On the Efficiency of Ray-shooting Acceleration Schemes

László Szirmay-Kalos, Budapest University of Technology*
Vlastimil Havran, Max-Planck-Institut für Informatik†
Benedek Balázs, Budapest University of Technology
László Szécsi, Budapest University of Technology

Abstract

This paper examines the efficiency of different ray-shooting acceleration schemes, including the uniform space subdivision, octree and kd-tree. We use simple computational models, which assume that the objects are uniformly distributed in space. The efficiency is characterized by two measures, including the expected number of ray-object intersections needed to identify the firstly intersected object, and the expected number of steps on the space partitioning data structure. We can come to the interesting conclusion that these numbers are constant and are independent of the number of objects in the scene. The number of intersections is determined by how well the cells of the partitioning data structure enclose the objects. Such analysis helps to understand why kd-tree is better than octree and uniform space subdivision and provides hints to improve their implementation.

Keywords: Ray-tracing, uniform subdivision, octree, kd-tree, performance analysis.

1 Introduction

Ray-shooting is a fundamental operation in ray-casting, recursive ray-tracing and random walk global illumination algorithms, thus its efficiency is a critical factor in developing fast rendering algorithms. In order to solve ray shooting efficiently, a space partitioning data structure is built in the preprocessing step. During execution, when a ray is given, this data structure is queried to determine those objects for which ray-intersection calculation should be performed.

According to a generally accepted criterion, a "good" ray-shooting algorithm runs in sub-linear time after sub-quadratic preprocessing and uses linear memory space. Thus instead of implementing the algorithms invented in computational geometry, computer graphics practitioners prefer heuristic ray-shooting speed-up techniques, including, for example,

- uniform space subdivision (also called regular grids) [5][1],
- octree [6][1],
- BSP or kd-tree [8],
- ray coherence methods [15][9],
- ray classification [6][1],
- Voronoi diagram based space partitioning [13].

The first three algorithms is particularly important and popular, therefore we shall concentrate on them.

These algorithms try to minimize ray-object intersection calculations by building a space partitioning data structure, which has two purposes. On the one hand, this data structure can select only those objects that are in the direction of the ray and can ignore those that are not in this direction and thus can have no intersection with it. Of course, this selection cannot be optimal, but we can be conservative. Thus the data structure usually provides more objects, including even those for which no intersection occurs, but we can be sure that the data structure will report all objects, for which intersection happens. From another point of view, it means that the space partition selected for a given ray encapsulates the points of object locations that can be intersected, but is usually larger than that.

The other main feature of these algorithms is that they sort the objects along the ray. It means that candidate objects that are in the ray direction are reported in such an order that if we find an intersection, then we can stop the calculations, because all other intersections are surely behind the found one. This objective is also approximately met in practical situations. In order to control the size of the data structure, cells in the data structure are often allowed to store more than one objects, for which no sorting is available. In the subsequent analysis, we shall be optimistic and assume that there are no space limits, and such ordering can be provided. Having carried out the theoretical analysis, we shall intuitively consider what happens when the limitations of the memory space should also be taken into account.

The efficiency of a ray-shooting algorithm depends on the wellness of this encapsulation and also the speed of traversing the data structure.

The approach proposed in this paper uses a theoretical, probabilistic model of the scene, thus in this sense, it corresponds to average case complexity analysis. However,

*szirmay@iit.bme.hu
†havran@mpi-sb.mpg.de

instead of calculating complexity measures, we intend to provide real numbers that well characterize the algorithms. The running time T of a heuristic ray-shooting algorithm can be expressed in the following form:

$$T = T_o + N_I \cdot T_I + N_S \cdot T_S,$$

where T_o is the time needed to find the cell of the starting point of the ray, N_I is the number of ray-object intersections needed to find the closest intersection, T_I is the time of a single ray-object intersection calculation, N_S is the number of cells that are visited and T_S is the time needed to step from one cell to the next one. In this paper, we are going to analyze the number of intersection calculations N_I required to identify the first intersection and the number of steps N_S on the data structure needed to find the intersection. Having computed these numbers and having measured the times T_I, T_o, and T_S on a particular computer, the running times can be presented in a numeric form.

In the following sections the previous work is surveyed first, then a probabilistic scene model is proposed. Using this model, the expected number of intersections and the expected number of visited cells are calculated for uniform space subdivision, octree and kd-tree. Finally, we evaluate the results and validate them with simulation data.

2 Previous work

The efficiency of the ray-shooting algorithms was first investigated rigorously in the context of worst-case complexity and it was concluded that in the optimal case logarithmic algorithms can be obtained with at least $\mathcal{O}(n^{4+\epsilon})$ storage and preprocessing time [3, 17]. Clearly, such an algorithm has prohibitive memory and preprocessing time requirements, thus it is not a surprise that nobody uses these algorithms in practice. Heuristic ray-shooting algorithms seemed to be better, and their superior performance has been addressed intuitively and using simple complexity models [7, 2]. Later, Márton [14, 17] showed theoretically why heuristic ray-shooting acceleration algorithms perform much better, even if their worst-case complexity is similar to that of the naive case. This analysis used average case rather than worst-case complexity measures and resulted in the statement that heuristic ray-shooting algorithms have constant (uniform subdivision, Voronoi partitioning) or at most logarithmic (octree) time complexity. Sbert et al. emphasized the applicability of the results of integral geometry in such calculations [10]. Shirley et al. aimed at the computation of the Monte-Carlo radiosity algorithm and as a subtask of the problem, analyzed the uniform subdivision [16, 18]. This approach was too pessimistic, since it did not take into account that when an intersection is found, the data-structure traversal can be stopped, thus resulted in $\mathcal{O}(n^{1/3})$ time complexity. In [8] a practice oriented efficiency model has been introduced, which includes the measured properties of the computer and the scene, and determines even the scalar factor of the running time, which is missing in all complexity approaches.

