Distributed Global Illumination Method Based on Photon Mapping

Xiang Xu[12], Lu Wang*[12], Yanning Xu[12], Chenglei Yang[12], Xiangxu Meng[12]
[1]*Department of Computer Science and Technology, Shandong University*
[2]*Engineering Research Center of Digital Media Technology, Ministry of Education*
Jinan, China
**E-mail:luwang_hcivr@sdu.edu.cn*

Abstract

As an excellent global illumination algorithm, photon mapping requires a large amount of memory to store photons, and it has obvious spatial locality in the step of photon searching. In this paper, we propose a distributed photon mapping algorithm. Firstly, a new delivering strategy of rendering tasks is proposed based on the ray intersection, while path tracing is used to calculate the direct illumination. Before the rendering, our work automatically synchronizes the acceleration of geometries and photons to realize data partitioning. Secondly, an overlapped KD-Tree is proposed to store photons in order to solve the problem of boundary data merging. The method of this paper is tested using multiple scenes. With ensuring good rendering results, both the rendering speed and storage load obtain the linearly optimization.

Keywords

Distributed Computing; Global Illumination; Photon Mapping; High Performance Computing;

I. Introduction

Rendering technology plays an important role in the production of film and 3D games. It's essential to enhance the rendering speed while pursuing high quality. Except for the improvement of the graphics algorithm, with the development of computer hardware and distributed computing, the concepts of cloud rendering, distributed rendering are proposed. In this paper we use primary ray rendering of ray tracing to calculate direct illumination. Photon mapping as a two-step global illumination rendering algorithm, we choose it for secondary illumination because the method only needs to access the photons near the hit point, which is more convenient for data distribution.

II. Related Work

A. Photon Mapping

Photon Mapping is a two-step global illumination algorithm, first proposed by Jensen [1]. In the first step, photons are traced form light source, and stored in a k-d tree when hiting a diffuse surface. In second pass, render image with a mainstream rendering algorithm (raytracing, path tracing, etc.), and the photons generating in the first step are around hit points. Hachisuka et al. [2] proposed progressive photon mapping(PPM) method in 2008, and proposed SPPM [3] method next year. Both of them solve the photon storage bottleneck in original PM algorithm by using an iteration of photon emission. However, the powerful storage capacity of the cluster can be a breakthrough in this problem. Meanwhile, compared to PPM and SPPM, the photon map of PM only need to be generated once in the prepocess stage, and can be reused in the next process of rendering, which leads to a faster rendering speed. For the improvement of the initial PM, [4] changed the structure of kd-Tree [5], and use the photon activity statistics to update the photon map information. It's possible to access to the photons that might need in real time photon mapping.

B. Distributed Rendering

Wald et al. [6] constructs a ray tracing cluster system, in which each slave stores the upper tree nodes, and caches the data needed from master to complete the rendering. Combined with shared memory(DSM) ray tracer [7], DeMarle et al. [8] use image parallel rendering with PDSM mode. Image parallel makes every slave node responsible for a different subset of the image space rendering. But the method uses micro-cell storage geometry, the speed is slower than some advanced scene acceleration structure, such as BVH trees [9], kd-trees [5]. Based on that, [10]realizes real-time ray tracer for visualizing massive models on a cluster. In the respect of the distributed photon mapping, D.Fradin [11] proposed a method applicable to large buildings. The method represented a scene into cells and used portals for scene segmentation and light transfering, so that each slave node only needs to store data of a cell. Thus, in the boundary of two cells, the performance of the algorithm is not very good. Therefore, Tobias Gunther and Thorsten Grosch [12] used portal to transfer information in the photon collection, which solves the problem of data consistent.

978-1-5090-5952-2/16 $31.00 © 2016 IEEE
DOI 10.1109/IIKI.2016.68

III. Distributed Photon Mapping

We use shared memory to share the rendering task information rather than geometry data, in which way to render a bucket in multiple nodes with local data. And we choose the overlapped KD-Tree to collect the needed photons to overcome the problem, when searching photons for the scene boundary intersection. Firstly, we apply SAH strategy [13] [14] to build the scene kd-tree to ensure the load balance of each subtree node. We suppose that there are 2^n slave nodes in the cluster. After the completion of the photon tracing work, the split plane are selected synchronously in the top n layers to build photon kd-tree according to the built geometry kd-tree. In addition, each slave node has a copy of the top levels of the tree, so that during the ray intersection, it's capable of traversing from the root tree node to the corresponding slave node to complete attribution assignment and the rendering work.

