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 Heuristic (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 in-
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% ~ 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:

$$C = \frac{SA_L}{SA} \cdot C_L + \frac{SA_R}{SA} \cdot CR$$

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 the 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 ~ 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 than 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 an 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 relax 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:

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

(2)

where $\text{numOfVisRay}$ and $\text{numOfRay}$ 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 V RD-Grid to indicate the ray distribution. (a) The scene with its V RD-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.](image1)

![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.](image2)
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 voxels 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:

\[
C = p_L \cdot C_L + p_R \cdot C_R = \frac{\sum_{\text{voxel} \in \text{grid}} \text{numOfRay} (\text{voxel})}{\sum_{\text{voxel} \in \text{grid}} \text{numOfRay} (\text{voxel})} \quad (3)
\]

\[
P_{L/R} = \frac{\sum_{\text{voxel} \in \text{grid}} \text{numOfRay} (\text{voxel})}{\sum_{\text{voxel} \in \text{grid}} \text{numOfRay} (\text{voxel})}
\]

\[
C_{L/R} = c_i \cdot N_{L/R}
\]

where grid is a portion of VRD-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-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:

\[
\text{Resolution}_i = \frac{B_i}{\max \{B_x, B_y, B_z\}} \times e \times \sqrt{N} \quad (4)
\]

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 VRDH is that the SAH only considers the actual geometry distribution, while our algorithm considers geometry distribution as well as ray distribution based on VRD-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-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:

\[
\max \left| \sum_{k \leq i} \text{bin} (k) - \sum_{j > i} \text{bin} (j) \right| \quad (5)
\]

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
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 described 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.

Table 1: Number of primitives of test scenes.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Number of Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bunny</td>
<td>69,451</td>
</tr>
<tr>
<td>Fairy</td>
<td>174,117</td>
</tr>
<tr>
<td>Sibenik</td>
<td>80,054</td>
</tr>
<tr>
<td>Cloister</td>
<td>81,354</td>
</tr>
<tr>
<td>Sponza</td>
<td>67,461</td>
</tr>
<tr>
<td>Conf.</td>
<td>282,755</td>
</tr>
</tbody>
</table>

Figure 4: Test scenes with regular geometry distribution.

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% ~ 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% ~ 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% ~ 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
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.

<table>
<thead>
<tr>
<th>Scene</th>
<th>OSAH</th>
<th>Fast Approx.</th>
<th>VRDH-1</th>
<th>VRDH-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sibenik</td>
<td>-17%</td>
<td>-4%</td>
<td>-14%</td>
<td>-3%</td>
</tr>
<tr>
<td>Cloister</td>
<td>n.a.</td>
<td>+1.4%</td>
<td>-28%</td>
<td>-13.15%</td>
</tr>
<tr>
<td>Sponza</td>
<td>-20%</td>
<td>-13.7%</td>
<td>-31%</td>
<td>-5.3%</td>
</tr>
<tr>
<td>Conf.</td>
<td>n.a.</td>
<td>-6.5%</td>
<td>-68%</td>
<td>-53%</td>
</tr>
</tbody>
</table>

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 Approximation 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.

If we combine on-demand building strategy into...
<table>
<thead>
<tr>
<th>Scene</th>
<th>Int. Num. ((M))</th>
<th>Render. Perf. ((fps))</th>
<th>Image Time ((ms))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fast SAH</td>
<td>VRDH</td>
<td>Fast SAH</td>
</tr>
<tr>
<td>Bunny</td>
<td>7</td>
<td>5.2</td>
<td>-24.6%</td>
</tr>
<tr>
<td>Fairy</td>
<td>15.4</td>
<td>12.2</td>
<td>-21%</td>
</tr>
<tr>
<td>Sibenik01</td>
<td>5.8</td>
<td>4.5</td>
<td>-22%</td>
</tr>
<tr>
<td>Sibenik02</td>
<td>15.1</td>
<td>10.2</td>
<td>-32%</td>
</tr>
<tr>
<td>Sibenik03</td>
<td>16.7</td>
<td>12</td>
<td>-28%</td>
</tr>
<tr>
<td>Cloister01</td>
<td>7</td>
<td>4.7</td>
<td>-33%</td>
</tr>
<tr>
<td>Cloister02</td>
<td>19</td>
<td>14.3</td>
<td>-24%</td>
</tr>
<tr>
<td>Cloister03</td>
<td>18.8</td>
<td>12.9</td>
<td>-32.8%</td>
</tr>
<tr>
<td>Sponza01</td>
<td>1.5</td>
<td>1.1</td>
<td>-27%</td>
</tr>
<tr>
<td>Sponza02</td>
<td>18.6</td>
<td>10.5</td>
<td>-44%</td>
</tr>
<tr>
<td>Sponza03</td>
<td>24.4</td>
<td>12</td>
<td>-49.8%</td>
</tr>
<tr>
<td>Conf.01</td>
<td>13.2</td>
<td>4.5</td>
<td>-66%</td>
</tr>
<tr>
<td>Conf.02</td>
<td>42</td>
<td>18</td>
<td>-57%</td>
</tr>
<tr>
<td>Conf.03</td>
<td>41</td>
<td>17.3</td>
<td>-58%</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Scene</th>
<th>Memory Consumption</th>
<th>Building Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>OSAH</td>
<td>Fast Approx.</td>
</tr>
<tr>
<td>Sibenik</td>
<td>+18%</td>
<td>n.a.</td>
</tr>
<tr>
<td>Cloister</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Sponza</td>
<td>+22%</td>
<td>n.a.</td>
</tr>
<tr>
<td>Conf.</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Sponza</td>
<td>+424%</td>
<td>+5.8%</td>
</tr>
<tr>
<td>Cloister</td>
<td>n.a.</td>
<td>n.a.</td>
</tr>
<tr>
<td>Sibenik</td>
<td>+330%</td>
<td>+2.3%</td>
</tr>
<tr>
<td>Conf.</td>
<td>n.a.</td>
<td>+7.6%</td>
</tr>
</tbody>
</table>

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

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% as well as improve 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%.

![Figure 7: Animation scenes.](image-url)
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.
Table 5: Comparison with normal building and on-demand building algorithm in memory consumption.

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

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 observation 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:


