**Time-critical rendering algorithm with incorporation of LoD, visibility culling and object impostor**

**By Mingmin Zhang*, Zhigeng Pan and Pheng-Ann Heng**

Given enough CPU time, present graphics technology can render near-photorealistic images. However, for real-time graphics applications such as virtual reality systems, developers must make explicit programming decisions, trading off rendering quality for interactive update rates. In this paper we present a new algorithm for rendering complex 3D models at near-interactive rates that can be used in virtual environments composed of static or dynamic scenes. The algorithm integrates the techniques of level of details (LoD), visibility computation and object impostor. The method is more suitable for very dynamic scenes with high depth complexity. We introduce a new criterion to identify the occluder and the occludee: the object that can be replaced by its LoD model and the one that can be replaced by its impostor. The efficiency of our algorithm is then illustrated by experimental results.

Copyright © 2003 John Wiley & Sons, Ltd.

Received: 2 January 2003; Revised: 7 April 2003

**KEY WORDS:** time-critical rendering; occluder; occludee; object impostor; LoD

---

**Introduction**

For highly interactive programs, such as virtual environment applications, the human perceptual system drives the rendering requirements, especially the need to maintain a minimum of 10 frames per second or faster. At the same time, developers are trying to present a high-fidelity graphics environment with a more realistic scene and a richer set of perceptual cues to help immerse the user. This time/quality conflict forces the developer to identify when the rendering engine is overloaded and to manually implement time-critical rendering techniques in order to maintain immersive frame rates. Recent efforts have been focusing on employing visibility culling, image impostor and level of detail (LoD) for the time-critical rendering paradigm.

Visibility culling algorithms determine which subset of the model is visible and therefore only render the visible portion. The simplest visibility culling methods are backface culling and view frustum culling. They are much faster, but do not provide the exact occlusion relationship of each triangle in the mesh. For the triangles that are not backfaced and within the view frustum, hierarchical visibility culling, hierarchical occlusion maps, and other algorithms can serve the purpose. Yet, all these methods require pricey pre-process, and could not be used when many objects in a scene change dynamically.

To accelerate the rendering speed, several walkthrough systems use object impostor to replace geometry mesh. Aliaga improved upon a cells and portals framework by using image-based rendering techniques. They replaced the visible geometry through portals with images. Thus, we reduce the rendering complexity of the current cell.

LoD is a common technique that authors several versions of a model at various levels of detail; a detailed triangle mesh is used when the object is close to the viewer, and coarser approximations are substituted as
the object recedes.\textsuperscript{7, 8} LoD is a useful method for time-critical rendering and real-time geometry transmission.\textsuperscript{9} It can control the error, and the simplification process can be stopped when the time budget is reached. It can also be used in dynamic scenes in real time. However, when the scene is very dynamic, the models will be very much simplified because the LoD method will trade off image quality with frame rate, which will cause a high degree of error.

In this paper we present a new algorithm which integrates LoD, visibility computation and object impostor. Our algorithm determines which subset of the model would most likely be occluders. It then exploits the LoD and object impostor methods to simplify the occludees. The algorithm is more suitable for high depth and highly dynamic scenes. Some characteristics of our algorithm are as follows.

**Low-Complexity Preprocess**

In the preprocess step, our algorithm calculates the weight according to the size and the normal vector of the triangle and the visible possibility (namely ‘solidity’ in our algorithm). It creates an octree and saves the occlusion relationship between two adjacent triangles in one object. For a mesh including dozens of thousands of triangles, the preprocessing can be completed within a minute.

**No Need to Choose Occluders Beforehand**

With our method, we can calculate the possible occluder and occludee very quickly. When the scene is rendered, there is no need to predict the viewpoint and view direction.

**Adaptive Rendering**

Our algorithm is adaptive and can render complex scenes in real time on any computer and graphic system, regardless of their performance. The better the performance, the better is the image quality.

**Automatic Selection**

Our method can automatically identify occluders and occludees. For the occludees, our method can also automatically select objects that can be replaced with LoD models and object impostors.

