Efficient Point Based Global Illumination on GPU

Beibei Wang and Zhen Xu and Yanning Xu and Xiangxu Meng

Shandong University, China

Abstract

Point based rendering can simulate global illumination phenomenon fast and is widely used in movie production. This paper proposes a point based color bleeding algorithm based on GPU efficiently. In our algorithm, we reorder the shading points according to the similarity of them to use the coherency of the GPU memory. Then, we propose a novel idea named chunking to accelerate the point cloud traversal process by using the constant memory of GPU.

1. Introduction

In this paper, we propose an efficient PBGI method based on GPU. We reorder the shading points before computing the indirect illumination (IDI). The shading points with similar distance and similar normal have a high possibility to traverse the point cloud kd-tree with the same path. So we can arrange these shading points in the same block when doing GPU computing. Another contribution of this paper is that we propose a novel idea named chunking to further improve the speed of kd-tree traversal by using the constant memory efficiently.

2. Related work

Point-based global illumination method was proposed by Christensen in [Chr08]. The method uses a point cloud to represent the direct illumination (DI), and then the point cloud is used to compute the IDI. Kontkanen [KTO11] proposed a coherent out-of-core point-based global illumination method. It extends the standard PBGI by using an out-of-core octree constructing and traversing method between the main memory and disk. It also uses a chunk strategy to increase access coherency and decrease I/0. Ritschel uses a novel technique for scalable and parallel final gathering on GPU [REG*09]. This method does not consider using the GPU resources efficiently. Maletz and Wang proposed an importance-driven method for GPU-based final gathering in [MW11]. Similar to [Chr08], it uses a point cloud to represent direct illumination, and then projects and splats the points to the microbuffers of shading points based on the importance of each point. This method is implemented on GPU, and is proved to be efficiency. But it has the same problem as [REG*09].

3. Algorithms

Figure 1 shows the main building blocks of our algorithm. And in this paper, we focus on the shaded part.

3.1. Shading Points Reordering

When computing the IDI of shading points, we traverse the point cloud kd-tree, and there is a traversing path for each shading point. And the criterion whether the children of the current node should be traversed is determined by comparing a solid angle and the preset max solid angle. And the definition of the solid angle is listed in (1).

$$s = A/d^2 \tag{1}$$

where A is computed by evaluating the spherical harmonic function representation for cluster area and represents the distance between shading points and kd-tree node. The spherical harmonics coefficients for the projected area of each surfel is computed using (2) [Chr08].

$$\int_{\phi=0}^{2\pi}\int_{\theta=0}^{\pi} A_i(d*n_i)Y_im(\theta,\phi)sin\theta d\theta d\phi \tag{2}$$

where θ,ϕ are spherical coordinates, d = (sin θ cosϕ ,sinθ sinϕ,cosθ), and Y_im represents a spherical harmonic basis function.

From (1) and (2), we can find that the path with which the shading points traverse the kd-tree is determined by their position and normal.

So we can conclude the shading points with similar position and similar normal have the high possibility to have the same traversal path.

DOI: 10.2312/conf/EG2012/posters/017-018

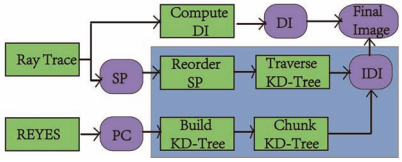

Figure 1: *The pipeline of our algorithm. The shaded box shows the contribution of our paper. The SP represents shading points; PC represents point cloud.*

So instead of arranging the shading points to the GPU threads arbitrarily, we reorder the shading points according to the position and the normal information, and arrange the similar shading points to the same warp. And the similarity of two points (p_1,n_1) and (p_2,n_2) is defined in (3).

$$S = w_1 * |p_1 - p_2| + \frac{w_2}{n_1 * n_2 + K} \quad (3)$$

Where w_1,w_2 represent weight of position and normal, and represents a constant offset of normal.

We implement reordering by organizing the shading points into a kd-tree. And the overhead caused by reordering is proved to be little.

3.2. Point Cloud Chunking

The speed to access constant memory is much faster than to access global memory in GPU. But the size of the former one is much smaller. It 's not possible to store the whole point cloud kd-tree in the constant memory, so we propose an idea to divide the point cloud kd-tree into chunks, and each chunk can be fit into the constant memory. The process to do this division is called chunking. In our implementation, a chunk may contain 128 nodes.

At first, the nodes in the same chunk should be stored in coherency global memory. Then we can transfer the chunk data between the global memory and constant memory directly. Chunking is illustrated in Figure 2.

When doing IDI computing, the point cloud kd-tree is traversed from top down. And the chunks are transferred to the constant memory one by one. When traversing in a chunk, if the node in the chunk is active that means the node is far enough, and then the node can be projected to the current shading point 's hemicube, or its children is pushed into traversing stack. If the current node is not in the memory, then it is not processed until the related chunk is transferred in. After all the chunks are processed, all the shading points finish traversing the whole tree.

3.3. Results and Discussion

We have implemented our method on GPU. All the results are computed on 2.00GHz Intel Xeon CPU RAM 16G,

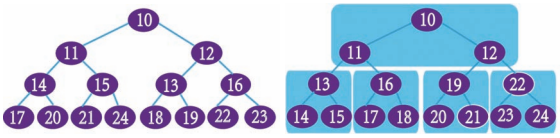

Figure 2: *Desk results: **Left:** the original nodes' index in the array; **Right:** the nodes' index in the array after chunking. The part in a shaded box is called a chunk, and it has a sequency index in the array.*

NVIDIA Quadro FX 4800 with 24 multiprocessors and 192 CUDA cores. The GPU part is in on top of OpenCL. The desk examples are 400×400 resolutions; the super sampling value is set as 4×4; the shading rate value is set as 20.The reordering method improves the speed about 16.6%. The bedroom example is 800×600 resolutions. The super sampling value is set as 2×2. The reordering process improves the speed about 14%. The data is described in Figure 3.

4. Conclusion and Future work

We have proposed an efficient PBGI based on GPU. The shading points are reordered to further improve the efficiency. We also propose a new idea to use the constant memory in GPU by chunking the point cloud. The chunking part to use constant memory is under implementation.

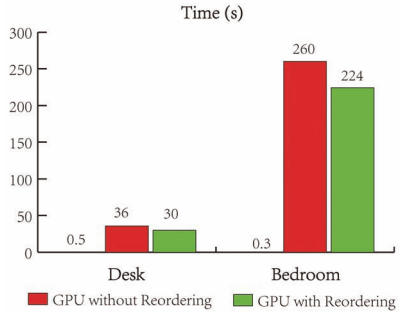

Figure 3: *Bedroom and Desk examples: time cost by reordering; time cost by IDI computing without reodering and with reodering*

References

[Chr08] CHRISTENSEN P.2008. Point-Based Approximate Color Bleeding. Pixar Technical Memo.

[KTO11] KONTKANEN J.., TABELLINE E., OVERBECK R.Overbeck. 2011.Coherent Out-of-Core Point-Based Global Illumination, Eurographics Symposium on Rendering 2011. 30,4 (2011).

[MW11] MALETZ D.WANG R.2011.Importance Point Projection for GPU-based Final Gathering, Eurographics Symposium on Rendering 2011. 30, 4 (2011).

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C. 2009. Micro-rendering for scalable, parallel final gathering. ACM Trans. Graph. 28, 5 (2009), 1-8. 1, 2, 8, 9