When It Makes Sense to Use Uniform Grids for Ray Tracing

Michal Hapala
Czech Technical University in Prague
Faculty of Electrical Engineering
Czech Republic
hapalmic{@}fel.cvut.cz

Ondřej Karlík
Czech Technical University in Prague
Faculty of Electrical Engineering
Czech Republic
karliond{@}fel.cvut.cz

Vlastimil Havran
Czech Technical University in Prague
Faculty of Electrical Engineering
Czech Republic
havran{@}fel.cvut.cz

ABSTRACT

Commonly used hierarchical data structures such as bounding volume hierarchies and kd-trees have rather high build times, which can be a bottleneck for applications rebuilding or updating the acceleration structure required by data changes. On the other hand uniform grids can be built almost instantly in linear time, however, they can suffer from severe performance penalty, in particular in scenes with non-uniformly populated geometry. We improve on performance using a two-step approach that combines both approaches: first we build a uniform grid and test its performance. Second, using an estimate on the number of rays to be queried we either continue using the grid or build a hierarchical data structure instead. This way we select a more efficient data structure given a particular implementation of the algorithms which yields with high probability an overall smaller computational time. We evaluate the properties of this method for a set of 28 scenes.

Keywords: ray tracing, ray casting, uniform grid, kd-tree, hierarchical data structures.

1 INTRODUCTION

Ray tracing is a technique that can be used for generating images by shooting rays into a 3D scene and finding closest intersections among rays and the scene objects. This basic visibility computation is used in a core of many rendering algorithms. Although ray tracing has been known for over last four decades [App68, Gla89], it is still considered relatively slow to be massively used in real-time applications particularly for animated scenes.

Different data structures have been proposed, each one with its own advantages and disadvantages. Most commonly used are hierarchical data structures, e.g. a kd-tree [Ben75] and a bounding volume hierarchy (BVH) [Kay86]. Their main advantage is their capability to adapt to the distribution of geometric primitives – they can deal with non-uniform geometry distribution, including so-called "teapot in a stadium" type of scenes. Because of that they perform well in a vast majority of scenes encountered in real-life use. They are typically built in $O(N \log N)$ or $O(N \log^2 N)$ time using the *surface area heuristic* (SAH) [Wal06a]. Super-linear time complexity of their build algorithms can be a bottleneck in particular when tracing rays in applications with real-time requirements.

Another type of an acceleration structure is a uniform grid [Fuj86]. In its simple form it divides a scene regularly and non-adaptively into equally-sized voxels and sets the primitive references to each cell overlapped by the primitives. Ray then traverses cells along the ray path and only geometric primitives in these cells are tested for an intersection. Although this data structure can perform well in certain types of scenes (typically with uniformly distributed primitives), its performance degrades drastically in scenes with a non-uniform distribution. Despite this fact, uniform grids can still be advantageous for usage in real-time applications because their build algorithm has only a linear time complexity.

The availability of two approaches with different properties presents us with a choice whether to use either an adaptive data structures that will most likely be efficient for shooting rays, but have higher time complexity for building, or a simple regular one like the aforementioned uniform grids, which have lower build time but can have a severe performance penalty.

In this paper we propose and study such an algorithm that uses the estimate of performance properties choosing acceleration data structures on the fly. The algorithm uses a calibration phase which requires a set of scenes of different properties such as number of geometric primitives and their spatial distribution. The calibration phase is executed only once before the algorithm is used for an application on a particular hardware. During the calibration phase we measure the implementation and hardware constants for building up data structures and also a practical efficiency of shooting rays. Given an unknown scene we build a uniform grid first in a short time and test its performance by sampling a small set of representative rays. Using the data from this test and data from the calibration phase we estimate if it is more advantageous to use the uniform grid or to discard it and build a hierarchical data structure instead. To make a correct decision we need to know at least roughly the number of rays to be shot in the application.

This paper is further structured as follows. Section 2 describes the previous work on the most relevant data structures. Section 3 describes the proposal of our algorithm. Section 4 shows the results obtained from the set of 28 scenes. Section 5 concludes the paper with some prospectives for future work.

