Improved Computational Methods for Ray Tracing

HANK WEGHORST, GARY HOOPER, and DONALD P. GREENBERG
Cornell University

This paper describes algorithmic procedures that have been implemented to reduce the computational expense of producing ray-traced images. The selection of bounding volumes is examined to reduce the computational cost of the ray-intersection test. The use of object coherence, which relies on a hierarchical description of the environment, is then presented. Finally, since the building of the ray-intersection trees is such a large portion of the computation, a method using image coherence is described. This visible-surface preprocessing method, which is dependent upon the creation of an "item buffer," takes advantage of a priori image information. Examples that indicate the efficiency of these techniques for a variety of representative environments are presented.

Categories and Subject Descriptors: I.3.3 [**Computer Graphics**]: Picture/Image Generation—*display algorithms*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*color, shading, shadowing, and texture*

General Terms: Algorithms

Additional Keywords and Phrases: Computer graphics, ray tracing, visible-surface algorithms, hierarchical data structures

1. INTRODUCTION

In the traditional ray-tracing algorithm, the global illumination is approximated by "tracing" rays from the eye, through the viewing plane, and into the environment. At the closest surface intersected by a ray, reflected and/or refracted rays may be spawned. Each of these must be recursively generated to establish which surfaces they intersect. As the ray is traced through the environment, a ray-intersection tree is constructed for the sample point. The final intensity is determined by traversing the tree depth first and computing the intensity

This research was performed at the Program of Computer Graphics at Cornell University and supported by the National Science Foundation under grant MCS 8302979. Additional student support was provided by the Natural Science and Engineering Research Council of Canada
Authors' address: Program of Computer Graphics, Cornell University, Ithaca, NY 14853

contribution of each node according to the reflection model. Although ray tracing was first suggested by Appel [1] and later used by MAGI [4] to solve the hidden-surface problem, this approach was first implemented for rendering purposes by Kay [8] and Whitted [12].

The intensive computational operations in a ray-tracing system are the building operations of the ray-intersection trees and the traversal of those trees for the intensity calculation. This latter step, which includes shadow testing, has been found to be very expensive computationally, particularly for environments that contain complex lighting schemes.

Since the computational expense of producing ray-traced images is excessive, it is obvious that new techniques must be found to reduce the calculation time. These techniques will inevitably utilize both object and image coherence [11]. The term *coherence* is used to describe the extent to which an environment or an image is locally constant. Object coherence relies on the known relationship between objects in an environment, and this information is used to reduce the visible-surface computations. Image-coherence techniques take advantage of the fact that the image does not change rapidly from point to point or scan line to scan line on the viewing plane. They capitalize on the lateral separation of the image changes to reduce the depth comparisons. Neither of these two types of coherence has been used extensively in ray-tracing schemes to date.

The results of several investigations related to reducing the computational expense of ray tracing have recently been published. To improve the performance of their ray-tracing system, Rubin and Whitted used both bounding volumes and a hierarchical description of the environment [9]. According to the authors, the use of spherical bounding volumes substantially reduced computation time, and this procedure is now common practice in ray-tracing. The partitioning of an environment has also improved the efficiency of visible-surface algorithms. In their implementation, the environment was subdivided into hierarchical, orthogonal subspaces, except for the lowest level primitives, which were bounded by arbitrarily oriented, rectangular parallelepipeds. During picture generation, each ray was transformed to align with the axes of the parallelepipeds, so that the intersection tests reduced to simple comparisons against the limits of the bounding boxes. The computational times for their ray traced images were significantly reduced.

Crow [3] also described an image generation system which determined separability of objects during scene analysis. The environment is partitioned by crude overlap and depth tests, and priorities for rendering are established. Although this partitioning allows different rendering algorithms to be used for each object, as well as parallel processing, it is difficult to apply to ray-tracing algorithms which require global information.

More recently, Kajiya published an article improving intersection algorithms for the ray tracing of three types of procedurally defined objects, namely, fractal surfaces, prisms, and surfaces of revolution [7]. Since fractal surfaces are defined recursively, Kajiya determined probabilistic "extents" of the fractal surface, which, in a sense, act as bounding volumes prior to the fractal surface definition. Only after it is known that the ray intersects an extent is the fractal surface

Table I. Definition of Terminology

(a) A light is a geometric entity and an associated list of light attributes. Light attributes describe the intensity of the light, its goniometric diagram, and color spectrum. Lights are emitters.
(b) An object is a geometric entity and an associated list of object attributes. Object attributes describe the surface properties of the object, such as surface roughness and reflectance properties. Objects, in contrast to lights, are not emitters, but they may reflect light.
(c) An item is a light or an object. It consists of a geometric entity and an attribute list. Currently implemented geometric entities are spheres, cylinders, other quadrics, parametric surfaces, and polyhedra.
(d) A cluster is an entity which groups together other clusters and/or items in such a way that no cluster can have a descendant which is also an ancestor. Also, each element in the cluster must have a unique parent, that being the cluster. This ensures that the structure is a tree. In addition, each cluster has a unique name associated with it.
(e) The world is a cluster which is the uppermost node in the hierarchical description of the environment. It has no parent. Only those items contained within the world, either directly or indirectly, can be rendered.
(f) An element is either a cluster or an item. All elements, with the exception of the world, must belong to exactly one cluster.
(g) A ray is a vector with a specific origin and arbitrary direction.
(h) A ray-intersection tree is a collection of rays representing the geometric path of light propagation, starting at the eye, passing through the viewing plane, and extending into the environment.
(i) An intersection list contains elements which are to be tested in determining the closest item intersected.

generated. His solution to the prism and surface of revolution problems reduced the three-dimensional ray-intersection tests to two dimensions.

