Real-Time and interactive browsing of massive mesh models

Xian Wu, Yan Huang*

School of Computer Science and Technology, Shandong University, Jinan, China

Received 1 March 2014, www.tsi.lv

Abstract

We present an efficient method for out-of-core construction and real-time interaction of massive mesh models. Our method uses face clustering on an octree grid to simplify and build a Level-of-Detail (LOD) tree for the model. Each octree node leads to a local LOD tree. All the top layers of the local LOD trees are combined together to make the basis of the global LOD tree. At runtime, the LOD tree is traversed top down to choose appropriate local LOD trees given the current viewpoint parameters. The system performance can be dramatically improved by using hierarchical culling techniques such as view-frustum culling and back-face culling. The efficiency and scalability of the approach is demonstrated with extensive experiments of massive models on current personal computer platforms.

Keywords: massive mesh model, out-of-core, level-of-detail, mesh simplification, culling

1 Introduction

3D mesh models are dominant in computer graphics. Applications employing meshes include movies, games, computer aided design, simulation, art and history etc. Today with the fast development of 3D acquisition, modelling and simulation technologies, we have much more complex and accurate mesh models. For instance, mesh models of gigabytes size are not uncommon nowadays.

In the last several decades, the performance of CPU and GPU has improved tremendously. However, the memory bandwidth especially disk bandwidth grows much slower. Therefore the bottleneck lies on the fact that our processor has to wait for the data stored on disk. There is a wide range of simplification methods and multiresolution models have been proposed to solve this problem, but most of them fail to perform either scalable simplification or efficient viewpoint dependent visualization of massive models.

Our contribution of this work is to find a solution for real-time and interactive browsing of massive models on personal computer platforms. In human visual system, the sensitivity to details is inversely proportional to the distance between the view point and the observed point. Thus we can construct a hierarchy of multiple resolution representations of the original model. At run-time, we dynamically and adaptively select the needed level-of-detail (LOD). We build LOD trees through hierarchical face clustering. Our algorithm reveals an out-of-core nature, since we use an octree data structure to partition the model and build the local LOD tree for each octree node. Then we combine all the top layer of the local LOD trees and take it to build the global LOD tree. When in real-time browsing, we mainly interact with the global LOD tree and use it as an entry point to access the corresponding local LOD trees. Frustum culling and backface culling were used to accelerate the interaction speed. By combining a large set of technologies, our system shows good performance, better visual results, and a highly scalable architecture.

2 Related work

The research on interactive processing of complex models has over 30 years' history [1, 2]. The traditional approaches focus on how to reduce data complexity, manage data organization and utilize the new hardware technology [3]. In recent years, due to the widespread use of massive data sets, no single method could provide satisfying solution. A number of state-of-the-art systems utilizing different sets of technologies have been proposed to tackle this issue.

LOD based mesh visualization. LOD is useful because it is able to adjust the appropriate approximation given some viewing parameters [4]. For example, the Quick-VDR system [5, 6] represents the model as a clustered hierarchy of progressive meshes (CHPM) [7]. It uses the cluster hierarchy for coarse-grained selective refinement and progressive meshes for fine-grained local refinement. The Adaptive TetraPuzzles (ATP) system [8] uses a regular conformal hierarchy of tetrahedra to spatially partition the model. Each tetrahedral cell contains a precomputed simplified version of the original model, which is constructed off-line during a fine-to-coarse parallel out-of-core simplification of the surface contained in diamonds.

Real-time ray tracing. Some other systems diverge from the normal rasterization approach by incorporating a real-time ray tracing algorithm. By using spatial

* ***Corresponding author*** e-mail: yan.h@sdu.edu.cn

indexing, ray queries can be determined in logarithmic time. The OpenRT real-time ray tracer [9, 10] uses a two-level kd-tree hierarchy as spatial index. It also incorporates a custom memory management subsystem to deal with scenes larger than physical memory.

The volumetric approach. All the above systems assume that the multiresolution models are triangle-based or point-based. But the Far Voxels [11] system adopts a volumetric approach which uses small volume clusters to represent local datasets. By using a coarser granularity in the LOD structure, the cost of data management, traversal and occlusion culling can be reduced dramatically. Tian proposes Adaptive Voxels system [12] based on the Far Voxels system to make use of a novel adaptive sampling method to generate LOD models.

