Volume 22 (2003), number 1 pp. 35–48 COMPUTER GRAPHICS *forum*

Creation and Control of Real-time Continuous Level of Detail on Programmable Graphics Hardware

Richard Southern and James Gain

Collaborative Visual Computing Laboratory, University of Cape Town, Cape Town, South Africa

Abstract

Continuity in level of detail sequences is essential in hiding visual artefacts that occur when switching between discrete levels of detail. However, construction and implementation of these sequences is prohibitively complex. We present a new structure, the g-mesh, which greatly simplifies the implementation of continuous level of detail in large scenes. We also introduce a novel greedy predictive level of detail control system suited to the g-mesh. Finally we achieve a dramatic improvement in the rendering of morphing sequences by exploiting current graphics hardware.

ACM CSS: I.3.5 Computational Geometry and Object Modeling—*Geometric Transformations, Object Hierarchies*, I.3.6 Methodology and Techniques—*Graphics Data Structures*

1. Introduction

Level of detail blending through geometric morphing (or *geomorphing* [1]) is effective at hiding visual discontinuities in level of detail (LOD) sequences (this artefact is often called "popping"). Geomorphing involves an interpolated transition between LOD models, designed to ensure that model transitions are unnoticeable. Although widely employed in terrain visualization, geomorphing techniques are typically difficult to apply to true three-dimensional (3D) objects. Most applications are forced to use models with low polygon counts or discrete levels of detail, in order to optimize rendering performance. There are several reasons for this:

- *Performance:* The number of vertex interpolations per frame of a geomorph sequence, if performed in software, directly impacts on rendering performance. 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 when performing interpolations between LOD models.
- *Animation:* The combination of geomorphing and animation techniques, such as matrix palette skinning (sometimes referred to as *bones* [2]) or *wires* [3] has yet to be explored. Our interpolation technique can be combined with the deformation techniques of *wires* or *bones* to enable time-dependent animation that is independent of switches in LOD.
- *Continuous LOD Control:* Continuous LOD presents unique challenges when attempting to maintain a constant frame-rate. Switching between levels of detail 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 geomorphs described by Hoppe [1] based on the progressive mesh are difficult to construct. The relative independence of each vertex split operation in a progressive mesh must be tested. 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.

In this paper we present a novel mesh representation, the geomorph (or *g*-) mesh, which greatly simplifies the gen-

eration of continuous levels of detail for arbitrary meshes. We outline the construction and application of the g-mesh, and describe how its design allows for considerable acceleration on current graphics hardware. As the g-mesh requires only one parameter to control the current continuous level of detail, an overall LOD control system is easy to define. Using current hardware it is easy to incorporate animation techniques based on vertex transformations, such as bones [2] and wires [3].

It is important to consider the limitations of this work:

- *View Independence:* The method described is only defined for discrete LOD representations, not a view-dependent vertex hierarchy. This limits the application of this technique to dynamic models or objects in a virtual environment.
- *Untextured Surfaces:* While the technique has been applied to textured models, the visual results can be displeasing. Texture tends to "slide" across the surface due to the simultaneous interpolation of multiple texture coordinates. This problem is generally applicable to all schemes employing vertex interpolation techniques.
- *Memory Overhead:* As with most LOD sequences, the memory overhead is greater than storing just the original surface. The memory overhead of the g-mesh (as described in Section 4) also suffers from this limitation. This issue is further discussed in Appendix 7.

2. Related Work

Surface blending can be defined as any technique that attempts to reduce the visual effect of transitions between discrete multi-resolution models. Surface blending techniques can be divided into two categories: smooth terrain transitions [4–7] and more general model-based blending methods [1,8–11].

Funkhouser and Séquin [9] describe a technique that blends between two independent LOD representations using alpha blending to "phase out" the old model and "phase in" the new. However, Ferguson *et al.* [7] claim that such blending techniques are visually distracting. Graphics toolkits such as IRIS Performer [12] and Renderman [13] provide facilities for surface blending techniques by making use of hardware alpha blending and interpolation between level of detail models.

Model blending techniques with terrain models employ heuristics based on their particular surface connectivity. Terrain models are typically represented by regularly spaced height-fields. Transforming these height-fields into a triangle mesh results in a surface where every vertex (except those at the edges) has valence six (the *valence* of a vertex is defined as the number of faces which include that vertex).

Taylor and Barret [5] exploit the inherent 2D 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 a Delaunay triangulated terrain, Cohen-Or and Levanoni [6] introduce a hierarchical Delaunay representation that allows for smooth transitions between different triangulations. Their technique bears a resemblance to the method of Hoppe [1], in that edges are collapsed iteratively in the re-triangulated region.

Lindstrom *et al.* [4] use the regular nature of terrain meshes to construct a subdivision hierarchy. Simplification is achieved by triangle fusing, where two adjacent triangles are merged by removing a shared vertex. Transitions between different levels of the hierarchy can be smoothly performed. 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 vertex in the mesh. In this case the parameter t refers to a scalar 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 exhibits C_0 *positional* continuity.
(2) *Block Continuity:* Neighboring cells of the terrain model must align so that the mesh does not have T-vertices and potential gaps. This differs from geometric continuity in that a terrain model may consist of unconnected patches whose points may be duplicated on the surface.
(3) *Rendering Continuity:* The number of polygons sent 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].
(4) *Polygon Density Continuity:* The number of polygons used to describe an area with respect to the view parameters is continuous, i.e. the distribution of polygons over an area is continuous according to the view point.

