1998 International Computer Symposium
Workshop on Computer Graphics and Virtual Reality
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

CLUSTERING BASED HIERARCHICAL LEVEL-OF-DETAIL WITH BOUNDED ERROR

Kuo-Chou Tseng and Chin-Ho Cheng

Department of Computer Science and Information Engineering
Fu Jen University, Hsinchuang, Taipei 24205, Taiwan, R.O.C.
Email:{alf85, chcheng}@csie.fju.edu.tw

ABSTRACT

With the fast development of virtual reality (VR), many people notice that it will add the reality to the VR system if we can use different levels of detail (LOD) of 3D models to implement the VR system. In the field of 3D computer graphics, we usually use polygons to compose a 3D model. The more polygons we could use, the more detail we could see. The aim of LOD generation is to retain the important visual characteristics of the original object, and generate a whole series of simplifications. In this paper, we will propose a new algorithm to solve this LOD problem. It is clustering-based and uses pair-wise mesh merging techniques to generate new meshes. During the merging process, we still consider some clustering constraints in order to preserve geometry primitives. This algorithm is well proved and tested. It is an efficient and fast algorithm, and has good reduction rate.

Keywords: computer graphics, level of detail, simplification, polygon, mesh

1. INTRODUCTION

As soon as 1976, Clark suggested to use simpler versions of the geometry for objects that had lesser visual importance, such as those who would be far away from the viewer [1]. These simplifications are called Level-of-Detail (LOD). The aim of LOD is not to remove the geometry primitives and reduce the number of polygons in a 3D model. Additionally, we still need to consider the trade-off between frame rate and visual effect, since the 3D model with less polygons would have poor visual effect. In order to reach this object, many researchers devoted themselves to this field, and proposed many solutions to this problem. Most of these algorithms tried to simplify an input model and get a model, which had the fewer triangles. Additionally, these algorithms also preserved geometry primitives, preventing from over-simplifying. In this paper, we will propose a new and efficient algorithm to solve this LOD problem. It makes use of the technique of pair-wise merging between mesh and mesh to generate new and larger meshes. (A mesh is a 3D surface patch with closed boundary edges, and this surface patch may have holes inside. For a 3D model, it may be composed of many meshes.) Then we can straighten the boundary edges of these merged meshes to get meshes with fewer boundary edges. After boundary straightening process, we re-triangulate these meshes to get a simplified 3D model. We repeat these three steps, and use well-designed data structures to compactly store the simplified model during each iteration.

Our paper is organized as follows. In Section 2, we survey the related work. In Section 3, we describe our algorithm overview. In Section 4, we present the clustering methods. In Section 5, the boundary straightening technique is explained. In Section 6, the polygonal triangulation is discussed. In Section 7, experimental results are illustrated. Finally, conclusions are given in Section 8.

2. RELATED WORK

In 1993, Hinker and Hansen [3] used face-merging tricks and considered the angle beween two faces. If the angle is inside their predefined angle tolerance, then merge these two faces. It will create coplanar polygon sets and re-triangulate these coplanar polygon sets to get a simplified 3D model. However, it assumes that degenerated polygons are not created during the face-merging phase. It therefore does not check for them. While Kelvin and Taylor [5] also took a similar strategy, but he noticed this problem. The greatest affinity is that they generate meshes by grouping near coplanar triangles during each iteration, while we generate our meshes by grouping near coplanar meshes, not triangles, during each iteration. There are some well-collected WWW homepages, which offer rich information about LOD [10, 11, 12].

3. ALGORITHM OVERVIEW

Our algorithm is based on the idea of clustering techniques. At each iteration, we will take those meshes generated at the previous iteration as input data. The reason why we do so is that we can make it converge more quickly, especially as the number of triangles becomes large. We think that it is more efficient to use this strategy; and besides, there are many triangles, which can be merged into coplanar sets in most 3D models. Many LOD algorithms must re-triangulate all mesh sets at each iteration, and take these triangles as input data, which will be processed at next iteration. We think we do not have to do so. Therefore, we generate new meshes by merging previously merged meshes. Then we perform boundary straightening. In this phase, we will remove edges that are co-linear in order to reduce the number of triangles after re-triangulation. Finally, we will perform triangulation for all processed meshes. These three phases: clustering, boundary straightening and triangulation are performed at each iteration. Note that, we must organize these meshes in the form of tree structure during each iteration. This tree structure is called *level tree*. Each node in

the level tree represents a mesh. (See Fig.1.)

