A Stateless Client for Progressive View-Dependent Transmission

Richard Southern Simon Perkins Barry Steyn Alan Muller Patrick Marais Edwin Blake

Collaborative Visual Computing Laboratory
University of Cape Town

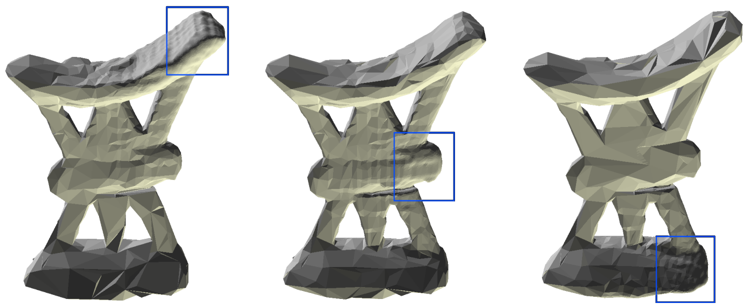

Figure 1: Selective Refinement. The headrest is an object of African cultural heritage. In each frame the selected region is refined progressively.

Abstract

We present a framework for real-time view-dependent refinement, and adapt it to the task of browsing large model repositories on the Internet. We introduce a novel hierarchical representation of atomic operations based on a graph structure, and provide a correspondence between the nodes of this hierarchy and a spatial representation of these operations, called *visibility spheres*. Selective refinement is achieved by performing a breadth first search on the graph. We show that the graph representation allows for significant space savings. The framework presented makes options available for performance tailoring. By efficient traversal of the graph structure an ordered list of refinements can be generated which are progressive and evenly distributed over the refinement area. This list can easily be truncated to comply with polygon limitations indicated by a client.

CR Categories: I.3.2 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Surfaces and Object Representations

Keywords: triangle mesh simplification, view dependent transmission, level of detail, Java 3D

1 Introduction

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[1] and works of art[8, 11]. 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.

A simple solution to this problem is to store many level-of-detail models on the server, and transmit to the client a model of sufficient complexity that it complies with their system limitations. Unfortunately, remote learning applications of highly detailed models typically requires a high degree of refinement (like the chisel marks of Michelangelo).

An alternative is a selective or view-dependent transmission framework, where only the details requested by a client are transmitted and refined. Previous techniques of view dependent transmission [15] 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.
- **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 propose a framework for 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. We adapt the framework presented to allow for the browsing of large model repositories on the Internet.

In Section 2 we discuss the work related to this paper, including a short background of progressive meshes, view dependent refinement and progressive transmission. Section 3 describes a technique to perform view-dependent refinement, while it's conversion into a view-dependent transmission application is discussed in Section 4. In Section 5 we describe our implementation in Java3D, the results of which are discussed in Section 6. Finally we conclude and offer suggestions for future work in Section 7.

2 Background

Progressive meshes [4] provide effective tools for multi-resolution analysis. The model $\hat{M}$ is coarsened into a sequence of lower resolution representations by means of reversible atomic operations which collapse individual edges of a model (see Figure 2). The application of each individual edge collapse (*ecol*) operation results in multiple representations of the original mesh $\hat{M} = M^n$,

$$\hat{M} = M^n \overset{ecol_{n-1}}{\longrightarrow} M^{n-1} \overset{ecol_{n-2}}{\longrightarrow} \ldots \overset{ecol_1}{\longrightarrow} M^1 \overset{ecol_0}{\longrightarrow} M^0.$$

The resulting sequence of meshes $M^0, \ldots, M^n$ provide effective view independent level of detail control, and also permit smooth geometric transformations during the transformation between models (called *geomorphs*). We define the base points in Figure 2 to be the points within the faces surrounding the vertices v_l and v_k which are not affected during the edge collapse or vertex split operation. An edge collapse or vertex split operation is considered to be *legal* if the necessary faces and vertices are present.

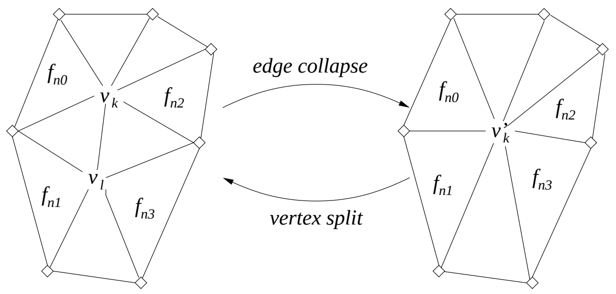

Figure 2: The reversible atomic operations used to produce a multi-resolution representation of the original model. Vertices marked with diamonds indicate the base points of the region which do not change during the decimation.

