International Journal of Grid and Distributed Computing
Vol. 9, No. 11 (2016), pp. 49-60
http://dx.doi.org/10.14257/ijgdc.2016.9.11.04

Research on Parallel KD-Tree Construction for Ray Tracing

Zhang Peicheng[1], Xu Huahu[2], Bian Minjie[1] and Gao Honghao[1, 3,*]

[1] *School of Computer Engineering and Science, Shanghai University, 200444, Shanghai, China*
[2] *Shanghai Shang Da Hai Run Information System Co., Ltd*
[3] *Computing Center, Shanghai University, 200444 Shanghai, P.R. China*
{bigticket, gaohonghao}@shu.edu.cn

Abstract

Many computer graphics rendering algorithms and techniques use ray tracing for generation of photo-realistic images, and kd-tree is the most popular acceleration data structure for ray tracing. In order to avoid the inefficient parallel performance of kd-tree construction based on surface area heuristic (SAH), an algorithm using Morton code and path compression was present in this paper. Instead of building a kd-tree layer-by-layer, the proposed approach can be performed in parallel from bottom of a conceptual perfect binary tree. Experimental results on GPU show that our work achieves a faster kd-tree construction procedure.

Keywords*: ray tracing, kd-tree, kd-tree construction, Morton code, GPU*

1. Introduction

1.1. The Present and Problems of Ray Tracing

Ray tracing is an important rendering algorithm of global illumination model, and is also a subject extensively concerned in 3D simulation field. The goal of the algorithm is to generate photo-realistic visual effect, which achieves a better performance than the traditional algorithm of ray-casting. The complexity of ray tracing is associated with the testing of rays for intersection with the objects of the scene. Almost 75%~95% of the overall time for ray tracing was taken by intersection test [1]. For a scene with N graphics primitives, when rendering a picture with M pixels, ray tracing without acceleration would have a time complexity of $O(MN)$. Accordingly, the key to faster ray tracing is how to speed up the intersection testing of the ray and the scene. By separating and organizing the space of scene properly, intersection testing can be reduced as well as the time complexity of the algorithm. This is the most important and mainstream way of accelerated rendering, with the kd-tree and BVH are two widely used acceleration structure.

Standard ray tracing has been well-studied as a realistic rendering algorithm [2], which evolved from ray casting [3]. Whitted [1] proposed ray tracing algorithm in 1980. It's a recursive algorithm, which enables direct sunlight, specular reflection and refraction effects. Many improved approaches based on Whitted's work was discussed by researchers: Cook *et al*. [4] introduced distributed ray tracing algorithm, and Kajiya extended it to Monte Carlo ray tracing algorithm [5].

While ray tracing generates high realism of image, it requires a large amount of light to be calculated. Beside the calculation of objects in the scene affected by the light source, the algorithm also needs to calculate the effects of light which is reflected from an object in the scene, refraction or scattering. The sequential ray tracing has been unable to meet the needs of practical application, and the parallel one is getting more and more attention. Zdippe *et al*. [2] subdivided the ray tracing process, and implemented this subdivision on

ISSN: 2005-4262 IJGDC

parallel hardware. Reinhard *et al*. [6] summarized the advantages and shortcomings of two basic approaches to parallel rendering: demand driven and data parallel, then proposed a hybrid approach to render large models.

1.2. The Present and Problems of KD-Tree

As a significant space subdivision structure, kd-tree is commonly used to ray tracing. Selection of the subdivision plan is the key issue during the construction of a kd-tree. Subdivision plan not only is related to the quality of kd-tree directly but it also affects the performance of the ray tracing algorithm. Early researches focused on the culling of the triangles, which are calculated intersection with rays in the scene. Spatial median splitting and SAH are both well discussed and widely used.

Spatial median splitting is a simple but often-used method. Without considering the specific distribution of the triangles in the scene, the selected subdivision plan of this method is positioned at the spatial median of the longest axis of the tree node volume. In this way, a mass of computation are avoided. However, if using ray tracing which is based on spatial median splitting only, the quality of the kd-tree it generates is too poor to reach a high efficiency rendering result [7], especially for the scene with highly random distributed primitives.