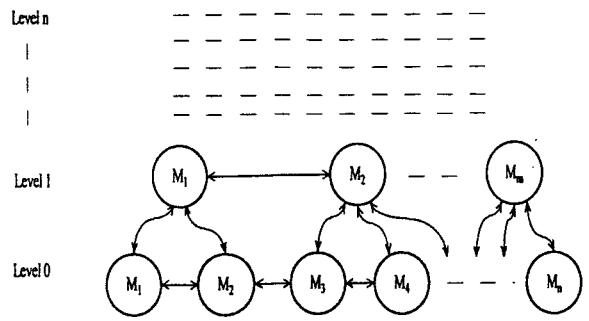

Fig. 1 The level tree structure

3.1 The Construction of Level Trees

During the simplification process, we will record each mesh's left and right siblings as well as its parent and children. A mesh is said to be a parent node in a level tree if merging its children nodes generates it. The reason we store this information is that we can easily add new functions to our algorithm. Once we find a neighbor mesh, which satisfies the merging constraints, then we can add this mesh to our level tree. The way we use to construct the level tree is bottom-up(see Fig.2). The lower level is, the more detailed model is. The higher level is, the less detailed model is. The advantage of constructing the level tree is we can pick an arbitrary tree node and traverse along tree link to find all neighbor meshes. Additionally, we still can perform local expansion during the traversal.

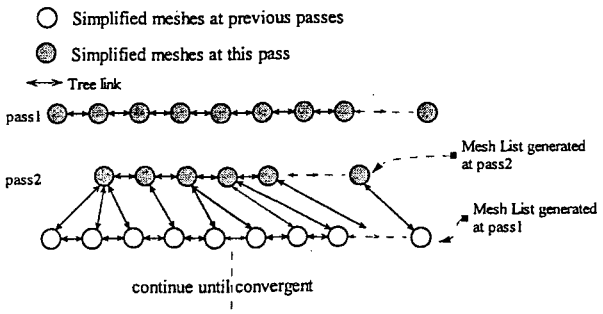

Fig.2 Construction of a level tree

There are three main phases in our algorithm. They are clustering, boundary straightening and triangulation. These three phases will be illustrated in Sections 4, 5, and 6, respectively.

4. CLUSTERING

Before each clustering process, we first randomly pick a mesh as a seed mesh. Then in all neighbor meshes of the seed mesh, we try to find candidate meshes, which satisfy all clustering constraints. If the angle between the seed mesh's normal and picked neighbor mesh's one is inside the error tolerance of the current level, then we can merge the seed mesh with this neighbor mesh. As the level grows, the error tolerance also increases. Therefore, the reduction rate of simplification will increase. The error tolerance space will form a cone. Each time when we finish pair-wise merging, we must retrace the new boundary edges. Assume that two meshes M_1 and M_2 will be merged, and the numbers of boundary edges are N_1 and N_2, respectively. We find that each of the new boundary edges will be visited once, while others will be visited twice. We remove edges which are visited twice, and retrace boundary edges. Therefore, it takes $O(N_1+N_2)$ time to retrace boundary edges. That is linear time. The advantages of this method are that it is simple, fast as well as it can find all holes in the mesh. After finding all polygons, which compose this mesh, the polygon with longest perimeter is the boundary edges, and others are holes. Compared with the binary search method proposed by Hinker and Hansen [3], our method is better. Since the binary search method spends $O(NlgN)$ time to sort all edges first, then it takes $O(NlgN)$ time to find all polygons.