2 PREVIOUS WORK

In this section we briefly recall the most important work on uniform grids and hierarchical data structures. As the number of papers is huge, we select only the most recent and important work to our approach.

Uniform Grids. The uniform grids, also called regular subdivision, were proposed by Fujimoto et al. [Fuj86]. Cleary and Wyvill [Cle88] analyse the properties of ray tracing with uniform grids in dependence on its resolution. They study the performance when the number of cells in the uniform grid is proportional to the number of objects. Other methods were also studied by Ize et al. [Ize07]. The performance of ray tracing with uniform grids has two important factors. First, the initial setup time given a ray is relatively high. Second, when the distribution of primitives in a scene is highly uniform, it is likely that the ray will stop its traversal after only a few traversal steps. Therefore, the uniform grids are for such types of scenes even more efficient than hierarchical data structures that require initial traversal phase to a first leaf. However, for moderately to highly non-uniform distribution of geometric primitives in space the uniform grids are rather inefficient as studied for example by Havran et al. [Hav00b].

Given an arbitrary ray the number of traversed cells is of order $O(\sqrt[3]{N})$ in the worst case, where N is the number of object primitives hence the number of all grid cells. The second important property is the time needed for building a uniform grid, which is only $O(N)$ provided each geometric primitive is assigned to only a constant number of cells. Wald et al. [Wal06b] studied the coherent traversal algorithm for primary rays that reaches real-time framerates. Recently, Kalojanov and Slusallek [Kal09] presented the algorithm for parallel building of uniform grids on a GPU.

Hierarchical Data Structures. Hierarchical data structures for ray tracing were studied in a number of papers. Chang [Cha04], Wald [Wal04], and Havran [Hav00a] provide the survey on the spatial data structures for ray tracing static scenes with the focus on the hierarchical data structures. The common property of the data structures is that the time complexity for building is $O(N \log N)$ since it corresponds to sorting in 3D space. While the time complexity for building is higher than the one for uniform grids $O(N)$, the time needed for ray query can be estimated by $O(\log N)$. The performance of the ray is hence much less dependent on the number of objects than for uniform grids as we also show further in the paper. The properties and relations between these data structures in general were discussed by Havran [Hav07]. Another study that compares the performances of a grid and a kd-tree was presented by Szirmay-Kalos et al. [SKH02].

Selection Algorithm. We are aware of only two algorithmic proposals that considers the use of different data structures. The first one proposed by Havran et al. [Hav00b] is based on statistical properties for the same input data. They analyse the distribution of objects in the scene and if selected statistical characteristics are low without giving any threshold, they suggest to use uniform grids, otherwise kd-trees or other hierarchical data structures such as adaptive grids. The statistics measures that are easy to compute from only the distribution of geometry in the scene are sparseness, maximum number of primitives referenced in the cell, and statistical moments as mean, variance (hence also standard deviation), skewness, and kurtosis. We recall below the formulas for these measures computed over t cells of a grid taking into account the references of geometric primitives in the cells denoted by N_i for the i-th cell.

Scene sparseness is the ratio of empty cells to all cells:

$$sparseness = \frac{\#empty\ cells}{N} \qquad (1)$$

High values of sparseness could indicate inferior grid performance, as adaptive data structures generally deal better with cutting off empty space. Big gaps with no geometry in the scene create many empty cells in the grid which have to be traversed unnecessarily.

Maximum number of primitives in any cell is defined simply as:

$$maxRefs = \max(N_i), i \in 1 \ldots t \qquad (2)$$

High value could be useful in detecting a "teapot in a stadium" type of scene.

Mean represents an average number of references per cell:

$$mean = \frac{1}{t}\sum_{i=1}^{t} N_i \qquad (3)$$

Because the grid is built to have the number of cells proportional to the number of geometric primitives in the scene, high mean values indicate low performance as that means that there are many primitives overlapping multiple cells.

Variance gives us information about how much values differ from the mean value. It is computed using this formula:

$$variance = \frac{1}{t-1}\sum_{i=1}^{t}(N_i - mean)^2 \qquad (4)$$

Higher variance corresponds to the higher differences in the data: there may be many empty cells but also many cells with high number of primitives. Standard deviation σ is computed from variance as $\sigma = \sqrt{variance}$.