Hall utilized adaptive tree-depth control to reduce computational expense [5] [6]. Traditionally, ray-intersection trees for all sample points are constructed to an arbitrary, prespecified depth to ensure that all relevant reflections and refractions are captured for the image. However, the upper bound of the contribution of any node in the ray-intersection tree to the final color of the sample point can be determined according to the diffuse, specular, and transmissive properites of the intersected surface. Thus, by establishing a contribution threshold, the depth of the tree can be adaptively controlled. Statistics show the significant savings that can be obtained, even for highly reflective environments.

This paper describes additional procedures which have been implemented to reduce the computational expense of producing ray-traced images. In Section 2 the selection of bounding volumes is examined to reduce the computational cost of the ray-intersection test. A hierarchical description of the environment, which relies on object coherence, is presented in Section 3. Last, since the building of the ray-intersection tree is such a large portion of the computation, a new approach using image coherence is described in Section 4. This visible-surface preprocessing method, which is dependent upon the creation of an "item buffer," takes advantage of a priori image information, and substantially reduces the computational time. Examples that indicate the efficiency of these techniques for a variety of representative environments are presented.

For clarification of the following discussion, specific definitions of the terminology used are presented in Table I.

2. BOUNDING VOLUMES

In creating a node in the ray-intersection tree, and during shadow testing for color calculation, it is necessary to determine which intersected item, if any, is nearest to the origin of the ray. The item that is the closest is the one visible. Since the origin and direction of the ray are arbitrary, all items in the environment must be considered during the intersection process.

The amount of time needed to test an item for intersection depends on its geometric description. Typically, a sphere is the simplest of shapes to intersect. The intersection time for a polyhedron generally depends on the number of faces that it contains although Kajiya has shown an improved intersection test for the special case of prisms [7]. Other shapes, such as surfaces of revolution and parametric surfaces, may require more time to test.

Unfortunately, typical environments do not consists of spheres and simple polyhedra. Instead, the geometries are usually complex and of various definitions. For this reason, it is often desirable to enclose complex objects with simpler, abstract ones, such as spheres or rectangular parallelepipeds. These enclosing shapes are called bounding volumes, as suggested by Clark [2] and implemented by Whitted [12]. Only if a ray happens to intersect the simpler bounding volume is the enclosed complex item tested. However, if the ray does not intersect the bounding volume, then there is no need to test the complex item within. If the bounding volumes are chosen wisely, this scheme can greatly reduce the cost of intersection testing.

Defining the optimal bounding volume, however, is difficult. The common practice of selecting bounding volumes on the basis of the simplicity of their intersection tests should not be the only consideration. Other factors, such as the projected void area, should also be examined.

The void area is the difference in the projected areas of the bounding volume and the item (Figure 1). If the cost of intersecting the bounding volume is, for the moment, ignored, then, for a given ray, the best bounding volume is the one that produces the least void area when orthogonally projected onto a plane perpendicular to the ray and passing through the origin of the ray. Minimizing the void areas for all rays produces a bounding volume that is identical to the item itself. If restricted to a given shape of bounding volume, minimizing these void areas results in the selection of the optimal bounding volume of that geometry.

Obviously, one cannot ignore the cost of intersecting the bounding volume in this selection process, as it is definitely not advantageous to select the item itself. For many environments, one should consider the void area of the item for rays of specific origin and direction. Initially, consider an environment containing only surfaces that are totally diffuse, and having either light emanating from the viewer position or ambient lighting. Because there are no reflected or refracted rays, all rays emanate from the eye and pass through the viewing plane. Thus the origin and direction for all rays are known, and the void areas should be used in the selection of bounding volume. However, in the general case, as the illumination becomes more complex, or the environment becomes specular or transparent, this factor becomes more difficult to consider since rays are more likely to arrive in any direction.

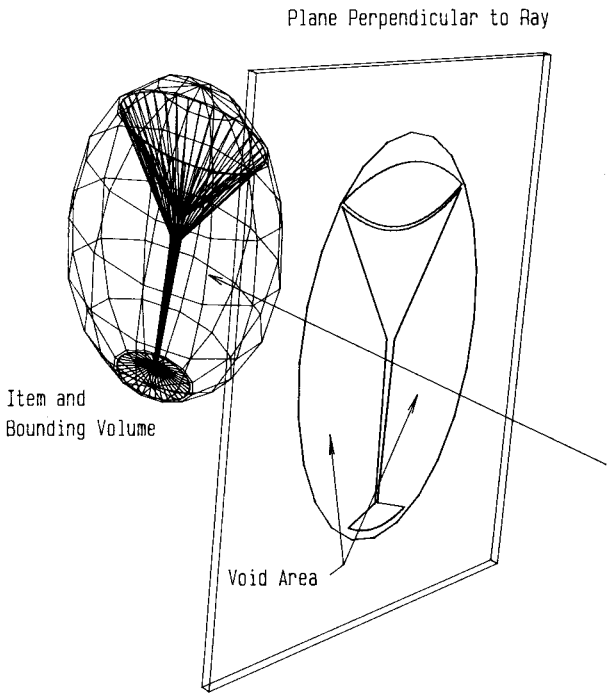

Fig. 1. Projected area of item and its bounding volume.

To illustrate the concept of void area, three different views of the same item are shown (Figure 2). The front and side views were each rendered using alternately a bounding sphere and a bounding rectangular parallelepiped. In the case of the front view, the image using the bounding sphere was rendered in 88 percent of the time of that using the parallelepiped. This reduction arises from the fact that the void area is slightly less for the sphere. However, the side view using the parallelepiped was rendered in only 47 percent of the time needed to render the image using the bounding sphere. For this view, even though the sphere is less costly to test for intersection, the void area of the parallelepiped is a mere fraction of that of the sphere. Thus, the optimal choice is ray dependent.

