Portland State University

PDXScholar

Dissertations and Theses Dissertations and Theses

1988

Implementing ray tracing algorithm in parallel environment

Tjah Jadi
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds

Part of the Electrical and Computer Engineering Commons

Let us know how access to this document benefits you.

Recommended Citation

Jadi, Tjah, "Implementing ray tracing algorithm in parallel environment" (1988). *Dissertations and Theses.* Paper 3935.
https://doi.org/10.15760/etd.5818

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF Tjah Jadi for the Master of Science in Electrical and Computer Engineering presented May 23, 1989.

Title: Implementing Ray Tracing Algorithm In Parallel Environment

APPROVED BY MEMBERS OF THE THESIS COMMITTEE:

[redacted]

Faris Badi'i, Chairman

[redacted]

Jack C. Riley

[redacted]

Michael A. Driscoll

[redacted]

Bradford R. Crain

Ray tracing is a very popular rendering algorithm in the field of computer graphics because it can generate highly-realistic images from three-dimensional models. Unfortunately, the computational cost is very expensive. To speed up the rendering process we present both static and dynamic scheduling (balancing) strategies for a multiprocessor system. Hence, the load balancing among the processors is the most important problem in parallel processing. The implementation of the algorithm is based on a modified octree structure.

IMPLEMENTING RAY TRACING ALGORITHM
IN
PARALLEL ENVIRONMENT

by

TJAH JADI

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University
1989

TO THE OFFICE OF GRADUATE STUDIES:

The members of the Committee approve the thesis of Tjah Jadi presented May 23, 1989.

[redacted]

Faris Badi'i, Chairman

[redacted]

Jack C. Riley

[redacted]

Michael A. Driscoll

[redacted]

Bradford R. Crain

APPROVED:

[redacted]

Rolf Schaumann, Chairman, Department of Electrical Engineering

[redacted]

William Savery, Vice Provost for Graduate Studies

ACKNOWLEDGEMENT

I would like to give special thanks to my advisor, Dr. Faris Badi'i, for giving me the opportunity to develop this Ray Tracing algorithm. He also provided guidance in the management of the research work and assisted with concluding my thesis.

In addition, I wish to thank a number of faculty and staff members and graduate students in the Electrical and Computer Engineering Department for their help and time. In particular, Ms. Shirley Clark, Mr. Jack C. Riley and Mr. Janaka Jayawardena are warmly appreciated for their help with my research and with the preparation of this thesis.

Portland, Oregon

T. Jadi

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION TO RAY TRACING

This chapter describes the principles of ray tracing. The ray equations and the light illumination models are discussed in detail in the second and third chapter respectively. The fourth chapter provides a short overview of the previous work that has been done in the field of ray tracing. The fifth chapter consists of details of ray tracing algorithm which has been implemented on Sequent's "Balance 8000" computer system. The last two chapters discuss results from synthesized image experiments, conclusions, and suggestions for future work that might be done.

Generating a highly-realistic visual image is one of the primary goals of computer graphics. A lot of effort has been spent in developing techniques that can be used to generate high-quality three-dimensional images. Such pictures have broad applications in the areas of art works, complex molecular models in biochemistry and physics, Computer Aided Design in engineering, project design in architecture, graphics presentations in advertising and visual effects in the animated film industry.

In computer graphics, ray-tracing algorithm is one of the most popular techniques to generate highly realistic three-dimensional synthesized images. Other methods, such as Z-buffer, list priority, area subdivision, and scan line hidden-surface algorithms either could not generate quality images as produced by a ray-tracing algorithm or they failed to create realistic effects, particularly in the area of shadows, reflections, and refraction effects. The basic elements that contribute toward realistic images are : three-dimensional view, shading, hidden surface removal, shadows, specular effects, reflection, refraction, and color texture.

The details of the above realism effects are explained below :

1. Three-dimensional view : Display the images in perspective or parallel view.

2. Shading : Determine the appearance of visible surfaces under an illumination of light, which causes different shades of intensity at different points on the surface.

3. Hidden surface removal : The portions of the object that are hidden and are not visible to an observer's eye are not shown.

4. Shadows : The shadows cast by the objects under an illuminated environment. In other words, shadow rays are traced from the visible surface in the direction of light sources in order to determine which surface could not be seen from the light sources.

5. Reflection, refraction, specular effects, and color texture : All of these effects play an important role in rendering a realistic image of objects with such surface characteristics.

The ray tracing algorithm can handle all of the above realistic features in a single algorithm. It is a simple way of rendering very high quality realistic images on a raster display device. This realism is further enhanced by the technique of distributed ray-tracing described by Cook and Torrance [1]. The ray tracing technique is also very powerful and easily extendable method available in computer graphics today. It can be used to simulate primitive objects like polygons, spheres, cylinders, cones, etc. and almost any type of geometrically defined surfaces, such as quadratic surfaces, polyhedral volumes, bi-polynomial parametric surfaces, cubic splines, etc.. It can also add a variety of new effects to the final picture.

Ray tracing algorithm implements a camera model to simulate the model of a physical environment. It is obvious that camera model is an appropriate approach to generate computer synthesized images on a two-dimensional raster display device. The camera model uses a viewpoint as its focal point and the two-dimensional image plane (pixels array or screen) represents the camera's film. This camera model is divided into two distinct views as described below.

- Parallel view : The viewpoint is located at an infinite distance from the screen.
- Perspective view : The focal point is located in an finite distance from the screen.

In this camera model, a viewpoint located on one side of the screen "sees" a modeling scene on the other, as described in Figure 1.

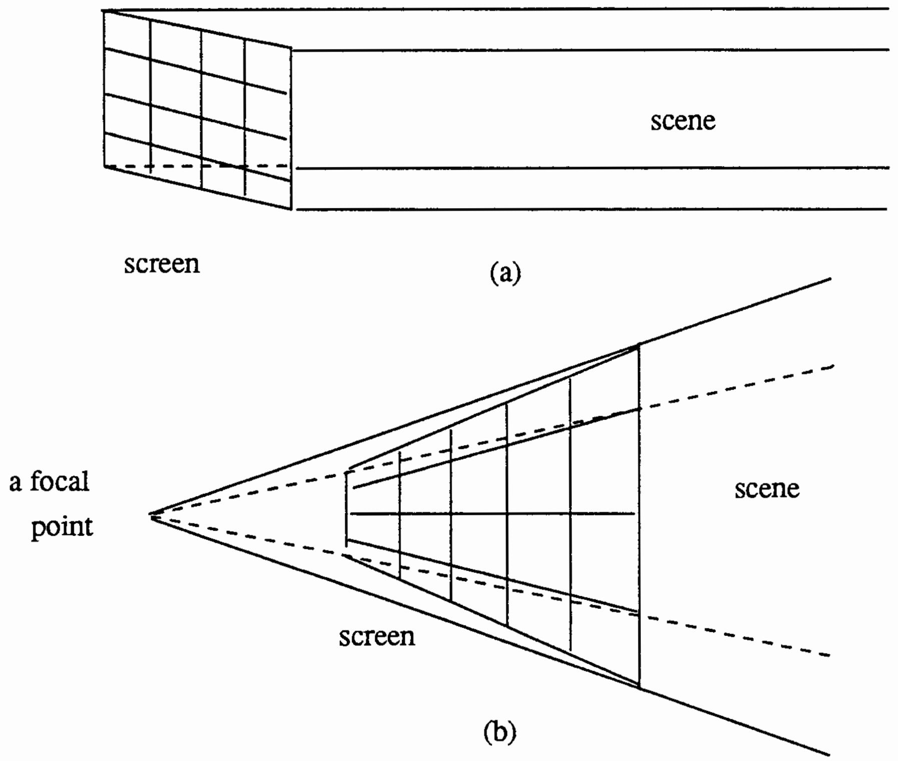


Figure 1. Typical camera models : (a) parallel and (b) perspective views.

Ray tracing is a brute force technique. The basic fundamentals of a ray tracing are described as follows. The ray tracing techniques are modeled by casting a ray from a viewpoint to a pixel on the image plane. It traces the light ray's path backward from the viewpoint through points on the image plane, further into the scene to identify the surface visibility and until to the light source. Some rays are reflected and refracted. They may encounter more than one surface of the scene before reaching the light source thus altering the surface visibility. The light ray paths from a viewpoint, through a pixel ("n" number of pixels per ray), into the scene and unto the light source, is shown in Figure 2.

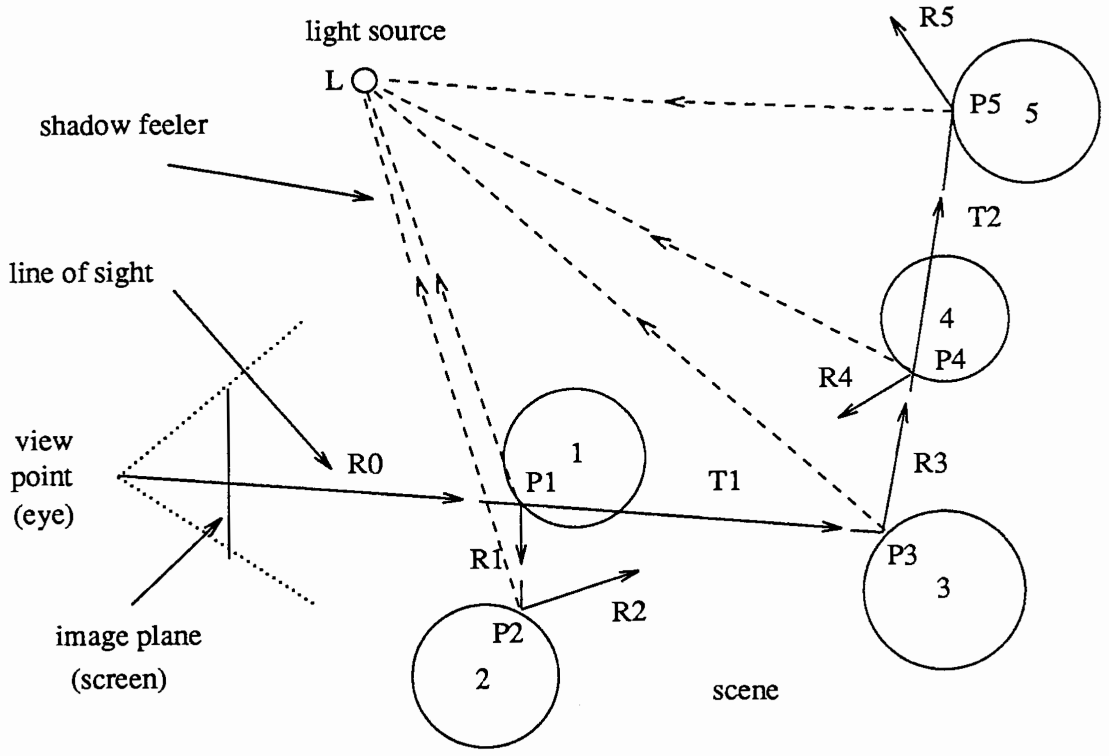


Figure 2. Ray tracing surface reflections and refractions.

As shown in Figure 2, rays are cast from the viewpoint into every pixel of the image plane and traced as they are reflected and/or refracted by the objects. When a ray strikes an object, it may generate new, secondary, rays due to the current surface attributions. If the the secondary rays are included, these rays must further trace into the scene. Therefore, the hierarchy of an intersection tree becomes very useful for capturing all the intensity contributions of each ray to simulate the realism effects on the images. As each ray encounters an object along it's path, a new tree node must be created. In this case, the branch nodes in the tree represent the intersection of the ray with objects. The leaves represent either light sources or rays leaving the surface.

However, in reality, an observer sees an object when light rays propagating from the light sources strike the objects in the scene. After the rays interact with the model of a physical environment, some of them travel through the viewing plane and finally reach the observer's eye. As a matter of fact, an infinite number of rays from the light sources to the observer's eye can be traced. Only those rays that travel through the pixels on the picture plane and reach the viewpoint generate an image. Hence, most of the light rays will never reach the viewer's eye. Consequently, the process of following all of the rays from the light source could become computationally very inefficient.

When the viewpoint is located at an infinite distance from the screen (image plane), the computation of a ray tracing algorithm becomes very simple, because all the incoming primary rays are aligned with the z axis of world coordinate axis. This arrangement will simplify the primary ray-surface computing intersections.

To produce a synthesized image for a simple opaque surface criterion, each light ray's path must be traced to determine which objects in the scene, are intersected by the ray. Thus, every object in the environment must be examined for every ray. If a ray intersects an object in the scene, the intensity at that particular point on the surface is calculated by using the attributes of the object. Its intensity value will be taken as the current sample pixel's intensity. However, if a ray had multiple intersections with multiple objects in the scene, the intersection points are sorted in depth order with respect to minimum distance from the current point under consideration. The visible surface is the closest distance from the ray's origin. For example, the nearest intersection point from the viewpoint (ray's origin) will represent the visible surface for that current sample pixel. However, if there is no intersection, the current sample pixel intensity value will equal to the background intensity. The intensity value of the current sample pixel under consideration will be stored in a file, namely an image file. This procedure is repeated for each pixel on the picture plane. Once the intensity of all the pixels has been calculated, the results of the synthesized picture can be displayed on the screen in such a way that they can be perceived correctly. This technique can also be used to generate wire frame line drawing pictures for solid objects in a modeling environment.

However, if the researchers include the realistic features under consideration, such as shadows, reflections, and refractions, then each pattern of the light ray's path must be recursively traced to establish at what surface the ray intersects. In this case, a shadow test for each current intersection points has to be performed. When the ray encounters a surface, three things may occur, depending on the surface attributions as follows :

1. An opaque surface : The current sample pixel's intensity is equal to the intensity at that intersection point on surface.