Skewness describes asymmetry of the distribution and *kurtosis* describes "peakedness" of the distribution belong to higher statistical moments and are defined as:

$$skewness = \frac{1}{t}\sum_{i=1}^{t}\left(\frac{N_i - mean}{\sigma}\right)^3 \qquad (5)$$

$$kurtosis = \frac{1}{t}\sum_{i=1}^{t}\left(\frac{N_i - mean}{\sigma}\right)^4 - 3 \qquad (6)$$

The aforementioned metrics can be computed directly from the uniform grid based on a voxelisation approach proposed by Klimaszewski [Kli94].

Another approach to selecting a better acceleration structure on the fly was proposed by Müller and Fellner [MF99]. They create a bounding volume hierarchy for a given scene and try to find regions (nodes) that contain uniformly distributed objects. A uniform subdivision of space to a predetermined number of voxels is then created in these regions.

3 ALGORITHM OUTLINE

In this section we present the algorithm that given a scene suggest to use either the uniform grids or a hierarchical data structure in the dependence on the number of rays. We analyse such case and suggest an algorithm that estimates if it is more convenient to use an already built grid or to build up a hierarchical data structure. Our decision algorithm can be used for virtually any application of ray tracing that implements grids and hierarchical data structure in the framework. The data from the implementation are extracted in the *calibration phase* that is executed only once on a given hardware/implementation on a set of scenes.

We verified the observation by Havran et al. [Hav00b], if the suggested selection algorithm between grids and kd-trees is valid for another set of scenes. We have found out that on our set of scenes (see Figure 3) there is no scene with such low standard deviation, skewness, and kurtosis as in the study that could justify the use of uniform grids based only on the scene statistics. However, when analysing the statistical characteristics in [Hav00b] it appears that the threshold for standard deviation should be very low, such as 2.0, to justify the use of uniform grids. This leads to the higher performance of ray shooting irrespective to the number of rays even if we ignore the time needed to build the data structure.

In this paper we study another case when the number of rays to be shot is known or well estimated in advance and we account for the time needed to build the data structure. The algorithm requires the calibration phase over all s scenes in a set S_{cal} (i.e. $s = |S_{cal}|$).

The calibration phase computed for i-th scene having $N(i)$ geometric primitives for all the scenes in the set S_{cal} has four steps:

1. Build a uniform grid [Fuj86] over $N(i)$ geometric primitives of the i-th scene with the number of cells proportional to $N(i)$. Measure the time $T_B^G(i)$ to build the uniform grid.
2. Measure the time $T_R^G(i)$ for $M(i)$ ray queries using the ray traversal algorithm over the uniform grid.
3. Build a hierarchical data structure over $N(i)$ geometric primitives. Measure the time $T_B^H(i)$ needed for the build.
4. Measure the time $T_R^H(i)$ for $M(i)$ ray queries (the same ray queries as for uniform grid) using the ray traversal algorithm over the hierarchical data structure.

The data from the calibration phase are then used for an application scenario given an unknown scene S with N geometric primitives. To improve on the performance we decide on both cases assuming the knowledge or the rough estimate for the number of rays R to be queried.

Decision algorithm:

1. Build a uniform grid [Fuj86] over N geometric primitives of scene S with the number of cells proportional to N. Measure the time $T_B^G(i)$ to build the uniform grid.
2. Estimate the time t_R^G needed for computing a single ray with the uniform grid. This is carried out by sampling using a small set of rays.
3. Estimate the time T_B^H to build a hierarchical data structure from the calibration phase and from N.
4. Estimate the time t_R^H to ray trace a single ray from the calibration phase and from N.
5. If $t_R^H \geq t_R^G$ then use the uniform grid to shoot all the rays. Finish.

6. Estimate the critical point, it is the number of rays R_C, when the uniform grid and hierarchical data structure yields the same computation time taking into the account the estimated build time of the hierarchical data structure. This is computed as: $R_C = T_B^H / (t_R^G \cdot (1+\varepsilon) - t_R^H)$. The parameter ε is used only to avoid division almost by zero, we use the value such as $\varepsilon = 0.01$.