Hoppe [1] recommends the progressive mesh as a multi-resolution hierarchy constructed using the edge collapse ($ecol_i, i = 1 \ldots n$) and the vertex split ($vsplit_i, i = 1 \ldots n$) operations. Here, n defines the number of levels of detail generated with the technique. These operations allow a sequence of multi-resolution models $M^i, i = 0 \ldots n$:

$$M^0 \xleftarrow{ecol_1} M^1 \xleftarrow{ecol_2} \ldots \xleftarrow{ecol_{n-1}} M^{n-1} \xleftarrow{ecol_n} M^n$$

$$M^0 \xrightarrow{vsplit_1} M^1 \xrightarrow{vsplit_2} \ldots \xrightarrow{vsplit_{n-1}} M^{n-1} \xrightarrow{vsplit_n} M^n$$

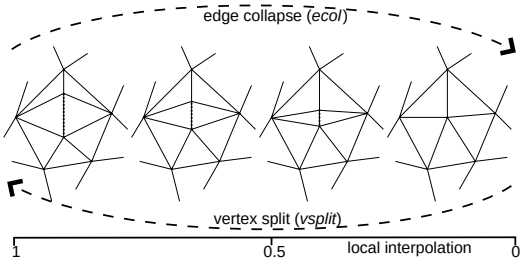


Figure 1: *Interpolation of a vertex split / edge collapse operation. The local interpolation parameter is equivalent to the function $g_i(T)$ defined in Section 4.2.*

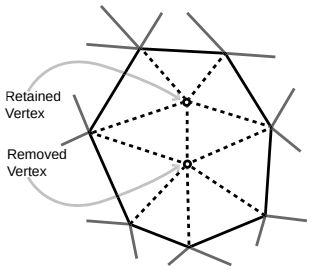


Figure 2: *Batched hierarchy construction. After an edge collapse operation, the dashed edges are removed from the priority queue.*

Hoppe addresses the problem of smooth model transitions between levels of detail in a progressive mesh structure. The progressive mesh is structured as a base mesh and a sequence of refinements necessary to reconstruct the original model. Progressive mesh refinements are implemented as vertex splits, which can be animated by interpolating the new vertex positions over time (see Figure 1). Hoppe defines a *geomorph* as a geometric transition between two level of detail representations, such that multiple independent geomorphs can be performed simultaneously.

In order to establish which vertex splits can be performed in parallel, the list of available vertex splits must be traversed to determine their validity. A vertex split is valid if the vertices and faces on which it depends are present [1,14]. Southern *et al.* [14] avoid this step by dividing vertex split operations into batches during simplification. The vertex split operations within a batch can all be performed simultaneously.

Reddy [15] presents an overview of factors upon which an 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 [16] propose a general LOD control mechanism for determining scene representations in time-critical systems. Their method, based on quadratic optimization and the visual error measures of Reddy [15], 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 framework, they do not discuss the overhead of computing smooth geometric interpolations between frames of their scene. We show in Section 6 that for large scenes this computational overhead can prove prohibitively costly.

Grabner [17] describes an interactive view-dependent framework. He defines a local interpolation parameter $\alpha_s \in [0,1]$ (analogous to the local interpolation parameter t presented in Section 4.2). The number of frames taken to perform a geomorph operation in this technique is dependent on screen space error, rather than time.

Zach [18] presents a predictive system for level-of-detail management. The parameter τ, which defines the number of frames which can be used for geomorphing, can be adjusted to deliver more control over visual and frame-rate discontinuities, in a similar manner to the T_{change} parameter (defined in Section 5.4).

However, both these techniques [17,18] differ fundamentally from the technique presented here, in that the g-mesh only performs interpolations as a function of the viewers position – it is not dependent on a fixed number of frames or time.

Batched Operations

Southern *et al.* [14] modify the progressive mesh representation to form a *batched* hierarchy of operations. This method forms the basis of an automatic LOD partitioning, where each consecutive model in the LOD sequence consists of roughly half the number of vertices in the preceding model, depending on the topology of the surface.

Progressive simplification algorithms make use of a priority queue of potential edge collapse operations, sorted according to a measurement of their impact on the current mesh. Typically after a successful operation, all affected operations in the queue are updated. In constructing a batched hierarchy, edge collapse operations involving the edges originating from the retained vertex (indicated by dashed edges in Figure 2) are removed from this queue. Once empty, the queue is rebuilt from the current mesh, and the process repeats. This method forms the basis of a batching hierarchy, where

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

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

Although any *vsplit* operation 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, and the largest batch is applied to reach the final mesh M^n. The independence of *vsplit* operations in each batch increases the set of possible mesh configurations, as compared with the standard linear hierarchy. The mesh representations $M^0, M^p, \ldots, M^n$ are equivalent to the attribute and connectivity sets defined in Section 4.

3. Continuous Level of Detail

In this paper we present a technique for continuous level of detail control of surfaces with arbitrary topology. Lindstrom *et al.* [4] define four types of continuity appropriate to continuous LOD terrain techniques (described in Section 2). Items (3) and (4) depend on viewing parameters, defining distribution and polygon density within the view frustum. These concerns do not apply to our technique, which does not refine models based on the view frustum. We believe 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 disguise changes in 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 as a change in frame-rate. Note that we only address visual discontinuities caused by geometric artefacts, and not those caused by texture.
- *Geometric Continuity:* Block continuity [4] is required to prevent cracks in terrain models. However, for an arbitrary model with unconstrained topology, there are no strict boundaries available to construct these blocks. Any definition of geometric continuity needs to encompass these concerns. We define geometric continuity in terms of the insertion and removal of vertices in the mesh: *a vertex can only be inserted or removed from the mesh at the position of a vertex that is already present.* If we regard removed vertices as present in the mesh (sharing their 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 [19] state that any disruption can cause a break in the feeling of presence. Such disruptions can take the form of popping between LOD models (*visual discontinuities*), or a sudden decrease in frame-rate, resulting in reduced interactivity or *lag* [15]. Frame-rate continuity is similar to rendering and polygon density continuity (described in Section 2), since it is achieved by constraining the number of polygons in the scene.

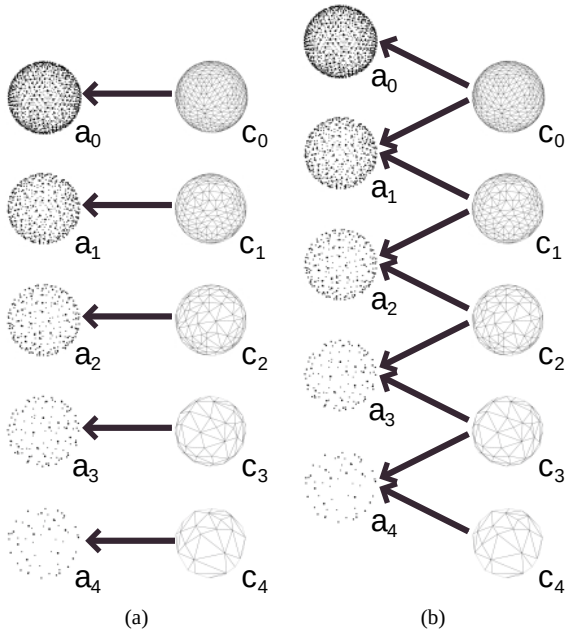


Figure 3: *The g-mesh structure. A standard discrete LOD model sequence is shown in (a), while in (b) we present the g-mesh structure. A g-mesh is a hierarchy of attributes (a_i) (such as vertex positions and color) and connectivity (c_i) (the manner in which the attributes are connected). In (a) each 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 4.2 as $g_i(T)$) determines how these two attribute sets are combined.*

Frame-rate and *visual continuity* may be irreconcilable (for instance, 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 (as with Mason [20]), but this cannot constrain the *rate* at which the models change. If transitions are performed too quickly, popping may result.