Macdonald *et al*. [8] proposed SAH, which has been turned out to be a more effective heuristic space subdivision algorithm. This algorithm transform the issue of spatial spitting to a global optimization issue of intersection. It can split the spatial of the scene efficiently by taking the cost of ray-triangle intersection as the target function SAH to optimize, and evaluating the SAH cost function. Kd-tree constructed based on SAH can combine with the actual position distribution of the primitives, avoiding the random construction caused by spatial median splitting. Wald *et al*. [9] using the method of local greedy SAH heuristic and "perfect splits" reduced the time complexity of kd-tree construction based on SAH to $O(NlogN)$.

When constructing kd-tree from top to bottom, as the number of the nodes at the upper tree levels is very small, the traditional SAH algorithm can't running with high parallelism. Zhou *et al*. [10] present a hybrid scheme of median splitting and SAH, a node is deemed as "large node" if the number of triangles in it is greater than a specified threshold which set by user, otherwise it is "small node". Median splitting was used for large nodes to estimate the costs while SAH was used for small nodes. This algorithm running on GPU is significantly faster than others.

Instead of resorting to SAH or top-down manner, Li *et al*. [11] presented a novel scheme which based on Morton code and path compression procedure. The algorithm aimed at solving the difference parallelism between large nodes and small nodes, and was proved that it outperforms previous SAH kd-tree construction [7]. Unfortunately the algorithm didn't provide methods for the case of a large space scene, the settled strategy of the space splitting may cause the quality degrading of kd-tree.

2. The Basic Idea of Ray Tracing

Standard ray tracing algorithm steps are as follows:

(1) The initial rays were shot to the screen according to the sampling strategy. Usually, each pixel samples once, that means one pixel generates only one initial ray.
(2) Judging the direction of the initial rays. If there is no intersection between the ray and the primitives in the scene, then the tracing ends and the ray shoots out of the scene; if there is an intersection, find the nearest intersection of ray-triangles, and then judge the direction of the ray at the intersection point.
(3) Classically, there are three possible trends at the intersection point: When illuminated at the ideal diffuse reflection surface, then the ray will terminate at the ideal diffuse

reflection surface, tracking ends; if the object surface is an ideal mirror, the ray will be reflected by the mirror according to the law of reflection in physics, and then continue to track the reflected ray; if the intersection of the object is a regular surface, the ray will be refracted according to the law of refraction in physics, and then continue to follow the transmission direction of light transmission.

Recognize that when the surface of the object is an ideal mirror or a regular surface, the tracing will continue on. The process of ray tracing can be realized by recursion. But infinite recursion is impossible to achieve, therefore, the final judgment condition of ray tracing must be given. Such like the following case:

(1) The ray is outside of the picture frame, that is, there is no intersection point between the ray and the primitives in the scene.
(2) The surface of the nearest object, which is at the intersection of the ray, is a diffuse reflection surface.
(3) Maximum trace is defined by user in the initial stage of ray tracing, and the number of layers is larger than the maximum number.
(4) Threshold of brightness is set by user, and the ray' contribution to the brightness value of the pixel is lower than the threshold.

2.1. Acceleration Structure

The main factor limiting the efficiency of ray tracing algorithm is the significant amount of computation, which is caused by the uncertainty of the intersection point. To determine the intersections, intersection testing and intersection calculation must be done. During the intersection testing, the naive approach tests the intersection of each given ray with every object in the scene one by one. In this way, many intersection points will finally be got, but only the nearest one is useful for this algorithm. So how to determine the nearest intersection point quickly and to avoid useless computing is the key to improve the efficiency of ray tracing algorithm.

Usually a geometric scene is indicated by the triangles, and a complicated scene usually needs tens of thousands of triangles. So effective data structure is the key to improve the performance of ray tracing algorithm. First of all, the hierarchical tree structure of the scene needs to be divided. This division of tree structure reduces the computation of light and useless triangle intersection, as well as the calculation of the light to pass through the blank space. At present, there are some acceleration structures that widely used, such as bounding volume hierarchies (BVHs), Octrees, different variants of grids, kd-trees, etc. [12, 13].