5. BORDER STRAIGHTENING

After completing clustering phase, we will proceed to do border straightening. The purpose of border straightening is that it greatly simplifies the input model by straightening boundary. Many strategies can be adopted. For example, we can use the error distance strategy mentioned by Kelvin and Taylor [5] to measure whether a vertex can be discarded or preserved. From this viewpoint, two methods can be used, one of which is called 'maximum edge merging', and the other is an aggressive method. For the maximum edge merging strategy, once we find a common segment(polyline), we first connect the head and tail vertices of this segment. However, this will cause over-simplifying; therefore, we should do edge-split to avoid this problem. For the aggressive strategy, if some vertex's error distance is greater than the threshold, then we will subdivide this reduced segment. The cost of this aggressive strategy is more expensive than maximum edge merging strategy, but the simplified result is better than maximum edge merging strategy is. In this paper, we calculate the angle between two consecutive edges. If the angle is greater than our predefined angle tolerance, then we preserve the common vertex of these two consecutive edges. Otherwise, we discard it. With the increasing of level, the angle tolerance also increases. As a result, the reduction rate increases.

6. TRIANGULATION

A triangulation is a partition of an arbitrary polygon into many triangles. Planar triangulation is an important topic in computational geometry [8, 9]. Many algorithms have been proposed to solve this problem [7, 9]. For LOD applications, we always find that there are many polygons with holes, and have to triangulate those polygons. But most of these triangulation algorithms are unable to solve this problem for those polygons with holes. If we want to use such kinds of algorithms to solve the triangulation problem, we must first decompose the polygon with holes into several pieces of polygons, which have no holes in them. Then, apply any of these triangulation algorithms for those output polygons with no holes. For example, the Kelvin and Taylor's superface approach follows this way[5].

It first takes $O(N^2)$ time to decompose an arbitrary *N-gon* into star-polygons (with no holes). Then, apply any triangulation algorithm for these star-polygons. It takes time no more than $O(N^2)$ time. That is inefficient. Another triangulation method is taken by Hinker and Hansen[3]. It sequentially visits the boundary, and stores triangles as they are discovered. This algorithm involves examining whether a triangle is inside a polyon, and this examining work spends O(*N*) time. The time complexity of this triangulation algorithm is $O(N^2)$. It is also inefficient. In this paper, the triangulation algorithm we use is proposed by Hartel and Mehlhorn[2]. It is a sweep-line algorithm and can efficiently triangulate any polygon with holes. It takes O(*NlgN*) time and O(*N*) space.

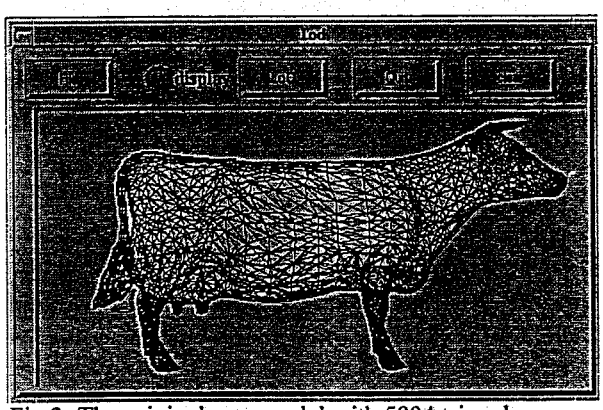

Fig.3: The original cow model with 5804 triangles.

7. EXPERIMENTAL RESULTS

We use two test models for experiments. The platform is Sun UltraSparcII 200MHz with one CPU installed. The experimental results are illustrated in Sections 7.1 and 7.2.

7.1 Experiment 1

The input model is a cow composed of 5804 triangles (Fig. 3). First, we initially use the error tolerance with $\pi/20$ for this original cow model, and show the experimental result in Table 1.

When the level increases, the error tolerance increases, too. As the program converges at pass 4, the error tolerance is $4\pi/20$ and the reduction rate reaches 30%. The simplified cow model in each pass is illustrated in Fig.4 ~ Fig.7. These models are drawn by meshes only, and their corresponding triangulated models are shown in Fig.8 ~ Fig.11.