Another important criterion in the selection of a bounding volume is the complexity of the item being enclosed. The total cost function of the intersection test for an item is

$$T = b * B + i * I, \tag{1}$$

where

T is the total cost function;
b is the number of times that the bounding volume is tested for intersection;
B is the cost of testing the bounding volume for intersection;
i is the number of times that the item is tested for intersection, which is equivalent to the number of times that the bounding volume is intersected ($i \leftarrow b$);
I is the cost of testing the item for intersection.

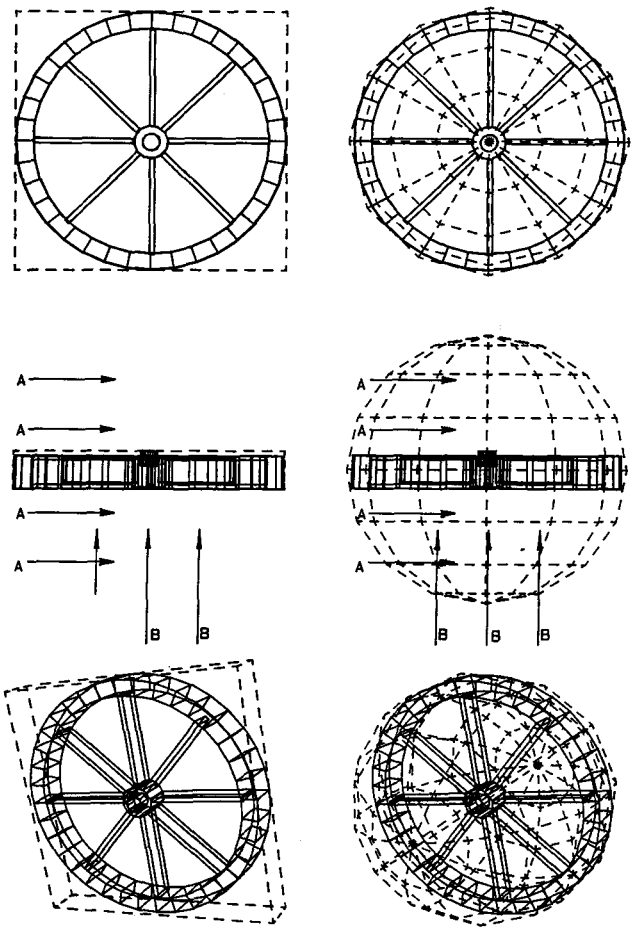

Fig. 2. Bounding volume selection.

It is desirable to minimize this function for all items. For a specific item, in a given environment, with a given view, b and I are constant. However, by manipulating the shape and size of the bounding volume, B and i can be varied to reduce the total cost function T. Reducing the complexity of the bounding volume generally results in a decrease of its cost function B, but an increase in i. Alternatively, increasing the complexity generally results in an increase of B and a decrease in the number of item intersection tests i. Neither adjustment guarantees a decrease in the total cost function T.

Bounding volumes are automatically assigned, but only to those items whose intersection tests are sufficiently complex to warrant one. Certain items, such as spheres, cylinders, and rectangular parallelepipeds, do not require the addition of this entity. The first step in this assignment is the determination of a set of three-dimensional points which "surrounds" the item. These coordinates define the vertices of some convex polyhedron which completely encloses the item. There are obviously many choices for this set, but it is optimally selected in such a way that the volume of the polyhedron less the volume of the item is minimal.

For a polyhedral item, these points are simply its vertices. (The inclusion of interior points of concave shapes is not detrimental.) For nonplanar surfaces, these points represent a polyhedral approximation which at least surrounds the item.

Having determined this set, a bounding volume is now selected. Three candidates currently exist for the shape of the bounding volume, namely, a sphere, a rectangular parallelepiped, and a cylinder. Associated with each candidate is a factor that is indicative of the relative complexity of its intersection test, with the sphere having the lowest factor, and the cylinder, the highest. A potential bounding volume of each shape is created for the set of points to be enclosed. The selection is made by minimizing the product of the associated complexity factor and the volume of the potential bounding shape. Volume is considered, since it intuitively considers the projected void areas for all directions. Although it is not accurate, it is representative and easy to calculate. To compensate for the item complexity factor being ignored, an interactive program is provided to allow the user to override any automatic bounding volume selection.

3. HIERARCHICAL ENVIRONMENT

The concept of bounding volumes is extended to exploit properties of the environment which are independent of the observer's position. Groups of items, which are in close proximity to each other, can be clustered within a single bounding volume. Higher level clusters can be created by grouping clusters and/or items together. Thus, in addition to the geometric information, a hierarchical description of the environment can be created where the leaves are items, and the interior nodes are clusters (e.g., Figure 3 shows the hierarchy for the environment of Table VII).

When intersecting a ray with the environment, one first tests the top level and recursively descends the hierarchy only along those branches where intersections occur (Figure 4). Computational savings occur because the algorithm can save ancestral information and thus avoid needless intersection tests. That is, it is only necessary to test individual elements in a group if the parent bounding volume is intersected.

Care must be taken in creating the hierarchy for an environment. Elements that are not near each other should not be placed in a cluster. If they are, then many rays will intersect the cluster's bounding volume but not the elements within, thereby defeating the purpose of the cluster. Also, if clusters have few children and the hierarchy is deep, excessive bounding volume tests and intersections may occur.

The hierarchical structuring assigned to an environment is basically the one defined by the user during the modeling process. The only difference is that clusters containing only one element are removed from the hierarchy. Although the structuring used during modeling is not necessarily ideal for ray tracing, it is often sufficient, since users naturally tend to cluster elements within close proximity to each other. In either case, a user seldom creates a cluster that is very detrimental to the performance of the ray-tracing renderer. Bounding volumes are automatically assigned to clusters by considering all descendent items.

