SCIENCE CHINA
Information Sciences

• RESEARCH PAPERS •

August 2010 Vol. 53 No. 8: 1539–1545
doi: 10.1007/s11432-010-4022-9

GPU-based dynamic quad stream for forest rendering

LIU Feng, HUA Wei* & BAO HuJun

State Key Lab of Computer Aided Design and Computer Graphics, Zhejiang University, Hangzhou 310058, China

Received January 25, 2010; accepted April 1, 2010

Abstract Rendering large-scale forest scenes in real-time is a challenging problem, since the geometric complexity is far beyond the capabilities of current graphics hardware. In this paper, we propose a novel approach that adopts a dynamic quad stream residing on GPU buffers for view-dependent forest rendering. The incremental quad stream updating process is well-suited to the massively parallel architecture of the GPU pipeline, and it requires no continuous data exchange between CPU and GPU. The implementation of the algorithm is quite straightforward. Despite its simplicity, the algorithm achieves good performance with low memory, storage, and CPU costs.

Keywords real-time rendering, large-scale forest, quad stream, geometry shader

Citation Liu F, Hua W, Bao H J. GPU-based dynamic quad stream for forest rendering. Sci China Inf Sci, 2010, 53: 1539–1545, doi: 10.1007/s11432-010-4022-9

1 Introduction

In recent years, highly detailed tree models can be easily created using specialized vegetation modeling tools (e.g. XFrog, natFX, SpeedTreeCAD). Meantime the increasing programmability of modern graphics hardware makes it a fantastic machine to execute data-parallel algorithms. Because a large-scale forest always leads to heavy vertex throughput and huge memory costs, techniques that utilize the modern GPU power to address these problems have been receiving much attention in current research.

Early work focuses on geometry simplification algorithms [1–5]. These techniques simplify detailed models into multi-resolution LODs automatically. Although these methods produce acceptable results with trunks and branches, it is hard to simplify foliage due to the vast amount of unconnected polygons.

Image-based techniques are introduced to sharply simplify tree models and to accelerate rendering performance. Billboard [6, 7] is popular for real-time rendering because of its low costs. Billboard Clouds [8–10] directly analyzes tree models and simplify them into a set of textured planes. Stochastic simplification [11] is able to aggregate details by randomly sampling a subset of the geometric elements and altering them statistically to preserve the overall appearance. Layered depth images (LDIs) [12] use a set of pre-sampled parallel billboards to represent a tree. Hierarchical Layered Assembled Billboard Packs (HLABPs) [13] extends LDIs in a hierarchical way and accelerates rendering performance at the cost of increasing storage overhead.

*Corresponding author (email: huawei@cad.zju.edu.cn)

© Science China Press and Springer-Verlag Berlin Heidelberg 2010

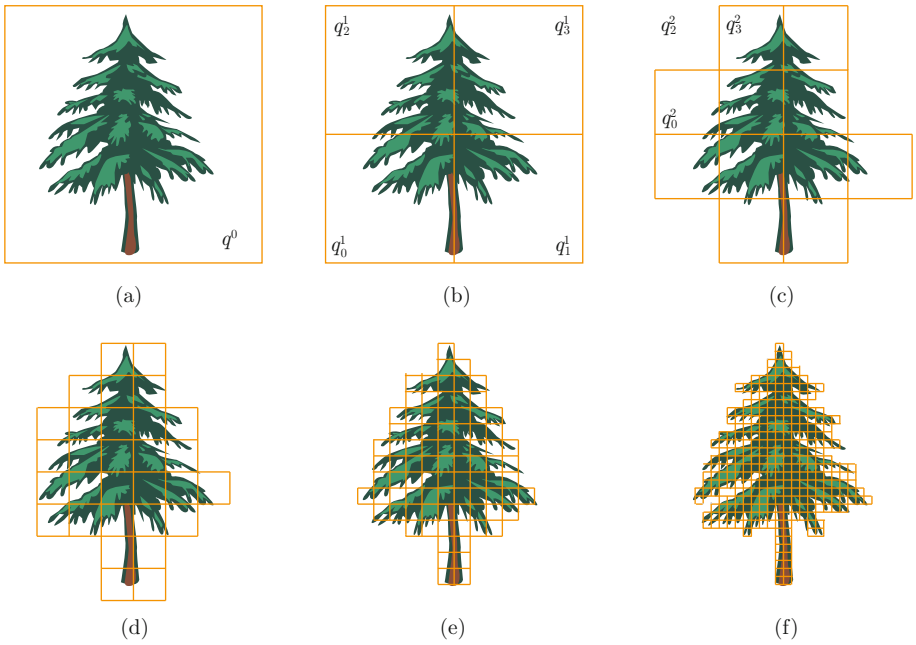

Figure 1 Quad mesh refinement (in image space).