3 Mesh simplification

Mesh simplification is the cornerstone of LOD tree construction. The decimation methods can be classified into two major categories: clustering and incremental decimation. Vertex clustering and face clustering are two major clustering methods. The incremental decimation can have operators such as vertex removal, edge collapse, half-edge collapse etc. We choose face clustering to do mesh simplification for several reasons. First, it is much more efficient to use a clustering algorithm than an incremental decimation one. Second, face clustering provides better visual results than vertex clustering. Third, it is natural to pick face region as the unit not only in LOD tree construction but also in the real-time and interactive browsing of the model.

Suppose the initial mesh model contains N triangles. The overall framework of our algorithm is as follows:

```
Select K representative faces randomly
While(true)
{
    Region Growing
      If (termination criterion)
        Break;
      Update K representative faces
}
Face merging
Edge merging
```

FIGURE 1 Face clustering algorithm

We use a k-means based clustering to do the region growing process. When the K clusters are settled, we merge all the triangles inside one cluster into one super face. We call it super face because it is bounded by the boundary edges and is usually not flat. The effect of face clustering on an example mesh is demonstrated in Figure 2. The original mesh, the clustering result and the face merging result are shown in Figure 2. Looking at Figure 2c carefully, we find that two adjacent super faces normally shares more than one edge. This is a subtle aspect which can affect the overall performance of the whole system. Not significantly in this picture, but imagine if we have a model of millions of triangles and clusters it into thousands of super faces. Then we'll see large super faces with lots of small edges jagged together.

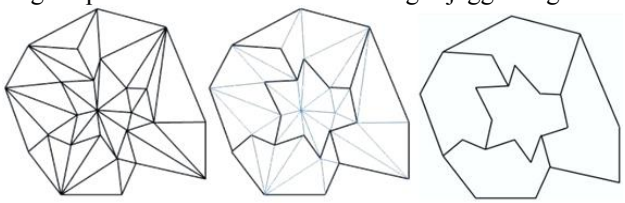

FIGURE 2 Face clustering example

The way to solve this is to do edge merging. We merge those edges that are shared by two adjacent super faces to ensure that only on edge exists between two adjacent upper faces. The process is described as follows: After merging all the triangles, we mark each vertex which is shared by three or more than three super faces as an anchor vertex. Since we need at least three vertices to determine a face region, we require every super face to have at least three anchor vertices. If a super face does not meet this requirement, we just randomly pick a certain number of non-anchor vertices to be anchor vertices. Finally, the boundary edges of super faces are determined by those anchor vertices. Sequentially connecting those anchor vertices will lead to "boundary-straightened" super faces (see Figure 3).

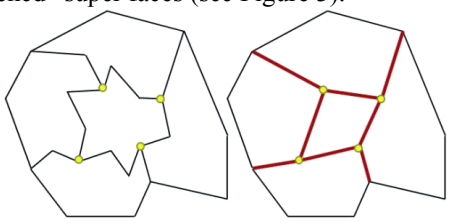

FIGURE 3 Edge merging

4 Multiresolution model

In this paper, we build a Level-of-Detail tree based interactive browsing system of the massive mesh model. We propose a novel approach to LOD tree construction

We use an octree structure to partition the original model to support out-of-core processing. Each local LOD tree corresponds to an octree partition region. We control the octree depth to make the memory usage of each local LOD tree construction under a predefined upper limit. We combine all the top layer of local LOD trees and take it as the bottom layer to construct the global LOD tree. We control the height of all the local LOD trees so that the construction of the global LOD tree can fit in memory. The overall structure of the LOD hierarchy is shown in Figure 4.

By organizing our data in this way, we sort of make a distinction between model's overall look and model's local region display. We can use the global LOD tree to support interactive display of the whole model. When the user is interested in some particular area, the corresponding local LOD trees can be loaded into memory to explicitly show the focused region. The global LOD tree serves as an entry point to find and load the

local LOD trees. Thus the global LOD tree plays a central role in the whole interaction period. During the loading and recycling of the local LOD trees, we can use scheduling policies based on user viewpoint to optimize the overall performance.

