In recent years a rendering technique known as ray tracing has firmly established itself as a tool of the computer graphics community.1,2 For those who are not familiar with the technique, Rogers3 can be recommended as an excellent introduction.

Ray tracing is currently well known for providing the highest quality of image synthesis. The superior image quality achieved by ray tracing—which uses a global illumination model rather than the local illumination models used by traditional scan-line algorithms—is perhaps most evident when corresponding images produced by the different algorithms are compared.3 Despite the impressive images, however, many improvements can still be made to further upgrade the image quality. These improvements include more realistic illumination models,3 antialiasing,3,9 fuzzy shadows and dull reflections,10 and diffuse reflection from distributed light sources.11 Hence much of the research effort in ray tracing is devoted to these problems.

However, in a cost/performance comparison with traditional, mostly scan-line methods, ray tracing appears to be seriously handicapped. Roth6 noted that many experts in the CAD/CAM field have doubts about the sufficiency of ray casting and consider it an impractical, brute-force method. The calculation speed of the ray-tracing method is undoubtedly one of the basic problems that must be dealt with. Until we solve this problem, we cannot expect ray tracing to achieve the widespread use that the scan-line methods enjoy, even though the latter produce far inferior images.

Why is ray tracing so computationally expensive? The main cause was clearly identified at the very moment ray tracing first entered the field of computer graphics. According to Whitted,2 for simple scenes 75 percent of the total time is spent on calculating intersections between rays and objects. For more complex scenes the proportion goes as high as 95 percent. The time that must be spent calculating the intersections is directly
related to the number of objects involved in the scene.

Scan-line efficiency is achieved by well-established methods such as incremental calculation of geometry based on object and/or image coherency. Various forms of coherency commonly exploited in other rendering methods unfortunately are not easy to exploit in the case of ray tracing. In general each ray is traced independently without taking advantage of the information calculated previously for its neighbors.

Much recently reported research has concentrated on speeding up the computations involved in ray tracing. The attack has been concentrated mainly on reducing the complexity of intersection calculation. (The present article is no exception in this respect.) It is perhaps worthwhile to emphasize that virtually all the published approaches are based on a single common notion, which is that of associating a bounding volume or extent with each object in the scene.

To reduce the number of surfaces that must be checked against a given ray, the ray is first checked against gross volumes, or extents, that bound the objects. Extents have been in use since ray tracing was first introduced and have usually been specified in the hierarchical data structures describing the environment. This method, initiated by Rubin and Whitted, has now become common practice in ray tracing.

The differences among the various approaches based on the application of extents stem from the following aspects:

- shapes of the extents
- hierarchical relations between extents
- level of coherency that is exploited

These aspects are naturally interrelated: Certain shapes preclude the use of certain hierarchies and certain hierarchies preclude the use of certain forms of coherency. A discussion of variously shaped extents and the advantages of each shape can be found, for example, in Hall and Greenberg.

Most of the early developments were based on (1) object coherency, using a hierarchical object description, or (2) hierarchical clustering of objects, either provided by the application or automatically generated during the rendering process. This approach produced certain improvements in calculation speed, but it did not represent the kind of breakthrough that could reduce the rendering time to the level achieved by traditional image synthesis methods.

A hierarchy based on the octree data structure in a form suitable for ray tracing was first advocated by Fujimura et al. and almost simultaneously by Matsubara and Murakami. It was later also used by Glassner. Recently Kaplan has used a data structure called BSP but having essentially the same fundamental properties as the octree. The results presented in these papers show that exploiting octree coherency yields much better results than previous methods in which the hierarchy was based directly on object coherency. In general, an improvement in speed of an order of magnitude has been achieved.

Identifying and exploiting various forms of coherency is one of the most fundamental aspects of computer graphics. Practice shows that it is a basic precondition for efficient execution of computer graphics programs. Exploiting coherency in its ultimate form leads to algorithms based on an incremental application of very simple operations that in many cases can use "integer logic" to replace time-consuming floating-point arithmetic. Incremental techniques, by nature very elegant and simple, are generally made feasible in computer graphics by the existence of various forms of coherency.

We felt that the previously published results did not yet represent the maximum level to which the coherency of an octree hierarchy could be exploited. As a next step in this direction we proposed a new, totally incremental tool for traversing the octree structure: Three Dimensional Digital Differential Analyser, or 3DDDA.

