Octree-R: An Adaptive Octree for Efficient Ray Tracing

Kyu-Young Whang, *Senior Member, IEEE*, Ju-Won Song, Ji-Woong Chang, Ji-Yun Kim, Wan-Sup Cho, Chong-Mok Park, and Il-Yeol Song

***Abstract*—Ray tracing requires many ray-obect intersection tests. A way of reducing the number of ray-object intersection tests is to subdivide the space occupied by objects into many nonoverlapping subregions, called *voxels*, and to construct an octree for the subdivided space. In this paper, we propose the *Octree-R*, an octree-variant data structure for efficient ray tracing. The algorithm for constructing the Octree-R first estimates the number of ray-object intersection tests. Then, it partitions the space along the plane that minimizes the estimated number of ray-object intersection tests. We present the results of experiments for verifying the effectiveness of the Octree-R. In the experiment, the Octree-R provides a 4% to 47% performance gain over the conventional octree. The result shows the more skewed the object distribution (as is typical for real data), the more performance gain the Octree-R achieves.**

***Index Terms*—Ray tracing octree, Octree-R, space subdivision.**

I. Introduction

Ray tracing is a technique for rendering pictures from a three-dimensional model by following the paths of simulated light rays through the scene. One of the most serious problems of ray tracing is that it requires a relatively large amount of computation time for it must perform the ray-object intersection test for each ray and each object in the scene. Techniques using the bounding volume and space subdivision have been two major approaches for improving the efficiency of ray tracing [1], [3].

A bounding volume is a volume of a simple shape that surrounds a given object. Using the idea of the bounding volume simplifies the examination of a ray-object intersection. That is, for the examination of a ray-object intersection, the bounding volume containing the object is examined first. The object is examined against the ray only when the bounding volume is found to intersect the ray. Even though using the idea of the bounding volume reduces the time-consuming examination of the ray-object intersections, the computation time for ray tracing is still proportional to the number of objects. To solve this problem, the hierarchical bounding volume has been widely used [6], [9], [11], [15]. The hierarchical approach places objects in minimum bounding volumes and groups them recursively to form a minimum volume enclosing the entire world of the scene. As a result, the computation time is proportional to the logarithm of the number of objects.

K.-Y. Whang, J.-W. Song, J.-W. Chang, J.-Y. Kim, W.-S. Cho, and C.-M. Park are with the Computer Science Department and Center for Artificial Intelligence Research, Korea Advanced Institute of Science and Technology, Taejeon, 305-701, Korea; e-mail: kywhang@mozart.kaist.ac.kr.

I.-Y. Song is with the College of Information Studies, Drexel University-Philadelphia, PA 19104 USA.

To order reprints of this article, e-mail: transactions@computer.org, and reference IEEECS Log Number V95028.

Space subdivision is another popular technique to speed up ray tracing. This method divides a space into nonoverlapping unit elements of space. The unit element of space thus created is called the *voxel* [4], [5]. The major characteristic of space subdivision is that objects are examined in the order along the path of a ray. Hence, in order to find the closest object that intersects a ray, there is no need to sort all the objects. Besides, since only objects that lie in or near the path of a ray need to be examined, the method reduces the number of objects to be examined significantly.

Space subdivision methods are classified into two types depending on whether or not the space is uniformly divided [1]. The uniform subdivision method has the advantage of effective identification of the voxels that a ray intersects. Such identification can be done by simple calculations utilizing the regularity of the uniform structure [7], [4]. However, since this method divides the space uniformly regardless of object distribution, it loses adaptability. Adaptability means that the degree of space subdivision changes depending on the object distribution. In particular, as the space is divided into smaller voxels, a greater number of empty voxels are generated. Hence, not only the time is wasted for a ray to pass these empty voxels, but there is also memory overhead caused by the three-dimensional data structure representing the voxels.

These problems of the uniform subdivision methods can be overcome by using nonuniform subdivision methods. In nonuniform subdivision methods, a dense area of the space is divided into a large number of voxels and a sparse one into a small number of voxels. This way, the space is properly divided according to the object distribution, and thus, the time and the memory space are effectively saved. As the data structure used in nonuniform subdivision, Glassner [5] uses an octree, and Kaplan [8] a binary space partitioning tree (bintree). Both use spatial median to divide the subspace. The spatial median represents half the range of the subspace along each axis.

Several variants of the above methods have been introduced in literature. Scherson and Caspary [14] have proposed a unified technique exploiting the advantages of both hierarchical bounding volumes and octrees. MacDonald and Booth [10] have examined two heuristics for space subdivisions using bintrees: one based on the intuition that surface area is a good estimate of intersection probability, and the other based on the fact that optimal splitting plane lies between the spatial median and the object median of a volume. The *object median* is the splitting plane that places one half of the objects on each side of the plane.

1077-2626/95$04.00 © 1995 IEEE

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

In this paper, we propose the *Octree-R* (Octree for Ray Tracing), an octree variant, for efficient ray tracing. We also present the algorithm for constructing the Octree-R. The algorithm first estimates the number of ray-object intersection tests. Then, it partitions the space along the plane that minimizes the estimated number of the tests. Hence, the Octree-R naturally reduces the number of ray-object intersection tests compared with the simple octree using the spatial median.