Kd-tree is a special form of binary search tree. By splitting the space of the scene, it forms a retrieval structure for geometry. The "space" here is referring to space covered by the axis aligned bounding box (AABB) of the scene. The KD-Tree's internal node represents a partition of a space, and each division generates two sub-trees, that is, two subspaces. The splitting executes on the sub-trees recursively in the same way. Leaf node is the end point of splitting. The leaf node represents a series of spaces that are ultimately subdivided, and saves the indexes of the objects which are actually intersect with the ultimate subdivided space.

As shown in Figure 1 and Figure 2, the key of kd-tree construction for a scene is the determination of the split plane in each axis. The traditional process of constructing kd-tree is splitting space of the initial scene gradually. Initially, there is only one leaf node (the root node) in kd-tree, and all the geometry belong to the same set. As the construction progressed, the geometry is split into different subsets by the split plane.

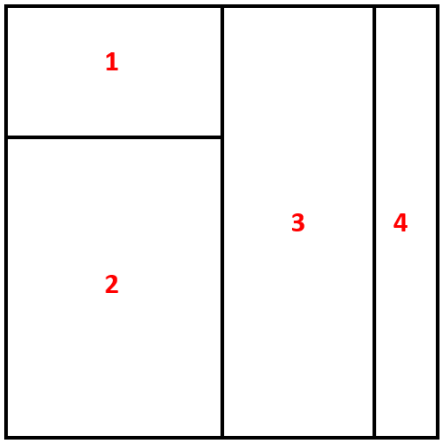

Figure 1. Example Scene Division

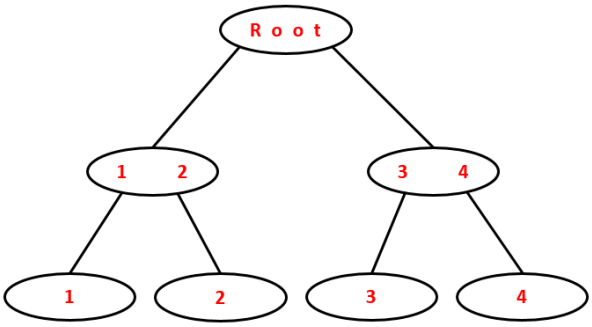

Figure 2. Corresponding Nodes in the Kd-tree

There are three common methods of splitting, spatial median, object median and SAH. Spatial median splits the space into two average in a certain dimension, usually chooses the dimension which is the largest extension of the space. But it is not applicable to the scene with a strong randomness, because it will cause the extremely uneven number of triangles between the left and right node. For those scene with highly random triangles, it will generate very low quality kd-tree. Object median algorithm chooses the splitting plane which makes the number of the geometric equally in the two subspaces. Macdonald *et al*. [8] proposed kd-tree construction based on SAH. Using a quality evaluation model based on space surface area, the SAH algorithm evaluates a set of split candidate, and then finds the best split plane.

Before splitting based on SAH, several assumptions should be made [9]:

(1) The rays are uniformly distributed, infinite lines.
(2) The cost for both a traversal step and for a triangle intersection are known.
(3) The cost of each ray-triangle intersecting are the same, and ray-triangle intersecting do not affect each other.

The advantage of kd-tree is that its essence is a binary tree, then it has the excellent characteristics of the binary tree. Meanwhile, in the process of construct a kd-tree based on SAH, heuristic splitting allows the generation of trees to be more balanced. By such heuristic split, the number of intersection between the ray and the subspace is greatly reduced, and the efficiency of the algorithm is vastly improved.

Steps of kd-tree construction based on SAH are as follows:

(1) Estimate its SAH cost for each the candidate splitting plane.
(2) Select the lowest cost of the plane, in accordance with this plane will be divided into two children node.
(3) The triangle in the node is split into the corresponding child node according to the split plane.

2.2. Morton Code

Morton code, also called z-order or Morton order, is a function which maps multidimensional data to one dimension while preserving locality of the data points [14].It was introduced in 1966 by G. M. Morton. Morton code has a very good local properties,

adjacent grid in the original space is mapped adjacent in one dimension curve (except for the singular point).

Taking 2D space as an example, each grid in space can be represented by the value of its x and y coordinates. The coordinate values expressed in binary, and Morton code can be obtained by inserting digit of Y value before each digit of X value. As showing in Figure 3. One-to-one correspondence between the grid in two dimensional space and the Morton code. The order of the grid according to the value of its Morton code. As seen from Figure 4, after ordering the grid, the curve presents "Z" shape by sequential traversal of the grid, this is the origin of the name "Z-order".