We also felt that the octree structure, while being a function of object shape and scene topology, is not necessarily the form of auxiliary structure that provides the optimum degree of coherency. This is because ray tracing in an octree involves complex processing for traversal of the tree structure. This led us to adopting and investigating the structure we call SEADS (Spatially Enumerated Auxiliary Data Structure). The structure of SEADS is actually completely independent of object shape and topology and thus fails to take any advantage of object coherency. On the other hand, the level of coherency of this auxiliary structure corresponds to the level of coherency that the raster grid provides for a conventional DDA (Digital Differential Analyser) line generator.

---

**SEADS provides an environment for ray tracing that outpaces the hybrid octree approaches ... by an order of magnitude.**

Various experiments have proved that the combination of 3DDDA and SEADS provides an environment for ray tracing that outpaces the hybrid octree approaches presented in previous papers by an order of magnitude. Various experimental results have shown that the rendering time is virtually independent of the number of objects in the scene. When the number of objects is very large, ray tracing—despite its reputation for inefficiency—actually becomes faster than other rendering methods.

**3DDDA**

The purpose of this section is to describe 3DDDA, the basic tool for traversing SEADS. The extension of 3DDDA that enables it to traverse the octree will be discussed after the octree structure is introduced.

We start with the remark that SEADS can be thought of as a 3D extension of the raster grid, with pixels becoming voxels. Voxels in SEADS are orthogonal cuboidal cells. The inherent 2D coherency of a raster grid is thus naturally extended to 3D. Such an extension of a data structure calls for an extension of the tool that traverses it—the line generator. As described in any introductory text on computer graphics, a line generator can be thought of as a very efficient tool for identifying rectangular pixels pierced by a 2D line placed on a raster grid. Modifying the basic algorithms in order to identify not some but all pixels pierced by the line is fairly straightforward. The extension of this notion to the identification of SEADS voxels pierced by a ray (3D line) is obvious and does not require further elaboration. We proceed directly
to technicalities.

The line generator can be implemented in many ways (see Rogers\textsuperscript{,} pp.30-39, and Fujimoto and Iwata\textsuperscript{22}), but it works basically as follows: During the process of generating each consecutive pixel's coordinates, the coordinate corresponding to the driving axis (DA) is unconditionally incremented by one unit. The DA (or "axis of the greatest movement") is determined by the slope of the line and is one of the coordinate axes (Figure 1). At the same time a control term—an "error term" which is traditionally measured perpendicular to the DA—is updated by subtracting from it the slope value and then checking whether it is still smaller than half the pixel size. When this test fails, a unit increment (or decrement) of the coordinate perpendicular to the DA is performed. The control term is corrected by adding the value corresponding to one pixel whenever underflow occurs.

The process described above generates the center coordinates of those pixels in closest proximity on the left or right side of the mathematical line. The objective of a DDA used for line generating is to enumerate those pixels that lie close to the true line. While a line may be displayed using many different pixel configurations, one configuration is usually preferred. The configuration corresponding to the optimal line is influenced by factors such as uniform pixel intensity or brightness.\textsuperscript{22} In general the pixels representing the optimal line, in addition to being the closest to the true line, also satisfy a condition that only one pixel is generated for each vertical column (Figure 1).

On the other hand, the objective of a DDA used for cell identification purposes is to enumerate those cells that are pierced by the straight line (Figure 1). Two rather simple modifications are necessary to convert the above scheme into one that can identify the pixels through which the line passes. First, movement perpendicular to the DA, which in the standard DDA is coupled with that of the DA, must be separated from it in order to ensure that the DDA always goes from one pixel to an adjacent one without skipping any on its way. Second, the threshold value against which the control term is checked must be made zero instead of half of the pixel size. With these modifications the DDA, instead of generating pixel coordinates closest to the center line, will generate the coordinates of all the pixels through which the line passes.

The above scheme, however, will work only if the traced ray's origin and end point are located at the centers of cells. This obviously is not generally the case. Here we cannot overemphasize the proper initialization of DDA. In Fujimoto and Iwata\textsuperscript{22} such initialization was necessary to provide higher addressability, going beyond that of the raster memory resolution; here it is essential for proper cell identification. Notice that the control term can also be measured in the direction parallel to the DA (Figure 1). Its units will then correspond to pixel units, and the range of the control term will correspond to the inverse value of the slope. Actually this approach was adopted to generate smooth vectors,\textsuperscript{22} when a control term measured directly in gray-scale units along the DA was used for generating the pixel's coordinates and intensity (for antialiasing). In the present application, however, measuring the control term along the DA is vital because it corresponds to the cell size unit. This control term is essential in providing the proper order of cell entry in the case of the 3D cuboidal grid. Before explaining that, let us first extend the notion of the DDA to three dimensions.