A. Task Delivery

In the task partition strategy, we adapt image parallel method [8], that each computer is responsible for a number of rectangular render bucket (render block). Master node splits screen space and distributes to each slave node averagely in a certain order. After obtaining the task, the slave node determines the task attribution first. Each slave node stores the same top levels of kd-tree, when the primary ray traverses to the splitted layer, it can judge the subtrees the intersection may access next.

First, if it is the case of the algorithm except for *N4* and *P4* for every shading point in a bucket, we can determine a specific slave node where the subtree data are stored. And for *P4* and *N4* cases(see Figure 1(a)), In the stand-alone rendering process, for example, the tree node *A* will be recorded as near node and node *B* as far node. If the light in *A* intersects to the geometry, then take the result of *A* as the final result. Otherwise, continue to shade with the result of *B*. However, in distributed rendering, the communication between slave nodes is asynchronous. We can't allow the slave *B* to wait for the result of slave *A* and decide whether to do the intersection again, which would lead to severe network jam. In this paper, we use the stategy that multiple slave nodes render respectively, and the master merge all the results according to the priority of the block in each slave node.

B. Overlapped KD-tree

After the calculation of direct illumination by path tracing, we use photon mapping as indirect illumination. When the ray intersects with a primitive, it will search the surrounding photons to calculate global illumination. Therefore, for the hit points near the splitted scene boundary, we need to visit the photon data stored in nearby slave nodes. To solve this problem, this paper proposes an approximate kd-tree model based on overlapped region. We set a range at the split plane, and partition certain region as the overlapped region. the points in the range are belong to both the left child node and the right child node. Actually we don't have to build all the trees in master, but traverse to the entire array of photons once. To every photon, we calculate its distance to the split plane in x, y and z directions, and determine one or several slaves the photon should be passed to. And then the slave side builds their own photon map for the obtained photons in parallel. Later in the process of photon gather, we only need to query the local photon map of current slave which is consistent with the result by non-distributed rendering algorithm(see Figure 1(b)).

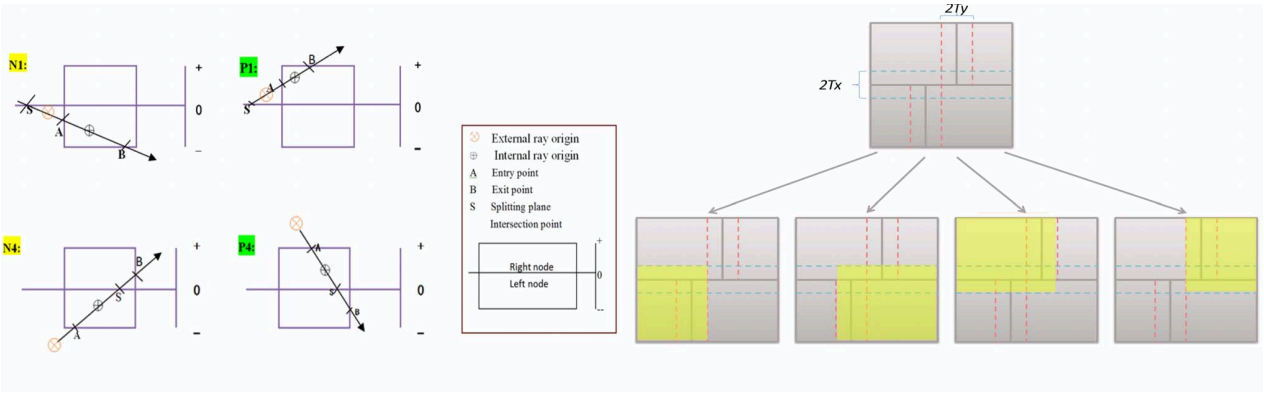

Figure 1. (a) 4 different conditons for a ray intersected with a tree node using haveran algorithm. In case $N1$ or $P1$, the ray only intersects with one of the child nodes, but both in case $N4$ or $P4$. (b) Overlapped KD-tree.

