A Fast Kd-tree Construction for Ray Tracing based on Efficient Ray Distribution

Xiao Liang[1,2]
[1]College of Computer Science, Sichuan University, Chengdu, China
[2]College of Computer Science, Southwest Petroleum University, Chengdu, China
Email: xiaoliang.edu@gmail.com

Hongyu Yang[1,2] and Yinling Qian[1,2] and Yanci Zhang[*1,2]
[1]College of Computer Science, Sichuan University, Chengdu, China
[2]National Key Laboratory of Fundamental Science on Synthetic Vision, Chengdu, China
Email: yanghongyu@scu.edu.cn, arthurqian@foxmail.com, yczhang@scu.edu.cn,

***Abstract*—Construction of effective acceleration structure is an essential and challenging problem in ray tracing. The surface area heuristic (SAH), regarded as the standard cost function for construction, is based on the assumption that rays are uniformly distributed. This simplification ignoring actual ray distribution results in a reduction both on construction and rendering performance. Unlike previous methods, we exploit ray distribution during construction in two steps. First, we propose an improved cost metric for constructing an efficient kd-tree by exploiting the visible primitives to approximate the ray distribution. Then, we device a stream based partial construction to prune the invisible primitives from building as early as possible, and improve memory access coherence. We also introduce a termination criterion for two-levels hierarchical construction to balance the construction time and memory consumption. Our experiments demonstrate that the algorithm can produce a kd-tree more efficient than the standard SAH, and a significant reduction on construction time and memory consumption.**

***Index Terms*—ray tracing, the SAH, ray distribution, kd-tree, partial construction**

I. Introduction

Ray tracing has long been an important method for realistic rendering using global illumination simulation, but was limited to static scenes due to its enormous computational demand. Visibility computation is the primary bottleneck in ray tracing, which determines the surface in scenes visible from eyes and light sources. Since ray tracing requires global access to the entire scene, it is challenging to design an optimized acceleration structure to make visibility queries more efficient.

The kd-tree is a well-known space partitioning data structure for ray tracing. Traditionally, the surface area heuristics (SAH) [2] is considered as the standard criterion. It computes expected traversal cost for ray tracing and minimizes the cost by selecting the optimal splitting plane. The SAH cost function is built on the assumption that rays are uniformly distributed, infinite lines in space. The simplification ignoring the actual ray distribution entirely, an important characteristic of scenes [10], would bring two problems. First, computation of traversal cost without considering the ray distribution tends to reduce the accuracy of estimation. Second, the SAH constructs an acceleration structure for the whole scene. However, for most scenes, especially complex or high-occluded scenes, only part of scene contribute to the result image. Although completed construction is trivial for small-scale scenes, when the count of primitives grows up to more than 100K, the construction takes over too much time in each frame, which prevents ray tracing from interactive and dynamic scenes.

In this paper, unlike the conventional SAH methods, we present a kd-tree building algorithm by making use of the actual ray distribution during all the construction. First, we introduce a new cost metric, which achieves higher rendering performance than conventional SAH computation by exploiting the distribution of visible primitives to approximate the distribution of rays. Then, we use a stream based partial construction to prune the invisible primitives from building. Moreover, we also provide a termination criterion for two-level hierarchical construction to balance the construction time and memory consumption.

II. Related work

The well-optimized SAH [1] computes the traversal cost at the maximum and minimum projection boundary for each primitive, and selects the slitting plane with the least cost. This construction produces a high quality kd-tree, but cannot be applied in interactive scenes for ray tracing due to expensive computation, even for moderate scenes. Wald et al. [3] provided an $o(n \log n)$ algorithm that sorted the bounding box extents of primitives in three coordinate axes only once, and preserved and reused the sorted list during construction. However, the algorithm still spend much time in large scenes.

Some fast algorithms are proposed as a trade-off between tree quality and construction time. Hunt et al. [5]

doi:10.4304/jsw.9.3.596-604

approximated SAH cost function with a piecewise quadratic function. Popov et al. [6] linearly approximated the SAH with uniformly distributed samples. These methods only estimate the SAH cost in discrete binned positions, then, interpolate the SAH cost function between the bins. Fan et al. [13] developed a heuristic algorithm to position the preferable planes.

Recently, parallel construction [11, 12, 19] have been received much attentions. Generally, these algorithms build a kd-tree in two steps, that is, construction of a top tree with a fast scheme and a bottom tree with an optimized SAH criterion. Shevtsov et al. [11] realized 4-core parallel construction, but the quality of kd-tree was degenerated by 15% as it used triangle-count middle splitting in top level stage. Zhou et al. [12] introduced a real-time kd-tree algorithm on the GPU, whose 128-core GPU version was 4 ~ 7 times faster than single-core algorithm. However, since the method adopts spatial middle splitting, rendering performance is reduced by 15% once applied in large-scaled scenes with more than 100K primitives.

Some methods tend to avoid expensive reconstruction acceleration structure. Hunt et al. [20] employed a BVH (Bounding Volume Hierarchy) based scene graph to accelerate construction. Based on motion decomposition, Gunther et al. [21] constructed a kd-tree over fuzzy boxes instead of primitives themselves. But, these methods are limited to some particular scenes or assumptions.

Although above algorithms can generate a high-quality or preferable kd-tree through the standard SAH or approximation, the assumption that rays are uniformly distributed and infinite lines essentially degenerates the rendering performance. Some algorithms exploit the ray distribution to compute more optimized splitting planes.

