REAL-TIME RENDERING ALGORITHM BASED ON HYBRID STATIC AND DYNAMIC SIMPLIFICATION METHODS

Zhigeng Pan
Hong Kong Polytechnic University, Hung Hom,Kowloon, HK

Kun Zhou， Jiaoying Shi
Zhejiang University, Hangzhou, P.R. China

ABSTRACT

Level of detail (LoD) method is a key technique for real-time rendering. The generation algorithms of LoD models may be divided into two categories: static or view-independent algorithms and dynamic or view-dependent algorithms, each has its own advantages and drawbacks. This paper presents a new real-time rendering algorithm incorporating both of the two kinds. We simplify polygonal models view-independently according to a user-specified approximation error first. Then the simplified models are used in a view-dependent real-time rendering algorithm. The paper presents a new view-dependent real-time mesh simplification algorithm. The algorithm can produce simplified models in real time while controlling the rendering pixel error. Examples illustrate efficiency of the algorithm.

1. Introduction

Many computer graphics applications such as virtual reality, scientific visualization and computer aided design, require complex, highly detailed models to maintain a convincing level of realism. Consequently, models are often created or acquired at millions of polygons, far more than current PCs or workstations can render at interactive frame rates. Many software techniques have been developed to accelerate rendering speed, such as visibility culling, level of detail (LoD), distributed computation and image based rendering. Visibility culling algorithms determine which subset of the model is visible and only render that portion of the model [1]. LoD algorithms select appropriate level of detail model (containing different number of polygons) to render according to the distance between the viewpoint and the model in a scene [2]. The LoD models are created beforehand or on the fly. Distributed computation algorithms divide the rendering task into several sub-tasks to different processors [3]. Image based rendering algorithms replace arbitrary portions of a scene with some pre-created images and synthesize new images from these images to fit the change of viewpoint. The most advantage of image based rendering algorithms is that they are independent of the scene complexity [4].

LoD models can be generated with mesh simplification algorithms. The algorithms may be divided into two categories: static (also called view-independent) algorithms and dynamic (also called view-dependent) algorithms, each has its own advantages and drawbacks.

View-independent simplification algorithms can produce multiple simplified models using different approximation errors, and these simplified models are saved as different levels of detail for using in real-time rendering. The rendering algorithm selects an appropriate level of the model according to the distance of the model to the viewpoint. Typical algorithms are listed as follows. Schroeder[5] describes an algorithm based on vertex removal. Turk[6] presents an algorithm based on re-tiling. Rossignac[7] presents an algorithm based on vertex clustering. Hoppe [8] introduces an incremental representation for mesh model, called progressive mesh. Cohen [9] uses envelopes to control the global approximation error caused by simplification. Low [10] improves Rossignac's original vertex clustering algorithm using floating-cell vertex clustering. Soucy [11] simplifies mesh models with color attributes using texture mapping. Garland [12] develops a robust surface simplification algorithm using quadric error metrics. Cohen [13] samples the surface position, curvature and color attributes of the mesh model and converts the model to a representation that decouples the sampling of these three attributes, storing the colors and normals in texture and normal maps. This simplification algorithm could preserve the appearance of the model very well.

View-dependent simplification algorithms generate simplified models dynamically according to the viewpoint. Important algorithms are as follows. Hoppe [14] and Xia [15] propose two view-dependent algorithms based on progressive mesh representation. Luebke [16] constructs a hierarchical dynamic simplification for arbitrary polygonal environments and produced an appropriate simplified scene view-dependently.

View-independent simplification algorithms generate simplified models and save them beforehand. Their disadvantages are obvious. First, many simplified models occupy large spaces. Secondly, when the rendering algorithm switches from one level of a model to another level, visual discontinuity is introduced. View-dependent simplification algorithms create simplified models dynamically according to the viewpoint. It can guarantee smooth transition between different LoD, but existing algorithms have to construct a simplification structure for the whole scene and the structure contains a great amount of positive and retrorse information about the simplification operation. If the scene becomes complex, the structure will consume very large main memory.

In this paper we present a real-time rendering algorithm which incorporates view-independent simplification operations and view-dependent simplification operations. We first simplify polygonal meshes view-independently according to a user-specified approximation error. Then the simplified models are used in a view-dependent real-time rendering algorithm. Furthermore, the view-dependent real-time mesh simplification algorithm presented in this paper has obvious features compared with the existing algorithms. The algorithm is based on floating-cell vertex clustering, takes the viewpoint, the line of sight and the resolution of screen window into account. And it is enhanced with viewing frustum culling and back-face culling. The algorithm can produce simplified model in real time while controlling the rendering pixel error.

The rest of this paper is organized as follows: In Section 2, we present the view-independent pre-simplification of models briefly. Section 3 describes the view-dependent real-time simplification algorithm in detail. Section 4 shows some results we have obtained by applying our algorithm to two models. Finally, Section 5 will end with some conclusions and discussion on future work.