3 Probabilistic scene model

We noted that due to the approximate encapsulation, the algorithm may try to compute more than one ray-object intersections until a real intersection is found. These trials will fail until a valid intersection is found. When the valid intersection is obtained, the search terminates. This means that the determination of the first intersection requires certain ray-object intersection calculations, whose number depends on the accuracy of the encapsulation.

In order to consider this problem formally, we will apply simple theoretical models. Thus the analysis will describe the majority of the cases. This corresponds to the fact that the authors of the papers proposing such algorithms usually intuitively justify why these methods are expected to run faster for "normal scenes" than the naive implementation and demonstrate this statement by simulation.

To carry out the average case analysis, the scene model, i.e. the probability distribution of the possible input configurations must be known. In practical situations, this probability distribution is not available, therefore it must be estimated, that is, a model of the configuration space must be established. Such a model cannot be very complicated in order to allow the calculation of the expectation of the computation time.

A possible, but also justifiable input configuration model for ray-shooting is the following [17]:

1. The object space consists of spheres of the same radius r.

2. The sphere centers are uniformly distributed in space.

Since we are interested in the asymptotic behavior when the number of objects is really high, uniform distribution in a finite space would not be feasible. On the other hand, the boundary of the space would pose problems. Thus instead of dealing with a finite object space, the space should also be expanded as the number of objects grows to sustain constant average spatial object density. This is a classical method in probability theory, and its known result is the Poisson point process [12, 11]. A Poisson point process $N(A)$ counts the number of points in subsets A of S in a way that

- $N(A)$ follows Poisson distribution of parameter $\rho V(A)$ where ρ is a positive constant called "intensity" and $V(A)$ is the volume of A, thus the probability that A contains exactly k points is

  $$\Pr\{N(A) = k\} = \frac{(\rho V(A))^k}{k!} e^{-\rho V(A)},$$

  and the expected number of points in volume $V(A)$ is $\rho V(A)$;

- for disjoint $A_1, A_2, \ldots, A_n$ sets random variables $N(A_1), N(A_2), \ldots, N(A_n)$ are independent.

The Poisson point process will be the basis of our input configuration model, which works with the following assumptions:

1. The object space consists of spheres of the same radius r.
2. The sphere centers are the realizations of a Poisson point process of intensity ρ.

Having constructed a model of the configuration space, we can start the analysis of the candidate algorithms.

4 Calculation of the number of intersections

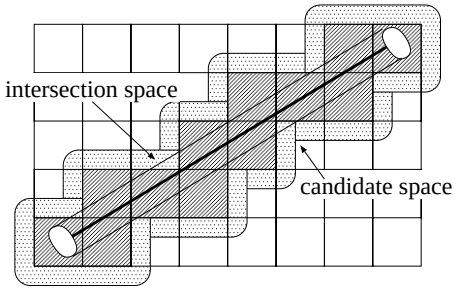


Figure 1: Encapsulation of the intersection space by the cells of the data-structure in a uniform subdivision scheme

Looking at figure 1 we can see a ray that passes through certain cells of the space partitioning data-structure. The collection of those sphere centers where the sphere would have an intersection with a cell is called the *candidate space* associated with this cell. The union of the candidate spaces enclose the visited cells but are greater than them by the radius of the spheres since a sphere is checked for intersection even if a small part of it is included by the cell. Formally, the candidate space is the Minkowski sum by a sphere of radius r over the cells that have intersection with the ray.

Only those spheres of radius r can have intersection with the ray, whose centers are in a cylinder of radius r around the ray. This cylinder is called the *intersection space* (figure 1). More precisely, the intersection space also includes two half spheres at the bottom and at the top of the cylinder, but these will be ignored.

As the ray-shooting algorithm traverses the data structure, it examines each cell that is intersected by the ray. If the cell is empty, then the algorithm does nothing. If the cell is not empty, then it contains, at least partially, a sphere, which is tried to be intersected. This intersection succeeds if the center of the sphere is inside the intersection space and fails if it is outside.

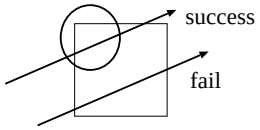


Figure 2: Probability of success

Note that the algorithm should try to intersect objects that are in the candidate space, but this intersection will be successful only if the object is also contained by the intersection space. The probability of the success s is the ratio of the projected areas of the intersection space and the candidate space associated with this cell. In general, the success probability depends on the size of the cell, and can thus vary along the ray. As we shall see in the analyzed algorithms, we can still assume that the non-empty cells have the same size, thus we shall work with constant success probability.

From the probability of the successful intersection in a non-empty cell, the probability that the intersection is found in the first, second, etc. cells can also be computed. Assuming statistical independence, the probabilities that the first, second, third, etc. intersection is the first successful intersection are s, $(1-s)s$, $(1-s)^2 s$, etc., respectively. This distribution is the geometric distribution and its expected value is $1/s$. Consequently, the expected value of the ray-object intersections for an acceleration scheme is:

$$E[N_I] = \frac{1}{s}.$$

In order to determine success probability s, the intersection and candidate spaces should be compared in different ray-shooting acceleration schemes. In the following subsections, the uniform subdivision, the octree and the kd-tree are examined.

4.1 Number of intersections in the uniform subdivision scheme

The projected size of the intersection space is obviously $r^2\pi$ since only those spheres may intersect the ray, which are not farther than r, thus the projection of their center is inside a circle of radius r.