7. If $R_C \leq R$ (R is the number of rays to be queried), use uniform grids for the rest of the computation. Finish.

8. Otherwise, discard the uniform grid and build the hierarchical data structure. Shoot all the remaining rays using hierarchical data structure. Finish.

To put it short the decision algorithm above simply computes the estimate whether or not it is more advantageous to use an already built uniform grid or if it pays off to build a hierarchical data structure.

The only information computed after building the uniform grid is the build time T_B. To estimate the critical point for the number of rays R_C we need to estimate the average time t_R^G to shoot a single ray using the uniform grid, the time needed to build the hierarchical data structure T_B^H, and the time t_R^H for shooting a single ray using this (unbuilt) data structure.

Below we describe how to estimate these qualitative performance characteristics. The average time t_R^G which gives an average time to shoot a single ray in uniform grid is estimated by sampling of small number of rays such as 100 to 1000 rays. This provides an accurate estimate, the only condition is that the sampling rays represent the distribution of all rays.

The time needed to build the hierarchical data structure T_B^H is estimated using the time complexity of a build $O(N \log N)$ and from the times $T_B^H(i)$ needed to build these data structure in the calibration phase as follows:

$$T_B^H = N \cdot \log_2 N \cdot \frac{1}{s} \sum_{i=1}^{s} \frac{T_B^H(i)}{N(i) \cdot \log_2 N(i)} \qquad (7)$$

Similarly, we can estimate the time to shoot a single ray T_R^H in a hierarchical data structure under the assumption of $O(\log_2 N)$ time complexity for this operation, using the time $T_R^H(i)$ needed for the same algorithm from the calibration phase as follows:

$$t_R^H = \log_2 N \cdot \frac{1}{s} \sum_{i=1}^{s} \frac{T_R^H(i)}{M(i) \cdot \log_2 N(i)} \qquad (8)$$

Analysis and Discussion

After building the grid the algorithm above provides three possible outcomes. First, if the estimated time t_R^G to shoot a ray in the grid is lower than the estimated time t_R^H to shoot a ray in the hierarchical data structure, it does not make sense to build a hierarchical data structure. Second, for the number of rays to be shot in the range between 0 and R_C it does not pay off to build up a hierarchical data structure. This is because the time to build a hierarchical data structure is relatively high even if it provides faster processing of a single ray and for relatively small number of rays it does not pay off. Third, for the number of rays larger than R_C it is then always more efficient to discard the uniform grid, build the hierarchical data structure and use it to shoot the rays.

Below we compare a proposed algorithm combining uniform grids and hierarchical data structures with a single use of the either two data structures. When we compare it to the use of only the grids, the proposed algorithm is more efficient as it is always of the same performance or provides the speedup in cases when we detect that the grids are inefficient.

We also compare the proposed algorithm to the use of only the hierarchical data structure as we need the additional time to build the uniform grid. Favourably, the time complexity $O(N)$ is asymptotically smaller than the time complexity needed to build the hierarchical data structure $O(N \log_2 N)$. Theoretically, the time complexity needed to build the uniform grid, which is possibly later discarded, gives the time complexity increase from $O(N \log_2 N)$ to $O((N(1+\log_2 N))$ that presents the slowdown of building of only a hierarchical data structure $1 + 1/\log_2 N$. In practice when we take into account the particular implementation, the constants behind the time complexities for building them are even higher for hierarchical data structure when compared to uniform grids. Therefore such slowdown is negligible. For the test scenes used in this paper the slowdown is only 4.5% on average, with a minimum value of 2.1% (1,070,671 triangles) and a maximum value of 15.4% (528 triangles).

We can also express the maximum theoretical speedup for using the combined solution when using only the hierarchical data structure. This is only for a small number of rays with a limit of $O(\log_2 N)$ and the speedup is then $\frac{T_B^H + t_R^H \cdot \log_2 N}{T_B^G + t_R^G \cdot \log_2 N}$. The speedup reaches the average value of 28.07 with a minimum of 6.5 (528 triangles) and a maximum of 46.50 (1,070,671 triangles). We avoid the discussion for a trival case – it does not pay off to build up any hierarchical data structure if the number of rays to be shot is smaller than $O(\log_2 N)$.

