ON IMPROVING KD-TREES FOR RAY SHOOTING

Vlastimil Havran[1]

Jiří Bittner[2]

[1]Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
havran@mpi-sb.mpg.de

[2]Center for Applied Cybernetics
Czech Technical University in Prague
Karlovo nám. 13
12135 Praha 2
Czech Republic
bittner@fel.cvut.cz

ABSTRACT

Efficient ray shooting algorithm is inherently required by many computer graphics algorithms, particularly in image synthesis. Practical ray shooting algorithms aiming at the average-case complexity use some underlying spatial data structure such as kd-tree. We show the new termination criteria algorithm that improves the space and time complexity of the kd-tree construction. It provides efficient ray-shooting queries and does not require any specific constants from a user. Further, we show how to apply a novel clipping algorithm into the kd-tree within construction phase in order to improve its properties.

Keywords: ray shooting, ray casting, spatial data structures, clipping, kd-tree.

1 INTRODUCTION

Many global illumination algorithms use a discrete sampling of visibility, carried out by a *ray shooting algorithm*. Huge number of ray shooting queries ($\approx 10^6 - 10^9$) are performed in order to compute one antialiased image of a commonly used resolution such as 640×480.

The ray shooting is a difficult problem in spite of its simple definition: given a ray find its nearest intersection with objects in the scene, if such an object exists. The trivial solution that tests every object against the object resulting in $\Theta(N)$ complexity is practically unusable when the number of objects N exceeds $10 - 100$. The worst-case complexity algorithms developed within computational geometry require $\Omega(N^4)$ space in the worst case [Szirm98] to achieve $O(logN)$ query time that is unusable for high number of objects. For this reason the average-case solutions [Arvo89] based on some spatial subdivision or hierarchy of objects are used in practice.

In this paper we deal with two methods that improve the construction of a kd-tree which serves as the base for ray shooting algorithms. The first method concerns the termination criteria of the kd-tree construction. We introduce a new *automatic termination criteria* by involving the cost of the kd-tree in commonly used *fixed termination criteria*. The second method called *split clipping* deals with shrinking bounding boxes of objects straddling the splitting planes during the construction of the kd-tree.

The paper is organized as follows: Section 2 recalls the basic properties regarding the construction of the kd-tree. In Section 3 we describe the kd-tree construction with automatic termination criteria. In Section 4 we show the split clipping algorithm for the kd-tree construction. Section 5 presents results and the practical evaluation of the proposed method on a set of scenes. Finally, Section 6 concludes the paper providing some directions for future work.

2 KD-TREES

The kd-tree is usually built over a set of points in R^d [Berg97], but in the case of ray shooting it is built over bounding boxes of scene objects. For more formal description of kd-tree and efficient ray traversal algorithms see [Havra00b, Havra97].

A kd-tree is constructed hierarchically. At the current leaf l of kd-tree a splitting plane is selected that subdivides the bounding box of l into two boxes corresponding to child nodes. The objects of l are distributed into the new descendants of l. The process is repeated recursively until certain *termination criteria* are reached.

An important feature of the kd-tree is its adaptability to the scene geometry, which can be significantly influenced by positioning of the splitting plane. In the paper introducing kd-trees in the context of ray shooting [Kapla85] the splitting plane is positioned in the mid-point of the chosen axis, and the order of splitting axes is regularly changed. Below we recall a more elaborate approach of the kd-tree construction.

2.1 Surface Area Heuristics

The *surface area heuristics* for the kd-tree construction was introduced by MacDonald and Booth [MacDo90]. Since this approach based on geometric probability is essential for our novel technique, we recall the method in more detail.

The surface area heuristics finds the position of the splitting plane by minimizing a *cost function*. The cost function estimates the *cost* of ray traversing through the kd-tree under some simplifying assumptions. It uses the probability $p(Y|X)$ that an arbitrary ray hits the region Y lying inside a region X once it passes through X. (see Fig. 1).

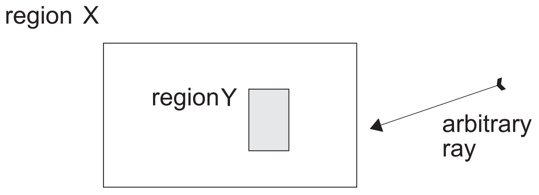

Figure 1: Geometry involved in determination of the probability $p(Y|X)$.