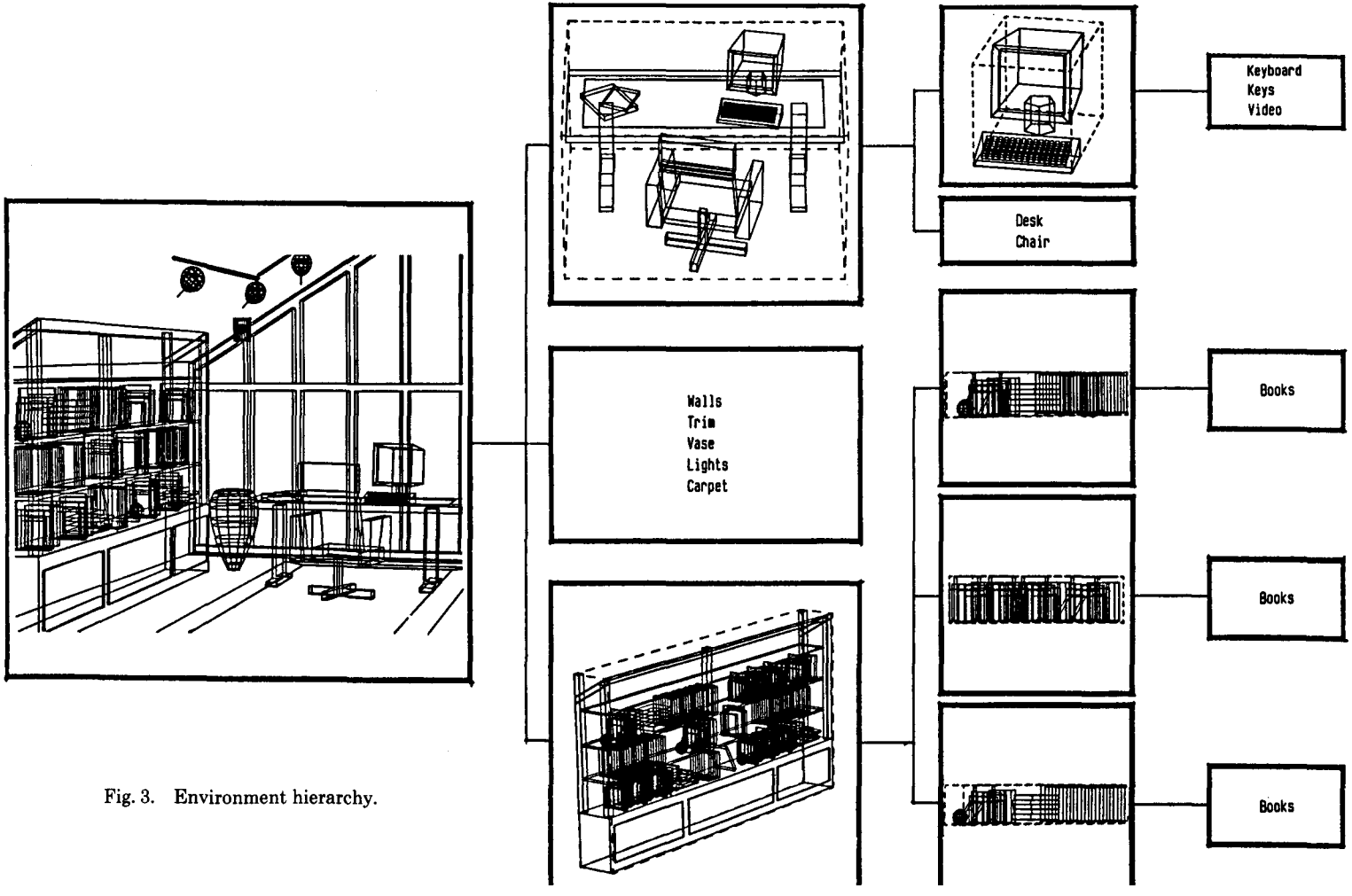

Fig. 3. Environment hierarchy.

```
function closest(world: bv; r: ray): item;

  var
    c, e:          element;
    distance:      real;
    elements:      list_of_bv;
    inside_bv:     boolean;

  begin
    elements := world.children;
    while (elements is non_empty) do
      begin
        while not (at end of elements) do
          begin
            c := current element;
            if (c has been tested) then
              advance to next element in elements
            else
              begin
                if (intersect(c, ray, distance, inside_bv)) then
                  begin
                    if (inside_bv) then
                      replace c with c.children
                    else if (distance > 0.0) then
                        advance to next element in elements
                    else
                        return(c); § This is the closest element†
                  end
                else
                    remove c from elements as it was not intersected;
              end;
          end;
        if (elements is non_empty) then
          begin
            e := element in elements with least intersection distance;
            if (e.category is BOUNDING_VOLUME) then
              replace e with e.children
            else
                return(e); § This is the closest item intersected †
          end;
      end;
    return(NIL); § No item intersected †
  end;
```

Fig. 4. Hierarchical intersection algorithm.

4. VISIBLE-SURFACE PREPROCESS

In past ray-tracing algorithms, the procedure for determining an intersection and the cost of that intersection are similar for all nodes in the ray-intersection tree. Although handling all rays similarly has obvious conceptual advantages in the implementation of the algorithm, it is not the most efficient approach to use

when considering computational optimization. An alternate technique, which distinguishes the first node in the ray-intersection tree from the remaining nodes, has been developed. When using the adaptive tree-depth method described earlier and noting that in most environments the average depth of the ray-intersection tree is not much greater than one, the computation reduction that could be achieved by reducing the time spent computing the first level of the tree becomes evident.

In the ray tracing of an image, the only rays whose origin and direction are easily determined, are those emerging from the eye and piercing the image plane. By taking advantage of this known information, the amount of time spent determining the first ray's intersection with the environment can be decreased. It is evident that a reduction in the number of items tested would proportionately reduce the computational expense. By utilizing image-space coherence for those rays originating at the eye, a reduction in the size of the intersection list can be realized.