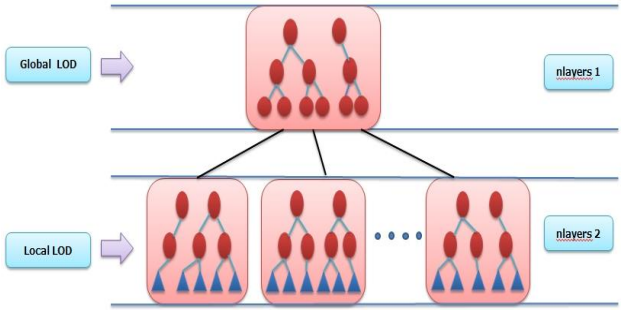

FIGURE 4 The structure of Local and Global LOD trees

5 Construction and serialization of LOD trees

5.1 OCTREE PARTITIONING

We use an octree data structure to partition the original model. We distribute the vertices and triangles of the model according to the following rules:

1) To vertex: we distribute it to its corresponding octree cell.
2) To triangle: since each triangle has three vertices, there are generally three cases:
   - If three vertices all fall into the same cell, then the corresponding cell is the one contains this triangle.
   - If two of them fall into the same cell, then the corresponding cell is the one contains this triangle.
   - If the three vertices belong to three different cells, we use the first vertex's containing cell to have the triangle.

Figure 5 shows an example for the 2D analogy of octree partition. The red number gives each vertex it's appearing order in the triangle, and the blue number inside each triangle represents the number of the cell containing this triangle. During the partitioning process, we also compute each triangle's area, barycenter, normal and store them in disk files for later use.

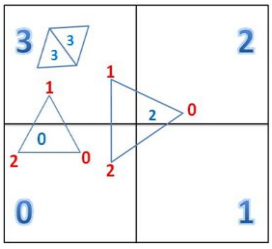

FIGURE 5 Example of 2D analogy for Octree partition

5.2 LOCAL LOD TREE CONSTRUCTION

Once we have done octree partitioning, we construct a local LOD tree for each octree node. Taking all the triangles of one octree node as input, we use K-means based face clustering algorithm to obtain a new simplified representation made up of K super faces to approximate the original one. Then we take those super faces as input and use face clustering again to get an even more simplified model. By continuingly doing so, we will get a hierarchy of multiresolution models. Each super face is derived by merging the sub faces it contains. We include this inheritance relationship to build a Level-of-Detail tree structure.

5.3 GLOBAL LOD TREE CONSTRUCTION

For each local LOD tree, we only keep the top layer in memory. The top layer is the simplified representation of the original model in the corresponding spacial region. After constructing local LOD trees for all the octree nodes, we combine all the top layers of those local LOD trees to form a complete simplified representation for the whole model. And we use this layer as the bottom layer to construct the global LOD tree.

There exists some freedom in choosing the octree depth and the height of the local LOD tree. But each octree node must contain at most some maximum number of triangles to make sure that the memory is sufficient in local LOD tree construction. Also, the height of the local LOD tree must be high enough so that when all the local LOD trees' top layers were combined together, there remains sufficient memory to build the global LOD tree.

5.4 SERIALIZATION OF LOD TREES

We need to design a format to represent a LOD tree, such that it can be stored in disk, loaded to memory and interpreted efficiently for real-time viewing. This is achieved through serialization. It is the process of converting an object into a writable format that can be persisted or transported. The complement of serialization is deserialization, which restores an object from a stream. In real-time browsing, we need to load the local LOD trees frequently, so finding an efficient way to do serialization/deserialization is critical.

We use a DFS-based approach for its simplicity to storing the tree structure and it only requires one scan and a small extra stack to do deserialization.

6 View-dependent rendering

We will use the constructed LOD trees to support real-time and interactive browsing of the massive mesh model. The browsing system first loads the global LOD tree. Each level of the global LOD tree represents a certain degree of approximation to the whole model. Given the current viewpoint, we use the model's projected screen space area to choose the suitable level to render. Because the Global LOD tree is resident in memory, we can efficiently doing the traversal and rendering. When the user moves its viewpoint to some particular region of the model, the node of the Global LOD tree cannot provide sufficient accuracy. We have to load the corresponding local LOD tree which represents

the viewer's interested region and select certain level according to the projected screen space area.

Due to the limited size of field-of-view, when we are focused on some particular part of the model, the other parts of the model either are out of the sight or are far enough. In practice, we only need to load several Local LOD trees at the same time. So we can allocate a buffer to contain the currently loaded Local LOD trees. If the viewpoint moves and the buffer is full, we can remove old Local LOD trees and load new Local LOD trees. Because viewers always move their sight in a continuous way, we can optimally remove the Local LOD tree that is furthest to our current viewpoint.