Suppose that both regions X and Y are convex. Then the conditional probability $p(Y|X)$ can be expressed as a ratio of the surface area of the region Y to the surface area of the region X (see [Arvo89]):

$$p(Y|X) \quad = \quad \frac{S_Y}{S_X} \tag{1}$$

The geometry of kd-tree induces axis-aligned bounding boxes for interior nodes and leaves. Further, we denote $AB(v)$ the axis-aligned bounding box associated with the node v. Let us assume the situation at the beginning of the kd-tree construction. The root node contains N objects. All of them would have to be tested for intersection with a ray passing through the scene. Assume that the intersection test for i–th object takes computational time T_i. The cost for such node is expressed as:

$$C = \sum_{i=1}^{N} T_i \tag{2}$$

Recursively subdividing the kd-tree decreases the number of intersection tests, but increases the number of interior nodes to be traversed (see Fig. 2). Now let us consider the case when the ray visits both left and right children of the interior node with some probability given by the surface areas as described above.

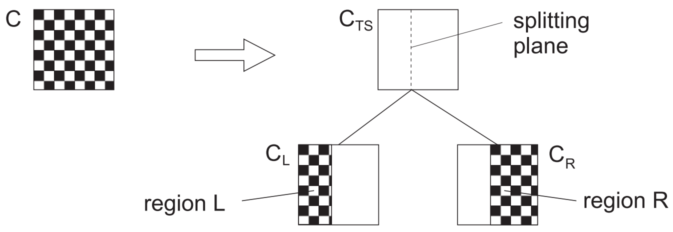

Figure 2: New costs after one splitting.

The node on the left side in Fig. 2 has been replaced by a tree shown on the right side. The original cost C has changed to C_{new} given as the sum of three terms – C_{TS}, C_L, and C_R. Term C_{TS} is the cost of traversing the interior node; it does not incorporate any ray-object intersection tests. Costs for left and right child nodes, C_L and C_R, depend on the conditional probability that a ray hits the left and/or right child of interior node v once it visits v. The new cost C_{new} is given as follows:

$$\begin{aligned} C_{new} &= C_{TS} + C_L + C_R \qquad (3) \\ &= C_{TS} + p_L . \sum_{j=1}^{N_L} T_j + p_R . \sum_{k=1}^{N_R} T_k \\ p_L &= \frac{SA(AB(leftchild(v)))}{SA(AB(v))} \\ p_R &= \frac{SA(AB(rightchild(v)))}{SA(AB(v))} \end{aligned}$$

, where

T_j, T_k	is the time for intersection test with j-th and k-th object respectively,
p_L, p_R	is the probability that a ray will intersect the left and right child respectively,
$SA(AB(v))$	is the surface area of the bounding box associated with the node v,
N_L, N_R	is the number of objects intersecting the AB associated with the left and right node respectively.

The goal is to build a kd-tree with minimized global cost which is computed using the Eq. 3 recursively. Further, we focus on a greedy heuristics that tries to minimize the global cost.

2.2 Termination Criteria

The cost model provides a recipe for positioning the splitting plane in an interior node, but it does not answer an essential question: should we subdivide the node at all, or should we declare the node as a leaf?

Any node v in the kd-tree has the following basic characteristics: its depth $d(v)$ from the root node, the axis-aligned bounding box $AB(v)$ associated with v, and the number of objects N intersecting the interior of $AB(v)$. The construction of the kd-tree implies that the number of objects intersecting $AB(v)$ decreases with increasing depth of the node. Having a node v and its characteristics, we now face the question: *to subdivide or not to subdivide*?

Traditionally, this question is answered by *ad hoc termination criteria* (further abbreviated to AHTC) algorithm, which was given with the introduction of ray shooting algorithms based on the kd-tree [Kapla85] and octree [Glass84]. The AHTC follows the ”common sense” without thoroughly considering the distribution of objects in the scene: the current node v becomes a leaf when the number of objects intersecting the $AB(v)$ is lower than or equal to a fixed constant n_{max}, or its depth $d(v)$ reaches another fixed constant d_{max}. The two constants n_{max} and d_{max} are specified by a user.