One way to realize 3DDDA is to use two synchronized DDA's working in mutually perpendicular planes that intersect along the DA (Figure 2). In each plane the modified DDA explained above pursues the projection of a 3D line onto that plane. Now, since both control terms are measured along the same DA in units of the cubic size, it is clear that it is possible to provide proper identification of the string of cells pierced by 3DDDA simply by first processing that plane on which the smaller control term is generated. Later we shall see that expressing the control term in cell-sized units is also essential in efficient 3DDDA traversal of the octree. 3DDDA works very much like the two-dimensional DDA. After an unconditional step in the DA direction is executed, control terms in both planes are processed, and movement to the neighboring cell is performed if necessary. A special situation arises when such movement is necessary in both planes simultaneously (Figure 2). This means that on

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure1.png}
\caption{DDA (Digital Differential Analyzer) for line generation in two dimensions.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure2.png}
\caption{3DDDA (Three-Dimensional Digital Differential Analyzer) for ray tracing in three dimensions.}
\end{figure}
a particular step, both passive indices of a cell must be updated. If this operation is performed in the wrong order, an erroneous cell will be identified. The code for initialization and for the main part of the algorithm is very similar to the one presented in Fujimoto and Iwata.24 The only difference is that the control term is initialized and calculated not in gray-scale units but in cell-size units.

3DDDA is applied along the ray direction, and it directly identifies all three indices of the cell. Calculation of coordinates of the intersection with the cell mesh could easily be added, but it is not necessary for cell identification. Because incremental logic is inherent to 3DDDA, all intersections (with the exception of initialization) are processed without any multiplication or division.

Each time the 3DDDA hits any of the planes, the next cuboid representing a mesh element is identified and checked for being in HETERO state. If this is the case, then all of the segments it contains are checked for possible intersection with the ray. If an intersection is found, either ray spawning is performed or the particular branch of the shade tree is terminated, depending on the control parameters and/or model attributes. Otherwise (if HOMO or ray does not intersect any of the segments within the cuboid) 3DDDA continues pursuing the ray in the same direction until some object is intersected or until it leaves the mesh domain.

In conventional ray-tracing programs, increasing the complexity of the model or the number of objects usually results in a situation where a single ray pierces a considerable number of objects. This in turn requires searching for the surface nearest to the observer, or possibly to a spawned ray's origin. This is one of the factors contributing to the well-known phenomenon of ray tracing: Calculation time exhibits exponential growth with the complexity and number of objects in the scene. Here, it is worth noting that 3DDDA traces only objects relevant extents (cuboids), and it traces them in the appropriate consecutive order. No global sorting for hidden points is necessary. Local sorting with a rather limited number of items is occasionally necessary when more than one segment is hit within a single cuboid. In general, though, the number of cells containing more than one element will tend to decrease with increasing resolution of the mesh.

The octree

The octree encoding scheme is similar to the cell decomposition (spatial enumeration) method explained above. The information contained in the octree-encoded representation of a scene is identical to that available in the cell decomposition. From the storage point of view, however, the data are stored in a hierarchical tree structure with nodes representing disjoint cuboids of geometrically decreasing size. Each node of the tree corresponds to a region of the scene and has one or more values that define the region. If the value of the node completely describes the region, it is a terminal or a leaf. If not, an ambiguity exists and the node points to the eight children that represent eight subregions or octants of the parent node (Figure 3). In general the octree representation can be expected to take advantage of the spatial coherence found in most objects.

In this context let us analyze more closely the situation presented in Figure 3, which is actually a 2D quadtree. In this particular case it is evident that the total number of nodes in the quadtree encoding and the number of cells in simple spatial enumeration happen to be equal up to the level 3. This observation is important because in general it is more time-consuming to retrieve information from or traverse the octree than it is with the cell-decomposed structure. In this particular case it is evident that the difference between the total numbers of nodes and cells is expected to become more and more significant as the resolution is increased. However, even if it is assumed that this is generally the case, it will be difficult to predict the resolution at which the octree-encoded structure becomes superior to that of simple cell decomposition.

What resolution can we reasonably afford? Meagher,25 advocating usage of the pure octree, points out that the main disadvantage of the encoding technique is the large memory requirement. He presents proof that the quantity of memory required to store a 2D quadtree object is of the order of the perimeter of the object. Similarly, the memory and processing computation for a 3D object is on the order of the surface area of the object. Depending on the object and the resolution, this can still represent a large storage requirement. Several million bytes of node storage may be necessary to represent realistic situations.25 Such memory size will not always be justified or even feasible.