4 RESULTS

We have implemented a path tracer application in C++. We report here the results for a PC equipped with Intel Core 2 Duo E4300 1.8 G Hz (Allendale) and 6 GBytes of RAM, running Windows 7 operating system in Microsoft Visual C++ 2008. For testing we have used a set of 28 scenes, 20 of them unique and 4 scenes tessellated to triangles in two level of details, see Table 1.

For each scene we have measured uniform grid and hierarchical data structure build times and traversal times for two types of ray generation schemes. The first scheme uses rays generated from two points randomly generated on a bounding sphere of the scene, the second one uses rays generated by the path tracer. As a hierarchical data structure we used an implementation of a kd-tree. From the measurements we have computed exactly the critical point for the number of rays R_C and each scene where rendering using the uniform grid is faster according to the equations provided in the previous section. To test the quality of our estimate algorithm we have compared the exactly computed R_C and its estimated value R_{est} when the hierarchical data structure was not build. We report by how many percent the estimate of R_C is inaccurate (relative error $Err = 100 \cdot \frac{R_{est} - R_C}{R_C}$).

This was carried out for random combinations of calibration and estimated scenes as follows: always a certain number of scenes from the set C are used in the calibration stage, and the rest is used to test the accuracy of the estimate. This number C is increased from 1 to 27, thus for the first case one random scene would be the base for the calibration and twenty seven would be estimated and for the last one the situation is reversed. To gain some convergent data we have repeated the computation 5000 times for every of these cases. Both graphs in Figure 1 and Figure 2 show the average value of estimated relative errors in percent ($\frac{1}{5000}\sum Err$) in red and the average of absolute values of relative errors in percent ($\frac{1}{5000}\sum |Err|$) in blue colour.

For randomly generated rays (see Figure 1) the estimate is about 25 percent more optimistic about the quality of the grid and is quite stable in this prediction except for the extremes where there are either not enough calibration scenes or not enough estimated scenes. The estimate predicts that it is safe to shoot more rays with the grid still being faster than in reality.

For path traced rays (see Figure 2) the estimate is off by around 30 percent, but this prediction is not as stable as for the randomly generated rays and the tendency is to predict that the grid is worse than it really is. Since a big part of our path traced rays are primary rays or shadow rays, this is not an efficient sampling of the space of possible rays with regard to providing good calibration for other scenes. This can also occur for non-diffuse scenes, where glossy reflections will result in a non-uniform sampling.

From the results we see that the range of rays where its does not pay off to build the hierarchical data structure can be significant in particular for scenes with a higher number of geometric primitives. For example for the scene *phone-high* the critical point for the number of rays R_C is 2.7×10^6 rays to justify the use of hierarchical data structure for random rays and 856×10^3 rays for path tracing. The results for 28 scenes also show that without computing the estimate the selection cannot be made in general.

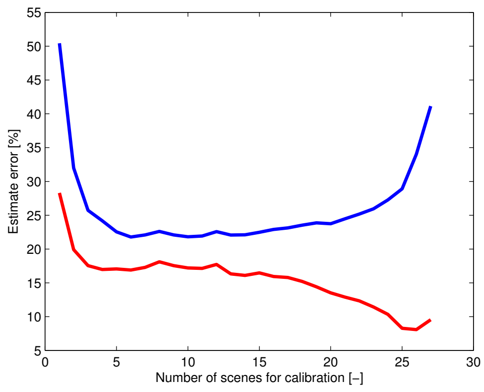

Figure 1: Estimate error for random rays calibrated on random rays.

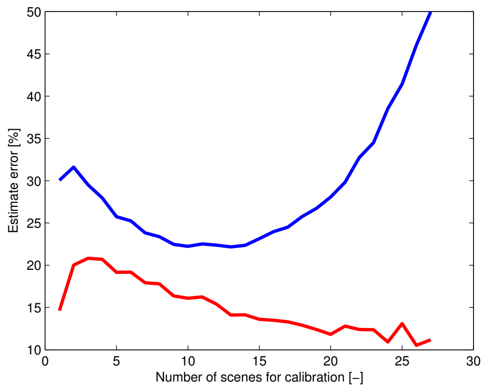

Figure 2: Estimate error for the rays from path tracing calibrated on rays from path tracing.