Unlike traditional assumptions, Fabianowski et al. [8] assumed that the origins of rays are uniformly distributed inside the scenes, and developed an alternative estimation model. It computes the probability of entering a bounding box for each potential origin of ray with the fractional solid angle subtended by the box. However, the improvement of rendering performance is just 5% than the SAH. Bittner et al. [9] used the count of rays to estimate the probability of rays traversing the tree nodes, but only achieved a minor speedup. Marek et al. [23] introduced a visibility driven algorithm to build BVH. Choi et al. [10] established voxel-visibility based models for primary rays and secondary rays respectively. Their new approach helps to produce a kd-tree with 40% performance improvement for static scenes, which demonstrates that the visibility is the crucial factor for ray tracing. Aiming to accelerate shadow ray traverse, Feltman et al. [15] developed a shadow ray distribution heuristic method which improved performance by 20%.

III. COST METRIC BASED ON RAY DISTRIBUTION

In this section, we analyze and re-estimate traversal cost by considering the distribution of rays. Then, we generate a cost metric, which computes a more optimized splitting plane by exploiting the visible primitives to approximate the distribution of rays.

A. Estimation Of Traversal Cost With Ray Distribution

As a standard construction for kd-tree, the SAH cost metric estimates the expected cost for traversing a voxel by considering many features which influence space partition [3]. Given the cost for both traverse and for ray-triangle intersection K_T and K_I, the average traversal cost is defined as follows:

$$C = K_T + \frac{SA_L}{SA_V} K_I N_L + \frac{SA_R}{SA_V} K_I N_R \qquad (1)$$

Where SA_L and SA_R are the surface areas of left and right child given a candidate splitting position, SA_V is the overall surface area of the voxel for splitting, N_L and N_R are the count of primitives in children. The ratio SA_L / SA_V and SA_R / SA_V are probabilities that ray intersects with children, and are denoted as P_L and P_R for short.

The SAH criterion is based on the assumption that the rays are uniformly distributed, infinite lines through space, which violate the ray distribution in actual scenes. Because, the rays are usually irregular distributed and are possible be blocked due to intersecting with opaque objects. The splitting plane computed by the SAH can not be considered as the best split for such scenes.

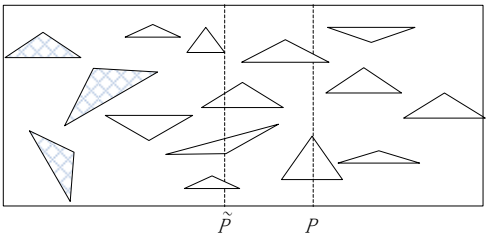


Figure 1. A scene with irregularly distributed rays. Only colored primitives are visible.

Fig.1 shows an instance that only part of the scene (marked by the colored primitives) is visible as a result of irregularly distributed ray distribution. Assuming that P is the best splitting plane computed by the standard SAH, $\tilde{P}$ is a candidate splitting plane. It is noted that $\tilde{P}$ is more near than P to the visible region. Notation N is indicated the count of primitives in a node for splitting, N_L and N_R are defined as the same as above. Let N^V as the count of visible primitives. In Fig.1, it is obviously showed that $N^V \in N_L$.

Then, let $\Re$ as the set of rays that intersect with the node, and M as the number of rays in set $\Re$. M_{OL}, M_{OR}, M_{LR} are the number of rays entering into left child only, right child only and both children, respectively. Set $\Re^V$ is the set of rays that intersect with the primitives, and M^V is the number of rays in $\Re^V$.

The objects are assumed to be opaque, hence all rays in set $\Re^V$ are blocked by primitives after intersection. It is noted that $M^V \in M_{OL}$.

For set $\Re$, the total traversal cost is:

$$Cost_P = K_T M + K_I (M_{OL} N_L + M_{OR} N_R + M_{LR} N) \quad (2)$$

Because only set $\Re^V$ actually intersect with primitives, the ideal traversal cost is:

$$Cost_I = K_T M^V + K_I M^V N^V \quad (3)$$

which is quite smaller than $Cost_P$.

Let M'_{OL} as the rays that enter into left child without intersecting with any primitives, so it is noted that $M_{OL} = M'_{OL} + M^V$. Let N'_L as invisible primitives in left child, so $N_L = N'_L + N^V$.

Therefore, the cost difference between $Cost_P$ and $Cost_I$ is as follows:

$$\Delta_P = K_T (M - M^V) + K_I (M'_{OL} (N'_L + N^V) + M^V N'_L + M_{OR} N_R + M_{LR} N) \quad (4)$$

Similarly, for candidate splitting plane $\tilde{P}$, the difference between $Cost_{\tilde{P}}$ and $Cost_I$ is as follows:

$$\Delta_{\tilde{P}} = K_T (M - M^V) + K_I (\tilde{M}'_{OL} (\tilde{N}'_L + N^V) + M^V \tilde{N}'_L + \tilde{M}_{OR} \tilde{N}_R + \tilde{M}_{LR} N) \quad (5)$$

Since the rays entering into right child, as in Fig. 1, are quite few, M_{OR} and M_{LR} can be considered as constants when the splitting plane is shifted from P to $\tilde{P}$. Therefore, the change of $M_{OR} N_R + M_{LR} N$ is ignorable. The difference between $Cost_P$ and $Cost_{\tilde{P}}$ can be approximated as:

$$\Delta \approx K_T ((M'_{OL} N'_L - \tilde{M}'_{OL} \tilde{N}'_L) + (M'_{OL} - \tilde{M}'_{OL}) N^V + M^V (N'_L - \tilde{N}'_L)) \quad (6)$$

Because the value of $N'_L - N'$ is difficult to estimate, which is determined by the distribution of primitives. We assume the primitives are uniformly distributed. As $M'_{OL} > \tilde{M}'_{OL}$ and $N'_L > \tilde{N}'_L$, it is viewed that $Cost_P > Cost_{\tilde{P}}$.