The values of these two constants n_{max} and d_{max} are left to the user's experience even in commercial rendering systems that use ray shooting based on kd-trees (Mental Ray [MenRa95]). Kaplan suggests that Constant n_{max} is set to 1 [Kapla85]. We are not aware of any justified recommendations for maximum leaf depth d_{max} in a research paper. In software packages some default values are provided, Mental Ray [MenRa95] uses the following defaults: $d_{max} = 24$ and $n_{max} = 4$. Incorrect setting of these constants can easily result in two problems. The first one is the high space and time complexity for the kd-tree construction. The second one is high time complexity of ray shooting queries. Below we develop an algorithm for automatic estimation of good termination criteria.

3 AUTOMATIC TERMINATION CRITERIA

Subramanian and Fussel [Subra91] coined the term *automatic termination criteria* (further abbreviated ATC), which should not require any user specific constants during kd-tree construction. However, they did not describe any particular algorithm. Motivated by the idea of ATC [Subra98], we have designed to our knowledge the first ATC algorithm based on the cost model.

We can estimate the global cost when kd-tree is being constructed using the Eq. 3 recursively for interior nodes and Eq. 2 for leaves. The cost should correspond to the cost of ray shooting queries after the construction. However, linear relationship between the real and estimated costs is impossible, since the cost model does not cover occlusion and uses just estimates for child nodes.

Fig. 3 shows the real cost in seconds per one ray obtained for ray tracing several scenes from the

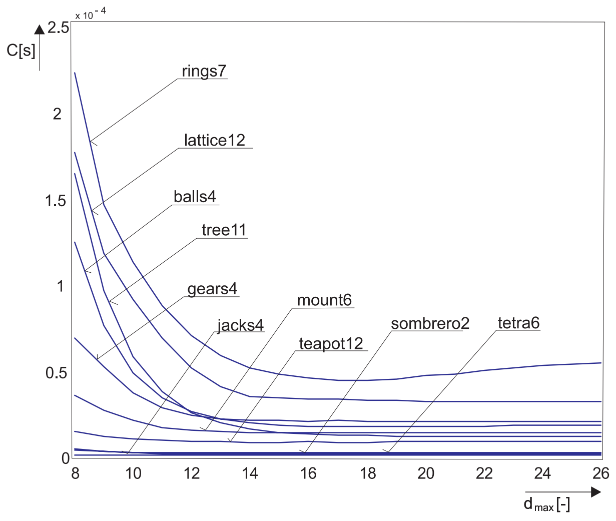

Figure 3: The time in seconds per one ray for different depth of kd-tree built with ad-hoc termination criteria for 10 scenes from SPD database, group G^4.

SPD database [Haine87] using different setting of d_{max}. The number of objects for scenes in Fig. 3 is about 10^4 (group of scenes G^4 as described in Section 5). When the minimum of the global cost would be found for some d_{max}, then the construction could be finished at this depth without significant increase of time complexity for ray queries. The global cost computed according to Eq. 3 follows the shapes of these curves enough precisely.

A simple ATC algorithm can follow the curves on these graphs. We could construct a kd-tree for some maximum leaf depth $d_{max} = d$ and compute the estimated global cost of the kd-tree $C(d)$ using Eq. 3. Then we can subdivide all the leaves of the current kd-tree one step further (if possible) using $d_{max} = d + 1$, and compute the new estimated global cost $C(d + 1)$ of this deepened kd-tree. When $C(d+1)$ is sufficiently lower than $C(d)$, then the kd-tree can be further deepened to depth $(d + 2)$ since it is likely that this subdivision step will decrease the global cost again. If not, kd-tree construction is finished.

However, incrementally increasing maximum leaf depth d_{max} for the whole kd-tree is not the only way to get the pseudo-minimum of the estimated global cost. Actually each node v to be potentially subdivided has its own history of splitting that is associated with all the nodes on the path between the root node and the node v. Setting only one maximum leaf depth d_{max} for the whole kd-tree is not very appropriate to the problem because of locality of the splitting and objects distribution. ATC should keep track of the history of splitting of the kd-tree nodes.