For uniform subdivision the cells have the same edge size a, thus the volume of center of spheres intersecting the cell is a cube with rounded corners of edge size $a+2r$. If the ray is parallel to one of the sides, then the projected size of the candidate space is $a^2 + 4ar + r^2\pi$ (left of figure 3). The other extreme case happens when the ray is parallel to the diagonal of the cubic cell (right of figure 3), where the projection is a rounded hexagon having area $\sqrt{3}a^2 + 6ar + r^2\pi$. The success probability for the uniform subdivision is then:

$$\frac{r^2\pi}{\sqrt{3}a^2 + 6ar + r^2\pi} \leq s_{uniform} \leq \frac{r^2\pi}{a^2 + 4ar + r^2\pi}.$$

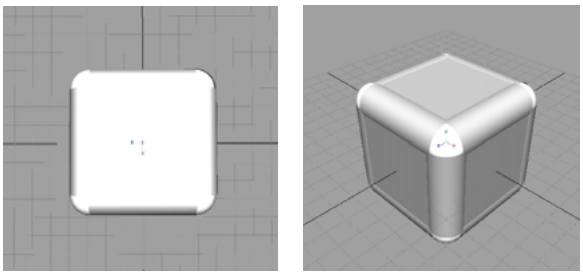


Figure 3: The candidate space from two viewpoints

Thus the average number of intersection calculations is:

$$\frac{1}{\pi}\left(\frac{a}{r}\right)^2+\frac{4}{\pi}\frac{a}{r}+1 \leq E[N_I^{uniform}] \leq \frac{\sqrt{3}}{\pi}\left(\frac{a}{r}\right)^2+\frac{6}{\pi}\frac{a}{r}+1.$$

Note that the number of intersections depend on refinement level $d = a/r$, which expresses the relative sizes of the cells and the objects. This is highly intuitive since the number of intersections should not change when a new unit is defined for the underlying coordinate system.

Another interesting observation is that the expected number of intersections is a finite constant even when we assumed infinitely many spheres. From another point of view, it means that it does not depend on the number of objects n, thus it is in complexity class $\mathcal{O}(1)$. This corresponds to the previous results published in [14, 17].

4.2 Analysis of the octree

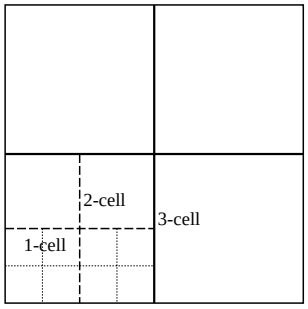


Figure 4: Octree space partitioning

For octrees, the cells can be bigger that the minimum cell, which would result in smaller success probability, which increases the number of intersection computations. This means that from the point of view of minimizing the number of calculated intersections, the non-empty cells of the octree data structure is worth refining until to their minimum size. If we use this optimization, the non-empty cells of the octree will also have similar size a. Since the success probability refers only to non-empty cells, this means that the success probability of octree is the same as the success probability of the uniform subdivision if the cells are refined to their minimum size:

$$\frac{1}{\pi}\left(\frac{a}{r}\right)^2+\frac{4}{\pi}\frac{a}{r}+1 \leq E[N_I^{octree}] \leq \frac{\sqrt{3}}{\pi}\left(\frac{a}{r}\right)^2+\frac{6}{\pi}\cdot\frac{a}{r}+1.$$

Note that this means that the octree is not better than the uniform subdivision in terms of the number of calculated intersections. However, it can save memory and steps from one cell to the next since the empty cells can be bigger. On the other hand, the determination of the next cell is more complex for an octree than for the uniform subdivision, thus there is no clear winner of this game.

4.3 Analysis of the kd-tree

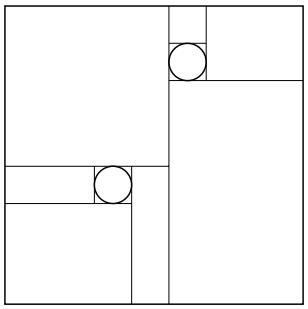


Figure 5: Kd-tree space partitioning with cutting off the empty space

Following the same argument we used for octrees, kd-trees can also be optimized for the number of the calculated intersections if the empty space is cut off from the objects. Since in our model all objects have the same size, this also results in the same size for non-empty cells: $a = 2r$. Note that this assumption implicitly involves that the objects can always be separated. This can be accepted if the scene is not very dense. It means that the expected number of spheres close to a give sphere is small (e.g. $\rho 4r^3\pi/3$ is small). When calculating the success probability, we have to take into account that unlike in uniform subdivision and in octree, where the location of the cell boundaries was independent of the spheres, in kd-trees the cell boundaries are set exactly at the extrema of the objects. It means that the candidate space is the cell itself and should not be extended by the size of the objects. If the ray is parallel to one of the sides, then the projected size of the candidate space is a^2. On the other hand, when the ray is parallel to the diagonal of the cubic cell, the projected size of the candidate space is a hexagon having area $a^2\sqrt{3}$. Using the $a = 2r$ substitution, the success probability is

$$\frac{\pi}{4\sqrt{3}} \leq s_{kdtree} \leq \frac{\pi}{4}.$$

In the case of the kd-tree, the cells enclose the objects thus we do not have problems with those objects that only partially contained by the cell. This fact together with the

recognition that both the cell and the object are convex allow the application of a fundamental result of integral geometry, which states that the conditional probability of intersection a convex enclosed object, given that it is intersected by the convex cell equals to the relative surface areas, thus we can also obtain an exact value instead of the bounds:

$$s_{kdtree} = \frac{4r^2\pi}{6a^2} = \frac{\pi}{6}.$$

The expected numbef of intersection calculations is then:

$$E[N_I^{kdtree}] = \frac{6}{\pi}.$$

5 Calculation of the number of cell steps

The other fundamental parameter of a heuristic ray-shooting algorithm is the expected number of cells N_S that should be visited until a valid intersection is found. In the following analysis the conditional expected value theorem will be used. An appropriate condition is the length of the ray until the first intersection is found. Using its probability density $p_{t^*}(t)$ as a condition, the expected number visited cells N_S can be obtained in the following form:

$$E[N_S] = \int_0^\infty E[N_S|t^* = t] \cdot p_{t^*}(t)\, dt,$$

where t^* is the length of the ray and p_{t^*} is its probability density.

