View-Dependent Level-of-Detail Modeling with Material Preserving

Jing-Yen Huang
jihuang@csie.nctu.edu.tw

Jung-Hong Chuang
jhchuang@csie.nctu.edu.tw

Jun-Ming Su
jmsu@csie.nctu.edu.tw

Department of Computer Science and Information Engineering
National Chiao Tung University
Hsinchu, Taiwan, Republic of China

Abstract

Level of detail (LOD) modeling or mesh reduction has been found useful in interactive walkthrough applications. In particular, view-dependent LOD modeling has its strength in selective refinement and effective view and back-face culling. In this paper, we propose a view-dependent LOD modeling in which the selective refinements is performed according to a dependency graph. The dependency graph is constructed based on a progressive meshing scheme that is clustering-based and takes both geometry and topology simplification into account. In the dependency graph, nodes represent the simplification step while the edge representing the relation between nodes. The relation associated with each edge is classified as either inheritance or dependence relation in order to facilitate the use of spatial and temporal coherence. In the run-time or selective refinement phase, a scheme is proposed to reduce the amount of selective and gauged nodes. Our experiments reveal that the proposed scheme provides a steady and efficient computational efficiency in the process of selective refinement.

Keywords: Virtual Reality, View-Dependent Level of Detail, Dependency Graph, Progressive Mesh, Material Preserving

1 Introduction

In virtual reality applications, maintaining a fast and constant frame rate is crucial for achieving a smooth and realistic visual perception. One way to achieve a fast frame rate is to reduce the polygon flow that is sent to the graphics pipeline for shading. Traditional methods, which involve clipping, hierarchical traversal, and culling, are no longer sufficient for complex virtual environments. Therefore, many techniques of reducing polygon have been introduced, such as level of detail (LOD), progressive mesh (PM) etc. Then, we introduce these relative techniques as following:

Levels of detail (LOD) is a common heuristic technique for polygon flow reduction [11, 16, 2, 10, 17]. A detailed representation is used for rendering when the object is close to the viewer, and substituted by its coarser approximations as the object recedes. LOD modeling can be performed as a preprocessing, or at run-time, rendering view-dependent LOD meshes, leading to view-independent LOD meshes.

Sets of view-independent LOD meshes are appropriate for many applications, but problems arise when rendering complex models. While changing the model between levels of LOD, non-continuous view-independent LOD would result in bothersome popping effect. Therefore, progressive mesh have been emphasized aiming to provide a continuous LOD, progressive refinement, and progressive transmission. Current progressive meshing algorithms, however, tend to collapse only edges or triangles, and hence possess a very long sequence of meshes. Another issue that is important but has been addressed less is the preserving of material property, especially the discontinuity of material attributes such as color.

However, many faces of the model may lie outside the view frustum and processing these will cause extra cost. Similarly, it is often unnecessary to render back-face that also produces cost. Hence, view-dependence LOD provides a solvable method. It only renders meshes which lie inside view frustum to accelerate rendering time. Unfortunately, the dependency problem maybe raise when each vertex-split or edge-collapsing during real-time simplification. Xia [19] first introduces the dependency problem that will cause incorrect result during vertex-split or edge-collapsing. This problem can be solved by that these dependencies are easily identified and stored in the progressive mesh structure during its creation. Although solving the dependency problem might sometimes get lesser simplification than disregarding it, it has the advantage of preserving the correct appearance of model.

In this paper, we propose a view-dependent LOD modeling according to a dependency graph to perform a fast selective refinement. The dependency graph is constructed based on a progressive meshing scheme using vertex clustering algorithm with geometry simplification and material preservation [20]. The principal contributions of this paper are:

1. To extern view-independent progressive mesh scheme [20] into view-dependent.

2. Proposing a dependency graph, which solves the dependence problem, reduces nodes operated to accelerate selective refinement efficiency.

3. Fast dynamic walkthrough by simple criteria of selective refinement and real time operation for local refining and coarsening.

The paper is organized as follows. Section 2 outlines related work. Our approach is described in Section 3 and 4. Implementation issues and experimental results are shown in Section 5. Finally, conclusions and future works are drawn in Section 6.

2 Related Work

Typical hierarchical representation based on view-dependent LOD modeling is the vertex-tree structure that is formed simplification sequence of edge collapsing [19, 9, 13, 3].

2.1 Edge Collapsing VS. Vertex Collapsing

Hoppe [8] describes a progressive meshing method based on edge-collapsing operation. Edges are first ordered according to the cost that is the result of an energy minimization function. The cost in general measures the amount of error introduced into the model as the result of collapsing the edge. Edges are then repeatedly collapsed. At each collapsing, the edge of the lowest cost is collapsed and the costs of adjacent edges are updated. Each edge collapsing yields a mesh with two triangles less then the mesh of previous level. The result is a *base mesh* together with a sequence of edge-collapsing records, each of which can be used to recover finer representations of the mesh. Edge-collapsing methods that incorporate with different cost evaluations have been described in [5, 14, 15, 1, 11].

Edge-collapsing algorithms make good preservation of model features; but one edge-collapsing removes two triangles at most. If a mesh is composed of m triangles and is simplified into a coarse mesh of m_0 triangles, the resulting progressive mesh will have $n=(m - m_0)/2$ vertex splits at least because each vertex split introduces two triangles at most. So edge-collapsing algorithms are very slow and cost a large memory. Evans [3] uses test of virtual edge to increase reduction-ratio for edge-collapsing algorithms, but increases the time complexity.

Xia [19] operates edge-collapsing like the [8], but choosing the representative vertex is different. During dynamic walkthrough, they maintain an *active node linked list* representing the recent model shape on vertex tree. Changing the level of LOD is equal to modify the active node linked list toward top (local coarsening) or bottom (local refining).

2.2 Vertex Clustering

The method uniformly divides the space occupied by a triangle mesh into cells, selects a representative vertex with the highest visual importance for each cell, and merges all vertices to these representative vertices [16]. The method may eliminate lots of vertices in single step, but the preservation of model features is not good because it does not keep more vertices at these characteristic portions.

Luebke [13] uses vertex clustering to create vertex tree. Similarly, during dynamic walkthrough he uses active node linked list on the vertex tree. Furthermore, he proposes the ideas of distinguishing the active node linked list by space and parallel processing of selective refinement and rendering pipeline. Unfortunately, he disregards the dependency problem and so can't prevent good appearance of model because of using vertex clustering.

2.3 Dependency Problem

Although we form the hierarchical architecture (as vertex tree) using edge-collapsing or vertex-clustering according to simplification process, arbitrarily moving the active node linked list toward up or down will result in incorrect appearance of model because of existing dependency problem. Dependency results from that geometric information of lines and faces may be incorrectly combined such as under some views judging that some vertexes are visual and other adjacent vertexes are not. One kind of dependency problem is denoted as foldover in [19]. He records the information of surrounding vertexes of collapsed edge at each simplification process. During dynamic walkthrough, checking these information determines whether the edge can be collapsed or spilt or not. Hoppe [9] records surrounding triangle numbers of collapsed edge to solve dependency problem.

[7, 6, 18] solve dependency problem by following the simplification order. Each vertex and triangle is endowed with vertex level and triangle level. Following the simplification process increases the levels of influenced vertexes and triangles. Then they create a Directed Acyclic Graph (DAG) according to these levels. By the DAG, they can produce correct appearance of model during dynamic walkthrough. Another similar method to maintain dependency is [3]. He only records the maximal or minimal value of surrounding vertex numbers to judge whether simplification is legal or not. The more relative details can be found in their papers.