We should remark that one subdivision step need not necessarily bring an improvement in cost immediately. It is possible that the estimated cost increases due to a current subdivision step, even if the splitting plane with minimum cost is selected. This can be caused by two possible reasons. First, the splitting plane could not separate enough objects. In this case, most objects are intersected by the splitting plane, thus a recursive ray traversal algorithm can require one more traversal step. Second, the linear cost estimate of the unsubdivided child node is just an estimate. If a subdivision step is unsuccessful, it does not mean that further subdivision of the corresponding child nodes cannot decrease the global estimated cost considerably. If many unsuccessful subdivision steps occurred on the path between the root node and the current node, this would indicate that objects in these spatial regions were difficult to separate. Avoiding further subdivisions steps in this case could decrease the global cost of the kd-tree.

We observed during experiments with termination criteria algorithms that using some maximum leaf depth d_{max} is a useful feature even in the ATC algorithm. First, the use of d_{max} bounds the maximum memory requirements for the kd-tree representation by limiting the number of kd-tree nodes to a constant ($2^{d_{max}}$). Second, since ABs associated with objects can overlap, the objects need not be separable by a splitting plane at all. In this case the number of objects in a leaf cannot become lower than or equal to a constant n_{max}.

However, setting of d_{max} should follow the number of objects N in the scene. If the objects with the shape of AB do not overlap in the projection to all coordinate axes and the kd-tree with one object in every leaf is required, the maximum depth of a leaf node for the balanced kd-tree is $d_{max} = log_2 N$. When we assume arbitrarily distributed objects in the scene with possible overlapping of objects, then the maximum leaf depth d_{max} to achieve the critical cost point could be higher. To bound this quantity we propose to express the maximum leaf depth as:

$$d_{max} = k_1 . \log_2 N + k_2 \quad (4)$$

The values of constants k_1 and k_2 can be chosen according to experiments on some set of scenes to achieve the critical performance point. We found for experiments on 30 scenes from SPD database [Haine87] that a suboptimal setting of these constants in our rendering system [GOLEM] is $k_1 = 1.2, k_2 = 2.0$.

Further, we discuss the setting of the constant n_{max}. Our observation is based on the case when we access a leaf with n_{max} objects during a ray traversal algorithm. Assuming that the cost of traversal step C_{TS} is sufficiently lower than the cost of ray-object intersection test C_{IT}, it should be advantageous to set $n_{max} = 1$. Then in the ideal case in the constructed kd-tree the leaves contain either one or no object. First, the empty leaves are traversed quickly. Second, if a leaf with one object is checked for ray-object intersection and the intersection exists, no further ray-object intersections need to be performed. Having one object per leaf is not always possible, since objects or/and ABs associated with the objects can overlap. Therefore, during the construction we have to detect these branches of the kd-tree where further deepening does not bring any improvement of the global cost of the kd-tree. In order to detect these cases we also use a cost model.

Below, we improve ATC algorithm by tracking the history of subdividing the spatial region using the cost model. When the node v with N objects is subdivided, its resulting estimated cost is computed according to Eq. 3. When the node v is declared a leaf, its cost is expressed using the Eq. 2. Then we can express the ratio of these two costs, which shows the *quality* r_q of the subdivision step:

$$r_q = \frac{C_{new}}{C}, \quad (5)$$

where C_{new} is computed according to Eq. 3 and C according to Eq. 2. The higher the quality of the subdivision step the lower r_q: for a successful subdivision step we assume $r_q < 1.0$. For example, if a node with AB of cubic shape that contains uniformly distributed objects is subdivided in its spatial median, then for $C_{TS} = 0$ we get the quality of the subdivision $r_q = 2/3$. If r_q is a constant greater than constant r_q^{min} (we can set r_q^{min} to one or some constant slightly lower than one), we can consider the subdivision step unsuccessful. However, the case that $r_q > r_q^{min}$ can occur only transiently, and the quality of subdivision step r_q can improve again in the subsequent subdivision step(s). Therefore, we keep track of the number of unsuccessful subdivision steps on the path from the root node to the current node. If the number of these unsuccessful subdivision steps is higher than some allowed fixed constant F_{max}, we declare the node a leaf, since further subdivision steps are unlikely to decrease the estimated global cost of the kd-tree. Intuitively, for scenes with higher number of objects the number of allowed unsuccessful steps should be higher. Since d_{max} is derived from the number of objects, we propose to compute F_{max} from the maximum leaf depth d_{max} as follows:

$$F_{max} = K^1_{fail} + K^2_{fail} . d_{max} \quad (6)$$

We found experimentally a suboptimal setting for the constants in the GOLEM rendering system: $K^1_{fail} = 1.0$, $K^2_{fail} = 0.2$, and $r_q^{min} = 0.75$.

