

Discrete Ray Tracing

Roni Yagel, Daniel Cohen, and Arie Kaufman

Abstract

We present a discrete ray tracing method, called 3D raster ray tracing, that is in practice insensitive to the complexity of the scene or the object type, and therefore provides a substantial improvement in computational speed over existing algorithms. In a preprocessing voxelization phase, the geometric model is converted into its discrete voxel-based representation within a 3D raster of atomic voxels. Next, a recursive ray tracing algorithm is engaged, in which 3D discrete rays are traversed through the 3D raster. Encountering a non-transparent voxel by the ray traversal algorithm indicates a ray-surface hit, thereby eliminating the need for the computationally expensive ray-object intersection calculations. We also discuss efficient incremental algorithms for discrete ray traversal, adaptive traversal of discrete rays, and improvements to image quality by employing geometric intersection verification. The 3D raster based approach to ray tracing is especially attractive for rendering complex geometric scenes and constructive solid models. In addition, it supports recursive ray tracing of 3D sampled and computed datasets possibly intermixed with geometric objects.

Introduction

Ray tracing, despite its superior image quality, popularity, power, and simplicity, has the reputation of being computationally very expensive. The basic operation of ray tracing is the calculation of the intersection points between rays and objects. This operation has been reported to consume at least 95% of the computation time in rendering complex scenes. A major strategy for reducing the cost of ray tracing is to reduce the average cost of ray-scene intersection, either by devising efficient algorithms for intersecting rays with specific objects or by reducing the number of objects tested against the ray. The latter is achieved mainly by two methods: hierarchical bounding volumes and space subdivision.

The space subdivision method divides the world space into a set of rectangular subvolumes called cells or voxels. We refer to subvolumes as *cells*, and reserve the label *voxel* for the atomic volume element, which is the smallest space enumeration entity. A cell consists of a list of all objects residing in that region of the world while a voxel maintains information of one object only. Space subdivision can be implemented in two variations: nonuniform and uniform spatial subdivision. Nonuniform subdivision partitions space into portions of varying sizes to suit the features of the scene, using either octrees or BSP trees. Uniform subdivision partitions space into uniform size cells organized in a 3D lattice [1, 3, 5]. ARTS [5] employs a 3D digital differential analyzer (3DDDA) for incrementally computing all cells in a uniform partition pierced by a ray. Only the lists of objects residing in the pierced cells are candidates for ordered ray-object intersection calculation.

When the scene to be rendered consists of volumetric datasets such as those arising in biomedical imaging and scientific visualization applications, *ray casting* methods have been applied, in which only primary rays are traced. Levoy [12] has used the term ray tracing of volume data to refer to ray casting with compositing of even-spaced samples along the primary viewing rays. Since ray casting follows only primary rays, it does not directly support the simulation of light phenomena such as reflection, shadows, and refraction. On the other hand, the study of rendering volume densities by ray casting has included, among others, a probabilistic simulation of light passing through and being reflected by clouds of small particles [2], and the light-scattering equations of densities within a volume grid, such as clouds, fog, flames, dust, and particle systems [6].

The approach taken in this paper is called *discrete ray tracing* or *3D raster ray tracing (RRT)*. Unlike existing ray tracing methods that use geometric representation for the 3D scene, RRT employs a 3D discrete raster of voxels for representing the 3D scene in the same way a 2D raster of pixels represents a 2D image. The 3D raster maintains in every voxel some attributes of the object piercing the volume unit that the voxel represents. Voxels of 3D rasters, like pixels of 2D rasters, are atomic in the sense that each voxel maintains attributes of only one object entity.

RRT operates in two phases: a preprocessing voxelization phase and a discrete ray tracing phase. In the voxelization phase, discussed in the next section, the geometric model is digitized, using 3D scan-conversion algorithms that convert the continuous representation of the model into a discrete representation within the 3D raster. For datasets that are already digitized, as in 3D medical imaging and 3D computational visualization, the discretization step is of course unnecessary. In the second phase a discrete variation of the conventional recursive ray tracer is employed. Unlike

conventional algorithms, in which analytical rays are intersected with the object list in order to find the closest intersection, in RRT 3D discrete rays are traversed through the 3D raster in order to find the first surface voxel. Encountering a non-transparent voxel indicates a ray-surface hit.

In conventional ray tracing, computation time grows with the number of objects [5], because in crowded scenes a ray may pierce a substantial number of objects and there is a considerable probability that a cell contains more than one object. Moreover, conventional ray tracers are extremely sensitive to the type of objects comprising the scene; intersection calculation between a ray and a parametric surface is significantly more complex than intersecting the ray with a sphere or a polygon. In contrast, RRT completely eliminates the computationally expensive ray-object intersections calculation, and instead relies solely on a fast discrete ray traversal mechanism and a single simple type of object – the voxel. Consequently, RRT is in practice independent of the number of objects in the scene or the object’s complexity or type. RRT performance, however, is sensitive to the resolution of the 3D raster. Therefore, for a given resolution, ray tracing time is nearly constant and can even decrease as the number of objects in the scene increases, as less stepping is necessary before an object is encountered.

