An Efficient Kd-tree Building Algorithm base on VRDH for Ray Tracing

Yue Cao
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, CHINA

Leiting Chen
School of Computer Science and Engineering
University of Electronic Science and Technology of China
Chengdu, CHINA

Xiao Liang
School of Computer Science
Southwest Petroleum University
Chengdu, CHINA

Abstract: The Surface Area Heuristic (SAH) is regarded as a standard heuristic for building acceleration structure for ray tracing. However, its assumption of uniform ray distribution is a gross simplification and might lead to unnecessary intersection tests. In this paper, we consider ray distribution and propose a novel heuristic based on Visual Ray Distribution Heuristic (VRDH) to build a high-quality kd-tree. Because only the rays intersecting with primitives will contribute to final image, we distinguish types of rays according to intersection results and improve cost metric through only estimating traversal cost of visual rays. Since the knowledge of ray distribution is only available during tracing process, temporal coherence is exploited. We construct an auxiliary 3D grid structure to sample visibility. The knowledge in grid in frame k is employed to build kd-tree of frame $k + 1$. Additionally, the boundaries of voxels in grid are regarded as main splitting candidates as well as two strategies are presented to choose more optimal splitting planes. The experimental results indicate that our algorithm can reduce the number of ray-primitive intersection tests by $20\% \sim 66\%$, meanwhile make a speedup of approximately 20% for scene with $300K$ primitives for overall frame performance.

Key–Words: ray distribution, visual ray, 3D grid, ray tracing

1 Introduction

Ray Tracing is a fundamental rendering technique which generates photo-realistic image due to support arbitrary point-to-point visibility queries. However, ray tracing is computationally demanding for considerable ray-primitive intersection tests, thus, has been used in off-line rendering until very recently.

Many acceleration structures have been proposed to cull the number of ray-primitive intersection tests, such as kd-tree [1], Bounding Volume Hierarchy (BVH) [2] [3] and grid [4]. Among those, kd-tree is regarded as a well-known spatial data structure for high performance. Although several heuristics tend to construct a high quality kd-tree, Surface Area Heuristics (SAH) [5] is the most popular method during last decades. The SAH estimates traversal cost for rays and chooses the splitting plane with the minimum cost. The key is to evaluate the probability that a random ray intersects with a node of kd-tree based on geometric probability theory [6]. Since the SAH is based on the assumption that the rays are infinite lines and distributed uniformly in the space, the probability is proportional to the surface area.

If we consider all the rays R in the space, their distribution roughly meet the assumption of SAH. But ray tracing only considers the rays emitted for eye and the reflected, refracted and shadow rays when they hit some objects in the scenes. These rays are only a small subset of R and their distribution is barely uniform. Besides, since some rays might be blocked by opaque objects, the ray distribution will be more irregular. Particularly, for scenes with depth complexity where only a small part of primitives are visible, the SAH estimates traversal cost of the whole scene. This estimation error might incur unnecessary ray-primitive intersection tests, hence, surface area is a gross simplification.

In this paper, we consider the actual ray distribution and derive a more accurate heuristic for ray tracing, called Visual Ray Distribution Heuristic (VRDH), to build a high quality kd-tree. Because only the rays intersecting with primitives will contribute to the final image, called visual ray in our algorithm, the heuristic distinguishes types of rays according to intersection results and estimates traversal cost of visual rays. This cost metric are able to reflect the distribution characteristics of rays in the scene. However, neither ray distribution nor intersection results is only available after building process, temporal coherency is exploited. The knowledge in frame k is computed and recorded in an auxiliary 3D grid structure to guide to build kd-tree in frame $k + 1$. Then, to reduce the amount of splitting candidates, only boundaries of voxels in 3D grid are used as the main splitting probes. Additionally, two strategies are proposed for choosing optimal splitting planes to prevent the degrading of quality of kd-tree.

It is viewed that our algorithm builds a specialized kd-tree dependent on rays distribution. We find the algorithm achieves to reduce the intersection tests by more than $20\% \sim 66\%$ for scenes with ray occlusion. Even the resolution of grid is sparse, it is still effective. For example, a grid with resolution of $20 \times 5 \times 13$ for scene Conference, is sufficient to improve traversal performance only with trivial overhead during traversal process.

The result of the paper is organized as follows. Section 2 discusses the previous work. Section 3 describes the Visual Ray Distribution Heuristic. Experimental results are given in Section 4. Finally, we make a conclusion of the paper in Section 5.

2 Related Work

Kd-tree is a well-known acceleration structure for high performance. For years, a great deal of algorithms have concentrated on the effectiveness of kd-tree. The key to build an optimized kd-tree is to determined the splitting plane. Among many heuristics, the Surface Area Heuristic (SAH) is considered as the most popular model. Given a splitting candidate, the SAH takes into account the average traversal cost and the probability for hitting a volume for each ray, then selects the candidate corresponding to the minimum cost as the optimal plane. The cost function is defined as the follows:

$$C = \frac{SA_L}{SA} \cdot C_L + \frac{SA_R}{SA} \cdot C_R \qquad (1)$$

where $\frac{SA_L}{SA}$ and $\frac{SA_R}{SA}$ is the probability of ray hitting the left and right child, C_L and C_R is the traversal cost for two children which is linear with number of primitives.

