A Hardware Accelerated Algorithm for Terrain Visualization

Mao-Jin Xie and Wei-Qun Cao

School of information science and technology, Beijing Forestry University,
Beijing 100083, China
tyxxyhm@hotmail.com, weiqun.cao@126.com

Abstract. In recent years, rapid development of graphics hardware technology made it possible to render a large scale model in real-time. In this paper, we present a hardware accelerated algorithm for large scale terrain model visualization based on the ROAM (Real-time Optimally Adaptive Meshes) algorithm to create LOD models. GPU programming is therefore employed to calculate the vertices' transform, normal vector, texture coordinate, texture sampling and fragment lighting, and to accomplish terrain rendering. Experimental results indicate that the presented algorithm works efficiently for real-time roaming of large scale terrain.

Keywords: GPU Programming, Terrain Visualization, ROAM.

1 Introduction

Due to the technological developments in radar, satellite and remote sensing, it is increasingly convenient to acquire massive terrain data. Meanwhile, the application of terrain data becomes more extensive in many fields such as 3D games, Virtual Reality, GIS, and computer simulation. However, constrained by hardware as well as the geometric complexity of terrain models, massive terrain data visualization remains as one of the most challenging problems in computer graphics.

In order to render large scale terrains rapidly with existing hardware, establishing various LOD(Levels of Detail) models became one of the most important and widely used methods since Clark[1] proposed the original model in 1976. As for the hardware, the development depends mainly on the capability improvement of graphics card. In addition, parallel computing and CPU multimedia instruction sets, such as 3DNow and SSE serial, are applied to enhance the 3D graphic rendering performance in CPU. In recent years, the emergence of and rapid progress in Programmable Graphic Processing Unit (GPU) provide strong supports for the fast rendering of complex graphics. Especially with the appearance of high level shading language, GPU programming based applications are becoming increasingly popular.

Static LOD algorithms [2, 3, 4, 5, and 6] usually generate respectively a set of models of different detail levels and select the one as simple as possible while with proper visual effects during real-time rendering. This approach works for some specific applications, but easily causes memory bottleneck for terrain data both because

C. Stephanidis (Ed.): Universal Access in HCI, Part III, HCII 2009, LNCS 5616, pp. 271–280, 2009.

the data themselves are of massive size and because, multiple models with different levels of detail exacerbate the burdens of storage. Moreover, switches among models of different detail levels could result in pop-up visual defects. Therefore, additional work for smooth transition is necessary.

The continuous level of detail (CLOD) algorithm employs view-dependent tessellation approach to generate continuous resolution models according to the error threshold in world-space and screen-space respectively. Continuous quad tree [7] LOD based on regular triangle mesh was proposed by Linstrom et al. A year later, Duchaineau et al. proposed the Real-time Optimally Adaptive Meshes (ROAM), one of the most widely used algorithms. CLOD needs no pretreatment but generates a model of proper resolution in real-time according to the movement of viewpoints. Therefore, compared with static LOD models, CLOD saves significant memory space. Moreover, this algorithm is compatible with commonly used regular mesh terrain data.

For CLOD [9], the estimation of model simplicity for multi-resolution models mainly depends on screen-space error metric or its other form—the roughness of the terrain. The screen-space error [10] is measured according to the projection principle. Du et.al [9] proposed a residual energy principle, which works well for the area far from the camera, from the perspective of energy propagation. Combined with the screen-space error principle, the residual energy principle is used to generate the view-dependent multi-resolution terrain model.

On the other hand, the speed of GPU technology development has been over three times faster than that of CPU supersession since it was originally proposed by NVIDIA Corp in 1999 [11]. More and more researchers start to use GPU to accelerate their algorithms.

Losasso and Hoppe presented a LOD algorithm that employs nested regular grids on geometry clipmaps [12] for terrain rendering. The algorithm provides a GPU friendly LOD framework that takes full advantage of the speed of concurrent consumer class GPUs, and achieves great success in the speedup of rendering. However, the algorithm is less satisfactory in compression and the error control.

Jens Schneider [13] improves the Geometry clipmaps through improving CLOD presentation with geometry morph and the support of texture. Subdivision in the threshold of user defined screen-space error and world-space error is therefore restricted to ensure rendering in high quality. The algorithm optimizes geometry filtering to anti-aliasing. To acquire wide-band efficiency, the algorithm first transfers the simplified mesh sets with discrete resolution, and then these discrete mesh models are interpolated and rendered on GPU to generate continuous resolution models.