We define a *volatile* scene as one in which situations arise where frame-rate continuity and visual continuity cannot be maintained. A region of volatility in such a scene is a sequence of frames that exhibit volatility. Figure 13(b), used in our experimental evaluation, has a region of volatility surrounding frames where the camera points down at a single dolphin, after which many dolphins are introduced into the view. In Section 5 we discuss the issue of volatility and show how the rate of transitions can be controlled to reduce popping.

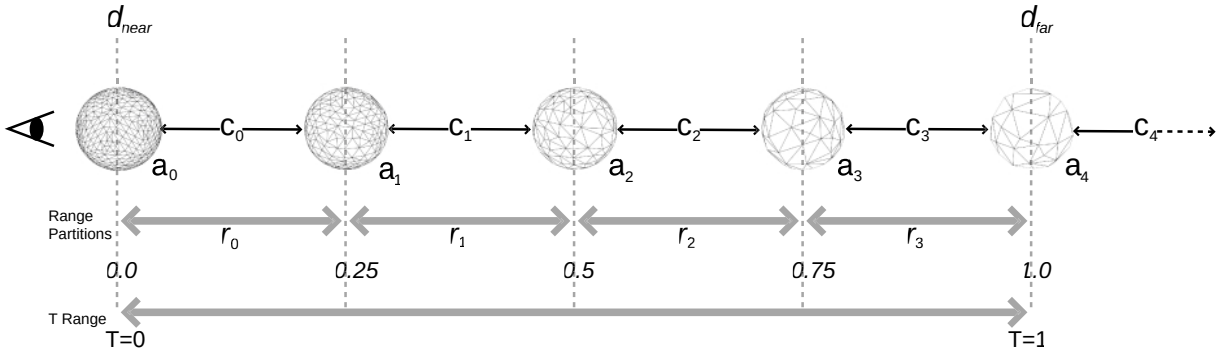


Figure 4: *An even LOD partition. This demonstrates a LOD control mechanism, where the parameter T determines a model representation defined by a connectivity set c_i. The function $g_i(T)$ maps the global interpolation parameter into a local interpolation parameter (see Section 4.2), which is used to interpolate 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 partitioning $r_i, i = 0, \ldots, 3$ are equal in size. For $T > 1$ the least complex model representation is used, made up of c_n and a_n.*

4. Defining the g-mesh

We introduce the geomorph mesh (or g-mesh) as a structure that 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 that share connectivity sets. This allows smooth transitions between attribute sets, and hence between LOD models. In this paper we use a superscript to specify individual components of a set, while a subscript denotes the set number.

A g-mesh is a hierarchical level of detail sequence, consisting of $n+1$ attribute information arrays $a_i; i = 0, \ldots, n$, and $n+1$ connectivity information sets $c_i; i = 0, \ldots, n$, each defining the shared connectivity across 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 3).

Connectivity set c_0 has the same number of polygons as the full resolution model, while connectivity set c_n is the final, or simplest level of detail, and only refers to attribute array a_n. The g-mesh is defined as

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

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

4.1. LOD Partitioning

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

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

T is used as a global LOD control parameter. $T > 1$ implies that the object has passed the maximum distance for LOD transition, and the lowest level of detail is visible. A value of $T < 0$ means that the object is at maximum resolution, or behind the viewer and so must be culled.

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

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

Figure 4 demonstrates an even partitioning 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 interpolation range.

4.2. Model Interpolation

We have shown how the active connectivity set, c_i, is determined by the LOD control parameter T. We now need a function which determines the current g-mesh attributes, such that geometric continuity is preserved. For this purpose, a basic linear interpolation function for each component of the mesh is sufficient.

We define a function $F^j(T)$ which determines the current position of attribute a^j. If R_i is the T value at which the

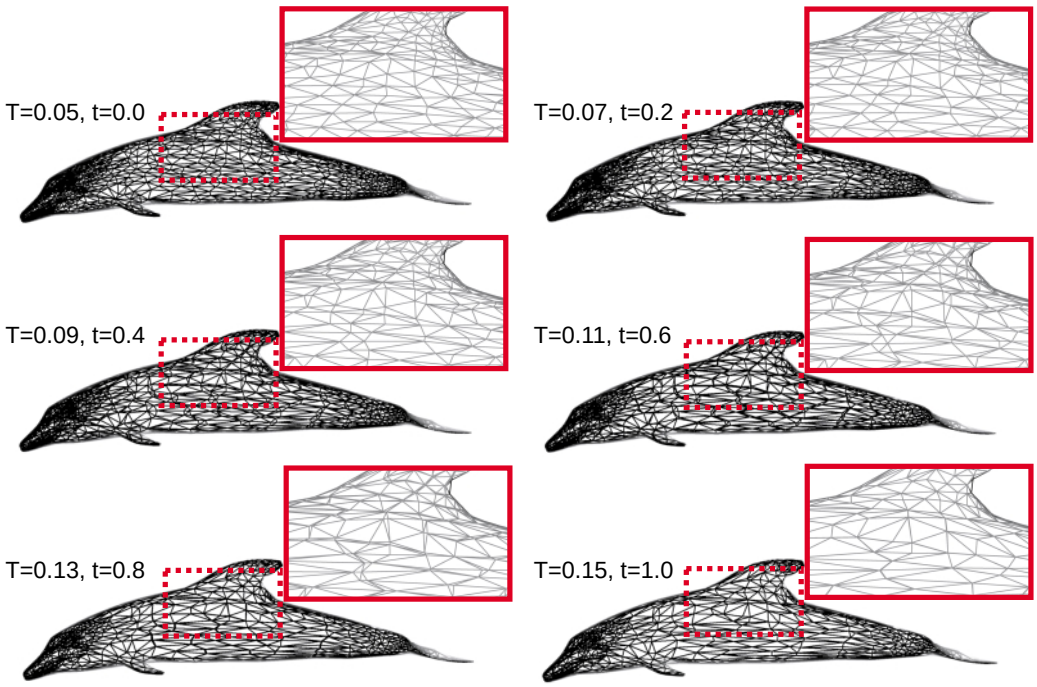


Figure 5: *An interpolation sequence. The dolphin is interpolated through its second level of detail partition, in this case between the range of* $T = [0.05, 0.15]$. *Note that the positions of the vertices at the end of each sequence are the starting positions for the next sequence, implying geometric continuity.*

range r_i begins, then

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

Next the function $g_i(T)$ is a mapping function between the global transition parameter T and the local transition parameter for connectivity set i:

