2
SIMPLIFYING MASSIVE DATASETS WITH COLOR AND TEXTURE IN 3D REAL-TIME GAME ENGINE DEVELOPMENT

Tan Kim Heok, Abdullah Bade, Daut Daman

INTRODUCTION

Computational demanding paradigm like three dimensional interactive applications always requires the simulation and display of a virtual environment (VE) at interactive frame rates. Even with the use of powerful graphics workstations, a complex VE can involve a vast amount of computation, inducing a noticeable lag into the system. This lag can severely compromise the display quality.

Therefore, a lot of techniques have been proposed to overcome the delay of the display. It includes motion prediction, fixed update rate, visibility culling, frameless rendering, Galilean antialiasing, level of detail, world subdivision or even employing parallelism (Reddy 1997). Level of detail is certainly a great way in resolving this problem.

A long time ago, programmers have used Level of Detail (LOD) techniques to improve the performance and quality of their graphics systems. The LOD approach involves retaining a set of representations of each polygonal object, each with different levels of triangle resolution. During the animation rendering stage, objects deemed to be less important are displayed with a low-resolution representation. Whereas object of higher importance is displayed with higher details (Figure 2.1).

Due to the increasing size of datasets, the problem of dealing with the meshes that are apparently larger than the main memory existed. Thus, these data can only be rendered on high end computer system. These massive data are practically impossible to fit in available main memory in desktop personal computer. As a result, it is very cost ineffective and not user friendly.

Figure 2.1 Different Level of Detail of Buddha Model

Suppose this problem can be solved by simplification process, however, the conventional simplification technique need to load the full resolution mesh into main memory in order to perform the task. Consequently, out-of-core approach is invented to overcome this deficiency. Lindstrom (2000) is the first

researcher who optimized the secondary memory usage instead of depending on main memory in enormous mesh simplification.

In many computer graphics application, the realism of the virtual environment is very important. Therefore, the preservation on surface attributes is essential during the geometric simplification process. Surface attributes, like normal, curvature, color and texture show the details of the object. Without them, the rendered scene will looked dull and unattractive to user.

PREVIOUS WORK

In this section, some previous works in existing simplification operators, error metrics, spatial data structure (tree), level of detail framework and some recent out-of-core approaches are presented. In these methods, attention will draw to vertex clustering simplification method, quadric error metrics, octree structure and view-dependent framework.

The geometric simplifications operators include vertex removal, edge collapse, half-edge collapse, vertex pair contraction (virtual edge), triangle collapse, vertex clustering and face clustering. Vertex removal, which is first introduced by Schroeder et al. (1992) performs by removing a vertex and applying patching to cover the created hole. Edge collapse (Hoppe 1996) is the highest quality operator that contracts an edge to be a single vertex. Half edge collapse chooses one of the vertex of an edge to be the representative vertex, thus the quality are poorer. Vertex pair contraction enable any pair of vertices to be merged even it is not an edge. Alternatively, triangle collapse converts three vertices into a vertex. Vertex clustering (Rossignac and Borrel 1993) contracts all the vertices in a cell into an optimal vertex. Whereas face clustering merge nearly coplanar faces into large clusters of faces and it is less popular due to its low quality.

From all the simplification operators, vertex clustering is the fastest algorithm. Low and Tan (1997) proposed a slight variation on vertex clustering by using "floating cell" instead of static grid. The most important vertex in a cell is chosen as representative vertex. It offers higher quality result but slower computation time. Luebke and Erikson (1997) use vertex clustering together with Octree in their view-dependent refinement.

Typically geometric error metric can be done locally or globally. The Hausdorff distance is probably the most well-known metrics for making geometric comparisons between two point sets. This metric is defined in terms of another metric such as the Euclidean distance. Quadric error metrics is based on weighted sums of squared distances (Garland and Heckbert 1997). The distances are measured with respect to a collection of triangle planes associated with each vertex. This technique is fast and good fidelity even for drastic reduction. This work is extended to preserve color and texture attributes (Garland and Heckbert 1998).

Spatial data structures are used to store geometric information. The data or the object can be divided uniformly or adaptively. BSP, quadtree, octree, kd-tree, R-tree are some examples of spatial data structures. BSP divides world using line (2D) or plane (3D). Quadtree is used in 2D environment whilst Octree in 3D environment. Shaffer and Garland (2001) uses two passes in out-of-core approach by first performing uniform clustering like Lindstrom (2000) then adaptive subdivision using BSP to re-cluster the mesh.