A typical hybrid approach is SpeedTree [14] which combines polygon stems with billboard foliage to reach a good trade-off between image quality and speed.

Although these techniques are able to reduce forest polygon size, most of them are at expense of heavy CPU load and continuous data transfer between CPU and GPU. Some of them are even too much complicated for implementation.

Our approach is based on HLABPs and introduces a new forest representation called quad stream instead of the hierarchical scene graph of HLABPs. The quad stream is small enough to be totally loaded on GPU buffers and progressively refined by geometry shaders for view-dependent rendering. It is able to represent trees with billboards for efficiency or with detailed quad meshes to provide high visual quality. This approach could render large-scale forest in real-time and tremendously reduce memory and storage costs, data transfer overhead and CPU workload. The limitation of the algorithm is that triangular tree models need to be sampled at pre-processing phase for run-time quad stream generation and refinement.

2 Quad stream representation

A quad stream is defined as a set of depth quads residing on GPU buffers, and it can be refined progressively to represent a forest in a view-dependent manner. We express depth quad as q_i^k, where i is the sequence number and k is the corresponding LOD level. Then, a quad stream is denoted by

$$S = \{q_0, q_1, \ldots, q_{n-1}\}, \tag{1}$$

where S is a quad stream that contains n depth quads at different LOD levels. In addition, each quad in S is a tuple $q = (V, T, \alpha)$, where V specifies quad geometry in image space (V will be transformed to object space at rendering phase), T is the corresponding texture coordinates, and α is a value for blending.

The initial quads in S are dynamically generated according to view position, and they represent LDIs that are created at pre-processing phase. We define the initial quads in the stream as LODs at level 0. Then a quad q^0 can be subdivided into four sub-quads $\{q_0^1, q_1^1, q_2^1, q_3^1\}$ at level 1 with one quad subdivision operation $qs(q^0)$. As shown in Figure 1(a), a depth quad q^0 created from an LDI is split into four sub-quads in Figure 1(b). If a sub-quad is blank, it will be discarded like q_2^2 in Figure 1(c). In Figure 1(f), the quad q^0 is finally refined into a detailed quad mesh after 5 refinement phases.

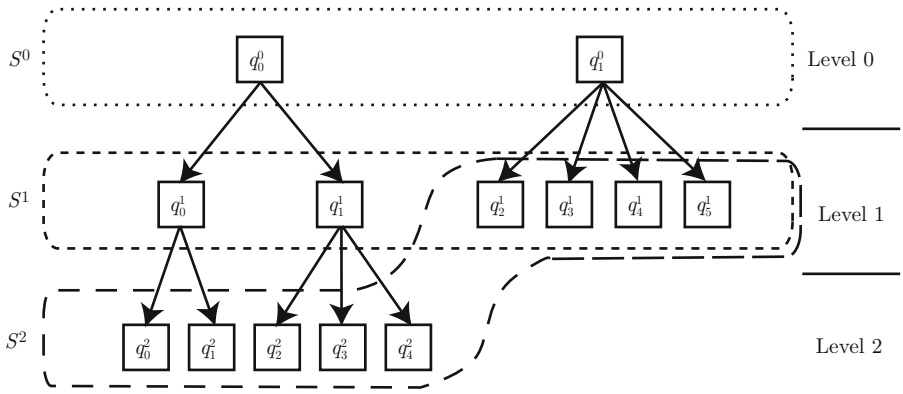

Figure 2 Quad stream refinement.