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

If T is not within the range $\left[R_i, R_{(i+1)}\right]$ the value for $g_i(T)$ will be outside of the range [0, 1]. Now the value f_i^j of a particular attribute j within an attribute set i is defined as

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

The sum of the functions

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

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

According to this notation, interpolations will take place even if the connectivity information for a particular LOD c_i does not require that attribute. This would result 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* with indexed primitives (described in Section 6). In Figure 5 a single g-mesh interpolation sequence is shown.

4.3. g-mesh Construction

The g-mesh is constructed during the surface simplification process, using the batched hierarchy defined in Southern *et al.* [14]. The mesh structure is initialized 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 and connectivity set is written to the mesh. Once a particular termination criterion has been satisfied, the final attribute and connectivity sets are written to the g-mesh, along with the number of levels of detail, n.

Although only a single attribute set and connectivity set pair are theoretically required at each level of resolution (see Figure 3), implementation constraints require the indexing

of the attribute sets at each level of resolution to correspond with the associated connectivity set. A number of potential memory configurations of the *g*-mesh structure are discussed in Appendix 7.

5. Continuous LOD Control

A continuous LOD algorithm presents a unique challenge to frame-rate control. Funkhouser and Séquin [9] state that only by prediction can a constant frame-rate be maintained. A predictive scheme attempts to maintain the desired frame-rate by computing the overall scene complexity before rendering and then adjusting individual model complexities accordingly. In the case of continuous LOD sequences an unguided predictive scheme will result in "popping" whenever there is a sudden and sufficient change in the complexity of the scene (for instance, when a new object is inserted). In these cases (called *regions of volatility*) frame-rate continuity and visual continuity are mutually exclusive.

We put forward a greedy global predictive algorithm for continuous LOD control which depends on a single global parameter T. This technique does not depend on the *g*-mesh structure, and could be used with any view independent LOD model sequence or structure. The technique presented is particularly relevant to the *g*-mesh as the derivation of the local interpolation parameter t (defined in Section 4.2) from the global interpolation parameter T is trivial.

The interpolation parameter T provides effective global control over the LOD of all models in a scene. In scenes containing several objects, T can be offset to increase or reduce the complexity of all the models in the scene. We call this offset bias T_{bias} (as explained in Figure 6). The change to T_{bias} in each frame is called T_{change}.

5.1. Determining T_{bias}

In order to maintain a given frame-rate (or, in this case, polygon count), with the least possible correction, we develop a novel global LOD control algorithm that determines the optimal value for the parameter T_{bias}. We achieve this by sorting the models in the scene on the distance from their nearest LOD partition, and incrementing (or decrementing) the T_{change} until the correction is sufficient to balance the number of polygons in the scene. Thus the goal is to restrict the number of polygons in the scene to a desired value p, with a permitted tolerance of ϵ.

Before rendering each frame, the number of polygons in the scene q, and required polygon correction $k = p - q$ are determined. Each object in the scene is assigned 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 or next connectivity levels, respectively. Similarly, P_{prev} and P_{next} are the number of polygons that would be removed from or added to the scene in changing to the previous or next connectivity level.

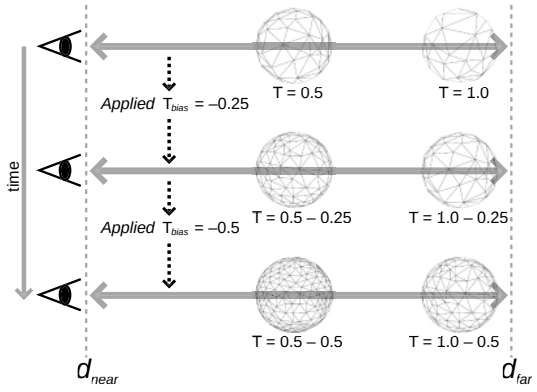


Figure 6: *Applying a* T_{bias}. *By gradually incrementing the value of the* T_{bias} *visual discontinuities caused by changing the parameter* T *can be reduced. In this example the frame-rate is stabilized by improving the overall quality of the scene. This is achieved by subtracting a value of* $T_{\text{change}} = 0.25$ *from the* T_{bias} *in each frame, gradually enhancing the overall detail.*

The algorithm for global level of detail control is 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 changed (i.e. the required polygon count 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) $T_{bias} = T_{bias} + T_{change}$

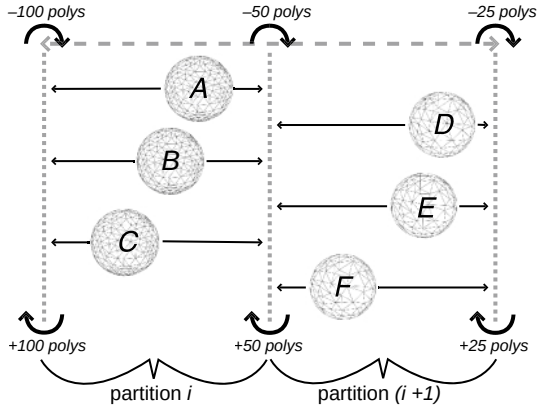


Figure 7: *A continuous level of detail control system. Six objects are scattered across two consecutive LOD partitions. In this example, the number of polygons in the scene needs to be reduced by* 45*. Firstly LOD control structures are constructed for each object, and then sorted in increasing order 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 causes an accumulated change of* 75 *(*$25 + 50$*), which is sufficient to balance the number of polygons in the scene. The adjustment to* T_{bias} *is the* T_{change} *necessary move the object A across the LOD partition.*

The execution of this algorithm is demonstrated in Figure 7. Note that in Figure 7, the correction overshoots the desired value by 30 polygons. If there is no change in the following frame it would be undesirable for the scheme to "counter-correct" so few polygons, since this would result in unnecessary oscillations. For this reason we permit a region of tolerance, ϵ about the desired polygon count. 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 $\epsilon = (|c_0| - |c_1|)$.