2. A reflective surface : The new reflected ray is generated in the direction of mirroring-reflected (the mirror reflected) model for which the angle of incidence and reflection are equal. This ray has to be traced further from the current intersection point through the environment.

3. A transparent surface : The new transmitted ray is generated while passing through the surface. It traces through the intersected surface of a translucent object. This ray may also be refracted by the object attributions which is called refracted ray.

In Figure 2, the line of sight R_0 is generated by casting a ray through a pixel of image. When the ray R_0 strikes object 1, it generates a branch node and three new rays R_1, T_1, and LP_1. Ray LP_1 terminates in the light source which is a leaf. Ray R_1 intersects object 2 and creates a branch node and two new rays R_2 and LP_2. Ray T_1 encounters object 3 and generates another branch node and two new rays R_3 and LP_3. After tracing all the rays, a hierarchical tree is constructed with a depth level of four. A leaf is a terminated node where a ray leaves the scene or reaches the light source, such as R_5, LP_1, etc.. On the other hand, when a branch node is represented, the intersection between the ray and the object in the scene has to be traced further into the scene.

Both reflected and refracted rays are named the secondary light rays. Their effects will make, fully or partially, a contribution to the pixel's intensity. The primary rays are created by casting rays from the viewpoint to the points on the image plane. In general, when a ray strikes a surface of an object it may be spawned into three new rays due to the surface criterions. This means that each time a ray leaves an object, there are up to three new rays that have to be traced. They include the diffusion ray, reflection ray, and refraction ray, as depicted in Figure 3.

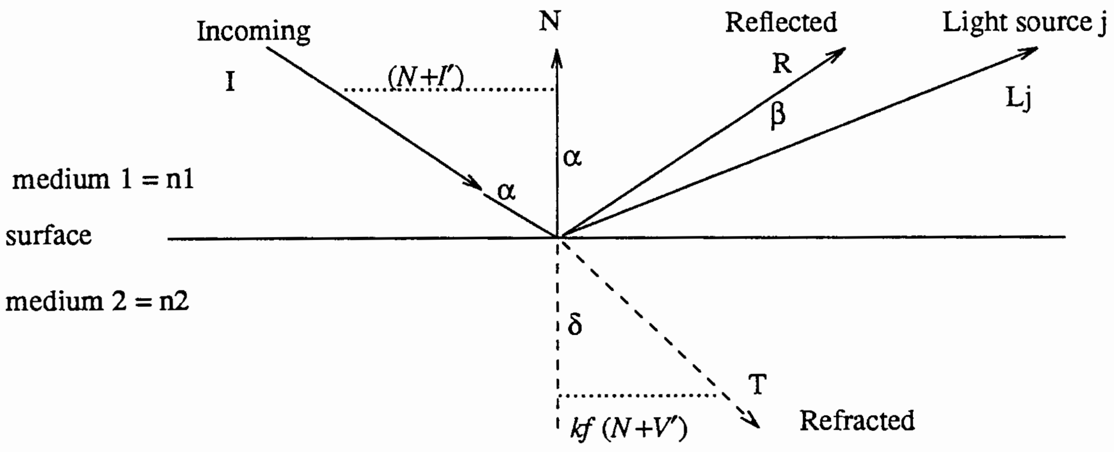


Figure 3. Geometry of reflection and refraction effects. The diffuse reflected ray is not included because it is scattered equally in all direction.

The calculation of the global illumination model does not end at the first intersection. Hence, the incoming ray I is reflected from the surface in the direction R and transmitted through the surface in the direction T as depicted in Figure 3. At each ray-surface intersection, the direction of the reflected and the refracted ray can be obtained by using the geometric optic's law as follows :

$$V' = \frac{I}{|I \cdot N|} \tag{1.0a}$$

$$R = V' + 2N \tag{1.0b}$$

$$T = kf\,(N + V') - N \tag{1.0c}$$

$$kf = (Kn^2\,|V'^2| - |V' + N|^2)^{-(1/2)} \tag{1.0d}$$

where :

$\mathrm{Kn} = \dfrac{n_2}{n_1}$ = the index of refraction
N = the unit normal vector
I = the incident ray unit vector
R = the reflected ray unit vector
T = the refracted ray unit vector

These three new rays occur due to the surface attributes of the object in the scene. Unfortunately, diffuse-reflected light generates an infinite number of rays in all directions. Therefore, the researchers traced rays from reflected and/or refracted light only. By recursively tracing these reflected and/or refracted rays, researchers could add considerable realism effects into the final image. The process, illustrated in Figure 2, is easily represented by using the tree structure as shown in Figure 4. However, this process is computationally very expensive.

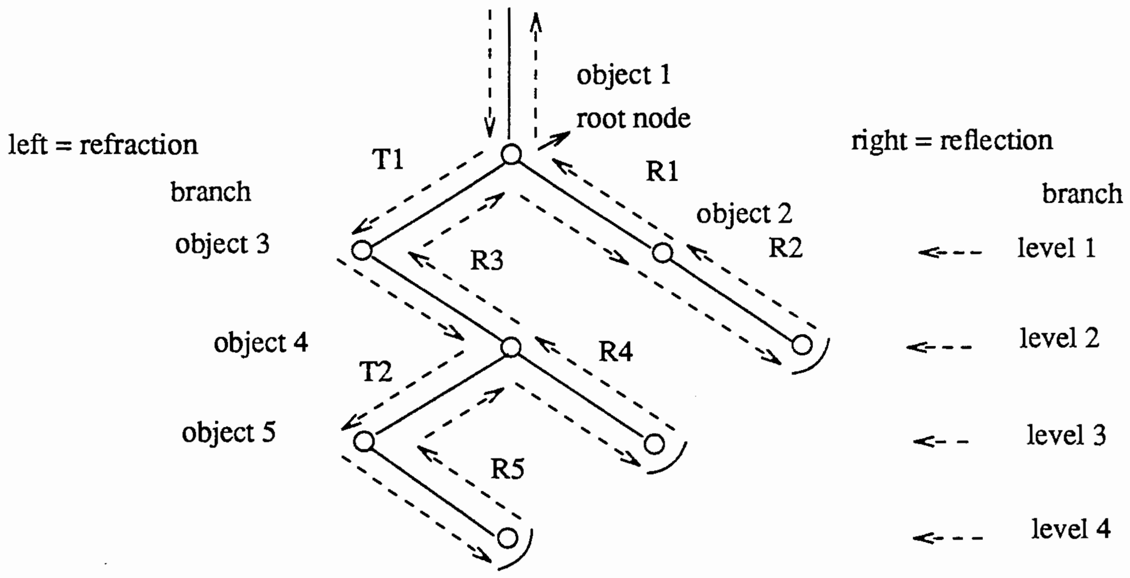


Figure 4. A hierarchy of the intersection tree.

As a ray is traced through the environment, the tree structure of ray-surface intersection is constructed along the way for each pixel on the image plane, because each of the intersection nodes contributed to the intensities of the root node. Each of the intersection nodes also added visual realistic effects into a final image. Thereby, after establishing an intersection tree, the final pixel intensity is determined by traversing the tree and summing the intensity contribution of each node from the bottom up according to the reflection model, as shown in Figure 4 by the dashed with arrows.

With reference to Figure 4, as intersection nodes are created in the tree, the resulting intensity at the current intersection point is calculated. The secondary ray, due to the attributions of the current surface, are also created. The operation of the intersection tree first generates the tree node along the left hand "refraction" branch from the root node until the branch terminates. The branch is then traversed upward, summing the intensity at each node until the root node is reached. The right hand "reflected" branch from the root node is then generated and traversed in the direction as shown in Figure 4. In other words, the downward pointing arrows indicate ray generation, and the upward pointing arrows indicate intensity generation. The root node intensity is stored into the image file because it contains the total intensity contributions.

If the above step includes shadow testing, it has been discovered to be very expensive computationally, particularly for environments that contain complex lighting schemes.

The ray tracing execution time is linearly increased as a function of variables named below :

- the number of surfaces in the scene,
- the number of light sources in the environment, and
- the size of a picture resolution.

All of these factors control the processing time. For example, the large physical sized image requires that a larger number of rays have to be traced and more time is required to generate an elegant image. The researchers also know that added realistic features, such as testing shadow, reflections, and refractions on the final image are computationally very expensive. This implies the intensity calculation step to be performed more times, thus, increasing the processing time. In other words, there is a trade off between highly-realistic images and the time spent to generate a picture synthesis. Therefore, rendering the scene with moderate complexity takes an enormous amount of the CPU time.

The prominent drawback of the ray-tracing technique is that it is extremely slow, because each ray that passes through the picture plane has to be tested against all the objects in the scene. This process will also find all possible intersections between rays and objects in the scene, but it is computationally very expensive. This may result in multiple intersections for multiple objects. If it is the case, then the intersection points are sorted in depth order with respect to a minimum distance to the current sample point under consideration. The computations involved in the ray-tracing technique grow linearly with the numbers of the objects in the scene. Whitted [2] indicated that time spent in examining the ray-surface intersection in the scene could use as much as ninety percent of the total computing time for generating an image. The ray tracing algorithm generally requires an enormous number of floating-point calculations. It is one of the computer graphics techniques that produces undesirable aliasing or sampling-point effects on the final image, which is a common problem with raster display algorithm.

The efficiency of the intersection routine has significant impacts on ray-tracing algorithms, because the most important factors of this method are to determine the visible surface in the environment. The amount of time needed to examine an object and a ray intersection depends on the objects' geometrical descriptions. It is very simple to determine the ray-surface intersection of typical objects, such as spheres, cylinders, cones, and polygons. A sphere has the simplest ray-surface intersection test. The intersection time for polyhedron objects generally depends on the numbers of the faces that it contains. Others shapes may require more time to check the ray-surface intersection. Unfortunately, in general, scenes do not consist of, easy to describe, simple objects, but instead, they are composed of a mixture of complex objects with complicated description; i.e., polyhedral volumes, quadratic surfaces, and bi-polynomial parametric surfaces. For this reason, it is often desirable to enclose complex objects' descriptions with simpler ones, such as rectangular parallelepipeds, spheres, and cylinders. The bounding volume method was first suggested by Clark [3] and later implemented by Whitted [2]. The common practice of using the bounding volumes on the basics of the simplicity of the ray-surface intersection tests, should not be the only consideration. Additional factors that have to be considered are : projecting the outermost bounding volume's vertices into the viewing plane to determine a rectangular region whose its rays have high potential to intersect the objects in the scene, using a cut off contribution threshold intensity to control the depth of the reflection and/or refraction hierarchical tree structure adaptively, and implementing a ray tracing algorithm in parallel processing. Therefore, The researchers must find a way to decrease the time spent in the ray-surface routine and make it more efficient. Most of the research in this area is concentrated on a better and more economic method in cutting down the time spent in intersection routine.

CHAPTER II

DETERMINING THE RAY EQUATION

In this algorithm, light rays are represented by straight line equation with a length parameter. All of the rays start at the viewpoint and pass through pixels on the image plane. The equation for each ray is described as follows :

Consider that the point of origin, P_0, of the ray is the viewer's eye. The coordinate of this viewpoint are (X_0,Y_0,Z_0). The next point, P_1, is a pixel on the viewing plane with coordinates of (X_1,Y_1,Z_1). Casting a light ray from the viewer's eye position to a point on the image plane generates a light ray's path (line of sight). Therefore, the current point $P_{current}(X_c,Y_c,Z_c)$ lies on the line of sight and its coordinate can be written as shown in equation (2.0) :

$$P_{current} = P_0 + (P_1 - P_0) \times t \qquad (2.0)$$

or this can be expressed in rectangular components, as described below :

$$X_c = X_0 + (X_1 - X_0) \times t \qquad (2.1a)$$
$$Y_c = Y_0 + (Y_1 - Y_0) \times t \qquad (2.1b)$$
$$Z_c = Z_0 + (Z_0 - Z_1) \times t \qquad (2.1c)$$

where t is parametric of a point on the line ($0<=t<=\infty$), and the slope or direction of the parametric line is given by dx, dy and dz where :

$$dx = (X_1 - X_0)$$
$$dy = (Y_1 - Y_0)$$
$$dz = (Z_1 - Z_0)$$

The above equations become :

$$X_c = X_0 + dx \times t \qquad (2.2a)$$

$$Y_c = Y_0 + dy \times t \tag{2.2b}$$
$$Z_c = Z_0 + dz \times t \tag{2.2c}$$

In producing a computer-generated image, the researchers examine the picture plane per scanline and per pixel from left to right and from top to bottom.

2.1 DETERMINING VALID RAY-SURFACE INTERSECTIONS

A valid intersection of a ray and a surface occurs when the following conditions are satisfied :

1. The intersection point lies on the surface.

2. The intersection point lies within the boundaries of the surface.

The procedures for checking the validity of the intersection are as follows : Let $P_0(X_0,Y_0,Z_0)$ and $P_1(X_1,Y_1,Z_1)$ be the coordinates of the viewpoint and the current pixel under consideration on the picture plane respectively.

In this case, the ray-surface intersection test is divided into the three sections which are described below.

2.1.1 The Ray-Polygon Intersection Test

Let the equation of the plane and the ray in equation (2.0) be described below :

$$plane: \quad A \times X + B \times Y + C \times Z + D = 0 \tag{2.3}$$

where, A, B, C, and D are constants. The A, B, and C are the components of the normal vector to the plane.

$$ray: \quad P_{current} = P_0 + (P_1 - P_0) \times t$$

or

$$X_{current} = X_0 + dx \times t$$
$$Y_{current} = Y_0 + dy \times t$$
$$Z_{current} = Z_0 + dz \times t$$

The light of sight is the ray directed from the viewpoint through the pixel in the image plane. The distance along this ray is determined by the parameter t. The value of the parameter t has to be greater than zero for given ray to intersect the objects in the scene.