Any change in the view-dependent parameters requires a traditional ray tracer to execute a full fledged rendering that consists of recomputing many view-independent attributes such as surface normal, texture color, and light source visibility and illumination. In contrast, RRT precomputes the view-independent attributes during the voxelization phase and stores them within each voxel. These attributes are readily accessible for multiple rendering of the fixed scene in which viewing, lighting, and shading parameters change.

While traditional ray tracers are capable of rendering only objects represented by geometric surfaces, RRT is also attractive for ray tracing 3D sampled datasets (e.g., 3D MRI imaging) and computed datasets (e.g., fluid dynamics simulations), as well as hybrid models in which such datasets are intermixed with geometric models (e.g., a scalpel superimposed on a CT image, radiation beams superimposed on a scanned tumor, or a plane fuselage superimposed on a computed air pressure) [10]. Unlike non-recursive ray casting techniques, RRT, which recursively considers both primary and secondary rays can model shadows and reflections for photorealistic imaging. RRT offers the use of ray tracing for improved visualization of sampled and computed volumetric datasets [16].

Ray tracing CSG (Constructive Solid Geometry) models is traditionally achieved by converting the CSG-tree into a boundary representation (B-rep) in an extremely time consuming process. Alternatively, the direct CSG-rendering approach delays the computation of the Boolean operations until the rendering stage. However, the fact that processing time increases more rapidly than the complexity of the model remains a major hurdle confronting this approach. In contrast, as the spatial enumeration of the 3D raster lends itself to voxel-by-voxel Boolean operations, RRT can easily support ray tracing of CSG models that are efficiently synthesized and constructed during the voxelization phase.

Although the voxelization and the ray traversal phases introduce some aliasing, the true normal stored with each voxel is used for spawning secondary rays, which in fact smoothes out the aliasing of the surface. The aliasing artifacts can be further reduced by generating antialiased non-binary objects during the voxelization phase, by super-sampling during the ray tracing phase, and/or by consulting an object table for the exact geometric structure at the hit point (see section “Reducing Aliasing” for more details).

The images shown in this paper, which are comparable in quality to images of equivalent resolution of conventional ray tracing, were generated on 80M and 128M byte machines from 256^3 through 320^3 spatial resolution images. The RRT approach assumes that the geometric model or the sampled dataset are discretized to the desired resolution of the pixel image, that is, the voxel footprint is approximately that of the pixel. Therefore, rendering a higher resolution image requires a higher resolution volume. However, as large memories prevail on graphics workstations, it is no longer questionable whether the voxel representation is feasible as a viable alternative for realistic high resolution rendering with ray tracing.

Voxelization Phase

RRT of geometric scenes commences with a preprocessing voxelization phase that precedes the actual ray tracing. In the voxelization phase, the scene is converted into a discrete voxel representation by 3D scan-converting (voxelizing) each of the geometric objects comprising the scene. A voxelization algorithm for a given geometric object generates the set of voxels that “best” approximates the continuous specification of that object and stores its discrete representation as a 3D array of unit voxels.

A voxel is the 3D conceptual counterpart of the 2D pixel; each voxel is a small quantum unit of volume that has numeric values associated with it representing some measurable properties or attributes of the real object or phenomenon at that voxel. Like a pixel, the voxel is very small and thus assumed to be homogeneous, that is, it contains information regarding one single object only. The aggregate of voxels tessellates the 3D grid of voxels which is termed the *3D raster* or the *volume buffer*.

The voxelization algorithms of RRT generate for each surface voxel its color, opacity, texture, and normal vector, by sampling those fields at the integral grid point located at the voxel center, although other sampling paradigms are also possible. In addition, two bits per light source are provided for each voxel to denote whether that light source is visible, occluded, or visible through a translucent material [15]. Actually, the view-independent parts of the illumination equation, that is, the ambient illumination and the sum of the attenuated diffuse illumination of all the visible light sources, can also be precomputed and added to the voxel color. All the view-independent attributes that are precomputed during the voxelization stage and stored with the voxel are then readily accessible for speeding up the ray tracing phase (see next section).