2.1 View-Dependent Refinement

Most view dependent refinement schemes are distinguished by the way in which the operations are ordered, and the determination of which operations to apply. In [5, 15] 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' (as shown in Figure 6). Xia *et al.* [16] 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.* [10] uses an octree of clusters of vertices and faces in order to collapse vertices together, allowing topology independent simplification. Guéziec *et al.* [3] make use of a directed acyclic graph (DAG) to store partial ordering of edge collapse operations based on their dependencies. We make use of a similar DAG to represent the ordering of the operations. This reduces the representation to roughly half of the nodes required to represent a vertex hierarchy.

The hierarchy of Xia *et al.* [16] constructs the dependencies between the vertices in the merge tree on the premise that no faces in the *region of influence*[1] of two edge collapse operations can be shared. Like Hoppe [5] we find this results in an unnecessarily deep tree, i.e. 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 $f_{n0}, f_{n1}, f_{n2}, f_{n3}$ and the vertex v_k (according to Figure 2).

Spatial subdivision for selective refinement on arbitrary surfaces has been performed with octrees [10], bounding spheres [5] and using subdivision (such as models provided by [7]). 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.

2.2 Progressive Transmission

Typical progressive transmission strategies (e.g. [4, 14]) initially transmit the base mesh M^0 to the client, followed by a sequence of refinements necessary to restore $\hat{M}$ progressively (in [14] refinements take the form of detail coefficients of increasing magnitude). Guéziec *et al.* [3] make use of a compressed representation to provide a mapping between $\hat{M}$ and each consecutive level of detail.

To *et al.* [15] provide a platform for progressive, selective (or view dependent) refinement by constructing a vertex hierarchy (similar to [5, 16]). 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 [16, 5] 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 — a considerable problem when rendering large models without expensive hardware. The technique of To *et al.* provides a technique to transmit a model selectively, but is not practical for the online browsing of large model repositories (such as models from [8]) on low end machines.

3 Method Overview

We construct our view-dependent framework upon the commonly used [6, 4, 16, 5] 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 op-

[1]Xia *et al.* define the *region of influence* of an edge collapse as all faces adjacent to both vertices affected (in Figure 2 these are v_l and v_k).

eration is visible and needs to be performed. These have attributes such as orientation, position, size and relative error (see Figure 3).

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.

3.1 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.

We construct the visibility spheres during the surface simplification process, like [16, 15], 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 [5]), 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 [5]), 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 3.

The orientation of a visibility sphere is represented by a *floating* normal cone $\{\mathbf{n}, \alpha\}$ (first introduced by Shirman *et al.* [13]) of the resultant normal of the one or two faces removed as a result of applying *atom*, and hence the orientation of the removed edge. The error ϵ associated with an *atom* is simply the distance between v_l (the vertex lost) and v_k (the vertex kept).

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

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

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,

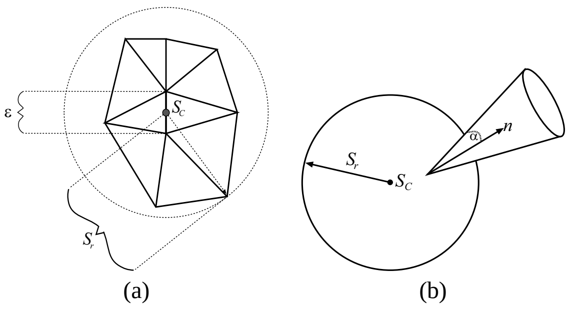

Figure 3: The Visibility Sphere. (a) The determination of the attributes of the visibility sphere. S_r represents the radius of a sphere originating from the centroid of the base points of the region S_C. The error term ϵ indicates the distance between the two points which would be collapsed during the 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).

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[5]. 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

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

Orientated Away

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

$$\frac{\mathbf{e} \cdot \mathbf{n}}{||\mathbf{e}||\,||\mathbf{n}||} > \sin(\alpha). \quad (2)$$

Several recent documents [10, 12] 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 direction vector $\mathbf{e}$ and eye position p, 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 3), as the farthest extents of the normal cone,

$$\epsilon_1' = \frac{\epsilon}{r}\cos(\lambda + \alpha) \text{ and } \epsilon_2' = \frac{\epsilon}{r}\cos(\lambda - \alpha) \quad (3)$$

If $\max\{\epsilon_1', \epsilon_2'\} < \tau$ then no vertex split operations enclosed within the tested visibility sphere would be noticeable when applied.

3.2 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 [5, 15], 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.