The rest of this paper is organized as follows: In Section II, we briefly explain the basic concept of the octree and ray tracing using the octree. In Section III, we review the theoretical background to derive the Octree-R and then propose an algorithm for constructing the Octree-R. In Section IV, we present our experimental results comparing the performances of the Octree-R and the octree. This comparison shows superiority of the Octree-R over the octree in a wide-range of the ray tracing problems tested. Finally, in Section V, we summarize our contributions made in this paper and discuss future research directions.

II. Preliminaries

In this section, we explain the basic concept of the octree and ray tracing using the octree. An octree [13] is a hierarchical data structure showing how objects are distributed in the object space. Having been mainly used in image processing or solid modeling areas, it was first used for ray tracing by Glassner [5]. In this section, we first discuss how to construct an octree from a given space and object distribution.

If we divide a three-dimensional space for each axis using the spatial median, we obtain eight subspaces, which can be represented by an octree. The root node of an octree represents the entire object space. If the entire space contains more objects than a given limit, the space is divided into eight subspaces represented by eight children nodes. A subspace thus created is defined as a *voxel*. These voxels are further divided into eight voxels, and this process is repeated until the voxels satisfy the given criteria. In general, the criteria used to determine whether or not the given octree should be divided further depend upon the number of objects intersecting with a voxel and the maximum depth of an octree allowed [5], [8].

The nodes in an octree are classified into two types: *internal* nodes, which contain the positions of the planes dividing the subspaces represented by the nodes, and *leaf* nodes, which contain lists of objects intersecting with voxels. Leaf nodes of an octree correspond to the voxels through which rays are actually passing. Internal nodes of an octree provide access paths to leaf nodes. Once an octree is constructed, instead of examining all ray-object intersection tests, we find voxels that a ray intersects and examine only the objects that intersects with these voxels. Hence, using the octree could provide significant performance gain if the time taken for a ray to pass voxels is relatively smaller than the gain in speed obtained by reducing the number of ray-object intersection test.

Ray tracing using an octree is a two-step process. First, we determine the identity of the neighboring node to be visited next. Second, we locate the neighboring node. This two-step process is called the *octree traversal scheme*. Various octree traversal schemes have been proposed in the literature. Interested readers are referred to [2] for the classification of these schemes.

III. Octree-R Construction Algorithm

In this section, we present the Octree-R data structure for more efficient ray tracing. An Octree-R has the same structure as an octree except that the space is subdivided at a point that is not a spatial median. In Section III.A, we present a cost model for ray tracing using conventional octrees. In Section III.B, we present a method of finding an optimal plane for spatial subdivision based on the cost model in constructing an Octree-R.

A. Cost Model for Ray Tracing Using Octrees

Before we discuss the Octree-R construction algorithm, we introduce the cost model, which was originally presented in the reference [1] for ray tracing using octrees. The cost here represents the time taken for ray tracing. We define the number of voxels the ray has passed as n_v, the number of ray-object intersection tests as n_t, the time for the ray to move to the next voxel as T_v, and the time taken for a ray-object intersection test as T_i. Then the time taken for ray tracing can be represented as follows:

$$\text{Cost} = n_v \cdot T_v + n_t \cdot T_i \tag{1}$$

The equation shows that the total time taken for ray tracing is the summation of the time for a ray to pass the voxels and the time to do ray-object intersection tests.

We now discuss the meanings of (1) using an example. Suppose we are building an octree by repeatedly subdividing a given space. We consider two different octrees: one having N voxels and the other having less than N voxels. The fact that we have constructed an octree having a larger number of voxels in the same space means that we divided the given space more frequently and thus, in general, we have a smaller number of objects intersecting with each voxel. Hence, as we have a larger number of voxels, the number of ray-object intersection tests n_t in (1) becomes smaller. When we have a larger number of voxels, however, the number of voxels the ray needs to pass also increases, and thus the gain obtained by reducing n_t in (1) is compromised. That is, n_v and n_t in (1) have an opposing relationship to each other. Nevertheless, n_v in (1) is proportional to the total number of voxels. Thus, if we assume the same number of voxels in the octrees to be compared, the one having the smallest value for n_t will have the smallest total cost, as represented in (1). This assumption is practically reasonable since the number of voxels determines the main memory space that the octree occupies. Thus, optimization of ray tracing cost is conditioned by constant memory space.

In the next section, we present an algorithm to construct an Octree-R so as to reduce the ray tracing cost. We first observe that, for a given space with the same number of voxels, the expected number of ray-object intersection tests can be different depending on the position of the dividing plane, and then construct the Octree-R algorithm so as to minimize the expected value.

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

B. Octree-R Construction Algorithm

In this section, we propose an Octree-R construction algorithm that tries to minimize the expected number of ray-object intersection tests. We first explain the basic theory, which is based on [1], for computing the expected number of ray-object intersection tests.