OEMM (Cignoni et al. 2002) uses Octree to partition the mesh and edge collapse to simplify the mesh. It is slower but higher quality. There are four types level of detail framework, including discrete LOD, continuous LOD, view-dependent LOD and hierarchical LOD. Discrete LOD generates all the static LOD during preprocessing stage but continuous LOD creates data structure from which a desired LOD can be extracted during runtime. View-dependent solved problem with large object and can

span several resolution over a mesh. However, it is slow. Hierarchical LOD can sit atop of discrete LOD and view-dependent LOD to solve simplification on small object. There are implementations on discrete LOD due to its simplicity like in (Correa 2003). View-dependent LOD and hierarchical LOD are also common nowadays. There is always combination of view-dependent LOD and hierarchical LOD in geometric simplification. There are some works on out-of-core simplification for view-dependent refinement. Hoppe (1998) partitions terrain in hierarchical format and seams are simplified further. Then, Prince (2000) applied it to arbitrary mesh, but it needs too much of RAM hence slow. Cignoni et al. (2002) extended it using an efficient data structures but it is not reported how this data structure applied to view-dependent refinement. El-Sana and Chiang (2002) also segment the mesh and use edge collapse technique but did not using any large model to test their out-of-core technique.

Lindstrom (2000) is the pioneer in out-of-core simplification field. He created a simplification method, called OOCS which is independent of input mesh and able to simplify extremely large datasets as long as the output size is smaller than the available main memory. Besides, less memory simplification was also introduced. Afterward, enhancement is done on dropping the dependency on output mesh as well in memory insensitive simplification - OOCSx (Lindstrom and Silva 2001). At the same time, many approaches were introduced by previous researchers in simplifying the massive datasets (Borodin et al. 2003; Guthe et al. 2003; Isenburg et al. 2003).

ALGORITHM FRAMEWORK

This research introduces an approach for end-to-end and out-of-core simplification and view-dependent visualization of large

surfaces. Besides, appearance preservation is proposed as well. Here, the arbitrarily large datasets can be visualized by given a sufficient amount of disk space. The work starts from a modified memory insensitive simplification (OOCSx) (Lindstrom and Silva 2001). Then, construction of a multi-resolution hierarchy is conducted. Finally, view-dependent rendering on output mesh is carried out during run-time. The framework overview is shown in Figure 2.2.

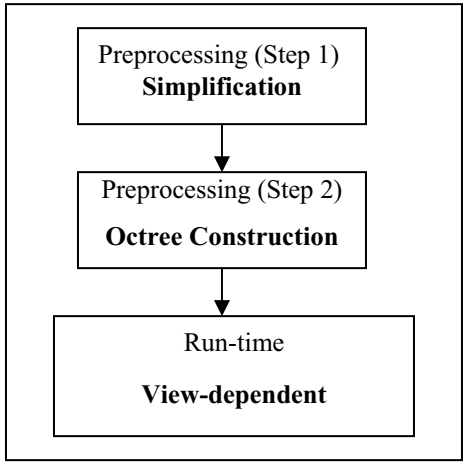

Figure 2.2 Framework Overview

Algorithm starts with a modified memory insensitive simplification (Lindstrom and Silva 2001). This technique eliminates the dependency on main memory size. The input mesh is subdivided into uniform rectilinear grid. Next, vertex clustering

simplification (Rossignac and Borrel 1993) is performed on each cell. This simplification operator may introduce non-manifold vertices and edges. Additionally, it may produce less quality output mesh. Nevertheless, it is fast and suitable for our interactive application. During simplification, generalized quadric error metric (Garland and Heckbert 1998) is used to find the optimal representation vertex and calculate the simplified color or texture attribute. These tasks are performed on-disk and therefore random accesses by using a sequence of external sorts should be avoided. The output of this stage is a set of triangles and a set of quadric matrices for its vertices.

In building the Octree, each triangle is associated with each node in the tree. Each of them represents a grid cell in partitioned mesh and has position and resolution. The triangles file and vertex file are sorted based on the grid cell so that the neighbors are stored together. The generated parent nodes are output sequentially to a temporary file for a certain resolution. It is repeated until all levels in Octree are represented. Lastly, we will get a temporary file with single root node. Then, hierarchical structure is created in a single file by linking all the levels from the coarsest to the most detail level. In the run-time phase, the visible nodes are extracted from Octree. Each of them is considered as active nodes and the vertex information is loaded into a dynamic data structure. The active nodes are expanded and collapsed based on view-dependent criteria. We make the rendering and refinement stages run in parallel.

SIMPLIFICATION

The main flow of work in OOCSx is similar with vertex clustering (Rossignac and Borrel 1993) simplification process. The only

differences are the way of data is handled the process to find the optimal vertex.