The paper is organized as follows. The next section introduces our theoretical background and cites related works. The third section describes the techniques used in our new algorithm, including the computation of weight and solidity for triangles, finding of the first occluder, and dynamic LoD generation based on vertex clustering. In the fourth section, some experimental results are given, and we draw the conclusion and list possible future work in the final section.

**Background and Related Work**

Scene complexity can be reduced by many methods for the walkthrough of static scenes. Here we will discuss levels of detail, visibility culling and object impostor.

**Level of Detail (LoD)**

These techniques model the objects in the scene at different levels of detail. They select a particular LoD for each object, based on various considerations such as the rendering cost and perceptual contribution to the final image. LoD models can be generated with mesh simplification algorithms.\textsuperscript{10–16} The algorithms may be divided into two categories: view-independent and view-dependent.

**View-independent Simplification.** Multiple simplified models can be produced using different methods and these models saved beforehand as different levels of detail for use in real-time rendering. Typical algorithms are listed as follows. Schroeder et al.\textsuperscript{11} described an algorithm based on vertex removal; Garland and Heckbert\textsuperscript{12} developed a robust surface simplification algorithm using quadric error metrics.

**View-dependent Simplification.** Simplified models can be generated dynamically according to the viewpoint movement. Luebke and Erikson\textsuperscript{16} constructed a hierarchical dynamic simplification for arbitrary polygonal environments and produced an appropriate simplified view-dependent scene. Hoppe\textsuperscript{14} proposed a view-dependent algorithm based on progressive mesh representation.

Dynamic LoD algorithms simplify the mesh model according to the viewpoint, the view direction, the mesh size, the distance and other factors. They can provide smooth change and do not need much memory to save various levels of detail.
Visibility Culling

Visibility culling algorithms, a much slower process, determine which subset of the model is visible and only render the visible portion. The simplest visibility culling processes are backface culling and view frustum culling. Although they are much faster, they do not give out the exact occlusion relationship of each triangle in the mesh. For the triangles that are not backface and within the view frustum, there are many methods to delete the occluded object from rendering:

- Zhang et al.\(^2\) used hierarchical occlusion maps to solve the visibility problem by using two hierarchies: an object-space bounding volume hierarchy and a hierarchy of image-space occlusion maps. For each frame, objects from a pre-computed database were chosen to be occluders and were used to cull geometry that could not be seen.
- To improve the efficiency of occlusion computation for densely tessellated models, Law and Tan\(^17\) introduced the use of simplification to automatically deduce virtual occluders. He illustrated the efficiency of the technique to pre-process occlusion for outdoors scenes. Using virtual occluders to pre-compute visibility, only visible surfaces may be refined for real-time selective refinement.

However, all these methods require costly preprocess, and could not be used when many objects in a scene change dynamically.

Object Imposter

The trade-off between rendering objects and displaying images has been leveraged elsewhere to provide speed-ups. Texture-based simplification of complex scenes,\(^4\) in which distant geometry is rendered into a texture and then cached, has shown some promising results. Image-based rendering has recently become a very popular topic, providing many techniques to handle scene complexity. Maciel and Shirley\(^5\) expanded an LoD system to allow a general set of impostors (LoDs, textured billboards, etc.). A visual navigation system was presented which uses texture-mapped primitives to represent clusters of objects to maintain high and approximately constant frame rates. In cases where there are more unoccluded primitives inside the viewing frustum than can be drawn in real time on the workstation, this system ensures that each visible object, or a cluster that includes it, is drawn in each frame. The system also supported the use of traditional ‘level-of-detail’ representations. Schaufler\(^6\) exploited frame-to-frame coherence, presenting a load-adaptive frame-to-frame algorithm to replace complex distant objects with an image of the respective object mapped onto them.

Time-Critical Rendering

Time-critical rendering of scenes composed of many objects is an open research area. It ensures guaranteed frame rates even for scenes with very high complexity. Therefore it provides a convenient framework for real-time rendering applications. Typically, a time-critical framework mainly consists of an LoD selection method, which chooses the most valuable representation for visible objects not exceeding the available rendering budget.