Suppose volume A contains volume B as shown in Fig. 1. We denote the conditional probability for a ray r to pass volume B when the ray passes A as $\Pr(B|A)$. Assuming the rays are uniformly distributed, the conditional probability $\Pr(B|A)$ becomes the ratio of B's average projected area to A's average projected area [6]. Furthermore, if the volume is convex, the average projected area is one-fourth of the volume's surface [17]. This is a model (introduced in 1981) more formal than MacDonald and Booth's assumption (presented in 1990) [10] that the surface area is a good estimate of ray-object intersection probability. Thus, assuming that A and B are convex,

$$\Pr(B|A) = \frac{< P(B, \mathbf{d}) >}{< P(A, \mathbf{d}) >} = \frac{S_B}{S_A} \tag{2}$$

Here, $P(V, \mathbf{d})$ is the projected area of V perpendicular to the directions $\mathbf{d}$, $< >$ represents the average value for all directions $\mathbf{d}$, and S_V is the surface area of the volume V.

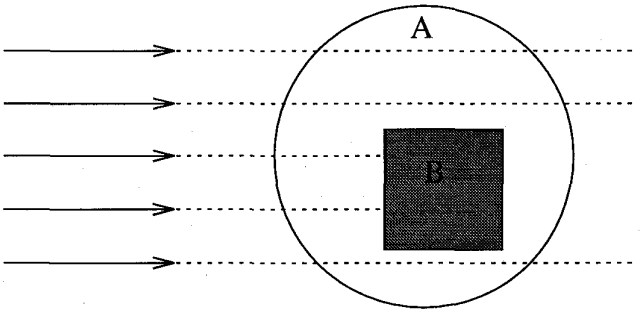


Fig. 1. Computing the conditional probability for a ray to pass through volume B once it passes through volume A.

We now estimate the expected number of ray-object intersection tests by applying (2) to the space subdivision technique. We assume that rays are randomly distributed over the entire space. Then, since the voxels created by dividing the given space are (convex) hexahedrons, we can apply (2). If we divide a given space R as shown in Fig. 2, the conditional probability for a ray to pass through the space A once it passes through the space R is as follows:

$$\Pr(A|R) = \frac{S_A(t)}{S_R} = \frac{2(tb + tc + bc)}{2(ab + ac + bc)} \tag{3}$$

Here, $2(tb + tc + bc)$ is the surface area of the hexahedron whose lengths of the sides are t, b, and c, along the axes x, y, and z, respectively; $2(ab + ac + bc)$ the surface area of the hexahedron whose lengths of the sides are a, b, and c, along the axes x, y, and z, respectively (see Fig. 2). Then, (3) can be simplified as follows:

$$\Pr(A|R) = \frac{t(b+c)+bc}{a(b+c)+bc} \tag{4}$$

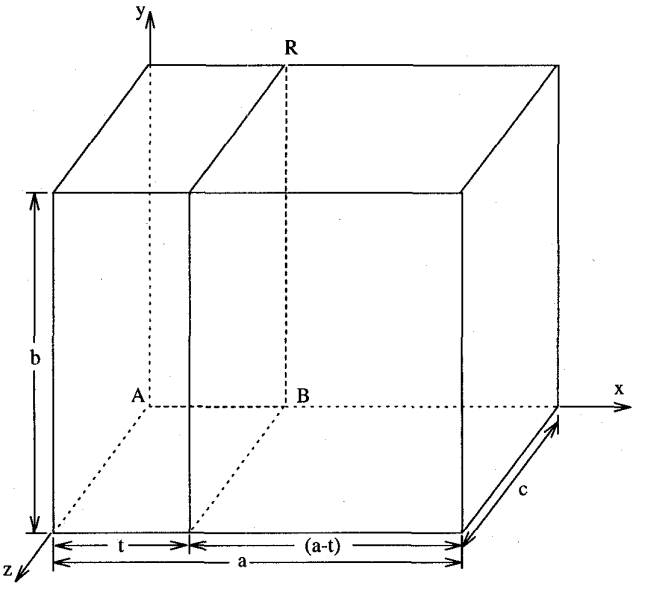


Fig. 2. Finding an effective dividing plane.

Similarly, the conditional probability for a ray to pass through space B once it passes through space R can be given as follows:

$$\Pr(B|R) = \frac{S_B(t)}{S_R} = \frac{(a-t)(b+c)+bc}{a(b+c)+bc} \tag{5}$$

Hence, assuming that no object intersects with the dividing plane and that there are $n(t)$ and $m(t)$ objects in spaces A and B, respectively, the expected number of ray-object intersection tests $E(t)$ as the ray passes through space R is given as follows:

$$\begin{aligned} E(t) &= \frac{S_A(t)}{S_R} n(t) + \frac{S_B(t)}{S_R} m(t) \\ &= \frac{t(b+c)+bc}{a(b+c)+bc} n(t) + \frac{(a-t)(b+c)+bc}{a(b+c)+bc} m(t) \end{aligned} \tag{6}$$