Morton code has the following two characteristics:

(1) Recursive between father and son.
This is the characteristic of all space filling curves which are generated by recursive method.
(2) Similar between brothers.
There is a simple conversion relationship between the locations of the nodes in the Z-Order (that is, Morton code) and the grid coordinates, the Morton code can be achieved by cross shift operation easily.

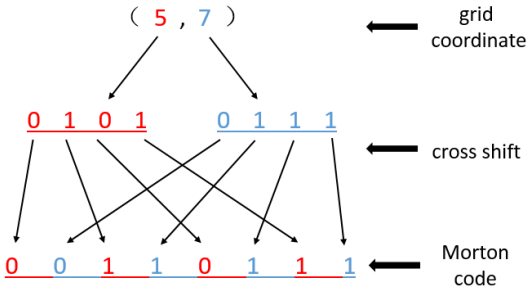

Figure 3. The Generation of Morton Code

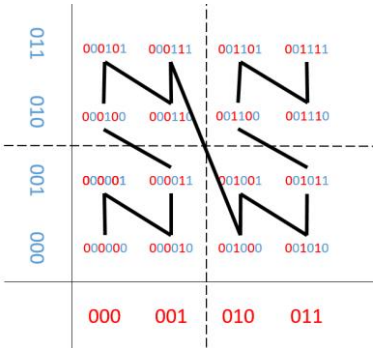

Figure 4. The Shape of Morton Order

In ray tracing algorithm, the main purpose of the application of Morton code is to parallelizing the construction of acceleration structure. The simplest approach to parallelizing the construction of acceleration structure is to reduce it to a sorting problem [15], and the Morton code corresponding to the Morton curve can be calculated directly from the geometric figures without complex construction cost. Therefore, Morton code is usually used to order the primitives in rendering.

3. Kd-Tree Construction based on Morton Code

In previous studies, kd-tree construction is starting from the root node and proceeds in a breadth-first order. When construct kd-tree in this manner on GPU, the construction of large nodes at upper tree layers is more time-consuming [10]. This is because large nodes are rare but each of them contains great number of triangles, and the upper layers is constructed with poor degree of parallelism. Besides, the parallelism can only be known after the previous level is constructed in the conventional process of kd-tree construction, this makes the programs on GPU reboot the kernels frequently and brings unnecessary overhead.

In this paper, the balanced binary tree which is very suitable for parallelization is fully utilized, and a theoretic perfect binary tree is created. Then the method used by Lauterbach *et al*. [15] is draw on to parallel construct BVH on many-core GPUs, and introduce Morton code for the scene spatial. In the last, a path compression is used to

determine the node and the node location of kd-tree from the bottom of a balanced binary tree.

In the practical application, the scene and the composition of the ray are very complex. In order to discuss the problem more conveniently, these assumptions made for the scene and rays in section 2 is still used. Moreover, we assume that each primitive in the scene is represented by its axis-aligned bounding box (AABB), and the enclosing AABB of the entire geometry is known.

3.1. The Construction of Kd-Tree based on Morton Code

In the first, a theoretic perfect binary tree is created, as showing in Figure5. Considering the Morton code which will be used latter, the height of the perfect binary tree is h, and h needs to meet the following formula:

$$2^h = n^2 \quad (1\text{-}1)$$

Where n is the number of grid in the direction of each coordinate. In Figure5, the height of the tree is 4.

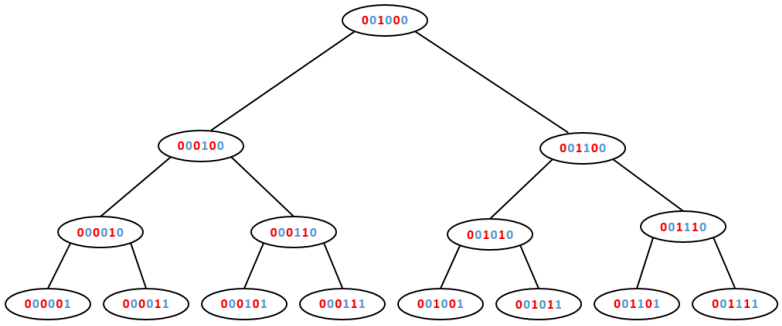

Figure 5. The Theoretic Perfect Binary Tree

Second, in the AABB of the entire geometry which is known, split the space into $n \times n$ grids evenly and assign Morton code to each grid according to Morton curve. For the perfect binary tree above, each node in the tree is also assigned to a Morton code. Thus, each node in the perfect binary tree corresponds to a grid of scene. Figure 6 shows the 4×4 splitting of the scene, and Figure 5 shows the correspondence between the node in perfect binary tree and Morton code.