The equation for the light of sight is substituted into the plane equation to find the value of the parameter t for the intersection of the line of sight and the plane in the scene. The t-value for the intersection point lies on the surface of the plane is given by :

$$t = -(A \times X_0 + B \times Y_0 + C \times Z_0) - (A \times dx + B \times dy + C \times dz) \qquad (2.4)$$

If the value of t in equation (2.4) is positive, it satisfies condition (1) as stated above. To satisfy the condition (2), the "point-inside-polygon" test can be performed to see if the current point is contained within the boundaries of the polygon on the plane surface. The procedure will be used in situations where the current point is the intersection point of the line of sight and the plane surface. The test can be performed by using the vertex-loop containment algorithm, which is based on the "ray-firing" technique. It is described as follows :

An infinitely long line is constructed on the plane, starting at the current point and passing through a test point on the boundary of the surface. The test point is initially selected to be the center-point of one of the edges of the polygon as shown in Figure 5. The number of intersections between this test line and all of the sides of the surface being checked are counted. If this number is found to be even, the point is declared to be outside the surface, otherwise it is within the surface therefore on the object.

Consider Figure 5, where ABCDEA represents a typical surface. If S is the current point and J is the test point, the line SJ can be seen to intersect twice with the edges of ABCDEA, along DC and EA. The two intersections confirm that S is outside the polygon. Similarly, for the current point R and the test point H, the line RH intersects

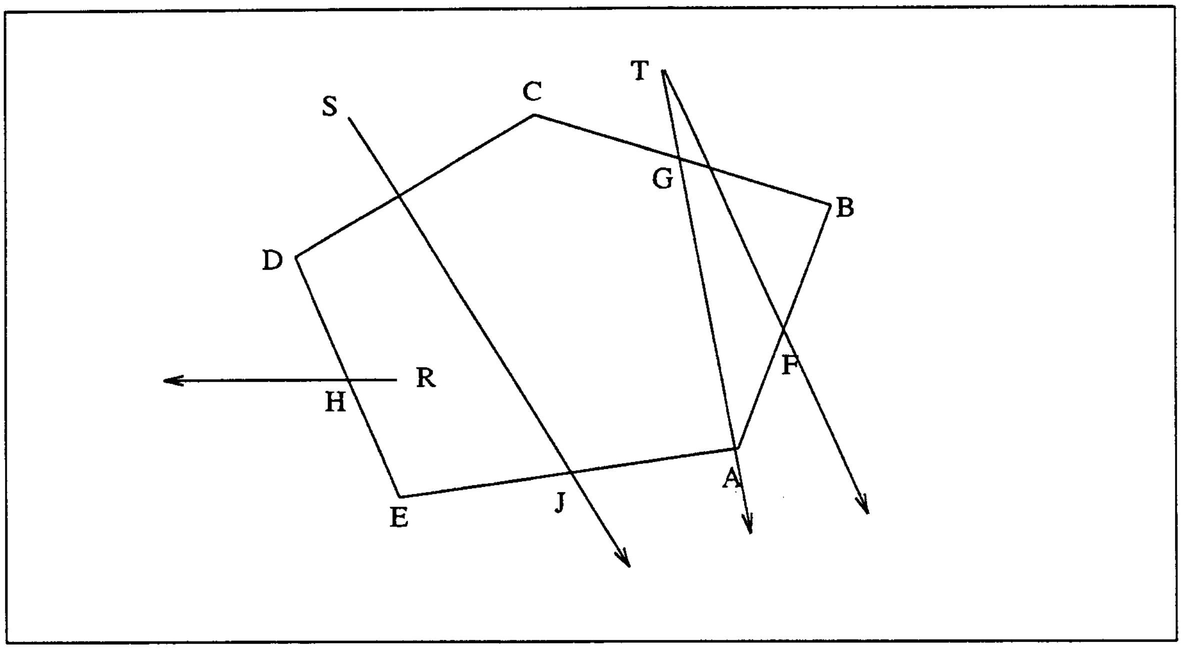


Figure 5. Testing for points inside a polygon.

only once with ABCDEA, at DE. Therefore, R is declared to be inside the polygon ABCDEA.

A difficulty arises when the infinite line of a ray passes through a vertex or coincides with an edges of the polygon. For example line TG passes through vertex A. This line could be considered to intersect ABCDEA at three places, at G with side CB, at A with side EA and at A, again, with side AB. Therefore, point T would be declared to be inside, although it actually lies outside the polygon ABCDEA.

The remedy for this problem is to change the test point which changes the infinite line of a ray automatically. The center-point of the next edge of the polygon is selected as the test point and the process is repeated; e.g., instead of G, F is considered to be the new test point. The above problem can also be avoided if the line intersection with the vertex is counted as one crossing instead of two. Therefore, point T could be considered outside the polygon ABCDEA because line TG has two intersections point at G with side BC and at A with either side EA or AB. The vertex-loop containment procedure can be made somewhat more efficient by including a "pre-inside" test. The initial test is performed by checking the current point against all the vertices of the polygon. This assures that the result of vertex coincidence would be noticed. Therefore, in such a case, there would always exist a test point for which the infinite line would not pass through any vertex.

The current point under consideration is a valid intersection point if it satisfies both conditions (1) and (2).

The surface normal, which is needed for intensity, reflection and refraction calculations, is equal to a cross product of two vectors on the plane. For example in Figure 5, the vector $\vec{N}$ can be written as follows :

$$\vec{N} = \vec{AB} \times \vec{AE}$$

In this case, $\vec{N}$ has direction out of the paper.

2.1.2. The Ray-Box Intersection Test

The ray-box intersection test can be accomplished by using the ray-polygon test previously discussed. A given ray can pierce any of the six sides of the rectangular box. Therefore, in finding a valid ray-box intersection, a given ray has to be tested against six of the box's faces and found the closest intersected point to the ray's origin, as depicted in Figure 6. These procedures can be divided into four steps as described below.

1. Determine the intersection of the ray with the plane of the box side.

2. Check the current intersection point in step 1 to see if it lies within the box side.

3. Repeat both step 1 and 2 for every side of the box.

4. Find the nearest distance, smallest value of t, of the results of step 3 from the ray's origin.

Consider Fig 6, where ABCDA represents a two-dimensional box. A given ray R could be considered to intersect at four places, at E with line AB, at F with line BC, at G

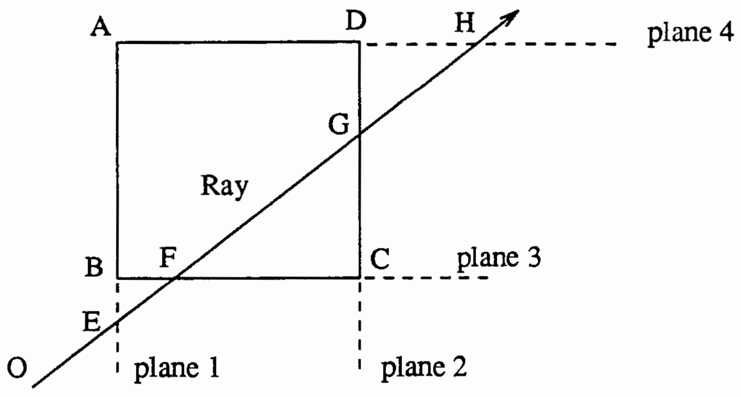


Figure 6. Finding a valid intersection point which lies within the a box.

with line CD, and at H with line AD. However, both intersection F and G are the only points which lie on the ABCDA box. However, point F is the valid intersection point of the ray R and the ABCDA box because it has the closest distance from the ray's origin O. The above processes could be computationally very expensive.

2.1.3. The Ray-Sphere Intersection Test

Let the equation of a sphere be described as follows :

$$sphere: \ (X - X_c)^2 + (Y - Y_c)^2 + (Z - Z_c)^2 = R^2 \tag{2.5}$$

where $X_c, Y_c, \text{and} Z_c$ are the coordinates of the center of the sphere and R is the radius of the sphere.

The parametric line in equation (2.0) is substituted into equation (2.5) and solved for the value of the parametric t. With reference to Figure 7, the t-value can be written as described belows :

IF (b - disc) is less than zero THEN

$$t = b + disc \tag{2.6a}$$

ELSE

$$t = b - disc \tag{2.6b}$$

where :

V(x, y, z) = Center(x, y, z) - $P_0(X_0,Y_0,Z_0)$

b = a dot product of a vector V and a vector of the ray's direction

P_0 and D are the origin and direction of the ray.

$$b = \vec{V} \cdot \vec{D} \tag{2.7}$$

$$disc = b \times b - (\vec{V} \cdot \vec{V}) + R^2 \tag{2.8}$$

There is a valid intersection point, if the value of both disc and parameter t are greater than zero. The surface normal vector is equal the vector from the sphere center to the intersected point, $\vec{CI}$, as described in Figure 7.

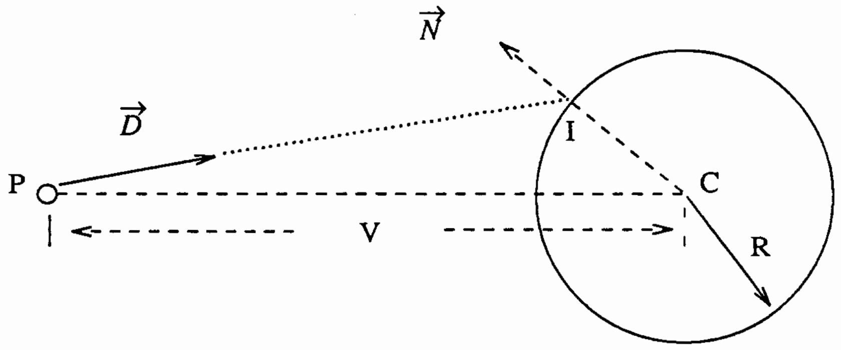


Figure 7. Finding an intersection point on a sphere.

CHAPTER III

A SIMPLE ILLUMINATION MODEL

In computer graphics, the illumination model requires the ability to model the physical behavior of the light ray particularly within the visible spectrum. Hence, the researchers should simulate the way light would propagate from the light source to the modeling environment which passes through pixels on the image plane, and finally reaches the observer's eye. This lighting model can be implemented by using the mathematical models of the physical laws governing the use of electromagnetic radiation. Once the physical behavior of the light ray has been established, the researchers can simulate the three-dimensional modeling environment.

When an incoming light ray strikes an object, part of its energy will be absorbed by the surface of the object being hit. The rest will be reflected and/or transmitted, thus making the object visible. The amount of the light absorbed, reflected, and/or transmitted depends on the surface characteristics. For example, some objects have shining surfaces, some have dull, or matte surfaces, some have opaque surfaces and others are made of transparent material. See Figure 3.

The illumination model is used to determine the intensity at the intersection point between a ray and an object in the scene. The intensity value is either displayed on the viewing plane or contributed to another intersection point on the ray-surface intersection tree structure, as described in Figure 4. It is divided into two parts :

- Local and
- Global illumination.

3.1 LOCAL ILLUMINATION

Light reflection as a result of direct illumination from the light source is called "local illumination".

The local illumination models treat reflection as consisting of three components : ambient, diffuse and specular intensity. The details of their characteristics are described below.

3.1.1 Ambient light

The ambient component represents light that produces a constant illumination on all surfaces, regardless of their orientation.

3.1.2 Diffuse light

Diffuse reflected light can be considered as an incident light that is radiated from the surface where the ray strikes. This light is evenly scattered in all directions and its intensity value depends on the surface normal and geometry of the light source. Hence, the position of the observer's eye does not affect the amount of reflected intensity seen by the observer. The equation for the contribution of a diffuse light is written as follows :

$$Diffuse\_Intensity = \vec{N} \cdot \vec{L} \tag{3.0a}$$

or

$$Diffuse\_Intensity = |N| \times |L| \, cos(\alpha) \tag{3.0b}$$

For multiple light sources the Eq(3.0a) can be written as Eq(3.0c).

$$Diffuse\_Intensity = \sum_{i=1}^{i=j} (\vec{N} \cdot \vec{L_j}) \tag{3.0c}$$

3.1.3 Specular light

The highlighting of the shining objects is due to specular reflection of light. The intensity of the reflected value is focused along the reflected vector. Therefore, the position of observer determines where the highlighted area appears to be. In other words, the value of specular light's intensity depends on the coordinate of the viewer's eye. Specular reflectance is directional and its intensity value depends on both the reflected light vector and the viewing direction of the current point under consideration. The equation used for finding the specular light contribution is given below.

$$Specular\_Intensity = (\vec{L} \cdot \vec{R})^n \tag{3.1a}$$

or

$$Specular\_Intensity = (|L| \times |R| \cos(\beta))^n \tag{3.1b}$$

For multiple light sources the equation (3.1a) becomes :

$$Specular\_Intensity = \sum_{i=1}^{i=j} (\vec{R} \cdot \vec{L_j})^n \tag{3.1c}$$

The Whitted illuminated model for local illumination can be written as follows :

$$I_l = I_a + \sum_{i=1}^{i=j} (\vec{N} \cdot \vec{L_j}) + \sum_{i=1}^{i=j} (\vec{R} \cdot \vec{L_j})^n \tag{3.2}$$

where :

- α is the angle between the surface normal and the line of light.
- β is the angle between the line of light and the reflected ray.
- |N| is the magnitude of $\vec{N}$
- |L| is the magnitude of $\vec{L}$
- |R| is the magnitude of $\vec{R}$
- I_l is the local intensity contribution at the current point.
- n is the specular shining factor.
- j is the number of the lights source.
- I_a is the effect contribution of the ambient intensity.