Let us consider a more general case allowing objects intersecting with the dividing plane. We denote the number of objects intersecting with plane t in Fig. 2 as $s(t)$, the number of objects completely included in the space A as $n(t)$, and the number of objects completely included in the space B as $m(t)$. Then, the expected number of ray-object intersection tests, $E(t)$, when the ray passes through the given space R is given as follows:

$$E(t) = \frac{S_A(t)}{S_R} n(t) + \frac{S_B(t)}{S_R} m(t) + \left\{ \frac{S_A(t)}{S_R} + \frac{S_B(t)}{S_R} \right\} s(t) \tag{7}$$

If we substitute (7) with (4) and (5), we obtain

$$\begin{aligned} E(t) = &\frac{t(b+c)+bc}{a(b+c)+bc} n(t) + \frac{(a-t)(b+c)+bc}{a(b+c)+bc} m(t) \\ &+ \frac{a(b+c)+2bc}{a(b+c)+bc} s(t) \end{aligned} \tag{8}$$

We have so far shown that, when we construct an Octree-R, the expected number of ray-object intersection tests depends on the position of the dividing plane in the space. Thus, we can

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

construct an Octree-R in such a way as to reduce the number of ray-object intersection tests by dividing the space with t that minimizes (8). In order to find the t value that minimizes (8), we would have to examine the entire space. Doing so, however, would cause too much overhead in constructing an Octree-R. Therefore, in this paper, we adopt the property identified by MacDonald and Booth [10] that the optimal splitting plane lies between the spatial median and the object median. The property was derived using a cost formula similar to (8).

In order to divide the given space into eight voxels, we first find the t values minimizing (8) for each axis of X, Y, and Z. The t value is found by examining the space at k + 1 different points equally spaced between the spatial and object medians. In the experiments, we have used ten for the value of k. We then divide the given space with these t values. In this process, the order of calculating three values of t is not important since we compute these values independently for each axis. Fig. 3 summarizes the Octree-R construction algorithm. In Fig. 3, MaxObj is the maximum number of objects allowed in a voxel. In procedure `Find_Dividing_Plane`, the object median is obtained by binary search with an error bound of the maximum of 5% of the number of the objects in the voxel and 1. The algorithm is identical to that of the octree except for the computation for optimal t. In the next section, we examine the effectiveness of this data structure through experiments.

IV. Experimental Results

In this section, we examine the performance of the Octree-R proposed in this paper through experiments. Section IV.A presents the method of generating data for the experiments, and Section IV.B interprets the experimental data.

```
Procedure Construct_Octree_R
```

1) Make the root voxel from the input data
2) Subdivide(`root_voxel`, MaxObj)

```
Procedure Subdivide(Voxel, MaxObj)
```

1) If the number of objects in Voxel ≤ MaxObj then return.
2) Find a dividing plane that minimizes (8) for each axis (X, Y, and Z);
   - t_x = `Find_Dividing_Plane`(X, Voxel);
   - t_y = `Find_Dividing_Plane`(Y, Voxel);
   - t_z = `Find_Dividing_Plane`(Z, Voxel);
3. Create eight subvoxel data structures from the parent voxel data structure using the point (t_x, t_y, t_z). Let Subvoxel[i] be the ith subvoxel data structure.
4) for ($i = 0$; $i < 8$; $i++$)
5) Subdivide(Subvoxel[i], MaxObj);

```
Procedure Find_Dividing_Plane(Axis, Voxel)
```

1) Find the object median by binary search with an error bound of max(5% of the objects, 1).
2) Find a dividing plane that minimizes (8) for Axis (X, Y, or Z) from k + 1 points equally spaced between the object median and the spatial median. These points include the two medians.
3) Return the dividing plane t for Axis.

Fig. 3. Octree-R construction algorithm.

A. Experiment Data

For the experiments, we have generated two different classes of random data. One class has uniform distributions and the other Gaussian distributions. Each class consists of two categories having different sizes of distributed objects. We also used four real data sets in the standard procedural database (tetra, rings, balls, and gears), the ray-traced views of which are shown in Figs. 4, 5, 6, and 7. Thus, we have a total of eight different data sets.

In Fig. 3, we set $k = 10$ since it is an empirically reasonable number. Sensitivity of the result on k is not very relevent since the construction time of Octree-R is only a small fraction of the ray tracing time.

The space in which objects are distributed is a three-dimensional one where each axis ranges between zero and one. The shape of each object is a triangle. The reason we assume a triangle for an object is that, in general, objects used in computer graphics can be represented by their surfaces and the objects' surfaces are represented by patches of triangles through triangulation. We used ten thousand objects for each data set (4,096–31,201 objects for the real data sets). The process of generating data consists of two steps. In the first step, the position of an object is determined in the three-dimensional space. In the second step, a triangle with the given size is generated at that position.

To generate the positions of the objects, we use a uniform random function and a Gaussian random function with the average of 0.5 and the standard deviation of 0.25. To generate triangles with the given size, we first make a sphere whose center is at the position determined in the previous step and whose radius is the given size. We then calculate the two points in the sphere using the uniform random function. The reason we use a uniform random function is to avoid generating triangles having identical sizes and directions. Now that we have located the three points, we can make a triangle by connecting the center and the other two points of the sphere. Even though the triangles generated in this way may intersect with each other, this does not create any problem since triangles are planes. We experimented with two different cases: in the first case the radius of the sphere has a constant value of 0.03; and in the second case the radius is uniformly distributed over the range between 0.005 and 0.03.

B. Experimental Results

When comparing the performance of the Octree-R with that of the octree, we have used the same number of voxels as has been discussed in Section IV.A. We have varied the number of voxels from 3,000 to 5,000 by varying blocking factors(the number of objects in a voxel) in order to see the effect of the degree of space partitioning.

The experimental results for the eight sets of the test data are shown in Figs. 8, 9, 10, and Fig. 11. The horizontal axis of Fig. 8 represents the number of leaf nodes (voxels), and the vertical axis the average number of ray-object intersection tests. Figs. 9, 10, and 11 show the effectiveness of the Octree-R compared with the octree that uses the spatial median. The effectiveness is defined as follows:

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

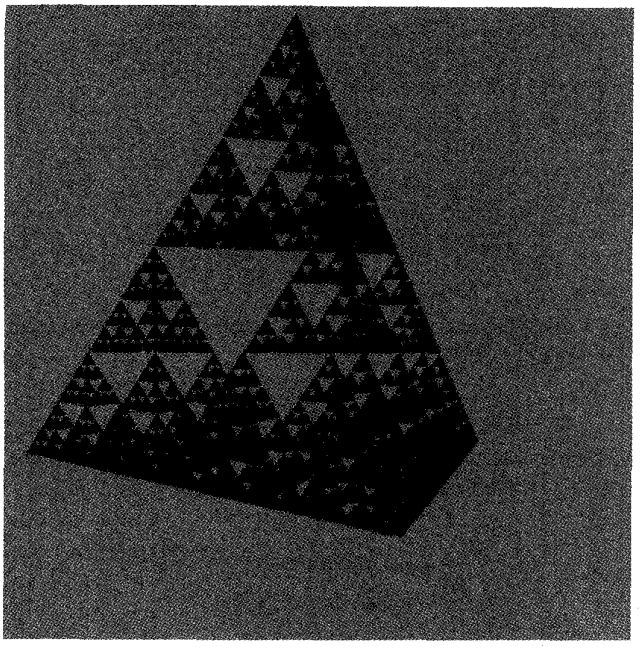


Fig. 4. The ray-traced view of the real data named "tetra" having 4,096 objects.

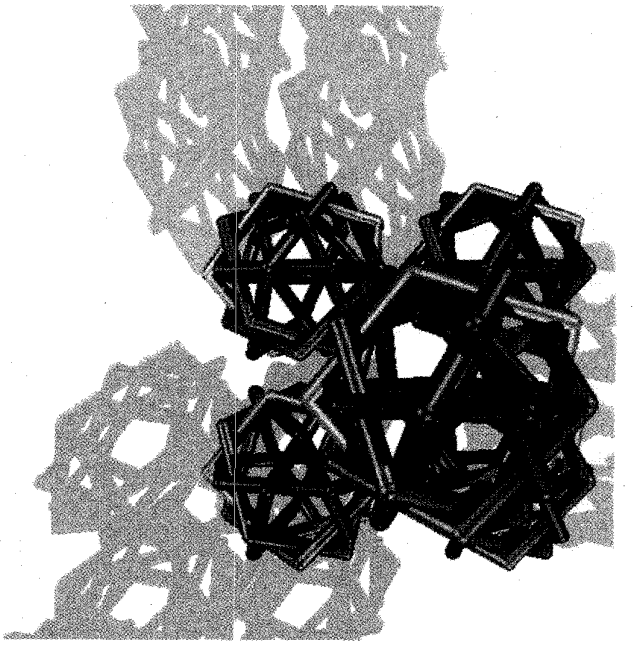


Fig. 5. The ray-traced view of the real data named "rings" having 31,201 objects.

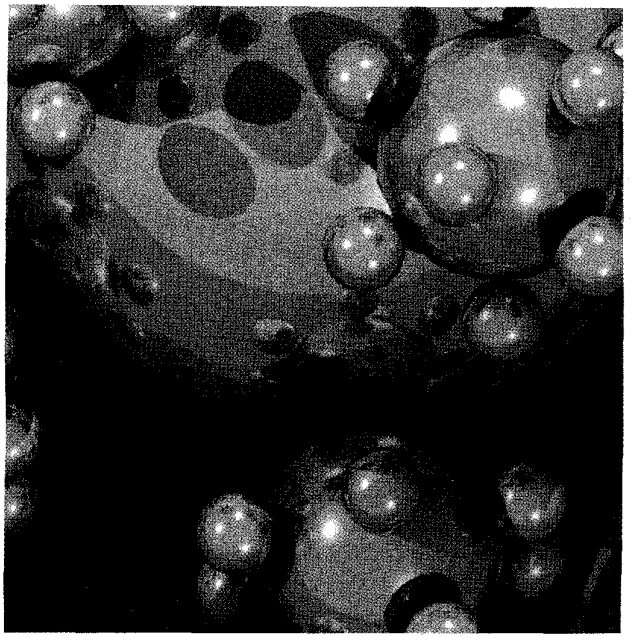


Fig. 6. The ray-traced view of the real data named "balls" having 17,473 objects.