Same to one quad refinement, a quad stream S^0 at level 0 can be refined into a detailed quad stream S^m after m successive refinement phases:

$$S^0 \xrightarrow{qs(S^0)} S^1 \xrightarrow{qs(S^1)} \cdots \xrightarrow{qs(S^{m-1})} S^m. \tag{2}$$

For the purpose of view-dependent refinement, screen-projected quad edge length is evaluated as an error metric at each quad subdivision. If the error of a quad does not meet the given threshold ϵ, then we refine it into next level, otherwise the quad is returned directly. As shown in Figure 2, a quad stream S^0 contains 2 quads at level 0, and it is refined into S^1 and S^2 according to view position. It should be noted that S^2 contains 4 quads at level 1 since their errors meet the given threshold ϵ.

As refinement operations are processed independently, all quads in quad stream S^i can be refined into S^{i+1} in parallel. This characteristic makes quad stream an efficient way for large-scale forest rendering.

3 Our approach

Our approach consists of two phases: pre-processing and rendering. In the pre-processing phase, we sample different tree models into LDIs which are used to reconstruct quad meshes at different levels. Then, the forest is sorted, and the results with quad mesh attributes are organized into textures. The rendering algorithm generates quad streams and refines them in a view-depended manner.

To reduce geometric complexity of polygonal tree models, we use an image-based approach to convert triangular models into LDIs. These LDIs are sampled at a number of predetermined camera positions that are uniformly scattered over the upper sampling hemisphere, as shown in Figure 3(a). The tree model is rendered through ray-tracing with shading and self-shadows. For each sampling direction, we use different clipping planes to get a group of parallel LDIs, as shown in Figure 3(b).

Given an LDI with resolution $w_{\max} = 2^r$ and the minimal quad size $w_{\min} = 2^m$ $(m > 0)$, we are able to build $l = r - m + 1$ levels of quad meshes (e.g. Figure 1 illustrates 6 levels of quad meshes). The procedure to create these quad meshes is similar to quad mesh refinement in Figure 1, but each depth quad is created through fitting a plane to the corresponding pixels.

Since quad mesh rendering involves alpha-blending, it is necessary to sort trees from back to front before drawing them. However, sorting trees immediately when the view position changes is a time-consuming task and hard to be implemented with the GPU pipeline. Therefore, we introduce an approximation that pre-sorts trees at a number of pre-defined directions uniformly scattered over the upper hemisphere of the forest bounding sphere, just like sampling trees in Figure 3(a). Then, we organize the sorting results into a texture for run-time use.

The rendering algorithm is a three-pass-algorithm as outlined in Figure 4. The algorithm is totally executed on GPU and each pass requires only vertex array draw calls with all dynamic data residing on GPU vertex buffers. The only thing to be transferred between CPU and GPU is the current camera. The initial quad stream is generated and refined by GPU shaders when the camera is updated.

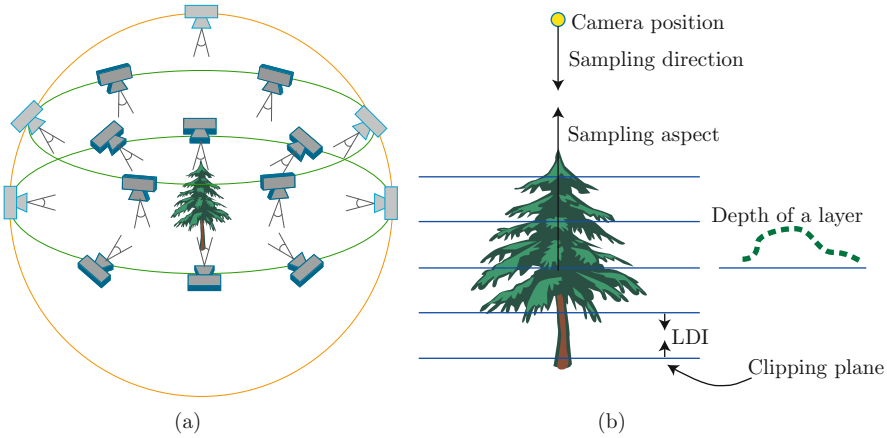

Figure 3 Sampling a tree model.