Funkhouser and Sequin\(^18\) proposed an adaptive display algorithm which guarantees that the LoDs of objects and rendering methods for generating the best image within a user-specified target frame rate can be chosen. Ohshima et al.\(^19\) devised an adaptive scheme to control the LoDs of rendered objects based on a human visual acuity model with gaze detection devices. Fujishiro et al.\(^20\) proposed a highly efficient time-critical rendering (TCR) approach to the LoD control of textures used in image-based virtual reality systems.

Gobbetti and Bouvier\(^21\) described a framework for TCR of graphic scenes composed of a large number of objects having complex geometric descriptions. Their technique relied upon a scene description in which objects were represented as multi-resolution (continuous LoD) meshes. As pointed out by Bouvier and Gobbetti\(^22\) the traditional approach to render these scenes in a time-critical setting is to pre-compute a small number of independent LoD representations of each object composing the scene, and to switch at run time between the LoDs. This technique has multiple drawbacks, both in terms of memory requirements, because of the need to store multiple LoDs, and quality of results, because of the NP-completeness of the problem. Bouvier and Gobbetti recently demonstrated that these drawbacks are overcome when using appropriate multi-resolution data structures (TOM) which enable expression of predictive LoD selection in the framework of continuous convex constrained optimization.\(^22\)

Klosowski and Silva\(^23\) presented a technique for optimizing the rendering of high-depth complexity scenes, which used prioritized-layered projection (PLP) to render an estimation of the visible set. Aliaga and Lastra\(^24\) presented an algorithm that took a three-dimensional...
model as input and automatically bounded the geometric complexity for all viewpoints and view directions in the model—thus allowing for a constant frame rate. The pre-processing algorithm determined the best subsets of a 3D model to replace with images. In our algorithm, we implemented a time-critical rendering system by exploiting the LoD and visibility culling method.

Zach et al.\textsuperscript{25} introduced a novel LoD selection method for real-time rendering that works on hierarchies of discrete and continuous representations. They integrate point-rendered objects with polygonal geometry and demonstrate our approach in a terrain flyover application, where the digital elevation model is augmented with forests. The vegetation is rendered as a continuous sequence of splats, which are organized in a hierarchy. Related work can be found in Farias et al.\textsuperscript{26} and Sloan et al.\textsuperscript{27}

Most of the aforementioned methods are for the walkthrough of a static scene. Few methods are suitable for a highly dynamic scene. We will discuss a method suitable for high depth and highly dynamic scenes in the following section.

**Overview of the Algorithm**

Our algorithm first selects some triangles in the scene as occluders according to their visible possibility (the solidity) and the computer performance. We then use the LoD method and a frame coherence object impostor to simplify the occludees. The algorithm is composed of three major steps listed below, which will be described in detail in the following subsections. The detail algorithm is shown as Figure 1.

**Divide the Space According to the Static Objects**

Even the most dynamic scene has many static objects (buildings, ground shapes, etc., for the instance as city model); we can divide the space according to the static objects, similar to other visibility culling methods.

There are many methods (e.g. Delaunay division) to divide the space. To simplify the implementation, we constructed an octree according to the bounding box of the original mesh. The bounding box value of the root of the octree is the bounding box of the original mesh. Then we inserted all the original vertexes into the octree. Figure 2 shows the pseudo code of the data structure of the octree, where min_side is the minimum side of the bounding box side of a node in the octree and max_objs_of_node is the maximum object number that can be contained in a node.

---

*Figure 1. The algorithm flow.*

```c
typedef struct octree
{
    OctreeNode *root;
    float min_side;
    float delta;
    int max_objs_of_node;
} Octree;

typedef struct OctreeNode
{
    struct OctreeNode *parent;
    Coord3 centre;
    Extents boundingbox;
    int num_elements;
    OctreeNodeData *elements;
    int child_elements;
    struct OctreeNode *child;
    Boolean flag;
    float importance;
} OctreeNode;
```

*Figure 2. Pseudo code of the data structure of the octree.*
The key step is to insert the vertexes into the octree and to split the octree node. It must meet the following criteria:

- If the side of the bounding box of a node is greater than min_side, split the node into eight sub-nodes.
- If the number of the vertex contained in a node is greater than max_objs_of_node, split the node into eight subnodes.