The perspective transformation of all items in the environment results in a two-dimensional projection of each onto the image plane. If an intersection list is constructed on a pixel-to-pixel basis, only those items that partially or fully cover the pixel area will be included. This abbreviated list can be used when testing for the closest item. In order to reduce this list to a single entry, typical visible-surface algorithms can be invoked.

The determination of visible surfaces within an environment, given a viewer position and frustum, can be performed in a variety of ways, each having its own advantages [11]. Unlike the ray-tracing technique, the computational expense of most image-space visible-surface algorithms is much less than directly proportional to the resolution of the image. This can be inferred from the fact that during ray tracing, all objects are tested at every pixel to determine the closest intersection. However, traditional image-space, visible-surface algorithms sort all items by depth only over their areas as projected onto the image plane.

Specifically, the algorithm is implemented as follows. As a preprocess before ray tracing, a z-buffer algorithm is executed using the same viewing parameters. The z-buffer algorithm is used since it can accommodate a wide range of geometric primitives. Instead of producing the traditional display buffer, the algorithm is modified to produce an *item buffer*. Each entry within the item buffer corresponds to some location on the image plane and contains an index to a list of the items within the environment. This index is simply the indirect storage of the identical information that would have been produced by the ray tracer at this level. The ray-tracing process may then substitute a simple indexing operation into the item buffer for the first ray-intersection test.

The visible-surface preprocess determines the visible item at discrete locations corresponding to the corners of the pixels of the image plane. If a ray passes through the corner of some pixel, then it is assumed to hit the item that has been deemed visible by the preprocess. However, when subdivision occurs, to sample a pixel more accurately, the ray passes through either an edge or the interior of the pixel. In either case, the abbreviated intersection list used by the ray tracer contains those item indices lying within a desired radius of the sample location.

5. RESULTS

In order to show the computational efficiency of the procedures described, various environments have been modeled and rendered (Tables II–VIII) and Figures 5–9. These environments differ greatly in complexity, and contain varying shapes of items, numbers of light sources, and specular and transmissive properties. For each ray-traced image generated, statistics are presented that indicate the reduction in the computational time when one or more of the techniques are invoked.

To indicate where the computationally intensive operations lie, and to show the benfits of the algorithms, the relative processing times for each image is divided into the following categories: the percentage of the total time required to produce the ray-intersection trees (tree generation); the percentage of time spent determining direct illumination (shadow testing); the percentage of time used in generating the item buffer (visible-surface preprocess); the percentage of time spent calculating color according to the reflection model (color calculation); and, total miscellaneous time, such as file handling (miscellaneous).

For each environment, a control image consisting of items enclosed only by spherical bounding volumes was generated. Similar images using all combinations of the improvement techniques were statistically measured. Tables II–VIII illustrate the results for the following cases:

C	control image, using only spherical bounding volumes;
CZ	spherical bounding volumes with visible-surface preprocess;
CH	spherical bounding volumes with hierarchy;
CHZ	spherical bounding volumes, hierarchy, and visible-surface preprocess;
S	selected bounding volumes;
SZ	selected bounding volumes with visible-surface preprocess;
SH	selected bounding volumes with hierarchy;
SHZ	selected bounding volumes, hierarchy, and visible-surface preprocess.

A composite table showing the reductions in computational times is presented in Table IX.

6. CONCLUSIONS

Previously published articles have described the advantages of using bounding volumes and adaptive tree-depth control. This paper has presented a variety of methods to further reduce the computational expense of producing ray-traced images. An improvement in the selection of bounding volumes has been discussed. To determine optimal bounding volumes, specific characteristics examined include the cost of intersecting the volume, the item complexity, and the projected void areas. A method of using the topological information of a hierarchically defined environment to reduce computational time has been presented, and the concept of an item buffer introduced. A visible-surface preprocess utilized image coherence to reduce the ray-intersection tree generation time. This improvement is most significant when the number of rays emanating from the eye and passing through the image plane represents a large portion of the total number of rays

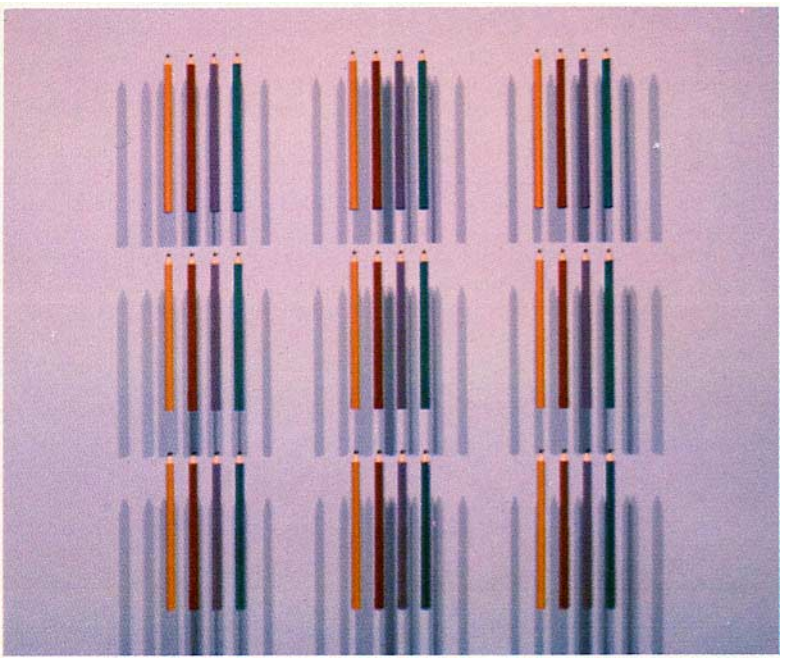

Fig. 5. Pencils.