3.2 GLOBAL ILLUMINATION

The effect of indirect illumination from other portions of the surfaces in the environment is referred to as "global illumination". In practice, the indirect effects come from the secondary rays that are reflected and/or refracted due to the properties of intersected surfaces. These visual effects, such as specular mirroring and transparency contribute to the realism computer generated-images.

In fact, a ray of light leaving the surface of an object is the sum of local and global intensity contributions. Each adds a new component to the intensity of the point being shaded. This can be described as follows :

3.2.1 Adding reflection effects

$$I = I_l + k_r \times I_r \tag{3.3}$$

3.2.2 Adding transparent effects

$$I = (1.0 - k_t) \times I_l + k_t \times I_t \tag{3.4}$$

where :

- k_r is the reflective of the surface being hit. Its value is between 0.0 and 1.0.
- $k_r = 0.0$ means the surface is opaque and
- $k_r = 1.0$ means the surface is mirror.
- k_t is the transparency of the current surface, which is range from 0.0 to 1.0.
- $k_t = 0.0$ means the surface is opaque.
- $k_t = 1.0$ means the surface is completely transparent.
- I is the total intensity at the current point under consideration.
- I_l is the local intensity contribution at the current point.
- I_r is the intensity of the object hit by the reflected ray.
- I_t is the intensity of the surface struck by the transmitted ray.

The following is a short overview of some previous works that have been done.

CHAPTER IV

THE PREVIOUS WORK

The technique of ray tracing was first suggested by Appel [4], and later was implemented by MAGI (Mathematics Application Group, Inc) [5] to solve the hidden surface removal problems. MAGI used a term "Geometric Ray Tracing" which is commonly referred to as ray tracing algorithm. They mentioned the use of a camera modeled to simulate the photographics process in reverse order. They also observed that the computation time is largely functions of the size of the image resolution, the number of the light sources in the scene, and complexity of the geometric model to be rendered. The Lambert's cosine law is used for calculating the pixel intensities on the picture plane.

Phong [6] proposed a reflection model that could be used to solve many of the shading problems and this technique has significantly improved the realistic effects of image synthesis. This model includes the specular term which provides the realistic highlights from the direct light source reflections. It also supports a multiple light sources.

In an attempt to overcome the global illumination effects, Whitted [2] extended the global illumination as mentioned by the Appel. He proposed an improved model for calculating the intensity of a single ray. His model is based on both the Lambert's Cosine Law and Phong's reflected model for shading. This improved model takes into account the intensity contributions due to the ambient light, the reflected rays, and the refracted rays. The intensity effects of global illumination must be stored in the branches of a ray-surface intersection tree. After establishing the ray-surface intersection tree, the shader then traverses the intensities contributions of the branches on the tree from the bottom up to determine the total intensity on each pixel of the image. Consideration of all of these factors allows the shader to accurately simulate true shadows, reflections, and refractions on a graphical image synthesis.

The geometries of the bicubic patches, polyhedral volume, quadratic surfaces and bipolynomial parametrics surface are usually complex and varied geometrically defined surfaces. For this reason, it is often desirable to enclose the complex object in the scene with a simpler one which is called bounding volume. Whitted used spheres as a bounding volume for the object in the scene. If a ray does not intersect the bounding volume of an item in the scene, then the object can be discarded from further consideration for that given ray. However, if a ray intersects the bounding volume, then that given ray has to be tested against an item residing within the bounding volume. Whitted also introduced an antialiasing technique which defined a given pixel as a square region whose corners are represented by four sample pixels. The intensities at the four corners of a pixel are found out; if their intensity values are not nearly equal to each other, then the four sample pixels of a region is subdivided into four subsquares. This process recursively divides the sample region until an adequate amount of information about the details within the sample region is obtained. The intensity contribution of each subregion is weighted by its area, and the results summed to obtain the pixel intensity. On the other hand, if the intensity values of the four sample pixels are nearly equal and no item lies in the region between them, then the average of the four values is a good approximation of the intensity within the region under inspection. Whitted also stated that the most time-consuming step in this algorithm is that of finding the ray-surface intersections which could take up to ninety five percent of the total computation time.

Whitted and Rubin [7] introduced the hierarchical representation for three-dimensional objects which have improved both time and space efficiency. These representations typically consist of the trees whose branches represent bounding volumes and whose terminal nodes represent primitive object elements. This homogeneity structure allows the visible surface rendering to be performed simply and efficiently. The advantage of this hierarchical representation is to cut down the visibility calculations. This is done by searching through the tree structure which corresponds to the terminal level bounding volumes and the current pixel. The bounding volumes chosen for this method are rectangular parallelepipeds oriented to minimize their size. During the picture generation, each ray is transformed to align with the axes of the rectangular parallelepiped, so that the intersection test reduces to simple comparison against the boundary of the bounding volume. As a result, memory requirements are minimized by expanding or fetching the lower level of the hierarchical tree only when it required.

Hanhara [8] has outlined a method that uses a symbolic algebraic system to derive the equation for the intersection between the ray and algebraic surfaces automatically. An exact polynomial root finding algorithm can be used to solve the previous equation discussed above. Many interesting surface can be represented by low degree of polynomial functions in three-dimensional coordinates.

Hall and Greenberg [9] introduced the concept of adaptive tree-depth control, which primarily cut down the ray-tracing computation time. Traditionally, ray intersection trees for all sample point are constructed to an arbitrary depth to ensure that all relevant reflections and refractions are captured in the final computer generated images. However, the upper bound of the contribution of any nodes in the intersection tree to the final intensity of the sample point can be determined according to the diffusive, specular, reflective, and transmissive properties of the intersected surface. Thus, by establishing a cut-off contribution threshold, the tree depth can be adaptively controlled during the ray tracing process. Statistics show that this algorithm will significantly reduce the rendering process without effecting the image, even for highly reflective environments.

Kajiya [10] presented a new improved intersection algorithm for ray tracing of three types of procedurally defined objects. They are fractal surfaces, prims, and surfaces of resolution. The fractal surface algorithm performs recursive subdivision adaptively. In this algorithm, Kajiya used the probabilistic "extents" of the fractal surface to eliminate unnecessary intersections tests. The fractal surfaces can be treated as bounding volumes prior to the fractal surface definition. If a ray intersects the bounding volume, it must be examined more closely. It means that the ray has to be tested against all of the subsurfaces inside the bounding volume. If the subsurfaces do not intersect for a given ray, they are discarded from further consideration. In this way, the ray has to be investigated against only a handful of polygons instead of the large collection of polygons making up the entire fractal surface. However, if there is no intersection between a given ray and the bounding volume, the bounding volume is pruned from further consideration. This procedure will decrease the processing time. A prim is an object generated by translating a plane curve along the vector for a given distance. For both, prim and surface of resolution, the three-dimensional ray-surface intersection problem is transformed into a two-dimensional problem which is solved by the strip tree method.

Weghort, Hooper and Greenberg [11] brought out an important point concerning an object residing within a bounding volume. The bounding volume should be chosen in such a way that the volume occupied by the object inside the bounding volume should be maximum with respect to the bounding volume itself. If the bounding volumes are selected wisely, this method can substantially reduce the cost of the intersection testing. Certain objects like spheres, cylinders, and rectangular parallelepipeds do not require bounding volumes because each type of these objects can be served as its own bounding volume. For complex objects a set of points which surround the object is determined. These points define the vertices of some polyhedron which completely encloses the object. The spheres, cylinders, and rectangular parallelepipeds are the candidate shapes which are proposed. A bounding volume for each of the three candidate shapes passing through the set of points is proposed. The purpose of a selection, among these three candidates shapes, is to maximize the ratio of the volume of the object to the volume of the bounding volume. An interactive program is provided to override any automatic bounding selection. Weghort, Hooper, and Greenberg also constructed a hierarchical environment by grouping objects which are in close proximity with each other. Higher level clusters are created by grouping clusters and/or objects together. When a ray intersects the outermost bounding volume which is the root node of the hierarchical tree, first the root node is tested and the hierarchical tree is recursively descends only along those branches for which intersection occurred. The authors also caution that the advantage of the hierarchical approach is lost if, by chance, objects which are not close to each other are placed in a single cluster. In addition to the above, an "item buffer" is constructed. A typical entry in the item buffer corresponds to a particular pixel in the image plane and contains the information about the closest object for a ray starting at the viewpoint and passing through a particular pixel. This information is generated by using the Z-buffer algorithm which is modified to meet the need of producing an item buffer. The process of finding the first intersection of a ray with objects is essentially that of finding a visible plane which is closest to the origin of the ray; the item buffer contains exactly the same information. Therefore, for the first intersect, the information in the item buffer is used. The motivation behind this is that :

1. The average depth of the ray intersection tree is not much more than one, and

2. the computation expense of the Z-buffer algorithm is more than compensated for by the advantage gained by decreasing the time spent in finding the first intersection.

Van Wijk [12a] described an algorithm for finding intersection between a ray and objects defined by sweeping planar cubic splines. He considered three types of objects, they are generalized by :

- translational sweeping,

- conic sweeping, and
- rotational sweeping.

The primary function of the algorithm is to reduce the three-dimensional intersection problem into a two-dimensional problem. He proposed another method for the ray tracing objects defined by sweeping a sphere [12b]. This technique is applicable to the class of objects generated by sweeping a sphere of varying radius along a three-dimensional trajectory.

Glassner [13] presented an algorithm based on the octree technique to improve the efficiency of dealing with visible-surfaces in the ray-tracing processes. In this implementation, the three-dimensional environment is subdivided into a hierarchical tree of compartments. In other words, each node of the octree structure represents a smaller compartment which is called a "voxel". The nodes in the tree are created dynamically, i.e. a new node can be created when the researchers need it. The subdivision of the outermost bounding volume is performed recursively until each voxel contains less than a minimum number of objects as previously defined. Each ray passing through the viewing plane has to be tested against the outermost bounding volume or the root node. If a ray does pierce the root node, it has to be examined more closely. The ray-voxel intersection tests are reduced to simple comparisons against the limits of the boundary boxes. The ray has to be traced along the intersected voxel in the octree structure. It also has to be checked for intersection only with those objects which belong to the current voxel and not with the entire cluster of objects in the scene. If there is no valid intersection with any of the objects contained within this voxel, the ray is traced further into the next voxel and the process is repeated. Glassner used a combination of hash table and linked list to represent the relationship between nodes in the hierarchical octree. Therefore, finding the node involves generating a name through a numerical manipulation of the intersected point's coordinates (x, y, z). This method is better because only a portion of the whole-scene information is examined.

These developments show that : most of the research is concentrated on speeding up the ray tracing process by reducing the time required for finding the ray-surface intersections. In this case, some of the traditional difficulties have already been overcome, while breakthroughs are yet to come. Despite the impressive images and decreasing the magnitude of CPU time spent during the rendering process, there are still many improvements that have to be made. The next chapter discusses the details of implementing the modified ray-tracing algorithm on a parallel processing system.

CHAPTER V

AN OVERVIEW OF THE ALGORITHM

The ray tracing algorithm was introduced by Glassner[GLAS84], is fairly fast and can be applied to any defined three-dimensional surfaces. It can also be used to simulate illuminated environments with any number of light sources and generate shaded images with multiple shadows and light reflections effects on the final image.

The algorithm developed here is based on the algorithm proposed by Glassner, with some modifications wherever necessary to speed up the rendering process and the way a ray would propagate through the environment. The modified algorithm can minimize ray-cell intersection tests because its tree structure consists of fewer empty cells. The major differences in our implementation involve the way in which the data structures is stored. For example, Glassner manipulates a combination of hash table and linked list structures to find a particular node in the tree. On the other hand, the researchers use a modified octree structure to store data structure in the tree.

Normally, when the researchers generate graphical images, they are interested only in the surface of the objects in the scene. In this algorithm, the assumptions researchers made are :

- the inside of the transparent objects is empty,
- the light source is a white light, and
- the light source is represented by a point of light, thus, there are no penumbra effects.

The basic fundamental concept of space-tracing or octree technique is to simplify the way ray-tracing should be performed against a known space, rather than the arbitrary objects in the space environment. This method makes hidden surface elimination more efficient because each ray is examined in known space. As a matter of fact, a ray of light does not know about the objects in its path until it is struck. The ray travels in the three-dimensional space in attempts for intersection with the objects in the environment. Therefore, each light ray's path has to be examined against all of the objects in the scene. The process of finding possible ray-surface intersections require an enormous amount of CPU time. On the other hand, the space-tracing tests a piercing ray on a known space against all of the surfaces residing within this current voxel only, rather than the whole scene. When a ray enters an area of space known to contain objects, most likely a ray-surface test will be performed. In this process, the researchers could eliminate unnecessary ray-surface calculations.

An hierarchical octree structure is normally used in generating solid modeling pictures. This tree helps to construct the shapes of the objects that are difficult to model with primitive surfaces; in the content, each subcell of the tree is either occupied by the surface, or it is empty. Fortunately, a very good scheme for dividing an environment space according to the objects in the scene and building up an hierarchical octree structure is available. An octree structure can dynamically divide the scene into smaller cubes until each cube is either filled with an object or empty. This technique is described very extensively in articles presented by Doctor and Torborg [14], Meager [15], Tanimoto and Jackins [16], and Gargantini [17].