2.4 Selective Refinement Criteria

For the criteria of selective refinement, most common approaches include view frustum test, back face culling, screen space projection error, silhouette boundaries test, local illumination, and triangle budget. Please refer to these papers of [13, 19, 9] for details.

2.5 Review of Material-Preserving Progressive Mesh [20]

Most traditional methods for generating progressive mesh based on geometric simplifications, such as edge or triangle collapsing and vertex decimation, and some local topology modification as well. The algorithm of Yang [20] aims to produce an effective progressive mesh by (a) allowing more than three vertices to be collapsed or clustered at each level, (b) employing geometric simplification as well as topology simplification that involves local and global topology modification, and (c) using effective criteria to preserve geometric shape, especially sharp feature, and color discontinuity.

His Algorithm begins with a preprocessing, in which each vertex is classified into five categories (see 2.5.1) and evaluated to yield a weight and a priority value, then the bounding box of the given mesh is uniformly subdivided into cells of size . The algorithm then enters a simplification loop, in which each cycle yields a simplified mesh. In each cycle of mesh simplification loop, doing the following:

1. Select a vertex with the highest priority value to be the representative for the next clustering operation.

2. Create a floating cell of size τ that is centered on the representative to confirm the spatial range of vertex clustering.

3. Start at the representative and generate the spanning tree for all vertices that are inside the floating cell and can be clustered to the representative.

4. Cluster all vertices in the spanning tree to the representative. Delete the triangles that contain two or three

clustered vertices, and replace the clustered vertex by the representative for triangles that contain one clustered vertex.

5. Record the clustered vertices, vanishing triangles, and the vertex replacements.

6. Update the weights and priority values for the representative and its neighboring vertices.

The cycle is repeated until a user specified reduction rate is reached. The loop yields a sequence of meshes M^n, M^{n-1}, ..., M^0, for some n, in which M^n is the original mesh and M^n is the most simplified mesh called base mesh. The resulting progressive mesh consists of the base mesh M^n and the sequence of recorded information necessary for the refinement.

2.5.1 Vertex Categorization

His method treats that all vertices of the given mesh are classified into five categories based on the material of triangles incident to the vertex. Such a vertex categorization is different from that found in mesh decimation [17]. A vertex is a simple vertex if all triangles incident to it form a loop and are of the same material. A simple vertex is an edge vertex if visiting the loop of incident triangles encounters two material changes. A vertex is a boundary vertex if it is geometrically on the boundary of the mesh. A simple vertex is a corner vertex if visiting the loop of incident triangles encounters more than two material changes. A vertex is a non-manifold vertex if it possesses more than one loop of incident triangles. See Figure 1 for illustration.

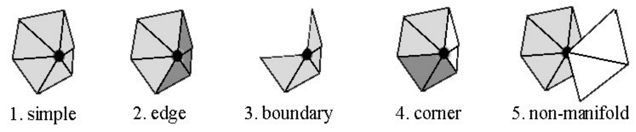

Figure 1: Vertex Categories.

3 Our Approach

Recently, most developed methods [8, 9, 12] often use edge collapsing to simplify geometric models. However, edge collapsing has the drawbacks of both slow refinement speed and a huge amount of data structure. Method of Yang [20] can solve those drawbacks, but it still has some problems. Because his approach that uses vertex clustering to simplify model clusters many vertexes once, it causes serious dependency problem. For correction of model checking all adjacent geometric elements also reduces efficiency of dynamic walkthrough. In addition, his approach is a view-independent LOD. Therefore, to accelerate rendering speed, we modify his approach into view-dependent LOD and to solve dependency problem, we propose a strategy to maintain simplified order distinguishing mutually relation of simplification operation into *replacement* and *dependency*.

We still utilize vertex-clustering method to simplify geometry model. Thus, during creating view-dependent reference structure each simplified representative vertex denotes one simplification operation, but it may be reiteration. Hence, it is not suitable for node or composite element of view-dependent reference structure unless renumbering. In view of above reasons, we give up traditional vertex tree and adopt step of simplification operation as node of hierarchical structure and dependency relation as edge to create *dependency graph*.

Then, during dynamic walkthrough, for different view different local LOD degree is equal to inserting or removing nodes from node set on the dependency graph. Therefore, we propose a real-time selective refinement scheme to reduce reiterative error criteria test and fast operate dynamic walkthrough on the dependency graph. A flow chart of our approach is shown in figure 2.

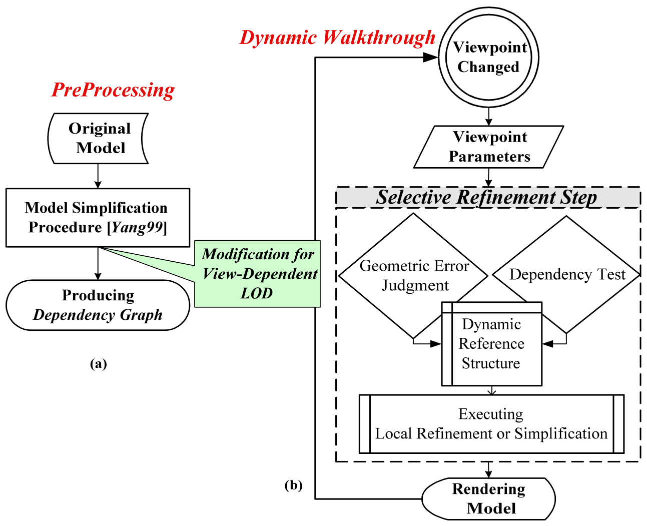
Dynamic Walkthrough
Viewpoint Changed
PreProcessing
Original Model
Model Simplification Procedure [Yang99]
Modification for View-Dependent LOD
Producing Dependency Graph
(a)
Viewpoint Parameters
Selective Refinement Step
Geometric Error Judgment
Dependency Test
Dynamic Reference Structure
Executing Local Refinement or Simplification
(b)
Rendering Model

Figure 2: Outline of System Architecture.

3.1 Modifying Simplification Approach for [20]

Original design of [20] is a technique of view-independent LOD, so we must modify his data structure to suit our approach for view-dependent LOD application. We modify two parts as following:

Maintaining Local Property for Each Simplification Step: For adjacent level of progressive model data produced by vertex collapsing, algorithm of [20] further does the merging and compression. This operation can let number of levels of progressive model data conform reasonably anticipative amount and delete repeated replacement, but lose the local property of simplification step. Therefore, we cancel the function of merging and compressing levels.

Promoting Quality of Simplification Procedure: Because his vertex-clustering algorithm can cluster many vertices once, especially between the two large amounts of simplification operations, it is hard to produce good simplification operation of vertex-clustering so that causing long and narrow triangles as figure 3a. Therefore, we forcibly execute *Global Error Bounding* and *Error Accumulation* of his algorithm to ensure that each error of vertex is not over threshold and avoid repeatedly simplifying geometric character for dense vertices.

Through above modification, clustering range of simplification operation in figure 3b is about one fourth of figure 3a.