From above analysis, it is observed that the SAH is not applicable in building high-quality acceleration structure when the rays are irregularly distributed. Moreover, according to formula (4) and (5), we find that the distribution of visible primitives N^V, which influence the rendering performance, can be exploited to improve quality of kd-tree tree.

B. Improved Cost Metric

Actually, the distribution of rays is influenced by many elements, including the size of primitives, the materials, the position of camera and light sources and so on. In this paper, instead of establishing an exact analytic expression, we suppose the distribution of visible primitives is known, and use it to approximate actual distribution of primary rays. According to frame coherency, the prior knowledge can be obtained straightly from last frame, hence, the assumption is practicable.

According to above analysis, the traditional probability computation is not applicable in scenes, especially for high-occluded scenes. We identify such kind of probability as P^{SAH}, and introduce another probability computation P^{Vis} to take actual rays distribution into consideration.

Let N^V to be the number of visible primitives of node for splitting as previous, N_L^V and N_R^V to be the number of visible primitives for left and right child. The probability P^{Vis} is defined as follows:

$$P^{Vis} = N_{L/R}^V / N^V \quad (7)$$

For a high-quality kd-tree, we use bounding boxes of primitives as the splitting candidates. In order to count the visible primitives, we introduce some notations first. $N_L^V(i)$ and $N_R^V(i)$ are the count of visible primitives in left and right side relative to the *i*th candidate plane. Used as proxies for primitives, $E(i)$ and $S(i)$ are End and Start events [3] for current candidate plane. $D(i)$ is to define whether the event is visible. Before each SAH computation, N^V is counted through a global array *g_primsvislist*, which is used to record whether the corresponding primitive is visible or not in rendering stage. $N_L^V(0)$ and $N_R^V(0)$ are initialized to 0 and the value of N^V respectively. When sweeping along each sample location, $N_{L/R}^V(i)$ is computed incrementally as the following formula:

$$N_L^V(i) = N_L^V(i-1) + Vis(i), Vis(i) = \begin{cases} 1, & \text{if E(i) is visible} \\ 0, & \text{or else} \end{cases}$$
$$N_R^V(i) = N_R^V(i-1) - Vis(i), Vis(i) = \begin{cases} 1, & \text{if S(i) is visible} \\ 0, & \text{or else} \end{cases} \quad (8)$$

Currently, we have two kinds of probabilities, they are, P^{SAH}, dominated by the spatial size of primitives, and P^{Vis}, mainly determined by the distribution of visible ones. Because both probabilities represent different characteristics of scenes, which influence the rendering performance, they are employed in our cost metric. The cost metric is defined as follows:

$$P_{L/R} = Weight\_Vis \cdot P_{L/R}^{Vis} + (1 - Weight\_Vis) \cdot P_{L/R}^{SAH}$$
$$C = K_T + K_I (P_L N_L + P_R N_R) \quad (9)$$

Weight_Vis is a coefficient varying from 0 to 1, which highly depends on the ray distribution of scenes. If the rays are nearly regularly distributed, meaning most of the primitives are visible, we set *Weight_Vis* to 0, and use P^{SAH} instead of P^{Vis}. This is because the standard SAH computation is applied quite well in such condition. On the other side, if the rays are extraordinarily irregular,

that is, a great deal of primitives in current node are occluded, we resort to P^{Vis} mainly aiming to separate invisible part from other parts of acceleration structure. It is noted that we set P^{Vis} to a higher value but never to 1, because the characteristics of spatial size of primitives, presented by P^{SAH} cannot be totally ignored.

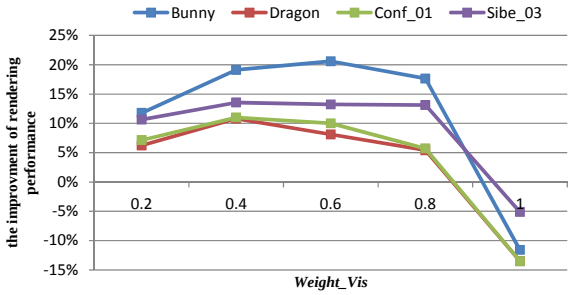


Figure 2. The relationship between *Weight_Vis* and improvement of rendering performance.

For other situations, the probability is the result of linear interpolation between P^{SAH} and P^{Vis}. Fig.2 sketches the relationship between $Weight\_Vis$ and improvement of rendering performance. We test scenes showed in Fig. 10 and Fig.11 with varying primitive distribution. It is suggested the rendering time changes little when $Weight\_Vis$ belongs to region [0.3, 0.8]. For all tested scenes, the minimum of rendering time happens when $Weight\_Vis$ belongs to region [0.4, 0.6]. Moreover, a significant reduction occurs when $Weight\_Vis$ approaches to 1, which demonstrates our previous analysis.