2. Static View-independent Pre-simplification

In this pre-simplification process, we generate only one simplified model according to a user-specified approximation error, instead of producing many simplified models and saving them. The simplified model will replace the original model in the following view-dependent real-time rendering process. We employ this pre-simplification process based on the following facts: (a) most mesh models are created using well-developed modeling software. To maintain a convincing level of realism, models are usually over-sampled greatly. For example, models created by CAD systems have many vertices, edges and triangles sharing the same plane and models generated using Marching Cube algorithms contain many redundant triangles too. (b) applications may require different levels of realism. Many applications put more emphasis on rendering in real time than on rendering precisely. Therefore, we can delete the redundant polygons in original models and simplify models within the user-specified approximation error.

We have listed many view-independent mesh simplification algorithms in Section 1. If the algorithm can control the approximation error between the simplified model and the original model, then it can be used as our pre-simplification algorithm. In this paper, we adopt the algorithm based on triangle collapsing operation developed by the authors in 1997 [17]. We assume that models consist of triangles only. This implies no loss of generality, since every polygon in original models can be triangulated easily. Figure 1 illustrates the operation of triangle collapsing operation. The basic operation in our algorithm is triangle collapse. The algorithm computes error matrix for every vertex in the original mesh, the location of the new point generated by the contracting operation is the optimized point according to the computation of distance of point to plane. We introduce an efficient method for transmitting simplification error, and can control the error between the simplified mesh and the original mesh. Readers may refer to [17] for detailed information.

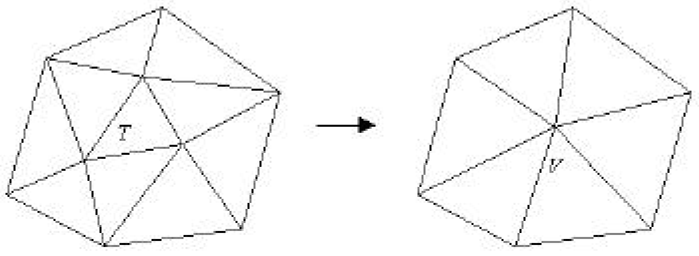


Figure 1 Triangle T collapses to vertex V

Figure 2 lists the results obtained by applying our pre-simplification algorithm to Bunny model. To evaluate the approximation error between the simplified model and the original model [12], we calculate the approximation error E_s between the simplified model M_o and the original model M_s using the following equation:

$$E_s = \frac{1}{|X_o| + |X_s|}\left(\sum_{v\in X_o} d(v, M_s) + \sum_{v\in X_s} d(v, M_o)\right) \quad (1)$$

where X_o and X_s are the vertex sets in the original model M_s and the simplified mode M_o respectively, $|X|$ is the number of vertices in vertex set X, $d(v,M) = \min_{p\in M} \|v - p\|$ is the minimum distance from the vertex v to the closest face of the mesh M. Table 1 gives the approximation errors of the models shown in Figure 2. According to our experiments, when the relative approximation error (the last column in Table 1) is less than 0.02%, we can't tell the difference between the simplified model and the original model by our eyes.

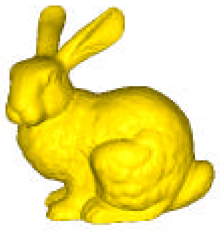
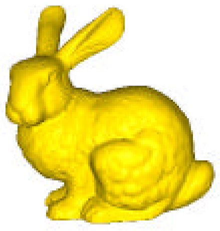
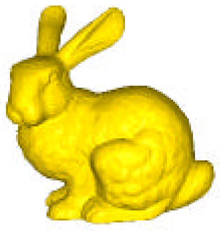
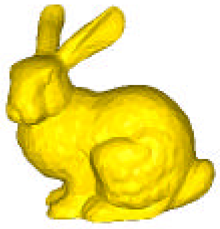
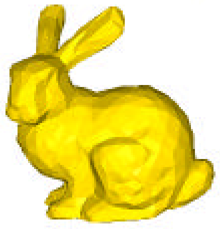
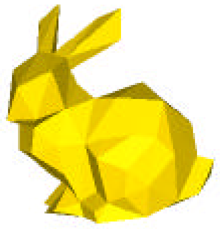
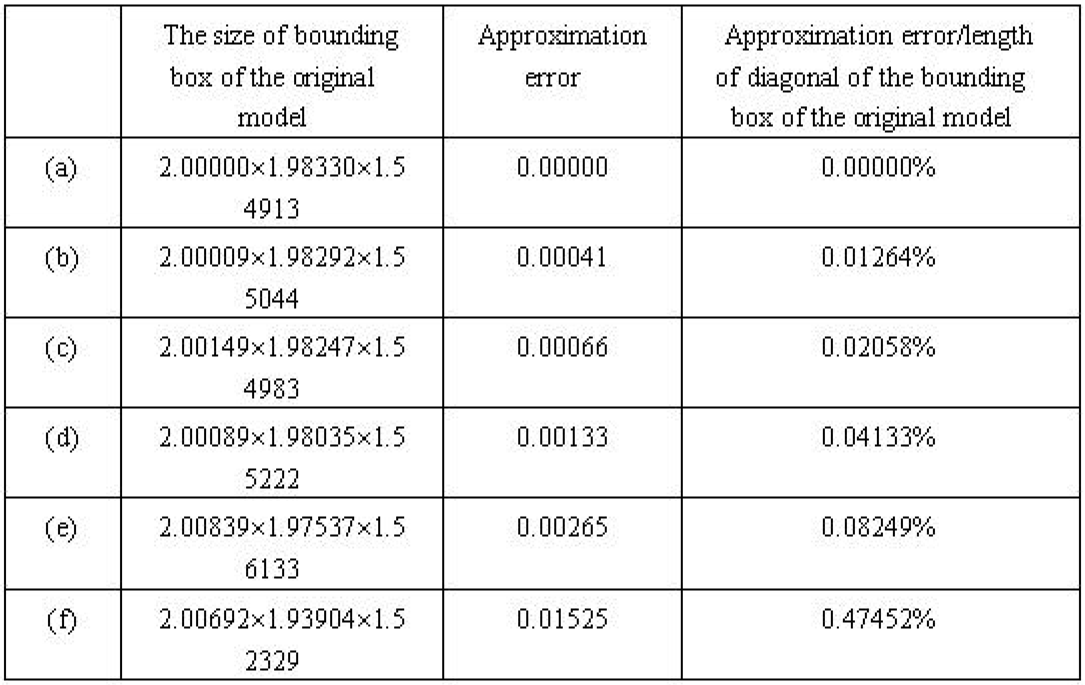
 *(a) The original model (69473 triangles)*	 *(b) 17279 triangles*	 *(c) 8584 triangles*
 *(d) 3416 triangles*	 *(e) 1361 triangles*	 (f) 166 triangles