The pseudo code of processing the octree is shown in Figure 3. Each node has its importance value. We will sort the importance values and render the objects that have the greatest importance value until the time budget is reached.

**Importance of the Objects in a Node**

The importance value of every octree node includes two parts: the importance value of the static object and the importance value of the dynamic object, as shown in equation (1).

\[
importance[i] = static_{-}Imp[i] + dynamic_{-}Imp[i] \\
0 < i < n
\]

where \( n \) is the number of nodes in the created octree.

Many existing algorithms use the distance between the viewpoint and the centres of the triangles as the criterion to select occluders (occluders are the most important objects in the scene), but our algorithm also considered the contribution of the triangles to the scene:

\[
static_{-}Imp = \sum_{i=1}^{m} weight_i \times solidity_i
\]

where \( m \) is the triangle number in the octree node.

The importance value of a static object is related to the weight and solidity of the objects in the node (see equation 2). In addition to these, the importance value of a dynamic object is also related to moving rates.

**Weight.** The weight of an object is used to represent its importance to the image quality. It tells us which object in the mesh is much more dedicated to the user’s visual sense to the scene, and which object can be removed first and will not affect the sense of the user.

We call the weight calculated in the pre-process step the static weight, a value that is different from the dynamic weight calculated in the real-time step. There are several methods to determine the weight of a triangle in an LoD algorithm. Here, we employed a simple method. Let \( Tvi \) be the set of the triangles sharing \( Vi \), then the static weight value \( Wi \) of vertex \( Vi \) and
The solidity value of triangle \( n \) can be calculated as

\[
S_n = k \times \left( 1 - \frac{\sum_{i=1}^{n_{mi}} \Delta m_i \times (\vec{v} \cdot \vec{N}_{mi})}{\sum_{i=1}^{\text{Num}} \Delta F_i} \right) \times (\vec{v} \cdot \vec{N}_n) \quad (4)
\]

where \( \Delta m_i \) is the area of triangle \( m_i \); \( n_{mi} \) is the number of the triangles that have been processed; \( \vec{v} \) is the view vector; \( \vec{N}_{mi} \) is the normal vector of the triangle \( m_i \). \( \vec{N}_n \) is the normal vector of the triangle \( n \). Num is the number of all the triangles. \( k \) is the production factor, which can be computed as equation (5).

In our implementation, we can simplify equation (4):

\[
\text{Sum} = \sum_{j=1}^{\text{Num}} \Delta F_i
\]

\[
f_1 = 0, \quad S_1 = 1
\]

\[
f_{i+1} = f_i + \frac{\Delta m_i \times (\vec{v} \cdot \vec{N}_{mi})}{\text{Sum}} \quad 0 < i < \text{Num}
\]

\[
S_{i+1} = k(1 - f_i) \times (\vec{v} \cdot \vec{N}_{mi}) \quad 0 < i < \text{Num}
\]

where \( S_i \) is the solidity value of the nearest triangle to the viewpoint.

Because equation (4) can only calculate the solidity value of continuous triangles, we can infer that for discrete triangles the longer the distance from the viewpoint to the object, the less is the production factor \( k \) (see Figure 5). The production factor \( k \) can be calculated using equation (5):

\[
k = 1 - \frac{\sum_{F \in \text{Face}} (\vec{n}_F \cdot \vec{v}) \Delta F}{\text{Width} \times \text{Height}}
\]

where Width and Height are the weight and height of the screen. \( F \) is the visible face in the surface of bounding box of the processed object. \( \vec{n}_F \) is the normal of face \( F \); \( \vec{v} \) is the view vector. \( \Delta F \) is the area of face \( F \).

By sorting the value of importance[i], we can get some triangles that have the highest value of importance[i] as the occluders. With our method, the occluders can match the following criteria:

- **Size**: Small objects will not be chosen as good occluders unless the viewer is very close to them.
- **Distance**: Since the solidity value is transferred from near triangles to far triangles, we can choose the near triangles as the occluders.
- **Opacity**: The triangles that have more possibility of occlusion by other triangles will not be chosen as good occluders.