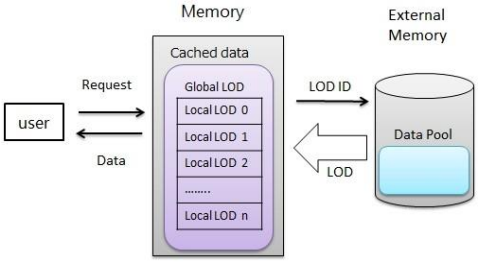

FIGURE 6 Data access framework

Figure 6 shows the whole system's data access framework. The global LOD tree is always in memory and acts as the entry point for accessing the local LOD trees. We have a buffer of size n to contain the currently loaded local LOD trees. A scheduler loads local LOD trees from the disk and removes local LOD trees from the memory if the buffer is full

6.1 VISIBILITY CULLING

Determining the visible parts of the scene is an important graphics problem. It is both inefficient and incorrect to render objects that are unseen. We have to remove those surfaces that are hidden from the viewer. Visibility culling [13] is the process of computing the visible subset of a scene. There are typically three culling techniques: view-frustum culling, back-face culling and occlusion culling. Occlusion culling is mainly used in scenes that contain many models. Because our focus here is on single massive mesh model, we only use view-frustum culling and back-face culling to accelerate our algorithm.

6.2 VIEW-FRUSTUM CULLING

Figure 7 shows a typical camera setup. Models or parts of models outside the frustum cannot be seen by the viewer. Because our LOD tree node represents a super face, we need to test whether this super face is outside the frustum.

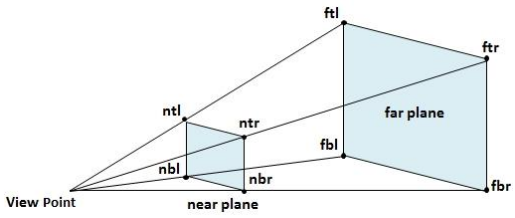

FIGURE 7 View-frustum

In practice, we use the bounding box approach. When we build the LOD tree hierarchically, we also compute the combined bounding box for each node from its children bounding box. This process is quite efficient and visibility test of box against frustum is also easy.

6.3 BACK-FACE CULLING

For a solid opaque object, the back of it is hidden from the viewing ray. Culling primitives that lie on the backs of objects can almost reduce half of the scene geometry to be rendered. Here, we use a clustered backface culling algorithm based on the normal cone [14]. The normal cone is represented by a central cone normal and a cone angle.

Like in Figure 8, we compute the normal cone for each tree node. To every current viewpoint, we also have a viewing normal cone. By comparing these two normal cones we can determine whether the node is back facing or not as show in Figure 9. To find an exact normal cone for a surface patch is a computational geometry problem and is rather slow. Instead, we use a bounding box approach which is fast and approximates well to the exact normal cone. The idea is that a bounding box of all the normal Ni is constructed. The cone normal is defined to be the vector from the origin to the centre of the bounding box. The direction from the origin to the eight corners of the bounding box will have eight angles with the cone normal. We take the largest one as the cone angle.

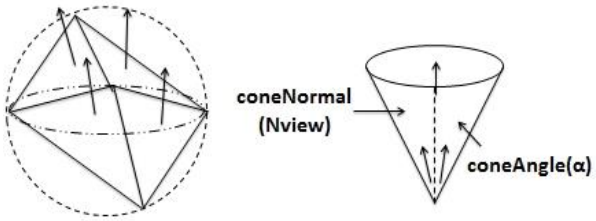

FIGURE 8 The left subfigure shows one node containing 4 triangles, the right subfigure shows the corresponding normal cone

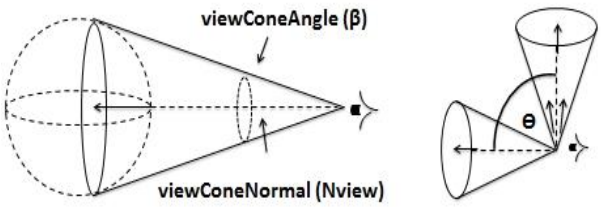

FIGURE 9 The left subfigure shows the viewing normal cone, the right subfigure shows its relation with one node's normal cone

7 Experimental results

7.1 PREPROCESSING

All the experiments were done on a Lenovo PC with 2.83GHz Intel Core 2 Quad CPU Q9500 processors, 4.0 GB of RAM. We have tested a number of massive models. The Figure 10 shows the four models we used for testing our system