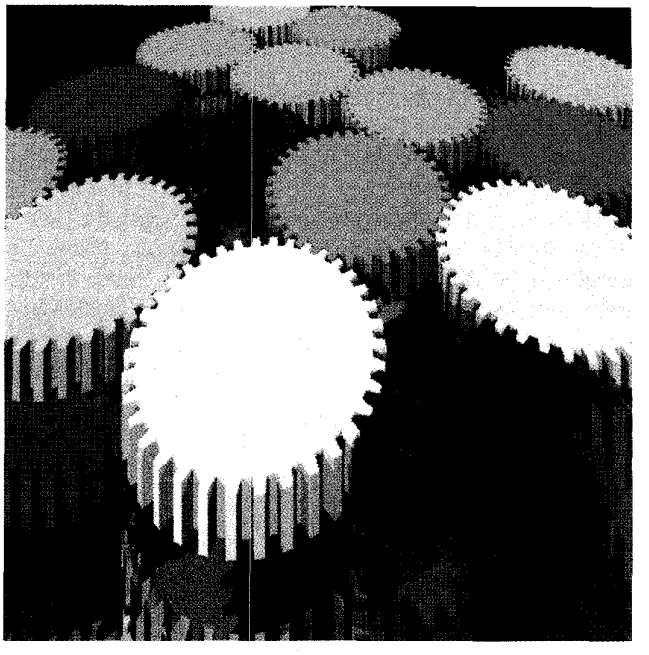


Fig. 7. The ray-traced view of the real data named "gears" having 9,345 objects.

$$\text{Effectiveness}\,(\%) = \frac{f(octree) - f(Octree\text{-}R)}{f(octree)} \times 100 \quad (9)$$

In (9), "f (octree)" represents the average number of ray-object intersection tests when the octree is used and "f (Octree-R)" when the Octree-R is used.

The figures show that the Octree-R gains from 4% to 47% over the octree. For the case of the uniform distribution, we observe the following trend: When the number of voxels is small, the effectiveness is about 4% and, as the number of voxels increases, it increases up to 11%. The reason the effectiveness is low when the number of voxels is small is that the dividing plane minimizing (8) is found near the spatial median since the data is uniformly distributed. If the number of voxels increases, the space is divided in a minute scale, and nonuniformity builds up locally moving the dividing plane of the Octree-R from the median. For the case of the Gaussian distribution, however, we observe that the effectiveness stays fairly flat around 12% regardless of the number of voxels. This phenomenon is easily understood since the Gaussian distribution is nonuniform.

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

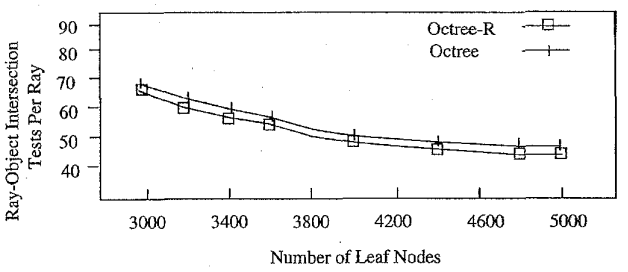


Fig. 8. Average number of ray-object intersection tests for the Octree-R with uniform distribution of spheres of radius 0.03.

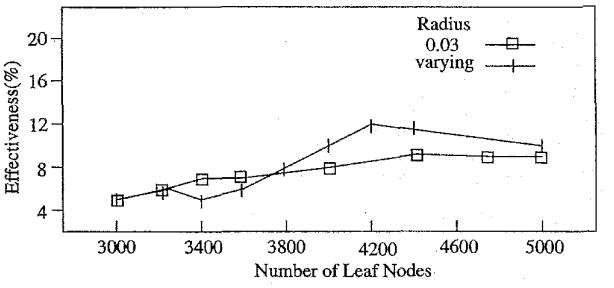


Fig. 9. The effectiveness of the Octree-R with uniform distribution of spheres of radius 0.03 and varying radii.

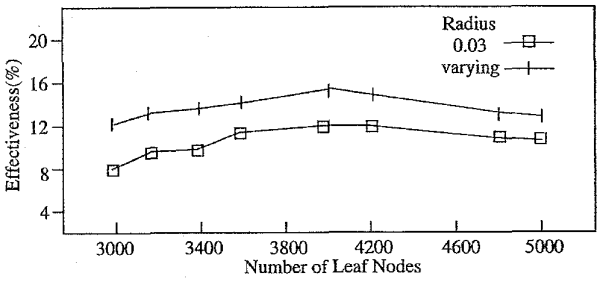


Fig. 10. The effectiveness of the Octree-R with Gaussian distribution of spheres of radius 0.03 and varying radii.