Clasen.M.et al [14] applies GPU based geometry clipmaps on spherical terrain data visualization. It calculates sample position on terrain height field according to vertex shift and viewpoint parameter through mapping texture coordinate.

This paper combines the traditional ROAM algorithm and GPU programming for terrain visualization.

2 Overview

This paper presents an improved ROAM algorithm that raises terrain rendering efficiency through utilizing current consumer class GPUs. To simplify algorithm description, we first give three definitions as follows.

- *Desired Triangle Number*: predefined triangle number generated in each frame, for example: 20000.
- *Frame Differential*: the difference between the Desired Triangle Number and the triangle number generated in the previous frame.
- *Frame Threshold:* the error threshold in the current frame. It is adjustable dynamically according to the triangle number generated in the current frame.

Our algorithm uses nonuniform sampling on the terrain data set according to the Frame Threshold. The algorithm constructs ROAM meshes in real-time and thereafter renders and outputs with GPU.

3 Implementation

3.1 Loading of Terrain Data Sets

Because terrain data sets are typically massive, we employ the mechanism of file mapping in Windows operation system to solve the memory storage problem. File mapping mechanism is the solution provided by Windows for reading, writing and sharing huge files. It uses high speed page and supports as huge as 4GB-sized files. File mapping is efficient and easy to manipulate.

3.2 Dynamic Adjustment of Frame Threshold

After data loading, the algorithm dynamically adjusts the Frame Threshold according to the Frame Differential. If Frame Threshold is kept adjusting whenever the Frame Differential is not zero, some parts of the meshes will flicker after roaming is stopped for a period of time because the auto adjustment of Frame Threshold makes Frame Differential close to zero and the Frame Threshold will keep oscillating in a small range around the balancing point. To avoid this flickering, our algorithm employs hysteretic treatment with which the Frame Threshold is not adjusted unless the Frame Differential exceeds a suitable range R and a coefficient S is multiplied to control the speed of Frame Threshold adjustment. The number of triangles generated every frame is therefore relatively stable. The constant S and R are experimental values whose relatively ideal values should be acquired through experiments in advance. Generally speaking, R should neither be too big nor too small (in our experiments, 500 is suitable). If R is too big, it will cause a severe delay for the frame adjustment rate, which will turn down the fps or generate too few triangles in each frame and decrease the image quality. On the other hand, too small value results in mesh flicker. The value S is in connection with R. If S is too big, the generated image may have slight pop up, or it will prolong the adjusting time. In our experiments we chose 50 and can hardly feel any visual sensory pop up. Our experiments show that if R and S are chose properly, the algorithm can generate more fluent and higher quality real time video.

3.3 Terrain Blocking

In 1998, Hoppe et al [15] employed View Dependent Progressive Meshes (VDPM) in terrain rendering and realized large-scale terrain roaming in real-time. The idea of

blocking in that algorithm is of great significance to large scale terrain roaming. Blocking is advantageous for clipping efficiency, data loading, texture mapping, and parallel rendering. It also makes the algorithm more adaptive for non-square terrain data.

In order to make the blocks seamless, we overlap the edges of each block. For instance, if each block is 64 by 64, the actual data will be 65 by 65. So all sides in each block have a row of data overlapped, except for the outside edges on the brinks of the whole terrain. Blocking and Overlapping are illustrated in Fig. 1.

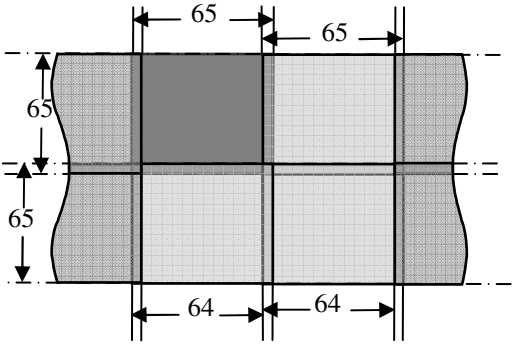


Fig. 1. Terrain Blocking and Overlapping

3.4 Frustum Pre-culling

To alleviate data processing in the first step, we make a view frustum pre-culling before the ROAM algorithm begins to run. Frustum Pre-culling is based on the terrain blocking. Taking the specialty of terrain roaming into consideration, the view region of the frustum can be simplified into a triangle region that is projected by left side plane, right side plane and far clip plane of the frustum. A coarse clip result is obtained. Therefore we flag each block into visible or invisible. Frustum pre-culling is illustrated in Fig. 2, in which the light color blocks are visible.