Table II. Image Timing Test for Pencils I (Figure 5)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	48.4	14.3	46.7	16.5	43.7	21.5	39.8	30.1
Shadow testing[c]	44.8	75.3	45.5	70.7	43.9	61.3	40.2	46.1
Visible surface preprocess[c]	—	0.3	—	0.3	—	0.3	—	0.7
Color calculation[c]	3.5	5.2	4.0	6.4	6.4	8.4	10.4	11.5
Miscellaneous[c]	3.3	4.9	3.8	6.1	6.0	8.3	9.6	11.6
Total relative time	1.00	0.67	0.86	0.54	0.54	0.40	0.32	0.28

[a] Specifications: 38 items (1806 polygons); 1 light source; nonreflective environment; ray-intersection tree depth of one.

[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.

[c] Shown as a percentage of total time

Note:

(1) The improvement of selected *bv*'s versus control *bv*'s due to the elongated nature of the items.

(2) The improvement due to a hierarchy that takes advantage of the proximity of the items. Each pencil is grouped with its three closest neighbors, creating nine clusters.

(3) The improvement of the visible surface preprocess during tree generation. This is due to the high ratio between the number of first level rays versus total number of rays.

Table III. Image Timing Test for Pencils II (similar to Figure 5)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	52.5	39.9	53.6	40.0	46.6	40.4	43.5	40.0
Shadow testing[c]	40.6	51.0	37.7	48.9	44.0	48.9	38.2	39.8
Visible surface preprocess[c]	—	0.2	—	0.2	—	0.2	—	0.3
Color calculation[c]	5.2	6.6	6.5	8.2	6.8	7.8	13.7	14.8
Miscellaneous[c]	1.7	2.3	2.2	2.7	2.3	2.7	4.6	5.1
Total relative time	1.00	0.79	0.79	0.62	0.76	0.65	0.37	0.34

[a] Specifications: 38 items (1806 polygons); 1 light source; reflective environment; ray-intersection tree depth of three.
[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.
[c] Shown as a percentage of total time
Note:
(1) The improvement of selected *bv*'s and the hierarchy in the tree generation process. In relationship to Table II, the improvement remains proportional with the increased depth of the tree.
(2) The improvement gained from the visible surface preprocess is the same as Table II. The percentage of the improvement decreases along with the first ray versus total ray ratio.

Table IV. Image Timing Test for Pencils III (similar to Figure 5)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	23.7	4.5	23.7	5.3	22.1	8.2	19.9	13.6
Shadow testing[c]	70.8	88.5	69.9	86.5	67.6	79.6	62.3	66.4
Visible surface preprocess[c]	—	0.1	—	0.1	—	0.2	—	0.3
Color calculation[c]	4.0	5.0	4.7	5.9	7.5	8.7	13.0	14.3
Miscellaneous[c]	1.5	1.9	1.7	2.2	2.8	3.3	4.8	5.4
Total relative time	1.00	0.78	0.82	0.66	0.51	0.44	0.28	0.26

[a] Specifications: 38 items (1806 polygons); 3 light sources; nonreflective environment; ray-intersection tree depth of one.
[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.
[c] Shown as a percentage of total time
Note:
(1) The improvement of selected *bv*'s and the hierarchy in the shadow testing process. In relationship to Table II, the improvement remains proportional with the increased number of shadow tests.

in the environment. Statistics for a variety of environments that differ in complexity, item geometries, numbers of light sources, and specular and transmissive properties consistently indicate improvement.

Further work must include the extension of the item buffer concept to individual light sources, as shadow testing still represents the dominant portion of computational expense. Improved bounding volume selection, including the use of multiple bounding volumes for single elements, is being investigated. Also, a means of automatically generating the hierarchy must be found.

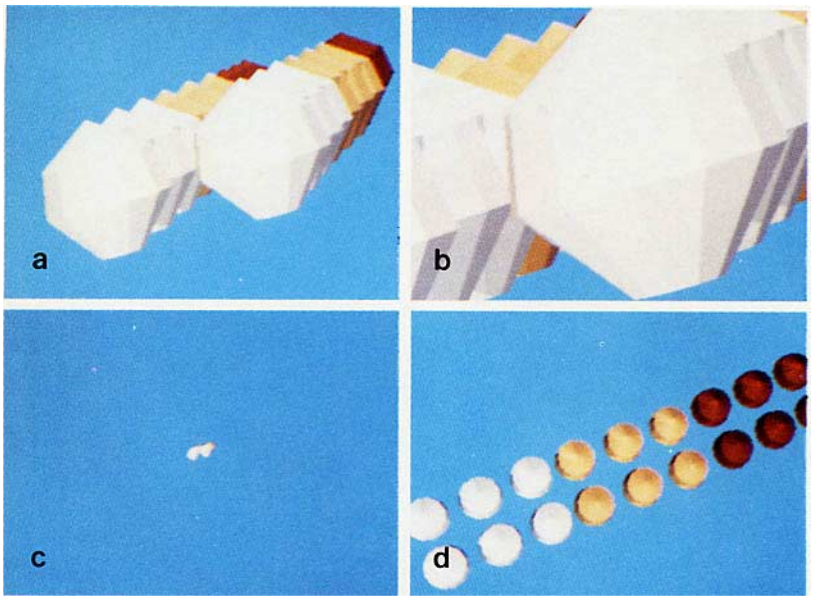

Fig. 6. Polygons.