Pass	Number of Meshes	Triangles	Reduction Rate	Error Tolerance
Initial	5804	5804	-	-
1	3493	5646	97%	$\pi/20$
2	1831	4313	74%	$2\pi/20$
3	904	2772	48%	$3\pi/20$
4	467	1746	30%	$4\pi/20$

Table 1: Total time : 153 seconds (Cow)

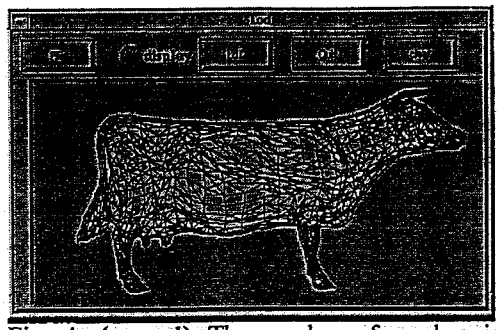

Fig. 4: (*pass 1*) The number of meshes is *3493*, and the error tolerance is $\pi/20$.

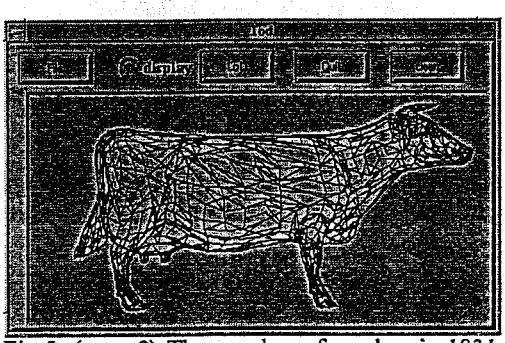

Fig.5: (*pass 2*) The number of meshes is *1831*, and the error tolerance is $2\pi/20$.

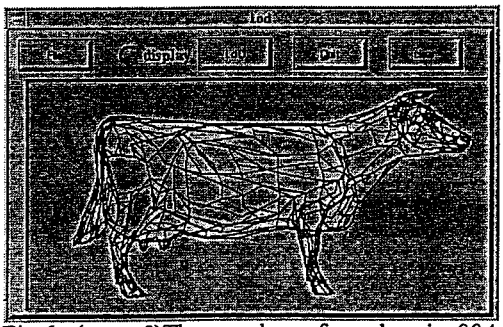

Fig.6: (*pass 3*)The number of meshes is *904*, and the error tolerance is $3\pi/20$.

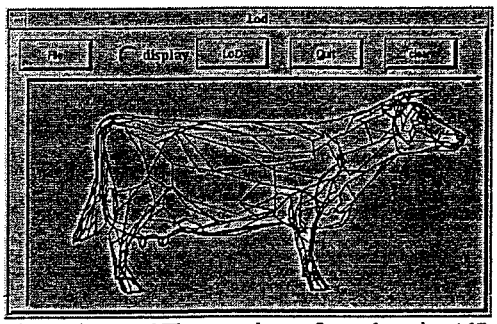

Fig.7: (*pass 4*)The number of meshes is *467*, and the error tolerance is $4\pi/20$.

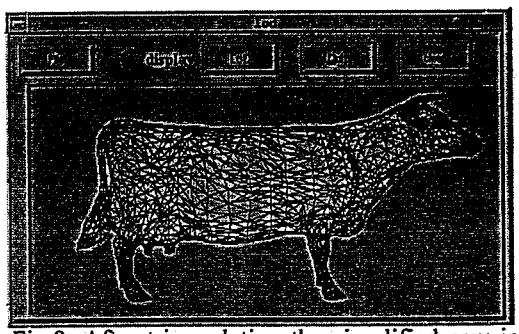

Fig.8: After triangulating the simplified cow in Fig.4, we get this model with *5646* triangles.

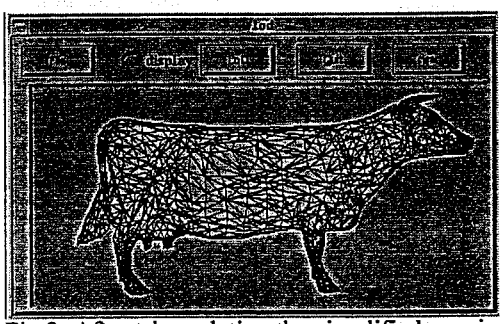

Fig.9: After triangulating the simplified cow in Fig.5, we get this model with *4313* triangles.

