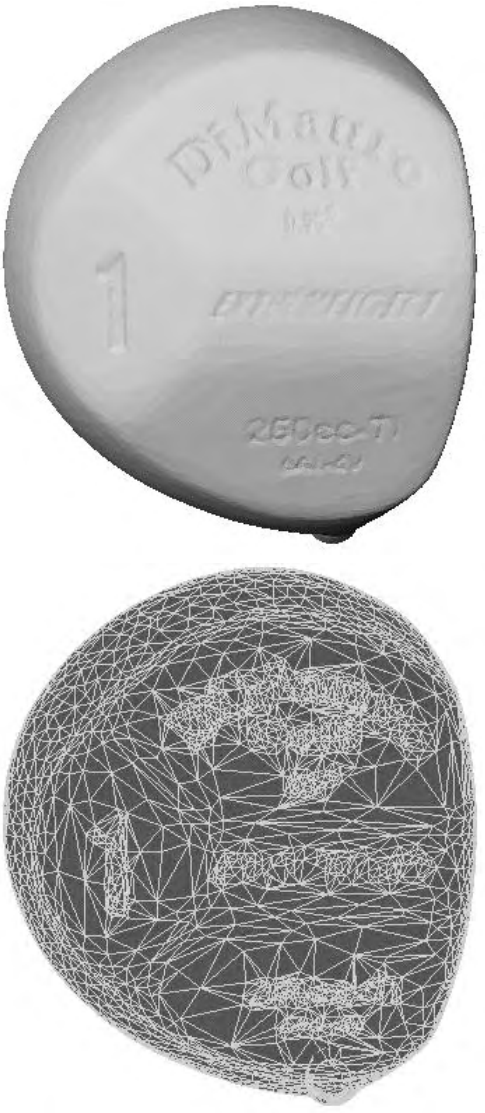


0_*bit*

Original Model

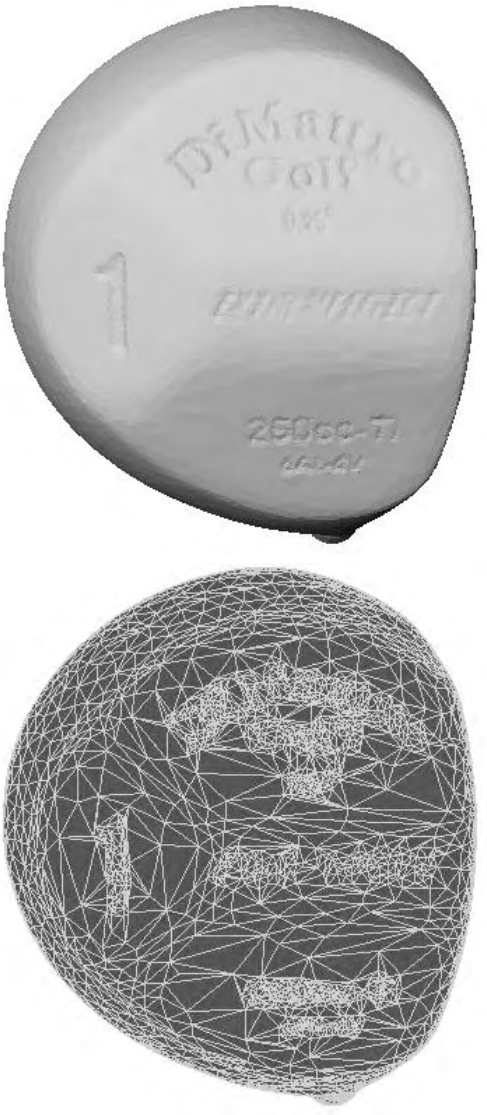
1_*bit*

Original Model



2_*bit*

Original Model



optimal

Bibliography

[Ack98] M. J. Ackerman. The visible human project. *Proceedings of the IEEE*, 86(3):504–511, March 1998.

[AD01a] Pierre Alliez and Mathieu Desbrun. Progressive compression for lossless transmission of triangle meshes. In *Proceedings of SIGGRAPH '21 (Computer Graphics)*, pages 195–202, 2001.