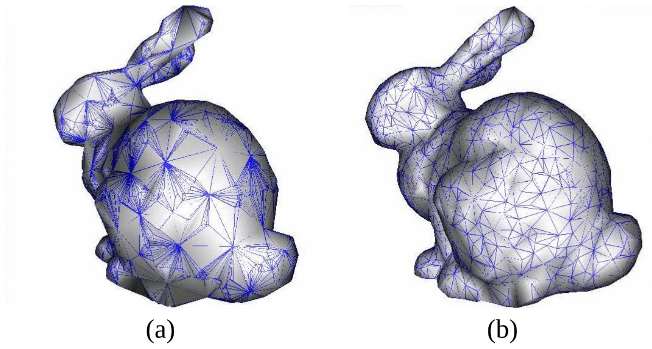

Figure 3: Bad Result vs. Good Result

Table 1: Dependency Relation in between Simplified Step

Name	Simple Symbol	Prior Order	Spatial Property
Replacing	p	$p < i$	Space of i is fully covered by p
Replaced	$R_1, R_2, R_3, \ldots, R_n$	$R_j > i$	Space of i contains range of $R_j, 1 \leq j \leq n$
Refined Dependency	$O_1, O_2, O_3, \ldots, O_n$	$O_j < i$	Boundary vertexes of simplified range O_j involve representative vertex of i and they are the partial overlap on space. $1 \leq j \leq n$
Simplified Dependency	$C_1, C_2, C_3, \ldots, C_n$	$C_j > i$	When i is simplified, representative vertex of i is the boundary vertex of simplified range and they are the partial overlap on space, $1 \leq j \leq n$

3.2 Dependency Problem Analysis

In this section, we discuss mutually influenced relation between simplification steps so called dependency problem.

Replacement and Dependency Relation: Each simplified coverage of vertex-clustering is defined: range is radiated formed by edges and simplified representative vertex is as center. Coverage of two arbitrary simplification step i, j for vertex-clustering can be classified into three conditions. 1: simplification step i, j are not intersecting each other, 2: partial overlap, and 3: i is fully covered by j. Shown in figure 4. According to the order of refinement step in which starting from the coarsest model, if number of step i is greater than number of step j, i is refined after j. In other words, j is coarsened after i. Therefore, coverage of j, namely range of vertex-cluster, is larger then i.

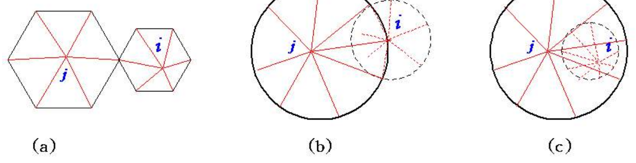

Figure 4: Coverage of Two Simplified Step i, j.

Hence, for any simplification step i, other simplification step is either fully no-intersection or any relation in table 1. Concise and to the point, relations of *Replacing* and *Replaced* are spatially full overlap. Relations of *Refined Dependency* and *Simplified Dependency* are spatially partial overlap. Therefore, Replacing and Refined Dependency are the simplification step before simplification step i. On the contrary, Replaced and Simplified Dependency are the simplification step after simplification step i.

Conditions for Executing Refinement and Simplification: To execute correctly refinement for step i must satisfy simultaneously following three conditions:

1. If projective error is over threshold, it must be refined.
2. Step *p* of Replacing Relation has been refined.
3. All step $O_1,\ldots,O_n$ of Refined Dependency have been refined.

In other words, for the correction, besides determining that step i must be refined, we need taking these steps before step i already have been refined certainly into account. On the opposite direction, we can correctly execute the simplification operation. That is to say, correctly simplification operation must be executed after all related step $R_1,\ldots, R_n$ and $C_1,\ldots, C_n$.

3.3 Dependency Graph Architecture

We propose a dependency graph whose nodes as the processes of simplification step and edge as the relations of simplification step. See the figure 5. Center of this graph is the base model that is the coarsest and nodes that represent the basic simplification step are distributed from the inside to the outside. The node of dependency graph includes geometric-refinement-operation record and error parameter produced by geometric-refinement-operation. In figure 5, we only denote the Replacing (white arrow) and Refined Dependency (black arrow). Inversing the arrow can denote the Replaced and Simplified Dependency.

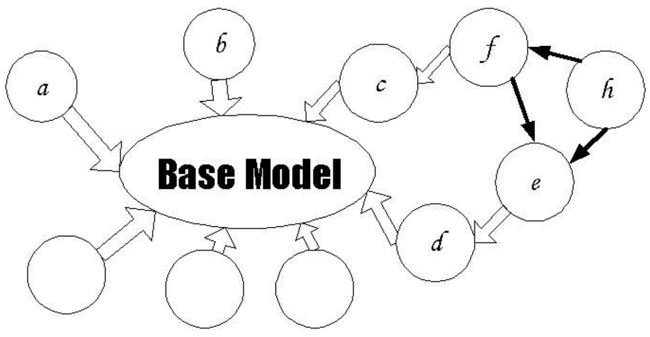

Figure 5: Dependency Graph Architecture

Sieving Dependence Relation: In fact, existing a lot of unnecessary check for dependency relation, let us see the right half-part in figure 5. $e < f < h$ (start from base model). If h depends on e and f; furthermore, f also depends on e, checking e is unnecessary when h is refined. By the refined rule, because f is refined, e must be refined before f. Therefore, we need not to check e for refining h. we can delete the edge (dependency relation) between h and e to save checking time and storing space. Table 2 shows the statistic information. By the result in table 2, we can know that general 3D model almost eliminate about half dependency relations.

Creating Dependency Graph: Creating dependency graph has two steps. First step: loading the progressive

Table 2: Sieving Result of Dependence Relation

Model (Level=Simplification Step)	Before Elimination Amount	After Elimination Amount	Eliminating Ratio (%)
Bunny(12551)	65187	33986	47.86
Ant(7219)	30993	17774	42.65

model, creating node information according to each refinement step (table 4), and getting mutual dependency relation between refinement steps. Second step: Sieving the dependency relation and arranging the spaces used.

We maintain a *VertexFan* table (table 3) to record that vertex is belong to which influence range of refinement step for each vertex of model during preprocessing. Processing order is from base model (coarsest) whose assigned number is 0 to the most refinement.

Table 3: Recording Replacement Relation Structure for Vertex and Adjacent Mesh

```
Typedef struct
{ int LargestRefine;     /* Setting to number of refined step which finally
                            splits or influences this vertex */
 LIST *FaceList;         /* Surrounding triangle number */
}VertexFan;
```

In table 4, we first find the representative vertex V_{rep} and then get *LargestRefine* of V_{rep} from *VertexFan* table. *LargestRefine* of V_{rep} is unique *Replacing (p)* relation of step i and dependency relation as *Refined Dependency* O_1, O_2, ..., O_n is gained by first searching boundary vertex V_{n1}, V_{n2}... of refinement step i from *FaceList* of V_{rep} and then getting *LargestRefine* of V_{n1}, V_{n2}....

Table 4: Algorithm for Creating Node Information of Refined Step

```
1   Initialize and loading PROGRESSIVE_MESH
2   From BaseModel to create initial VertexFan Table;
3   For each level {
4      Find representative vertex $V_{rep}$;
5      Unique Replacing = $V_{rep}$.LargestRefine;
6      From $V_{rep}$.LargestRefine to find boundary vertex $V_{n1}$,
       $V_{n2}$,... of influence range;
7      For each $V_{nj}$
8         { Refined Dependency $O_j$ = $V_{nj}$.LargestRefine; }
9      Computing geometric error parameter
10       /* Link to REPALCEMENT structure in [20] */
         Link REPLACEMENT structure;
11       Execute this refined step and update the VertexFan Table;
12   }
```

4 Dynamic Operation Scheme and Error Measurement

In this section, we show how to real time produce model according to view from dependency graph during dynamic walkthrough. Referring to right half of Figure 2 that shows the flowchart of dynamic operation scheme.