5.2. Maintaining Visual Continuity

Although the predictive LOD correction scheme described above ensures frame-rate continuity, it cannot guarantee visual continuity. However, popping caused by model interpolation between frames can be reduced by restricting T_{change}, the variation of T_{bias}, to ensure that visual differences are minimized. However, surfaces do not degrade linearly as they geomorph to lower resolution versions, since interpolated edges become longer as model resolution decreases.

In order to control T_{change} based on the image quality of the model, it would be necessary to either adjust it dynamically, according to the LOD partition in which the object lay, or shift the LOD partitions so that T_{change} remained 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. Instead we adjust the individual partitions of each object.

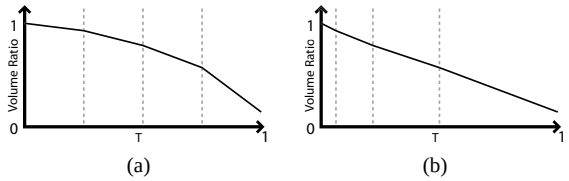


Figure 8: *Shifting the LOD partition.* (*a*) *As the surface is simplified, the volume of the enclosed object shrinks exponentially.* (*b*) *The LOD partitions (depicted by vertical dashed lines) are adjusted to ensure that the volume shrinkage is close to linear in* T*.*

5.3. LOD Partitioning based on Image Quality

It has been shown [21] that decay in the volume of a simplified model:

- correlates significantly (-0.75) with decrease in the measured image quality, based on the L_1 norm, and
- correlates significantly (-0.89) with decline in the quality of the objects silhouette.

Consequently, we use the decline in the volume of the model to construct LOD partitions which approximate image quality.

During simplificiation the volume of the object is measured after each batch has been completed. This volume is divided by the original volume of the object to yield a normalized volume measurement $V_i; i = 0 \ldots n$, for each batch ($V_0 = 1$, as it represents the volume of the unsimplified model). These volume measurements are used to determine $r_i = V_i - V_{i+1}, i = 0 \ldots (n-1)$. The effect of shifting the partitions according to volume is shown in Figure 8, and an example of a shifted LOD sequence based on volume degradation can be found in Figure 9.

Generally, this form of partitioning causes an increase in the extent of partitions as detail decreases, 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 when the model is at 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 certain objects in the scene with more importance (and more detail) than others, and can effectively guide the focus of the viewer.

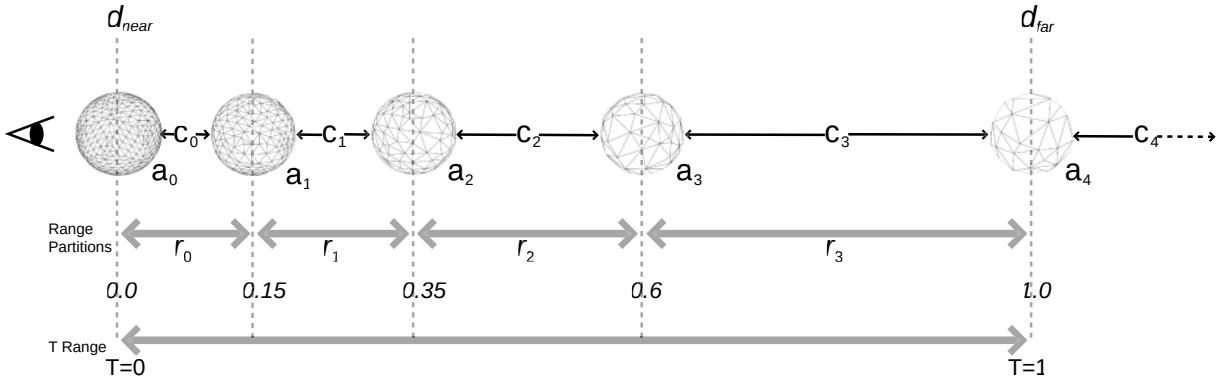


Figure 9: *A shifted LOD partition based on approximated image degradation. The definitions for $r_i, i = 0 \ldots 3$; a_i and $c_i, i = 0 \ldots 4$, and T are the same as in Figure 4. The partitions r_i are built by using volume degradation to approximate image degradation.*

5.4. Deciding on a minimum T_{change}

Several authors [9,15] have noticed that viewers are more tolerant of lower detail and visual discontinuities if the model is moving at a high velocity, or the viewers focus is shifted elsewhere. In these situations, it is permissible to use a large (or unconstrained) T_{change} between frames.

Fixing the value of T_{change} can have side effects for scenes with a great deal of *volatility*, since the LOD control system described in Section 5 cannot keep up with the changes taking place. This kind of volatility often occurs when a mass of occluded geometry is exposed (e.g. through a portal). When moving through a scene, we find *visual discontinuities* to be unnoticable but *frame-rate discontinuities* distracting.

To ensure a controlled frame-rate, T_{change} remains unconstrained when large amounts of new geometry could be exposed between frames (for example, while the user is moving through the scene). An unconstrained frame-rate guarentees *frame-rate continuity*. While the viewer is stationary, T_{change} is constrained to minimize visual discontinuities.

Our experiments evaluate 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 (see Figure 10). A higher maximum value for T_{change} permits more drastic visual changes in stabilizing the frame-rate. T_{change} could be more accurately determined on a per model basis using visual experiments such as the *just noticeable difference* test [22]. Such a visual measurement would more accurately measure a perceived deviation in model quality, but this is an area for future research.