[AD01b] Pierre Alliez and Mathieu Desbrun. Valence-driven connectivity encoding for 3d meshes. *EUROGRAPHICS*, 20(3):37–49, 2001.

[BG98] Georges-Pierre Bonneau and Alexandre Gerussi. Level of detail visualization of scalar data sets on irregular surface meshes. In *Proceedings of IEEE Visualisation*, pages 73–78, 1998.

[Bon98] Georges-Pierre Bonneau. Multiresolution analysis on irregular surface meshes. *IEEE Transactions on Visualisation and Computer Graphics*, 4(4):365–378, 1998.

[CC78] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10(6):350–355, November 1978.

[Cla76] J. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.

[CMRS98] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A general method for preserving attribute values on simplified meshes. *In Proceedings IEEE Visualisation*, pages 59–66, 1998.

[CMS98] P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22(1):37–54, 1998.

[COL96] Daniel Cohen-Or and Yishay Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In Roni Yagel and Gregory M. Nielson, editors, *Proceedings of IEEE Visualization '96*, pages 37–42, 1996.

[COLR99] Daniel Cohen-Or, David Levin, and Offir Remez. Progressive compression of arbitrary triangular meshes. In *Proceedings of IEEE Visualisation*, pages 67–72, San Fransisco, 1999.

[CPD+96] Andrew Certain, Jovan Popovic, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive multiresolution surface viewing. In *Proceedings of SIGGRAPH '96 (Computer Graphics)*, pages 91–98, 1996.

[Dau92] Ingrid Daubechies. *Ten Lectures on Wavelets*, volume 61 of *Regional Conference Series in Applied Maths*. SIAM, Philadelphia, 1992.

[DGSS99] Ingrid Daubechies, Igor Guskov, Peter Schröder, and Wim Sweldens. Wavelets on irregular point sets. *Phil. Trans. Royal Society London A*, 357(1760):2397–2413, 1999.

[DJL92] Ronald A. DeVore, Björn Jawerth, and Bradley J. Lucier. Image compression through wavelet transform coding. *IEEE Transactions on Information Theory*, 32(2):719–746, March 1992.

[Doo78] D. W. H. Doo. *A recursive subdivision algorithm for fitting quadratic surfaces to irregular polygons*. PhD thesis, Brunel University, 1978.

[EDD+95] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH '95 (Computer Graphics)*, pages 173–182, 1995.

[EM99] Carl Erikson and Dinesh Manocha. GAPS: General and automatic polygonal simplification. In *Symposium of Interactive 3D Graphics*, pages 79–88, 1999.

[FB97] P. Frey and H. Borouchaki. Surface mesh evaluation. In *6th International Meshing Roundtable*, pages 363–374, 1997.

[FEKR90] R. L. Ferguson, R. Economy, W. A. Kelly, and P. P. Ramos. Continuous terrain level of detail for visual simulation. In *IMAGE V Conference*, pages 144–151, June 1990.

[FMP98] Leila De Floriani, Paola Magillo, and Enrico Puppo. Efficient implementation of multi-triangulations. In *Proceedings IEEE Visualisation*, pages 43–50, October 1998.

[FS93] T. A. Funkhouser and C. H. Séquin. Adaptive display algorithm for interactive frame rates during visualisation of complex virtual environments. In *Proceedings of SIGGRAPH '93 (Computer Graphics)*, volume 27, pages 247–254, 1993.

[Gar99] Michael Garland. Multiresolution modeling: Survey and future opportunities. In *Eurographics*, pages 111–131, 1999.

[GB99] Enrico Gobbetti and Eric Bouvier. Time-critical multiresolution scene rendering. In *Proceedings of IEEE Visualization*, pages 123–130, San Francisco, CA, USA, October 1999.

[GH97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of SIGGRAPH '97 (Computer Graphics)*, pages 209 – 216, 1997.

[GH98] Michael Garland and Paul S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In *IEEE Visualisation*, pages 263 – 270, 1998.

[GSS99] Igor Guskov, Wim Sweldens, and Peter Schröder. Multiresolution signal processing for meshes. In *Proceedings of SIGGRAPH '99 (Computer Graphics)*, pages 325–334, 1999.