4.1 Selective Refinement Operation

Node of dependency graph doesn't only has dependency information relative to other nodes, but has a status flag to denote that current status of refinement step is refined or coarse on this node. All nodes of coarsest model are at coarse status, contrariwise. Therefore, when the view is changed, the status of node also is changed (figure 6). Besides, change of node status is according to two checks: **Geometric Error Criteria** and **Dependency Check** (Table 5).

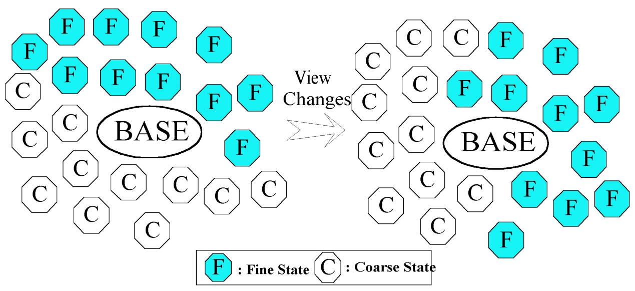

Figure 6: Change of Status for View Changed on Dependency Graph

Table 5: Change Criterion for Node Status

		Geometric Error Check	
		Under threshold	**Over threshold**
Dependence Check	**No Qualified**	*Wait?*	*Forced Refined*
	Qualified	*Simplified*	*Refined*

In table 5, if dependency check is qualified, node status is simplified or refined according to Geometric Error Check. If check result is over threshold and no qualified, node status must be forcibly refined including Replacing relation node *p* and Refined Dependency nodes O_1,... O_n for the correct appearance of model. If check result is under threshold and no qualified, we don't operate this node because other node probably forcibly refines it and check it at next time.

4.1.1 Finite State Graph of Node Status

After above specification, changing the node status needs two times for checking. To do it must waste much time, so we add two transitional statuses: **Refinement_Ready** and **Simplification_Ready** to reduce checking time (figure 7). Refinement_Ready denotes that coarse node passing dependency check can be refined whenever needed, and so is Simplification_Ready on the same way. In figure 7, node at the tail of yellow arrow actively checks geometric error and dependency. If checking is passing, node transforms original status into another status that at the head of arrow. However, status transform for blue arrow is passive. That is, status is modified only when other relative node status is changed. When node i was refined, the modify algorithm for modifying other relative node status is showed in table 6 and Figure 8 shows how did we operate relative nodes of node i. If status of node i is RR, Replaced Relation R_1, R_2,...,R_n and Simplified Dependency C_1, C_2,...,C_n must be in C ,and Replacing Relation

p and Refined Dependency O_1, O_2,...,O_n must be in SR. when node i was transferred from RR to SR, p and O_1, O_2,...,O_n directly transfer to FS, but R_1, R_2,...,R_n and C_1, C_2,...,C_n need to do dependency test first and then transfer status of nodes passing the test from C to RR. Because p and O_1, O_2,...,O_n needn't do dependency test and only R_1, R_2,...,R_n and C_1, C_2,...,C_n need to do, we can greatly reduce time of dependency test. The Modify algorithm of simplifying node i is the same as Modify algorithm of refining node i, but on the opposite direction.

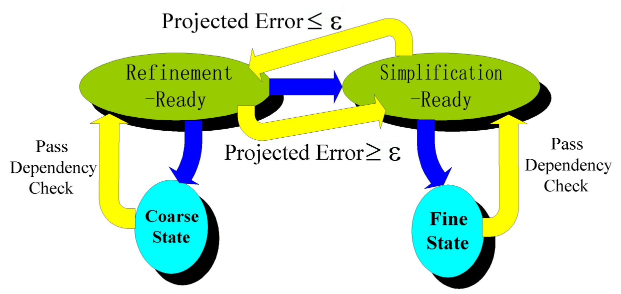

Figure 7: Finite State Graph for Changing Node Status

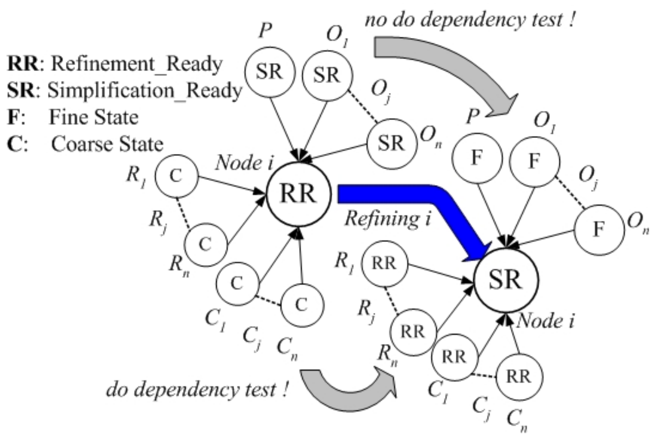

Figure 8: Refining node i from RR to SR

Table 6: Modify Algorithm of Refining Node i

```
Dependency_Testing_of_Refinement_Ready( Node i )
{ For each p and $O_j$ of i /* p: Replaced Relation */
                        /* $O_j$: Refined Dependence Relation */
  If p and $O_j$ = Fineness /* True: Passing Dependency Test */
    then i.qualify = True;
    Else i.qualify = False; }

/* Refinement Ready to Simplification Ready */
Modified_Relative_Node_State( Node j )
{ If Operation-Step = Replacing or Refined-Dependence-Step
    If j.status = Simplification_Ready
      Then j.status = Fineness;
  Else
  If Operation-Step = Replaced or Simplified-Dependence-Step
    If j.qualify = True
      Then j.status = Refinement_Ready; }
```

4.2 Dynamic Operation Scheme

After we add two transitional states, we can easily and fast operate dynamic walkthrough according to view. Referring the figure 9, we maintain two linked lists: **Refine-Candidates** and **Coarsen-Candidates**. Refine-Candidates list is composed of nodes in Refinement_Ready state and Coarsen-Candidates list is composed of nodes in Simplification_Ready state. Refine-Candidates and Coarsen-Candidates form a boundary on the dependency graph. Nodes outside of the boundary are all in *Coarse State*; Nodes inside of the boundary are all in *Fine State*. Therefore, we only modify this two lists, the model can be correctly changed.

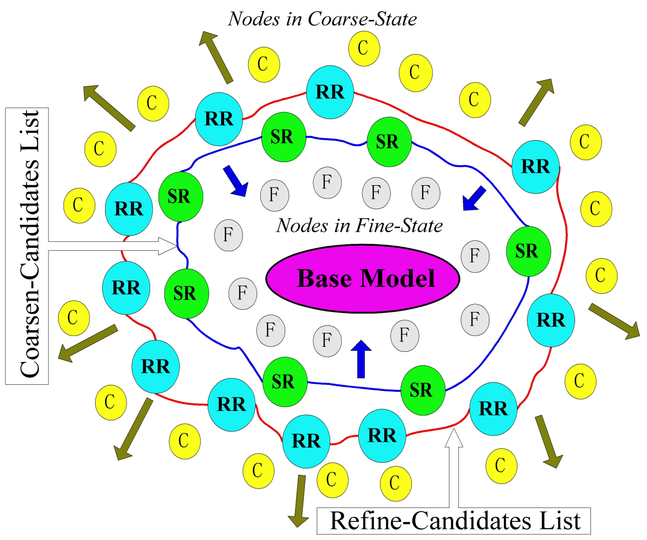

Figure 9: Dynamic Operation Scheme

Overall dynamic walkthrough-operation algorithms are showed in table 8,9, 10, and 11. First, we define some symbols in table 7 to denote geometry-test result of node. In order to avoid repeatedly testing geometry error for the same node during walkthrough operation, we use the Geometry-result array as a cache to store geometric error testing result. Main procedure of dynamic walkthrough operation shown in table 8 sequentially operates three steps: 1.clean Geometry-Result array, 2.modify the Refine-Candidates list, and 3.modify the Coarsen-Candidates list. It modifies mode shape according view changed.