Digitization of solids was commonly performed by spatial enumeration algorithms which employ point or cell classification methods in an exhaustive fashion, or preferably by recursive subdivision. However, subdivision techniques for model decomposition into rectangular subspaces are computationally expensive and thus inappropriate for medium or high resolution grids. The voxelization algorithms we employ, on the other hand, follow the same paradigm as the 2D scan-conversion algorithms commonly used in 2D raster graphics; they are incremental, use simple arithmetic (preferably integer-based), and have a complexity that is not more than linear with the number of voxels generated (see Table 2 for voxelization time of a scene with increased population of spheres).

The literature of 3D scan conversion is relatively small. Mokrzycki [13] employed a 3D curve algorithm where the curve is defined by the intersection of two implicit surfaces. We have introduced elsewhere [7] voxelization algorithms for 3D lines, 3D circles, and a variety of surfaces and solids, including polygons, polyhedra, and quadric objects. We have also described efficient algorithms for voxelizing polygons using an integer-based decision mechanism embedded within a scan-line filling algorithm [9]; parametric curves, surfaces, and volumes using an integer-based forward differencing technique [8]; and quadric objects such as cylinders, spheres, cones, and tori using “weaving” algorithms by which a discrete circle/line sweeps along a discrete circle/line [4]. All these algorithms preserve the topology of the voxelized objects and allow some control over their connectivity and thickness.

A voxel is assumed to be a tiny cube centered at the 3D grid point. Two voxels are said to be *26-adjacent* if they either share a vertex, an edge, or a face. Every voxel has 26 such adjacent voxels. Eight share a vertex (corner) with the center voxel, twelve share an edge, and six share a face. Every two consecutive voxels along a *26-connected* curve or line are 26-adjacent. Accordingly, face-sharing voxels are defined as *6-adjacent*, and every two consecutive voxels in a *6-connected* curve or line are 6-adjacent. Our voxelization algorithms can generate curves and lines that are either 6-connected or 26-connected.

A voxelized surface approximating a connected continuous surface has to be connected (e.g., 26-connected). However, connectivity does not fully characterize the surface because the voxelized surface may contain local discrete holes, termed *tunnels*, that are not present in the continuous surface. A tunnel is a passage by a discrete line through a voxel-based surface; the line crosses from one side of the surface to the other. A requirement of our surface voxelization algorithms is that the volumetric representation of a surface must be “thick enough” not to allow the discrete rays (3D lines) to penetrate them. A voxelized surface through which 6-connected rays do not penetrate is a *6-tunnel-free* surface, while a thicker surface through which even 26-connected rays can not pass is a *26-tunnel-free* surface [4]. Our voxelization algorithms can generate any of these types of surfaces; the decision which one to use depends primarily on the connectivity of the discrete rays employed during the discrete ray tracing phase.

Rasters, in general, lend themselves to block operations such as *bitblt* or their 3D counterparts, *voxblt* operations [11]. In particular, the 3D raster lends itself to Boolean operations that can be performed during the voxelization stage. This makes ray tracing of CSG models trivial. Voxel-by-voxel block operations during the voxelization phase add a variety of modeling capabilities which aid in the task of image synthesis by supporting the manipulation of true solids (i.e., exterior surfaces as well as interior). Figure 1 shows an RRT of a 256^3 reconstructed MRI head reflected in a mirror. The mirror and the head have been generated during the voxelization phase. The cut in the head was generated by a CSG operation of subtracting a box from the MRI dataset.

Discrete Ray Tracing Phase

Once the discrete representation of the scene is available, a discrete variation of the conventional ray tracing algorithm is employed. Rays are recursively traced through the scene by stepping along the 3D discrete line representation of the ray through the 3D raster of voxels. The discrete ray traversal algorithm is thus central to the discrete ray tracing stage, and in the next section we present an efficient algorithm for 3D discrete ray traversal.

Figure 1 : A 256^3 resolution reconstructed MRI head reflected in a circular mirror. A portion of the head was removed by a CSG subtraction. RRT time is 28.1 seconds.

For each screen pixel, our ray tracer computes the intersections of the ray emanating from that pixel with the boundary of the 3D raster, defining the two endpoints of the discrete ray. The discrete ray traversal commences at the closer endpoint – the discrete ray origin. The first non-transparent voxel encountered by the discrete ray traversal indicates a ray-surface hit. We use the term *hit* for the point where the discrete ray and the discrete surface meet and the term *intersection* for the traditional continuous ray-object crossing. When a hit occurs, the voxel encountered by the ray is known and its attributes are readily available. RRT thus eliminates the need to compute and search for an intersection among several objects that might be present in the cell. Furthermore, unlike traditional ray tracing techniques, RRT does not compute the normal at the intersection point. Instead, the true normal vector to the surface, which is stored in the hit voxel, is retrieved and used for spawning reflective and transmitted rays.