IV. Partial Construction Based on Ray Distribution

Expensive construction prevents ray tracing from dynamic scenes. In this section, we focus on reducing construction time and memory consumption by excluding invisible primitives as early as possible.

A. Partial Construction of Kd-tree

Previous researchers mainly use two approaches to construct a fast kd-tree. One is called comparison based algorithm, which builds kd-tree in $O(n\log n)$ - the theoretical lower bound. Another is binning based algorithm, which only places SAH samples in fixed positions. Although it uses an $O(m)$ radix sorting instead of an $O(n\log n)$ sorting, the construction time is still $O(n\log n)$ since each node needs similar computation. The above methods share the common point that they build the acceleration structure for the entire scene.

An observation is that construction time increases exponentially with the growing scale of primitives. Actually, the count of visible primitives which contribute to the final image does not change severely, especially in large scenes. Let v to be the count of visible primitives and n the count of primitives in scene, it is a feature for complex or high-occluded scene that $v << n$. Therefore, we tend to only construct the primitives visible to reduce both the time and memory consumption significantly.

The visibility of primitive is only known during rendering. Accordingly, we separate kd-tree construction into two phases and interleave it with rendering stage, rather than construct an overall kd-tree at once. In the first phase, the top tree is built in top-down manner. Then, the algorithm switches into rendering stage. We use *early primitives excluding* to avoid invisible primitives from constructing. Only when a ray intersects with the leaf of top tree, the second phase is triggered to construct a more refined kd-tree. Therefore, the algorithm refines the kd-tree gradually according to the actually ray distribution, and tends to avoid constructing invisible primitives.

Additionally, the cost to access memory ought to be considered carefully [24], especially for large scenes. This is because nowadays hardware architecture presents a distinct gap between sequential and random memory access pattern. An algorithm that tends to access data which has recently been accessed will benefit from caches and memory hierarchies. Conventional depth-first order construction and ray tracing belong to random access pattern. We devise a streaming based partial construction combined with rendering process as described in Fig.3.

```
1:  Procedure PartialConstruction(ray, node)
2:  begin
      // streaming construction for top tree
3:    toptree = kdtreebuild(PrimitivesSet)
      // early primitives excluding
4:    while (RaysSet != NULL) do
5:      leaf = Intersection(ray, toptree);
6:      if (leaf != NULL) then
          // the ray intersects with the leaf of top tree
7:        if (leaf.root == NULL) then
8:          leaf.flag = 1;
9:        end if
10:       leaf.rayslist.add(ray.ID, tmin, tmax);
11:     end if
12:   end while
      // streaming construction for bottom tree
13:   for each leaf with flag = 1 do
14:     kdtreebuild(leaf);
15:   end for
      // intersection between rays and the refined kd-tree
16:   for each leaf with flag = 1 do
17:     for each ray in leaf.rayslist do
18:       Intersection(ray, leaf)
19:     end for
20:   end for
21: end
```

Figure 3. Partial construction algorithm based on ray distribution.

We use breadth-first construction instead of depth-first order in *kdtreebuild()* similar with [6]. After constructing the top tree, *early primitives excluding* is applied to filter the primitives to send to next construction phase. Meanwhile, we also record the *ID* of each ray intersecting with leaf of top tree as well as intersection information, including *tmin* and *tmax*. After constructing the bottom tree, we access rays sequentially stored in *rayslist* for

each leaf of top tree, which improve memory access coherence for primitives. It is observed that the algorithm can also be executed in parallel platforms with little modification, including SIMD, multi-core CPUs and GPU.

B. Termination Criterion For Hierarchical Construction

Previous completed kd-tree building often use constant threshold as termination criterion for hierarchical construction. Shevtsov et al. [11] completes the first stage once the number of primitives is under the number of bins. Hunt et al. [5] switches to second stage once the number of primitives in node is less than a threshold number. Fan et al. [13] even considers a constant spatial size. These criterions are not self-adaptive once the distribution of geometry is changed. Moreover, for partial construction, the depth of the upper tree is a more important factor, as too large or small value might cause a nearly completed kd-tree building. Therefore, the value should be carefully chosen so as to provide a good balance between time and memory consumption.

We define the *MaxDepth* of kd-tree to be $8+1.3\log_2 N$, which is referenced from pharr et al. [4] and Kang et al. [16]. We evaluate the upper tree depth in different values to observe how it influences construction time and memory. Top tree depth is expressed as *MaxDepth/2* + *Step*, where *MaxDepth*/2 is a baseline and *Step* is a variable relative to baseline. Fig. 4 and Fig. 5 display the results, and Table I provides the count of visible primitives in tested scenes. All the scenes in the paper are classified as two types. The one is scenes with regularly distributed geometry and low occlusion. The other is complex or high-occluded scenes with irregularly distributed geometry. They are showed in Fig. 10 and Fig. 11, respectively.

It is viewed that the time and memory requirement are both in a nearly U shape distribution for partial construction. For moderate scenes, the minimum almost appear in *MaxDepth/2*. For large-scale scenes, such as Conf_01, the minimum tends to occurs on the right of baseline. That means, for moderate scenes, *MaxDepth/2* can be regarded as an applicable termination criterion for top level tree, and for large scenes, a little higher value should be better.