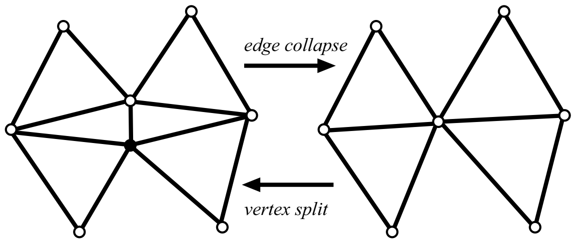

Figure 4: The dependent vertices for a vertex split are shown by the white filled points. The black filled point indicates the vertex which is lost after the edge collapse has been applied (earlier defined as v_l).

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 $f_{n0}, f_{n1}, f_{n2}, f_{n3}$ (Figure 2) need be present before a particular vertex split can be performed (see Figure 4). 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). 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*$_i^c$, $i = 1 \dots n$, and
- the visibility spheres associated with the children *vs*$_i^c$, $i = 1 \dots n$,

the visibility sphere associated with *vsplit*^{p} is then

$$vs^p \cup vs_1^c \cup \dots \cup vs_n^c.$$

This dependency structure results in a problem, shown in Figure 6. The actual spatial problem is shown in Figure 6(a) — a single vertex (v_1) is split twice into v_2 and v_3, by vertex splits vs_1 and

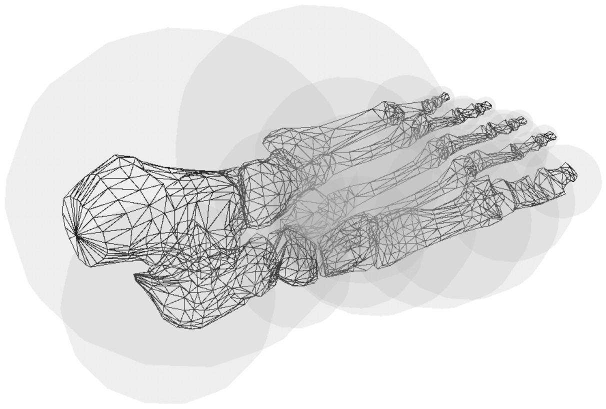

Figure 5: Bounding spheres. These spheres represent the spheres bounding the coarsest level of refinement associated with the root nodes of the vertex split hierarchy.

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 [5]) (in Figure 6(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 6(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 shared parent (in this case vs_p) since they both originate from the vertex v_1.

3.3 Selective Refinement

The vertex split hierarchy defined in Section 3.2 provides an elegant correspondence between the visibility spheres defined in Section 3.1 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.

4 View Dependent Transmission

In Sections 3.1 and 3.2 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 3.3. 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 [15, 10]). The client informs

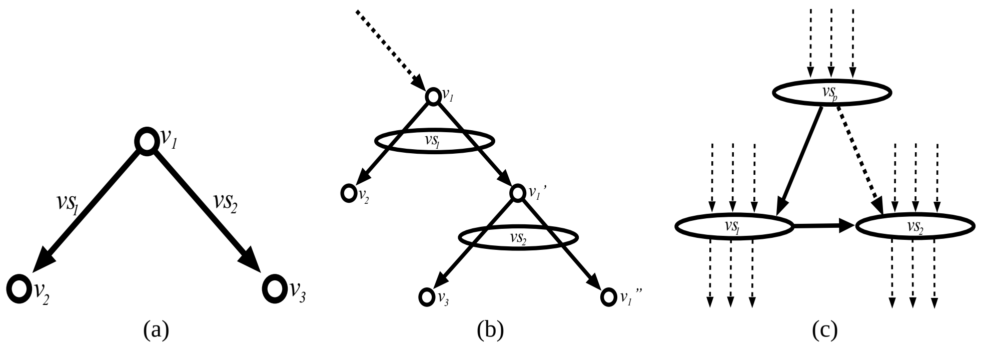

Figure 6: A comparison of a vertex hierarchy and a vertex split hierarchy. The initial problem is shown in (a), the vertex hierarchy solution in (b) and the vertex split hierarchy solution in (c). In (c) the removed dependency is indicated by a dotted arrow.

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 9).

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 [5, 10], 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 7 that the polygon count of a remotely viewed model can be kept relatively constant when applying coherence.

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 Implementation

Our web-based application is based on a classic client-server model, and allows users to retrieve individual models selectively from the Internet. The server is initialised with a sequence of specially formatted mesh files which reflect the hierarchy described in Section 3.2. These files are the result of an external preprocess which converts a version of the Progressive Mesh[4] format (i.e. $PM = \{M^0, vsplit_1, \ldots, vsplit_n\}$), modified such that each $vsplit_i$ element includes the indices of the dependent vertices indicated in Figure 4.