Each surface voxel also stores the color, illumination, and/or the texture color for that voxel. Since the texture mapping is view-independent, it is precomputed during the voxelization algorithm from a 2D texture/photo map or a 3D texture. There is no need, for example, to recompute for every different viewpoint the inverse mapping needed for texture mapping.

Figure 2 : Newell's teapot over a textured floor, mirrored in two mirror walls. RRT time for this 256^3 resolution image is 36.8 seconds.

Since the visibility of the light sources from each voxel is also viewpoint independent, shadow rays are fired from each surface voxel to all light sources during the voxelization preprocessing stage. The light visibility information is stored in two bits per light source per voxel, denoting whether that light source is visible, occluded, or visible through a translucent material. A shadow ray will be fired at rendering time only for the third case, in order to determine the actual light intensity. This provides substantial savings, since the vast majority of shadow (or illumination) rays will not be fired at all during the rendering phase. Furthermore, as mentioned in the previous section, the view-independent ambient and attenuated diffuse illumination of all the visible light sources are precomputed during the voxelization stage and stored with each voxel value. The ray tracing stage has to compute only the attenuated specular illumination for each light source. (In fact, the attenuation percentage can also be precomputed and saved with each voxel.) The sum of all the illumination values plus the recursive reflected and transmitted components are added during the discrete ray traversal phase to the illumination value stored at the voxel, producing the total illumination of that voxel.

The only geometric computations performed by the discrete ray tracing phase are for generating discrete rays, spawning secondary rays, and compositing colors. Since all objects have been converted into one object type – the unit voxel, the traversal of discrete rays can be viewed as performing an efficient ray-voxel intersection calculation. Having only one object type frees the ray tracer from any dependency on the type of objects composing the image. Figure 2 shows an image consisting of one Newell’s teapot that is ray traced in 36.8 seconds, which is approximately the same time as ray tracing one sphere in the same environment.

The RRT rendering time is sensitive to the total distance traversed by the rays, which depends primarily on the constant 3D raster resolution and the portion of the 3D raster occupied by the objects, and it is in practice insensitive to the complexity of the scene. It is thus not surprising to achieve improved performance when tracing scenes of higher complexity. See the section “Results” for examples.

The RRT approach provides true recursive ray tracing (rather than ray casting) of sampled/computed datasets, as well as hybrid scenes where geometric and sampled/computed data are intermixed. Figure 1 shows ray tracing of a medical dataset in which additional information becomes available to the observer upon utilizing reflection. The normal vector to the sampled or computed surfaces can be estimated, either during a preprocessing stage when it is stored with the voxel, or during the ray tracing stage. The normal estimation may employ one of many volume shading approaches such as: contextual shading, gray-level gradient, context sensitive normal estimation, or biquadratic local surface interpolation. We have chosen to employ a variation of the gray-level gradient approach [14] that examines all values in a $3 \times 3 \times 3$ neighborhood in order to enhance normal integrity.

Efficient Discrete Ray Traversal

In RRT, the computationally expensive intersection calculation is eliminated, and instead the load falls almost exclusively on the discrete ray traversal algorithm. We have observed in our experiments with RRT that up to 90% of the ray tracer execution time is consumed by the discrete ray traversal algorithm. An efficient discrete ray traversal algorithm is thus imperative for an efficient RRT. Such an algorithm utilizes a 3D discrete line voxelization mechanism to generate the sequence of voxels along the ray. Such line algorithms have previously been devised for uniform space-subdivision ray tracing. Fujimoto et al [5] have presented a 3DDDA algorithm. It generates a 6-connected line, which includes all of the cells pierced by the continuous line. A more efficient algorithm has been developed [1], and later has been elegantly presented [3], where suggestions for further optimization are also given. Basically, all

these algorithms generate 6-connected lines with floating-point calculations of a parametric line, using fixed point arithmetic. We have presented an integer based 26-connected line algorithm [7], which was latter extended to handle other types of connectivities [4]. Specifically, we have devised a 6-connected line algorithm that employs only integer arithmetic and requires less operations per voxel.

As was mentioned above, every two consecutive voxels in a *6-ray* (6-connected line) are face-adjacent, which guarantees that the set of voxels along the discrete line includes all voxels pierced by the continuous line. A 6-ray from (x,y,z) to $(x+\Delta x,y+\Delta y,z+\Delta z)$ (assuming integer line endpoints) has the total length of n_6 voxels to traverse, where

$$n_6 = |\Delta x| + |\Delta y| + |\Delta z|. \quad (1)$$

In a *26-ray* (26-connected line), each two consecutive voxels can be either face-adjacent, edge-adjacent, or corner-adjacent. The line has a length of

$$n_{26} = \text{MAX}(|\Delta x|, |\Delta y|, |\Delta z|). \quad (2)$$

Clearly, a 26-ray is shorter than a 6-ray:

$$1 \leq \frac{n_6}{n_{26}} \leq 3. \quad (3)$$

When the line is parallel to one of the primary axes $n_6=n_{26}$, while approaching the main diagonal increases the ratio n_6/n_{26} up to 3. We have observed in our experiments a random distribution of ray directions and a traversal speedup of 26-rays compared to 6-rays of up to 1.84, which is the expected speedup that can be deduced from the metrics of the two connectivities (Equations 1 and 2).