IV. Results

We implement our method based on Mitsuba renderer. We obtained our results using a cluster of 5 equal machines(1 master and 4 slaves), also 9 equal macheines but with a bottleneck. Each machine is equipped with an Intel Xeon 8 core CPU with 2.3 GHz, 8G RAM and 1 Gbits/s connection. The method is applied to several different scenes as shown in Figure 2. We test our method's computational efficientcy using 8 threads each machine. Images are rendered at 512 *

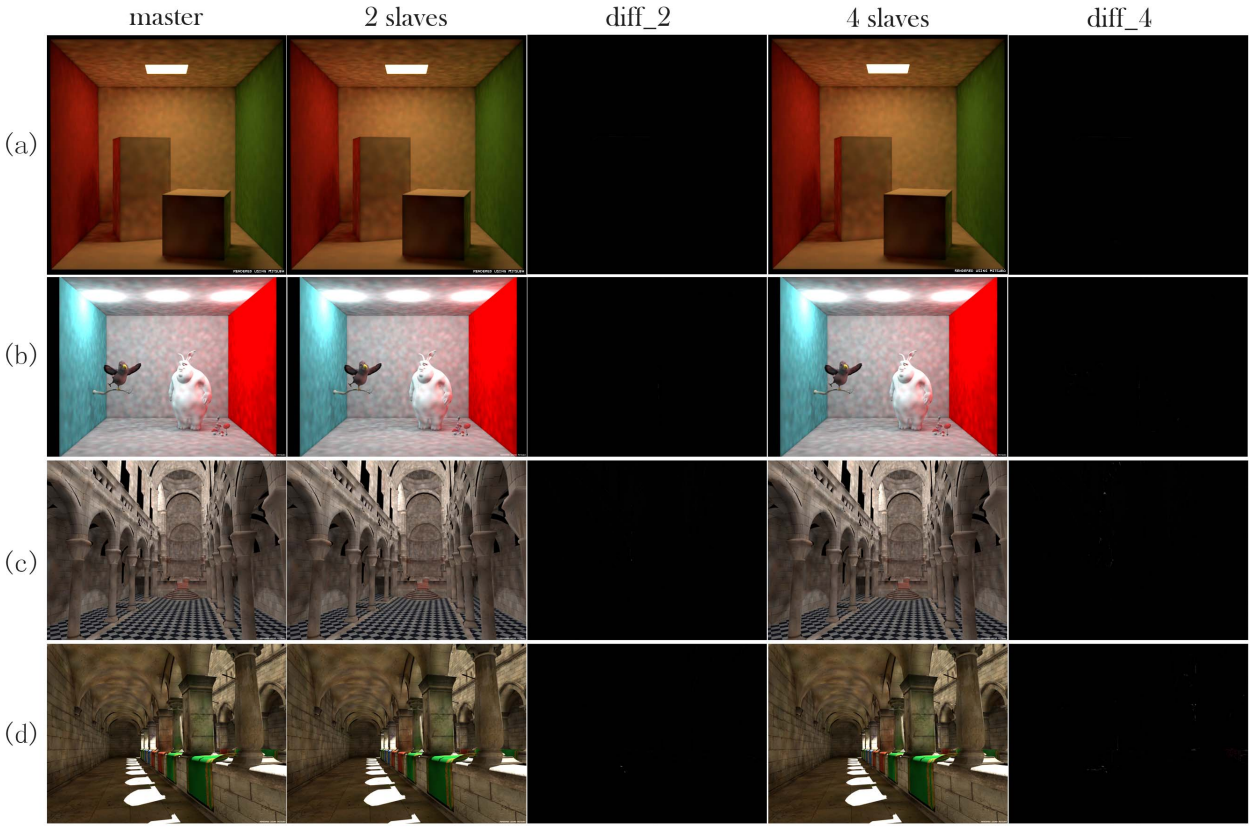

Figure 2. Four different scenes' rendering results with different number of slaves.

Table I
STATISTICS FOR RENDER TIME, MAXIMUM STORAGE LOAD OF SLAVE NODES AND IMAGE ERROR.

Scene	Time						Storage						Error	
	gather time			render time			photon			primitive			2	4
	0	2	4	0	2	4	0	2	4	0	2	4		
cbox	1.4s	2.1s	2.5s	6.4m	3.0m	2.0m	250000	138455	95882	38	30	24	1.5e-6	1.6e-6
bunny	1.5s	2.2s	2.6s	9.6m	3.4m	2.5m	250000	140743	78488	217270	147691	103073	7.0e-6	1.1e-5
sibenik	7.7s	10.5s	13.6s	5.9m	3.1m	2.1m	1000000	559033	332318	75282	41068	20868	4.6e-5	6.7e-5
sponza	19s	24.2s	29.5s	14.8m	8.1m	6.1m	2000000	1258026	683637	105400	54233	28312	1.8e-5	5.3e-5