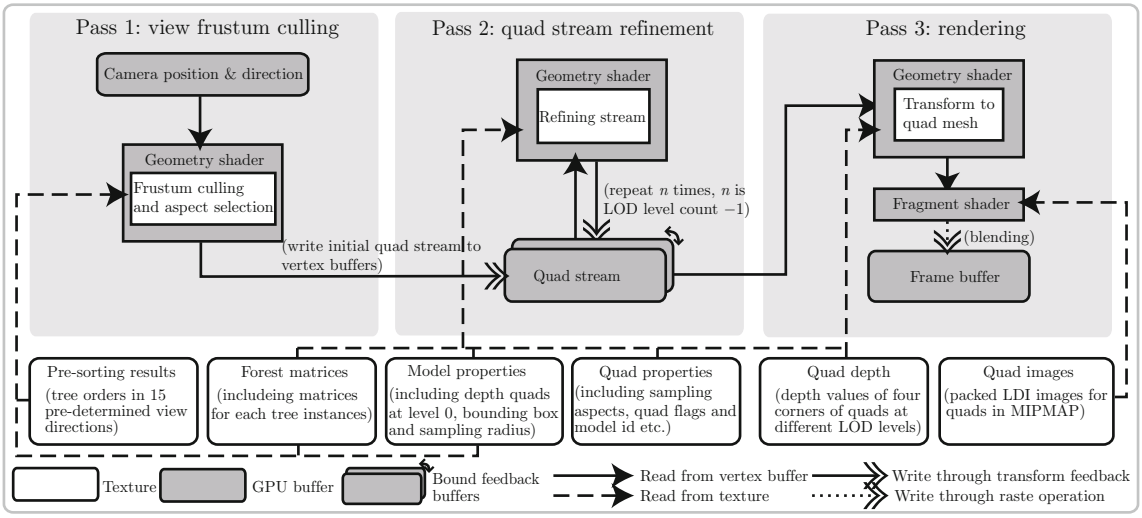

Figure 4 Outline of the multi-pass rendering algorithm on GPU.

When the camera position moves, the first pass is activated to generate a totally new quad stream that represents visible trees in a sorted order. According to the view direction, we choose a close pre-sorted result and apply frustum culling test to trees in back to front order. For each visible tree, we specify LDIs of 1–3 aspects that are facing user and calculate blending values respectively.

The initial quad stream is incrementally refined in the second pass. If the error of a depth quad q^i at level i is larger than the given threshold ϵ, then it will be refined into four sub-quads $\{q_0^{i+1}, q_1^{i+1}, q_2^{i+1}, q_3^{i+1}\}$ at most using pre-computed quad attributes stored in textures. If models of a forest scene have n levels of quad meshes, we will repeat this refinement process $n-1$ times to get the final view-dependent quad mesh. After refinement, the quad stream is ready for rendering.

As shown in Figure 5(a), each quad in quad stream is a tuple $q = (V, T, \alpha)$ that includes quad position in image space and texture coordinates. The depth values of four corners, sampling aspect and sampling radius can be retrieved from textures. With these attributes, we are able to generate four vertices to represent a quad primitive in object space, as shown in Figure 5(b). In Figure 5(c), the corresponding transform matrix of the tree instance is fetched to put the quad primitive to the right position in world space. Finally, as depicted in Figure 5(d) and (e), the quad primitive with texture is blended to the final image using the pre-computed α value.

4 Results

In our experiments, the algorithm performance was measured on a PC with an Intel Pentium D 2.66 GHz CPU, 2 GB RAM and a nVidia Geforce 8800GT (512 MB). The viewport size was set at 800×600.