5 CONCLUSION AND FUTURE WORK

We have proposed an algorithm that combines a uniform grid and a hierarchical data structure for ray tracing so that it takes advantages of both types. Based on the scene properties and a small number of rays computed using the grid we decide either to continue ray tracing with the grid or to build the hierarchical data structure such as kd-trees.

We show that the use of uniform grids is relatively limited for standard scenes with the exception of scenes with a special distribution of geometric primitives in space. To our best knowledge we present the first algorithm that decides when it is advantageous to use uniform grids in dependence on the number of rays to be shot. Compared to the use of only a hierarchical data structure the method has a slowdown of only $1+1/log_2N$ for the building of the data structure in the worst case. We can reach the average speedup 28.07 for a small number of rays. Our method can be used

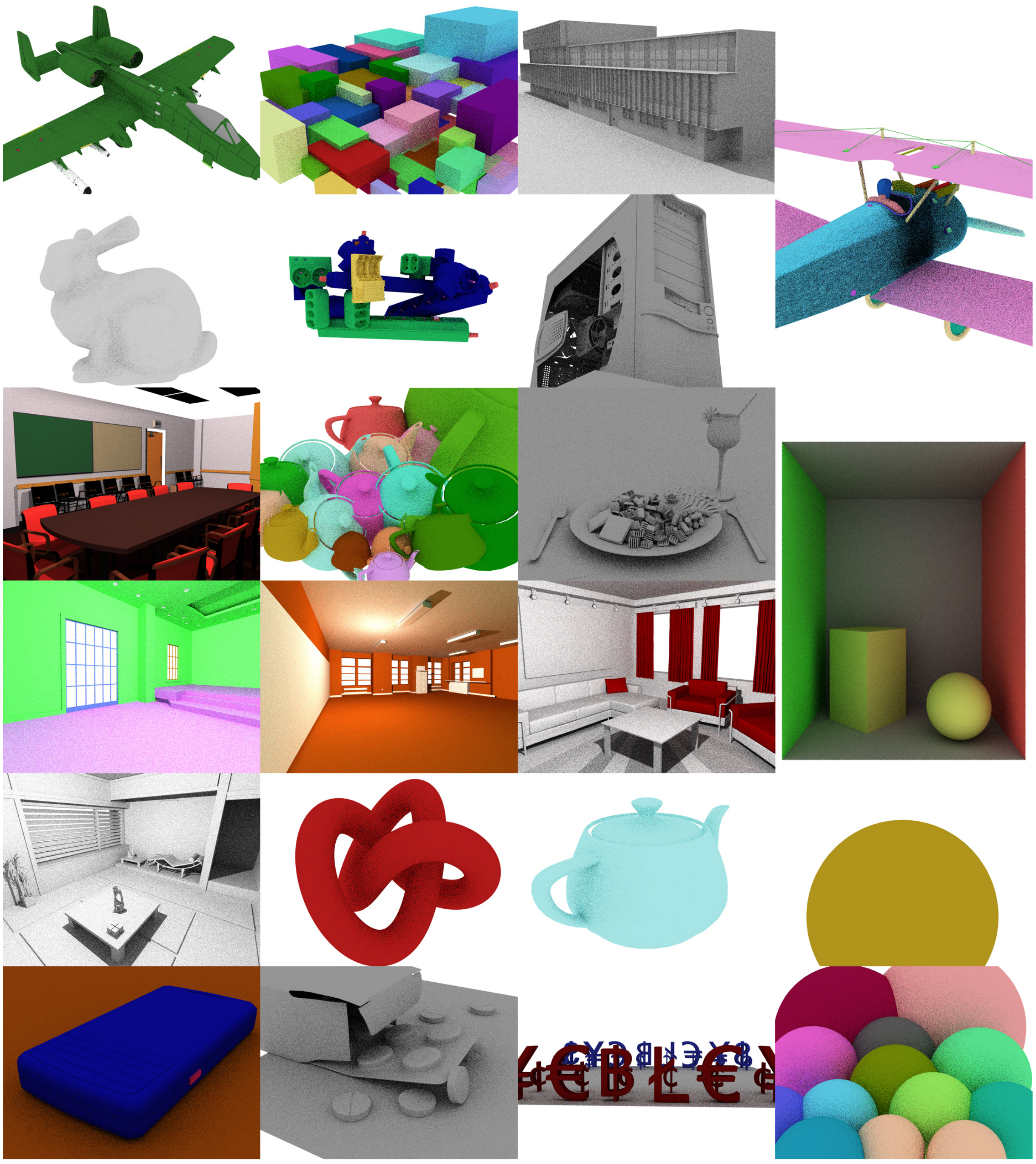

Figure 3: All used test scenes. There are multiple scenes with the same model which differ only in a polygon count. Only one picture for each such group is shown. Scenes from the top-left are: a10, boxes, building, camel, bunny, sockets, case, conference, teapots, hunger, cornell, interior-3, interior-dance, interior-deloix, interior-japan, knot, teapot, sphere, phone, pills, chess, spheres.

in any hardware platform and any implementation of ray tracing that uses uniform grid and hierarchical data structure. We show that the number of rays that gives the critical point for the same performance of grids and hierarchical data structure can be well estimated with only a low number of scenes.

As a future work we can improve on estimates for the time needed to shoot a ray using yet unbuilt hierarchical data structure and the time to build this data structure for example by using other statistical characteristics of the scene. This could provide more accurate estimate for the critical point.