[GTLH98] André Guéziec, Gabriel Taubin, Francis Lazarus, and William Horn. Simplicial maps for progressive transmission of polygonal surfaces. In *ACM Web 3D (VRML)*, pages 25–31, 1998.

[GVSS00] Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. Normal meshes. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 95–102, 2000.

[HDD+93] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *Proceedings of SIGGRAPH '93 (Computer Graphics)*, pages 19–26, 1993.

[HG97] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *Proceedings of SIGGRAPH '97 (Computer Graphics)*, 1997. Multiresolution Surface Modeling Course.

[Hop96] Hugues Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96 (Computer Graphics)*, pages 99–108, 1996.

[Hop97] Hugues Hoppe. View-dependant refinement of progressive meshes. In *Proceedings of SIGGRAPH '97 (Computer Graphics)*, pages 189–198, 1997.

[Hop98] Hugues Hoppe. Efficient implementation of progressive meshes. *Computers and Graphics*, 22(1):27–36, 1998.

[Hop99] Hugues Hoppe. New quadric metric for simplifying meshes with appearance attributes. In *IEEE Visualisation*, pages 59–66, 1999.

[KCS98] Leif Kobbelt, Swen Campagna, and Hans-Peter Seidel. A general framework for mesh decimation. In *Graphics Interface*, pages 43–50, 1998.

[KLS96] Reinhard Klein, Gunther Liebich, and W. Straβer. Mesh reduction with error control. In *IEEE Visualisation*, pages 311–318, 1996.

[KSS00] Andrei Khodakovsky, Peter Schróoder, and Wim Sweldens. Progressive geometry compression. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 271–278, 2000.

[LDSS99] Aaron Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *Proceedings of SIGGRAPH '99 (Computer Graphics)*, pages 343–350, 1999.

[LE97] David Luebke and Carl Erikson. View-dependant simplification of arbitrary polygon environments. In *International Conference on Computer Graphics and Interactive Techniques*, pages 199–208, 1997.

[Lin00] Peter Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 259–262, 2000.

[LKM01] Erik Lindholm, Mark J. Kligard, and Henry Moreton. A user-programmable vertex engine. In *International Conference on Computer Graphics and Interactive Techniques*, pages 149–158, 2001.

[LKR+96] P. Lindstrom, D. Koller, W. Ribarsky, L. Hodges, N. Faust, and G. Turner. Real-time continuous level of detail rendering of height fields. In *Proceedings of SIGGRAPH '96 (Computer Graphics)*, pages 109–118, 1996.

[LMWM01] Caleb Lyness, Otto-Carl Marte, Bryan Wong, and Patrick Marais. Low cost model reconstruction from image sequences. To Appear in Afrigraph, November 2001.

[Loo87] Charles T. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

[Lou95] John Michael Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, University of Washington, 1995.

[LRG+00] Marc Levoy, Szymon Rusinkiewicz, Matt Ginzton, Jeremy Ginsberg, Kari Pulli, David Koller, Sean Anderson, Jonathan Shade, Brian Cirless, Lucas Pereira, James Davis, and Duane Fulk. The digital michelangelo project: 3d scanning of large statues. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 131–144, 2000.

[LSS+98] Aaron Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *Proceedings of SIGGRAPH '98 (Computer Graphics)*, pages 95–104, 1998.

[LT98] Peter Lindstrom and Greg Turk. Fast and memory efficient polygonal simplification. In *IEEE Visualisation*, pages 279–286, 1998.

[LT00] Peter Lindstrom and Greg Turk. Image-driven simplification. *ACM Transactions on Graphics*, 19(3):204–241, July 2000.

[Mal89] Stéphane Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11:674–693, July 1989.

[Mas99] Ashton Mason. *Predictive Hierarchical Level of Detail Optimization*. PhD thesis, University of Cape Town, 1999.

[MPS+00] Alan Muller, Simon Perkins, Barry Steyn, Richard Southern, and Patrick Marais. View dependent refinement using progressive meshes. Technical Report CS00-22-00, University of Cape Town, 2000.

[PH97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *Proceedings of SIGGRAPH '97 (Computer Graphics)*, pages 59–66, 1997.