The SAH based kd-tree combined with frustum traversal [7] and ray packet tracing techniques [8] provide significant performance improvement in static scenes. Wald et al. [9] propose a $n \log n$ algorithm even with the same asymptotic complexity as median splitting heuristic, but still fail to allow to dynamic scenes due to evaluating cost on each primitive.

To tradeoff between building time and tracing performance, many SAH approximation algorithms have been proposed. Hurley et al. [10] first introduce a binning algorithm to restrict a great deal of splitting candidates to discrete positions. Hunt et al. [11] employ a piecewise linear function to approximate the SAH. Popov et al. [12] linearly approximate the SAH cost with 1024 uniformly distributed samples meanwhile reduce memory bandwidth.

Recently, parallelizing construction has received many attentions on different parallel platform, including multi-thread, multi-core on CPU and GPU versions. Ben et al. [13] design a 2-thread parallelizing method for creation and initial sorting of candidate lists. Shevtsov et al. [14] introduce a 4-core parallel kd-tree building algorithm with an improvement of 3.5 times for both building and rendering time. But its traversal performance does not scale well with parallelism and the algorithm will generate nearly 30% degrading due to employ a count median splitting in the top tree. The first realtime kd-tree is presented by Zhou et al. [15], which make best use of streaming architecture on GPU during construction. It is stated that its 128-core version achieves a speedup of $4 \sim 7$ times with well-optimized single-core CPU algorithms and is competitive with multi-core CPU algorithms. They use a spatial median heuristic in the upper levels of tree and the SAH in the bottom levels of trees respectively, which leads to approximately 10% degrading in tree quality as scene scales to more that $100K$ primitives. Choi et al. [16] provide a precise SAH based kd-tree construction on multi-core CPU.

Sorting subdivision data structure is time-consuming and could be a burden especially for complex scenes. Mora et al. [17] present an efficient ray tracing method nearly without conventional acceleration structure meanwhile employ limited memory. The algorithm achieves high performance in dynamic scenes, but at the expense of bundling much rays per rendering pass and less scalability on multi-core CPU or GPU platform.

In recent years, many publications have attempted to improve accuracy of cost model for the SAH. Havran et al. [18] first analyze the unrealistic assumption, and propose a general cost model aiming to approach ray distribution. But the model is regarded to be complicated for estimating the cost for hitting and miss rays. Bittner et al. [19] extend the SAH by employing a represent ray set to re-estimated the probabilities of rays intersecting with nodes, but achieve a minor speedup. Fabianowski et al. [20] develop a more accurate metric for rays originating inside the scene with a average speedup of 3.5% in performance. Vinkler et al. [21] construct a more efficient BVH by modifying cost function. They achieve a 102% improvement on traversal for path tracing for high occluded scenes. Choi et al. [22] introduce voxel visibility to enable more sophisticated cost model for static scenes. All these heuristics are mainly implemented in static scenes.

3 Visual Ray Distribution Heuristic

In this section, we relief the assumption of the SAH and aim to build a high quality kd-tree with the Visual Ray Distribution Heuristic.

3.1 Cost Metric based on VRDH

For a given voxel V, the rays hitting the voxel can be classified as two types. The first type is the rays that penetrate V but without any intersection with primitives. The other type is the rays that intersect with primitives and stop to be traced. We define the second type of rays that are hitting with the voxel as well as occluded by primitives as visual rays. Because only the intersection results of visual rays are contributed to the final image, previous algorithms that can not allow to distinguish these two types rays might overestimate the traversal cost. In this paper, we estimate the traversal cost of visual rays. For a ray known to hit a voxel, the probability of hitting its left or right child is defined as follows:

$$p_{L/R} = \frac{numOfVisRay(V_{L/R})}{numOfRay(V)} \qquad (2)$$

where $numOfVisRay$ and $numOfRay$ are the amount of rays occluded by primitives in the voxel and rays hitting the voxel respectively. V_L and V_R are children node.

For a standard ray tracing method, the knowledge of ray distribution and intersection results are known during tracing. To solve the problem, temporal coherence is exploited. For most animation scenes, the image results between successive frames are similar. Especially for high occluded scenes, the invisible primitives might be still invisible in next few frames. Therefore, we take advantage of the knowledge of ray distribution and intersection result of frame k to build the acceleration structure of frame $k+1$.

An auxiliary structure, 3D regular grid named VRD-$Grid$ (Visual Ray Distribution Grid), is established to save the knowledge of ray distribution, as Fig. 1 (a) shows. Since the distribution of visual rays varies among the whole space, for each regular voxel in VRD-$Grid$, we employ an value to represent the feature of visibility, called $visibility\ value$. Potentially, $visibility\ value$ will be high if most of the primitives in the voxel are visible. Usually, $visibility\ value$ for a voxel with less visible primitives might be lower that the voxel with more visible primitives. However, for most scenes, the geometry distribution is not regular. As Fig. 2 describes, it is possible that the voxel in (a) contains large size primitives while the one in (b) contains some small size primitives. Both of them have the same number of visible primitives, that is t_0 and t_1 in (a), t_0 and t_4 in (b). But more visible rays hit the left voxel. It is viewed that estimation of $visibility\ value$ only by number of visible primitives might be coarse so that can bring some deviation. In our algorithm, we estimate the value by the number of visible rays. For simplification, if a ray is occluded by a primitives, the value will be accumulated. Then, for the voxel in (a) and (b), the values are 5 and 2 respectively.