The ATC algorithm described here is an empirical algorithm based on experiments using 30 scenes of various number of objects from SPD database. We do not claim its optimality or the optimal setting of constants k_1, k_2, K^1_{fail}, and K^2_{fail}. Such setting of constants is implementation dependent, and particularly, it depends on the ratio C_{TS}/C_{IT}. The advantage of the ATC algorithm above is that after specifying these constants it does not require any user setting for each particular scene in runtime. Further, it preserves good time and space complexity of kd-tree construction with regard to the number of objects. The ATC algorithm provides a kd-tree of lower or approximately equal time complexity for ray shooting queries when compared with the kd-tree built with AHTC.

4 SPLIT CLIPPING ALGORITHM

The use of axis-aligned boxes for scene objects significantly simplifies the kd-tree construction.

Since an object O_i can straddle splitting planes, it can happen that after several subdivision steps the object as such does not intersect the bounding box $AB(v)$ of the current node v of the kd-tree hierarchy, although the $AB(O_i)$ of the object intersect the $AB(v)$. This case is depicted in Fig. 4 (a).

Although ABs of objects fit well in the kd-tree construction, there are several disadvantages arising from the simple use of ABs instead of objects' surfaces. First, objects that cannot have an intersection with a ray are also tested in some leaves. Second, objects only virtually present in an interior node v influence the estimated cost of v for both Eq. 3 and Eq. 2 and thus the process of kd-tree construction.

Apparently, one solution would be to postprocess all leaves of the kd-tree and check using an intersection test between the $AB(v)$ associated with a leaf v and the object for all leaves. In this case we do not avoid the problem of influencing the kd-tree construction by objects that only virtually intersect ABs of the interior nodes. In the postprocessed kd-tree we can also get subtree(s) with empty leaves only that should be consolidated to single empty leaves.

Applying the intersection test above for each object and for each position of the splitting plane inside the interior node during construction would be very costly. Therefore, we propose the novel solution based on sweeping technique, which solves the problem with minimum effort, but requires some special object-clipping procedures.

We assume that objects are present in the $AB(v)$ to be subdivided. Then we determine the position of the splitting plane using any method available. If the position of a splitting plane is known, we also know the objects straddling the splitting plane. For these objects we reduce the two ABs on both sides of the splitting plane as much as possible, as depicted in Fig. 4 (c). This step requires a special clipping method for all shapes of objects. For a given object O, its current AB, and the position and orientation of a splitting plane, we want to construct two tight ABs associated with the fragments of the object on both sides of the splitting plane. These reduced ABs for the both sides of the splitting plane must be passed during the construction together with the reference to the object. We call this method that clips the ABs of objects with regard to a splitting plane *split clipping* (further abbreviated as SC). The description of split clipping procedures for particular objects shapes is omitted due to the lack of space.

5 RESULTS

We verified experimentally the performance of kd-trees built with ATC using 30 scenes from Standard Procedural Database [Haine87]. We divide the scenes into three groups, G_3, G_4, and G_5, with different number of objects to justify our ATC algorithms. G_3 consists of scenes with about 10^3 objects, G_4 of scenes with about 10^4 objects, and G_5 of scenes about 10^5 objects, each group contains ten scenes.

The results are reported using the set of parameters given by *Minimum Testing Output* described in [Havra00a] for simple ray tracing. For testing we used parameters suggested in the `Readme.txt` file in the SPD package distribution [Haine87] (the number of primary rays shot is 513×513, depth of recursion is 4). All experiments were conducted on a Pentium III, 466 MHz, 128 MBytes RAM, running Linux. The test program in the GOLEM rendering system [GOLEM] was compiled using `gcc` with -O2 optimization. The parameters for the subset Θ of the minimum testing output were obtained using a software profiler tool.

We measured the results of AHTC, further denoted (d_{max}, n_{max}) for all three groups averaged for all the scenes within one group. Further, we measured the results for ATC with and without SC, also averaged over all scenes within one group. We also computed average ($\tilde{G}$) for all the scenes. The results are given in Table 1. In order to evaluate the memory complexity we use the number of nodes referenced $(N_G + N_E)$ and we can also compare the number of references to objects in the leaves (N_{ER}). For comparison of time complexity we consider Θ_{RUN}, which is virtually independent of hardware used.