The seriousness of this problem was also recognized by Kunii et al.19 The octree is an approximation of a smooth-surfaced object by small cuboids, so it is inevitable that
Octree encoding. The tree structure encoding will be explained by an example (Figure 4). For simplicity a quadtree is used in the example. Extension to the case of an octree is trivial. The scene consists of six objects numbered 0 to 5. The node arrangement and the corresponding tree levels are shown on the upper part of the figure. The legend below explains the meaning of the node status code. Information concerning the encoded tree structure is arranged in two one-dimensional arrays: NODE STATUS and POINTER. POINTER, depending on NODE STATUS of the particular entry, can contain either (1) a pointer to another entry in array POINTER itself, as is the case for a HETERO-BRANCH node, or (2) a pointer to object ARRAY, as is the case for the nonhomogeneous terminating node HETERO-LEAF. For HOMO nodes the contents of array POINTER are irrelevant. This arrangement enables 3DDDA to traverse the structure quickly, as will be explained in the next section.

Traversing the octree. In optimizing traversal of the octree structure by 3DDDA, special attention must be paid to the ordering of the octants of a node. The order adopted in this paper deviates from the conventional definition and follows instead the systematic order proposed by Yamaguchi et al. and shown in Figure 5. In this arrangement each digit of the binary node number corresponds to a cell index in the cell decomposition of a node. Since for a single node there exist exactly two octants in each principal direction \( x, y, z \), all three indices satisfy the condition \( 0 \leq i, j, k \leq 1 \). 3DDDA traverses a single node exactly as it traverses the cell-decomposed structure. When moving from one cell to an adjacent cell, it updates one of the three indices. Changing an index results directly in producing the number of the adjacent octant entered by 3DDDA.

The above description concludes the explanation of how the 3DDDA traverses octants in a node. Since this traversing takes place on a single level, let us call it a horizontal traversal. Horizontal traversal will be terminated when any of the three indices overflows or underflows. This corresponds to the situation when a ray leaves a node. In order to identify the adjacent node it is necessary to ascend the tree. This will be termed vertical traversing of the octree. Vertical traversing can also take place in the form of descending the tree, which is necessary each time 3DDDA identifies a HETERO-BRANCH.

<table>
<thead>
<tr>
<th>Quadtree level</th>
<th>Number of nodes</th>
<th>Total number of nodes</th>
<th>Resolution</th>
<th>Number of cells in spatial enumeration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
<td>16</td>
<td>4</td>
<td>16</td>
</tr>
<tr>
<td>3</td>
<td>48</td>
<td>64</td>
<td>8</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>164</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>5</td>
<td>260</td>
<td>424</td>
<td>32</td>
<td>1024</td>
</tr>
</tbody>
</table>

Figure 4. Quadtree representation.

the encoded object acquires some notched surfaces. In order to avoid displaying a jagged surface, the object must be represented by a very deep octree. A complex object with reasonably high resolution, then, requires enormous data storage and a high-speed processor. This means the most important advantages of the octree, namely processing speed and memory economy, are lost. The solution proposed by Kunii et al. is basically hybrid. Instead of using a pure octree structure, they proposed combining the octree representation with the surface model.

Another example of a hybrid approach to tree encoding, applied to FEM mesh generation, is presented by Yerry and Shephard. The hybrid approach is also adopted in the present paper: CSG versus octree (and cell decomposition). The hybrid approach appears at least for the time being to be more realistic than pure octree encoding or simple cell decomposition. However, the question of how much resolution can be justified still remains. A general representation depicting a solid body as a 3D array typically occupies \( 1000 \times 1000 \times 1000 \) cells. The maximum level value used by Kunii et al. was 10. This results in a similar resolution \( (2^{10} = 1024) \). Superiority of the octree over simple cell decomposition will be decided by experiment. However, even at high resolution it is possible to envisage a scene with many objects and low homogeneity for which the octree structure will not necessarily be justified.

Figure 5. Systematic order of octants.
node. The octree must be traversed downwards (sometimes for several levels) until a HETERO-LEAF or HOMO node is reached.

Due to space limitations it is impossible to explain here all the details concerned with the vertical traversing of the octree. The important fact is that vertical traversing of the octree is performed by making use of the 3DDDA byproducts. During descent of the tree corresponding pointers from array POINTER (Figure 4) are recorded in a separate short working array. The length of this array is equal to the number of levels in the tree. The pointer array is not linked backwards (upwards) and the pointers recorded in the working array are very helpful in ascending the tree. 3DDDA execution is suspended during vertical traversal. During descent of the tree the current values of its variables—specifically both control terms, the inverse slopes, and the octant size—determine which child of the node has to be entered. These values must be multiplied or divided by two on each vertical movement from one level to the next, but in fixed-point arithmetic this operation is simpler than, for example, an addition.