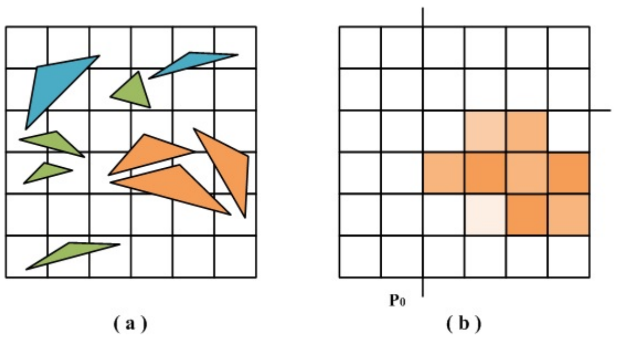


Figure 1: Using VRD-$Grid$ to indicate the ray distribution. (a) The scene with its VRD-$Grid$. Only primitives with orange color are visible. (b) $visibility\ value$ in each voxel. Deeper color in voxel indicates that more visible rays are occluded.

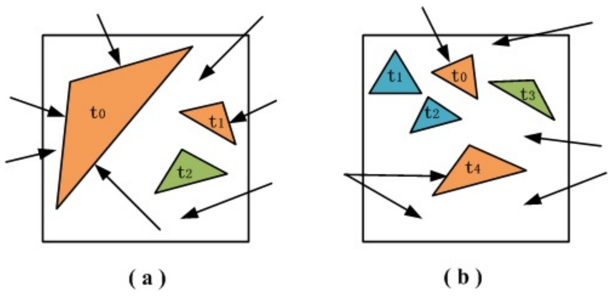


Figure 2: Evaluating the feature of visibility. (a) Primitive t_0 and t_1 are visible. (b) Primitive t_0 and t_4 are visible.

With the conception of $visibility\ value$, we represent the values of voxels in Fig. 1 (b). It is noted that only primitives with orange color are visible. The deeper ones are more visible than the lighter ones. For voxel containing many invisible primitives or empty voxel, the $visibility\ value$ is low or zero.

During ray tracing of frame k, the $visibility\ value$ of each voxel can be achieved through the results of ray-primitive intersection tests. Once a ray is found to be occluded, according to the intersection point, not the primitive, the $visibility\ value$ of associated voxel is increased.

In our algorithm, splitting axis is the one with longest extent of primitives overlap. Different from the SAH, the main splitting probes are placed along the boundaries of vexels to reduce amount of splitting candidates, such as P_0 and P_1 in Fig. 1 (a). This strategy as well as $visibility\ value$ can allow to separate visible primitives from invisible ones, which is explained in Section 3.3.

Based on the measure described above, the average cost function driven by visual rays is defined as follows:

$$\begin{aligned} &C = p_L \cdot C_L + p_R \cdot C_R \\ &p_{L/R} = \frac{\sum_{voxel \in grid'} numOfVisRay(voxel)}{\sum_{voxel \in grid} numOfRay(voxel)} \\ &C_{L/R} = c_i \cdot N_{L/R} \end{aligned} \tag{3}$$

where $grid$ is a portion of $VRD\text{-}Grid$ that current node for splitting corresponds to, $grid^{'}$ is the voxels set belonging to $grid$ that are hit by visible rays, $voxel$ represents the voxel in $grid$ or $grid^{'}$, c_i is intersection cost with primitives. N_L and N_R are the number of primitives in left and right child respectively. $numOfVisRay$ and $numOfRay$ are defined the same as previously.

3.2 Determining the Resolution of VRD-Grid

The resolution of $VRD\text{-}Grid$ should be carefully considered as only the boundaries of voxel are used as the main splitting candidates. Additionally, for dynamic scenes, the resolution would impact the availability of temporal coherence. Too large size of voxel might not be capable of separating visible primitives from invisible ones. On the other hand, too small voxels might incur a poor building performance due to an increased number of splitting candidates.

In our algorithm, we use a similar method as [23] to determine the resolution of grid. Given the intention of inhabiting the primitives more evenly into voxels, the resolution in x, y, z dimensions are regarded to be proportional with the number of primitives as follows:

$$Resolution_i = \frac{B_i}{max\,\{B_x, B_y, B_z\}} \times e \times \sqrt[3]{N} \tag{4}$$

where i indicates the dimension, B_i denotes the length of bounding box in dimension i, N is the number of primitives in scene. In practice, the coefficient of e is chosen experientially (some value between 0.4 and 0.6 is used in our implementation).

3.3 Choosing Optimal Splitting Probes

As described above, we use splitting planes at boundaries of voxels to reduce number of splitting samples. But using the kind of splitting planes over the whole scene might degrade the quality of tree. In this paper, we use two additional strategies to solve the problem. That is an early excluding process for the top hierarchy of tree, as well as an adaptive splitting probing strategy for the lower hierarchy.

The main difference between the SAH and VRD-H is that the SAH only considers the actual geometry distribution, while our algorithm considers geometry distribution as well as ray distribution based on $VRD\text{-}Grid$. As Fig. 1 shows, the left and the right one are the view of the SAH and VRDH respectively.