TABLE I.
COUNT OF VISIBLE PRIMITIVES

Scenes	Count of visible primitives
Sponza	10244
Conf_01	4128
Conf_03	2817
Sibe_03	6662

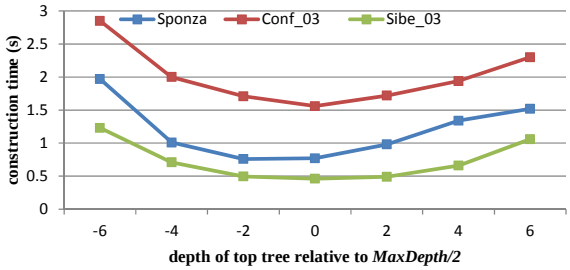


Figure 4. Construction time influenced by the depth of top tree.

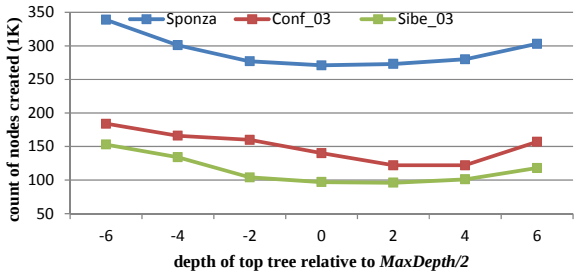


Figure 5. Memory consumption influenced by the depth of top tree.

Additionally, if the count of visible primitives for a node in top tree is lower than some threshold, we also stop to build it. Our cost metric is prone to separate the invisible portion from other portion. According to frame coherency, it is advisable to apply our cost metric in top tree levels to postpone the node with uncertain visibility to the second stage, which tends to reduce construction time and memory.

V. EXPERIMENTAL RESULTS

We implement the above approach on machine with Intel I3-2100 3.00 GHz CPU, equipped with 4.0G memory and an NVidia GeForce GTX 450 graphics card. All the result image is produced in the resolution of 800x600. Three different types of tests have been carried out, they are, completed construction on new cost metric, partial construction on the SAH, and partial construction on new cost metric. Also, we have realized optimized SAH-based construction [3] as a benchmark algorithm. For fair comparison, all methods are set to the same construction parameters. The primitives of leaf is less than 4. The maximum tree depth is $8+1.3\log_2 N$. The traversal cost and intersection cost is 1 and 60, respectively.

A. Rendering Performance With Improved Cost Metric

First, we compare our cost metric with the standard SAH. Table II summarizes the comparison results with low-occluded scenes of varying geometry complexity with primitive count range from 70K to 300K, as shown in Fig.10. It is observed that our cost metric improve rendering performance consistently for all tested scenes. To understand more profound behaviors of kd-tree, we

also inspect average search depth, times of ray-triangle intersection as well as successful ratio that rays intersect with primitives, which are essential factors influencing rendering performance. For Bunny, Dragon and Armadillo, the rendering performance is improved by 20%, 11% and 19%. Since the average search depth stay mostly the same with benchmark algorithm, for such kind of tested scenes, our method cannot increase the level of visible portion in hierarchical tree. But we achieve in performance primarily for a reduction on ray-primitive intersection and a significant increase in successful hit ratio. Particularly, the hit ratio almost is doubled in our cost metric. In addition, for Fairy, whose distribution of primitives is not as regular as other scenes, the improved rendering performance is attributed to the reduction of average search depth and times of ray-primitive intersection. Therefore, our algorithm can produce a more efficient kd-tree than the standard SAH cost.

For scenes with a high occlusion, we test our cost metric in different viewpoints for Conference and Sibenik, showed in Fig. 11 (a) ~ (d), (e) ~ (h). Table III illustrates the comparison results. We improve rendering performance for all test scenes, primarily due to the obvious reduction in average search depth. For most scenes, Fig. 7 shows that our algorithm provides 12% to 18% decrease in average search depth. This means, by exploiting distribution of visible primitives, our cost metric is prone to generate visible primitives in upper node of tree. For example, for all tested viewpoints for Conference, our method offers a speedup between 8% ~ 12% for the main reason that the average search depth is decreased by 13% ~ 18%. Particularly, it is observed that our algorithm perform quite well in reducing average search depth for Conf_02, which possesses with the property of high occlusion and complexity. For all viewpoints from Sibenik, our method provides similar performance improvement, except Sibe_02, as it is not satisfied with the statement that our algorithm performs well only if $v << n$.

B. Partial Construction With The SAH

We compare partial construction with completed construction. Here, we use the SAH in partial construction. We test moderate scenes and large-scale scenes, as showed in Fig. 11. For Bigguys, it consists of 128 Bigguys with 368,234 primitives in all. Fig. 8 and Fig. 9 describe the comparison in term of construction time and memory consumption. For these scenes, our algorithm is almost 2 to 4 times faster than the completed construction, meanwhile reduces significantly in memory requirement. For example, for Bigguy_02 and Sibe_02, which only contain few visible primitives, our algorithm almost speeds up the construction more than 50%, and restricts the memory consumption to a very low value. As the experimental results illustrate, partial construction performs quite well for scenes which meet the requirement that $v << n$. Moreover, the running time and memory requirement do not increase severely with the expanding of the scale of scenes.

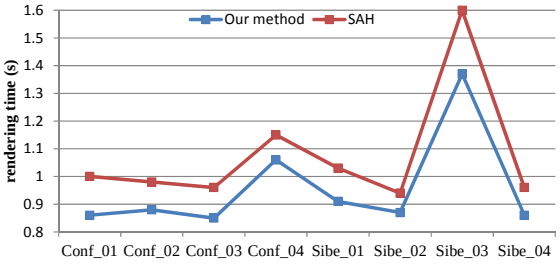


Figure 6. Comparison on rendering time.