In summary we can conclude that, during horizontal traversal of the octree, 3DDDA automatically generates all necessary variables without floating-point division or multiplication, exactly as it does when traversing the SEADS. Thus, no modifications in 3DDDA are necessary for the octree. Horizontal traversal is identical to the case of simple cell decomposition and vertical traversal is realized by making use of 3DDDA byproducts.

**Results.** All algorithms explained in the previous sections have been implemented in FORTRAN 77. Initial experiments were performed on a Digital Equipment Corporation VAX 11/750 (4M-byte CPU) under the VMS operating system.

**3DDDA.** For 3DDDA, the computational cost of moving from one cell to another was initially 72.3 microseconds. After further optimization—which involved removal of all subroutine calls, dimension declarations, and indices, and introduction of integer logic into the main part of the algorithm—this cost was further decreased to 20.6 microseconds. Our current 3DDDA is thus able to identify the cell index more than 13 times faster than Glassner’s proposed traversal algorithm\(^\text{11}\) can find coordinates to a next cell. (We implemented Glassner’s simple spatial enumeration algorithm for comparison purposes, and it requires 276.2 microseconds for that operation.)

When 3DDDA is used it has to be initialized once for each ray, a process requiring 0.591 milliseconds. For 512 \(\times\) 512 resolution, it will take less than three minutes to initialize 3DDDA for all pixels in the screen. (See the starting point of line A in Figure 6.) Line A shows the total overhead time, which varies with the number of cells. It should be obvious that this time is not dependent on the number of objects, or on the shape of the objects, or on the degree of coherency of the objects in the scene.

On the other hand, overhead time for the octree is quite heavily dependent on all these factors. To compare the overhead time for octree encoding and spatial enumeration, the total number of cells generated by the octree encoding program was recalculated to find the average number of cells along one axis. All the overhead times indicated in Figure 6 are for the situation in which all rays are traced throughout the encoded domain without terminating on pierced objects. Line C in Figure 6 represents overhead time for the original object (see Figure 7). Line G corresponds to the scene in which all 7011 atoms of the original model have been dispersed uniformly by the use.
of random numbers.

It is clear from the above results that the octree structure is very sensitive to lack of coherency and is put at a further disadvantage in comparison to spatial enumeration. It was expected that traversing of the octree structure would be more expensive than spatial enumeration. However, it was also expected that octree encoding would have an advantage for cases where high scene coherency results in a large percentage of empty areas. (The object in Figure 7 occupies less than 9 percent of the hexahedron domain in which it is embedded. In spatial enumeration it is possible to tailor the overall domain dimensions to fit the shape of the scene. This will result in a smaller number of cells. However, for comparison purposes the same hexahedron domain is used for spatial enumeration and octree encoding.) In octree encoding smaller cells are converging to the object surface and empty areas are encoded in bigger cells. This means that in comparison to spatial enumeration a ray can reach the surface of an object by traversing fewer cells, and the probability of hitting an object within the HETERO cell is bigger. Experimental results, however, suggest that all this was outpaced by the cost of vertical traversing of the octree. As was noted in the previous section, vertical traversing must be performed after at most four cells are identified through 3D DDA during horizontal traversing. But the average frequency of vertical traversing is much higher, for it may be needed as soon as one horizontal step is performed. Depending on

the depth of the octree, it may quite often be necessary to perform several steps of ascending and descending the octree.

From the above discussion it should be clear that, with respect to speed of calculation, the overhead time precludes any advantage of octree over spatial enumeration. This was confirmed by experiment. (Compare lines B and E with D and H in Figure 6.) Experiments have shown that there exists a clear optimum number of cells in spatial enumeration and an optimum number of levels for octree encoding for which the calculation time reaches its minimum. This minimum happens to correspond to a relatively limited number of cells that can be handled easily by most of the contemporary minicomputers or workstations. We performed a considerable number of experiments in order to find how this optimum number of cells is affected by such factors as scene coherency, object size, and number of objects. It is impossible to present all timing results here, but the results presented in Figure 6 were found to be fairly representative. In particular we have found that although scene coherency to some extent influences the overall calculation time—as does the object size—it does not influence the optimum number of cells. This optimum is slightly influenced by the number of objects and goes to the left (smaller number of cells) with decreasing numbers of objects.