First, the mesh is subdivided into a uniform rectilinear grid. Because our mesh is in three dimensions, therefore, the grid is constrained to only 2^m x 2^m x 2^m dimension. *m* is the number of space subdivision in order to get the most detail mesh after simplification. The most detail mesh here points to vertices in the leaf nodes. These leaf nodes will be stored as leaf nodes in (m+1) level in Octree later on. The root has cluster ID G(x) = 1. For the k^{th} child of a node's ID is calculated as 8 G(x), + k, where *k*=0, 1, 2, 3, 4, 5, 6, 7.

We process the mesh, which is represented as a triangle soup. A triangle soup is a sequence of triplets of vertex coordinates. Steps to perform simplification are similar with memory insensitive algorithm (Lindstrom and Silva 2001) as follows:

- Read one triangle, t = (x^t_1, x^t_2, x^t_3) at one time. Then, compute two orthonormal unit vectors e_1 and e_2 which lie in the plane for t. Determine the grid location $G(x^t_i)$ for each vertex (x^t_i). We generate two files :
  - Plane equation file : store < $G(x^t_i)$, e_1, e_2>for each vertex in each triangle
  - Triangle cluster: store 3 grid location < $G(x^t_1)$, $G(x^t_2)$, $G(x^t_3)$>(for vertices which fall into 3 different cluster)
- Sort plane equation file using G as primary key using *rsort* (Linderman 1996).
- Compute quadric q using e_1 and e_2 to get optimal vertices x with its color or texture attribute for each cluster (will be explained in following section). The output is <$G(x^t_i)$, x>.
- Replace cluster IDs in triangle file with corresponding vertices. It is done by creating triangle cluster ID with referring to the file which all the vertices of each triangle.

This stage generated two files for finest resolution in leaf nodes for Octree; the triangle file and the cluster ID. The cluster ID file is associated with its optimal vertex position and its surface attributes.

Vertex Clustering

Vertex clustering can also be called spatial clustering as it partitions the space into simple convex 3D regions. Because the mesh always represented in Cartesian coordinate system, the easiest way is to do rectilinear space partitioning. Figure 2.3 shows how the vertex clustering works in 2D diagram.

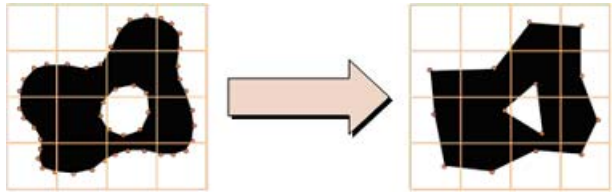

Figure 2.3 Spatial Clustering Process (Lindstrom 2003)

GENERALIZED QUADRIC ERROR METRICS

This generalized quadric (Garland and Heckbert 1998) is created from quadric error metrics (Garland and Heckbert 1997). This is because original quadric error metrics only handles geometry primitives (vertex position) in simplification. Although it is extended from previous quadric error metric, however, it needs the

normal of the plane and two orthonormal unit vectors e_1 and e_2. Let us look at where is the unit vector from in Figure 2.4.

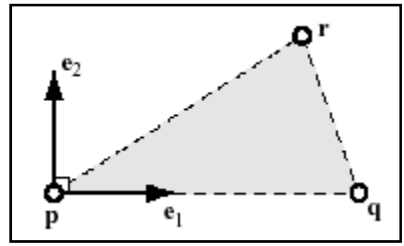

Figure 2.4 Orthonormal vector e_1 and e_2 define the local frame with origin *p* for triangle T (Garland and Heckbert 1998)

Consider the triangle T = (p, q, r) and we assume that all properties are linearly interpolated over triangles. If it has color attribute, then p=(p_x, p_y, p_z, p_r, p_g, p_b). If it has texture, then p=(p_x, p_y, p_z, p_s, p_t). To compute e_1 and e_2:

$$e_1 = \frac{q-p}{\|q-p\|} \qquad (1)$$

$$e_2 = \frac{r-p-(e_1 \bullet (r-p))e_1}{\|r-p-(e_1 \bullet (r-p))e_1\|} \qquad (2)$$

Squared distance D^2 of an arbitrary x from T is $D^2=v^TAx+2b^Tx+c$ like in original quadric error metrics where:

$$\begin{aligned} A &= I - e_1 e_1^T - e_1 e_1^T \\ b &= (p \bullet e_1)e_1 + (p \bullet e_2)e_2 - p \\ c &= p \bullet p - (p \bullet e_1)^2 - (p \bullet e_2)^2 \end{aligned} \qquad (3)$$

By solving Ax=b, we get the optimal vertex *x* with its simplified surface attributes as well. To simplify different types of meshes, refer Table 2.1 (Garland and Heckbert 1998).