					Random rays			Path tracing		
Scene	Primitives	σ	T_B^G	T_B^H	t_T^G	t_T^H	R_C	t_T^G	t_T^H	R_C
boxes	528	7.46	2	12	5.0	4.1	11,669	0.9	1.4	0
interior dance	1,990	10.84	3	50	5.0	4.2	55,328	0.7	1.3	0
cornell	2,450	8.73	3	42	7.9	5.5	15,773	3.0	3.0	38,177
sphere	2,880	4.85	4	74	7.7	7.9	0	0.6	1.6	0
teapot	3,080	7.50	32	73	6.8	6.0	78,320	0.8	1.6	0
interior 3	3,412	10.75	4	83	4.6	3.8	101,888	0.9	1.0	124,367
phone	7,716	15.85	6	207	3.3	2.9	441,922	3.3	0.9	53,984
spheres	31,460	7.70	22	696	11.2	8.1	211,736	2.0	4.9	0
pills	32,606	12.90	21	802	7.1	3.6	221,538	2.3	1.8	593,553
teapots	52,360	12.43	29	1,215	13.4	6.8	172,843	4.3	5.4	1,620,297
building	54,490	42.19	21	995	6.3	3.1	304,414	12.8	1.8	63,055
knot	56,448	6.97	35	1,300	20.2	12.1	147,560	1.3	3.3	0
bunny	69,473	7.48	35	1,639	33.1	10.8	69,587	1.3	2.5	0
interior japan	72,310	43.03	36	1,241	4.6	4.8	0	3.1	1.9	515,863
sphere-high	87,120	6.68	48	1,751	23.5	13.5	161,984	1.1	2.8	0
case	131,228	24.93	51	2,872	14.0	4.2	293,541	5.3	3.0	638,451
hunger	141,143	64.21	47	2,586	15.9	5.6	250,041	68.2	3.1	236,153
interior deloix	149,090	16.02	67	3,928	22.9	6.6	231,927	3.0	1.9	1,704,493
camel	178,102	72.49	73	3,734	23.0	6.0	216,668	3.3	2.5	1,734,172
sockets	187,330	15.93	102	5,229	22.8	8.9	358,906	1.2	2.8	0
conference	190,947	25.62	97	4,428	19.4	7.1	348,928	2.8	2.7	3,268,361
chess	249,608	12.65	103	5,799	16.4	5.1	543,065	1.4	1.1	6,037,314
phone-high	318,756	30.72	145	8,163	6.7	2.9	2,262,962	7.5	1.5	856,324
pills-high	590,626	15.23	283	18,694	17.7	4.3	1,401,905	4.6	2.9	5,082,077
a10	1,070,671	60.79	403	29,921	25.0	5.9	1,544,378	2.6	3.1	52,604,130
hunger-high	1,418,560	136.61	384	27,690	44.2	5.6	727,590	156.3	3.9	27,356
case-high	1,614,006	36.86	501	37,224	39.6	4.5	1,108,126	11.1	4.5	3,101,617
sockets-high	1,658,432	16.80	730	45,842	41.0	11.0	1,516,267	2.7	4.1	0