After the completion of the above preparatory work, the following work is to construct kd-tree according to the primitives which are corresponding to Morton code. A path compression method on the conceptual perfect binary tree is used. According to Morton code, primitives in the same subspace are organized together naturally, then insert these subspaces into the perfect binary tree as new leaf nodes using the insertion method of binary search tree. These new leaf nodes make the perfect binary tree into a common balanced tree, and they are also the leaf nodes of the kd-tree which will be obtain in the end. Because the primitives in the scene are generally irregular distribution, there are many subspaces which don't contain any primitive. This also shows that, there are actually a lot of redundant nodes in the balanced tree. Such as node"000001" and node"000011" in Figure 7, they don't correspond to any primitives. The node finally belongs to the kd-tree is called as the "essential node", and these essential nodes can be determined by the following steps:

(1) The inserted new leaf nodes are all essential nodes; expect the new leaf nodes, all other leaf nodes of initial perfect binary tree are redundant nodes.

(2) To judge the nodes of upper level. If the ancestor node has only one essential node, then the ancestor node are redundant and can be replaced by the essential child node; if the ancestor has two essential node, then it is an essential node too.
(3) To judge every node in each level from the bottom up to the root node.

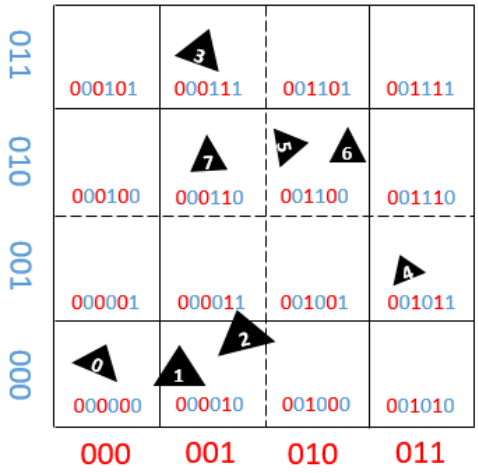

Figure 6. 4×4 Splitting of the Scene

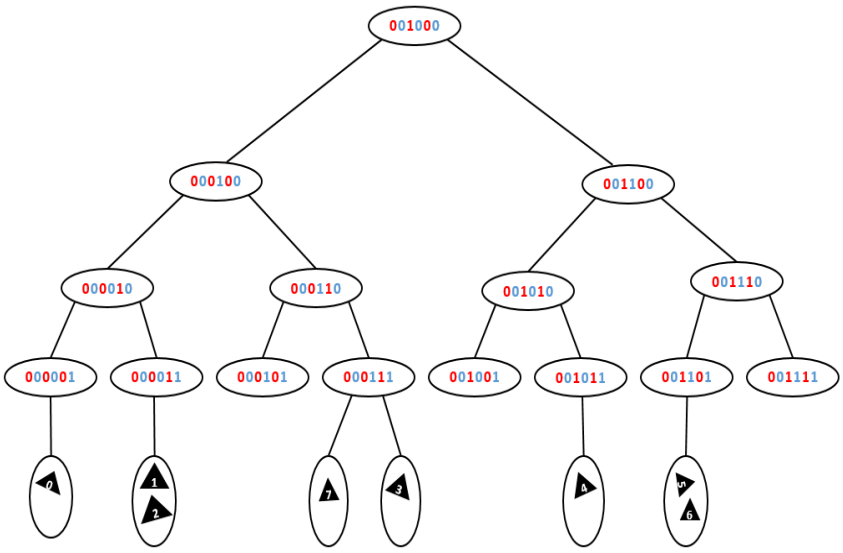

Figure 7. The Correspondence between Primitives and Morton Code

After these step, the essential nodes are connected and the redundant nodes are deleted, in this way the final kd-tree structure is obtained, and the split plane of the scene is obtained too (see in Figure 8 (d)). The whole process is shown in Figure 8.