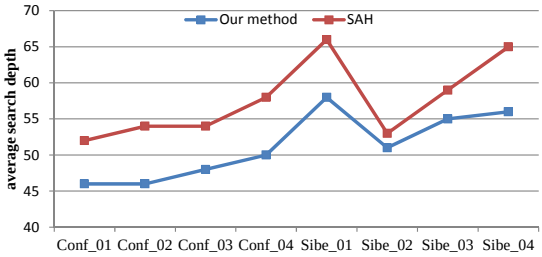


Figure 7. Comparison on average research depth.

TABLE II.
THE COMAPRISION BETWEEN OUR COST METRIC AND THE SAH FOR SCENES WITH REGULARLY DISTRIBUTION PRIMITIVES AND LOW OLLCUSION

Scenes	Rendering Time (s)		Average Search Depth		Times of Ray-primitive Intersection		Successful Hit Rate	
	Our method	SAH	Our method	SAH	Our method	SAH	Our method	SAH
Bunny	0.28	0.34	42	42	999,885	1,637,046	17.68%	9.89%
Dragon	0.33	0.37	53	53	1,090,330	1,720,314	15.11%	8.65%
Armadillo	0.28	0.35	49	47	1,441,094	2,741,665	12.05%	5.54%
Fairy	0.62	0.7	58	61	3,473,511	3,889,453	10.5%	9.18%

C. Partial Construction With Our Cost Metric

Finally, we incorporate our cost metric into partial construction. Two cost models are used according to ray distribution. Generally, we use new cost metric in top tree levels and the SAH in bottom tree levels as the rays can be regarded as nearly uniformly distributed. As showed in Table III, our approach achieves in rendering performance as described previously, meanwhile, preserves the advantages of partial construction. However, for some scenes, it performs a less efficient improvement in construction, as new cost metric will produce more nodes than SAH computation.

VI. Conclusion

In this paper, we have proposed a new cost metric for constructing an efficient kd-tree. Unlike previous approaches, we compute the optimal splitting plane by exploiting the visible primitives to approximate the distribution of rays. Then, we introduce a stream based partial construction to prune invisible primitives and improve memory access coherence. We have analyzed termination criterion for two-levels hierarchical construction to balance the construction time and memory consumption. The experiments demonstrate that our algorithm can produce an more efficient kd-tree than the standard SAH computation, meanwhile provide significant reduction in construction time and memory.

There are several directions for future work. First, the characteristics of scene play an important part during ray tracing. We plan to exploit both primary and secondary rays during construction. Then, construction of object-partition based acceleration structures, eg. BVH, are able to be speeded up by making use of ray distribution. Finally, as there exists code and data concurrency in stream based partial construction, a parallel algorithm is an interesting topic in future work.

Acknowledgment

The authors wish to thank the anonymous reviewers. This work was supported in part by a grant from National Natural Science Foundation of China (No. 60903118), and National Key Technology R&D Program of the Ministry of Science and Technology (No. 2012BAH62F03), and Youth Foundation of Southwest Petroleum University (No. 285).

TABLE III.
THE COMAPRISION BETWEEN OUR COST METRIC AND THE SAH FOR HIGH-OCCLUDED SCENES

Scene	Rendering Time (m)		Average Search Depth		Times of Ray-primitive Intersection		Successful Hit Rate	
	Our method	SAH	Our method	SAH	Our method	SAH	Our method	SAH
Conf_01	0.86	1.01	46	52	2,728,479	3,317,793	20.27%	16.9%
Conf_02	0.89	0.98	46	54	3,090,823	3,658,523	17.72%	15.1%
Conf_03	0.85	0.96	48	54	3,260,867	3,881,159	15.38%	13.1%
Conf_04	1.06	1.15	50	58	4,003,090	4,681,948	14.12%	12.3%
Sibe_01	0.91	1.03	58	66	2,938,532	3,211,977	16.63%	15.2%
Sibe_02	0.87	0.94	51	53	2,854,815	3,546,145	17.70%	13.9%
Sibe_03	1.37	1.6	55	59	10,075,449	10,859,283	4.86%	4.55%
Sibe_04	0.86	0.96	56	65	2,665,384	2,758,844	18.30%	17.7%

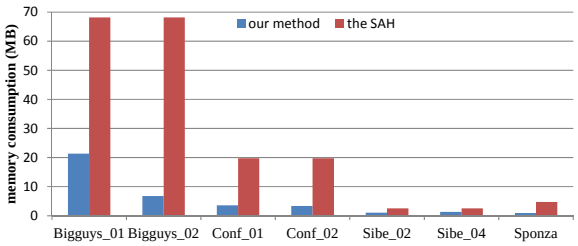


Figure 8. Comparison on memory consumption.