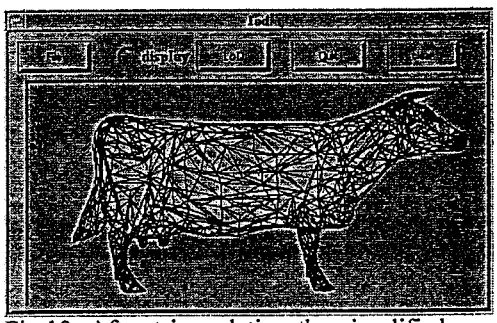

Fig.10: After triangulating the simplified cow in Fig.6, we get this model with *2772* triangles.

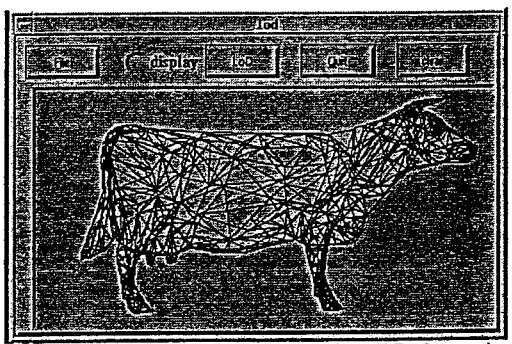

Fig.11: After triangulating the simplified cow in Fig.7, we get this model with *1746* triangles.

Pass	Number of Meshes	Triangles	Reduction Rate	Error Tolerance
Initial	5804	5804	-	-
1	2809	5402	93%	$\pi/10$
2	1121	3351	58%	$2\pi/10$
3	425	1796	30%	$3\pi/10$

Table 2. Total time: 150 seconds (Cow)

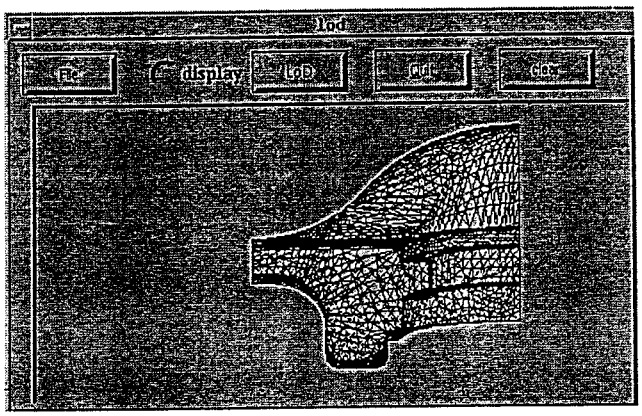

Fig.12: The original Fandisk model with 3392 triangles

Pass	Number of Meshes	Triangles	Reduction Rate	Error Tolerance
Initial	3392	3392	-	-
1	1407	3098	91%	$\pi/10$
2	499	1734	51%	$2\pi/10$
3	195	899	26%	$3\pi/10$

Table 3. Total time: 91 seconds (Fandisk)

Note that, as the error tolerance increases, the area of a mesh increases, too. However, the number of meshes decreases. That is because our simplified method is based on the mesh-merging strategy, not triangle-merging. Therefore, as the level increases, not only does the mesh number decrease, but also the edge number decreases. As a result, we get a simplified model.

Second, we also use the error tolerance with $\pi/10$ for this original cow model, and show the experimental result in Table 2.

From Table 2, we can observe that, after three passes, the reduction rate reaches 30%. Compared with the experimental result shown in Table 1, the program with larger initial error tolerance converges within less passes.

7.2 Experiment 2

The second test model is called 'Fandisk' with 3392 triangles(Fig. 12). We initially use the error tolerance with $\pi/10$ for this original model, and show the experimental result in Table 3.

After three passes, the reduction rate reaches 26%. The simplified Fandisk model in each pass is illustrated in Fig.13 ~ Fig.15. These models are drawn by meshes only, and their corresponding triangulated models are shown in Fig.16 ~ Fig.18.