During connecting the essential nodes, a lot of redundant nodes are deleted, the number of path in the balanced tree becomes less, so this method is called the path compression. Algorithm1 shows the process above.

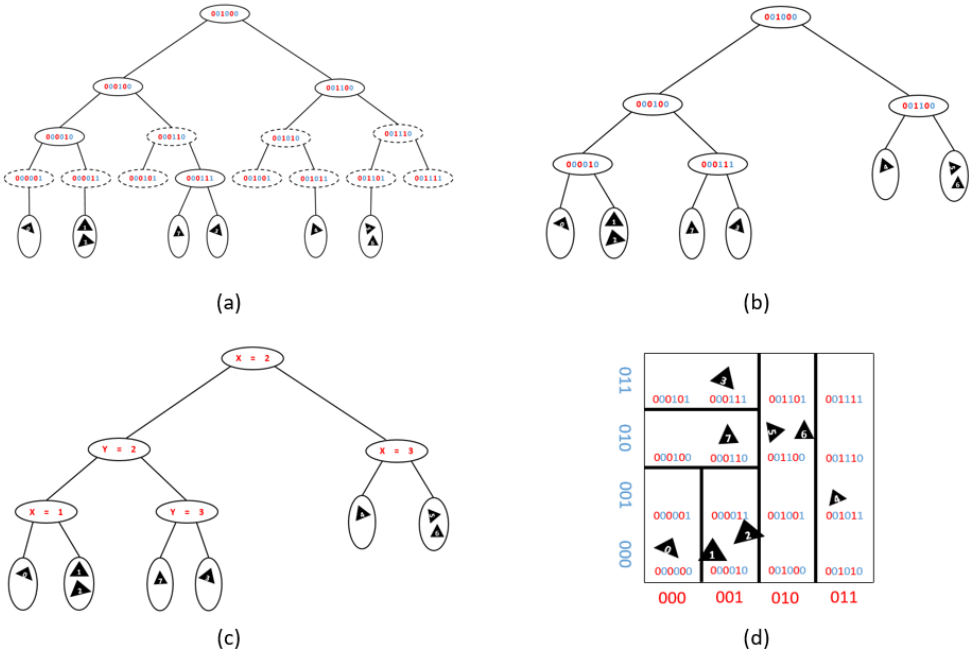

Figure 8. The Process of Kd-tree Construction Based on Morton Code

Algorithm 1. Essential Nodes Judgment

```
function JudgeEssentialNode(Node n)
begin
  /*NewLeafNode is the new leaf node which was inserted to the perfect binary tree*/
     if n is NewLeafNode then
        n is EssentialNode
        return true
     else
       if n.leftchild is EssentialNode and n.rightchild is EssentialNode then
         n is EssentialNode
         return true
       else
              n is not EssentialNode
         return false
end
```

The construction steps summarized as follows:

(1) Introduce a perfect binary tree of height *h*, then the scene is cut into $n \times n$ uniform grids, and the h and n must meet the formula (1-1).

(2) Assign a Morton code to each grid according to the location. For each the primitive in the scene, take barycenter of the triangle AABB as its representative point, so that each grid corresponds to only one grid. Each nodes in perfect binary tree are corresponded to a grid of the scene, and the corresponding Morton code is taken as the value of the node.

(3) At last, insert these grids into the perfect binary tree as new leaf nodes using the insertion method of binary search tree, then determine all the essential node and delete the redundant node. The parent-child relationships between essential nodes are determined by Morton code too. Connect the essential and the kd-tree is obtained.

In this paper, rather than constructing kd-tree layer-by-layer from the top, a theoretic perfect binary tree was proposed. Considering that in the actual rendering, the scene to be

processed can be very large and contains lots of primitive, the space can be cut off more precisely based on the formula (1-1), such as 16×16 or more splits. Although more splits may make the kd-tree a little higher, but the rendering result for large scene can be much finer.

3.2. Kd-tree based on Morton Code Construction on GPU

By using the Morton code and the theoretic perfect binary tree, kd-tree can be constructed from bottom up. For details, the essential node judgment and path compression for each leaf nodes can be executed in parallel. Consequently, maximized parallelism can be received on GPU (see Algorithm 2).