---

**Solidity.** The solidity of a triangle is its potential for occlusion by other triangles, which is similar to the opacity used in volume rendering. For the convenience of implementation, there are some differences between our solidity value and opacity value in volume rendering. First, the solidity value used in our algorithm can be larger than one. Second, the larger the solidity value, the less the possibility that the triangle may be occluded by an other triangle.\(^6\)

The adjoining triangles are called continuous triangles. Usually, one object is composed of a set of continuous triangles. Our algorithm begins with the set of continuous triangles which is nearest to the eye position.
we divide the movement of the object into three directions: $X$, $Y$, and $Z$.

$$\text{dynamic}_{\text{Imp}} = \text{dynamic}_{\text{Imp}_x} \ast \text{dynamic}_{\text{Imp}_y} \ast \text{dynamic}_{\text{Imp}_z}$$

In the $X$ direction, we can get equation (6):

$$\text{dynamic}_{\text{Imp}_x} =$$

$$\begin{cases} 
\frac{k_x}{w} \times \left( 1 - \frac{\text{Near} \times f_x}{\text{Dist}} \right) \times x, & \frac{\text{Dist}}{\text{Near} \times f_x} \leq x \leq \frac{w}{\text{Near} \times f_x} \times \text{Dist} \\
0, & x \geq \frac{w}{\text{Near} \times f_x} \times \text{Dist} \\
1, & x \leq \frac{1}{\text{Near} \times f_x} \times \text{Dist}
\end{cases}$$

(6)

where $w$ is the width of the near plane. For instance, for a screen of 640 * 480, the width of the near plane is 640. $\text{Near}$ is the distance from the near plane to the viewpoint, $\text{Dist}$ is the distance frame object centre to the viewpoint, $f_i$ is the frame time in which the computer refreshes the screen, and $r_x$ is the movement rate of the object in the $X$ direction.

By the same method, the importance value in $Y$ and $Z$ directions can be calculated using equations (7) and (8):

$$\text{dynamic}_{\text{Imp}_y} =$$

$$\begin{cases} 
\frac{k_y}{h} \times \left( 1 - \frac{\text{Near} \times f_y}{\text{Dist}} \right) \times y, & \frac{\text{Dist}}{\text{Near} \times f_y} \leq y \leq \frac{h}{\text{Near} \times f_y} \times \text{Dist} \\
0, & y \geq \frac{h}{\text{Near} \times f_y} \times \text{Dist} \\
1, & y \leq \frac{1}{\text{Near} \times f_y} \times \text{Dist}
\end{cases}$$

(7)

$$\text{dynamic}_{\text{Imp}_z} =$$

$$\begin{cases} 
\frac{k_z}{h} \times \left( 1 - \frac{\text{Far} \times f_z}{\text{Near} \times f_z} \times z \right), & 0 \leq z \leq \frac{\text{Far} \times \text{Near}}{f_z} \\
0, & z \geq \frac{\text{Far} \times \text{Near}}{f_z}
\end{cases}$$

(8)

where $h$ is the height of the near plane. As stated in the previous example, the near plane is 480, where $r_y$ and $r_z$
are the movement rate of the object in the Y direction and Z direction respectively. \( \text{Far} \) is the distance from the far plane to the viewpoint.

**For the Dynamic Objects**

We can create a queue according to the importance value of every node. Render the objects in the node according to the descendant order of importance value. In the static scene, when a viewpoint moves only a few members in the queue would change. However, in a dynamic scene, many dynamic objects change dramatically; hence the dynamic importance values will change too. To accelerate our algorithm, we incorporate two methods: simplify the small dynamic objects and estimate the movement of occluded dynamic objects.

**Simplifying the Small and Far Dynamic Objects.** It is very important to use LoD models to replace the original models of dynamic objects because the dynamic objects will be displayed repeatedly. We use LoD levels according to the distance from the viewpoint to the centre of the object-bounding box. We can assume, when the bounding box of dynamic objects is projected onto the screen as one pixel, that the objects could be removed. The radius of the bounding sphere can be calculated using the following equation (see Figure 5):

\[
R = \sqrt{\frac{W \times H}{\text{Width} \times \text{Height}}} \times \frac{\|OP\|}{\text{OP}} \times \left(\frac{90^\circ}{90^\circ - \theta}\right)^2
\]

where \( \text{Width} \times \text{Height} \) is the screen resolution and \( W \times H \) is the size of the near plane of the viewing frustum. The area in the near plane corresponding to one pixel in the window is then \( \frac{W \times H}{\text{Width} \times \text{Height}} \). \( \|OP\| \) is the length from viewpoint \( O \) to an arbitrary point \( P \), and \( P' \) is the projected point of \( P \) in the near plane. \( \theta \) is the angle between the view centre line and \( OP \). The vertexes within the bounding sphere, whose radius is \( R \), will then be projected to the screen within one pixel. All these vertexes can be clustered to one vertex.