We can observe that AHTC can increase memory complexity by a factor of up to 300% (for G_3 and (24,1) compared with ATC). When n_{max} is set too low, the time complexity is drastically in-

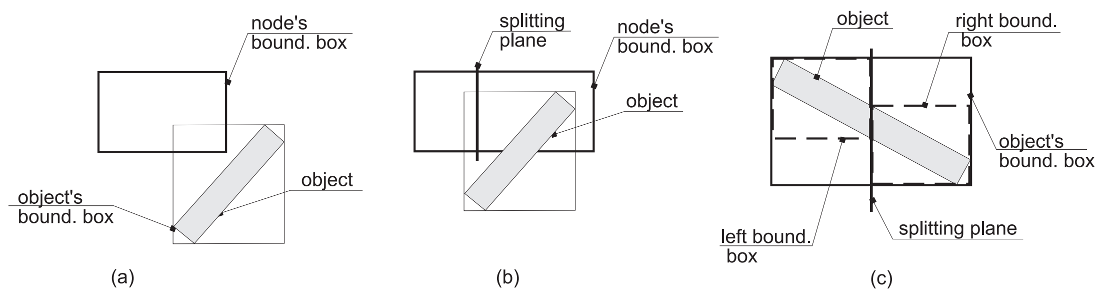

Figure 4: (a) Splitting during the kd-tree construction can result in references to objects that have no intersection with the leaves. (b) When the AB associated with an object straddles the slitting plane, it is not guaranteed that the object also straddles the splitting plane. (c) Split clipping – reducing the AB of the object by clipping, one box on the left side and second one on the right side of the splitting plane.

creased, which is most visible on group G_5 and (8,2). The best results when using AHTC are reached for (24,2) setting. However, this setting is not suitable when the number of primitives is small, as in G_3. On average for all 30 scenes, ATC compared with (24,2) decreases the time complexity of ray shooting queries by 4.2% decreasing the number of nodes by 7% and number of references to objects by 31%. At the best case, kd-tree with ATC improves the time complexity of ray shooting queries by 20% over (24,2) and in the worst case ATC is 10% slower than (24,2). The kd-tree with ATC performs faster for ray shooting than (24,2) for 23 of 30 scenes tested.

Further we discuss the results for the kd-tree built with SC algorithm that we used together with ATC. We compare this approach to the kd-tree built with ATC and without SC. The time T_B required to construct kd-tree is increased, the time of the ray tracing algorithm is decreased as expected. The time to build the kd-tree is increased by 140% (taken by split clipping of objects), the number of nodes is increased by 5.4% on average, and the number of empty leaves is increased by 30% on average (memory efficient, null pointer is used in the interior node containing empty leaf). The number of references to objects is decreased by 10.5%, the number of ray-object intersection tests is decreased by 21%, and the time complexity for ray shooting queries is improved by 9% on average. For large scale scenes, when the kd-tree is built in advance and saved to a file for interactive or real time rendering [Wald01], the increase of T_B is nothing to worry about, since the low memory requirements and performance of ray shooting is the main concern. The best improvement of time complexity for ray shooting for a single scene is 35%. This improvement is significant, since ray shooting algorithm with kd-tree nearly touches the theoretical lower bound $O(logN)$ of time complexity on average [Szirm98].

6 CONCLUSION AND FUTURE WORK

In this paper we have shown that the new automatic termination criteria provide the reasonable way for constructing kd-trees in the context of ray shooting algorithms. Even if the average improvement over the ad hoc termination criteria is not (and cannot be) overwhelming, we have shown that for scenes of various number of objects it provides a good setting that also minimizes the memory usage. The particular importance for the user of a rendering system with automatic termination criteria is that this method does not require any constants to be set for a particular scene.

The kd-tree construction with split clipping method solves correctly the problem of presence of objects inside the node to be subdivided. This method is optimal in the number of checked objects and provides a correct solution for referencing of objects in the leaves and during the construction. Even if it increases the building time of kd-tree, it further decreases the time complexity of ray shooting by 9% on average.