Since the intersection space is a cylinder if we ignore the half spheres around the beginning and the end, its total volume is $r^2\pi t$. Thus the probability that intersection occurs before t is:

$$\Pr\{t^* < t\} = 1 - e^{-\rho r^2 \pi t}.$$

Note that this function is the cumulative probability distribution function of t^*. The probability density can be obtained as its derivative, thus we obtain:

$$p_{t^*}(t) = \rho r^2 \pi \cdot e^{-\rho r^2 \pi t}.$$

The expected length of the ray is then:

$$E[t^*] = \int_0^\infty t \cdot \rho r^2 \pi \cdot e^{-\rho r^2 \pi t}\, dt = \frac{1}{\rho r^2 \pi}. \qquad (1)$$

The conditional expectation depends on the examined acceleration scheme, thus it will be determined separately for the discussed methods. In order to simplify the analysis, we shall assume that the ray is parallel to one of the main directions of the space partitioning scheme. Note that when the ray is not parallel, then it may intersect more objects but the length traveled in a single cell also increases. Thus the results obtained with this assumption will well approximate the properties of the general case as well.

5.1 Number of visited cells in a uniform grid

The analysis of the regular grid is quite simple. Since all cells have the same edge size a, the number of cells intersected by a t long ray is t/a. The expected number of visited cells is then:

$$E[N_S] = \int_0^\infty \frac{t}{a} \cdot \rho r^2 \pi \cdot e^{-\rho r^2 \pi t}\, dt = \frac{1}{a\rho r^2 \pi}.$$

Note that this number is also constant even for infinite number of objects, thus the number of cell steps is also in $\mathcal{O}(1)$. We have to recall that a basic assumption of the analysis was that a cell may contain at least one object. In order to guarantee that, the number of cells should be at least linear in terms of the number of objects.

5.2 Number of visited cells in an octree

Assume that the ray is contained by an octree node of 2^{3L} of elementary cells of size a^3, thus $t \approx 2^L \cdot a$.

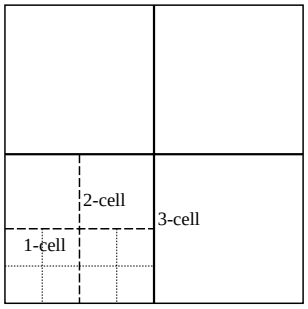


Figure 6: Octree as a collection of 1-cells, 2-cells, etc. k-cells

Looking at figure 6, we can recognize that if the octree is refined to the maximum level, then it will contain 2^{3L} number of elementary cells of edge size a, which corresponds to $2^{3(L-1)}$ number of nodes of edge size $2a$, and finally a single node of edge size $2^L a$. When a node is decomposed to 8 children, new boundaries are introduced in the space partitioning data structure. Let us call the boundaries introduced on that level which works with nodes of edge size $2^i a$ as the i-boundaries. If the ray is horizontal and the octree is fully refined, then it will intersect one L-boundary, two $L-1$-boundaries, and 2^{L-1} number of 1-boundaries.

However, the ray should not necessarily intersect all elementary cells and thus all boundaries, since the algorithm may stop refining the cells at higher levels. A cell is not decomposed further if it is empty. It also means that an i-boundary does not exist, and therefore the crossing is saved, if the embedding node of volume $2^{3i}a$ is empty. A cell is empty if there are no sphere centers in the set of

points being not farther than r from the cell. The volume of such set is

$$V_i = (2^i a)^3 + 6 \cdot (2^i a)^2 r + 3\pi(2^i a) r^2 + \frac{4r^3\pi}{3}.$$

The probability of the event of V_i being empty is

$$\Pr\{i\text{-boundary is saved}\} = e^{-\rho V_i}. \quad (2)$$

The number of visited cells or crossed boundaries equal to the number of the elementary cells 2^L minus the number of saved and thus not existing boundaries. Let us denote the indicator function of the event that the jth boundary does not exist by I_j (I_j is 1 if the jth boundary is saved and 0 if it exists). The number of crossed boundaries is then:

$$N_S^{octree} = 2^L - \sum_{j=1}^{L} I_j.$$

When the expected value is computed, we can take advantage that the expected value of an indicator function equals to the probability of the event it indicates:

$$E[N_S^{octree} | t^* = t] = 2^L - E\left[\sum_{j=1}^{L} I_j\right] =$$

$$2^L - \sum_{j=1}^{L} \Pr\{j\text{th boundary is saved}\}.$$

Considering that a horizontal ray encounters one L-boundary, two $L-1$-boundaries, and 2^{L-1} number of 1-boundaries, and substituting equation 2, we can obtain:

$$E[N_S | t^* = t] = 2^L \cdot \left(1 - \sum_{j=1}^{L} \frac{e^{-\rho V_j}}{2^j}\right).$$