Algorithm 2: Parallel Algorithm

```
Function FindEssentialNodes(BalanceTree T)
begin
      nodelist←new list
      kdlist←new list
      for each node tn in T.node
        if tn is leafnode then
          nodelist.add(tn)
      for each node nd in nodelist in parallel
      if JudgeEssentialNode(nd) is true then
                kdlist.add(nd)
      if nd is not RootNode then
              nd←nd.parent
       return kdlist
end
```

4. Experimental Results

4.1 Experimental Environment

Results from the algorithms described in the previous sections are now highlighted on several benchmark cases and scenarios . All algorithms run on a graphic workstation running Microsoft Windows 7 Professional with an Intel Core i7-2600 at 3.40GHz and a NVIDIA Quadro M4000 graphics card with 8 GB of memory, and the experimental results were implemented using the NVIDIA CUDA programming language. Several benchmark scenes of Stanford University are used to allow comparison to other published approach (see in Figure 9). The number of triangles in each scene is known, and all scenes were tested under a resolution of 1024×1024 with only primary rays.

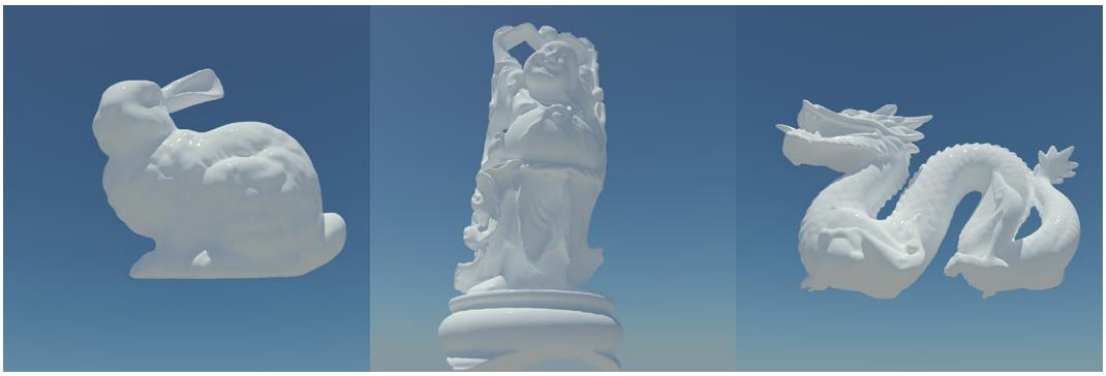

Figure 9. Rendering Results Based on Our Work

4.2 Experimental Results and Analyses

The experimental results were compared with the SAH kd-tree construction on GPU proposed by Wu *et al*. [7]. The comparison of construction time is reported in Table 1. By Table 1, the conclusion is that our algorithm is slightly faster than algorithm of literature [7] on the small testing scenes, such like Bunny, but significantly faster on the larger testing scenes, such like the Happy Buddha and the Dragon. Figure 10 can show this contrast intuitively.

Table 1. The Compare between GPU SAH Kd-tree and Our Work

Scene	Triangles	GPU SAH KD-Tree	Our Work
Bunny	69,451	0.031s	0.023s
Dragon	1,132,830	0.264s	0.043s
Happy Buddha	1,087,716	0.323s	0.051s

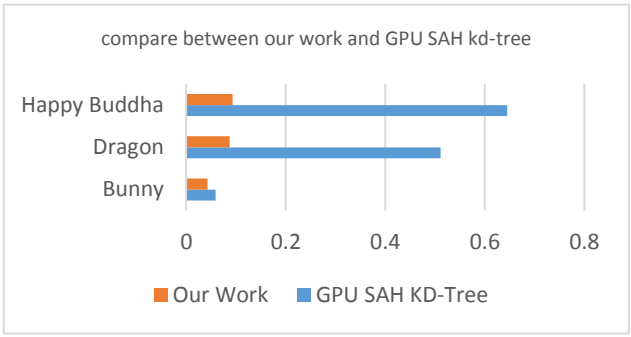

Figure 10. Compare between Our Work and GPU SAH Kd-tree

5. Conclusion and Future Work

In this paper, a faster parallel kd-tree construction algorithm for ray tracing have been presented. The algorithm takes advantage of a conceptual perfect binary tree and Morton code to avoid the problem that the upper layers is constructed with poor degree of parallelism when using algorithm based on SAH. It is different from previous algorithms which are constructing kd-tree from top down. The experiments proved that the performance of this method is better than the parallel construction of kd-tree based on SAH, especially when rendering large scenes on GPU.