For high occluded scene, $VRD\text{-}Grid$ is featured that only a few voxels are provided with high visibility as shown in Fig. 1 (b). It is advantageous to separate space with low visibility from other as early as possible during construction. Also, this property of ray distribution is prone to lead a sharply discontinuities cost function through our limited splitting probes. To prevent the problem, our algorithm chooses the splitting planes which allow to keep the probable discontinuities into an isolated volume.

After selecting the splitting axis, an early excluding process is implemented. This process includes two quick sweep over the splitting probes. First, we build an array of bins to keep the $visibility\ value$ for each splitting probes. For each bin, we accumulate the value of corresponding voxels. Then, we sweep the $bins$ again and pick the plane P_i satisfying the define as follows:

$$max\left|\sum_{k\leq i} bin(k) - \sum_{j>i} bin(j)\right| \tag{5}$$

where i, j and k are index of splitting probes. Through the strategy, P_0 and P_1 in Fig. 1 (b) are regarded as optimal planes. This strategy is implemented at the top of hierarchy where the visibility varies severely among space.

Towards the middle or bottom levels of hierarchy, where visual ray distribution is relatively smooth, we start a two-step splitting sample strategy. Given a node, first, we compute traversal cost at boundaries of voxels of the node. For each splitting probe P_i, the cost is denoted as $C(i)$. Usually, the loss of fidelity appears the segments where $C(i+1) - C(i)$ is minimum. Then, towards these segments, we dispose more splitting probes to approximate cost function.

3.4 Building a Specialized Kd-tree

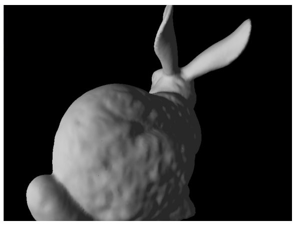
Finally, we have developed a greedy, top-down kd-tree construction based on VRDH. Rather than the SAH, it is a specialized kd-tree dependent on ray distribution. The cost metric and strategies describes above are primarily used on top and middle levels. Towards the bottom of the hierarchy, we still resort to the SAH for the reason that most primitives are visible in the volume.

Additionally, for scenes with depth complexity, only a small portion of which from a single viewpoint are visible, on-demand building strategy can be also combined to our algorithm. This strategy aims to only build visible portions of acceleration structure in each frame. It interleaves the building and traversal process so that only the volumes of hierarchy that are penetrated by rays are built.

4 Experiments and Results

We have implemented the VRDH on computer equipped with 4 cores Intel Xeon E3-1230 CPU and 8 GB memory. To evaluate the quality of kd-tree by our approach, two other well-known kd-tree methods are also implemented. One is the high-quality SAH kd-tree building algorithm [9], which takes splitting sample for each primitives, and is used for static scenes mainly. The other one is min-max binning algorithm [14] for dynamic scenes. We use the same building parameters for the three algorithms. The intersection cost for ray-primitives algorithm is 80, the traversal cost is 1, the maximum tree depth is $8 + 1.3 \log n$, and the primitives in leaf node is less than 16. All the traversal are parallelized by OpenMP if no on-demand building strategy is implemented.

To analyze the quality of the kd-tree, we use similar way introduced by [21] to design our test. Scenes with different geometry complexity as well as camera viewpoints are chosen, as shown in Fig. 3. For each architectural model, we test three camera viewpoints. They are one camera viewpoint outside the scene, two camera viewpoints inside the scenes with different geometry complex. We also test scenes with regular geometry complexity, as shown in Fig. 4. The size of scenes are listed in Tab. 1


(a) Bunny

(b) Fairy

Figure 4: Test scenes with regular geometry distribution.

Scene	Number of Primitives
Bunny	69, 451
Fairy	174, 117
Sibenik	80, 054
Cloister	81, 354
Sponza	67, 461
Conf.	282, 755

Table 1: Number of primitives of test scenes.

Performance comparison for three algorithms are described in Tab. 2, including the number of ray-primitive intersection tests, that is the computation bottleneck for ray tracing, traversal performance in fps as well as overall image time. Although our algorithm builds an unbalanced tree which are prone to produce more deeper tree depth than the SAH, the overhead of traversal for more deeper tree is compensated by the significant reduction of ray-primitive intersection tests by nearly $20\% \sim 66\%$. Particularly for Conference, our algorithm achieves to reduce the intersection tests by more than 50% in all camera viewpoints. Besides, the experimental results indicate that our algorithm boosts the performance of traversal by $30\% \sim 55\%$. This improvement is mainly owing to the fact that the visible primitives are prior to be traversed than invisible ones. It is observed that our algorithm is less effective for moderate scenes with uniform distributed primitives, such as Bunny and Fairy. This is because the overhead of intersection tests only occupies a minor portion of the overall image time.

We also compare overall image time that includes building time and traversal time. The improvement is about $15\% \sim 23\%$ for most scenes which is not as excellent as the result of traversal performance. This is because our current building algorithm is single-threaded while the traversal is optimized by OpenMP. If building process is paralleled, the improvement of overall image time is much better.