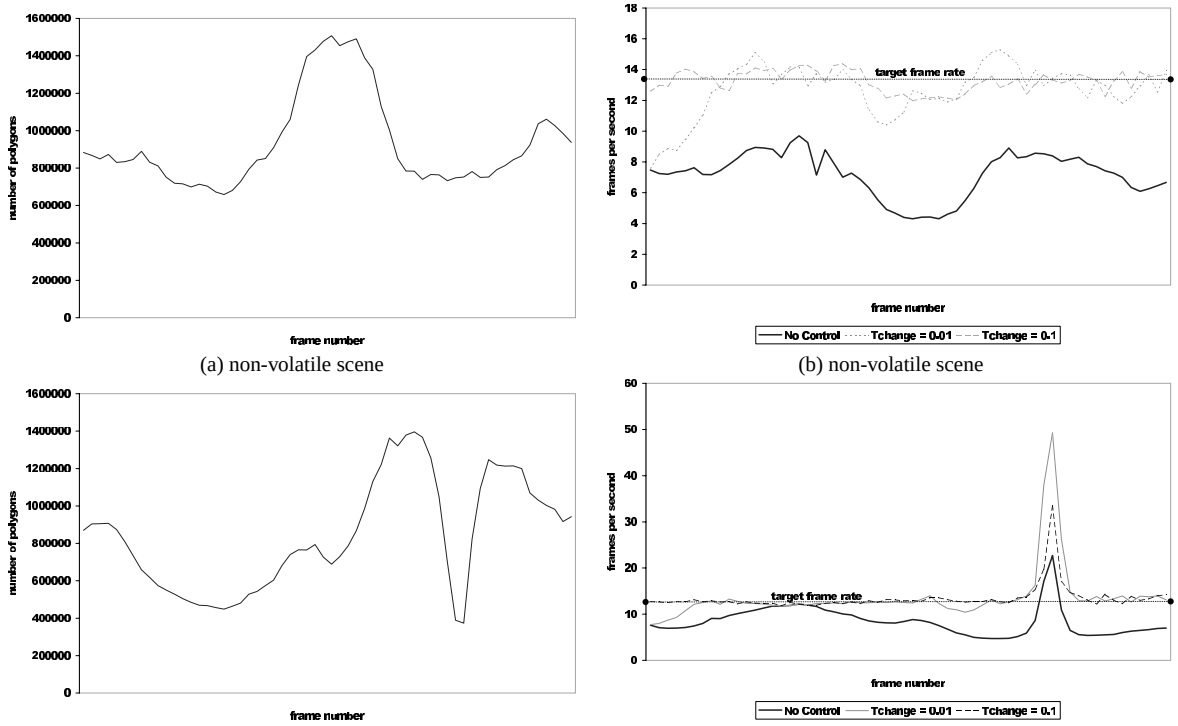
6. Implementation

In order to construct the g-mesh file we make use of the Generic Memoryless Polygonal Framework and batched hierarchy described in Southern *et al.* [14]. Each attribute set a_i and connectivity set c_i is written to the g-mesh file at the start and after the completion of each batch. This process is run until some stopping criteria is 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} [14] 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 necessary 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* [23] extension available for the NVidia GeForce 3. A graphic overview of our vertex program appears in Figure 11. The SIMD architecture of vertex programmable hardware permits parallel computation of vertex interpolations as part of the rendering pipeline. A code segment for the linear interpolation function is included in Figure 12.

Each attribute set was stored in a *vertex array*, and drawn with indexed primitives. The *vertex array range* extension was also used to accelerate memory access of these attribute sets. The speed up from using this extension was in some circumstances as much as threefold.

A simplified version of Wires deformation [3] is used to perform animation. The constant registers store the Wire parameters and an interpolation parameter (all animation constants are referred to in Figure 11 as *A*). This allows the dolphins' tails to flap in the experimental scene.



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

(c) volatile scene (d) volatile scene

Figure 10: *Controlling the number of polygons in the scene. Each graph depicts the measured number of polygons or frames per second over 120 frames along a predefined path. (a) and (c) show the polygon counts for uncontrolled non-volatile and volatile scenes respectively. In (c) the number of polygons in the scene drops suddenly at the point where the camera faces down (as in Figure 13 (b)). In (b) $T_{\text{change}} = 0.01$ is capable of maintaining the frame-rate close to the desired level. It is clear that in (d) a $T_{\text{change}} = 0.1$ 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 introducing a forced delay.*

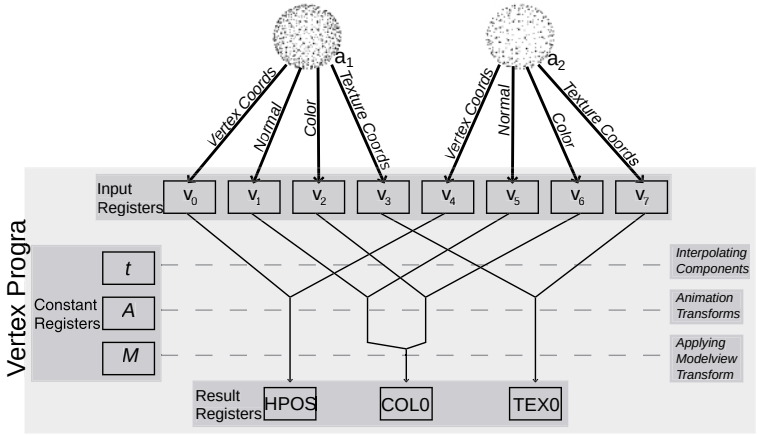
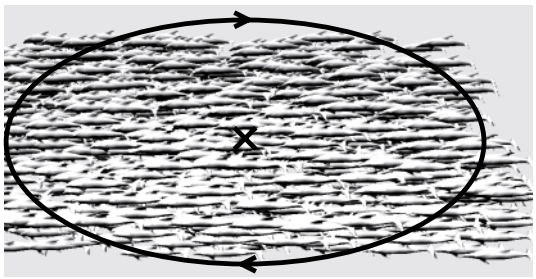


Figure 11: *Our Vertex Program. The two attribute sets a_1 and a_2 are passed through input registers $v_i, i = 0 \ldots 7$. The first stage combines these attributes with the t parameter, stored in a constant register. After the points in the mesh are determined, an animation routine can be applied (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 (color) and TEX0 (texture coordinates).*