Table 2.1 Space Requirement

Model type	**Vertex**	**A**	**Unique coefficients**
Geometry only	$(x\ y\ z)^T$	3x3	10
Geometry+2D texture	$(x\ y\ z\ s\ t)^T$	5x5	21
Geometry+color	$(x\ y\ z\ r\ g\ b)^T$	6x6	28
Geometry+normal	$(x\ y\ z\ a\ b\ c)^T$	6x6	28

OCTREE CONSTRUCTION

In this phase, we construct a coarse to refine level of detail representation of the mesh in a tree data structure, called octree.
From previous process, we have the simplified mesh in the triangle file. Next, we begin to construct the internal node of the Octree. It is done by scanning the triangle file (from previous stage) sequentially to generate every levels of Octree. As mentioned before, the triangles in level (m+1) is already generated in simplification process. Thus, we start from level L=m until m=1 (root node). The steps are similar with simplification process with little differences:

- Read one triangle, t = ($G(x^t_1)$, $G(x^t_2)$, $G(x^t_3)$) at one time. Then, compute two orthonormal unit vectors e_1 and e_2 which lie in the place for t. Determine the grid location $G(x^t_i)$ for each vertex (x^t_i). We generate two files :

- Plane equation file : store < $G(x^t_i)$, e_1, e_2 >for each vertex in each triangle
- Triangle cluster: store 3 grid location < $G(x^t_1)$, $G(x^t_2)$, $G(x^t_3)$>(for vertices which fall into 3 different cluster)

- Sort plane equation file using G as primary key using rsort (Linderman 1996).
- Compute quadric q using e_1 and e_2 to get optimal vertices x with its color or texture attribute for each cluster (explained in previous section). The output is <$G(x^t_i)$, x, q>.
- Additionally, file offset within V_L for each child node is recorded and stored with its parents.
- Lastly, we have the triangle file and a file with cluster ID with its optimal vertex for level L.

The final step is to link the nodes together in a single hierarchical structure file H using the offsets. Similar to multi-resolution methods, we store the multi-resolution structure from coarse to fine resolution.

VIEW-DEPENDENT RENDERING AND REFINEMENT

The refinement and rendering are run in two threads. In refinement process, we use Best First search in finding the active nodes. Best First search is a breadth first search with added heuristics. This search enables the mesh to be updated progressively. Thus, popping effects will not occur. This search also ensures the detail is paged in and added evenly over the visible mesh. These are all benefits from breadth first search. Meanwhile, we control our frame rate using some heuristics.

We begin refinement process by creating the root node into active node. In refinement process, we expand or collapse node. Child nodes are added when we need more detail, otherwise

collapse the nodes. However, collapse and expand action only can be applied to active nodes which are stored in dynamic data structure in main memory.

During rendering, we test on the boundary of node to the view-frustum planes. If it is visible, expand the nodes. Otherwise, collapse it in coarser form but not totally cull it off. This is to avoid sudden changes in viewing perspective.

If a node is visible, then compare the error threshold based on distance aspect with the node's quadric error. If quadric error is lesser, collapse the node otherwise expand the node.

EXPECTED RESULTS

The proposed framework is expected to handle datasets which is larger than available main memory size on low cost personal computer. At the same time, it is able to preserve the surface attributes. The surface attributes here are pointing to color or texture details only. It can generate at least 15 frame rates per second during run-time. Meanwhile, the output is view-dependent. Figure 2.5 below shows the example of our expected output.

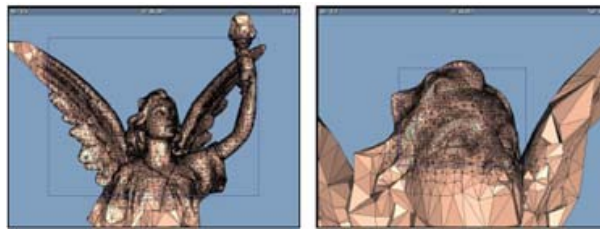

Figure 2.5 Simplification on Lucy Model. The Portion of the Mesh, which Lie Outside of View Frustum is in Low Resolution

CONCLUSION

We have proposed an outline to simplify the massive datasets, which has millions of polygon and preserve the surface attributes (color and texture) after simplification process. To handle out-of-core datasets, we have modified memory insensitive algorithm (Lindstrom and Silva 2001) so that it can be used in simplifying the mesh in geometry, color and texture aspects. Conventional vertex clustering simplification operator is also applied. Even though it produced a quite low quality output, but, it is enough for game application. Accuracy is not that vital here like in medical visualization. We have adopted the generalized quadric error metric (Garland and Heckbert 1998) as the original quadric error metric is not able to handle the surface attributes. This error metric is robust and pretty accurate. Octree is used to make the data more organized and easier to retrieve during run-time. Best first search used here has the potential to get a faster solution in tree searching. View-dependent refinement and rendering are run asynchronously.