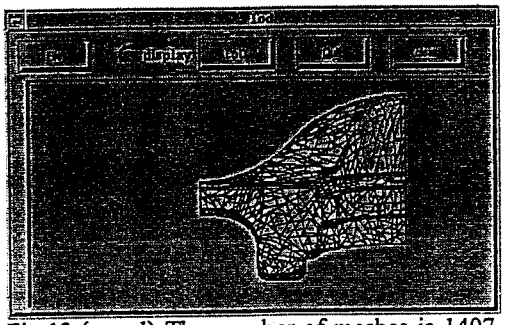

Fig.13:(*pass1*) The number of meshes is 1407, and the error tolerance is $\pi/10$.

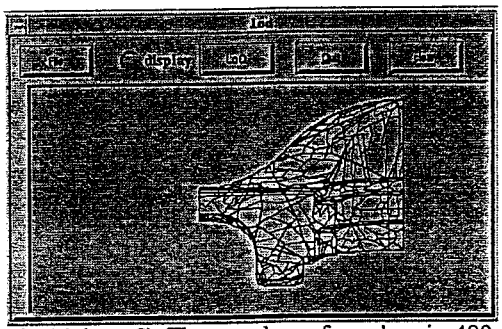

Fig.14:(*pass2*) The number of meshes is 499, and the error tolerance is $2\pi/10$.

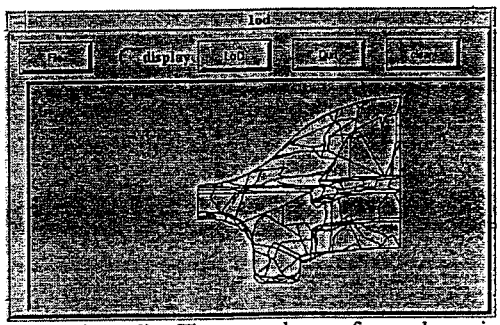

Fig.15:(*pass3*) The number of meshes is 195,and the error tolerance is $3\pi/10$.

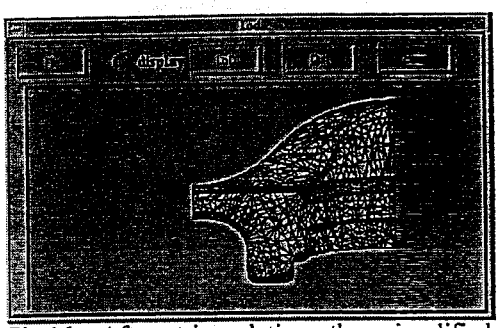

Fig.16: After triangulating the simplified Fandisk in Fig.13, we get this model with *3098* triangles.

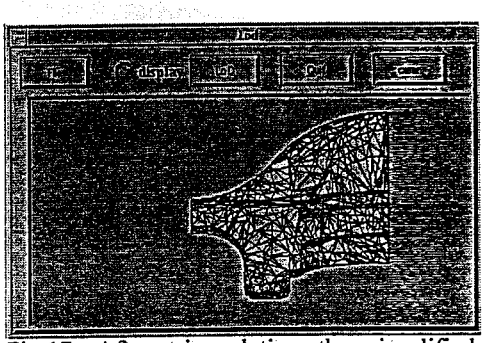

Fig.17: After triangulating the simplified Fandisk in Fig.14, we get this model with *1734* triangles.

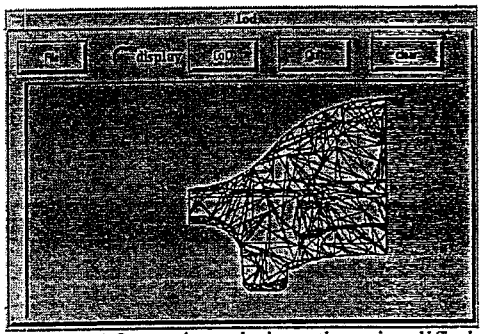

Fig.18: After triangulating the simplified Fandisk in Fig.15, we get this model with *899* triangles.