At the preprocessing stage, the octree technique encloses the whole-scene within a bounding volume which is referred to as the "outermost bounding volume". This outermost bounding volume is also called the root node of the hierarchy of tree. In this algorithm, the researchers use rectangular parallelepipeds as bounding volumes and its coordinates align with the world coordinate axis. This arrangement will simplify the way rays would travel from one compartment to another. After determining the limits of the outermost bounding volume, all its vertices are projected unto the viewing plane. From the projection points in the picture plane, a rectangular region is determined that for a given ray is likely to intersect the objects in the space environment. This region is called an "active-region".

After determining the active-region, the outermost bounding volume is subdivided into a hierarchical octree of voxels. Each node in the tree attaches a maximum of eight smaller compartments which are called subnodes. The researchers also keep the list of the objects residing within each of the compartments. The process of constructing an octree is dynamic. If any node does not satisfy the octree stopping conditions, then it must divide the node into eight subnodes recursively until all subnodes meet the stopping conditions. The modified octree structure is used to represent the relationship between the nodes in the tree. The octree stopping conditions are as follows :

1. It reaches empty cells.

2. It reaches the octree maximum level.

3. Each voxel contains less than maximum number of objects as previously defined.

After establishing the octree structure, conventional ray tracing techniques are applied. The image is processed per scanline and per pixel from left to right and from top to bottom and checked for each pixel's position on the picture plane. If the pixel's coordinate is outside the active-region, then the background intensity will be taken as the intensity of the pixel the ray passes through. Otherwise, the researchers find the intersection point between the ray and the most bounding volume and they examine the current bounding box more closely. The piercing ray in the cell must be tested against all the objects within the current cell under investigation. If the ray intersects the object inside the current cell, the ray-surface intersection point calculates. The intensity, reflected ray, and/or refracted ray at the current ray-surface point are calculated by using the attributes of the intersected object. If both reflections and refractions effects on the synthesized images are included, the reflected and/or refracted rays are traced further into the neighboring voxels. However, if there are no ray-surface intersections within the current voxel, the light ray's path has to trace further into a next neighboring cell. The above process is continued until the ray passes through the outermost bounding volume or its intensity satisfies the threshold stopping criterions. The above procedures are repeated for every pixel on the viewing plane.

In the space-tracing technique, one minute time spent in the preprocessing stage could be worth several hours of the rendering process because it performed a presorting the objects in the scene into a spatial (tree) structure. This tree configuration reduces the number of ray-surface intersection tests have to be performed. Therefore, it is reasonable to perform preprocessing step in order to reduce ray-environment intersection costs, providing that the information obtained during preprocessing can be applied to a majority of these intersection computations.

The algorithm produces an image intensity file. This file is made up of the values of red, green, and blue components of the colour intensity for every pixel in the picture plane. It also contains the coordinate of every pixel with the specified colour intensity. A display program reads the intensity of the image file and displays the computer-generated image on the screen of Sun workstation 3/110. These can be broken up into two steps, as follows :

1. Converting the image-file intensity into a "rasterfile" format which is available in Sun workstation.

2. Using Sun's system commands such as screenload and show to display the image file onto the screen.

These steps allow the ray-tracing program to run in the background mode with some other processes without interference from the standard I/O. This process will not slow down the rendering process. The modified ray-tracing algorithm is implemented in the "C" language on Sequent's "Balance 8000" computer system. At present, the Balance 8000 system is running with eight processors and no a floating point accelerator chip. A concise pseudo-code for generating computer images synthesis is presented in Figure 8.

```
Input the data;
Set up the preprocessing stage;
Set the number of processes used, N;
Divide the screen into M smaller packets;
FOR every packet distribute among the processors DO
BEGIN
  FOR every Process with a packet DO
  BEGIN
    FOR every ray (pixel) DO
    BEGIN
      IF (pixel is located inside the active region) THEN
      BEGIN
        Intensity = 0;
        Determine ray equation (generate a ray)
        Determine and check each encountered cell along the ray's path
        WHILE ray has not reached the outermost bounding volume DO
        BEGIN
          IF the current cell is not empty THEN
          BEGIN
            IF (the ray intersects objects) THEN
            BEGIN
              Determine the reflected ray
              FOR every light source DO
              BEGIN
                Procedure Intensity (Intensity) in Figure 18.
              END
              IF the object is transparent THEN
                new ray = determine the refracted ray
              IF the object is reflector THEN
                new ray = reflected ray
              Intensity = Intensity + global contribution
            END
            ELSE IF  no intersection THEN
              Intensity = Background Intensity
          END
```

Figure 8. The pseudo-code for generating computer synthesized images.
(Continued on next page.)

```
          Trace the ray's path further into the scene
            Determine and check neighboring cells along the ray's path
        END
      END
      ELSE THEN
        intensity = Background Intensity
      Write the intensity into two by two array, which represents the image
      plane
    END
  END
END
Write the array intensities into an output file.
```

Figure 8. The pseudo-code for generating computer synthesized images. (Continued from previous page.)

Producing an synthesized image could be considered as multi step processes. The major steps of the ray tracing algorithm are listed below.

1. Reads the input file.

2. Sets up the preprocessing stage.

3. Details the algorithm.

4. Implements the algorithm in parallel processing.

5. Calculates the image intensities.

6. Writes the output intensity of the colour elements (red, green and blue) into image file and converts it into a rasterfile format.

Detailed description of the above steps are described in the following subsections.

5.1 READING THE INPUT FILE

In this step, the researchers model the physical environment so it can be easily simulated through the ray-tracing technique. The system provides an user with the input file controls, such as it should be easy to specify the viewing (lighting) parameter and to process the object's description. The format of an input data file is described below.

- The coordinate of viewpoint;
- The size of image resolution (W × H);
- The color elements of the background intensity (red, green, and blue);
- The coordinates and brightness of the light sources;
- The characteristic properties of the objects in terms of ambient, diffuse, specular, shining, reflected, and refracted coefficient;
- The description of objects in terms coordinates and boundaries.

The user can interactively generate an image file according to his/her needs. It is very convenient to be able to change the input file easily. Files containing above input descriptions should be produced once and used for many different images. Figure 9 describes a typical input file which contains all the necessary data. It is presented as follows :

v $V_x V_y V_z$
resolution H W
b R G B
l $L_x L_y L_z$ brightness
f ambient(R,G,B) diffuse(R,G,B) specular(R,G,B) shining reflect refract
s Center(x,y,z) Radius
c Base : Center(x,y,z) Radius Top : Center(x,y,z) Radius
box Center(x,y,z) Side(S_x,S_y,S_z)
p n (number of vertices)
$V_0(X_0,Y_0,Z_0)$
$V_1(X_1,Y_1,Z_2)$
.
$V_n(X_n,Y_n,Z_n)$
where :
v = viewpoint
resolution = image resolution
b = background color intensity
l = light source
f = the object's attributions
s = sphere
c = cone or cylinder
box = box
p = polygon

Figure 9. A typical input data file.

The resolution size determines the size of the rendering images file which is made up of the width (W) and height (H) of the viewing plane (note, a default picture's size is equal to 512 by 512). The multiplication of the W and H is equal to the total number of rays that have to be traced. Therefore, the larger the image's resolution the more rays must be traced; it requires more time to produce a computer-generated image.

Since the rendering process is functions of the image resolution, the number of objects in the scene, and the degree complexities of the image, any changing of these variables will effect the processing time. For example, a smaller image file will require lesser time or a higher degree images complexity will require longer processing time to generate a synthesized image.

5.2 SETTING UP THE PREPROCESSING STAGE

The main priority of this stage is to cut down the computation costs during the rendering process. This stage can be broken into two distinct stages.

5.2.1 Stage I

At first, the researchers determined the scene's outermost bounding volume, which is called the octree root node. As its vertices were projected into the picture plane, a rectangular region on the image plane was determined. This region is called an "active-region" because a given ray that passes through this region has a high potential to intersect the objects in the scene; therefore, the ray must be inspected more closely. However, if a given ray passes a pixel located outside an active region, the ray should be discarded from further consideration or it should set the pixel intensity equal to the background intensity. This operation will speed up the first-level rays through the viewing plane. Figure 10, describes procedures of determining the active region on the image plane.

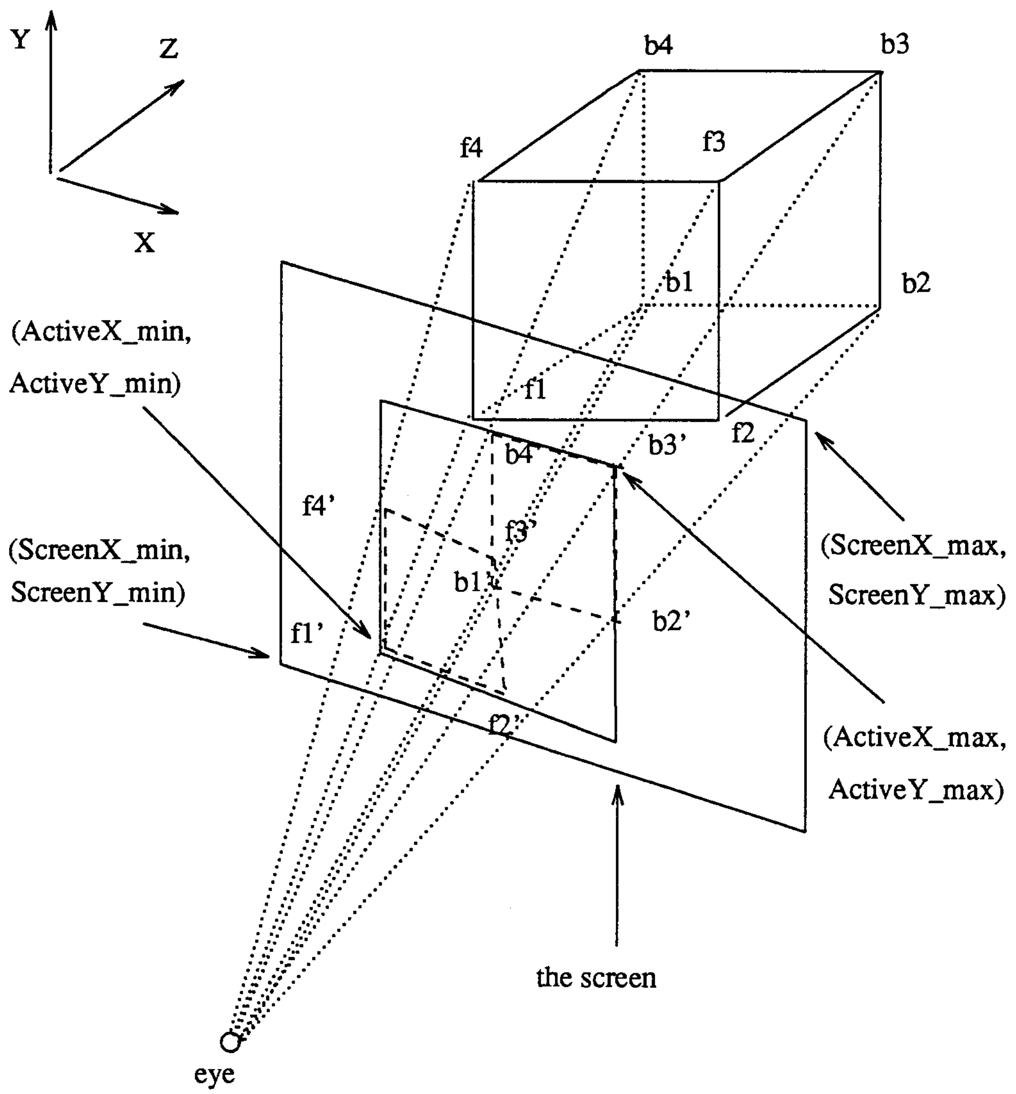

Y
Z
X
b4
b3
f4
f3
b1
b2
f1
f2
b4'
b3'
f4'
f3'
b1'
b2'
f1'
f2'
(ActiveX_min, ActiveY_min)
(ScreenX_min, ScreenY_min)
(ScreenX_max, ScreenY_max)
(ActiveX_max, ActiveY_max)
the screen
eye

Figure 10. Determining an active-region on the image plane.

Consider Figure 10, as the vertices of the outermost bounding volume ($f_1, f_2, f_3, f_4, b_1, b_2, b_3, b_4$) are projected into the image plane, the projecting points ($f'_1, f'_2, f'_3, f'_4, b'_1, b'_2, b'_3, b'_4$) are obtained. From these projecting points, the researchers sort and find the active region boundary in terms of ActiveX_min, ActiveX_max, ActiveY_min, and ActiveY_max as depicted in Figure 10. For a given ray that passes through the active region is likely to intersect the objects in the scene. In other words, the physical size of the active region will play an important role in determining the rendering process.

5.2.2 Stage II

The primary goals in this stage are to build a data base system that will allow the arbitrary ray-environment intersections to be calculated as quickly as possible. This data base is known as a hierarchical octree structure; it contains the information necessary to speed up the rendering operations.

The data base adaptively divides the environment space into a hierarchical structure of smaller compartments, with its axes aligned with the cartesian axes of the world coordinate system. The coordinate axes arrangement will simplify the way an arbitrary light ray would travel from one compartment to another. The subdivisions are based on the objects in the scene, which means that a subspace with high objects complexity can be recursively subdivided into smaller and smaller subspaces until it reaches the octree stopping criteria. As a space is divided into eight octants, an octant number is assigned to each node, as shown in Figure 11a.