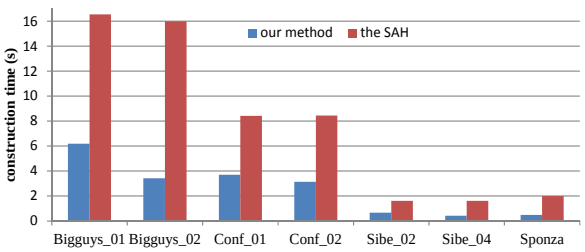
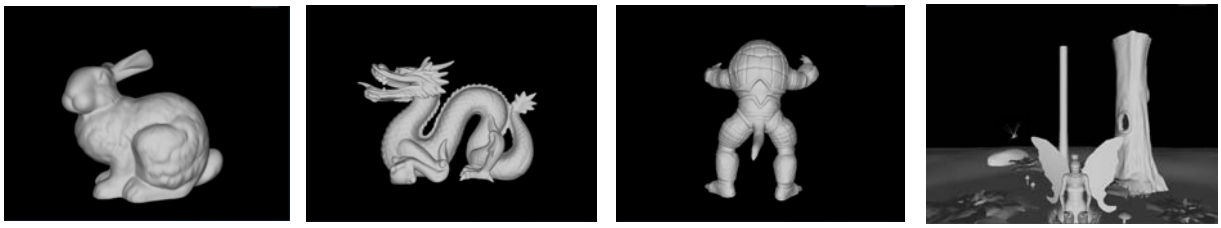
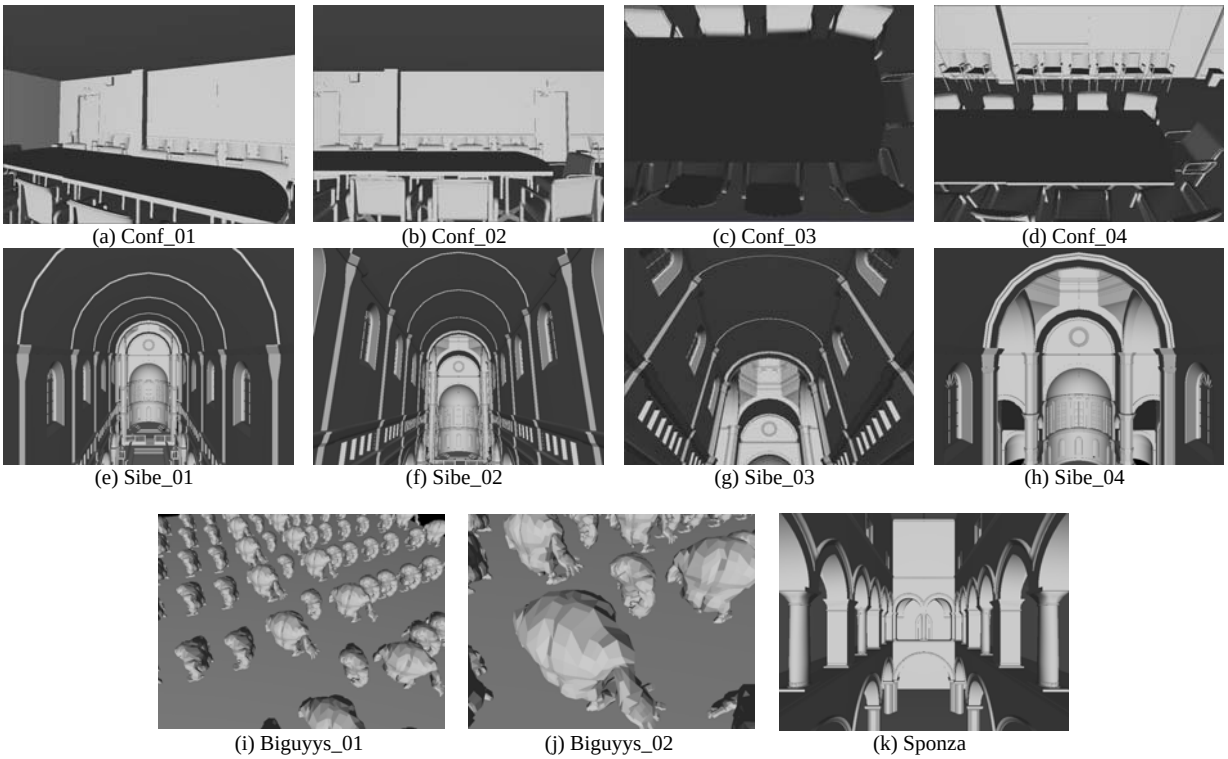


Figure 9. Comparison on construction time.



(a) Bunny: 69,451 (b) Dragon: 871,414 (c) Armadillo: 345,944 (d) Fairy: 173,397

Figure 10. Scenes with uniformly distributed primitives. We also list the total count of primitives below each scene.



(a) Conf_01 (b) Conf_02 (c) Conf_03 (d) Conf_04

(e) Sibe_01 (f) Sibe_02 (g) Sibe_03 (h) Sibe_04

(i) Biguyys_01 (j) Biguyys_02 (k) Sponza

Figure 11. Scenes with uniformly distributed primitives. (a) ~ (d) are four different viewpoints for Conference. (e) ~ (h) are different viewpoints for Sibenik. (i) ~ (j) are Bigguys. The primitive count of Conference, Sibenik, Bigguys and Sponza are 282775, 80054, 368234, and 67461.

REFERENCES

[1] V. Havran, "Heuristic ray shooting algorithms", PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2000.

[2] J. David MacDonald, Kellogg S. Booth, "Heuristics for ray tracing using space subdivision", The Visual Computer, 1990, vol. 6, no. 3, pp. 153–166.

[3] I. Wald, V. Havran, "On building fast kd-trees for ray tracing, and on doing that in o (n log n) ", IEEE Symposium on Interactive Ray Tracing, 2006, pp. 61–69.