Figure 2: Pre-simplification results of Bunny model



	The size of bounding box of the original model	Approximation error	Approximation error/length of diagonal of the bounding box of the original model
(a)	2.00000×1.98330×1.54913	0.00000	0.00000%
(b)	2.00009×1.98292×1.55044	0.00041	0.01264%
(c)	2.00149×1.98247×1.54983	0.00066	0.02058%
(d)	2.00089×1.98035×1.55222	0.00133	0.04133%
(e)	2.00839×1.97537×1.56133	0.00265	0.08249%
(f)	2.00692×1.93904×1.52329	0.01525	0.47452%

Table 1: Approximation errors according to the models illustrated in Figure 2

3. Dynamic View-dependent Real-time Simplification

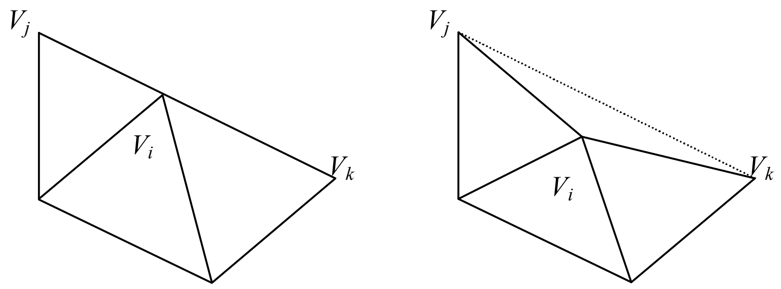
Considering the real-time requirement of the rendering algorithm, we adopt the mesh simplification algorithm based on vertex clustering [7,10]. To get rid of the shortcoming of the original simplification algorithm based on vertex clustering [7], Low presents a simplification algorithm using floating-cell clustering to generate simplified models [10]. The algorithm takes vertices with greater weight values (the more important vertices) as the centers of vertex clustering. Although the algorithm considers the distance between the viewpoint and the model in a scene when it calculates the radius of the bounding sphere used in vertex clustering, it doesn't emphasize particularly on real-time simplification, but takes the distance as an input parameter used in automatic simplification. It puts more emphasis on the calculation of the weight values of vertices for better preservation of important features of models, and it is regarded as a static mesh simplification algorithm. However, we extend the algorithm for dynamic real-time simplification. Compared with Low's algorithm, our algorithm use a simple method to calculate the weight values of vertices and focus particularly on rendering error (evaluated with pixels) of the model in screen window. It simplifies the model in real-time according to the viewpoint, the line of sight and the resolution of the screen window. Furthermore, to accelerate rendering speed, this algorithm is enhanced with viewing frustum culling and back-face culling. The whole algorithm consists of pre-computation of weight values of vertices and real-time simplification. We discuss them in detail respectively.

3.1 PRE-COMPUTATION OF WEIGHT VALUES OF VERTICES

This step calculates the weight values of the vertices in the simplified model obtained by the pre-simplification process, and sorts the vertices in decreasing order on their weight values. In fact, this step is performed as a preprocessing process. The sorted queue is to be used in real-time simplification process discussed in Subsection 3.2. There are many methods to calculate the weight value of a vertex. We employ a simple method. For non-boundary vertex V_i (the triangles sharing the vertex form a ring), let Ts_i be the set of the triangles sharing V_i, then the weight value C_i of V_i can be calculated using the following equations:

$$\bar{n}_i = \frac{\sum_{T \in Ts_i} \Delta T \cdot \bar{n}_T}{\sum_{T \in Ts_i} \Delta T}, \qquad C_i = \frac{\sum_{T \in Ts_i} \angle(\bar{n}_i, \bar{n}_T)}{|Ts_i|} \tag{2}$$

Where ΔT is the area of triangle T, $\bar{n}_T$ is the normal vector of the triangle T, $\angle(\bar{n}_i, \bar{n}_T)$ is the angle between the vector $\bar{n}_i$ and the vector $\bar{n}_T$, $|Ts_i|$ is the number of triangles in the triangle set Ts_i. For boundary vertex V_i (refer to Figure 3), if vertex V_i, V_j and V_k are on the same line, the weight value of V_i is the same as non-boundary vertex. Otherwise, the triangle formed by V_i, V_j and V_k has to be considered when calculating $\bar{n}_i$. The weight value can represent the importance of a vertex in geometry: the greater the weight value of the vertex, the more important the vertex.