Many recent publications also consider ray distribution to achieve a high performance acceleration structure. We compare the reduction of ray-primitive

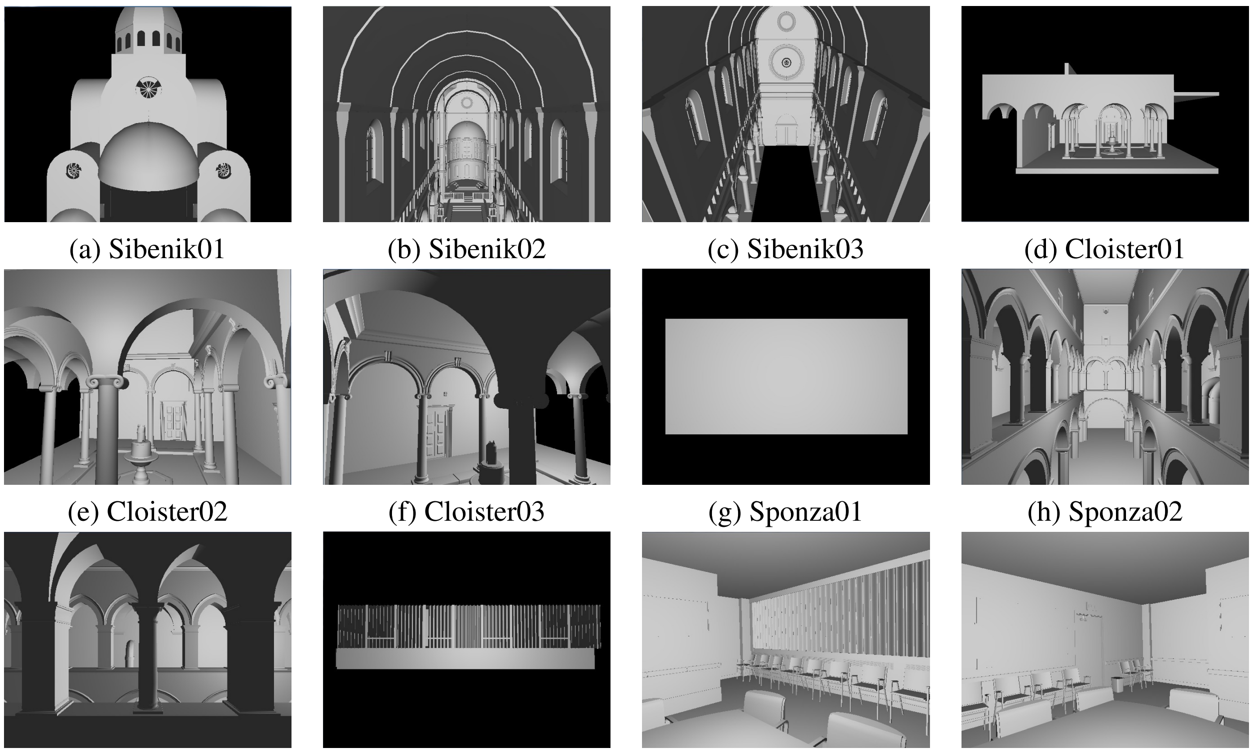

(a) Sibenik01 (b) Sibenik02 (c) Sibenik03 (d) Cloister01
(e) Cloister02 (f) Cloister03 (g) Sponza01 (h) Sponza02
(i) Sponza03 (j) Conf.01 (k) Conf.02 (l) Conf.03

Figure 3: Test scenes with different geometry complexity as well as different viewpoints for primary rays.

intersection tests of our algorithm with some recent methods. They are OSAH [21] and Fast Approximation method [20], both of which take each primitive as splitting candidate. The relative reduction ratio of ray-primitives intersection tests for some complex scenes are illustrated in Tab. 3. The fourth and fifth column show the data of VRDH with only primary rays and extra shadow rays. The scenes rendered by the VRDH with primary and shadow rays are shown in Fig. 5. It can be seen that our algorithm tends to generate higher quality kd-tree than these methods for highly complex scenes.

Scene	Reduction of intersection tests			
	OSAH	Fast Approx.	VRDH-1	VRDH-2
Sibenik	-17%	-4%	-14%	-3%
Cloister	n.a.	+1.4%	-28%	-13.15%
Sponza	-20%	-13.7%	-31%	-5.3%
Conf.	n.a.	-6.5%	-68%	-53%

Table 3: Comparison with recent publications in number of ray-primitive intersection tests.

We also compare the memory consumption and building time with OSAH and Fast Approximiation in Tab. 4. Although no exact memory consumption data are presented for Fast Approximation algorithm, they declare the memory consumption are more than the SAH. Therefore, the memory consumption of our algorithm is less than other algorithms. As to building time, our method outperforms both of them an order of magnitude.

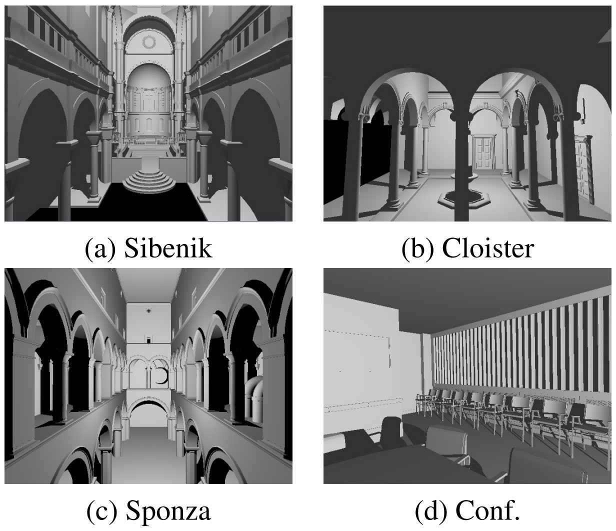

(a) Sibenik (b) Cloister
(c) Sponza (d) Conf.