The kd-tree construction with split clipping offers some space for further investigation concerning the efficient split clipping methods for various shapes of objects. We did not optimize these clipping algorithms. The automatic termination

Scene	Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
G_3	(8,1)	140	141	29	1292	52.09	12.44	2.76	0.74	0.07	21.02	13.29	0.77	37.51
	(8,2)	137	138	26	1292	52.95	12.18	2.71	0.59	0.06	21.03	13.29	0.77	37.59
	(16,1)	3690	3691	767	4844	10.30	22.03	4.24	1.69	0.17	12.61	13.29	0.31	19.43
	(16,2)	1870	1871	255	3590	12.93	18.97	3.71	1.05	0.13	13.10	13.29	0.39	20.57
	(24,1)	8978	8979	1133	12429	10.30	23.72	4.52	1.73	0.31	13.22	13.29	0.29	20.23
	(24,2)	4303	4304	298	8663	13.02	19.83	3.85	1.05	0.20	13.19	13.29	0.38	20.19
	ATC	2043	2044	501	2937	12.80	19.25	3.78	1.55	0.13	12.83	13.29	0.37	19.51
	ATC+SC	2142	2143	602	2793	10.31	19.54	3.84	1.77	0.22	12.08	13.29	0.33	17.65
G_4	(8,1)	172	173	24	8757	250.54	12.62	2.67	0.69	0.80	75.45	13.73	0.94	175.32
	(8,2)	171	172	22	8757	250.75	12.50	2.64	0.65	0.78	75.52	13.73	0.94	175.62
	(16,1)	12421	12422	3042	21654	13.03	26.67	4.86	2.14	1.34	16.18	13.73	0.32	24.75
	(16,2)	8317	8318	1263	19515	15.07	24.47	4.49	1.61	1.26	16.76	13.73	0.38	25.98
	(24,1)	65655	65656	8352	91893	10.84	31.37	5.65	2.35	2.73	16.86	13.73	0.26	25.45
	(24,2)	33433	33434	2414	66189	13.32	26.92	4.88	1.66	2.02	16.77	13.73	0.33	25.27
	ATC	16873	16874	3836	25848	12.31	26.70	4.86	2.19	1.45	16.01	13.73	0.31	24.14
	ATC+SC	17462	17463	5282	22824	9.78	26.81	4.89	2.56	2.60	14.82	13.73	0.27	22.04
G_5	(8,1)	181	182	23	104396	2659.95	12.66	2.62	0.63	14.01	883.41	13.40	0.99	2696.34
	(8,2)	180	181	22	104396	2660.08	12.62	2.61	0.62	14.01	884.45	13.40	0.99	2698.10
	(16,1)	24068	24069	4124	147680	40.80	27.54	4.79	1.91	19.60	26.24	13.40	0.59	46.44
	(16,2)	23113	23114	3580	147514	41.69	26.99	4.71	1.78	19.52	26.35	13.40	0.60	46.27
	(24,1)	476453	476454	83642	663728	11.53	38.32	6.55	3.00	35.32	23.46	13.40	0.26	31.60
	(24,2)	251440	251441	28940	498597	14.18	33.74	5.76	2.21	28.70	21.09	13.40	0.34	30.14
	ATC	249915	249916	55707	366588	12.36	35.23	6.01	2.90	27.26	19.94	13.40	0.29	28.78
	ATC+SC	262549	262550	73790	328496	9.60	35.10	5.97	3.40	66.34	18.51	13.40	0.25	26.11
	(8,1)	165	166	25	38149	987.52	12.57	2.68	0.69	4.96	326.62	13.47	0.90	969.73
$\tilde{G}$	(8,2)	162	163	23	38148	987.93	12.43	2.65	0.62	4.95	327.00	13.47	0.90	970.44
	(16,1)	13393	13394	2644	58059	21.38	25.41	4.63	1.91	7.04	18.34	13.47	0.41	30.20
	(16,2)	11100	11101	1700	56873	23.23	23.48	4.30	1.48	6.97	18.74	13.47	0.46	30.94
	(24,1)	183695	183696	31042	256017	10.89	31.14	5.57	2.36	12.79	17.85	13.47	0.27	25.76
	(24,2)	96392	96393	10551	191150	13.51	26.83	4.83	1.64	10.31	17.01	13.47	0.35	25.20
	ATC	89610	89611	20014	131791	12.49	27.06	4.89	2.21	9.61	16.26	13.47	0.32	24.14
	ATC+SC	94051	94052	26558	118038	9.90	27.15	4.90	2.58	23.05	15.14	13.47	0.28	21.94