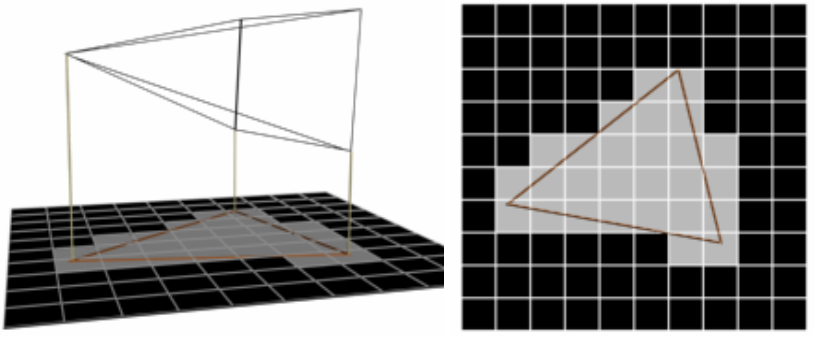


Fig. 2. Frustum Pre-culling

3.5 GPU Based Terrain Rendering

Calculating the Normal. We need the normal information on each vertex to calculate lighting on the models. Traditional approach is to pre-calculate the normal information and save it as a normal map from which sampling is carried out. But if the terrain data are large-scale, a storage bottleneck will severely restrain the terrain size to be visualized. For example, a terrain of 8192*8192 has 64M vertex. With 4 Bytes to represent a float, the normal map will have a size of 64M*3*4=768M.

Another approach is to calculate the vertex normal on the fly by CPU. But the cross product and normalization of the vectors involve multiple basic arithmetic operations and square root, which consumes considerable CPU periods. If 10 000 triangles are needed for each frame and each vertex normal is approximated on average with four triangle normals around it, there will be 120 000 operations which would easily generate calculation bottleneck.

Our algorithm calculates normal information in real-time with GPUs. It efficiently utilizes the float point vector performance of GPUs. The calculation can be executed conveniently with the cross product, normalization and linear interpolation functions and texture sampler texRECT provided by the Cg shading language.

Taken the height map of the terrain as an illumination texture, the algorithm employs the texRECT sampler to sample on the height map and obtains the height values of four neighboring vertexes. The vertex normal is calculated based on these five vertexes. Experimental results indicate that compared with CPU calculation, this processing significantly improves the performance efficiency. In our experiments, we acquired about 15 fps improvement. The frame rate also becomes more stable as the fps rises.

Lighting on the Ground. The ground lighting is calculated with the Phong model. Given that programmable GPUs can readily adjust graphics generation and terrains usually do not have specular reflection, the algorithm disregards specular terms to accelerate the rendering.

4 Experimental Results

The experimental results and timings are illustrated in Fig. 3 and Fig. 4, and summarized in Table 1 and Table 2. All timings were done on a computer with P4D 2.8GHz CPU, 1GB DDRII RAM, a GeForce 7800GT GPU with 256MB and a standard 160GB SATA hard disk. All datasets were rendered to a 1024×768 view port. The developing environment is Microsoft Visual C++6.0 with service pack 5.0 and Cg compiler version 1.50. The operation system was Microsoft Windows XP Pro. (Version 2002) with Service pack 2.0.

To explore the effectiveness of GPU acceleration, we partitioned the Desire Triangle Number into 9 levels and compared the frame acceleration rates on 5 terrains of different sizes respectively with GPU employed group A and non-GPU group B. The statistic results are demonstrated in Table 1 and Table 2, and illustrated in Fig. 3.