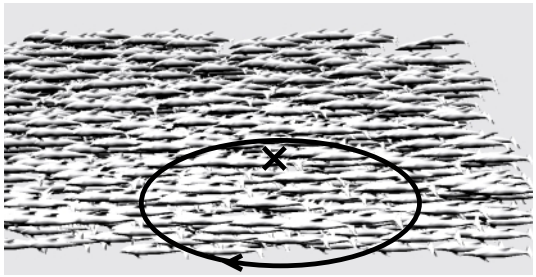
```
char VPGeomorph [] =
"!!VP1.0\n"
// attributes for attribute sets i, i+1:
// v0,v4 position
// v1,v5 normal
// v2,v6 color
// v3,v7 texture coordinates

// linear interpolation in model space:
"MUL    R1, v[0], c[9].y;"
"MAD    R1, v[4], c[9].x, R1;"
// and the same for the normals:
"MUL    R2, v[1], c[9].y;"
"MAD    R2, v[5], c[9].x, R2;"
// and the same for the color:
"MUL    R3, v[2], c[9].y;"
"MAD    R3, v[6], c[9].x, R3;"
// and the same for the texture
// coordinates:
"MUL    R4, v[3], c[9].y;"
"MAD    R4, v[7], c[9].x, R4;"
// Transform to clip space:
"DP4    o[HPOS].x, R1, c[0];"
"DP4    o[HPOS].y, R1, c[1];"
"DP4    o[HPOS].z, R1, c[2];"
"DP4    o[HPOS].w, R1, c[3];"
// Transform normal according to
// world matrix:
"DP3    R0.x, R2, c[4];"
"DP3    R0.y, R2, c[5];"
"DP3    R0.z, R2, c[6];"
// Dot normal with light direction in
// world space:
"DP3    o[COL0], R0, c[8];"
// Pass through texture coords:
"MOV    o[TEX0], R4;"
"END";
```

Figure 12: *A vertex program for linear attribute interpolation. This code segment is based on the basic linear interpolation in Figure 11. The modelview matrix is stored in constant registers c[0 - 3], while the world matrix is stored in registers c[4-6]. The values of t and $1-t$ are stored in c[9].x and c[9].y respectively.*

An important concern is the potential memory overhead for repeated models. As model attribute and connectivity data must be accessed quickly it is stored in video or AGP memory. So as not to repeat this data we instantiate only one data object, and create model instances which store their own state and point into the data store. On our test platform, there was little performance improvement in using memory on the graphics card over AGP memory (although both were a significant improvement over standard memory).



(a) non-volatile scene



(b) volatile scene

Figure 13: *Our test scene consists of a thousand dolphins, with $|c_0| = 9936$ and $|c_6| = 200$ polygons. The 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 center 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 10(c)).*

Experimental Design and Results

Our test platform is based on the implementation described. An un-textured dolphin model is utilized in the experiments. The g-file for the dolphin has 6 levels ($n = 6$) and varies from 10000 to 200 faces. The initial scene holds 1000 dolphins, which are translated to be within the model interpolation region (i.e. between d_{near} and d_{far}). Objects outside of the view frustum are culled.

Two paths are tested during evaluation. The first (Figure 13(a)) is a non-volatile scene, as there are relatively small fluctuations in the number of polygons. Our LOD control system maintains a steady frame-rate by restricting the number of polygons to be exactly within the tolerance $\epsilon = 5000$. The second (Figure 13(b)) produces a volatile scene, where frame-rate fluctuations cannot be contained (see Figure 10), 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 13(a)) show a speed up

from 71.42 *seconds 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 attributable to the large number of interpolations which are required within each frame of the sequence (for 1000 dolphins at the 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 when the model is close to the camera position causes the texture on the surface to slide considerably. The work of Sander *et al.* [24] would significantly reduce this artefact, and could easily be adapted during model simplification. We leave this as a suitable area for future work.

7. Conclusion and Future Work

We have defined three types of continuity for level of detail sequences: visual, geometric and frame-rate continuity. Visual discontinuity is caused by popping between different levels of detail, and can be minimized by enforcing geometric continuity. In volatile scenes, visual continuity and frame-rate continuity may be irreconcilable.

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 construction using a batched hierarchy during simplification. Every attribute of the resulting mesh has C_0 (positional) continuity, which is important in ensuring visual continuity.

We describe a level of detail control mechanism based on a global control parameter T. This allows the LOD of a scene to be accurately controlled using a single value that corrects the current polygon count. By restraining 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 abrupt level of detail changes.

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

There are several areas of future work that stem from this research. Minimizing visual discontinuities based on volume degradation of the enclosed surface has been shown to work only with untextured models. We have found that texture sliding is a significant artefact. Sander *et al.* [24] provide a method for minimizing texture deviation, which could be used to derive better texture coordinates during simplification. However a method for determining LOD partitions based on textured models is an open problem.

Table 1: *Decreasing attribute and connectivity set sizes of the dolphin model.*

$\lvert c_0\rvert = 9936$	$\lvert a_0\rvert = 5001$
$\lvert c_1\rvert = 5418$	$\lvert a_1\rvert = 2501$
$\lvert c_2\rvert = 2926$	$\lvert a_2\rvert = 1342$
$\lvert c_3\rvert = 1578$	$\lvert a_3\rvert = 712$
$\vdots$	$\vdots$

We have shown that attributes of the g-mesh are C_0 continuous over the range of interpolation. Higher orders of continuity could easily be constructed based on these attribute 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 useful in morphing between a sequence of different models with no distinct one-to-one mapping. This technique would require models with topological equivalence, and a means of finding point correspondences between them.

Acknowledgements

The authors would like to thank Edwin Blake and Patrick Marais for their insightful comments, and Shaun Nirenstein for his assistance with implementation and memory management issues. The cranium model is used with kind permission of the eSkeletons Project (*http://www.eskeleton.org*). Special thanks must be extended to members of the CVC Lab for contributing to our exciting working environment.

Appendix: Derivation of memory overhead

Although dependent on the genus of the original model, the experiments performed by Southern *et al.* [25] indicate that level of detail sequences generated using a batched hierarchy [14] typically reduce in size geometrically, and by a factor ≈ 2 (see Table 1).

So the relationship between consecutive attribute and connectivity sets becomes:

$$|a_{i+1}| = \frac{1}{2}|a_i| \qquad |c_{i+1}| = \frac{1}{2}|c_i|, \tag{2}$$

where a_i and c_i are the attribute and connectivity sets (as defined in Section 4) of a mesh in an LOD sequence of n models, $i = 0, \ldots, n$.