If all the vertexes in a node are clustered, the flag of the node is marked as TRUE, meaning we do not need to draw the dynamic objects that only occupied several pixels when it is projected to the screen.

**For the Static Scene Behind the Dynamic Objects**

As there are still many far and small triangles that cannot be clustered, we can use an object impostor to simplify rendering. An impostor is a simple polygon with an image of the complex object mapped onto it.

According to the structure of the pre-processed octree (shown as Figures 2 and 3), the bounding boxes of all the children are within the bounding box of their parents. So the children’s mesh can be replaced with their parent’s bounding box impostor. Controlling the number of the object impostor can control the simplification error of the occludee mesh.

From the root of the octree, search the octree. According to the pseudo code of Figure 9, we select some (let it be 16, which can be predicted by the computer performance: the better the computer performance, the more object impostors can be chosen) object impostors. Because occludees are small and distance nodes, they occupy very limited texture memory and can be reused with small errors.

Frame sequences contain considerable coherence. Rasterized images of objects are not discarded after each frame but are reused in subsequent frames as impostors. The reuse of rasterized object images and the duration of their validity are discussed in the following.

During movement of the viewpoint, we distinguish two worst cases: translation relative to an object and moving towards an object. As long as the maximum angle remains below a certain threshold (for example \( \alpha_{\text{screen}} \)), the rasterized object image is considered valid and can be reused. For translations of the viewpoint, consider Figure 7.

The maximum angle of error introduced by the use of an impostor occurs along the cube’s diagonal as the user...
moves normal to the diagonal. The diagonal endpoints $P_1$ and $P_2$ are overlapped in the projection from $V_1$, and will be separated from $V_2$; the error angle is $\alpha$.

For zoom of the viewpoint consider Figure 8. Assume the impostor in plane $P$ has been generated from point of view $V_1 - P_1$ appears as $P'$ on the impostor. When the user moves from $V_1$ to $V_2$, the maximum difference between the user view of the object and the impostor is bounded by the angle $\alpha$.

If $\alpha$ is less than threshold $\alpha_{\text{screen}}$, the impostor is considered valid, and can be reused.

### Experimental Results

We implemented the real-time rendering algorithm on a computer with a Pentium II 300 MHz as CPU and 128 MB memory using OpenGL. Users can change the...

---

**Figure 8.** Error angle $\alpha$ due to move-in.

**Figure 9.** Pseudo code to search for the object impostor.

**Figure 10.** Bicycle: many small triangles are used to describe the details of the water bottle and the footboard. These triangles are deleted in the right simplified model.
viewpoint in real time. In order to get the constant frame rates, the scene should be rendered in a given time budget. For example, if there are only static buildings in the scene, a slow frame rate is suitable for the computer to provide high-quality images, and vice versa. We provide examples as follows. For Figure 10, 1000 triangles are used as occluders, with about 12 frames/s walkthrough. For Figure 11, 3000 triangles are used as occluders, with about 10 frames/s walkthrough. Figure 12 is a dynamic scene with both static objects and dynamic objects (the cars in front of the buildings).

Since our algorithm is a time-critical rendering algorithm, the process is stopped when the triangle budget is reached. The rendering algorithm can assure uniform frame rate during walkthrough. We applied the algorithm to various scenes. Figure 14 shows the frame rate during walkthrough of a city (see Figure 13). Table 1 lists some data of the walkthrough processes.

Figure 11. ACME building: some small triangles on the top floor are deleted. The roof is shown with the previous frame image as object imposter.