Fig. 3 illustrates two frame rate curves derived from the controlled experiments that employed the same ROAM algorithm. The frame rate data are gathered in a 30-second roaming process after roaming 100 seconds, the purpose of which is to ensure

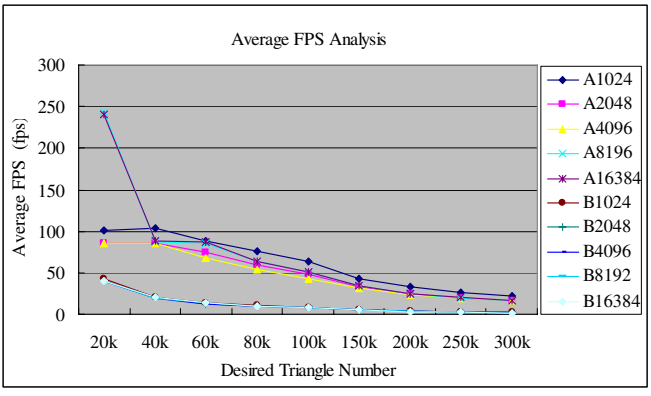


Fig. 3. Average Frame Rate Analysis

adjusted sufficiency. The testing numbers are coded as follows: A represents GPU employment, B represents non GPU employment, and numbers represent the sizes of terrains. E.g., A1024 represents GPU acceleration with terrain size of 1024×1024.

In Fig. 3, all 5 curves in group A are above the 5 curves in group B. The average frame rates in each group and the gain factor of acceleration in group A are demonstrated in Table 1 and Table 2 respectively. In general, the gain factor is positively correlated with Desire Triangle Number, which implies that the computation power of GPU programming is significant.

Fig. 3 indicates that the average frame rate is negatively correlated with the Desire Triangle Number. In general, average frame rate is not correlated with terrain size, which implies that the improved ROAM has significant practical value for large-scale terrain visualization. Appropriate Desire Triangle Number can be decided adaptive to specific hardware profiles to balance between fps and visual sensory.

To test the effects of terrain roughness on the proposed algorithm, a relatively smooth terrain of 8192×8192 was used for another 5 testing, the results of which illustrated in Fig. 4 were compared with the data already gathered. The average frame rate curves in Fig. 4(a) are almost coincident, which indicates that roughness does not have tangible impacts on the average frame rate. As illustrated in Fig. 4(b), the frame rate variance of rough terrains is in general greater than that of smooth terrains. This implies that roughness to certain degree affects the stabilization of the frame rate. Some images of relatively rough and smooth terrains are demonstrated in Fig. 5.

The roughness of terrain affects the visual sensory and even leads to pop-ups when the Desire Triangle Number is relatively small. With the same Desire Triangle Number, models of rough surface do not appear as good as that of smoother surface. This is quite reasonable since more polygons are needed to approximate a more complex surface. Fig.5 indicates that with the same Desire Triangle Number of 20,000, the rougher model in Fig.5(a) is more angulate than the smoother in Fig.5(b).

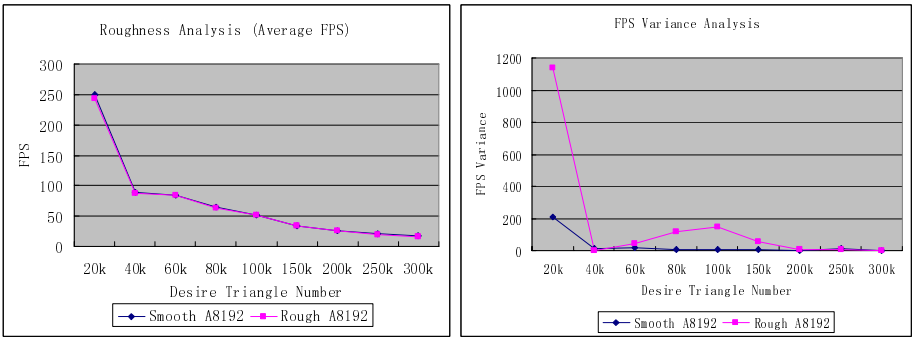
For a terrain as rough as that in Fig. 5(a), regardless of frame rate and image quality, a Desire Triangle Number of 60000 is sufficient in most real-time applications according to our test result. While for a terrain as smooth as that in Fig. 5(b), a Desire Triangle Number of 20000 is enough. Fig. 6(a) is a screen shot and Fig. 6(b) is the LOD mesh of Fig. 6(a).