(a) V_i, V_j and V_k are in the same line (b) V_i, V_j and V_k are not in the same line

Figure 3: The weight value calculation of boundary vertex

3.2 REAL-TIME SIMPLIFICATION

Real-time simplification algorithm simplifies the model in real time according to the viewpoint, the line of sight and the resolution of the screen window. To accelerate rendering process, our algorithm combines with viewing frustum culling and back-face culling. Although many graphics packages contain viewing frustum culling and back-face culling, but they have to execute many other operations before executing these two operations in the rendering pipeline. Our algorithm integrates these two operations very well, on the one hand it saves rendering time, on the other hand it is helpful to preserve the silhouette when the model is rendered in screen window. We will discuss this further in the following section. The algorithm framework is discussed below.

3.2.1 Algorithm framework

Let ***N*** be the number of vertices, ***queue[N]*** be the sorted queue described in Section 3.1, and ***Map[N]*** be the vertex mapping array which is the key data structure in our algorithm. The real-time simplification algorithm can be described using the following pseudo-code:

```
begin
//Step 1: Initialization, each vertex is mapped to itself.
for (i:=1; i<=N; i++)
Map[i] := i;
//Step 2: Viewing frustum culling and back-face culling.
for ( each vertex $V_i$ in the mesh )
if ( $V_i$ is not in the current viewing frustum  or  $V_i$ is back-face vertex )
        Map[$V_i$] := -1;
//Step 3: Vertex clustering.
for (i:=1; i<=N; i++)
{
$V_i$ := queue[i];
if ( Map[$V_i$] = $V_i$ )
{
Calculating the radius $R_i$ of the bounding sphere according to the current
        viewpoint, the line of sight and the resolution of screen window ;
```

```
  for ( each vertex $V_k$ falling in the sphere with center $V_i$ and radius $R_i$ )
        if ( Map[$V_k$] = $V_k$ )
              Map[$V_k$] := $V_i$ ;
  }
}
//Step 4: Generating the simplified mesh.
for ( each triangle $T_i$ in the mesh )
{
if ( Map[$V_{i1}$] ¹ -1 and Map[$V_{i2}$] ¹ -1 and Map[$V_{i3}$] ¹ -1 )     // $V_{i1}$, $V_{i2}$ and $V_{i3}$ are vertices of $T_i$
  {
        if ( Map[$V_{i1}$] = -1 )
              V[1] := $V_{i1}$;
        else
              V[1] := Map[$V_{i1}$];
        if ( Map[$V_{i2}$] = -1 )
              V[2] := $V_{i2}$;
        else
              V[2] := Map[$V_{i2}$];
        if ( Map[$V_{i3}$] = -1 )
              V[3] := $V_{i3}$;
        else
              V[3] := Map[$V_{i3}$];
        Output the triangle formed by V[1], V[2] and V[3] to the simplified mesh ;
  }
}
end
```

3.2.2 Viewing frustum culling and back-face culling

Figure 4 illustrates the relations among viewing frustum, line of sight and models in rendering process. For viewing frustum culling, we first calculate the equations (represented by *Plane[6][4]*) of the six planes of the viewing frustum and the coordinate of an arbitrary point in the viewing frustum (for example, the center of the viewing frustum *Q[4]*), then we can decide whether a spatial *P[4]* is in the viewing frustum by the following method: for all *1£ i £6* ,if *(Plane[i]·Q)*(Plane[i]·P)>0*, then *P* is in the viewing frustum; Otherwise, *P* is not in the viewing frustum.

Let *d[3]* be the vector from a vertex to the viewpoint and *n[3]* be the normal vector of the vertex, if $(d \cdot n)<0$, then the vertex is a back-face vertex. We point out particularly that one or two vertices of some triangles in the mesh model (for example, triangle T_1 and T_2 in Figure 4) are outside the viewing frustum or are back-face vertex, the other two or one vertices of them are not. Our algorithm does not cull these vertices but keep these triangles as a whole. However, these triangles are usually in the silhouette of the model when they are rendered, so our algorithm keeps the silhouette of the model very well.