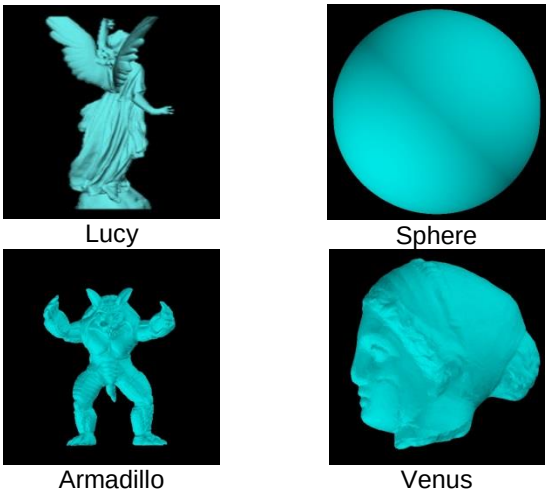

FIGURE 10 Four test models

From Table 1 and Table 2, we can see that our tested models contain tens of millions of and even hundreds of millions of triangles. The model venus has size of over 11GB. The octree partitioning process takes four and a half hours. The construction of LOD takes about six hours. The time used for pre-processing is acceptable since we only have to do it once. Once these LOD trees are built, we simple use these structures in later real-time browsing.

Our algorithm uses face clustering to do mesh simplification. Below, we show some simplified representations of several models in their global LOD tree layers in Figure 11.

From Figure 11, we can see that the simplification algorithm is effective even we simplify the model to only several hundreds of nodes. We compare our algorithm with the Adaptive Voxels system proposed by Tian [12]. Because our algorithm mainly deals with manifold surfaces, we test our system using different models. But we can still compare the two algorithms when the models' sizes were at the same size level.

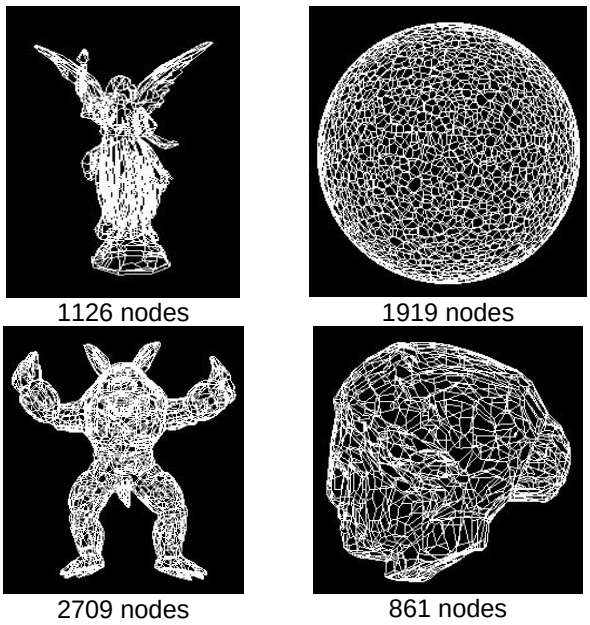

FIGURE 11 Some simplified representations of test models

Table 3 gives the numerical comparisons of our method and the Adaptive Voxels method. Our tested model venus is roughly 77% of the size of the Boeing 777 model. The pre-processing time used in our method is about 25% of the Adaptive Voxels method. The disk space usage is also much smaller in our method. Further, our algorithm is quite scalable and can be easily made parallel. The octree partitioning process and the local LOD tree construction can use parallel computing to largely accelerate the processing. But the Adaptive Voxels method uses BSP tree to construct the scene graph. The structure is intensely correlated, so it is not suitable for parallel computing.

TABLE 1 Numeric results of pre-processing

Model	#Vertices	#Triangles	Size (GB)	Maximum Memory (MB)
Lucy	14,027,872	28,055,742	1.05	900
Sphere	31,457,282	62,914,560	2.55	180
Armadillo	44,280,834	88,561,664	3.48	700
Venus	135,430,146	270,860,288	11.49	400

TABLE 2 Numeric results of pre-processing

Model	Octree		LocalLOD		GlobalLOD	
	Time(min)	*depth*	*layers*	*Time(min)*	*layers*	*Time(s)*
Lucy	27.82	2	3	43.63	4	0.35
Sphere	61.96	3	3	84.93	4	7.19
Armadillo	85.59	3	4	121.81	4	2.16
Venus	273.24	4	5	370.40	4	1.01

TABLE 3 Numerical comparisons

	Model	Faces (Million)	Pre-processing Time (min)	Size (GB)
Adaptive Voxels	Boeing 777	350	2,729	49.4
Our Method	Venus	270	643	28.2

7.2 REAL-TIME AND INTERACTIVE BROWSING

We devised a set of inspection paths to verify the real-time performance of our system. We include the typical tasks such as rotation, translation and scale. We also consider rapid changes from overall scenes to some

specific model regions. The window size is 1024×1024, the pixel tolerance for each projected node size is 1 pixel. The numeric results are shown in Table 4. The column 4 gives the system setup time. And the last two columns give the average frame per second without and with culling techniques.