Table 1. Average frame rates gathered in each experiment

DTN[1] / TG[2]	20k	40k	60k	80k	100k	150k	200k	250k	300k
A1024	101.47	103.1	88.6	75.8	63.5	43.4	33.8	26.9	22.6
A2048	85.694	85.89	74.5	60	47.8	33.2	24.4	19.7	16.3
A4096	85.864	85.69	67.4	53.6	42.7	31.8	22.9	19.2	16.4
A8192	243.22	87.68	85	62.9	51.3	33.9	25.4	20	16.9
A16384	240.43	88.07	86.8	63	51.4	34.2	24.5	20.1	16.9
B1024	42.21	21.29	14.3	10.4	8.14	5.07	3.98	3	2.36
B2048	40.274	20.15	13.1	9.8	7.91	5.03	3.82	3	2.46
B4096	39.633	20.03	13.1	9.8	7.87	4.99	3.76	3	2
B8192	40.466	19.62	13.4	9.85	7.8	4.99	3.94	3	2
B16384	39.756	20.52	13.1	10	7.88	4.96	3.45	3	2

Table 2. Multiplicity of FPS obtained by using GPU acceleration in each controlled experiment

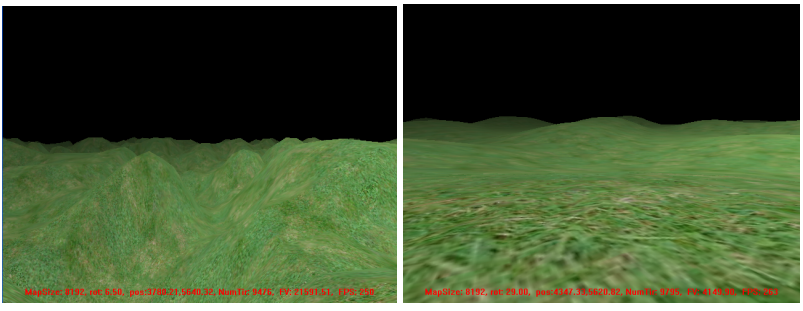
DTN / TS[3]	20k	40k	60k	80k	100k	150k	200k	250k	300k
1024	2.4039	4.84	6.2	7.29	7.8	8.56	8.48	8.98	9.58
2048	2.1278	4.262	5.69	6.12	6.04	6.6	6.39	6.56	6.6
4096	2.1665	4.277	5.13	5.47	5.42	6.37	6.08	6.4	8.2
8192	6.0104	4.47	6.34	6.39	6.57	6.8	6.43	6.68	8.47
16384	6.0476	4.292	6.61	6.29	6.53	6.89	7.09	6.7	8.44



(a) Average FPS (b) FPS Variance

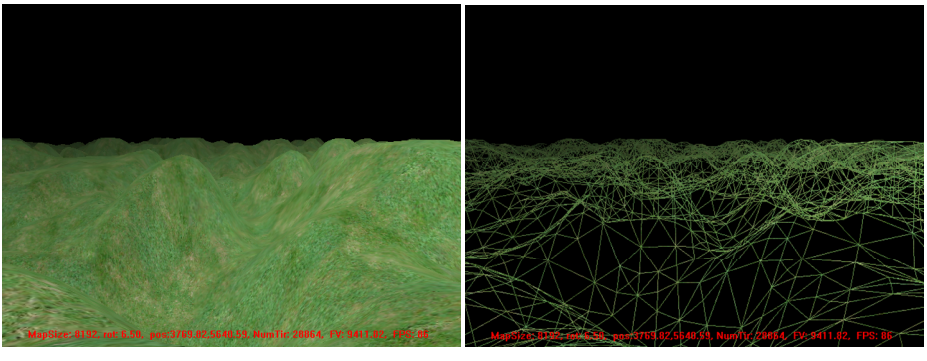
Fig. 4. Roughness Analysis

[1] *DTN* is short for *Desire Triangle Number*.
[2] *TG* is short for *Testing Group*.
[3] *TS* is short for *Terrain Size*.



(a) Rough Terrain (b) Smooth Terrain

Fig. 5. Roughness test



(a) Running Effect Screen Shot (b) LOD Mesh

Fig. 6. Testing Results

5 Summary and Future Work

This paper proposes a terrain visualization method that is based on the combination of improved ROAM algorithm and GPU programming. Technologies such as blocking and frustum pre-culling are applied with the improved ROAM algorithm to generate in real time a continuous LOD model for objects in the visible range. GPU programming is employed afterwards to calculate the vertex normal, illumination, texture sampling, and therefore surface rendering. Experiments indicate that this algorithm both saves memory space and has significant efficiency. It can visualize massive terrain data with high frame rates, stable animation, and satisfactory visual sensory. The algorithm is capable of rendering larger-scale terrain in real-time and has considerable practical value.