The method of kd-tree construction proposed in this paper is only tested on GPU of a single machine. When GPU is busy with essential nodes judgment and path compression, the CPU utilization is very low. In order to achieve higher rendering quality on the limited hardware resources, we would like to classify the computing tasks to combine the computing resources of CPU and GPU. In addition, cluster and cloud computing has a strong parallel computing power, it would be interesting to extend our approach to work for cluster rendering and cloud rendering.

Acknowledgements

This paper is supported by National Natural Science Foundation of China (NFSC) under Grant No.61502294, Natural Science Foundation of Shanghai under Grant No.15ZR1415200, Innovation Project of Next Generation Internet Technology of CERNET under Grant No. NGII20150609, Foundation of Science and Technology Commission of Shanghai Municipality under Grant No. 14590500500, and The Special Development Foundation of Key Project of Shanghai Zhangjiang National Innovation Demonstration Zone under Grant No. 201411-ZB-B204-012.

References

[1] T. Whitted, "An improved illumination model for shaded display", Communications of The ACM, vol. 23, no. 6, **(1980)**, pp. 343-349.
[2] M. Zdippe and J. Swensen, "An adaptive subdivision algorithm and parallel architecture for realistic image synthesis", Computer Graphics, vol. 18, no. 3, **(1984)**, pp. 149-158.
[3] A. Appel, "Some techniques for shading machine renderings of solids", American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, **(1968)**, pp. 37-45.
[4] R. L. Cook, T. Porter and Loren Carpenter, "Distributed Ray Tracing", Computers & Graphics, vol. 18, no. 3, **(1984)**, p. 533.
[5] J. Tkajiya, "The rendering equation. Computer Graphics", vol. 20, no. 4, **(1986)**, pp. 143-150.
[6] E. Reinhard and F. Wjansen, "Rendering large scenes using parallel ray tracing", Parallel Computing, vol. 23, no. 7, (**1997**), pp. 873-885.
[7] Z. Wu, F. Zhao and X. Liu, "SAH KD-tree construction on GPU", ACM Siggraph/eurographics Conference on High PERFORMANCE Graphic, **(2011)**, pp. 71-78.
[8] D. Jmacdonald and K. Sbooth, "Heuristics for ray tracing using space subdivision", The Visual Computer, vol. 6, no. 3, **(1990)**, pp. 153-166.
[9] I. Wald and V. Havran, "On building fast kd-Trees for Ray Tracing, and on doing that in $O(NlogN)$. 2006 IEEE Symposium on Interactive Ray Tracing, IEEE Computer Society, **(2006)**, pp. 61-69.
[10] K. Zhou, Q. Hou and R. Wang, "Real-time KD-tree construction on graphics hardware", ACM Transactions on Graphics, vol. 27, no. 5, **(2008)**, p. 126.
[11] Z. Li, T. Wang and Y. Deng, "Fully parallel kd-tree construction for real-time ray tracing", Meeting of the ACM SIGGRAPH Symposium on Interactive 3d Graphics and Games, **(2014)**, pp. 159-159.
[12] J. Arvo and D. Kirk, "A survey of ray tracing acceleration techniques", An Introduction to Ray Tracing, **(1989)**.
[13] V. Havran, "Heuristic ray shooting algorithms", PhD thesis, Czech Technial University in Prague, **(2001)**.
[14] https://en.wikipedia.org/wiki/Z-order_curve, 2016/5/20.
[15] Lauterbach C, Garland M, Sengupta S. Fast BVH Construction on GPUs. Computer Graphics Forum, vol. 28, no. 2, **(2009)**, pp. 375-384.

Authors

Zhang Peicheng, he is the author is a Master candidate in graduate students in School of Computer Engineering and Science Shanghai University. His main research is about multimedia and cloud computing.