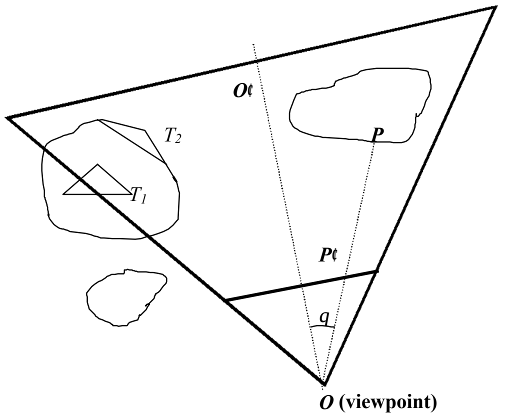


Figure 4: Viewing frustum, line of sight and models

3.2.3 VERTEX CLUSTERING

The key of vertex clustering is to calculate the radius of the bounding sphere, it is also the key of our real-time mesh simplification algorithm. We analyze how to calculate the radius of the bounding sphere according to the rendering pixel error in screen window (refer to Figure 4). If the resolution of the screen window is *(Width*Height)* and the size of the near plane of the viewing frustum is *(W*H)*, then the area in the near plane corresponding to one pixel in the window is *(W*H)/(Width*Height)*. If a vertex *P* of the model is projected to *P¢* (as in Figure 4) in the near plane and the rendering error is within *K* pixel(s), then the radius *R* of the bounding sphere at vertex *P* can be calculated using the following equation:

$$R = \sqrt{\frac{K * W * H}{Width * Height}} * \frac{\|OP\|}{\|OP'\|} \tag{3}$$

Considering the following fact about the line of sight, the farther the model is away from the central line of sight, the blurrier the model is. Let the angle between *OP* and the central line of sight *OO¢* be q and we assume that the radius of the bounding sphere is increased with q as a 2-power function (other functions are available), the equation for computing radius R is as the following:

$$R = \sqrt{\frac{K * W * H}{Width * Height}} * \frac{\|OP\|}{\|OP'\|} * \left(\frac{90^0}{90^0 - q}\right)^2 \tag{4}$$

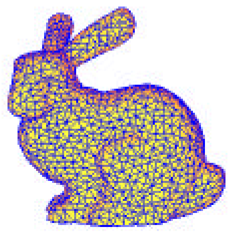
3.2.4 GENERATING THE SIMPLIFIED MODEL

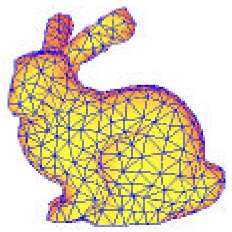
After the vertex mapping array is obtained from vertex clustering, we only need to decide whether each triangle in the input mesh can still form a triangle after the vertices are mapped. If it can, it is output to the simplified mesh. Figure 5 illustrates several simplified bunny models according to different rendering pixel errors.

(a) The original model (69473 triangles)

(b) 1 pixel error (18085 triangles)

(c) 2 pixels error (6083 triangles)


(d) 4 pixels error (166 triangles)


(e) pixels error (1769 triangles)

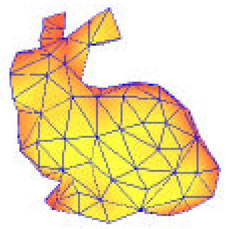

(f) 16 pixels error (839 triangles)

Figure 5: Simplify bunny model with different rendering pixel errors

4. Implementation and experiment results

We implemented the hybrid real-time rendering algorithm on Pentium Pro 233M PC with 64M memory using OpenGL. Users can change viewpoint, line of sight, viewing frustum and rendering pixel error in our system. Functions for recording and playing walkthrough paths are also provided.

We applied the algorithm to various models, two models (Bunny and Helicopter) are shown below. For each model, we recorded a path through the model. Figure 6 and Figure 7 illustrate three frames during the walkthrough[1]. (a), (b) and (c) are rendered without using LoD method, (d), (e) and (f) are rendered using our dynamic simplification method (without pre-simplification), (g), (h) and (i) are rendered using our hybrid algorithm, while (j), (k) and (l) are the differences between (a) and (g), (b) and (h), (c) and (i), respectively. From these images, it is obvious that our algorithm can control the rendering pixel error very well.