Figure 5: The test scenes with primary rays and shadow rays.

If we combine on-demand building strategy into

Scene	Int. Num. (M)			Render. Perf.(fps)		Image Time (ms)	
	Fast SAH	VRDH		Fast SAH	VRDH	Fast SAH	VRDH
Bunny	7	5.2	-24.6%	4.9	6.2	275	220
Fairy	15.4	12.2	-21%	3.7	3.3	492	480
Sibenik01	5.8	4.5	-22%	7.2	10	277	246
Sibenik02	15.1	10.2	-32%	3.6	5.1	349	297
Sibenik03	16.7	12	-28%	3.7	5.1	360	307
Cloister01	7	4.7	-33%	6.2	8.1	248	202
Cloister02	19	14.3	-24%	3.6	4.8	368	306
Cloister03	18.8	12.9	-32.8%	3.7	5.3	339	285
Sponza01	1.5	1.1	-27%	7.4	10	206	180
Sponza02	18.6	10.5	-44%	3.6	5.1	349	270
Sponza03	24.4	12	-49.8%	3.7	5.2	339	278
Conf.01	13.2	4.5	-66%	4.6	6.8	573	407
Conf.02	42	18	-57%	2.4	3.8	766	597
Conf.03	41	17.3	-58%	2.5	3.9	750	557

Table 2: Comparison of important features of kd-tree, using same building parameters and termination condition with a resolution of 640×480.

Scene	Memory Consumption		
	OSAH	Fast Approx.	VRDH
Sibenik	+18%	n.a.	-2.7%
Cloister	n.a.	n.a.	-30%
Sponza	+22%	n.a.	-28%
Conf.	n.a.	n.a	-11.2%

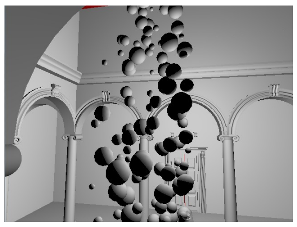
Scene	Building Time		
	OSAH	Fast Approx.	VRDH
Sponza	+424%	+5.8%	-85.44%
Cloister	n.a.	n.a.	-86.57%
Sibenik	+330%	+2.3%	-84%
Conf.	n.a.	+7.6%	-87%

Table 4: Comparison with recent publications in memory consumption and building time.

building algorithm, more building time and memory consumption are saved. As Tab. 5 shows, the VRDH consumes comparable memory as Fast SAH and less memory than the SAH. As to on-demand building, for the four test scenes, we save 64.5%, 27.49%, 44.2% and 25% in memory consumption than normal building method respectively.

Most recent publications aiming to a high-quality kd-tree with consideration of ray distribution [19][20][21][22] are limited to static scenes. However, our algorithm can be implemented to dynamic scenes. To test the availability of the regular $VRD\text{-}Grid$, we test our algorithm under the situation of dynamic viewpoints for different scenes. The rendering time and overall image time of 40 continuous frames are listed in Fig. 6 (a) $\sim$ (d). Fast SAH used in dynamic scene is employed as test baseline. As the result shows, our algorithm outperforms Fast SAH in both rendering time and image time for all test scenes.

We test two dynamic scenes with our algorithm. As Fig. 7 (a) and (b) show, one is a scene with nearly deformation animation, where a great deal of marbles move in Cloister. The other is a scene with rigid body movement, where a wooden doll is running around a table in Conference. Because our heuristic would rebuild from scratch according to ray distribution, both the dynamic scenes are provided with moving viewpoints to improve the dynamic characteristic further. As Fig. 6 (e) and (f) describe, for marble animation, we make an improvement in rendering time of 23% in average as well as speed up the overall performance by 17%. For wooden doll animation, we make an improvement in rendering time of 25% as well as improve the overall performance by 16%.


(a) Marble animation

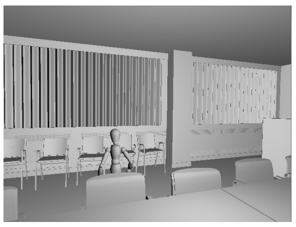
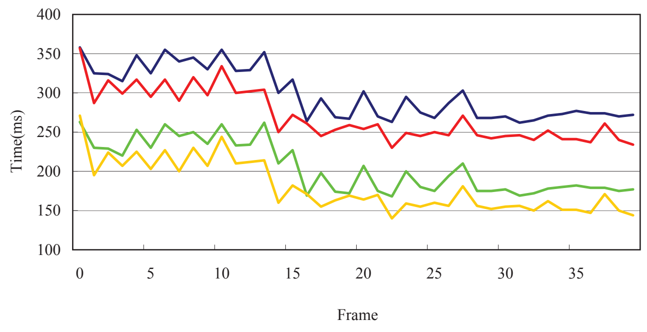
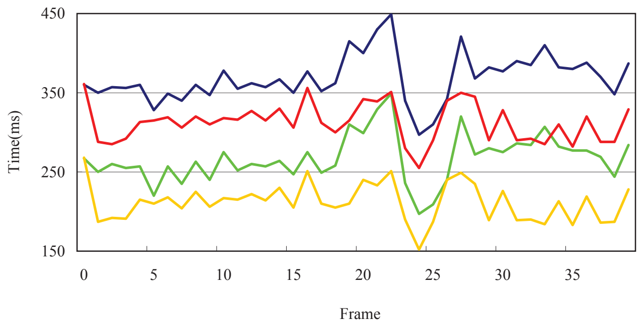