A client specifies a server, and requests a model from a list of those available. The server spawns a new thread, and transfers the base mesh M^0 to the client. Client then sends view parameters to

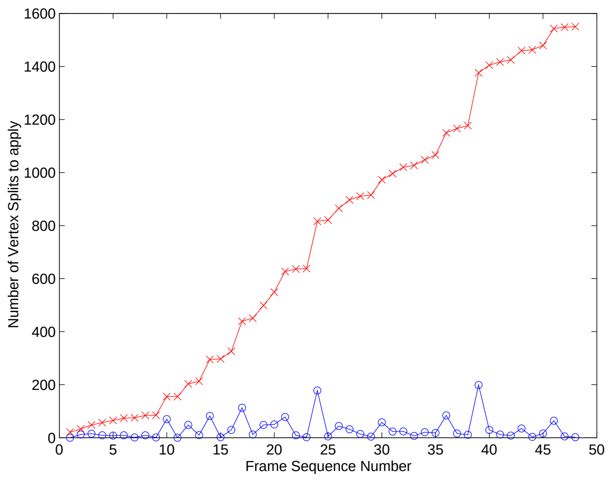

Figure 7: The benefit of coherence. The view frustum is moved across a surface in a preset path. Crossed points indicates transmission with refinement (vertex splits) only, while circled points indicate transmission of both vertex split and edge collapse operations taking advantage of coherence between 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.

the server, to which the server responds with an ordered sequence of refinements. The optimisations in Section 4 are also available to the client.

We implemented our server and client with Java3D. It offers high-performance graphics in addition to the portability offered by the Java platform. The hierarchy was linearised to improve access times of elements within the graph. The client process consists of two threads, a listening thread and a main thread. The main thread is used for interacting with the user and sending requests to the server, while the listening thread waits for a responses from the server. The communication between the client and the server is therefore asynchronous. This allows the user to manipulate the mesh interactively.

We use TCP/IP sockets as the communication protocol between the client and the server. **BufferedOutputStream** and **BufferedInputStream** classes are used to buffer the output and the input socket streams in order to increase performance. We use an **IndexedTriangleArray** geometry class for the purposes of rendering our meshes. Each time a sequence of atomic operations is applied to the mesh, we generate a new **IndexedTriangleArray** object instead of maintaining a single **IndexedTriangleArray** which contains space for all of the mesh geometry. We do this in order to minimise the number of triangles being processed for rendering. We use Java3D's mixed immediate-retained rendering mode since we are constantly changing the geometry of our object. Unfortunately the continuous modification to the models geometry did not permit the optimisations gained by compiling retained mode Scene Graph geometry.

We implement our cache using a fixed size queue and a hash table. We use the queue to determine the age of the atomic operations within the cache. If there is no space in the queue, the atomic operations at the front of the queue (i.e. the oldest operations) are removed.

6 Results

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 $\mathcal{O}(\log n)$) to traverse, but since the vertex list must be traversed every time in order to transmit the necessary refinements, both run in $\mathcal{O}(n)$. It is clear that the vertex split hierarchy has roughly half the number of nodes, and is about 70% of the physical size of the vertex hierarchy.

	Vertex Hierarchy		**Vertex Split Hierarchy**	
Mesh	# nodes	size (bytes)	# nodes	size (bytes)
bones	4143	99 146	1989	63 648
bunny	70 395	1 548 690	34 448	1 102 336
headrest	288 826	6 354 172	144 189	4 614 048

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 1, 8 and 9. In Figure 9 the region surrounding the eye of the bunny is refined with various polygon limitations. In Figure 8 and 1 three selectively refined regions are shown in each case.

7 Conclusion and Future Work

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. We are in the process of implementing a web-based model repository of un-textured African art models (such as those in Figure 1) using this technique. The techniques described could also be integrated into existing web-based 3D applications.

In the future we would like to see the translation of the appearance preserving simplification representation of Cohen *et al.* [2] and progressive hulls of [12] into a view-dependent refinement technique. The partitioning system of [9] used for mesh simplification could also be applied to surfaces to provide selective refinement across surface patches of large models. Discontinuities at seams of these patches would have to be addressed to prevent unsightly cracks during selective refinement.

Acknowledgements

We would like to thank the Stanford repository for their use of the *bunny* and *bones* models, and the Contemporary African Music and Arts Archive (CAMA), Otto-Carl Marte, Caleb Lyness and Bryan Wong for the use of the *headrest* model.

References

[1] M. J. Ackerman. The visible human project. In *Proceedings of the IEEE*, 1998.