Since the performance of RRT depends almost solely on the total number of voxels traversed by the discrete rays, our algorithm traverses 26-rays, which have much fewer voxels than 6-rays. Our ray traversal is based on a fast version of the 26-connected line algorithm which employs a tree of condition statements and a pointer into the 3D array (raster) that has to be incremented only once per iteration. The algorithm is actually an enhanced double Bresenham's 2D line algorithm. It takes unit steps along the direction with the largest extent and uses a decision mechanism for the other two axes.

While 26-rays have the clear speed advantage, their use implies that the surfaces voxelized during the preprocessing stage are 26-tunnel-free, that is, they must be thick enough to eliminate possible penetration by a 26-ray. In addition, a 26-ray does not traverse all voxels pierced by the continuous ray and thus may skip a voxel in which the ray hits the object. Consequently, ray-surface hits may be missed at the object

silhouette (see point C in Figure 3 (a)) and occasionally hit a voxel which is underneath the true surface. Although we found, from our experience, that the percentage of 26-rays that missed was usually less than 1.5%, it is an image dependent artifact that could force us to abandon the 26-rays speed advantage for the 6-ray accuracy. In this case, we can also save some preprocessing time by voxelizing only 6-tunnel-free surfaces.

Our solution to the this speed vs. accuracy dilemma is to use 26-rays to rapidly traverse the empty space and employ the 6-rays only when approaching the scene objects. However, this approach requires both a line algorithm that can efficiently switch between connectivities and a mechanism to sense object proximity. We observe that since both the 26- and 6-ray algorithms use the same variables and only the decision mechanism is different, it is possible to adaptively alternate between 26- and 6-rays with no extra cost. To implement the feature of proximity sensing, one has to add (during the voxelization stage) a proximity flag for all the voxels around the object surface to indicate that we are in the vicinity of an object.

The traversal starts as a 26-ray which rapidly skips over the transparent regions. Whenever a proximity flag is encountered, the 26-line algorithm adaptively “slows down” and takes 6-connected steps in order to avoid misses at the object surface. The adaptive line algorithm performs the switching between connectivities without noticeable time penalty. The extra proximity flags in the adaptive algorithm add only one test per step to the overall execution time of the algorithm. Employing 26-rays has an overall speedup compared to 6-rays in all views and not only from the diagonal directions (where $n_6 = 3n_{26}$), since primary rays are not parallel to each other and secondary rays are spawned in arbitrary directions.

Reducing Aliasing

Spatial aliasing in traditional ray tracing is caused by point sampling in screen space and a common cure for it is supersampling in screen space. RRT also supports screen supersampling by adopting a 3D line algorithm that can handle sub-pixel addressing of the ray origin (see Figure 5(b) for example).

(a)

(b)

Figure 3: (a) A 26-ray passing through a volume consisting of three elliptical objects. At point C the ray has a miss hit and at point B it has a false hit. (b) The passage of a 6-ray in the same scene. The 6-ray has a false hit at A and B but did not miss the hit at C.

RRT also suffers from another type of aliasing caused by point sampling in object space, as the values of the voxel attributes are determined by sampling the object at integral coordinates (voxel centers). Another manifestation of object space aliasing is the false hit problem encountered in both 6- and 26-rays (see Figure 3, point B). It stems from the fact that a discrete hit point between the discrete ray and the discrete surface does not necessarily indicate an intersection between their continuous counterparts, that is, the continuous ray and the continuous surface can pass through the same voxel and still not meet each other. From our experience, however, we found that only 1-8% of the rays experience a false hit.

The remedy to object-space aliasing is based on storing in each voxel the information regarding the geometric definition of the object occupying that voxel. Specifically, during voxelization each voxel belonging to an object is assigned an object-id instead of the normal vector information, as in the basic version of RRT. The object-id is used as an index into an object table consisting of the objects' geometric definition. Whenever a discrete ray hits a voxel, the hit is verified by computing the true intersection between the continuous ray and the geometric object definition. If there was no intersection, the ray would resume its discrete traversal of the 3D raster. Figure 4(a) shows the 320^3 820-sphereflake ray traced with 6-rays and no intersection verification, while Figure 4(b) shows the same image ray traced with true intersection verification. The two images are hardly distinguishable, in this high volume resolution, and their ray tracing time is about the same (74.9 seconds and 85.0 seconds, respectively).