The researchers also determine which cells the objects in the scene belong to. This operation can be implemented by keeping a list of all objects whose surfaces intersect or reside within a cell. Each compartment (voxel) must contain sufficient information to fully describe the node description. This information includes the voxel's boundary and ID number, object's normal vector (if it is possible), objects' attributions, the ray's origin, the ray's direction, and reflected ray. A voxel (cell) can be represented as a node in the tree. The octree consists of leaf nodes and non-leaf nodes, each of them described as follows :

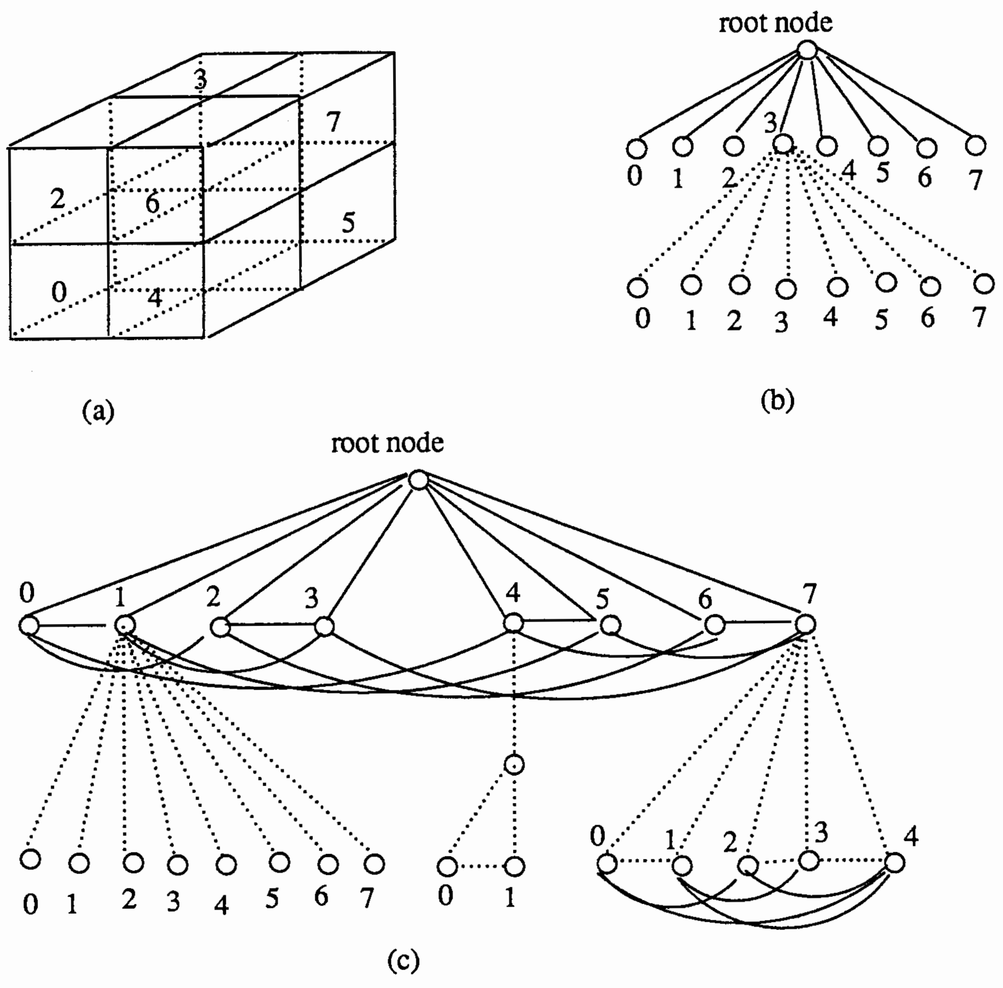


Figure 11. The hierarchy of an octree.

1. The leaf node : it consists of an object or no object.

2. The non-leaf node : it is also called the branch node which contains leaf nodes and/or branch nodes.

The hierarchy of an octree is organized in such a way that a node in the tree has a maximum of eight child nodes directly attached to it. A pseudo-code for constructing a modified hierarchical tree depicted in Figure 11c, is presented in Figure 12.

```
PROCEDURE Octree (node); {node is a branch in the tree}
BEGIN
  IF objects > maximum object in the cell  THEN
  BEGIN
      Set_flag_type
       Create_subcells (node, type);
       Increment tree level;
       Create_octree (node);
  END
  ELSE IF objects = 2  THEN
  BEGIN
        IF two objects are overlap  THEN
          Set_flag_type
       ELSE THEN
          Sorting (node, type);
   END
   ELSE IF objects > 2  THEN
       Sorting (node, type);
   ELSE IF no object  THEN
      error no object in the scene
END
```

(a)

```
PROCEDURE Sorting (node, type)
BEGIN
         Set_flag_type
         Create_subcells (node, type);
         Determine_mid_point (node, type);
         Sort_theobjects (node, type);
         Links_number_subcells (node, type);
         Determine_cell_bounding_volume (node, type);
END
```

(b)

```
PROCEDURE Create_octree (node) {node is a branch in the tree}
BEGIN
  IF tree level <= maximum tree level THEN
  BEGIN
    Determine_mid_point (node, 8); ;Determine the mid axes of the cell
    Sort_theobjects (node);  ;Determine which cell the objects belong to
    FOR cell number < 8  DO
    BEGIN
      Links_number_subcells (node, type); ;Link and number the cell
      Determine_cell_bounding_volume (node, type);
```

Figure 12. The pseudo-code for constructing the hierarchy of a modified octree.
(Continued on next page.)

```
    IF object > maximum object in the cell THEN
    BEGIN
     Set_flag_type              ;type of cell
     IF cluster (node) THEN    ;check if the objects can be clustered
        Sorting (node, type);  ;into a cell which has volume less
                               ;than half of the current cell
     IF tree level = maximum tree level THEN
        Sorting (node, type);
      ELSE THEN
      BEGIN
      Create_subcells (node, type);
      Increment tree level
      Create_octree (node)
     END
    END
    ELSE IF objects = 2 THEN
     BEGIN
     IF cluster (node) THEN
        Sorting (node, type);
      IF two objects are overlap THEN
       Set_flag_type
     ELSE THEN
        Sorting (node, type);
     END
    ELSE IF objects > 2  THEN
     BEGIN
      Set_flag_type
     IF cluster (node) THEN
        Sorting (node, type);
      IF tree level = maximum tree level THEN
        Sorting (node, type);
      ELSE THEN
      BEGIN
      Create_subcells (node, type);
      Increment tree level
      Create_octree (node)
      END
    END
     IF cell number = 7 THEN     ;If a cell contains only one object it
       combine_subcells (node);  ;can be merged with three other
   END                          ;empty cells into a bigger cell.
  END
END
```

(c)

Figure 12. The pseudo-code for constructing the hierarchy of a modified octree. (Continued from previous page.)

As a matter of fact, the hierarchy of an octree has a uniform tree structure as shown in Figure 11b. This structure ensures that a ray has a predictable path and manner. It also allows faster access to nodes in the tree, unless the propagation overhead of rays from one compartment to another is very high. The advantages of the uniformity is the ability to examine the subspaces in an arbitrary order and to access nodes in the tree more efficiently. However, the drawback of dividing an object space into equidimensional cells is that a high number of empty cells are generated. Therefore, when the hierarchy of an octree structure becomes very deep, the propagation of a ray from one cell to another becomes very slow because it performs the ray-cell intersection tests excessively. The overhead associated with this process is very high due to ascending and descending of the tree which requires traversing between subspaces. Thus, every addition of a level to the tree may cause a linear increase in the access time.

Hence, the researchers need a technique that improves the octree algorithm. A modified octree structure with fewer empty spaces would speed up the propagation of a ray from one cell to another. Therefore, the empty cells should be eliminated and the uniformity of the tree should be maintained. This is done by merging three empty cells with a cell whose volume is either full or empty in to a bigger cell. The modified algorithm has a semi-uniform tree structure. The modified tree keeps the geometrical relationship of each subspace, eliminates the undesired empty nodes, and connects the face of neighboring subspaces (cells) at the same level directly to one another. It also uses a cluster bounding volume within the cell as depicted in Figure 11c. This modified structure ensures that a ray can quickly travel through the space environment as it encounters cells along its propagation path. In this case, it encounters fewer cells. The modified tree also allocates fewer memory spaces than the octree structure. The octree and modified octree structure can be illustrated in Figure 13 and Figure 14.

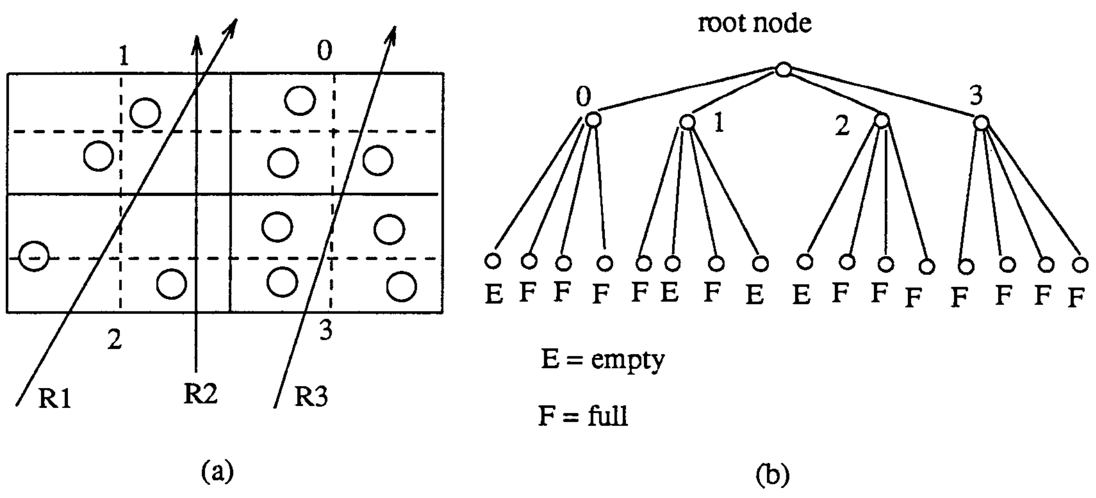


Figure 13. A 2-D environment is divided and constructed by using the octree method.

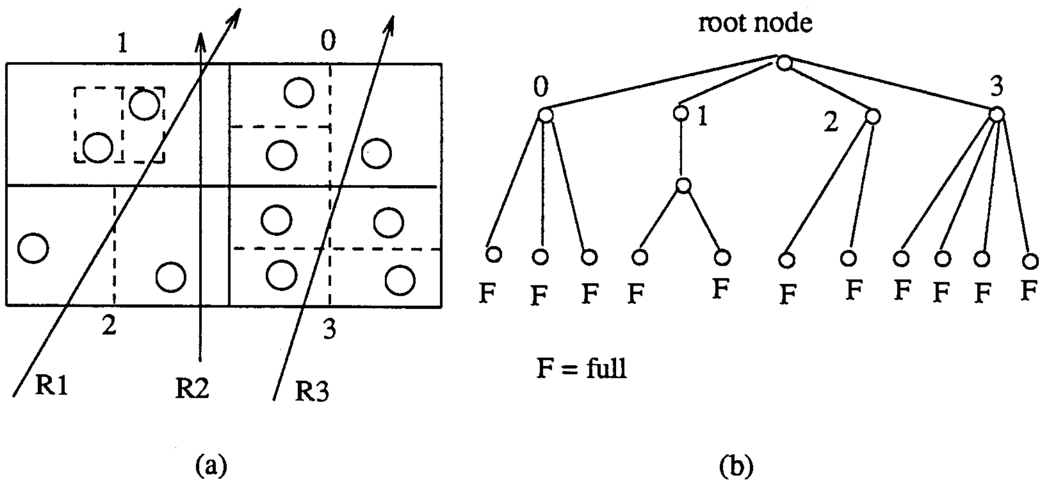


Figure 14. A 2-D space is subdivided and built by using the modified octree algorithm.

With reference to Figure 13b and Figure 14b, these observations show that the modified octree structure has fewer tree nodes than the octree structure. In this case, the modified octree and octree structure have 16 and 21 nodes respectively. Consider Figure 13a and Figure 14a, the performance of ray-voxel and ray-surface intersection tests for both the modified octree and octree algorithms are tabulated in Table I.

TABLE I

THE NUMBER OF RAY-VOXEL AND RAY-SURFACE INTERSECTION TESTS WHICH MUST BE PERFORMED

Ray	Number of ray-environment intersection tests			
	octree		modified_octree	
	Voxels	Surfaces	Voxels	Surfaces
R1	6	3	6	3
R2	5	2	3	1
R3	6	4	5	4

The results in Table I indicates that the ray propagates easily through a hierarchy of a modified octree than through the octree structure. In other words, the modified algorithm performs fewer ray-surface intersection tests. Therefore, the modified octree algorithm is better and more efficient than the octree method. In general, a ray-cell intersection test may need at most three infinite planes to be examined. These planes are part of the bounding cell. Therefore, performing a ray-cell test in a three-dimensional space is computationally expensive. However, if the propagation of a ray through the space compartment were substantially slower than the ray-surface intersection test, then this technique does not significantly improve upon the conventional method.