Then, let we explain how to modify Refine-Candidates list (RC List) during walkthrough. See the figure 10 and refer the table 9, we first test geometry error for each node in RC list (row 1) and then if test-result of node i in figure 10.a denotes that it need be refined, that is, transforming status of node i from RR to SR, we delete it from RC list, then add into CC list, and split it (row 2-5). Because node i was refined, we need to modify relative node p and O_j of node i for its dependency. So, statuses of node p and O_j are transformed into F (row 6-7). After above operation, we test geometry error for each relative node R_j, C_j, and C_k of i. By the geometry error test R_j and C_j need be refined but C_k needn't, so R_j and C_j are transformed to RR and pushed into RC list (row 11). Result of above steps is shown in figure 10.b.

See the figure 10.c, because R_j and C_j have been pushed into RC list, for the correct model shape we need to forcibly refine node P_{cj} and O_{cj} of node

C_j no matter its dependency (row 12). Function of Force_Eliminate_Dependency is shown in table 10. We let geometry error of its p as P_{cj} or O_j as O_{cj} equal the GEOTEST_FORECE (row 3), and then if p or O_j is in Coarse State, we recursively operate it to solve dependency problem (row 5). If it is in RR, we split it and push into CC list (row 7-12). If it is already in SR list or Fine State, which exactly is our want, we don't care it (row 13-14). On the contrary, in table 9, if test-result is GEOTEST_FOFF, namely in coarse state, we only test geometry error for node C_j of i. If node C_j need be refined, it also is added into Refine-Candidates (row 15-21). The final result of modifying node i in Refine-Candidates list is drawn in figure 10.d.

In table 11, operating Coarsen-Candidates list is roughly similar to Operate Refine-Candidates Algorithm, but operating the former needn't forcibly coarsen. When geometry error is under the threshold, we coarsen the node i and push its p and O_j passing dependency test into Coarsen-Candidates list. Therefore, by these algorithms, we can easily and real time maintain correct model during dynamic walkthrough.

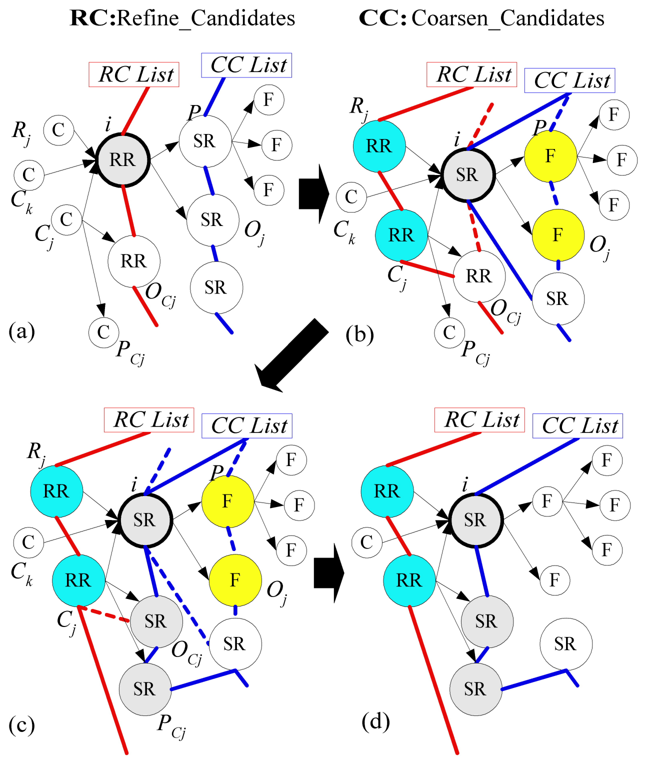

Figure 10: Dynamic Operation Scheme

4.3 Selective Refinement Criteria and Decision

In this paper, we adopt three measurements: 1.outside the view frustum? 2.at back face? 3. projection error is greater than tolerance? to determine whether the model has to be refined. Many techniques about these have been proposed. Especially, Hoppe's method [9] can show a good efficiency, so we adopt his method to implement selective refinement criteria. At here, we only explain the detail of projection error measurement during our implementation. As for other detail can be found in [9].

Table 7: Symbol Definition

LIST Refine-Candidates;	
LIST Coarsen-Candidates;	
ARRAY	
Geometry-result[number_of_node];	/* store geometric error testing result of node under current view.*/
define GEOTEST_TODO 0	/* no test yet */
define GEOTEST_FOFF 8	/* testing result: must at coarse state*/
define GEOTEST_BASE 16	/* testing result: must at fine state*/
define GEOTEST_FORCE 32	/* must be refined because of dependency of child nodes*/

Table 8: walkthrough Operation: **Main( )** Algorithm

```
/* **Initialization:**
Coarsen-Candidates list = NULL;
Refinement-Candidates list = Replacing Relation $p$ of nodes
is base model; */

**Previous step:** *View Changed*
For i = 0 to size of Geometry-Result[ ] do
    /* reset node testing result*/
    Geometry-Result[ i ] = GEOTEST_TODO;
For i = 0 to size of Refine-Candidates list do
    /* modify Refine-Candidates list*/
    Call Operate_Refine_Candidates(i);
    /* modify Coarsen-Candidates list*/
For i = 0 to size of Coarsen-Candidates list do
    Call Operate_Coarsen_Candidates(i);
**Next step:** *Rendering*
```

See figure 11 and figure 12, Error is the distance between V_{rep} and V_c and E_{max} is the maximal plane error. We utilize projection error equation in [9] to obtain E_{max} as equation 1.

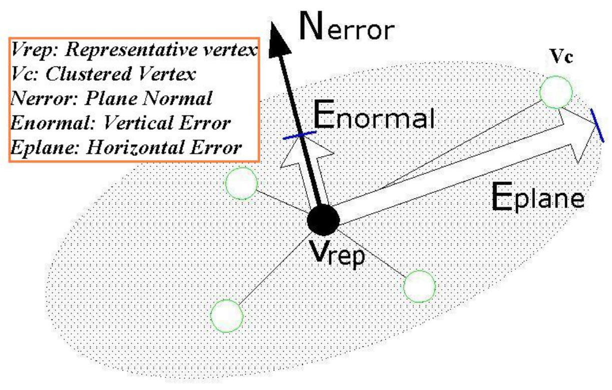

Figure 11: Error Distance of E_{plane} and E_{normal} on Space Using Vertex-Clustering

$$E_{max}^2 = \quad 4.0 * \tan^2 \frac{\alpha}{2} * \tau^2 * ||V_{rep} - V_{eye}||^2 \qquad (1)$$