(b) Wooden doll animation

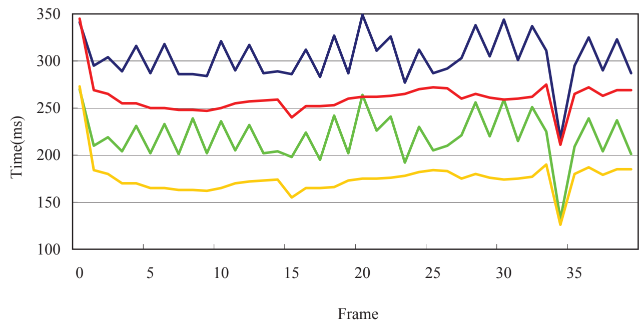
Figure 7: Animation scenes.



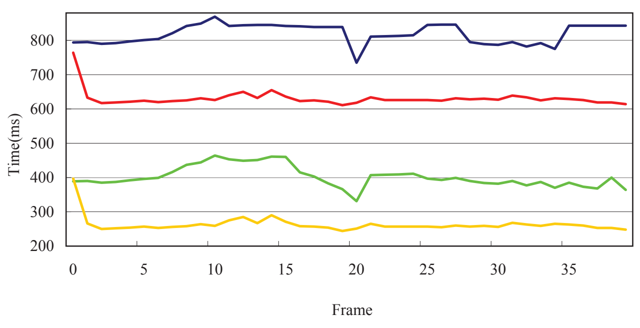
(a) Sibenik



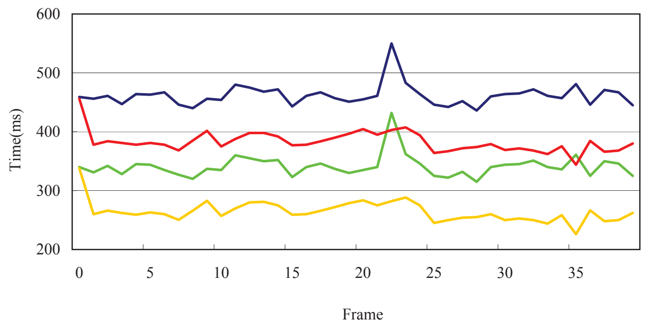
(b) Cloister



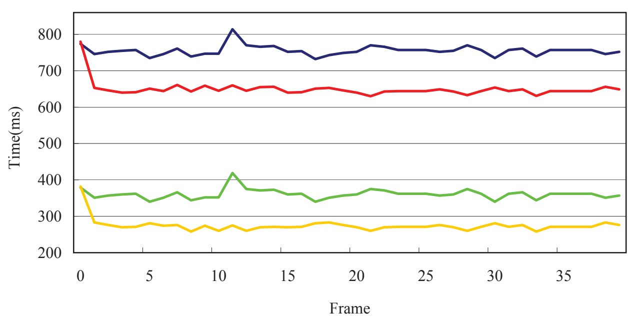
(c) Sponza



(d) Conference



(e) Marbles move in Cloister



(f) Wooden doll runs around Conference

Figure 6: Rendering time and image time for all animation scenes. It is noted all the scenes are provided with moving viewpoints and are rendered with a resolution of 640 × 480. The green and orange line sketch the date of rendering time of Fast SAH and VRDH. The blue and red line indicate the date of image time of Fast SAH and VRDH respectively. (e) and (f) are dynamic scenes with rigid-body movement and nearly deformation respectively.

Scene	Number of nodes of kd-tree			
	the SAH	Fast SAH	VRDH	VRDH with on-demand building
Sibenik01	88, 825	83, 648	82, 296	29, 197
Cloister02	155, 571	89, 259	91, 468	66, 325
Sponza03	109, 631	79, 579	81, 409	45, 406
Conf.02	1, 184, 561	321, 823	217, 725	162, 336

Table 5: Comparison with normal building and on-demand building algorithm in memory consumption.

5 Conclusion

The SAH is regarded as a popular heuristic for kd-tree, but might lead error estimation of traversal cost due to its assumption to reduce the quality of acceleration structure.

In this paper, we introduce a novel heuristic for ray tracing to build a high quality kd-tree. We exploit the obversion that the system has the knowledge of ray distribution to build a new cost metric. Then, an auxiliary grid is employed to record the knowledge of visible ray distribution in frame k and to guide to build kd-tree in frame $k+1$. Two strategies are proposed to choose optimal splitting plane. The experimental results demonstrate that the SAH based methods do not always provide high quality kd-tree especially for large and complex scenes. On the other hand, our algorithm can achieve an average of 30% traversal improvement and 20% overall improvement for complex scenes over SAH-based methods. Even compared with some time-consuming high quality kd-tree building algorithms, our algorithm still has the performance advantages.