The terms of the sum decrease rather quickly, thus we obtain a lower bound for the sum keeping only the first term:

$$\sum_{j=1}^{L} \frac{e^{-\rho V_j}}{2^j} \geq \frac{e^{-\rho V_1}}{2}.$$

On the other hand, V_j is an increasing series, thus we obtain an upper bound if the exponentials of all terms is replaced by the first exponential:

$$\sum_{j=1}^{L} \frac{e^{-\rho V_j}}{2^j} \leq \sum_{j=1}^{L} \frac{e^{-\rho V_1}}{2^j} \leq e^{-\rho V_1}.$$

Using these bounds and substituting $2^L = t/a$, we get the following final result:

$$\frac{t}{a}(1 - e^{-\rho V_1}) \leq E[N_S^{octree} | t^* = t] \leq \frac{t}{a}(1 - \frac{e^{-\rho V_1}}{2}).$$

The unconditional expected number of visited cells is:

$$\frac{1}{a\rho r^2\pi}(1 - e^{-\rho V_1}) \leq E[N_S^{octree}] \leq \frac{1}{a\rho r^2\pi}(1 - \frac{e^{-\rho V_1}}{2}).$$

5.3 Number of visited cells in a kd-tree

Unfortunately, we could not obtain bounds for the expected number of visited cells in a kd-tree. However, a rough approximation was possible.

Suppose that we are having a cell of volume V, which is further decomposed by the kd-tree method. The construction of kd-trees is controlled by the distribution of the objects. The subdivision planes are placed at the maximum or minimum points of objects along a given coordinate axis. It means that if the number of objects is n, then $6n$ planes are inserted. Note that each subdivision plane subdivides a cell into two, thus increases the number of cells by one. Consequently, the number of cells in V will be $6n + 1$. Taking advantage that the number of spheres in a cell follows Poisson distribution, the expected number of cells in V is the following:

$$\sum_{n=0}^{\infty} (6n + 1) \cdot \frac{(\rho V)^n}{n!} e^{-\rho V} = 6\rho V + 1.$$

It means that the average volume of a single cell is

$$\frac{V}{6\rho V + 1} \approx \frac{1}{6\rho}$$

if V is large enough to contain possibly many spheres. The cells are bricks and the kd-tree building algorithm tries to form these bricks to be close to the cubical shape. It means that the edge size of an average cell can be approximated by $1/(6\rho)^{1/3}$. The number of cells visited by a horizontal ray is the length of the ray divided by the edge size of the cells. According to equation 1, the average length of a ray is $1/(\rho r^2\pi)$, thus the average number of visited cells is:

$$E[N_S^{kdtree}] \approx \frac{(6\rho)^{1/3}}{\rho r^2 \pi}.$$

6 Evaluation and comparison of the three ray-shooting acceleration schemes

We are already concluded that the expected number of ray-object intersections and the expected number of visited cells are finite even for infinite number of objects, thus both of these functions are in $\mathcal{O}(1)$. Recall that the total time of ray-shooting is characterized by the following formula

$$T = T_o + N_I \cdot T_I + N_S \cdot T_S.$$

The intersection time T_I is constant and is independent of the particular scheme. For uniform subdivision, the initial location time T_o and the step time T_S are also constant, thus this algorithm is in $\mathcal{O}(1)$ altogether. For octree and kd-tree, the initial location requires a search in the data structure, which is usually started at the root of the tree and completed by the tree traversal. If there are n number

of objects, such traversal requires $\mathcal{O}(\log n)$ time. Stepping from one onto the next cell is also traced back to location. The ray is intersected with the boundary of the current cell, the ray-parameter is increased a little and the cell of the new point is searched in the data structure. In this case T_S is also in $\mathcal{O}(\log n)$. Summarizing, for very large number of objects, the uniform subdivision seems to be the best scheme. However, the difference between constant and logarithmic functions is not too significant if the number of objects is not extremely high. In such cases the decision can be made according to the studied variables N_I and N_S.

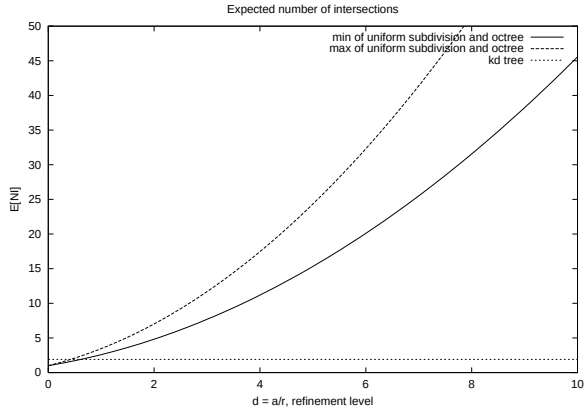


Figure 7: Expected number of ray-object intersection calculations

Concerning the expected number of intersection calculations, the first important observation is that the performance of an octree is not better then the performance of uniform subdivision. In fact, the expected values are the same if the non-empty octree cells are always refined to the minimum size. However, if the octree building is stopped when the cell contains only a single, or a few objects, then the number of calculated intersections increases. According to the theoretical results, the number of intersections is the reciprocal of the probability of a successful intersection in a non-empty cell. The uniform subdivision and the octree contain cell boundaries that are static and their location is independent of the objects. This independence makes the success probability low. The kd-tree, on the other hand, places the boundaries around the objects, especially if the empty space is cut off, thus it can result in much higher intersection probability. Figure 7 compares the number of calculated intersections for uniform subdivision (the curve of octree is identical to the curve of uniform subdivision) and for kd-tree. Since in our model, the minimum cell size in kd-tree corresponds to $d = 2$ refinement level, a fair comparison can be made for $d > 2$ values. Examining the situation at this value, we can see that the kd-tree needs at least 2–3 times less number of intersections. Another lesson learned from the analysis is that it is worth using higher refinement levels if the computation burden of a single intersection is high (note the quickly increasing function in figure 7). This trick can also be adapted in kd-trees, when not only the empty space is cut off, but even objects are cut in pieces in order to make the bounding more compact.