From Table 4 we can see that all the average numbers of FPS with culling are above 18. In reality, humans take actions normally three to five times per second. Our system satisfies the real-time interaction rates. The last two columns of Table 4 shows that, by using hierarchical culling techniques, the system can save almost two fifth of the time. The average FPS of the Adaptive Voxels system is 8. Our system shows a significant improvement in the real-time performance. Another advantage is that our system is particularly efficient at viewing the local regions of the model since we use an optimal scheduling policy to load and recycle the local LOD trees thus minimizing the data access time.

TABLE 4 Numeric results for real-time rendering

Model	Resolution	Pixel Error	Setup Time	Avg FPS (No Culling)	Avg FPS (With Culling)
Lucy	1024×1024	1	1.57	16	24
Sphere	1024×1024	1	2.57	13	22
Armadillo	1024×1024	1	3.12	13	21
Venus	1024×1024	1	4.86	12	18

8 Conclusions and future work

In this paper, we build an LOD tree based real-time and interactive browsing system for massive mesh models. By using octree partition and face clustering, we construct the local LOD trees and the global LOD tree to provide an efficient out-of-core data access framework. We have tested on a set of massive mesh models and obtained good experimental results.

In the future, improvements could be made to our system. For instance, our current system has not exploited the parallelism inherent in components of the algorithm, such as octree partitioning and local LOD tree construction. We can also use data compression techniques to further reduce the data to be stored in disk and to be loaded to memory.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant No. 61303083), NSFC Joint Fund with Guangdong under Key Project (Grant No. U1201258) and the Scientific Research Foundation for the Excellent Middle-Aged and Youth Scientists of Shandong Province of China (Grant No. BS2011DX017).

References

[1] Kasik D J, Manocha D, Slusallek P 2007 *IEEE Computer Graphics and Applications* **27**(6) 17-19
[2] Gobbetti E, Kasik D, Yoon S 2008 Technical strategies for massive model visualization *Proceedings of the 2008 ACM symposium on Solid and physical modeling ACM* 405-15
[3] Dietrich A, Gobbetti E, Yoon S-E 2007 *IEEE Computer Graphics and Applications* **27**(6) 20-34
[4] Luebke D P 2003 Level of detail for 3D graphics Morgan-Kaufmann
[5] Yoon S E, Salomon B, Gayle R, Manocha D 2004 *IEEE Computer Society* **67**(14) 131-8
[6] Yoon S E, Salomon B, Gayle R, Manocha D 2005 *IEEE Transactions on Visualization and Computer Graphics* **11**(4) 369-82
[7] Hoppe H. 1996 Progressive meshes *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques ACM* 99-108
[8] Cignoni P, Ganovelli F, Gobbetti E 2004 *ACM Transactions on Graphics (TOG)* **23**(3) 796-803
[9] Wald I, Purcell T J, Schmittler J, Benthin C, Slusallek P 2003 Realtime Ray Tracing and its use for Interactive Global Illumination *In Eurographics State of the Art Reports*
[10] Wald I, Dietrich A, Slusallek P 2004 An Interactive Out-of-Core Rendering Framework for Visualizing Massively Complex Models *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)* 81–92
[11] Gobbetti E, Marton F 2005 *ACM Transactions on Graphics* (TOG) **24**(3) 878-85
[12] Fenglin T, Hua W, Dong Z, Bao H. 2010 Adaptive voxels: interactive rendering of massive 3D models *The Visual Computer* **26**(6-8) 409-19
[13] Akenine-Möller T, Haines E, Hoffman N. 2011 Real-time rendering [3rd], *CRC Press* **14** 660-70
[14] Shirmun L A, Abi- Ezzi S S 1993 The cone of normals technique for fast processing of curved patches *Computer Graphics Forum Blackwell Science Ltd* **12**(3) 261-72

Authors

Xian Wu, born in 1988, Huainan, Anhui, China

University studies: master student, school of computer science and technology, Shandong University.
Scientific interest: computer graphics.

Huang Yan, born in 1974, Tongling, Anhui, China

Current position, grades: associate professor in the school of computer science and technology, Shandong University.
Scientific interest: mainly in large scale 3D data visualization, intelligent multimedia data analysis.