The above experiments provide an important hint to those who advocate a pure octree with a considerable number of levels. Unless the octree-encoded structure can be traversed rapidly enough, pure octree encoding may not necessarily speed up the calculations. (It should be stressed here that we are limiting ourselves to ray-tracing applications. Each ray is traced independently, one for each pixel on the screen.) In the present application the hybrid approach proved essential in obtaining considerable speed improvement. Obviously traversing speed is influenced by more than just the efficiency of the algorithm. It can be assumed that when the same program is run on a supercomputer or special-purpose hardware, a more than proportional improvement in speed would result; the overhead time decrease would move the optimum further to the right, resulting in better performance.

Enumeration. When comparing SEADS with the octree approach, we should also discuss the encoding time. This time is an order of magnitude shorter when SEADS is used. Encoding the DNA model shown in Figure 6 took 17 seconds in SEADS, while octree encoding took over three minutes (207 seconds). In both cases reencoding is necessary only when the topology of the objects changes; reencoding is not necessary when the whole scene undergoes rigid body transformations (rotation and translation) or when the viewpoint is changed. Nevertheless, even when encoding has to be performed, the time required by SEADS can be virtually ignored; it is just a fraction of the rendering time even for very large numbers of objects.

B-reps. Further experiments have shown that the calculation time is only very slightly influenced by the number of objects in the scene, as illustrated by Figure 8. (Actually, this was the goal of the present research.) As a next step in evaluating the proposed algorithm and data structure, a comparison with scan-line method was performed. The popular general-purpose computer graphics package MOVIE.BYU was used. Objects were changed from spheres to hexahedrons. MOVIE.BYU works with the boundary representation scheme, so approximating
the sphere by a number of polygons would put it at a
disadvantage.) Results are presented in Figure 9. Unfor-
tunately, the internal limitations of MOVIE.BYU are such
that, unless the number of bits in certain data structures
is increased, it cannot handle more than 8192 polygons.
This corresponds to 1365 hexahedrons. From simple
extrapolation, however, it can be deduced that the tech-
nique presented here, Accelerated Ray Tracing (ART), is
as fast as MOVIE.BYU when the number of objects in the
scene approaches 1600. Beyond that limit, ART actually
becomes faster. For example, 16,000 hexahedrons are
calculated within 17 minutes by ART, whereas it would
take about two hours to calculate the same scene using
MOVIE.BYU.

The experimental results above suggest that ART can
be used for fast rendering of models approximated by
polygonal facets or tiles, as in the B-rep (boundary rep-}
resentation) method. This kind of representation is actually
used in the majority of solid modelers used in commercial
CAD/CAM systems. In order to make direct rendering of
B-reps possible, we have implemented a nonsolid primi-
tive: polygon.

A B-rep model of one of the newest cars made by
Mazda, the RX-7, has been used for timing experiments.
This model is approximated by 46,000 polygons (Figure
10). Less than one-and-a-half minutes was needed to
enumerate this amount of data in SEADS using a VAX
11/785. It took 25 minutes for our ART system to syn-
thesize the image. It should be noted here that rendering
one object (polygon) that covers approximately the same
area on the screen requires about 7.5 minutes.

Unfortunately, we are unable to provide any experi-
mental comparison, since none of our scan-line-based
rendering programs can handle this amount of data.
(MOVIE.BYU, for example, can handle only a fraction of
it.) However, some comments are in order here. When an
object involving tens of thousands of polygons is dis-
played on a relatively low-resolution screen (512 × 512 in
this example), most of its individual polygons affect very
few pixels—very often only one—and there are some
cases where the polygons are simply too small to be dis-
played. In such a situation, it becomes virtually impos-
sible to use object coherency: Various increments are
calculated for each polygon but never used, or used
so few times that incremental calculation cannot be
justified at all. For this reason we expect that it would be
difficult for any renderer based on an incremental
method to reach the performance cited here.

We are aware of the fact that the above-cited timing
data must be modified if calculation is performed for
higher resolution. Our results on antialiasing, however,
suggest that there is hardly any need to go to a resolution
higher than 512 × 512, at which all images presented in
this article were calculated and displayed. In the section
on antialiasing we noted that processing for antialiasing is
in general proportional to the total length of the edges.
This might suggest that in our particular example involve-
ing 46,000 polygons, the total length of the edges will be
so great that the processing time for antialiasing becomes
intolerable, but this is not in fact the case. An edge as
processed by our antialiasing algorithm does not neces-
sarily correspond to the edge of the geometric entity; an
dge is detected only where the change in intensity is
greater than a certain threshold at which perceptible
aliasing occurs. In our example most of the actual geo-
metric edges are simply not processed, because the dif-
fERENCE in shade level between adjacent polygons is too
small to produce a perceptible aliasing effect. In practice
very few edges other than those corresponding to the
outline of the car had to be antialiased.