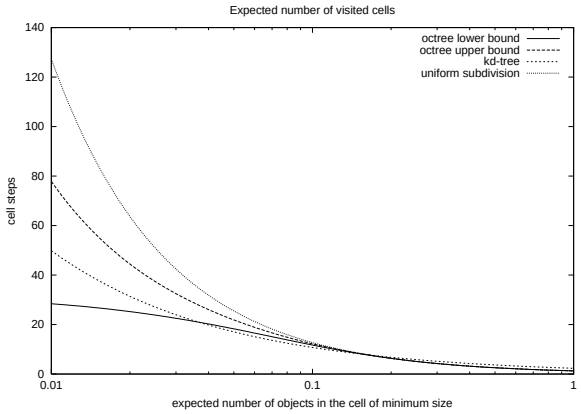
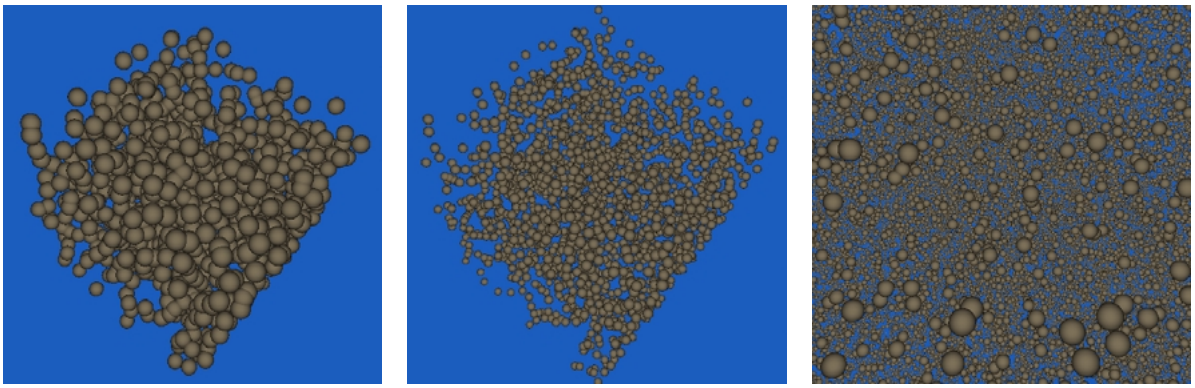


Figure 8: Number of visited cells for octree, kd-tree and uniform subdivision, assuming normal refinement ($a/r = 2$)

Figure 8 compares the number of visited cells for octree, kd-tree and for uniform subdivision. The graph has been plotted as a function of the expected number of sphere centers in a single cell of the uniform subdivision, i.e. the horizontal axis is ρa^3. Note that when the cell size of the octree is similar to the size of the objects, the octree and the kd-tree perform similarly.

When carrying out the analysis we made several assumptions on the objects and their distribution, which poses the question whether or not the obtained results can be used generally. Such assumptions include that the objects are similar spheres and the cells are of the same size. In global illumination algorithms, for instance, the surfaces are built of not too long triangles of roughly the same tessellation level and of different orientations. When the octree and the kd-tree are built, the check whether a cell includes a part of a triangle is carried out by examining whether the cell includes a part of the bounding box of this triangle. The bounding box, in turn can be well approximated by a sphere. On the other hand, when the real ray-triangle intersection is calculated, then instead of the bounding box (or sphere), the triangle is intersected. The probability that a ray intersects a triangle given that the ray intersected its bounding box is about 0.4 for not thin triangles. This value is independent of the data structure thus the ratio of the intersection numbers will be similar to the result obtained with spheres. When absolute data is needed, we have to multiply the values obtained with spheres by about 2–3.

We also supposed that the data structure is fine enough to contain at most one object in each cell. In practice, more than one objects may share a single cell, when all objects must be intersected. This corresponds to the sit-



$n = 1000$, $\rho = 0.05$ $n = 2000$, $\rho = 0.01$ $n = 100000$, $\rho = 0.001$

Figure 9: Theoretical test scenes

uation when the cost of a ray-object intersection calculation is multiplied by the number of objects sharing the cell. Thus the theoretical N_I parameter can still be used, having multiplied T_I by the maximum number of objects in a cell. The last critical assumption regards the uniform distribution of the objects. This may seem not very realistic assumption when we look at the scene from a larger perspective. However, adaptive space partitioning schemes, including both the octree and the kd-tree, step over the empty regions quickly and concentrate on the populated regions. On this low level, the objects seem to be randomly and uniformly distributed, thus the assumption on uniform random distribution becomes acceptable.

6.1 Simulation results

In order to validate the theoretical results we took several scenes and measured the analyzed parameters. When doing so, we could not use infinitely large spaces and infinitely many objects, but a finite space of volume V and a finite number of objects n are considered. The theoretical formulae contain the density of the objects ρ, which can be estimated as $\rho = n/V$.