[4] Matt Pharr, Greg Humphreys, "Physically Based Rendering: From Theory to Implementation", Morgan Kaufman, 2004.

[5] Warren Hunt, William R. Mark and Gordon Stoll, "Fast kd-tree construction with an adaptive error-bounded heuristic". IEEE Symposium on Interactive Ray Tracing, 2006, pp. 81–88.

[6] Stefan Popov, Johannes Gunther, Hans-Peter Seidel and Philipp Slusallek, "Experiences with streaming construction of SAH kd-trees", IEEE Symposium on Interactive Ray Tracing, 2006.

[7] W. Hunt, "Corrections to the surface area metric with respect to mail-boxing", IEEE Symposium on Interactive Ray Tracing, 2008, pp. 77–80.

[8] B. Fabianowski, C. Fowler and J. Dingliana, "A cost metric for scene-interior ray origins", Eurographics, Short Papers, 2009, pp. 49–52.

[9] J. Bittner, V. Havran, "RDH: Ray Distribution Heuristics for Construction of Spatial Data Structures", 25th Spring Conference on Computer Graphics (SCCG 2009), ACM, Budmerice, Slovakia, 2009, pp. 61–67.

[10] B. Choi, B. Chang, I. Ihm, "Construction of efficient kd-trees for static scenes using voxel-visibility heuristic", Computers & Graphics, vol. 36, no. 1, 2012, pp. 38-48.

[11] Shevtsov M., Soupikov A. and Kapustin A., "Highly Parallel Fast k-D Tree Construction for Interactive Ray Tracing of Dynamic Scenes", Computer Graphics Forum, vol. 26, no. 3, 2007, pp. 395–404.

[12] Zhou K., Hou Q., Wang R., Guo B, "Real-time k-D Tree Construction on Graphics Hardware", SIGGRAPH Asia, vol. 27, 2008, pp. 1–11.

[13] Fan Wen-Shan, Wang Bing, “Fast KD-Tree Construction Method by Probing the Optimal Splitting Plane Heuristically”, Chinese Journal of Computers, 2009, vol. 32, no. 2, pp. 185–192.
[14] Guo Jie, Xu Xiao-yang, Pan Jin-gui, “Build KD-Tree for Virtual Scenes in a Fast and Optimal Way”. Acta electronica Sinica, 2011, vol. 39, no. 8, pp. 1811–1817.
[15] Nicolas Feltman, Minjae Lee, Kayvon Fatahalian. “SRDH: Specializing BVH Construction and Traversal Order Using Representative Shadow Ray Sets”. High Performance Graphics, 2012, pp. 49–55
[16] Y.S. Kang, J.H. Nah, W.C. Park and S.B. Yang, “gkdtree: A group-based parallel update kd-tree for interactive ray tracing”, Journal of Systems Architecture, 2011.
[17] W. Hunt, W.R. Mark, D. Fussell, “Fast and lazy build of acceleration structures from scene hierarchies”, IEEE Symposium on Interactive Ray Tracing, 2007, pp. 47–54.
[18] Peter Djeu, Warren A. Hunt, Rui Wang, Ikrima Elhassan, Gordon Stoll and William R. Mark. “Razor: An architecture for dynamic multiresolution ray tracing”, ACM Transaction Graph, 2011, vol. 30, no. 5, pp. 115.
[19] Piotr Danilewski, Stefan Popov, Philipp Slusallek. “Binned sah kd-tree construction on a gpu”, Technical report, Saarland University, 2010.
[20] Wald I., Mark W.R., Gunther J., Boulos S., Ize T., Hunt W., Parker S.G., Shirley P.. “State of the Art in Ray Tracing Animated Scenes”, Computer Graphics Forum, 2009
[21] Warren Hunt, William R. Mark, Don Fussell. “Fast and Lazy Build of Acceleration Structures from Scene Hierarchies”, IEEE Symposium on Interactive Ray Tracing, 2007, pp. 47–54
[22] Johannes Gunther, Heiko Friedrich, Ingo Wald and etc. “Ray tracing animated scenes using motion decomposition”, Computer Graphics Forum, Proceedings of Eurographics, 2006, 25(3): 517–525.
[23] Marek Vinklera, Vlastimil Havranb and Jiri Sochor. “Visibility Driven BVH Build Up Algorithm for Ray Tracing”, Journal of Computer & Graphics, 2012, 32(4): 283–296
[24] Koji Nakamaru and Yoshio Ohno. “Breadth-First Ray Tracing Utilizing Uniform Spatial Subdivision”, IEEE Transactions on Visualization and Computer Graphics, 1997, pp. 316–328

Xiao Liang was born in 1983. She is a Ph.D. candidate in Sichuan University. Her research interest include realistic rendering, computer graphics and virtual reality.

Hongyu Yang was born in 1967. She is a professor and Ph.D. supervisor of college of computer science of Sichuan University. She received Ph.D. degree from Sichuan University in 2008. She is a managing director of China Society of Image and Graphics. Her research interest include virtual reality and computer graphics.

Yinling Qian was born in 1989. He is a Master candidate in Sichuan University. His research interest include parallel rendering and computer graphics.

Yanci Zhang was born in 1975. He is an associate professor and master supervisor of college of computer science of Sichuan University. He received Ph.D. degree from Institute of Software Chinese Academy of Sciences. He is a professional member of CAD and Graphics of China Computer Federation. His research interest include global illumination, fluid simulation and parallel rendering.