(a) Original 17856 triangles  
(b) 3000 triangles as occluders

Figure 12. The castle. Top: 119,258 triangles (original). Bottom: 29,304 triangles. Our algorithm discards the far small triangles and the top floor of the right tower, because in the octree for static objects, the right node is less important than the central one.
Figure 13. Houston City. Top: 109,465 triangles (original). Bottom: 29,556 triangles. Our algorithm discards some triangles of the right building, and some small windows of the left front house. Compared with the static objects, more details of the dynamic objects remain, because the importance value of the dynamic object includes two parts: the static_Imp and the dynamic_Imp.

Figure 14. Frame rate during walkthrough.
During walkthrough, we search the first vertex in the octree. Only when the first vertex is changed does the solidity array change. Thus if the eye position moves within the same node of the octree, the solidity array remains unchanged. As a result, the octree can be drawn much faster, achieving 7–8 frames/s with a budget of 6000 triangles.

## Conclusion and Future Work

As scene geometry becomes complex (such as a scene with millions of polygons), and there are many dynamic objects in a scene (such as one with many cars moving in a highway), even the most advanced rendering hardware is not able to provide interactive rates. Much of real-time computer graphics has been dedicated to find ways to increase the frame rate, trading off image quality.

In this paper, we proposed a new time-critical rendering algorithm, which incorporates the techniques of LoD, visibility culling and the object impostor method. The new algorithm is an extension of our existing time-critical rendering algorithm developed by our research group,\(^{28}\) which only integrated techniques of LoD and visibility culling. In the extended algorithm, we used an object impostor to represent far objects, exploited the LoD method to simplify complex scenes, and incorporated the visibility culling method into the LoD method to discard triangles from behind to front. For high-depth scenes, we can draw the front and large triangles as soon as possible. Our algorithm is more suitable for high-depth scene walkthrough in virtual reality.

Our future work will focus on the viewpoint and view gaze prediction. We will divide the scene space into various solidity areas and improve the frame rate when the view frustum changes dramatically. In addition, we will combine our algorithm with other time-critical rendering algorithms and develop methods to render models with constant frame rates.

## ACKNOWLEDGEMENTS

This research work is supported by 973 project (grant no. 2002CB312100), NSF project in China (grant no. 60083009), Excellent Youth NSF of Zhejiang Province (grant no. RC40008) and TRAPOYT Program in Higher Education Institution of MOE, PRC.

## References


Authors' biographies:

Mingmin Zhang is an associate professor in the Computer and Engineering Department of Zhejiang University. She studied for her Bachelor degree at the Computer Science Department, Nanjing University in 1990, and for her Masters degree at the Computer Science and Engineering Department, Zhejiang University in 1995. She has published more than 10 papers in international journals, national journals, and conferences in recent years. She is the co-author of two books related to computer graphics and multimedia. Her research interests include virtual reality/virtual environment, multi-resolution modeling, real-time rendering, distributed VR, visualization, multimedia and image processing.

Zhigeng Pan is a professor in the Computer and Engineering Department of Zhejiang University. He received his Bachelor degree and Masters degree from the Computer Science Department, Nanjing University in 1987 and 1990 respectively, and his PhD from Computer Science and Engineering Department, Zhejiang University in 1993. He has published more than 70 papers on international journals, national journals, and conferences in recent years. He is the co-author of two books related to computer graphics and multimedia. His research interests include distributed VR, multimedia, multi-resolution modeling, real-time rendering, virtual reality/virtual environment, visualization and image processing.

Heng Pheng-Ann received his BSc from the National University of Singapore in 1985. He received his MSc (CS), MArt (Applied Math) and PhD (CS) from Indiana University, USA in 1987, 1988, 1992 respectively. He is currently a professor at the Department of Computer Science and Engineering at CUHK. In 1999, he set up the Virtual Reality, Visualization and Imaging Research Centre at CUHK and serves as the Director of the centre. He is also the Director of the CUHK Strategic Research Area in Computer Assisted Medicine, established jointly by the Faculty of Engineering and the Faculty of Medicine in 2000. His research interests include virtual reality applications in medicine, scientific visualization, 3D medical imaging, user interface, rendering and modeling, interactive graphics and animation.