[PR00] Renato Pajarola and Jarek Rossignac. Compressed progressive meshes. *IEEE Transactions in Visualisation and Computer Graphics*, 6(1):79–93, January 2000.

[PSS01] Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parameterizations. In *Proceedings of SIGGRAPH '01 (Computer Graphics)*, pages 179–184, 2001.

[RB93] J. Rossignac and P. Borrel. Multiresolution 3D approximations for rendering complex scenes. In *Proceedings of Geometric Modeling in Computer Graphics*, pages 455–465, 1993.

[Red97] Martin Reddy. *Perceptually Modulated Level of Detail for Virtual Environments*. PhD thesis, University of Edinburgh, 1997.

[RH94] J. Rohlf and J. Helman. IRIS performer: a high performance multiprocessing toolkit for real-time 3D graphics. In *Proceedings of SIGGRAPH '94 (Computer Graphics)*, pages 381–394, 1994.

[Ros97] Jarek Rossignac. Geometric simplification and compression. In *SIGGRAPH '97 Course Notes #25*, 1997.

[Ros99] Jarek Rossignac. Edgebreaker: Compressing the incidence graph of triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January 1999.

[SAE93] L. A. Shirman and S. S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. *Computer Graphics Forum*, 12(3):261–272, 1993.

[SDS95] Eric Stollnitz, Tony Derose, and David Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann Publishers, 1995.

[SF98] Karan Singh and Eugene Fiume. Wires: A geometric deformation technique. In *Proceedings of SIGGRAPH '98 (Computer Graphics)*, pages 405–414, July 1998.

[SGG+00] Pedro Sander, Xianfeng Gu, Steven Gortler, Hugues Hoppe, and John Snyder. Silhouette clipping. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 327–334, 2000.

[SS95] Peter Schröder and Wim Sweldens. Spherical wavelets: Efficiently representing functions on the sphere. In *Proceedings of SIGGRAPH '95 (Computer Graphics)*, pages 59–66, 1995.

[SS00] Mel Slater and Anthony Steed. A virtual presence counter. *Presence*, Volume 9(Issue 5):413–434, October 2000.

[SSGH01] Pedro V. Sander, John Snyder, Steven J. Gortler, and Hugues Hoppe. Texture mapped progressive meshes. In *Proceedings of SIGGRAPH '01 (Computer Graphics)*, pages 409–416, 2001.

[SZL92] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen. Decimation of triangle meshes. In *Proceedings of SIGGRAPH '92 (Computer Graphics)*, pages 65–70, 1992.

[TB94] David C. Taylor and William A. Barret. An algorithm for continuous resolution polygonizations of a discrete surface. In *Proceedings of Graphics Interface*, pages 33–42, 1994.

[TLG99] Danny To, Rynson Lau, and Mark Green. A method for progressive and selective transmission of multi-resolution models. In *ACM Virtual Reality Software and Technology*, pages 88–95, 1999.

[TR98] G. Taubin and J. Rossignac. Geometric compression through topological surgery. *ACM Transactions on Graphics*, 17(2):84–115, April 1998.

[Tur92] Greg Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH '92 (Computer Graphics)*, volume 26, pages 55–64, 1992.

[Ups90] Steve Upstill. *The Renderman Companion.* Addison Wesley Publishers, 1990.

[WDSB00] Zoë Wood, Mathieu Desbrun, Peter Schröder, and David Breen. Semi-regular mesh extraction from volumes. In *Proceedings of SIGGRAPH '00 (Computer Graphics)*, pages 275–282, 2000.

[Web78] E.H. Weber. *De Tactu (The Sense of Touch).* Published for the Experimental Psychology Society. Academic Press, London, 1978. D.J. Murray (Translator).

[WFM01] B.A. Watson, A. Friedman, and A. McGaffey. Measuring and predicting visual fidelity. In *Proceedings SIGGRAPH '01 (Computer Graphics)*, pages 213–220, August 2001.

[XESV97] Julie C. Xia, Jihad El-Sana, and Amitabh Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):171–183, April 1997.

[ZSS99] Dennis Zorin, Peter Schröder, and Wim Sweldens. Interactive multiresolution mesh editing. In *Proceedings of SIGGRAPH '99 (Computer Graphics)*, pages 259–268, 1999.