Because of the absence of anti-aliasing measurement, there appear some fidelity errors in the images such as zigzag contours. An appropriate measurement of anti-aliasing is necessary to improve the quality of the images. Since partitioning texture is applied, the texture repetition is patent for textures distant from the viewpoint.

Another area of future work is to apply suitable perturbation on the texture coordination to generate more natural terrain scenes.

Through testing, the workload of CPU is still found heavy because the square rooting operation cost a lot of CPU periods to subdivide the terrain surface. Another important reason is the frequent access of CPU to the height field data in the map file. The algorithm performance can only be improved through optimizing of or shifting computation from this segment.

GPU technology advances rapidly with high-end GPUs appear continuously on the market. Till present, the rendering speed has reached 100M triangles per second and the processing speed on vertex is increasingly accelerated. Meanwhile, GPUs possess strong capacities for parallel computation. Another refinement direction is to further enlarge the terrain data scale that can be rendered in real time and to increase efficiency in both texture sampling and terrain sampling through sufficiently utilizing the powerful functions of GPU to render and compress terrain data as well as to partition height filed texture.

Acknowledgments. This work was supported by Natural Science Foundation of China (No.60703006), National Hi-Tech 863 Program of China (No.2006AA10Z232), and National Key Technology R&D Program of China (No. 2006BAD10A03).

References

1. Clark, J.H.: Hierarchical Geometric Models for Visible Surface Algorithm. Communications of the ACM 19, 547–554 (1976)
2. Schroder, W., Zarge, J., Lorensen, W.: Decimation of Triangle Meshes. In: Proceedings of the SIGGRAPH 1992, pp. 65–70. ACM Press, Chicago (1992)
3. Hoppe, H., De Rose, T., Duchamp, T., et al.: Mesh Optimization. In: Proceedings of the SIGGRAPH 1993, pp. 19–26. ACM Press, New York (1993)
4. Hamann, B.: A Data Reduction Scheme for Triangulated Surface. Computer Aided Geometry Design 11, 197–214 (1994)
5. Hoppe, H.: Progressive Meshes. In: Proceedings of the SIGGRAPH 1996, pp. 99–108. ACM Press, New Orleans (1996)
6. Garland, M., Heckbert, P.S.: Surface Simplification Using Quadric Error Metric. In: Proceedings of the SIGGRAPH 1997, pp. 209–216. ACM Press, Los Angeles (1997)
7. Linstrom, P., Koller, D., Ribarsky, W., et al.: Real-Time, Continuous Level of Detail Rendering of Height Fields. In: Proceedings of SIGGRAPH 1996, pp. 109–118. ACM Press, New Orleans (1996)
8. Duchaineau, M., Wolinsky, M., Sigeti, D.E., et al.: ROAMing Terrian: Real-Time Optimally Adapting Meshes. In: Proceedings of the 8th conference on Visualization, pp. 81–88. IEEE Press, Phoenix (1997)
9. Du, J.L., Du, W., Chi, Z.X.: Vision Theory Based Multi-solution Terrain Generation Principles (in Chinese). Journal of Image and Graphics 8, 1295–1298 (2003)
10. Tang, Z.S.: 3D Data Field Visualization (in Chinese). Tsing Hua University Press, Beijing (1999)
11. Wu, E.H.: The Present Technique Situation and Challenge of the GPU Used as a GPGPU (in Chinese). Journal of Software 15, 1493–1504 (2004)

12. Losasso, F., Hoppe, H.: Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids. In: Proceedings of the SIGGRAPH 2004, pp. 769–776. ACM Press, Los Angeles (2004)
13. Schneider, J., Westermann, R.: GPU-Friendly High-Quality Terrain Rendering. Journal of WSCG ISSN 14, 1213–6972 (2006)
14. Clasen, M., Hege, H.C.: Terrain Rendering Using Spherical Clipmaps. In: IEEE VGTC Symposium on Visualization, pp. 91–98. IEEE Press, Lisbon (2006)
15. Fernado, R., Kilgard, M.J.: The Cg Tutorial: the Definitive Guide to Programmable Real-Time Graphics (in Chinese). Posts & Telecom Press, Beijing (2006)
16. Fernando, R.: GPU Gems Programming Techniques, Tips, and Tricks for Real-Time Graphics (in Chinese). Posts & Telecom Press, Beijing (2006)