[1] More results can be found at http://cad.zju.edu.cn/home/kzhou/HybridLoD.html

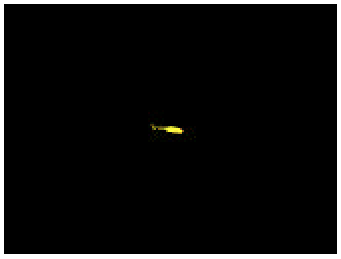
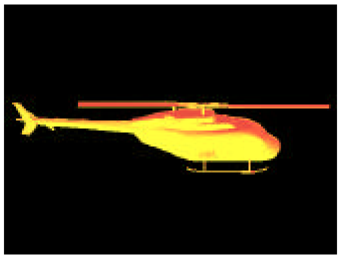
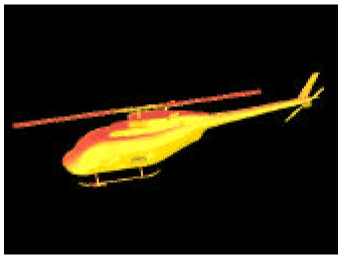
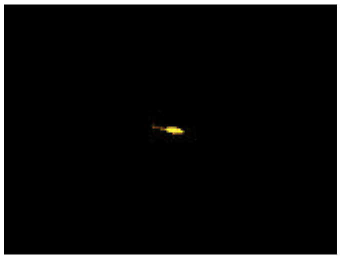
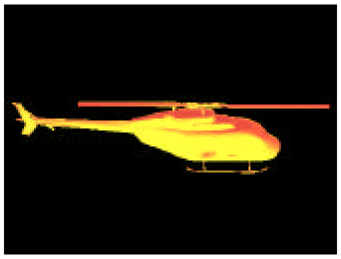
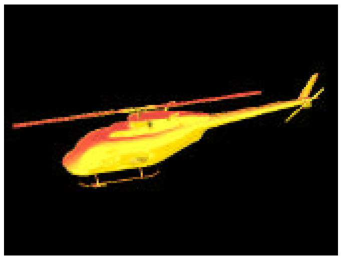
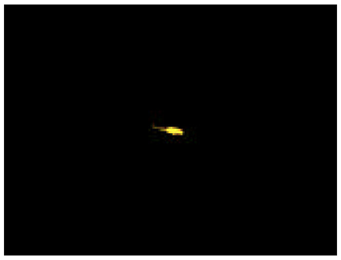
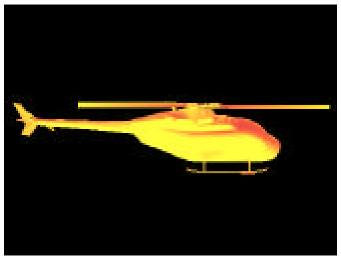
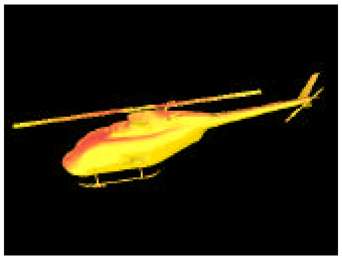
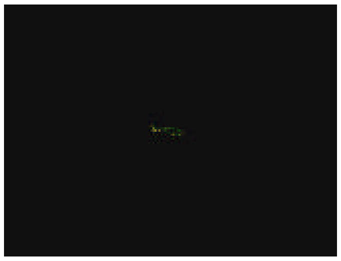
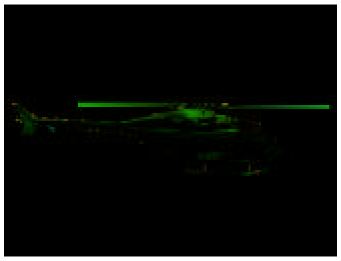
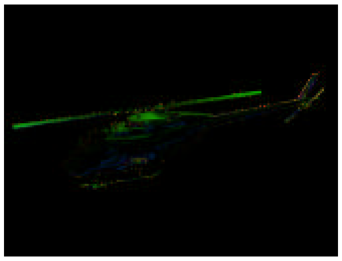
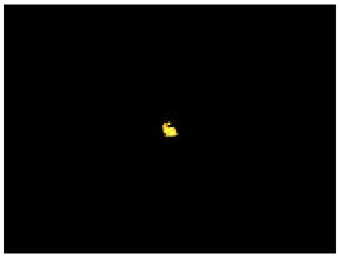
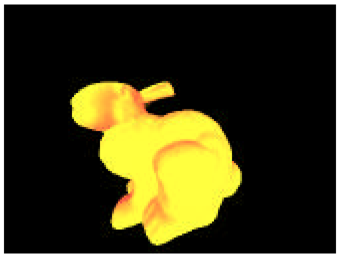
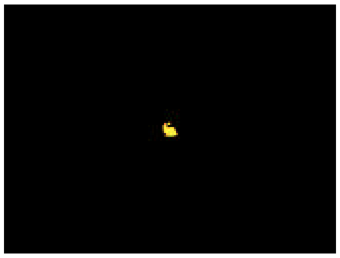
 *(a) No.20 frame*	 *(b) No.100 frame*	 *(c) No. 232 frame*
 *(d) 609 triangles*	 *(e) 4865 triangles*	 *(f) 4916 triangles*
 *(g) 346 triangles*	 *(h)1900 triangles*	 *(i) 1880*
 *(j) (a)-(g)*	 *(k) (b)- (h)*	 *(l) (c)- (i)*

Figure 6: Rendering of Helicopter


(a) No.0 frame

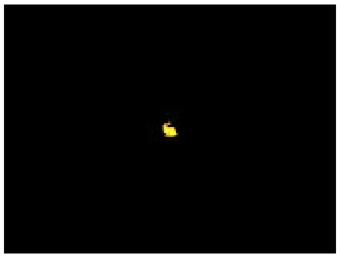

(b) No.88 frame

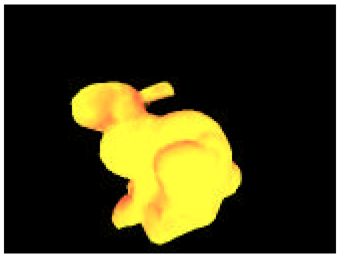
(c) No.232 frame


(d) 1464 triangles

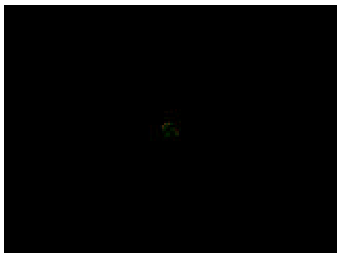
(e) 6235 triangles

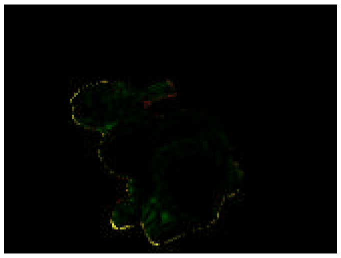
(f) 7223 triangles


(g) 979 triangles


(h) 3525 triangles

(i) 4632 triangles


(j) (a)-(g)