Table V. Image Timing Test for Polygons (Figure 6)[a,b]

	A		B		C		D	
	C	SHZ	C	SHZ	C	SHZ	C	SHZ
Tree generation[c]	46.5	36.2	45.4	38.3	52.0	22.0	43.9	35.0
Shadow testing[c]	42.2	49.0	45.9	51.1	2.8	3.9	39.4	44.2
Visible surface preprocess[c]	—	1.0	—	0.6	—	5.0	—	1.4
Color calculation[c]	5.2	6.4	5.5	6.2	9.7	13.8	6.1	7.0
Miscellaneous[c]	6.1	7.4	3.2	3.8	35.5	55.3	10.6	12.4
Total relative time	1.00	0.82	1.00	0.86	1.00	0.66	1.00	0.86

[a] Specifications: 21 items (1980 polygons, 3 spheres); 3 light sources; adaptive ray-intersection tree depth.

[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.

[c] Shown as a percentage of total time

Note:

(1) In order to isolate the effect of the visible surface preprocess, the background was not modeled as a polygon, and thus does not receive shadows or require shadow testing. Thus shadow testing time is a function of the number of tree nodes that are not background nodes.

(2) The tree generation time is view dependent and a function of the nonbackground screen area.

(3) The decrease in computation time resulting from the visible surface preprocess takes place entirely in the tree generation. Within this stage, relative improvements on all four images are approximately equal.

(4) Total CPU time for control images: (a) 1.57 hours; (b) 3.12 hours; (c) 0.24 hour; (d) 0.89 hour.

Fig. 7. Camera.

Table VI. Image Timing Test for Camera (Figure 7)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	60.9	56.0	60.6	56.4	57.3	51.9	56.1	52.5
Shadow testing[c]	30.6	34.6	30.0	33.0	30.1	33.7	28.1	30.2
Visible surface preprocess[c]	—	0.2	—	0.2	—	0.3	—	0.3
Color calculation[c]	3.0	3.2	3.3	3.7	4.4	4.9	5.5	5.9
Miscellaneous[c]	5.5	6.0	6.1	6.7	8.2	9.2	10.3	11.1
Total relative time	1.00	0.92	0.90	0.80	0.67	0.60	0.53	0.49

[a] Specifications: 39 items (1392 polygons, 1 sphere); 1 light source; adaptive ray-intersection tree depth.

[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.

[c] Shown as a percentage of total time

Note:

(1) The improvement gained using the visible surface preprocess. In this case, due to there being only one light source, the ratio of first level rays versus total number of rays is high.

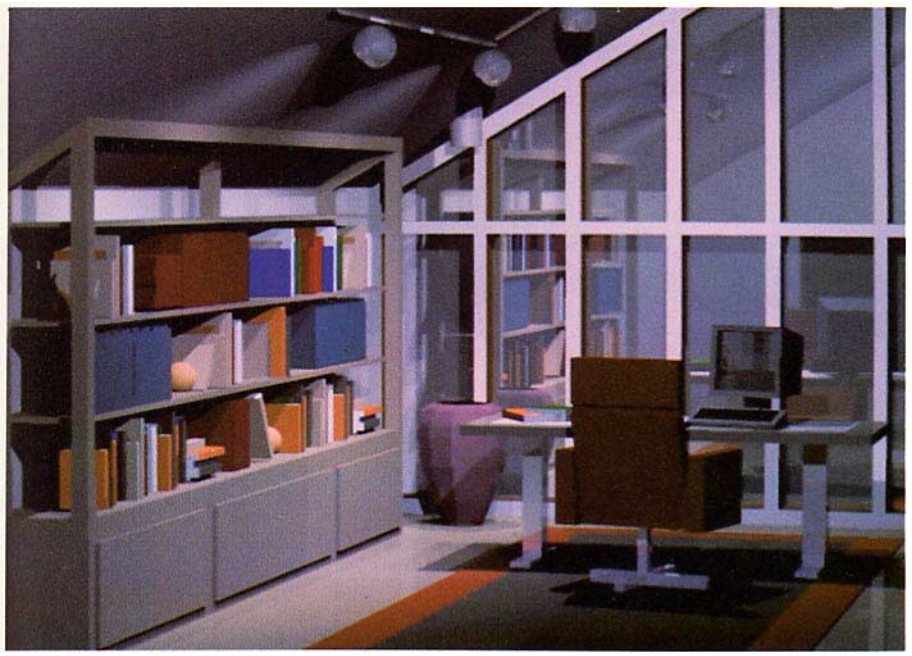

Fig. 8. Office Scene.

Table VII. Image Timing Test for Office Scene (Figure 8)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	21.5	9.6	21.8	10.5	21.3	10.4	22.3	12.3
Shadow testing[c]	74.1	85.0	72.3	82.9	73.8	83.8	70.2	78.9
Visible surface preprocess[c]	—	0.2	—	0.2	—	0.2	—	0.3
Color calculation[c]	3.4	3.9	4.6	4.9	3.8	4.3	5.8	6.5
Miscellaneous[c]	1.0	1.4	1.3	1.5	1.1	1.3	1.7	2.0
Total relative time	1.00	0.87	0.74	0.67	0.90	0.79	0.56	0.49

[a] Specifications: 63 items (2101 polygons, 8 spheres); 5 light sources; adaptive ray-intersection tree depth.

[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.

[c] Shown as a percentage of total time

Note:

(1) The improvement gained from the use of the hierarchical data structure for which this environment lends itself.

(2) Shadow testing emerging as the dominant area of computation. This will be the case in most images of moderate complexity, illuminated with multiple light sources.