First we considered scenes containing uniformly distributed spheres, which corresponds to the theoretical assumptions. In order to make enough space for spheres, we set density ρ and expanded the volume according to $V = n/\rho$. Since we constrained the maximum level of the octree, it means that the leaf cells in the octree and in the regular grid have edge size $a = (n/\rho)^{1/3}/2^6$, since the maximum level of subdivision was 6. The radius of spheres is $r = 1$, thus $d = a$. The number of objects and the density were in the ranges [1000, 100000] and in [0.002, 0.12], respectively, thus the refinement level d was in [0.3, 5]. The test scenes are shown in figure 9 and the measurement results in tables 1–5.

Recall that according to the theoretical results the average number of intersections should be similar for octree and uniform subdivision, and it is an increasing function of the refinement level. Due to the $(n/\rho)^{1/3}$ characteristics of the refinement level, we expect the results in the range of [2, 19], and increasing for larger number of objects and smaller densities. The results in tables 1 and 2 generally meet this, but are worse when the objects are large compared with the cells, and better when the objects are small and are not dense. The explanation of worse numbers for larger objects is that for inclusion test we used the bounding box of the sphere, which reduced the success probability. The reason of these numbers being smaller than

n	ρ					
	0.002	0.004	0.01	0.02	0.05	0.12
1000	5.03	5.5	5.9	5.9	5.7	5.5
2000	5.8	6.3	6.6	6.5	6.2	5.8
5000	7.4	7.5	7.4	7.0	6.5	6.0
10000	8.6	8.5	8.0	7.4	6.8	6.2
20000	9.8	9.4	8.5	7.8	7.1	6.4
50000	11.4	10.5	9.0	8.1	7.2	6.6
100000	12.5	11.1	9.3	8.3	7.3	6.7

Table 1: Average number of ray-object intersections per ray for uniform subdivision

n	ρ					
	0.002	0.004	0.01	0.02	0.05	0.12
1000	4.2	4.4	4.5	4.4	4.5	4.7
2000	5.0	5.1	5.0	4.8	4.7	4.8
5000	6.4	6.3	5.8	5.5	5.2	5.1
10000	7.6	7.2	6.5	6.0	5.6	5.3
20000	9.3	8.5	7.4	6.7	6.1	5.8
50000	12.2	10.9	9.1	8.1	7.2	6.6
100000	16.2	14.0	11.3	9.7	8.6	7.6

Table 2: Average ray-object intersections per ray for octree

n	ρ					
	0.002	0.004	0.01	0.02	0.05	0.12
1000	2.8	3.0	2.8	2.6	2.8	4.0
2000	3.6	3.4	3.0	2.7	2.4	3.1
5000	4.3	4.0	3.3	2.8	2.5	3.1
10000	4.9	4.5	3.5	2.9	2.6	3.1
20000	5.6	5.0	3.8	3.1	2.8	3.3
50000	6.5	5.4	3.9	3.1	2.7	2.8
100000	7.0	5.7	4.0	3.1	2.7	2.8

Table 3: Average ray-object intersections for kd-tree

expected for small number of rarely distributed objects is that in this case the ray can have no intersection at all with a not negligible probability, which is against the assumption of the theoretical analysis that we have infinitely many objects, and thus all rays intersect an object sooner or later.

For kd-trees, the average intersection number is expected to be smaller than for octree and uniform subdivision and is independent of density and the number of objects. Looking at table 3, we can recognize that the values are indeed close to the theoretically expected value of 2 and are fairly independent of the density, except for large n and small density values. This case corresponds to the situation when the objects are extremely small compared to the total volume covered by the kd-tree. Very small spaces are not cut off by the tree construction algorithm, which decreases the success probability and increases the number of intersections.

n	ρ					
	0.002	0.004	0.01	0.02	0.05	0.12
1000	30.1	29.2	27.4	24.7	22.3	19.4
2000	35.3	33.5	29.6	26.3	22.7	19.3
5000	44.1	40.1	33.3	27.9	23.1	18.7
10000	50.8	44.3	34.8	28.0	22.6	17.7
20000	56.3	47.3	35.1	27.3	21.4	16.3
50000	62.1	49.0	33.7	25.1	19.1	14.3
100000	61.8	46.4	30.6	22.3	16.8	12.6

Table 4: Average traversal steps per ray for octree

The numbers of visited cells meets out theoretical expectations stating that the octree and kd-tree behave similarly.

Having carried out the analysis with spheres, we checked the validity of the results for scenes consisting of random triangles instead of random spheres. We concluded that when the density of the random triangles have been set in the range of $[10^{-4}, 3 \cdot 10^{-4}]$ and the size of the triangles in the range of [15, 30], then the number of intersections for a kd-tree remained in the range of [5, 7]. Thus the intersection number is independent of the density and of the size of the objects as expected by the theoretical

n	ρ					
	0.002	0.004	0.01	0.02	0.05	0.12
1000	26.2	23.8	21.4	19.8	18.3	17.5
2000	31.4	28.8	25.0	22.1	20.1	18.1
5000	40.5	35.7	29.5	25.3	22.2	19.7
10000	47.3	40.4	32.2	27.3	23.6	20.7
20000	54.7	45.1	34.7	28.9	24.8	21.7
50000	64.0	50.9	37.9	31.1	26.5	22.6
100000	70.6	54.6	39.7	32.4	27.5	23.5

Table 5: Average traversal steps per ray for kd-tree

results.

Scene	octree	kd-tree
Cornell box	9.1	5
Chickens	10.3	6.3
Kitchen	12.0	6.7

Table 6: Number of intersection N_I for real scenes

Scene	octree	kd-tree
Cornell box	82	9
Chickens	94	15.7
Kitchen	71	13.2

Table 7: Number of visited cells for real scenes

At the final stage of the validation, realistic scenes have also been used. To allow some realistic distribution of rays, we used bi-directional path tracing. Tables 6 and 7 include the numbers of intersections and cells steps for three scenes shown in figure 10. The tessellation level has been set in a way that the average size of triangles is comparable with the size of an elementary cell of the octree (this corresponds to $d = 2$ in figure 7). Note that according to the theoretical expectations, the numbers of intersections in an octree are roughly twice the numbers of intersections in a kd-tree. The cell steps, on the other hand, are similar, as we have anticipated that.