Table 1: Average values for scene groups G_3, G_4, G_5; $\tilde{G}$ is average for all 30 scenes. Notation: (d_{max}, n_{max}) – ad hoc termination criteria with d_{max} and n_{max}, ATC – automatic termination criteria, SC – split clipping. Minimum testing output: N_G – number of interior nodes, N_E (N_{EE}) – number of (empty) leaves, N_{ER} – total number of objects references in leaves, r_{ITM} – ratio between the number of intersection tests to minimum number of intersection tests, $\tilde{N}_{TS}$ – average number of nodes accessed per ray, $\tilde{N}_{ETS}$ ($\tilde{N}_{EETS}$) – average number of (empty) leaves accessed per ray, T_B – time required to build kd-tree, T_R – running time of the ray tracing, Θ_{APP}, Θ_{rat}, Θ_{RUN} – invariants of the hardware related to ideal ray shooting time. Particularly interesting is Θ_{RUN} as the ratio of the time spent on ray shooting to the ideal ray shooting time [Havra00a, Havra00b] since it is the ratio between the real time and the solution of $O(1)$ time complexity.

criteria algorithm as presented in this paper is the first step in the direction to construct the data structures without requirements of any constants from the user and providing good performance for scenes of different complexities.

Acknowledgements

This research was partially supported by the Czech Ministry of Education under Project LN00B096 and the Aktion Kontakt OE/CZ grant number 1999/17.

REFERENCES

[Arvo89] Arvo,J, Kirk,D: A survey of ray tracing acceleration techniques, In *An Introduction to ray tracing*, pp. 201–262, Academic Press, 1989.

[Berg97] de Berg,M, et al:*Computational Geometry: Algorithms and Applications*, Springer–Verlag, 1997.

[Glass84] Glassner,A Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, Oct. 1984.

[Haine87] Haines,E: A proposal for standard graphics environments, In *IEEE Computer Graphics and Applications*, Vol. 7, No. 11, pp. 3–5, 1987, Available at `http://www.acm.org/pubs/tog/resources/SPD/overview.html`

[Havra97] Havran,V, Kopal,T, Bittner,J, Žára,J: Fast robust BSP tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, 2(4):15–23, December 1997.

[Havra00a] Havran,V, Purgathofer,W: Comparison methodology for ray shooting algorithms. *Technical Report TR-186-2-00-20*, Institute of Computer Graphics, Vienna University of Technology, Favoritenstrasse 9/186, A-1040 Vienna, Austria, November 2000. Human contact: technical-report@cg.tuwien.ac.at. Under submission.

[Havra00b] Havran,V: Heuristic Ray Shooting Algorithms, Ph.D. thesis, Czech Technical University, November 2000. Available at `http://www.cgg.cvut.cz/~havran/phdthesis.html`.

[GOLEM] Havran,V: GOLEM rendering system, 2000. HOME page at `http://www.cgg.cvut.cz/GOLEM`.

[Kapla85] Kaplan,M: The Use of Spatial Coherence in Ray Tracing, *ACM SIGGRAPH'85 Course Notes 11*, pp. 22–26, July 1985.

[MacDo90] MacDonald,J, Booth,K: Heuristics for ray tracing using space subdivision, *The Visual Computer*, Vol. 6, No. 3, pp. 153–166, 1990.

[MenRa95] SOFTIMAGE: *Mental Ray, A Programmer's Reference Guide*. Gesellschaft für Computerfilm and Maschinintelligenz Gmbh & Co. KG, Berlin, 1995.

[Subra91] Subramanian,K, Fussell,D: Automatic Termination Criteria for Ray Tracing Hierarchies, *Proceedings of Graphics Interface '91*, pp. 93–100, 1991.

[Subra98] Subramanian, K: Personal communication, 1998.

[Szirm98] Szirmay-Kalos,L, Márton,G: Worst-case versus average case complexity of ray-shooting. *Computing*, 61(2):103–131, 1998.

[Wald01] Wald,I, Slusallek,P, Benthin,C, Wagner,M: Interactive Rendering with Coherent Ray Tracing. *Eurographics'2001* conference, pp. 153-164, 2001.