Table 1: Measurements for all tested scenes for both ray generation schemes. T_B^G and T_B^H are uniform grid and kd-tree build times. t_T^G and t_T^H are uniform grid and kd-tree per-ray traversal times. Build times are in milliseconds and traversal times are in microseconds. R_C (the critical point) is the exact number of rays for which the uniform grid is equal in performance of the kd-tree.

ACKNOWLEDGEMENTS

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under research programs MSM 6840770014, LC-06008 (Center for Computer Graphics), MEB-060906 (Kontakt OE/CZ), the Grant Agency of the Czech Republic under research program P202/11/1883, and the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/289/OHK3/3T/13.

REFERENCES

[App68] Appel, A. Some techniques for shading machine renderings of solids. In *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45, New York, NY, USA, 1968. ACM.

[Ben75] Bentley, J.L. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.

[Cha04] Chang, A. Y.-H. *Theoretical and Experimental Aspects of Ray Shooting*. PhD thesis, Polytechnic University, USA, 2004.

[Cle88] Cleary, J.G. and Wyvill, G. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[Fuj86] Fujimoto, A., Tanaka, T., and Iwata, K. ARTS:

Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[Gla89] Glassner, Andrew S. An introduction to ray tracing, Academic Press Ltd.,London, UK, 1989

[Hav00a] Havran, V. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[Hav00b] Havran, V., Prikryl, J., and Purgathofer, W. Statistical comparison of ray-shooting efficiency schemes. Technical Report TR-186-2-00-14, Vienna University of Technology, May 2000.

[Hav07] Havran, V. About the relation between spatial subdivisions and object hierarchies used in ray tracing. In Mateu Sbert, editor, *23rd Spring Conference on Computer Graphics (SCCG 2007)*, pages 55–60, Budmerice, Slovakia, May 2007. ACM.

[Ize07] Ize, T., Shirley, P., and Parker, S. Grid creation strategies for efficient ray tracing. In *RT '07: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, pages 27–32, Washington, DC, USA, 2007. IEEE Computer Society.

[Kal09] Kalojanov, J., and Slusallek, P. A parallel algorithm for construction of uniform grids. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pages 23–28, New York, NY, USA, 2009. ACM.

[Kay86] Kay, T.L. and Kajiya, J.T. Ray tracing complex scenes. In David C. Evans and Rusell J. Athay, editors, *SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, August 1986.

[Kli94] Klimaszewski, K.S. *Faster ray tracing using adaptive grids and area sampling*. PhD thesis, Brigham Young University, dec 1994.

[MF99] Müller G. and Fellner D.W. Hybrid Scene Structuring with Application to Ray Tracing. In *Proceedings. of Intl. Conf. on Visual Computing ICVC '99*, 1999.

[SKH02] Szirmay-Kalos, L., Havran, V., Balázs, B. and Szécsi, L. On the efficiency of ray-shooting acceleration schemes. In *Proceedings of the 18th Spring Conference on Computer Graphics (SCCG 2002)*, pages 89–98, Budmerice, Slovakia, May 2002.

[Wal04] Wald, I. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[Wal06a] Wald, I. and Havran, V. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proc. IEEE Symposium on Interactive Ray Tracing 2006*, pages 61–69, September 2006.

[Wal06b] Wald, I., Ize, T., Kensler, A., Knoll, A. and Parker, S.G. Ray Tracing Animated Scenes using Coherent Grid Traversal. *ACM Transactions on Graphics*, pages 485–493, 2006. (Proceedings of ACM SIGGRAPH 2006).