Conclusions

This paper computed the average number of intersections and visited cells needed to find the firstly intersected object in various ray-tracing acceleration schemes, including uniform subdivision, octree and kd-tree. These numbers are constant even for infinitely many objects and infinitely large space. We concluded that due to the adaptive encapsulation property of kd-trees, they need significantly less ray-object intersection calculations. Concerning the number of visited cells, the octree and the kd-tree have similar performance if the cell size is comparable to the size of the

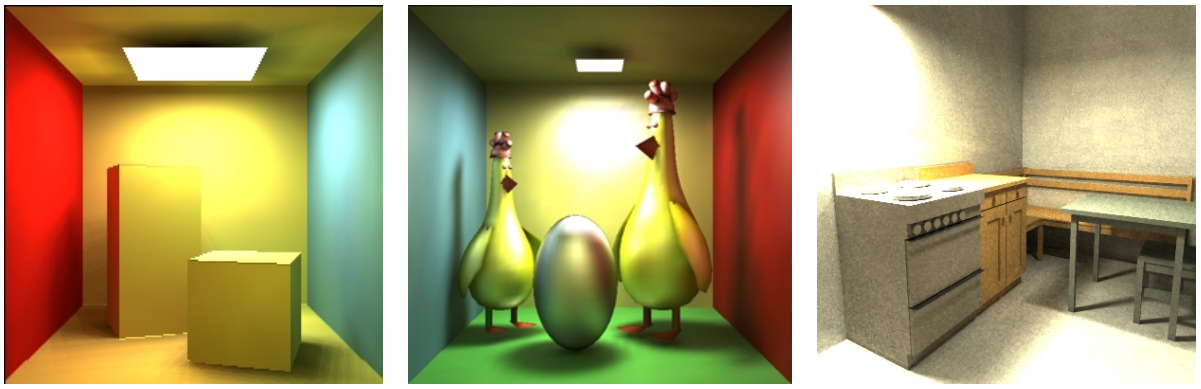


Figure 10: Practical test scenes

objects. For rarely populated scenes, both the kd-tree and the octree are better than the uniform subdivision.

The theoretical analysis used some idealistic assumptions thus the results are usually worse in practice. Such analysis helps to understand the behavior of ray-shooting acceleration algorithms, and the comparisons of the theoretical and measured values can also be used to tune these data structures. We demonstrated that the currently available program is not too far from the ideal performance, but could also find improvement possibilities. Another application would be to use the data to find optimal compromises for the different phases of ray-shooting. For example, the cell size of a uniform subdivision or an octree affects the preprocessing time, the number of intersections and the number of cell steps. From the point of view of the number of intersections, the cell size should be small. On the other hand, the minimization of the preprocessing time and the number of cell steps would require large cells. Knowing the density of the objects and the preprocessing, intersection and step times, an optimal compromise can be found between these contradicting criteria.

References

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In Andrew S. Glassner, editor, *An Introduction to Ray Tracing*, pages 201–262. Academic Press, London, 1989.

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

[3] M. de Berg. *Efficient Algorithms for Ray Shooting and Hidden Surface Removal*. PhD thesis, Rijksuniversiteit te Utrecht, The Nederlands, 1992.

[4] O. Devillers. The macro-regions: an efficient space subdivision structure for ray tracing. In *Eurographics '89*, pages 27–38, 1989.

[5] A. Fujimoto, T. Takayuki, and I. Kansei. Arts: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

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

[7] A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press, London, 1989.

[8] V. Havran. *Heuristic Ray Shooting Algorithms*. Czech Technical University, Ph.D. dissertation, 2001.

[9] T. Horváth, P. Márton, G. Risztics, and L. Szirmay-Kalos. Ray coherence between sphere and a convex polyhedron. *Computer Graphics Forum*, 2(2):163–172, 1992.

[10] A. Iones, S Zhukov, A. Krupkin, and M. Sbert. Answering to "what is the criterion?" questions using integral geometry tools. Technical report, Eurographics 2001, Tutorial notes, 2001.

[11] S. Karlin and M. T. Taylor. *A First Course in Stochastic Processes*. Academic Press, New York, 1975.

[12] J. Lamperti. *Stochastic Processes*. Springer-Verlag, 1972.

[13] G. Márton. Acceleration of ray tracing via voronoi-diagrams. In Alan W. Paeth, editor, *Graphics Gems V*, pages 268–284. Academic Press, Boston, 1995.

[14] G. Márton. *Stochastic Analysis of Ray Tracing Algorithms*. PhD thesis, Department of Process Control, Technical University of Budapest, Budapest, Hungary, 1995.

[15] M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. L. Kunii, editor, *Computer Graphics 1987. Proc. CG International '87*, pages 303–314, 1987.

[16] P. Shirley. Time complexity of Monte-Carlo radiosity. In *Eurographics '91*, pages 459–466. Elsevier Science Publishers, 1991.

[17] L. Szirmay-Kalos and G. Márton. Worst-case versus average-case complexity of ray-shooting. *Journal of Computing*, 61(2):103–133, 1998.

[18] B. Walter and P. Shirley. Cost analysis of a monte-carlo radiosity algorithm. Technical report, Program of Computer Graphics, Cornell University, 1995. Technical report PCG-95-3.