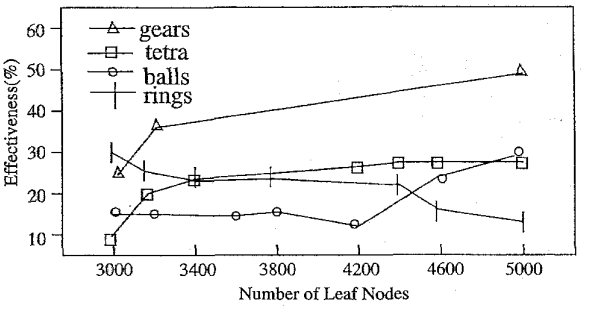


Fig. 11. The effectiveness of the Octree-R with a real data set in Fig. 8.

Finally, Fig. 11 shows that the effectiveness for real data is much higher between 9% and 47%. The result is a direct consequence of highly skewed (i.e., nonuniform) distribution of objects in the real data sets.

In Table I, we compare the time for constructing the Octree-R and the time for ray tracing for a real data set, "balls." The results show that although constructing an Octree-R takes somewhat more time than doing an octree, the construction time for the octree or Octree-R is far smaller than the ray tracing time and thus has a negligible effect on the overall ray tracing costs.

TABLE I
A COMPARISON OF CONSTRUCTION AND RAY TRACING COSTS FOR THE OCTREE AND OCTREE-R FOR A REAL DATA SET "BALLS"

Number of voxels	octree construction cost (sec)	Ray Tracing of octree cost (sec)	Octree-R construction cost (sec)	Ray Tracing of Octree-R cost (sec)
3,000	4.2	1023.3	34.5	821.3
4,000	4.9	855.5	37.5	715.1
4,900	6.0	830.9	41.4	607.2

V. CONCLUSIONS

In this paper, we have proposed a data structure for improving ray tracing. This structure, named the Octree-R, is constructed by dividing the space with the plane minimizing the number of ray-object intersection tests. Octree-R reduces the number of ray-object intersection tests in comparison with the octree that divides the space with a spatial median.

We have made experiments with two sizes of the objects and two different distributions as well as four real data sets. The experimental results have shown that, overall, the Octree-R has better performance than the octree by 4% to 47%. Regarding object distributions, the Gaussian distribution consistently shows better effectiveness than the uniform distribution. This implies that the Octree-R is more effective when the objects are nonuniformly distributed than when the data is uniformly distributed. The reason is that, when the data is uniformly distributed, similar-sized voxels contain similar numbers of objects, and thus the dividing plane minimizing (8) is near the spatial median. This argument is strongly supported by experiments using real data sets with highly nonuniform distributions. Here, we obtained much higher effectiveness than from other data sets.

The Octree-R construction algorithm proposed in this paper first finds the most effective plane for each axis, and then uses the set of three planes to divide the current voxel. This set of planes may not be optimal since each axis is considered independently. Finding the optimal set of planes is left as a further study.

REFERENCES

[1] J. Arvo and D. Kirk, "A survey of ray tracing acceleration techniques," *An Introduction to Ray Tracing*, A.S. Glassner ed., Academic Press, 1989.

[2] R. Endl and M. Sommer, "Classification of ray-generators in uniform subdivisions and octrees for ray tracing," *Computer Graphics Forum*, vol. 13, no. 1, pp. 3-19, 1994.

[3] J.D. Foley, A. van Dam, S.K. Feiner, and J.F. Hughes, *Computer Graphics: Principles and Practice*, second ed., Addison-Wesley, 1990.

[4] A. Fujimoto, T. Tanaka, and K. Iwata, "ARTS: Accelerated ray tracing system," *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16-26, Apr. 1986.

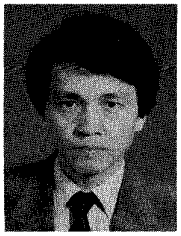
Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.

[5] A.S. Glassner, "Space subdivision for fast ray tracing," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 15-22, Oct. 1984.
[6] J. Goldsmith and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *IEEE Computer Graphics and Applications*, vol. 7, no. 5, pp. 14-20, May 1987
[7] P.K. Hsiung and R. Thibadeau, "Accelerating ARTS," *The Visual Computer*, vol. 8, pp. 181-190, 1992.
[8] M.R. Kaplan, "The use of spatial coherence in ray tracing," *Techniques for Computer Graphics*, D.F. Rogers and R.A. Earnshaw eds., Springer-Verlag, 1987.
[9] T.L. Kay, and J.T. Kajiya, "Ray tracing complex scenes," *Computer Graphics*, vol. 20, no. 4, pp. 269-278, Aug. 1986.
[10] J.D. MacDonald and K.S. Booth, "Heuristics for ray tracing using space subdivision," *The Visual Computer*, vol. 6, pp. 153-166, 1990.
[11] S.M. Rubin and T. Whitted, "A 3-dimensional representation for fast rendering of complex scenes," *Computer Graphics*, vol. 14, no. 3, pp. 110-116, July 1980.
[12] H. Samet, "Implementing ray tracing with octrees and neighbor finding," *Computers and Graphics*, vol. 13, no. 4, pp. 445-460, 1989.
[13] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, 1989.
[14] I.D. Scherson and E. Caspary, "Data structures and the time complexity of ray tracing," *The Visual Computer*, vol. 3, pp. 201-213, 1987.
[15] H. Weghorst, G. Hooper, and D.P. Greenberg, "Improved computational methods for ray tracing," *ACM Trans. on Graphics*, vol. 3, no. 1, pp. 52-69, Jan. 1984.
[16] T. Whitted, "An improved illumination model for shaded display," *Comm. ACM*, vol. 23, no. 6, pp. 343-349, June 1980.
[17] H.C. Van de Hulst, *Light Scattering by Small Particles*. Dover Publications, 1981.