Table 9: Operate_Refine_Candidates(Node i) Algorithm

```
Call Geometry_Error_Test( i );
If testing-result = GEOTEST_BASE or GEOTEST_FORCE
/*must at fine state*/
{ i.status = Simplification_Ready;
Delete i from Refine-Candidates and
            add i into Coarsen-Candidates;
Call Execute_Refine_Operation( i ) to change model geometry;
Call Dependency_Testing_of_Refinement_Ready( i );
Call Dependency_Testing_of_Simplification_Ready( i );

For each child of i do
{ Call Geometry_Error_Test( child );
  If ( testing-result = GEOTEST_BASE or GEOTEST_FORCE)
    { add child into Refine-Candidates;
      Call Force_Cancel_Dependence_Problem( child ); }
  Else
  { Call Dependency_Testing_of_Refinement_Ready( child );
    If child.qualify = True then
      add child into Refine-Candidates; }
}Else
If( testing-result = GEOTEST_FOFF /*must at coarse state*/
{ for each Simplified Dependency $C_j$ of i do
  { Call Geometry_Error_Test( $C_j$ );
    If( testing-result = GEOTEST_BASE or GEOTEST_FORCE)
    { add $C_j$ into Refine-Candidates;
      Call Force_Cancel_Dependence_Problem( $C_j$ );}
  }
}
```

Table 10: Force_Cancel_Dependence_Problem(Node i) Algorithm

```
For each p and $O_j$ of i
{ k = p or $O_j$;
/*forced refinement*/
  Geometry-Result[k] = GEOTEST_FORECE;
  Switch( k.status )
  {   Case Coarse_State:
        Call Force_Cancel_Dependence_Problem(k);
            break;
      Case Refinement_Ready:
        k.status = Simplification_Ready;
        Executing Geometric_Split_Operation(k);
        Add k into Coarsen-Candidates;
        Call Dependency_Testing_of_Refinement_Ready(k);
        Call Dependency_Testing_of_Simplification_Ready(k);
            break;
      Case Simplification_Ready:
      Case Refinement_Ready: break; /* don't care */
  }
}
```

Thus, comparing E_{max} with E_{plane} and E_{normal} is the selective refinement criteria. Only if any of E_{plane} and E_{normal} is greater than E_{max}, the node has to be refined. In other words, we refine the node if the equation 2 is true.

$$\begin{array}{ll} E_{max}^2 * ||V_{rep} - V_{eye}||^2 & < E_{plane}^2 * ||V_{rep} - V_{eye}||^2 \cos^2\gamma \\ \text{or} & \\ E_{max}^2 * ||V_{rep} - V_{eye}||^2 & < E_{normal}^2 * ||V_{rep} - V_{eye}||^2 \sin^2\gamma \end{array} \tag{2}$$

In order to accelerate calculated efficiency, we use equation 3 to reduce computing time.

Table 11: Operate_Coarsen_Candidates(Node i) Algorithm

```
Call Geometry_Error_Test(i);
If testing-result = GEOTEST_FOFF
{ i.status = Refinement_Ready;
  Delete i from Coarsen-Candidates
    add i into Refine-Candidates;
Execute_Coarse_Operation(i)
    to change the model geometry;
  Call Dependency_Testing_of_Simplification_Ready(i);
  For each $R_j$ and $C_j$ of i do
  { k = $R_j$ or $C_j$;
    Call Dependency_Testing_of_Coarsen_Ready(k);
    If k.qualify = True
    then add k into Coarsen-Candidates; }
}else if testing-result != GEOTEST_FOFF
{ /* don't care */ }
```

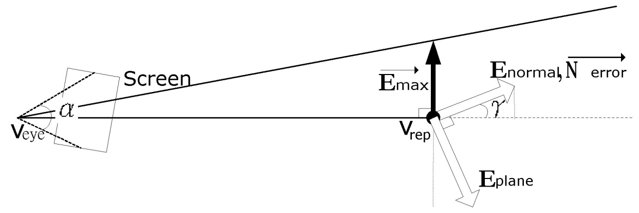

Figure 12: Calculate E_{max}

$$\begin{array}{rl} ||V_{rep} - V_{eye}|| \cos\gamma = & (V_{rep} - V_{eye}) \cdot (\vec{N_{error}}) \\ ||V_{rep} - V_{eye}||^2 \sin^2\gamma = & ||V_{rep} - V_{eye}||^2 - \\ & ||V_{rep} - V_{eye}||^2 \cos^2\gamma \end{array} \tag{3}$$

By the equation 1, 2, and 3, we can fully avoid to evaluate division, cos^{-1}, and tan^{-1} etc. Therefore, walkthrough efficiency can be greatly increased.

5 Experimental Results

We have implemented our approach using C language and OpenGL library. We tested the models on SGI Octane MXI R10000 workstation with 256MB RAM and Pentium III PC with 550MHz CPU and 512MB RAM. SGI Octane has hardware to accelerate rendering, but Pentium III PC is not.

Experimental Symbol Definition:

R : Diameter of Sphere ringing the object model.

Run Time : Selective Refinement Time + Rendering Time ScreenError

Tolerance τ : $\sqrt{32/ImageSize(512x512)}$

We adopt a typical vertex tree using edge collapsing to compare with our method. For model simplification of comparative approach, which is like in [19], we utilize a more important vertex between two vertices as representative vertex to merge, so it doesn't produce new vertex after edge collapsing. Creating binary vertex tree starting from the coarsest model uses a way of Top-Down as in [9] and operating dependency problem as in [18] adopts state number of vertex and triangle to solve it.

Model information is shown in table 12. We use different experimental parameter to test our method: Approach1 (B) with large clustering range and Approach2 (C) with small clustering range, so approach1 produces more small simplified steps as nodes. Direction of dynamic walk-through is from 1R to 11R that is the distance between viewpoint and model and the speed of movement rate is 0.001R. Thus, Total is 10000 frames. See the testing result in table 13 for detail.

Table 12: Model Data

Original Model	**Vertices**	**Triangles**	**R**
Bunny	34834	69453	125.1

Table 13: Testing Result

Item	**Comparison (A)**		**Approach1 (B)**		**Approach2 (C)**	
Nodes:						
Simplified Steps		34831(depth:26)		12678		17040
Triangles:						
1R		16173		21589		15041
Average		2841		6749		4905
Operating Nodes:						
(Selective Refinement)						
1R		26703		10388		8253
1.001R		4473		256		230
11R		288		95		67
Average		1303		290		247
(Average Time: ms)	**Octane**	**Pentium**	**Octane**	**Pentium**	**Octane**	**Pentium**
Selective Refinement	66.85	29.49	27.64	13.53	26.38	12.54
Rendering Time	164.59	654.57	222.60	774.15	149.24	603.87
Total Time	231.44	684.06	250.24	787.68	175.76	616.41

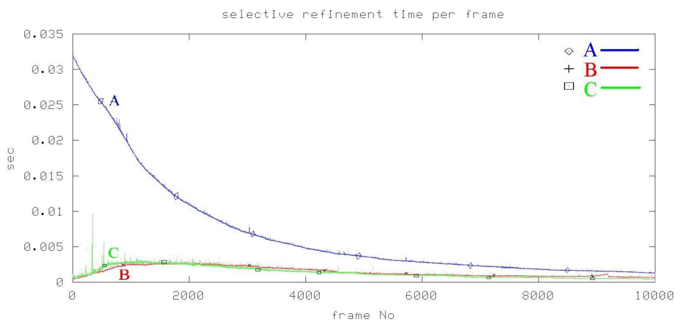

Figure 13: Selective Refinement Time Per-Frame (Octane)

In table 13, we can find that our method operates the nodes far less than comparison during selective refinement and the average time also is. However, amount of triangles of B and C is always more than A. See the figure 16, we investigate the problem to find that ours still maintains a fine shape at 11R, but A already can't. Because Octane has hardware to accelerate rendering, its rendering time is faster than Pentium. Following, diagrams of curve are shown in figure 13, 14, and 15. The figure 18 shows the model shape at different distance (R).