The model in Figure 11 was rendered without smooth
shading in order to make the polygon tiles visible. When
smooth shading is applied, it is possible to achieve
basically the same visual effect by approximating the car
with far fewer polygons. However, having a virtually
constant rendering time can be significant for rendering
free-form surfaces. Much of the recent research effort in
ray tracing has concentrated on speeding up the intersec-
tion calculation for surface patches, which is known to be
especially expensive. In spite of significant improvements
in calculation speed, the process of rendering patches is
generally so slow that it is practically excluded from
practical applications. The above cited timings suggest
that ART can be used for efficient rendering of free-form

Figure 9. A comparison of the calculation times for ART
and MOVIE.BYU.

Figure 10. “Brain” is a density cloud created using the
DNA model in Figure 6. Density for each individual
electron was represented by
Gaussian bump.
Note that the colors representing different types of
atoms are also fused. Preprocessing time was eight
seconds and rendering time was 45 minutes on a VAX
11/785.

April 1986
patches approximated to the desired tolerance by polygon tiles.

**Metaball.** Various primitives have been used in experiments to show the universality of the proposed method. The concept of a complex primitive called metaball or density cloud was first introduced by Nishimura (the Links ray-tracing system). In the English-language literature it was later introduced by Blinn as a result of independent development.

The model of a density cloud used in our experiments contained all 7011 of the atoms displayed in Figure 7 as simple spheres. The complex surface shown in Figure 11 corresponds to a certain threshold of electron density distribution resulting from a fusion of intensity distribution of the individual atoms. Calculation of the intersection of a ray and such a surface is generally far more expensive than it is in the case of ordinary primitives. The intersection cannot be found algebraically; numerical calculation is required. (Regula falsi and Newton iteration are used.)

The problem is aggravated by the number of atoms in the cloud. The intensity distribution of each atom is defined as a Gaussian bump. It reaches its maximum at the center of the atom and becomes zero at an infinite distance from this center. Since the intensity distribution of each atom spreads to infinity, the resulting intensity at an arbitrary point in space becomes influenced by each atom. So for a model containing more than a few atoms, the summation of a function over all the atoms is computationally out of the question. However, we can economize considerably by using in the calculation only those atoms "close" to the point being considered. The term close means in practice that the density influence of a particular atom is above a certain allowable error threshold which can be defined by the user. Such a threshold will result in enclosing each atom in a sphere of influence. This information is used during the enumeration phase and, with any particular atom, only those voxels that happen to be interior to such a sphere are culled and enumerated.

To achieve a similar effect, Blinn has implemented a scheme using three levels of culling. The first level is introduced in the outer loop corresponding to the $y$ direction when moving from one scan line to the next. The second level is performed in the $x$ direction when moving from one pixel (ray) to another. The final level is in the inner loop in the $z$ direction, along the ray. Since culling is performed during rendering, in the inner loop, the total rendering time was significantly influenced. For a small number of atoms (timing figures are presented only for the case of 64 atoms) the time spent for culling is less than 30 percent of the total rendering time. However, increasing the number of atoms will inevitably present us with the situation where the renderer spends most of its time culling.

In the present algorithm, this culling is first of all not performed during the rendering process. Rather, it is naturally incorporated in the enumeration phase. Enumeration time for the model shown in Figure 11 is virtually the same as for the model in Figure 7. Even for a significant number of objects this time (less than 10 seconds) corresponds to a negligible fraction of the rendering time (less than one percent). As mentioned before, the intersection calculation for density clouds is expensive because the Gaussian bump used for expressing the density function necessitates numerical calculation. We expect to improve the calculation speed of the density cloud primitive still further, by implementing a polynomial function such as the biquadratic. Clouds defined using this function will not require numerical solution for the intersection calculation.

**Antialiasing**

The main problem with antialiasing for ray tracing is that not enough information is associated with a single ray. Rays allow us to sample at the one point in the center of the pixel. Without firing additional rays, there is no way of knowing or calculating what else is visible in the neighborhood surrounding the sample point.