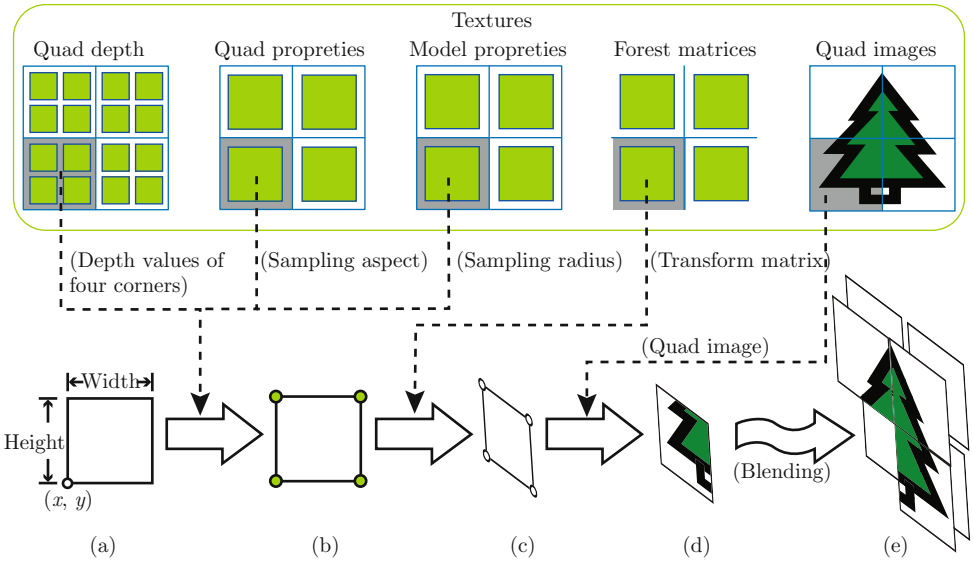

Figure 5 A quad is transformed and blended to the final image. (a) Quad $q = (V, T, \alpha)$ in image space; (b) quad in object space; (c) quad in world space; (d) quad with texture; (e) final image.

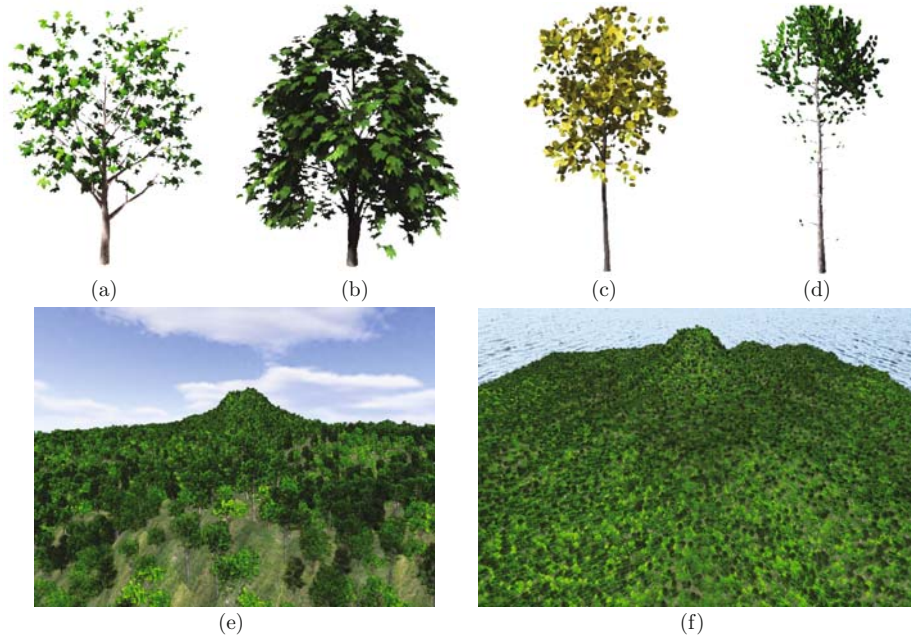

Figure 6 A quad is transformed and blended to the final image.

Sampling densities affect model storage cost, rendering performance and image quality, so we need to consider the tradeoff. In the experiment, we sample tree models at 15 pre-defined sampling positions, and 2 LDIs with resolution 512×512 are obtained in each direction. As the minimal quad size $w_{\min}$ is defined as 2^2, we have 8 levels of quad meshes for each LDI. Figures 6(a)–6(d) demonstrate that the sampled models provide satisfactory images when viewed from close-up positions. Figures 6(e) and 6(f) show two different views of a forest with 5 tree models and 60000 instances observed near the ground and at the top of the forest respectively. Benefiting from the dynamic quad stream refinement and depth quad blending, there is no noticeable visual artifacts during flyover or walkthrough.