In the future, we aim to test our algorithm with more dynamic scenes with large-scale deformation, and explore temporal coherence in ray tracing to improve overall performance further.

Acknowledgements: The authors wish to thank the anonymous reviewers. The research is supported by the National High-Tech Research and Development Program of China (863 Program) with grant No. 2012AA011503, and Youth Foundation of Southwest Petroleum University with grant No. 285.

References:

[1] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Communications of the ACM*, Vol.18, No.9, 1975, pp. 509-517.

[2] S. Rubin, T. Whitted, A 3-dimensional representation for fast rendering of complex scenes, *In Proceedings of SIGGRAPH*, 1980, pp. 110-116.

[3] KAJIYA, J, A, The rendering equation, *In Proceedings of SIGGRAPH*, 1986, pp. 143-150.

[4] Akira Fujimoto, Takayuki Tanaka, and Kansei Iwata, Arts: Accelerated ray-tracing system, *IEEE Comput. Graph. Appl.*, Vol.6, No.4, 1986, pp. 16-26.

[5] Goldsmith, Jeffrey and Salmon, John, Automatic Creation of Object Hierarchies for Ray Tracing, *IEEE Comput. Graph. Appl.*, Vol.7, No.5, 1987, pp. 14-20.

[6] H. Solomon, Geometric Probability, *J.W. Arrowsmith Ltd*, 1978.

[7] Alexander Reshetov, Alexei Soupikov, Jim Hurley, Multi-level ray tracing algorithm, *ACM Trans. on Graphics*, 2005, pp. 1176-1185.

[8] Ingo Wald, Vlastimil Havran, Interactive Rendering with Coherent Ray Tracing, *Computer Graphics Forum*, 2001, pp. 153-164.

[9] Ingo Wald, Philipp Slusallek and Carsten Benthin and Markus Wagner, On building fast kd-trees for ray tracing, and on doing that in $O(N \log N)$, *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 2006, pp. 61-69.

[10] Jim Hurley, Er Kapustin, Er Reshetov, Alexei Soupikov, Fast Ray Tracing for Modern General Purpose CPU, *In Proceedings of Graphicon*, Vol.7, No.5, 2002, pp. 2002.

[11] W. Hunt, W.R. Mark, G. Stoll, Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic, *Symposium on Interactive Ray Tracing*, Los Alamitos, CA, USA, 2006, pp. 81-88.

[12] Stefan Popov, Johannes Gnther, Hans-Peter Seidel, Philipp Slusallek, Experiences with Streaming Construction of SAH KD-Trees, *PROCEEDINGS OF THE 17TH EUROGRAPHICS SYMPOSIUM ON RENDERING*, 2006, pp. 139-149.

[13] Benthin, C., Realtime Ray Tracing on Current CPU Architectures, *PhD thesis*, Saarland University, 2006.

[14] Maxim Shevtsov, Alexei Soupikov, Alexander Kapustin, Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes, *Computer Graphics Forum*, Vol. 26, No. 3, 2007, pp. 395-404.

[15] Kun Zhou, Qiming Hou, Rui Wang, Baining Guo, Real-time KD-tree construction on graphics hardware, *ACM SIGGRAPH Asia*, 2008, p-p. 126:1–126:11.

[16] Byn Choi, Rakesh Komuravelli, Victor Lu, Hyojin Sung, Robert L. Bocchino, Sarita V. Adve, John C. Hart, Parallel SAH k-D tree construction, *High performance Graphics*, 2010, pp. 77-86.

[17] Mora, Benjamin, Naive ray-tracing: A divide-and-conquer approach, *ACM Trans. Graph.*, Vol.30, No.5, 2011, pp. 77-86.

[18] Vlastimil Havran, Heuristic Ray Shooting Algorithms, *Ph.D. Thesis*, 2000.

[19] Bittner, Jiří and Havran, Vlastimil, RDH: ray distribution heuristics for construction of spatial data structures, *Proceedings of the 25th Spring Conference on Computer Graphics*, 2009, p-p. 51-58.

[20] Bartosz Fabianowski, Colin Fowler, John D-ingliana, A Cost Metric for Scene-Interior Ray Origins, *Eurographics Short Papers*, 2009, p-p. 49-52.

[21] Marek Vinkler, Vlastimil Havran, Jiri Sochor, Visibility driven BVH build up algorithm for ray tracing, *Comput. Graph.*, Vol.36, No.4, 2012, p-p. 283-296.

[22] Byeongjun Choi, Byungjoon Chang, Insung Ih-m, Construction of efficient kd-trees for static scenes using voxel-visibility heuristic, *Comput. Graph.*, Vol.36, No.1, 2012, pp. 38-48.

[23] Matt Pharr, Greg Humphreys, Physically Based Rendering, Second Edition: From Theory To Implementation, *Morgan Kaufmann Publishers Inc.*, San Francisco, CA, USA, 2010.

[24] Ingo Wald, William R. Mark, State of the Art in Ray Tracing Animated Scenes, *Computer Graphics Forum*, Vol.28, No.6, 2009, pp. 1691-1722.