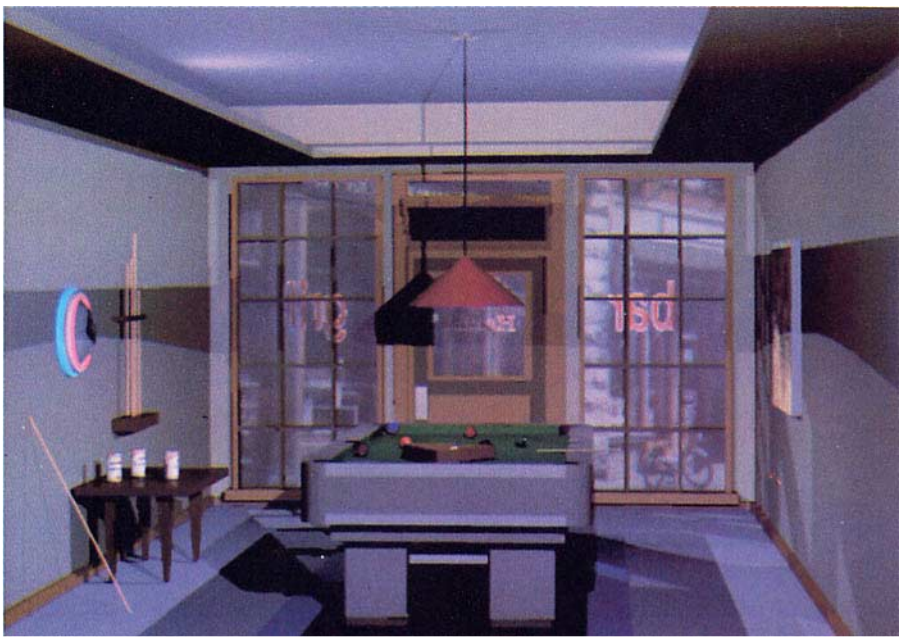

Fig. 9. Pool Room.

Table VIII. Image Timing Test for Pool Room (Figure 9)[a,b]

	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Tree generation[c]	21.7	10.3	21.5	10.6	22.5	14.0	21.8	15.8
Shadow testing[c]	75.2	86.1	74.9	85.3	72.9	81.0	71.3	76.8
Visible surface preprocess[c]	—	0.1	—	0.1	—	0.1	—	0.2
Color calculation[c]	2.2	2.6	2.5	2.9	3.4	3.5	5.0	5.3
Miscellaneous[c]	0.9	0.9	1.1	1.1	1.2	1.4	1.9	1.9
Total relative time	1.00	0.84	0.87	0.74	0.65	0.60	0.45	0.41

[a] Specifications: 38 items (835 polygons, 13 spheres, 15 cylinders); 5 light sources; adaptive ray-intersection tree depth.

[b] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.

[c] Shown as a percentage of total time

Note:

(1) The improvement as a result of using selected bounding volumes.

(2) Shadow testing once again becoming the dominant portion of the computational expense.

Table IX. Total Time Comparisons[a]

	Total[b]	C	CZ	CH	CHZ	S	SZ	SH	SHZ
Pencils I	6.02	1.0	0.67	0.86	0.54	0.54	0.40	0.32	0.28
Pencils II	11.94	1.0	0.79	0.79	0.62	0.76	0.65	0.37	0.34
Pencils III	17.07	1.0	0.78	0.82	0.66	0.51	0.44	0.28	0.26
Camera	6.61	1.0	0.92	0.90	0.80	0.67	0.60	0.53	0.49
Office	14.6	1.0	0.87	0.74	0.67	0.90	0.79	0.56	0.49
Pool Room	14.33	1.0	0.84	0.87	0.74	0.65	0.60	0.45	0.41

[a] C: Spherical bounding volumes (control); S: Selected bounding volumes; H: Hierarchical environment structure; Z: Visible surface preprocess.
[b] CPU time in hours for control image (*C*) on VAX 11/750 computed at a resolution of 256 × 240.

ACKNOWLEDGMENTS

The authors wish to thank Roy Hall for his groundwork, Channing Verbeck for his implementation of light sources, Gary Meyer for his assistance in color science, and Dave Immel for testing environments.

REFERENCES

1. APPEL, A. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the ACM National Conference* (Oct.), ACM, New York, 1967, pp. 387–393.
2. CLARK, J.H. Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10 (Oct. 1976), 547–554.
3. CROW, F.C. A more flexible image generation environment. In *SIGGRAPH '82 Conference Proceedings* (Boston, Mass., July 20–30). ACM, New York, 1982, pp. 9–18.
4. GOLDSTEIN, R.A., AND NAGEL, R. 3-D visual simulation. *Simulation* (Jan. 1971), 25–31.
5. HALL, R.A. A methodology for realistic image synthesis. Master's thesis, Cornell University, Ithaca, New York. 1983.
6. HALL, R.A., AND GREENBERG, D.P. A testbed for realistic image synthesis. *IEEE Computer Graphics and Applications 3*, 10 (Nov. 1983), 10–20.
7. KAJIYA, J.T. New techniques for ray tracing procedurally defined objects. *ACM Trans. Graph. 2*, 3 (July 1983), 161–181.
8. KAY, D., AND GREENBERG, D. Transparency for computer synthesized images. In *SIGGRAPH '79 Conference Proceedings* (Chicago, Ill., Aug. 8–10). ACM, New York, 1979), 158–164.
9. RUBIN, S.M., AND WHITTED, T. A three-dimensional representation for fast rendering of complex schemes. In *SIGGRAPH '80 Conference Proceedings* (Seattle, Wash., July 14–18). ACM, New York, 1980, pp. 110–116.
10. SCHUMACKER, R., BRAND, B., GILLILAND, M., AND SHARP, W. Study for applying computer generated images to visual stimulation. AFHRL-TR-69-14, United States Air Force Human Resources Laboratory, 1969.
11. SUTHERLAND, I.E., SPROULL, R.F., AND SCHUMACKER, R.A. A characterization of ten hidden-surface algorithms. *Comput. Surv. 6*, 1 (Mar. 1974), 1–55.
12. WHITTED, T. An improved illumination model for shaded display. *Commun. ACM 23*, 6 (June 1980), 343–349.

Received April 1984; revised July 1984; accepted July 1984