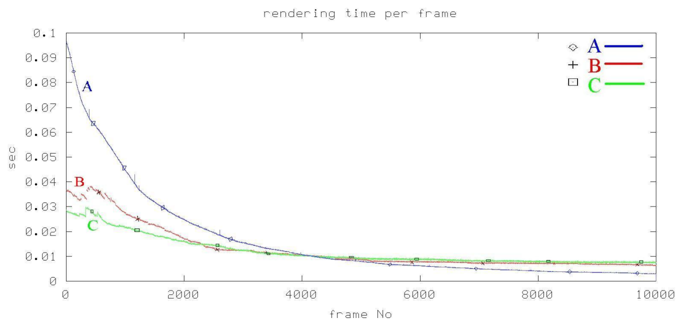

Figure 14: Rendering Time Per-Frame (Octane)

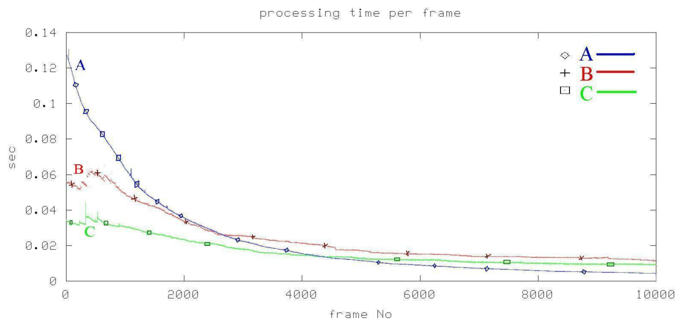

Figure 15: Total Time Per-Frame (Octane)

For amount of stored data, our approach only needs about 3Mbytes, but comparative approach needs about 5Mbytes. Although architecture of binary vertex tree is simpler than ours, it produces huge amount of node. So this confirms that our dependency graph can reduce amount of memory.

In addition, we adopt **SNR (Signal Noise Rate)** criteria as equation 4 to compare the quality of rendering image with the original model image.

$$SNR = \frac{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\hat{f}(x,y)^2}{\sum_{x=0}^{M-1}\sum_{y=0}^{N-1}\left[\hat{f}(x,y)-f(x,y)\right]} \qquad (4)$$

Form the figure 17, we can know that comparative approach has more small triangles than ours, but its quality is rather less than ours. Ours always keeps better quality from start to final.

Finally, in figure 19, we using coloring Easter model to show our material preserving capability.

Discussion From the above experimental result, efficiency of selective refinement for dependency graph is better than binary vertex tree. We can attribute the fine result to that operating nodes are greatly reduced. However, we also find that amount of triangles and operating nodes suddenly rises at some time. We can explain that this phenomenon is due to that splitting a node with a large number of dependency results in that many nodes must be spilt.

We think that large clustering range needs to split more neighboring triangles and vertices to maintain correct appearance. So, in order to refine a vertex, it has to forcedly split other unnecessary refined vertices. Therefore, resulting in producing many triangles. This is why B's amount

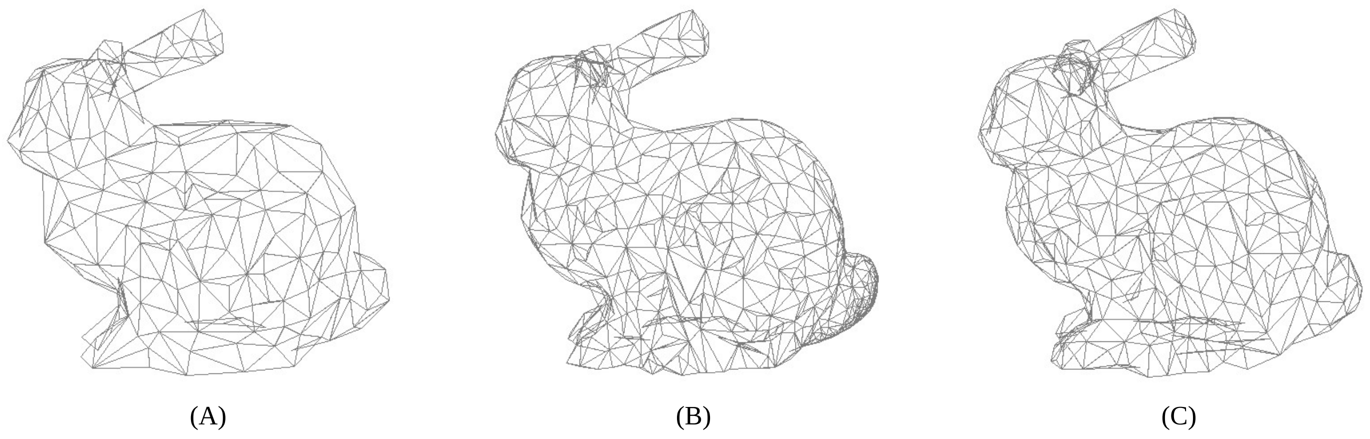

Figure 16: Rendering Image at 11R

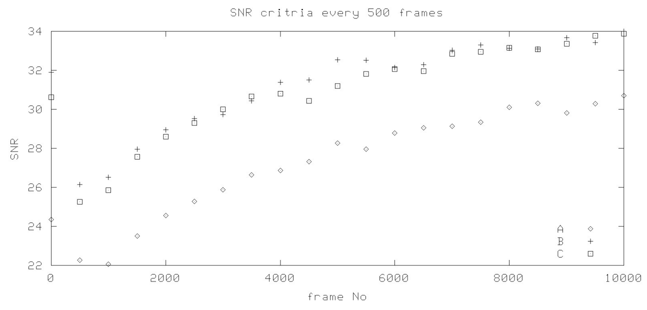

Figure 17: SNR Criteria Every 500 Frame

of triangle is more than C and A.

Moreover, edge collapsing can produces less triangles than ours. Because it uses binary tree to dynamic choose vertex and reform triangle by state number, it can efficient reduce amount of triangles and exceed our dependency graph between medium and far distance R. However, reforming triangle is more serious for destroying shape of model. Therefore, its capability for preserving appearance can't compare with our dependency graph.

6 Conclusions and Future Works

In this paper, we propose a view-dependent LOD modeling base on a view-independent LOD scheme that uses vertex-clustering algorithm with geometry simplification and material preservation [20]. We find the dependency out from simplified step and neighboring geometric-feature to create dependency graph preventing dependency problem. We also develop a dynamic walkthrough scheme to support fast view-dependent selective refinement operation.

After the implementation and experiment, proposed scheme can greatly reduce the checking nodes to accelerate selective refinement operation. However, because a large refining range produces more many triangles, it slightly lowers the efficiency of rendering time. Therefore, global efficiency probably is a little less than edge collapsing between medium and far distance, but our approach can provide stable efficiency from near to medium distance. In addition, no matter the distance how far from viewpoint to model, proposed method still preserves the good appearance of model. On the contrary, edge collapsing can't do this.

There are a number of areas for future work, including:

Reducing amount of triangles: By the experimental result, our dependency graph produces triangles greater than edge-collapsing approach. Thus, we have to investigate how to modify our dependency graph to reduce amount of triangles.

Accelerating rendering efficiency: Because our approach doesn't achieve good efficiency during rendering phase, we will utilize Triangle Strip [4] technique to accelerate rendering speed.

References

[1] M.E. Algorri and F. Schmitt. ”mesh simplification. *Computer Graphics Forum*, 15(3):77–86, August 1996.