(k) (b)- (h)

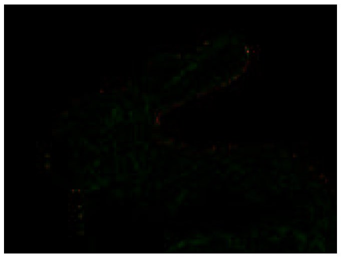

(l) (c)- (i)

Figure 7 Rendering of Bunny

Table 2 lists the frame rates of the walkthrough processes. Compared with the frame rates of the rendering process without using LoD method, the frame rates of the rendering process only using dynamic simplification method is improved, but not greatly. When the static pre-simplification process is added, the frame rate is improved greatly, while the rendering pixel error is very small.

	Helicopter	Bunny
The resolution of the screen window	400×300	400×300
The rendering pixel error	1 pixel	1 pixel
The number of frames	600	300
The number of triangles in the original model	14168	69473
The number of triangles in the pre-simplified model	3514	17279
Approximation error between the pre-simplified model and the original model	0.00896%	0.01264%
Frame rate without using LoD method	5.76	1.30
Frame rate with using dynamic LoD method (without pre-simplification)	8.90	3.23
Frame rate using our algorithm	22.39	8.81

Table 2: Frame rates of the rendering processes

It should be noted that the two examples given here are individual frames grabbed from the walkthrough process, and the scene is composed of a single model. However, our algorithm can be applied to scenes composed of multiple objects. And with the increase of models, the effect of viewing frustum culling is more obvious, the algorithm will become more efficient.

5. Conclusions and future work

In this paper, we present a hybrid real-time rendering algorithm incorporating static view-independent simplification method and dynamic view-dependent simplification method. Our algorithm makes use of the advantages of these two methods, it improves the rendering speed greatly while the visual effect is preserved. In addition, compared with the existing view-independent mesh simplification algorithms, the algorithm presented in this paper has its own features, it is based on floating-cell vertex clustering and simplifies the model in real time according to the viewpoint, the line of sight and the resolution of screen window. And it is enhanced with viewing frustum culling and back-face culling. Our algorithm is expected to be applied in virtual reality, interactive visualization and CAD systems.

We are currently investigating ways to deal with the color and texture features of mesh models. Although some algorithms are available [11,13], but they can't run in real time. Furthermore, we are developing methods to rendering models with constant frame rates, which is very important in virtual environment. In addition, the way to combine our algorithm with other real-time rendering techniques is also under exploration

REFERENCES

1 Airey, J. Towards image realism with interactive update rates in complex virtual building environments. *Symposium on Interactive 3D Graphics*, 1990, 41-50.

2 DeHaemer, Jr., M. and Zyda, M. J. Simplification of objects rendered by polygonal approximations. Computers & Graphics, 15(2):175-184, 1991.

3 Pan Zhigeng, Zhang Mingmin, Shi Jiaoying. Time-critical computing in virtual environment. Proceedings of CAD/Graphics'95, 1995, 1078-1083.

4 Chen, S. E. and Williams L. View interpolation for image synthesis. Computer Graphics (*SIGGRAPH'93*), 1993, 27:279-288.

5 Schroeder, W.J., Zarge, J.A. et al. Decimation of triangle meshes. Computer Graphics, 1992, 26(2):65-70.

6 Turk, G. Re-tiling polygonal surface. Computer Graphics, 1992,26(2):55-64.

7 Rossignac, J. and Borrel, P. Multi-resolution 3D approximation for rendering complex scenes. In Falcidieno, B. and Kunii, T., editors, Geometric Modeling in Computer Graphics, Springer Verlag.1993: 455-465.

8 Hoppe, H. Progressive meshes. Computer Graphics (*SIGGRAPH'96*), 1996, 30:99-108.

9 Cohen, J., Varshney, A., Manocha, D et al. Simplification envelopes. Computer Graphics (*SIGGRAPH'96*), 1996, 30:119-128.

10 Low, Kok-Lim and Tan, Tiow-Seng. Model Simplification Using Vertex-Clustering. *Symposium on Interactive 3D Graphics*, 1997, 75-81.

11 Soucy, M., Guy, G., Marc, R.. A texture-mapping approach for the compression of colored 3D triangulations, The Visual Computer, 1996, 12:503-514.

12 Garland, M., Heckbert, P.S. Surface simplification using quadric error metrics. Computer Graphics (*SIGGRAPH'97*), 1997, 31:209-216.