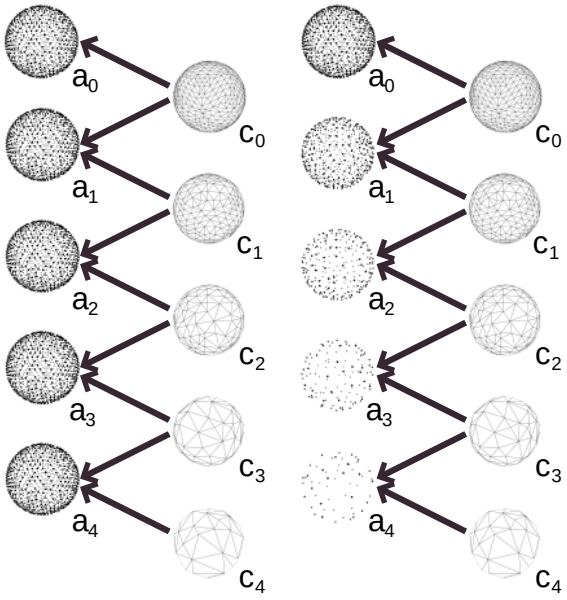


Figure 14: *Two approaches to g-mesh implementation. In (a) the g-mesh an inefficient approach requires all attribute sets in a level of detail to have the same size as the original model. In (b) each connectivity set c_i has two associated attribute sets a_i and a'_i which are used to determine the interpolated model.*

For a theoretical *g*-mesh sequence in which there is no memory wastage (such as in Figure 3(b)), the memory overhead would be the sum of all attribute and connectivity sets of an equivalent level of detail sequence:

$$D = \kappa \sum_{i=0}^{n} 2^{-n}|a_i| + \rho \sum_{i=0}^{n} 2^{-n}|c_i|, \qquad (3)$$

where D represents the total memory overhead of the sequence, and κ and ρ are constants representing the size of a single entry in the attribute and connectivity sets respectively. Clearly, as n tends to infinity, the memory overhead of this LOD sequence is limited to $2\,(\kappa|a_0| + \rho|c_0|)$, which is twice the memory usage of the original model.

In practice, however, a scheme which allows data to be archived in this format while still being quickly accessible for a vertex program approach (outlined in Section 6) has yet to be found.

A naive approach would be to give all attribute sets a_i the same size, i.e. $|a_i| = |a_0|$ for $i = 0, \ldots, n$ (shown in Figure 14(a)). The memory overhead for this technique is prohibitive, since it is equivalent to storing n versions of the model at the same resolution.

As a compromise, we make use of a duplicate (and redundant) attribute set a'_i, which corresponds to the connectivity information stored in c_i, but duplicates attribute information from a_{i+1} (see Figure 14(b)). With this method, the memory overhead D becomes

$$D = \kappa \sum_{i=0}^{n-1} 2^{-n}\left(|a_i| + |a'_i|\right) + \kappa|a_n| + \rho \sum_{i=0}^{n} 2^{-n}|c_i|, \quad (4)$$

where D, κ and ρ are as defined in Equation 3 above. Note that $|a'_i| = |a_i|$, so as n tends to infinity, the total memory overhead cannot exceed $4\kappa|a_0| + 2\rho|c_0|$.

References

1. H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH '96 (Computer Graphics)*, pp. 99–108. 1996.

2. E. Lindholm. Vertex programs for fixed function pipeline. In *Technical report*. NVIDIA, November 2000.

3. K. Singh and E. Fiume. Wires: A geometric deformation technique. In *Proceedings of SIGGRAPH '98 (Computer Graphics)*, pp. 405–414. July 1998.

4. 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)*, pp. 109–118. 1996.

5. D. C. Taylor and W. A. Barret. An algorithm for continuous resolution polygonizations of a discrete surface. In *Proceedings of Graphics Interface*, pp. 33–42. 1994.

6. D. Cohen-Or and Y. Levanoni. Temporal continuity of levels of detail in delaunay triangulated terrain. In R. Yagel and G. M. Nielson (eds), *IEEE Visualization '96*, pp. 37–42. 1996.

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

8. G. Turk. Re-tiling polygonal surfaces. In *Proceedings of SIGGRAPH '92 (Computer Graphics)*, vol. 26, pp. 55–64. 1992.

9. 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)*, vol. 27, pp. 247–254. 1993.

10. J. M. Lounsbery. Multiresolution Analysis for Surfaces of Arbitrary Topological Type, PhD thesis, University of Washington, 1995.

11. E. J. Stollnitz, T. Derose and D. Salesin. *Wavelets for Computer Graphics*. Morgan Kaufmann Publishers, 1996.

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

13. S. Upstill. *The Renderman Companion*. Addison Wesley, 1990.

14. R. Southern, P. Marais and E. Blake. Generic memoryless polygonal simplification. To Appear in ACM *Afrigraph* '01.

15. M. Reddy. Perceptually Modulated Level of Detail for Virtual Environments, PhD thesis, University of Edinburgh, 1997.

16. E. Gobbetti and E. Bouvier. Time-critical multiresolution scene rendering. In *Proceedings of IEEE Visualization*, IEEE Computer Society Press, San Francisco, CA, USA, pp. 123–130. October 1999.

17. M. Grabner. Smooth high-quality interactive visualisation. In *Proceedings of the 17th Spring Conference on Computer Graphics*, pp. 139–148. 2001.

18. C. Zach. Integration of geomorphing into level of detail management for realtime rendering. In *Proceedings of the 18th Spring Conference on Computer Graphics*. April 2002 `http://fractal.dam.fmph.uniba.sk/~sccg/proceeding/-2002/zach.christopher.L1.ps.gz`.

19. M. Slater and A. Steed. A virtual presence counter. *Presence*, 9(5):413–434, 2000.

20. A. Mason. Predictive Hierarchical Level of Detail Optimization, PhD thesis, University of Cape Town, 1999.

21. R. Southern, E. Blake and P. Marais. Evaluation of memoryless simplification. In *Technical Report CS01-18-00*. University of Cape Town, 2001, Available at `http://www.cs.uct.ac.za/Research/CVC/Techrep/CS01-18-00.pdf`.

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

23. H. M. Erik Lindholm and Mark Kilgard. A user-programmable vertex engine. In *SIGGRAPH'01 (Computer Graphics)*, pp. 149–158. 2001.

24. P. V. Sander, J. Snyder, S. J. Gortler and H. Hoppe. Texture mapped progressive meshes. In *Proceedings of SIGGRAPH '01 (Computer Graphics)*, pp. 409–416. 2001.

25. R. Southern, E. Blake and P. Marais. Gems: Generic memoryless polygonal simplification. In *Technical Report CS00-10-00*. University of Cape Town, 2000.