REFERENCE

BORODIN, P., GUTHE, M. AND KLEIN, R. 2003. Out-of-Core Simplification with Guaranteed Error Tolerance. In *Vision, Modeling and Visualization 2003*, Munich, Germany, 309-316.

CIGNONI, P., MONTANI, C., ROCCHINI, C. AND SCOPIGNO, R. 2002. External Memory Management and Simplification of Huge Meshes. *Visualization and Computer Graphics, IEEE Transactions*, 9(4), 525-537.

CORREA, W.T. 2003. *New Techniques for Out-of-Core Visualization of Large Datasets*. Ph.D Thesis, Princeton University.

GARLAND, M. AND HECKBERT, P. S. 1997. Surface Simplification Using Quadric Error Metrics. In *Proceedings of SIGGRAPH 97*, ACM Press. Los Angeles, California, Whitted, T. ed., 209-216.

GARLAND, M. AND HECKBERT, P. S. 1998. Simplifying Surfaces with Color and Texture Using Quadric Error Metrics. In IEEE Visualization '98, Ebert, D., Hagen, H. and Rushmeier, H. eds., 263-270.

GARLAND, M. 1999. *Quadric–Based Polygonal Surface Simplification*. Ph.D. Thesis, Carnegie Mellon University.

GUTHE, M., BORODIN, P. AND KLEIN, R. 2003. Efficient View-Dependent Out-of-Core Visualization. In *Proceeding of The 4th International Conference on Virtual Reality and Its Application in Industry (VRAI'2003)*.

HOPPE, H. 1996. Progressive Mesh. In *Proceeding of SIGGRAPH 96. Computer Graphics Proceedings, Annual Conference Series*, New Orleans, Louisiana, Rushmeier, H. ed., 99-108.

HOPPE, H. 1998. Smooth View-Dependent Level-of-Detail Control and Its Application to Terrain Rendering. In *IEEE Visualization '98*, IEEE, Research Triangle Park, North Carolina, D. Ebert, H. Hagen, and H. Rushmeier, Eds., 35–42.

ISENBURG, M., LINDSTROM, P., GUMHOLD, S. AND SNOEYINK, J. 2003. Large Mesh Simplification using Processing Sequences. *Proceedings of Visualization 2003*, IEEE, Seattle, Washington, 465-472.

LINDSTROM, P. AND SILVA, C. 2001. A Memory Insensitive Technique for Large Model Simplification. In IEEE Visualization 2001, San Diego, CA, 121-126.

LINDSTROM, P. 2000. *Model Simplification using Image and Geometry-Based Metrics*. Ph.D Thesis, Georgia Institute of Technology.

LINDSTROM, P. 2003. *Out-of-Core Surface Simplification*. California: *University of California, Davis, lectures on "Multiresolution Methods" February 2003.*

LOW, K. L. AND TAN T. S. 1997. Model Simplification using vertex-clustering. In *1997 ACM Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, Phode Island, Cohen, M. and Zeltzer, D. eds., 75-82.

LUEBKE, D. AND ERIKSON, C. 1997. View-dependent Simplification of Arbitrary Polygonal Environments. In *Proceedings of SIGGRAPH 97*, ACM Press, Los Angeles, California, T. Whitted, Ed., Computer Graphics Proceedings, Annual Conference Series, 199–208.

PRINCE, C. 2000. *Progressive Meshes for Large Models of Arbitrary Topology*. Master's Thesis, University of Washington.

REDDY, M. 1997. *Perceptually Modulated Level of Detail for Virtual Environment.* Ph.D Thesis, University of Edinburgh.

ROSSIGNAC, J. AND BORREL, P. 1993. Multi-resolution 3d Approximations for Rendering Complex Scenes. In *Modeling in Computer Graphics*, Springer-Verlag, Falciendo, B. and Kunii, T. L. eds., 455-465.

SCHROEDER, W. J., ZARGE, J. A. AND LORENSEN W. E. 1992. Decimation of Triangle Meshes. In *Computer Graphics (Proceeding of SIGGRAPH 92)*, Chicago, Illinois, Catmull, E. E. ed., 65-70.

SHAFFER, E. AND GARLAND M. 2001. Efficient Simplification of Massive Meshes. In *12th IEEE Visualization 2001 Conference (VIS 2001)*, San Diego, CA, 127-134.