The slight computation overhead involved in intersection verification can also provide us with the true intersection point and the normal vector, solving the problem of object space aliasing in terms of surface attribute sampling. Figure 5 compares the four different variations of RRT: the basic RRT (described above), RRT with screen supersampling, RRT with intersection verification, and RRT with both screen supersampling and intersection verification. Figure 6 shows an image of Whitted's classical spheres and checkerboard where the true intersection point and the true normal vector are used for image generation with supersampling.

It should be observed that although we maintain now the geometric information about the objects, we remain loyal to the 3D raster approach in which a voxel is assumed to be atomic in the sense that it contains information only about one object. This assumption is similar to that of 2D rasters in which a pixel maintains only one color entity. In contrast, space-subdivision methods (e.g., ARTS) assume that each cell maintains a list of geometrically defined objects. This distinction poses some limitation on scene density in RRT in the sense that image quality will degrade as the scene becomes too crowded relative to the voxel size and many objects tend to occupy the

(a)

(b)

Figure 4: (a) A 820-sphereflake of 320^3 resolution generated with 6-rays RRT in 74.9 seconds. (b) The same scene ray traced with true intersection verification in 85.0 seconds.

(a)

(b)

(c)

(d)

Figure 5: A comparative ray tracing of 91-sphereflake in 320^3 resolution: (a) was ray traced in 72 seconds with the basic RRT algorithm. RRT with supersampling yielded (b) in 342 seconds. RRT with intersection verification produced (c) in 75 seconds, while intersection verification with supersampling yielded (d) in 352 seconds.

same voxel.

In summary, comparing RRT to existing ray tracing methods, RRT suffers from large memory requirement or some limitation on scene density. On the other hand, the primary advantages of RRT, even when it employs intersection verification, include its practical independence of scene complexity (i.e., number of objects), the ability to ray trace sampled and computed datasets, the support of voxblt and CSG operations, and the ability to precompute all the view independent attributes. When RRT does not perform intersection verification its performance is also insensitive to surface complexity (i.e., type of surface).

Results

The results described here were obtained by running our RRT software on one 20MIPS (25MHz) processor of an SGI 4D/240GTX and are reported in CPU time. Similar results were obtained on HP 400s and Sparcstation-1 workstations. Images were generated on 80M and 128M byte machines from 256^3 and 320^3 spatial resolution 3D rasters. All ray tracings were done with two generations of secondary rays and shadow rays to two light sources. Shadow rays were not precomputed, which could have further improved the reported results by about 20% for two light sources.

Table 1 shows rendering time of various 256^3 resolution test scenes using the basic RRT algorithm. The table indicates that RRT is in practice insensitive to the number or type of objects. Our 256^3 resolution test images are ray traced in less than forty seconds, almost with no degradation in performance even when image complexity increases or when surface type changes. For example, although there is an order of magnitude increase in the number of objects ray traced for the 91-, 820-, and 7381-sphereflake images, the performance degrades very slowly (26.5, 29.3, and 38.4 seconds, respectively). This slight slowdown in speed is *not* caused by the increase in the number of objects but by the inflated number of rays bouncing between the many tiny structures of this fractal-like scene instead of terminating by reaching the 3D raster boundary.

As discussed above there is a significant difference between the length of the 26- and 6-rays. A test image was ray traced with 26-rays in 32.3 seconds, 23.5 of which (73%) were spent in the 26-ray traversal algorithm. The same image took 56.2 seconds in 6-connected tracing, which in turn spent 47.3 seconds (85%) in the 6-ray traversal algorithm. Similar results were observed from various viewpoints, with a speedup factor of up to 1.84 for the traversal of 26-rays. We also discovered that less than 1.5% of the 26-rays missed a surface voxel pierced by a 6-ray. A performance comparison

Table 1: Rendering time of the basic RRT of various 256^3 resolution scenes.

Scene	Seconds
91-sphereflake	26.5
820-sphereflake	29.3
7381-sphereflake	38.4
MRI head	28.1
Newell’s teapot	36.8

between the 26-ray algorithm and the adaptive algorithm based on proximity bit showed a time penalty of less than 2.9% while completely eliminating the “miss” problem. Voxelization with proximity flags does not require any additional computations, only more memory-write operations.

The performance of the RRT algorithm when employing intersection verification does depend on the type of surfaces comprising the scene. In the case where surfaces are easy to intersect (such as spheres), no significant performance penalty was observed; ray tracing a 320^3 volume consisting of a 91-sphereflake with and without intersection verification took 75 and 72 seconds, respectively. In terms of image quality, the number of rays that benefit from intersection verification by eliminating false hits depends on the scene and on the complexity of its silhouettes. For example, in the 91-, 820-, and 7381-sphereflake images, 1.72%, 5.06%, and 7.94% of the rays, respectively, experienced a false hit that was remedied by intersection verification.