[2] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *Proceedings of SIGGRAPH*, 1998.

[3] A. Guéziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for polygonal transmission of polygonal surfaces. In *VRML*, 1998.

[4] H. Hoppe. Progressive meshes. In *Proceedings of SIGGRAPH*, 1996.

[5] H. Hoppe. View-dependant refinement of progressive meshes. In *Proceedings of SIGGRAPH*, 1997.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. Technical report, University of Washington, Seattle, 1993.

[7] A. Lee, W. Sweldens, P. Schröder, L. Cowsar, and D. Dobkin. Maps: Multiresolution adaptive parameterization of surfaces. In *Proceedings of SIGGRAPH*, 1998.

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

[9] P. Lindstrom. Out-of-core simplification of large polygonal models. In *Proceedings of SIGGRAPH*, 2000.

[10] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proceedings of SIGGRAPH*, 1997.

[11] C. Lyness, O.-C. Marte, and B. Wong. Model reconstruction for a virtual interactive environment. Technical Report CS99-11-00, University of Cape Town, 1999.

[12] P. Sander, X. Gu, S. Gortler, H. Hoppe, and J. Snyder. Silhouette clipping. In *Proceedings of SIGGRAPH*, 2000.

[13] L. A. Shirman and S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Eurographics*, 1993.

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

[15] D. To, R. Lau, and M. Green. A method for progressive and selective transmission of multi-resolution models. In *ACM Virtual Reality Software and Technology*, 1999.

[16] J. C. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. In *IEEE Transactions on Visualization and Computer Graphics*, 1997.

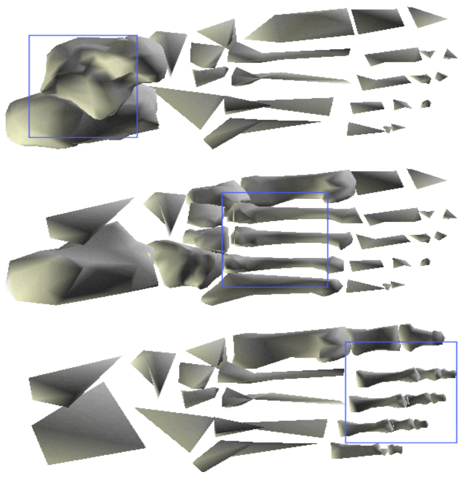

Figure 8: Selective Refinement. Separate sections of the model are refined in each frame.

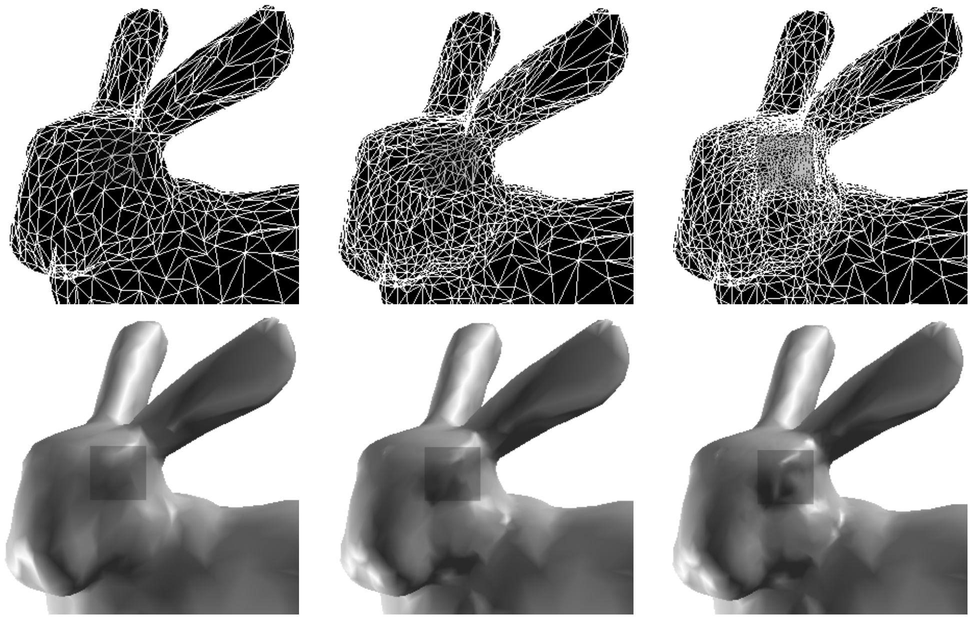

Figure 9: Progressive Refinement. The bunny's eye is progressively refined by setting the polygon limit to 1000, 1500 and 2000 polygons respectively.