5.3 DETAILS OF THE ALGORITHM

Once the preprocessing stage has been completed, the hierarchical octree structure can be applied to quickly determine the ray-surface and/or ray-environment intersections. The intersection points found in this manner will be exactly the same as if they have been computed using the conventional ray-tracing technique. Using this structure, the researchers only examine the scene along the ray's path which represents a small portion of the environment. Now, a traditional ray-tracing algorithm can be applied to generate a graphical image. The rays are generated, in order, from left to right, top to bottom by two nested loops. The two nested loops are described as follows :

1. The outer loop is from 0 to H, where H is the number of vertical pixels of the image plane.

2. The inner loop is from 0 to W, where W is the number of horizontal pixels of the image plane.

As researchers go through the picture plane per scanline and per pixel, they check the pixel's position. If its coordinate is outside the active region, the ray casting step is completed. This means that the pixel intensity is equal to the background intensity and that researchers start generating another new ray. However, if the pixel's position is inside the active region, its ray has to be examined further into the scene. The above procedures are repeated until all of the image pixels are calculated. The above procedure can be slightly improved by scanning the active region per line and per pixel rather than the entire image plane. In this case, it is not necessary to examine the pixel's position whether it is outside or inside the region. However, before applying the image scan procedure, researchers initialize pixels on the image plane with the background intensity. This initialization can be parallelized in multiprocessor system. The process of ray-environment intersection is performed as follows :

1. Determines the ray's origin on the root node.

2. As the researchers traverse the hierarchy of an octree, they determine which cell has to be searched so it searches through the appropriate link of the nodes in the tree. This is done by comparing the (x, y, z) values of the ray's origin with each of the subnodes' boundarier or the decision of selecting a subcell depends on the relationship between the coordinate of the ray's origin and the boundary of the current cell. In this case, only the appropriate nodes encountered along the ray's path are searched. Eventually, a leaf node will be reached.

3. Check node's flag type described as follows :

• If the cluster's flag is true, it performs ray-cluster intersection test. If there is an intersection between a ray and the cluster within the current cell, it goes to step 2. Otherwise, it goes to step 6.

• If the further flag is true (the current node has more than one subnodes), it goes to step 2.

• If no object is true, it goes to step 6.

• If the object flag is true or if tree level is equal to maximum plus one, it goes to step 4.

4. Find possible intersection between a ray and objects within the current node. For each intersection of objects, find the intersection point between the ray and intersected surface. make sure that the point is within that voxel. if a ray does not intersect objects in the current cell, it goes to step 6.

5. If step 4 has a multiple intersections, a list of intersected items is generated and sorted in order of increasing t-values. The next step is to determine the closest intersection point from the ray's origin. This point is a valid intersection point of that ray. When the intersection point is found, the secondary ray is calculated by using the intersected surface's attributions and the shading algorithm is applied to calculate the intensity at that point. The secondary ray is traced further into the scene until it reaches the stopping criteria.

6. Determine the next neighboring cell and find the intersection point between the ray where it leaves the current cell and the cell's boundary. The intersection point becomes the ray's origin of the next voxel. If the ray has reached the outermost bounding volume, it is terminated. Otherwise, researchers proceed with step 1.

If each compartment contains a small number of objects, researchers can process that compartment quickly. If they are lucky, only a small number of objects in the current voxel must be checked. However, if the researchers are unlucky, the ray may hit nothing but the outermost bounding volume. Researchers would still be better off because of fewer ray-objects tests performed.

5.4 IMPLEMENTING THE ALGORITHM IN PARALLEL PROCESSING

5.4.1 Balance Architecture

In computer generated-images, the processes of calculating the pixels' intensity are independent from one another and they can be simulated in the same manner. Therefore, the image's intensity can easily be calculated in parallel processing. The recent development in VLSI technology makes it possible to achieve large-scale multiprocessor system. The main issue in parallel processing is to keep the workloads balanced among processors in the system and to minimize the intercommunication between processors and the common memory via the common bus or between one processor and another. This is one of the ways to utilize the power of multiprocessor system. If one of the processors is heavily loaded, it affects the overall system performance drastically because the busiest processors will dominate the total amount of the processing time. For example, if the workloads are not evenly distributed among processors, some of the busiest processors are still running while other are idle. In this case, only a fraction of the power of a multiprocessor system is used.

The researchers used the Balance 8000 to implement the ray-tracing algorithm, because it meets our criteria to speed up the rendering process. In the Balance multiprocessing architecture, a number of independent processors are connected to the high-speed common bus as shown in Figure 15.

The above arrangement is called a tightly coupled architecture because there is a global bus or a global shared memory. This multiprocessor may not contain a large number of processors. The data partitioning technique is suitable to ray-tracing algorithm

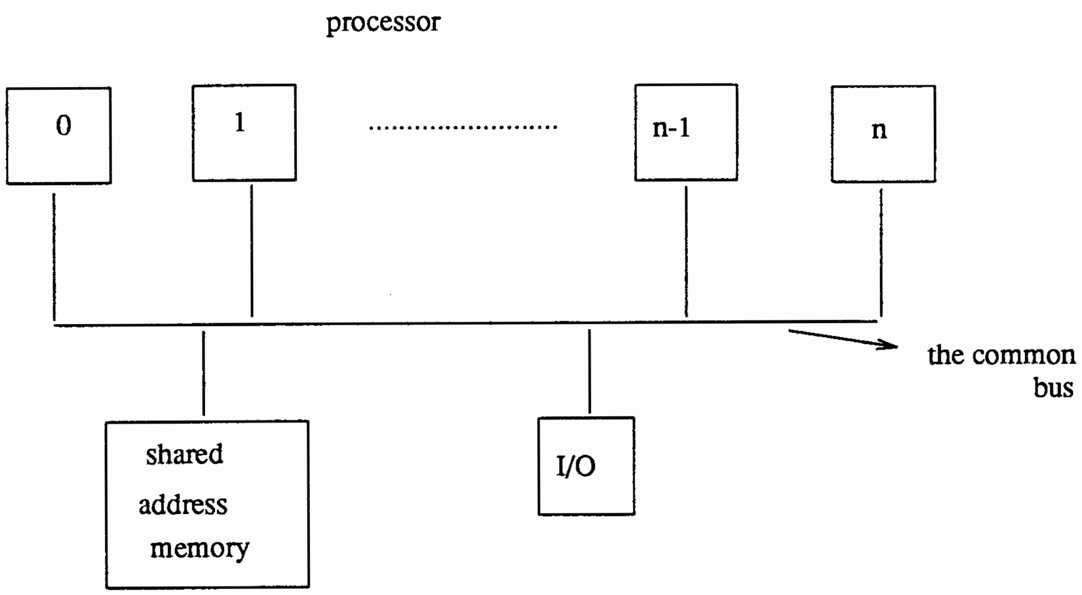


Figure 15. The Balance 8000 architecture.

because it performs the same operation repeatedly on the large collection of data. The advantages of this method are :

- Easily balance the workload among processors, and
- Adapt programs automatically to the number of processors in the system.

In parallel processing, there are three types of methods for scheduling tasks among processors. They are described as follows :

1. The prescheduling : before the program is compiled, the user determines the workload division.

2. The static scheduling : the processes schedule the workloads at run time, but they are divided in some predetermined way.

3. The dynamic scheduling : the process schedules its own tasks at run time by checking a task queue array index.

In this algorithm, the researchers implemented both the static scheduling and dynamic scheduling methods. Dynamic scheduling method keeps all the processes running as long as there is task to be done. It always distributes tasks to idle processors and balances the workloads among processors. Although it is more effective than static scheduling to a certain degree, this method at run time is quite expensive and tends to increase overhead. The overhead is due to checking the shared queuing array index : before it schedules another task, it always checks the shared memory queuing array index to find if there is an idle processor. Dynamic scheduling is more appropriate for similar computation in which each workload takes much different amount of time to be executed. However, if each workload takes approximately the same amount of the processing time, then the static scheduling is cheaper and more effective than dynamic scheduling. The processes automatically balance the system because each processor runs at the same rate.

Since a graphical image can be generated in parallel processing, the two-dimensional active-region on the image plane can be sliced into smaller rectangular regions of pixels. These smaller regions are distributed among processors, as depicted in Figure 16. In this case, there is intercommunication only between the processors and the global memory (no intercommunication between processors).

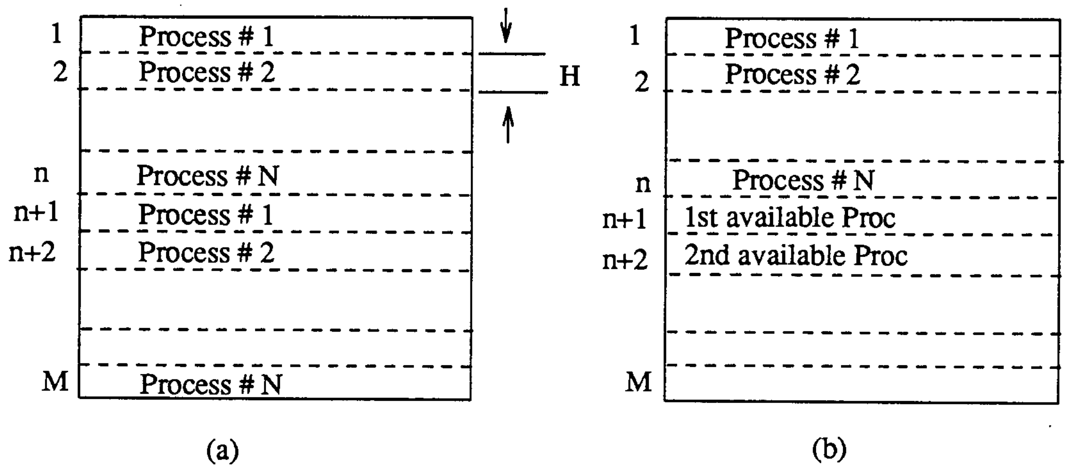


Figure 16. Geometrical slicing of the active region on the image plane. The packets distribute in (a) static or (b) dynamic scheduling.

This rectangular region is called packet. The parallelism used on the screen enables the calculation of the global intensity of each pixel. The intensity of pixels are calculated by the processors simultaneously. During the ray-tracing operation, the researchers distribute the packets among the processors running in the system. Each processor duplicates necessary information for speeding up the computation into the local memory. In this case, each processor has its own parts on the screen and merely creates the distributed portions of the image. By adjusting the H parameter of the packet, the researchers can balance the workloads among the processors. The H parameter controls the time required for each packet to accomplish its task. For example, as the value of H parameter increases, the system becomes unbalanced because each packet has a different processing time. For example, some processors may have a shorter processing time than others which dominated the overall rendering time. Before the researchers execute the loop subprogram into a parallel processing, they have to determine how many parallel processors will be used.

Consider Figure 16, the implementation of scheduling tasks on parallel processing is divided into two parts as described below.

• Static scheduling : The portions of an image are distributed to a fix processor. For example, packets 1, (n+1), etc. are assigned to processor number 1 and packets number 2, (n+2), etc. are distributed to the processor number 2, etc.

• Dynamic scheduling : The portions of an image are distributed to available processors. In other word, the first "n" packets are assigned to "n" available processors and the next packet will be assigned to the next available processor. For example, if processor number 2 finishes its task earlier than other processors, then packet number (n+1) will be assigned to processor number 2.

In parallel processing systems, the maximum attainable speedup is equal to the number of the processors are running. However, in practice, this full speed rarely can be achieved because of many factors that effecting the processing system. They are :

• The bottleneck : after it has reached a certain number of processors used, the speed of performance begins to decline. This is associated with the intercommunication bottleneck between processors and the shared memory. In this case, it is impractical to build large systems of this configuration because only so many processors can share a common bus before the bus becomes saturated.

• The unbalance processing environment : the workloads do not evenly distribute among the processors.

5.5 CALCULATING THE IMAGE INTENSITY

In this step, the intensity contribution of a particular light source is determined. Three conditions have to be satisfied before the intensity contribution can be calculated. These conditions are :

1. The diffusion contribution is determined by the dot product of two unit vectors. They are :

- The intersected surface normal vector, and
- The line of a light source (This line joins the current intersected point and the light source under consideration).

The value of a dot product has to be positive.

2. No shadow effect. It means there are no other surfaces that block the current intersected point from the light source.

3. The specular highlight contribution is computed by the dot product of two unit vectors as follows :

- The reflected ray's vector, and

- The line of a light source.

Its value has to be greater than zero. Testing the first condition in Figure 17 is explained in the following discussion :

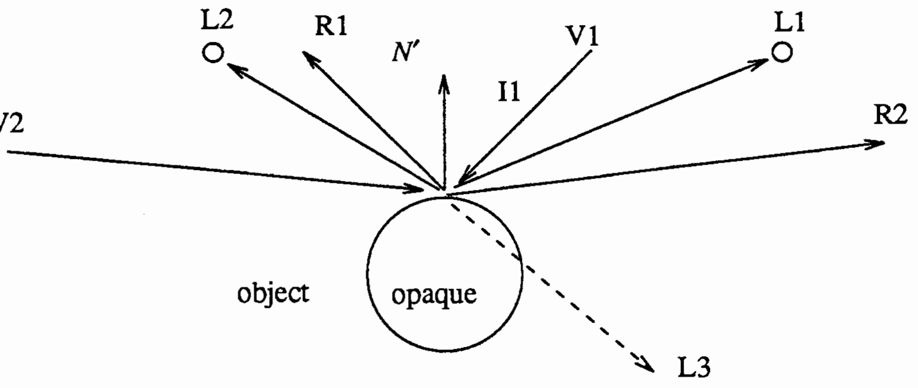


Figure 17. Determining diffuse, shadow and specular effects.

consider Figure 17, let V_1 and V_2 be the viewpoints, L_1,L_2, and L_3 be the light sources and $\vec{R}_1$ and $\vec{R}_2$ be the reflected ray's vector from the V_1 and V_2 , respectively. Let us also consider P_c is the current intersected point on the opaque surface S whose surface unit normal vector is $\vec{N}$.