In classical ray tracing, antialiasing is usually done by adaptive subdivision of pixels near large intensity changes or small objects. This method, now almost universally employed, attempts to use heuristic criteria to probe the image frequently enough that small details will not be overlooked. Depending on the criteria, it will sometimes subdivide too little, resulting in aliasing, or too much, in which case processing time will be wasted. Heckbert and Hanrahan have proposed a different approach, in which a polygon scan converter with a pixel integrator is adopted. This method of antialiasing is sometimes used in scan-line algorithms for continuous-tone images. However, it was possible for Heckbert and Hanrahan to adopt it because the beam-tracing approach they were presenting uses a polygon renderer for scan conversion to form the final image. Polygon edges are determined during scan conversion so that information can be used directly for antialiasing.

It is sometimes argued that the only way to antialias within standard ray tracing is to go to a higher resolution. The approach presented in this article shows that this is not exactly the case. The Adjustable Fourier Window Technique reported previously showed that high-quality antialiasing can be performed with a very low computational cost. More recently it has been shown that it can also be used successfully for antialiasing with subpixel resolution. Objects smaller than a pixel can be properly rendered. At the same time it was shown that full information about the antialiased edge (position and
slope) was not necessary for its antialiasing. What is essentially needed is the distance to the edge from a neighboring pixel measured perpendicular to the Driving Axis. Whether the DA is the x-axis or the y-axis can be determined in 75 percent of cases by simple comparison of the difference of generated pixel intensity, both vertically and horizontally. In the remaining 25 percent of cases an additional ray must be fired to resolve the ambiguity. After the DA is determined, the position of the edge intersection measured perpendicular to the DA can be determined with sufficient precision (one-eighth of pixel size) after firing only three rays. In general not more than four additional rays have to be fired to antialias an edge.12,22

Conclusions

This article has introduced a general method of improving the computational speed of the ray-tracing method to a level where the image synthesis time is practically constant, and thus independent of the number of objects in the scene. Experimental results show that for a very large number of objects the rendering process is actually executed faster than in other currently used methods. It has also been shown that the proposed method can efficiently handle both models defined by the use of B-reps and models defined as unions of solid primitives.

This is achieved essentially by creating an environment where the number of objects that have to be checked against each ray is limited to an insignificant fraction of the total number of objects involved in the scene. This fraction very often corresponds to a single object.

The above scheme is realized by imposing on the object an auxiliary structure characterized by a high level of coherency (SEADS), and developing a traversing tool in the form of a line generator (3DDDA) that takes advantage of this coherency in a very efficient manner based on incremental integer logic.

Acknowledgments

We would like to express our gratitude to Mr. Christopher G. Perrott, chief engineer at Graphica Computer Corporation, for his advice and fruitful critical remarks concerning the content of this article; for his help in optimizing the code; and finally for proofreading the article. We also extend our gratitude to Dr. Nelson Max from Lawrence Livermore Laboratory. It was he who suggested that our accelerated ray tracing could efficiently handle a complex primitive such as the density cloud, and encouraged us to implement it. Dr. Max also provided us with various DNA models that were very helpful in testing our algorithms.

References


April 1986


Akira Fujimoto, formerly Wieslaw Rom-anowski, is a chief engineer with the Software Research Division of the Graphica Computer Corporation, a company that produces computer graphics and image processing systems. His research interests include computer graphics for raster-scan devices, applications of computer graphics in scientific and engineering analysis, and CAD.

Fujimoto received an ME in mechanical engineering from the Technical University Szczecin (Poland) and from the University of Tokyo, where he subsequently obtained his PhD. He is a member of the Society of Naval Architects of Japan, the Computer Graphics Society (GCS), and the GKS Japan Committee of GCS.

Takayuki Tanaka is an analyst-programmer in the Software Research Division of Graphica Computer Corporation. His research interests include operating systems, high-level languages, artificial intelligence, and computer graphics algorithms. He has taken part in the development of a number of image-rendering software packages for Graphica Computer Corporation.

Tanaka graduated from Sundai Electronic Institute in 1984.

Kansci Iwata is president of Graphica Computer Corporation, which he founded in 1975. From 1970 to 1975, he was employed by Iwatsu Electronic Company, Ltd., where he was chief of the Electronic Circuit Research Laboratory at the Technical Institute in 1974 and 1975. His research interests are related to pulse transmission and electronic circuit design.

Iwata received his BA and PhD in electrical engineering from Tohoku University in Sendai, Japan. He is a member of the IEEE and the ACM.

The authors address is Graphica Computer Corporation, 6-21-6 Nagayama Tami-shi, Tokyo 206, Japan.

---

Additional Readings