13 Cohen, J., Olano, M et al. Appearance-preserving simplification. Computer Graphics (*SIGGRAPH'98*), 1998, 31:189-198.

14 Hoppe, H. View-dependent refinement of progressive meshes. Computer Graphics (*SIGGRAPH'97*), 1997, 31:189-198.

15 Xia, J. El-Sana, J. and Varshney, A. Adaptive real-time level-of-detail-based rendering for polygonal models. IEEE Transactions on Visualization and Computer Graphics, 1997, 3:171-183.

16 Luebke, D. and Erikson, C. View-dependent simplification of arbitrary polygonal environments. Computer Graphics(*SIGGRAPH'97*), 1997, 31:209-216.

17 Zhou Kun, Pan Zhigeng, Shi Jiaoying. A new mesh simplification algorithm based on triangle collapse. The Journal of Computer, 1998,21(6):506-514 (in Chinese).

BIOGRAPHIES

Zhigeng Pan

Dr. **Zhigeng Pan**, is a full professor at the State Key Lab. of CAD&CG, Zhejiang University, a member of the IS&T, a member of the committee on Multimedia Computing, Chinese Image and Graphics Association. He held the position of the Scientific Assistant to the Director from 1995-1997, responsible for coordination and strategy activities. Now he is the vice director of Hangzhou Center, CAD Training Network in China. He is on the Editorial Board of Chinese Computer Image & Graphics Journal.

Since 1993, he has been working at the State Key Lab of CAD&CG on a number of academic and industrial projects related with distributed graphics, virtual reality, multimedia. He have been engaged in research and development on VR/VE since 1991. Now he is the project leader or co-leader of 4 projects on VR.

He has published more than 40 papers on international journals, national journals and international conferences. He is the author or co-author of two books related to computer graphics. He and Prof. Jiaoying Shi, the academic director of State Key Lab of CAD&CG, were invited to write papers for ACM Computer Graphics to introduce CG achievements in China(Computer Graphics, No.2, No.3,1996). He acted as one of the Guest Editors of a Special Issue of the international journal Computers & Graphics. His current research interests include virtual reality/virtual environment (multi-resolution modeling; time-critical rendering; distributed VR, et al.), distributed graphics, and multimedia (collaborative CAD, SCW, shared application, et al).

Contact information:

Professor Zhigeng Pan
Computing Department
Hong Kong Polytechnic University
Hung Hom, Kowloon, HK
Email link: mailto:cszgpan@comp.polyu.edu.hk

Jiaoying Shi is a professor of the Department of Computer Science and Engineering at Zhejiang University which is located in Hangzhou, Zhejiang Province of China. He is now the Director of Academic Committee of State Key Lab of Computer Aided Design and Computer Graphics (in short as State Key Lab of CAD&CG). Professor Shi is the Deputy Chairman of China Image and Graphics Association, the Deputy Chairman of China CAD and Graphics Society under China Computer Federation.

Prof. Jiaoying Shi was born in Ningbo of Zhejiang Province in 1937. He finished his School education in Ningbo, and graduated from Department of Physics at Leningrad University of USSR in 1960. He worked as an assistant Prof. at North-West Polytechnic University in Xi'an from 1960 to 1963. He worked as an assistant Prof. at Shanghai University of Science and technology in Shanghai from 1963 to 1973. Then he joined the Radio_Electronics Department of Zhejiang University in 1973. In 1978 he transferred to the Department of Computer Science and Engineering at the same University. In the years from 1982 to 1984 he worked as an adjunct associate professor at Department of Electrical and Electronics of University of Florida, USA. After he returned from USA, he rejoined Zhejiang University and is working there until now. He was promoted as a full professor in 1987. He served as the vice Chairman of the Department of Computer Science and Engineering(1985-1988), the Director of University Division of Scientific Research Affairs (1988-1991), and the Director of State Key Lab of CAD&CG for 10 years (1989-1998).

In 1960s he did basic research and teaching in Nuclear Physics area. In 1970s he changed his research direction into minicomputer hardware design and applications. Since 1980 his research interests lie on computer aided design, especially on geometry modeling and numerical control programming. In late 1980s' he was actively involved in the research works in computer graphics area. Since 1990 his works are concentrated in the visualization in scientific computing and virtual environment. He has published more than 100 papers and two books. the State Key Lab of CAD&CG he lead is evaluated as one of top 10 excellent research labs in China (see 1995 November issue of American Journal of Science).

Contact information:

Prof. Jiaoying Shi
State Key Lab of CAD&CG
Zhejiang University
Hangzhou, 310027, P.R. China
Email link: mailto:jyshi@cad.zju.edu.cn

Kun Zhou is a graduate student in the Department of Computer Science and Engineering at Zhejiang University. His research interests include time-critical rendering, multi-resolution modeling, and image-based rendering/modeling. He has published around 10 papers in various journals and proceedings related to VR. In 1999 he was awarded a Microsoft fellowship.

Contact information:

Kun Zhou
State Key Lab of CAD&CG,
Zhejiang University, Hangzhou, 310027, China
Phone: +86-571-7951045
Fax: +86-571-7951780
Email link: mailto:kzhou@cad.zju.edu.cn