512 or 1024 * 768 resolution, and the render buckets are delivered to each slave node at 32 * 32. We also compare the difference of the rendering results between our distributed method and the origin photon mapping algorithm in Mitsuba. Overall, we have a negligible error on the split border line, and exactly unbias in other regions of the image. As shown in Table I, we record the rendering time and storage of per computer in different numbers of slave node, such as 0, 2 or 4. Firstly, for the time efficiency, our method get a speedups of double or trible with 2 slaves, and is further accelerated linearly when extended to 4 slaves. The increase of preprocessing time is caused by the network transmission for photons, which is insignificant compared to the decreased rendering time. Secondly, we record the maximum of photon number and primitive number devided into each slave. The cluster would be enlarged to 8 or 16 slaves, but accompanied with a bottleneck in time and space efficiency, caused by the network delay and border problem metioned. As a whole, our method achieves good results for difffuse scenes using small-scale clusters. Statistically, in all the experiments above, we obtained a linear speed up, also the storage per computing node is reduced linearly.

V. SUMMARY AND DISCUSSION

We realized a distributed photon mapping algorithm for the diffuse scene, and obtained an acceleration of time and reduction of the storage load linearly. In all scenes, experimental results show that all the CPU resources are occupied. In order to further optimize the efficiency, the reduction in network traffic may be considered.

ACKNOWLEDGMENT

This work was partly supported by the National Natural Science Foundation of China under Grant Nos. 61472224, 61472225, the national high-tech research and development plan of China under grant No.2014AA01A302, the Special Funding of Independent Innovation and Transformation of Achievements in Shandong Province of China under Grant No. 2014ZZCX08201, the Shandong Key research and development program under grant No. 2015GGX106006, the young scholars program of shandong university under grant No. 2015WLJH41 and the Special Funds of Taishan Scholar Construction Project of China.

REFERENCES

[1] H. W. Jensen, "Global illumination using photon maps," in *Rendering Techniques 96*. Springer, 1996, pp. 21–30.

[2] T. Hachisuka, S. Ogaki, and H. W. Jensen, "Progressive photon mapping," *ACM Transactions on Graphics (TOG)*, vol. 27, no. 5, p. 130, 2008.

[3] T. Hachisuka and H. W. Jensen, "Stochastic progressive photon mapping," *ACM Transactions on Graphics (TOG)*, vol. 28, no. 4, p. 141, 2009.

[4] T. R. Jozwowski, "Real time photon mapping," Ph.D. dissertation, Michigan Technological University, 2002.

[5] A. Reshetov, A. Soupikov, and J. Hurley, "Multi-level ray tracing algorithm," *ACM Transactions on Graphics (TOG)*, vol. 24, no. 3, pp. 1176–1185, 2005.

[6] I. Wald, P. Slusallek, and C. Benthin, *Interactive distributed ray tracing of highly complex models*. Springer, 2001.

[7] B. Corrie and P. Mackerras, "Parallel volume rendering and data coherence," in *Parallel Rendering Symposium, 1993*. IEEE, 1993, pp. 23–26.

[8] D. E. DeMarle, C. P. Gribble, S. Boulos, and S. G. Parker, "Memory sharing for interactive ray tracing on clusters," *Parallel Computing*, vol. 31, no. 2, pp. 221–242, 2005.

[9] I. Wald, S. Boulos, and P. Shirley, "Ray tracing deformable scenes using dynamic bounding volume hierarchies," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 1, p. 6, 2007.

[10] T. Ize, C. Brownlee, and C. D. Hansen, "Real-time ray tracer for visualizing massive models on a cluster." in *EGPGV*, 2011, pp. 61–69.

[11] D. Fradin, D. Meneveaux, and S. Horna, "Out of core photon-mapping for large buildings." in *Rendering Techniques*, 2005, pp. 65–72.

[12] T. Günther and T. Grosch, "Distributed out-of-core stochastic progressive photon mapping," in *Computer Graphics Forum*, vol. 33, no. 6. Wiley Online Library, 2014, pp. 154–166.

[13] J. Goldsmith and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *Computer Graphics and Applications, IEEE*, vol. 7, no. 5, pp. 14–20, 1987.

[14] J. D. MacDonald and K. S. Booth, "Heuristics for ray tracing using space subdivision," *The Visual Computer*, vol. 6, no. 3, pp. 153–166, 1990.