In pre-processing phase, sampling a tree model and constructing corresponding textures take 16–31 min. The storage cost of a tree model is about 1.5–2.0 MB after compression. Then, we are able to build a forest scene with less than 20 MB storage cost using ten models. As a result, the storage cost is $O(N)$

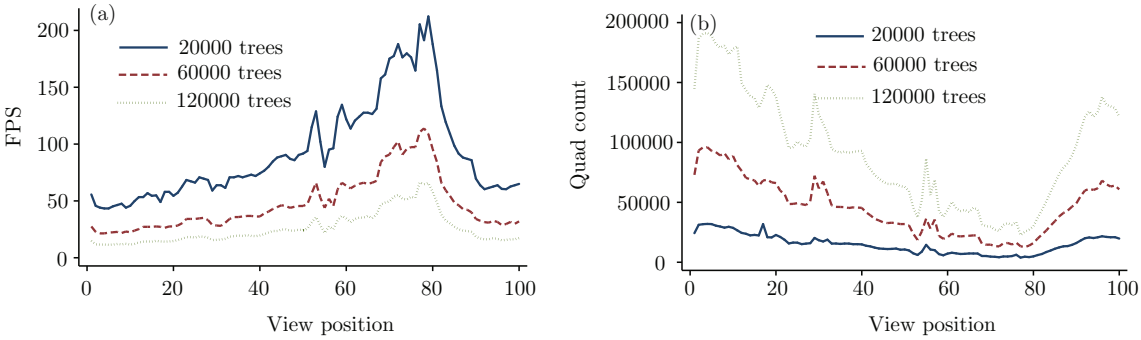

Figure 7 Rendering performance (threshold $\epsilon = 16$).

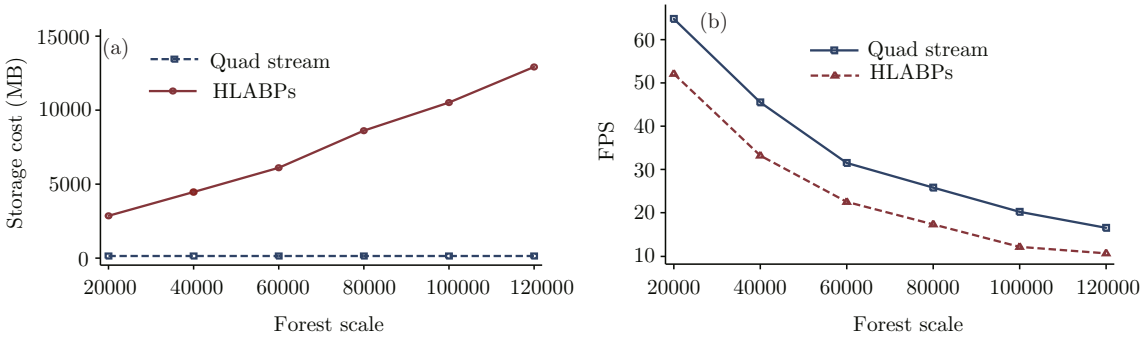

Figure 8 Storage cost and FPS of quad stream vs HLABPs.

(N is the number of tree models), which is irrelevant to tree instance count. This beats HLABPs since it may takes more than 10 GB storage cost for a forest with 100000 trees.

For a tree model, the size of textures loaded on GPU is about 15 MB. The texture cost complexity of a forest is $O(N)$ (N is the number of tree models). Since depth quads can be encoded as vertices stored on vertex buffers and only a small part of them need to be refined, the cost of a quad stream with 100000 trees takes 20 MB at most. The memory complexity is $O(N)$ (N is the number of quads in stream).

Table 1 shows the performance of our algorithm. In the experiments, six forests were observed at a close-up view and a bird's eye view. The pass of culling takes a little more time since it involves more operations of vertex buffer writing than the others. The refinement and rendering passes are quite efficient and directly related to the quad count in stream.