For purposes of comparison, we have implemented and tested the performance of the uniform space subdivision algorithm ARTS [5] for tracing several scenes. These scenes were generated by randomly distributing an increasing number of spheres inside the 3D raster. The top row of Table 2 shows that indeed voxelization time grows linearly to the scene complexity. The results, shown in the bottom row of Table 2 and by the solid graph in Figure 7, indicate that once the scene gets relatively crowded, RRT’s ray tracing time might even decrease because the total space traversed by the rays gets shorter. ARTS, on the other hand, still performs intersection calculation in each cell and therefore remains sensitive to the scene and object complexity. When ARTS subdivision resolution equals that of RRT 3D raster, the costly intersection calculation is performed by ARTS just for the sake of distinguishing between multiple objects in a single voxel – an infrequent phenomena in high resolution rasters and common scenes. On the average, when comparing both methods under the same parameters (e.g., both

Figure 6: Turner Whitted's spheres and a checkerboard floor in 320^3 raster resolution ray traced in 377 seconds with intersection verification and supersampling.

Figure 7: A comparison between RRT and ARTS. The dashed lines show ARTS ray tracing time employing 40^3 (ARTS40) and 120^3 (ARTS120) uniform space subdivision. The scene was composed of randomly distributed spheres of radius 10 in a 256^3 world.

Table 2: *Voxelization and discrete ray tracing times (in seconds) of a 256^3 resolution scene with increased population of spheres.*

No. of spheres	10	100	200	400	1000	5000
Voxelization	0.07	0.74	1.49	2.98	7.46	36.01
Ray tracing	20.2	23.9	30.1	30.4	28.9	21.5

operate in the same space resolution and same number of spheres), the RRT method is 9.1 times faster than ARTS. In reality, ARTS does not require that its cell footprint will be larger than a pixel and therefore can more efficiently operate in lower space resolutions. The upper two dashed lines in the graph of Figure 7 show the degradation in ARTS performance for 40^3 and 120^3 space subdivisions. It should be noted that this example is favorable to ARTS since ray-sphere intersection calculation is simple, and if teapots or fractals, for example, were used instead, the difference would have been much larger.

Conclusions

We have presented a new ray tracing approach, RRT, which is based on a 3D raster representation of the scene and a discrete ray traversal. RRT, in its basic version, ray traces voxelized scenes in record speeds, at the price of negligible degradation in image quality. The image can be *progressively refined* by employing the more sophisticated variations of the algorithm that employ supersampling and intersection verification. The processing time of the most elaborate variation of the algorithm, is still attractive compared to existing techniques, with image quality comparable to that of contemporary techniques. RRT is especially suitable for very complex scenes and constructive solid models, exploiting voxel representation for ray tracing geometry. RRT also offers the use of ray tracing for photorealism of sampled and computed voxel datasets, or mixtures thereof with synthetic models.

Acknowledgments

This project has been partially supported by the National Science Foundation under grants MIP-8805130 and IRI-9008109, and by a grant from Hewlett Packard. We would like to thank Dan Gordon, Pat Hanrahan, Marc Levoy, and Nelson Max for reviewing the manuscript and for their helpful suggestions, and Qiang Zhang for her help in implementing parts of RRT.