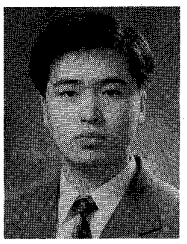
Kyu-Young Whang (M'83–SM'88) graduated (Summa Cum Laude) from Seoul National University, in 1973, and received the MS degrees from the Korea Advanced Institute of Science and Technology (KAIST) in 1975 and Stanford University in 1982. He earned the PhD degree from Stanford University in 1984.

From 1983 to 1991, he was a research staff member at the IBM T.J. Watson Research Center, Yorktown Heights, N.Y., where he performed various research projects in databases, office systems (including Office-by-Example), and expert systems. He is now a full professor in the Computer Science Department of KAIST and the director of the Database and Knowledge Engineering Laboratory of the Center for Artificial Intelligence Research. His research interests encompass multimedia databases, object-oriented databases, engineering databases, advanced storage systems, and GIS.

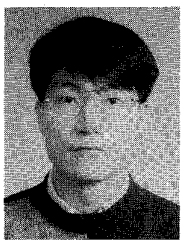
Dr. Whang served as an IEEE Distinguished Visitor from 1989 to 1990; received the Best Paper Award from the *Sixth IEEE International Conference on Data Engineering*; served the *Fifth IEEE International Conference on Data Engineering* as a program cochair; and has served program committees of numerous international conferences, including ACM SIGMOD and VLDB. He twice received the External Honor Recognition from IBM. He is on the editorial boards of *VLDB Journal* and *International Journal of Geographic Information Systems*. Being on the board of directors, Dr. Whang is the editor-in-chief of the *Journal of Korea Information Science Society*. He is a senior member of the IEEE and a member of the ACM.



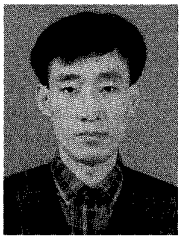
Ju-Won Song received the BS degree in electrical engineering from Kyungpook National University in 1981, the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1983. Since 1983, he has been a research engineer in Korea Telecom (KT). He is also a PhD candidate in computer science at KAIST. His research interests include spatial database systems, geographical information systems, and multidimensional access methods.



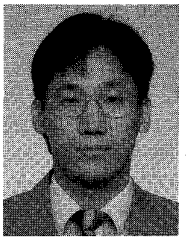
Ji-Woong Chang graduated from Yonsei University in 1993 and received the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1995. He is currently working toward the PhD degree in computer science at KAIST. His research interests include storage systems, concurrency control, and memory resident databases.



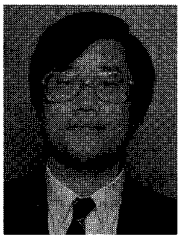
Ji-Yun Kim received the BS degree in electronics engineering from Seoul National University, Seoul, Korea, in 1990, and the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST), Taejon, Korea, in 1992. He is currently a PhD candidate at KAIST. His research interests include computer architecture, parallel and distributed systems, and computer graphics.



Wan-Sup Cho received the BS degree in statistics from Kyungpook National University in 1985 and the MS degree in computer science from Korea Advanced Institute of Science and Technology (KAIST) in 1987. From 1987 to 1990, he served as a research engineer in the Electronics and Telecommunications Research Institute, Korea. He is currently working toward the PhD degree in computer science at KAIST. His research interests include object-oriented database systems, database design, graphical/visual user interface, and computer graphics.



Chong-Mok Park graduated from Yonsei University in 1990 and received the MS degree from the Korea Advanced Institute of Science and Technology (KAIST) in 1992. In 1993, he worked as a summer student at Hewlett-Packard Laboratories, Palo Alto, Calif. He is now a PhD student in the Computer Science Department of KAIST. His research interests include object-oriented databases, multimedia databases, and hypermedia.



Il-Yeol Song received the BS degree from Han-Yang University in 1975, the MS and the PhD degrees from Louisiana State University in 1984 and 1988, respectively. From 1975 to 1982, he was a researcher and a senior researcher in the Agency for Defense Development, Korea. He is now associate professor in the College of Information Studies of Drexel University. His research interests encompass database management systems, object-oriented databases, knowledge-based systems, multimedia information systems, and digital library. Dr. Song has served on program committees of numerous international conferences, including CIKM and ER approach and as a guest editor of the *Journal of Computer and Software Engineering*.

Authorized licensed use limited to: Korea Advanced Institute of Science and Technology. Downloaded on July 23, 2009 at 08:24 from IEEE Xplore. Restrictions apply.