[2] M. Eck, T. DeRose, T. Duchamp, H. Hoppe, M. Lounsbery, and W. Stuetzle. Multiresolution analysis of arbitrary meshs. *In Proceedings of ACM Siggraph'95*, pages 173–182, August 1995.

[3] J. El-Sana and A. Varshney. Generalized view-dependent simplification. *Computer Graphics Forum*, 18(3):83–94, 1999.

[4] F. Evans, S. Skiena, and A. Varshney. Optimizing triangle strips for fast rendering. *In IEEE Visualization '96*, pages 319–326, October 1996.

[5] M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. *In Proceedings of ACM Siggraph'97*, pages 209–216, August 1997.

[6] A. Gu'eziec, G. Taubin, and B. Horn. A framework for streaming geometry in vrml. *IEEE Computer Graphics and Applications*, 19(2):68–78, April 1999.

[7] A. Gu'eziec, G. Taubin, F. Lazarus, and W. Horn. Simplicial maps for progressive transmission of

polygonal surfaces. *In Proceeding ACM VRML98*, pages 25–31, 1998.

[8] H. Hoppe. Progressive meshes. *In Proceedings of ACM Siggraph'96*, pages 99–108, 1996.

[9] H. Hoppe. View-dependent refinement of progressive meshes. *In Proceedings of ACM Siggraph'97*, pages 189–198, August 1997.

[10] H. Hoppe. Smooth view-dependent level-of-detail control and its application to terrain rendering. *In IEEE Visualization'98*, pages 35–42, October 1998.

[11] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *In Proceedings of ACM Siggraph'93*, pages 19–26, August 1993.

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

[13] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. *In Proceedings of ACM Siggraph'97*, pages 199–208, 1997.

[14] J. Popovicacute and H. Hoppe. Progressive simplicial complexes. *In Proceedings of ACM Siggraph'97*, pages 217–224, August 1997.

[15] R. Ronfard and J. Rossignac. Full-range approximation of triangulated polyhedra. *In Proceedings of EUROGRAPHICS'96*, pages 67–76, 1996.

[16] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. *Modeling in Computer Graphics: Methods and Applications*, pages 455–465, June 1993.

[17] W. Schroeder, J. Zarge, and W. Lorensen. Decimation of triangle meshes. *Computer Graphics*, 26(2):65–70, July 1992.

[18] D. To, R. Lau, and M. Green. A method for progressive and selective transmission of multi-resolution models. *ACM Virtual Reality Software and Technology*, pages 88–95, 1999.

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

[20] S.-K. Yang and J.-H. Chuang. Discontinuity material-preserving progressive mesh using vertex-collapsing simplification. *Submitted fo publication*.

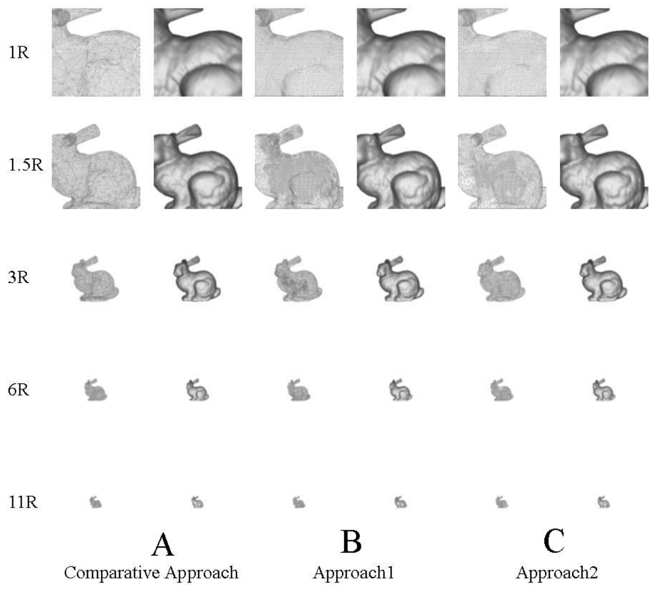

Figure 18: Model Shape at Different Distance (R)

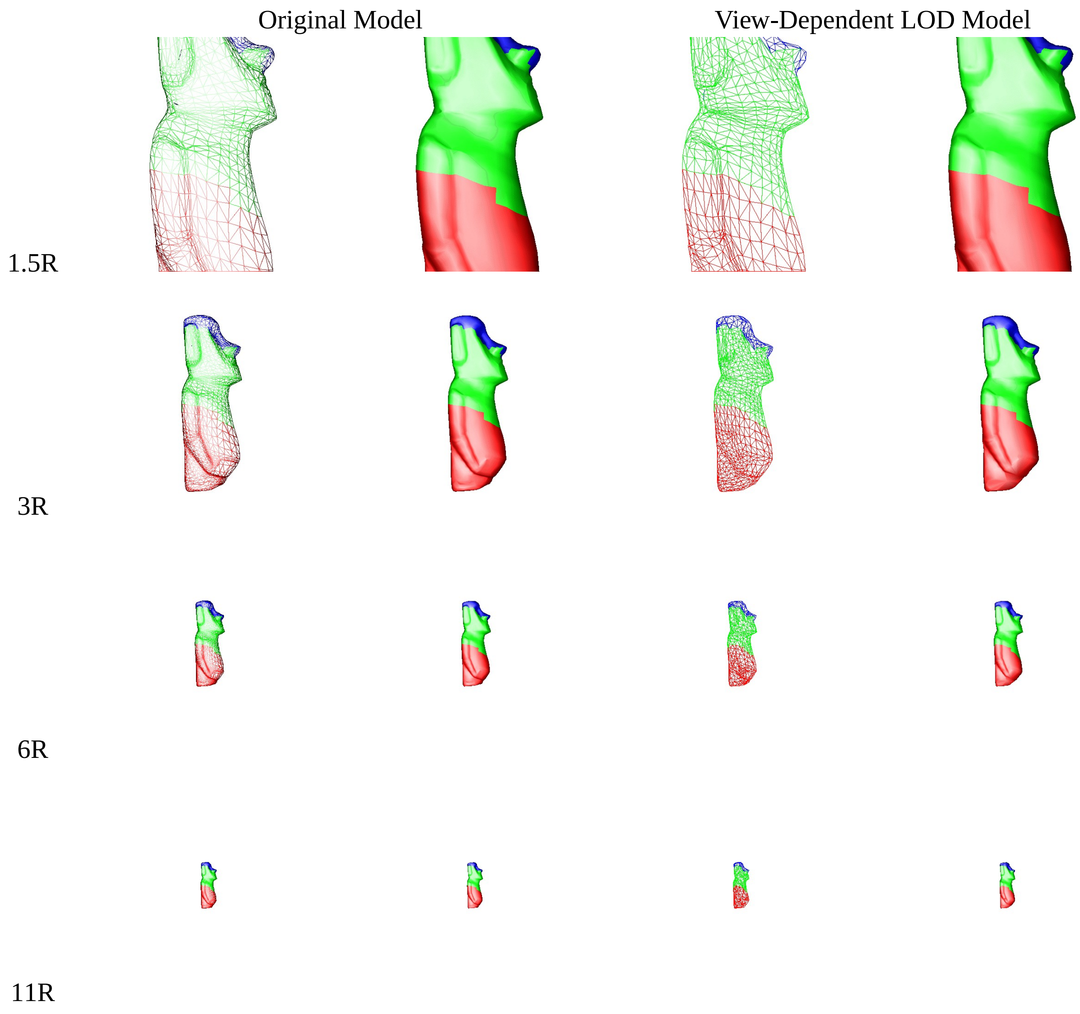

Figure 19: Material Preserving Effect