In order to measure the overall performance, we examined three forests in a same walkthrough path. Figures 7(a) and 7(b) illustrate FPS and quad count at 100 view positions in the path. The performance

Table 1 Quad stream performance (ms)

View position	Forest scale	Visible trees	Quad count	Culling	Refinement	Rendering	Total
1	20000	6181	17507	6.3	3.1	5.7	15.1
1	40000	12062	36059	8.6	4.6	11.9	25.1
1	60000	18532	51737	12.1	6.3	15.3	33.7
1	80000	24564	68201	15.5	7.8	21.3	44.6
1	100000	30319	84740	18.4	9.3	28.1	55.8
1	120000	36553	103112	21.2	10.9	34.3	66.4
2	20000	10511	24176	7.9	1.5	8.8	18.2
2	40000	31938	62024	10.6	1.6	15.6	27.8
2	60000	47471	92207	16.8	3.2	23.9	43.9
2	80000	64040	124352	19.3	6.1	31.7	57.1
2	100000	79922	155234	22.8	6.4	39.0	68.2
2	120000	94728	183977	27.8	6.7	47.9	82.4

is closely relative to the quad count in stream. Due to quad count change at different view positions, the performance results shown in Figure 7 exhibit some peaks and sharp changes. The frame rate fluctuates from 15 to 200 FPS depending on how many trees in the view frustum and how detailedly the quad stream is refined. Figures 8(a) and 8(b) show that our algorithm significantly reduces storage cost and achieves better performance than HLABPs.

5 Conclusion and future work

In this paper, we present a real-time forest rendering approach which precomputes collections of LDIs, and uses adaptive refinement at run time to adapt the pixel complexity to the point-of-view. The results of our implementation proves that, a large-scale forest can be significantly simplified at a pre-processing phase, and the GPU-based algorithm is able to refine the forest to provide a convincing level of realism in real-time with low memory, storage, and CPU costs.

For future work, we will further improve some additional features including dynamic lighting and deformation and implement them with GPU shaders.

Acknowledgements

This work was supported by the National Basic Research Program of China (Grant No. 2009CB320802), the National Natural Science Foundation of China (Grant No. 60773184).

References

1 Erikson C, Manocha D. GAPS: general and automatic polygonal simplification. In: Proceedings of ACM Symposium on Interactive 3D Graphics, Atlanta, Georgia, USA, 1999. 79–88
2 Garland M, Heckbert P S. Surface simplification using quadric error metrics. In: Proceedings of ACM SIGGRAPH, Los Angeles, California, USA, 1997. 209–216
3 Hoppe H. Progressive meshes. In: Proceedings of ACM SIGGRAPH, New Orleans, Louisiana, USA, 1996. 99–108
4 Remolar I, Chover M, Belmonte O, et al. Geometric simplification of foliage. In: Proceedings of Eurographics Short Presentations, Saarbrücken, Germany, 2002. 397–404
5 Decaudin P, Neyret F. Rendering forest scenes in real-time. In: Proceedings of Eurographics Workshop on Rendering, Norköping, Sweden, 2004. 93–102
6 Meyer A, Neyret F, Poulin P. Interactive rendering of trees with shading and shadows. In: Proceedings of Eurographics Workshop on Rendering, London, UK, 2001. 183–196
7 Jakulin A. Interactive vegetation rendering with slicing and blending. In: Proceedings of Eurographics 2000 (Short Presentations), Brno, Czech Republic, 2000
8 Décoret X, Durand F, Sillion F X, et al. Billboard clouds for extreme model simplification. ACM Trans Graph, 2003, 22: 689–696
9 Fuhrmann A, Umlauf E, Mantler S. Extreme model simplification for forest rendering. In: Proceedings of Eurographics Workshop on Natural Phenomena, Dublin, Ireland, 2005. 57–66
10 Lacewell J D and Edwards D, Shirley P, et al. Stochastic billboard clouds for interactive foliage rendering. J Graph Tool, 2006, 11: 1–12
11 Cook R L, Halstead J, Planck M, et al. Stochastic Simplification of Aggregate Detail. In: Proceedings of ACM SIGGRAPH, San Diego, California USA, 2007
12 Shade J, Gortler S, He L W, et al. Layered depth images. In: Proceedings of ACM SIGGRAPH, Orlando, Florida, USA, 1998. 231–242
13 Zhang H S, Hua W, Wang Q, et al. Fast display of large-scale forest with fidelity. Comput Animat Virtual Worlds, 2006, 17: 83–97
14 Interactive Data Visualization Inc. http://www.speedtree.com