References

1. Amanatides, J. and Woo, A., "A Fast Voxel Traversal Algorithm for Ray Tracing", *Proceedings of EUROGRAPHICS '87*, Amsterdam, The Netherlands, August 1987, 3-9.
2. Blinn, J. F., "Light Reflection Functions for Simulation for Clouds and Dusty Surfaces", *Computer Graphics*, **16**, 3 (July 1982), 21-29.
3. Cleary, J. G. and Wyvill, G., "Analysis of an Algorithm for Fast Ray Tracing using Uniform Space Subdivision", *The Visual Computer*, **4**, (1988), 65-83.
4. Cohen, D. and Kaufman, A., "Scan-Conversion Algorithms for Linear and Quadratic Objects", in *Volume Visualization*, A. Kaufman, (ed.), IEEE Computer Society Press, Los Alamitos, CA, 1990, 280-301.
5. Fujimoto, A., Tanata, T. and Iwata, K., "ARTS: Accelerated Ray-Tracing System", *IEEE Computer Graphics and Applications*, **6**, 4 (April 1986), 16-26.
6. Kajiya, J. T. and Von Herzen, B. P., "Ray Tracing Volume Densities", *Computer Graphics*, **18**, 3 (July 1984), 165-174.
7. Kaufman, A. and Shimony, E., "3D Scan-Conversion Algorithms for Voxel-Based Graphics", *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, Chapel Hill, NC, October 1986, 45-75.
8. Kaufman, A., "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes", *Computer Graphics*, **21**, 4 (July 1987), 171-179.
9. Kaufman, A., "An Algorithm for 3D Scan-Conversion of Polygons", *Proceedings of EUROGRAPHICS'87 Conference*, Amsterdam, The Netherlands, August 1987, 197-208.
10. Kaufman, A., Yagel, R. and Cohen, D., "Intermixing Surface and Volume Rendering", in *3D Imaging in Medicine. Algorithms, Systems, Applications*, K. H. Hoehne, H. Fuchs and S. M. Pizer, (eds.), Springer-Verlag, 1990, 217-227.
11. Kaufman, A., "The voxblt Engine: A Voxel Frame Buffer Processor", in *Advances in Graphics Hardware III*, A. A. M. Kuijk and W. Strasser, (eds.), Springer-Verlag, Berlin, 1992.
12. Levoy, M., "Display of Surfaces from Volume Data", *IEEE Computer Graphics and Applications*, **8**, 5 (May 1988), 29-37.
13. Mokrzycki, W., "Algorithms of Discretization of Algebraic Spatial Curves on Homogeneous Cubical Grids", *Computers & Graphics*, **12**, 3/4 (1988), 477-487.
14. Tiede, U., Hoehne, K. H., Bomans, M., Pommert, A., Riemer, M. and Wiebecke, G., "Investigation of Medical 3D-Rendering Algorithms", *IEEE Computer*

Graphics & Applications, **10**, 3 (March 1990), 41-53.

15. Woo, A. and Amanatides, J., "Voxel Occlusion Testing: A Shadow Determination Accelerator for Ray Tracing", *Proceedings of Graphics Interface'90*, 1990, 223-230.
16. Yagel, R., Kaufman, A. and Zhang, Q., "Realistic Volume Imaging", *Proceedings of Visualization'91*, San Diego, CA, October 1991, 226-231.

Biographies

Roni Yagel is an Assistant Professor in the Department of Computer and Information Science at The Ohio State University. Previously he was a researcher in the Department of Anatomy and the Department of Physiology and Biophysics in the State University of New York at Stony Brook. His research interests include algorithms for voxel-based graphics, imaging, and animation, three dimensional user interfaces, hardware for volume viewing, and visualization tools for biomedical applications.

He received his PhD in 1991 from the State University of New York at Stony Brook. He received his B.Sc. Cum Laude and M.Sc. Cum Laude from the Department of Mathematics and Computer Science at Ben Gurion University of the Negev, Israel, in 1986 and 1987, respectively.

His address is:

Department of Computer and Information Science
The Ohio State University
228 Bolz Hall
2036 Neil Av. Columbus, Ohio 43210-1277
Phone: (614) 292-0060, Fax: (614) 292-9021
email: yagel@cis.ohio-state.edu

Daniel Cohen is lecturer at the Department of Computer Science in Ben-Gurion University and at the School of Mathematics at Tel-Aviv University. Currently, he also developing a real time ray tracer of terrain systems at Milikon, Ltd. In 1987 he was a software engineer at Afkon, Ltd. working on bitmap graphics.

His research interests include rendering techniques, volume visualization, architectures and algorithms for voxel-based graphics. He received a BSc Cum Laude in both Mathematics and Computer Science (1985), an MSc Cum Laude in Computer Science (1986) from Ben-Gurion University, and a Phd from the Department of Computer Science (1991) at State University of New York Stony Brook.

His address is:

Department of Computer Science

School of Mathematics

Tel-Aviv University, Ramat Aviv, ISRAEL

Phone: +971 3 545-0037, Fax: +971 3 640-9347

email: daniel@math.tau.ac.il

Arie E. Kaufman is a Professor of Computer Science and the director of the Cube project for volume visualization at the State University of New York at Stony Brook. He has conducted research and consulted in computer graphics for 18 years specializing in volume visualization; graphics architectures, algorithms, and languages; user interfaces; and scientific visualization.

Kaufman received a BS in Mathematics and Physics from the Hebrew University of Jerusalem in 1969, an MS in Computer Science from the Weizmann Institute of Science, Rehovot, in 1973, and a PhD in Computer Science from the Ben-Gurion University, Israel, in 1977. He is a member of ACM, SIGGRAPH, IEEE-CS, IEEE-EMBS, and Eurographics. Currently, he is the chair of the IEEE Computer Society Technical Committee on Computer Graphics.

His address is:

Department of Computer Science
State University of New York at Stony Brook
Stony Brook, NY 11794-4400
Phone: (516) 632-8441, Fax: (516) 632-8434
email: ari@cs.sunysb.edu.