In reference to Figure 17, it is obvious that light source L_3 does not contribute to the intensity at P_c because it is located behind the object itself. The researchers can observe that the vectors $\vec{PL}_1$ and $\vec{PL}_2$ form acute angles with respect to the normal vector $\vec{N}$, whereas the vector $\vec{PL}_3$ forms an obtuse angle. Therefore, the dot product of two unit vectors described as follows :

- $\vec{N} \cdot \vec{PL}_1$ is equal positive,
- $\vec{N} \cdot \vec{PL}_2$ is equal positive, and
- $\vec{N} \cdot \vec{PL}_3$ is equal negative.

From the above discussion, researchers can conclude that light sources L_1 and L_2 contribute diffuse intensity to the current point P_c. On the other hand, the light source L_3

contributes zero intensity to the point P_c.

Thus, the test involves the construction of a dot product of the line of light source and the current surface normal. The researchers check the sign of the result of the dot product. The following description details the manner in which the second condition is checked. Let the current point be $P_c(X_c,Y_c,Z_c)$ and the current location of light source $L_c(L_x,L_y,L_z)$. The equation of the line between these two points can be written as :

$$X_l = X_c + (L_x - X_c) \times t \tag{5.1a}$$
$$Y_l = Y_c + (L_y - Y_c) \times t \tag{5.1b}$$
$$Z_l = Z_c + (L_z - Z_c) \times t \tag{5.1c}$$

The above equations represent the line segment between the current point and the light source under evaluation.

Every surface other than the current surface is checked for intersection with this segment of equation(5.1). If the shadow ray's path (line of light) intersects a surface such that the parametric t lies between 0 and 1, there is zero intensity contribution due to the particular light. The shadow ray is traced from the current intersected point on the surface in the direction of light sources, in order to determine which surface could not be seen from the light sources. In this shadow test, the researchers need to identify only the first ray-surface intersection.

The evaluation of the third condition is discussed as follows :

From the observation of the Figure 17, the vectors $\overrightarrow{PL}_1$ and $\overrightarrow{PL}_2$ form acute and obtuse angles respectively with respect to $\overrightarrow{R}_1$. Thus, the dot product of two unit vectors is described below :

- $\overrightarrow{R}_1 \cdot \overrightarrow{PL}_1$ is equal negative,
- $\overrightarrow{R}_1 \cdot \overrightarrow{PL}_2$ is equal positive.

On the other hand, the vectors $\overrightarrow{PL}_1$ and $\overrightarrow{PL}_2$ form obtuse and acute angles with respect to $\vec{R_2}$. The dot product of two unit vector becomes as follows :

- $\vec{R_2} \cdot \vec{PL_1}$ is equal positive,
- $\vec{R_2} \cdot \vec{PL_2}$ is equal negative.

If the value of a dot product is positive, there is specular contribution toward the current point intensity P_c. Otherwise, there is zero specular contribution to point P_c.

Before the actual calculation of the intensity contribution is performed using the Whitted's illuminated model, which is described in equation (3.1) and (3.2), it is necessary to check all three conditions. It is described in Figure 18.

Lets consider the dot product of the diffuse and the specular highlight contribution be a "diffuse" and a "specular", respectively. The researchers also use an "Icp" to represent the intensity at the current point.

```
Procedure Intensity (Icp)
BEGIN
IF (diffuse is positive) THEN
 BEGIN
   IF (no shadow) THEN
     BEGIN
       set Icp = diffuse_intensity
       IF (specular is greater than zero) THEN
         BEGIN
           calculate the specular intensity by using the equation (3.2)
           add Icp = Icp + specular highlight intensity
         END
     END
  END
END
```

Figure 18. The subroutine for calculating the Whitted's model for a light source.

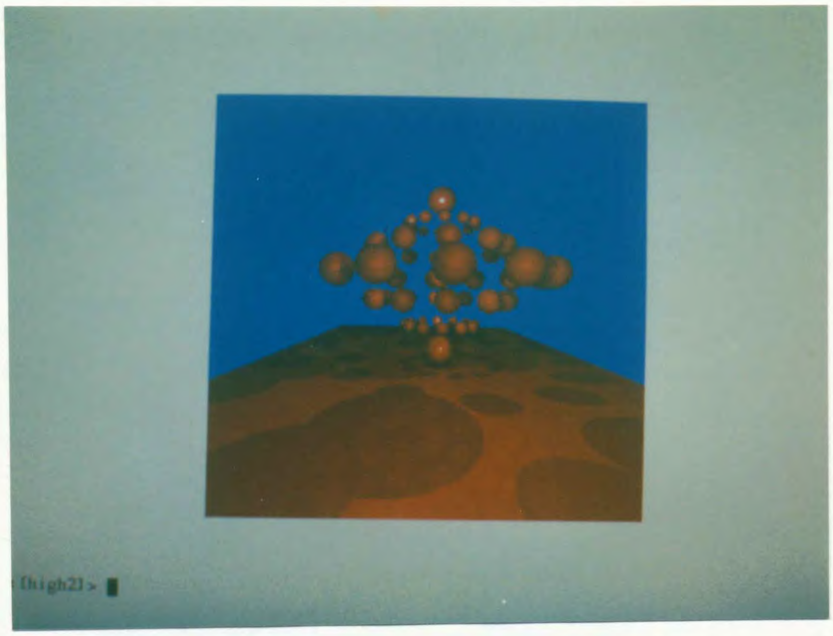
5.6 WRITING THE OUTPUT

After the intensity contribution of all the light sources is added to the final color intensity, it is stored in the image output file. In computer graphics display, each pixel corresponds to a spot on the image. The value of a pixel's intensity is made up of the values of red, green, and blue components of the color. Each of the color components has a value between 0 and 255 depending on the attributes of the intersected surface. The above process is repeated for every pixel in the image plane. Once the researchers have generated the image file, they change the format of the image file to a rasterfile format by converting each pixel's intensity into an integer number between 0 and 255. Its value depends on the combination values of red, green, and blue components of color. The "rasterfile" file is used as input data for the Sun's system commands such as screenload and show to display the shaded image on the screen.

CHAPTER VI

RESULTS

This chapter presents some of the examples of the shaded images that are generated using the modified algorithm previously discussed. These computer synthesized-images are shown below.



```
[high2]>
```

Figure 19. An image with 62 spheres, 1 polygon, 2 lights and 4 sample pixels per ray.

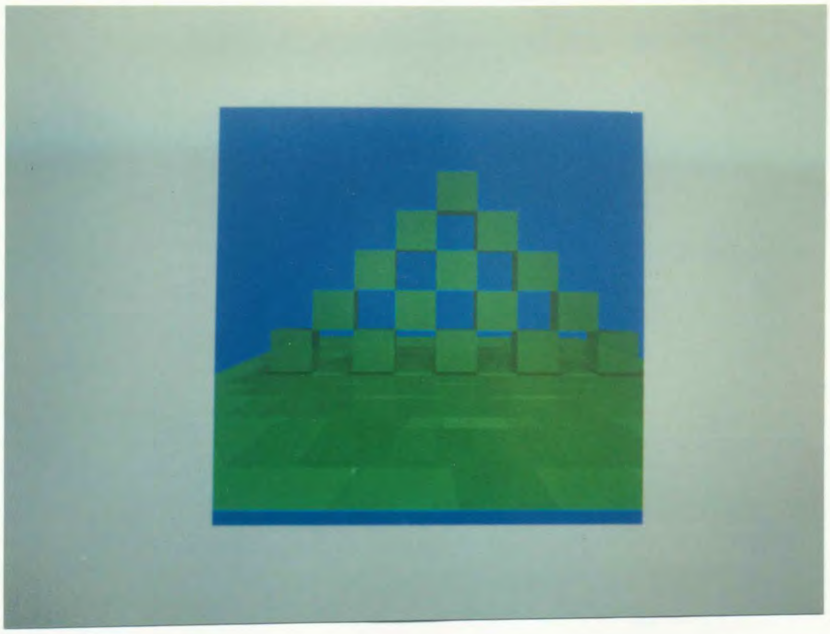


Figure 20. An image with 15 boxes, 1 polygon, 10 lights and 1 sample pixel per ray.

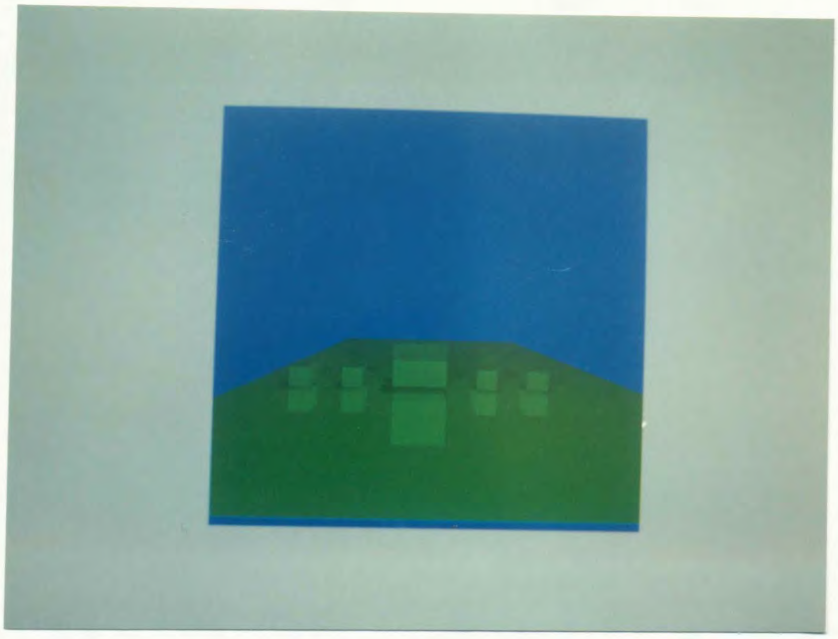


Figure 21. An image with 5 boxes, 1 polygon, 3 lights and 4 sample pixels per ray.

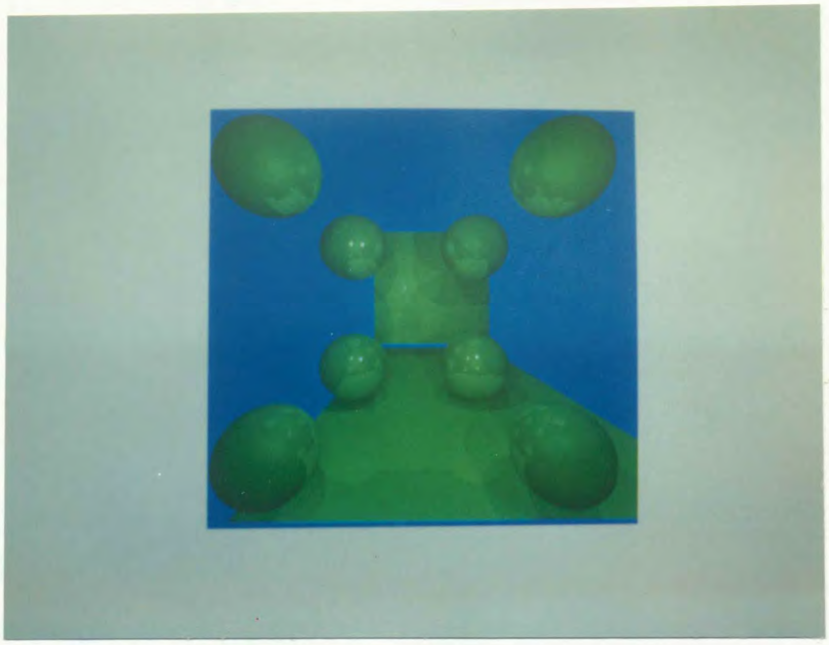


Figure 22. An image with 8 spheres, 2 polygons, 6 lights and 1 sample pixel per ray.

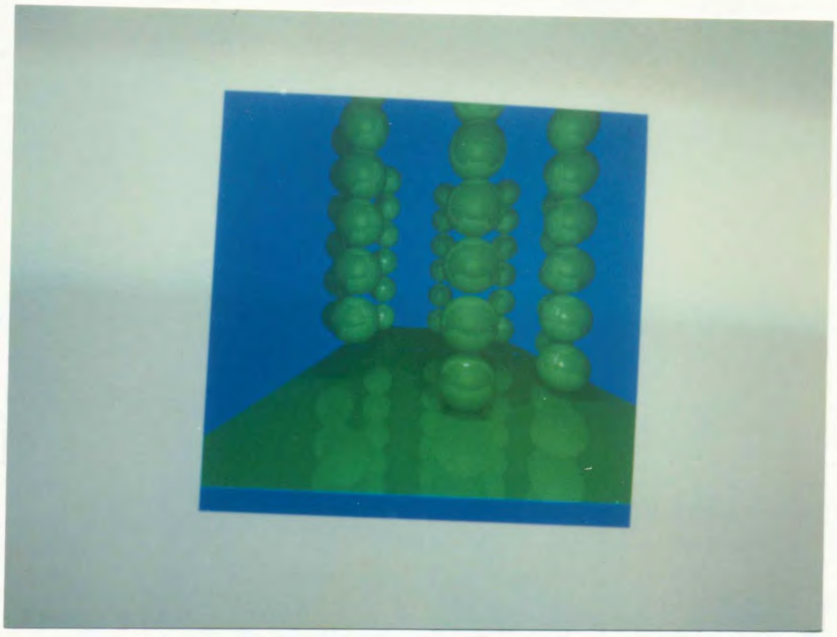


Figure 23. An image with 48 spheres, 1 polygon, 6 lights and 1 sample pixel per ray.

Evidence has demonstrated that the performance of the space-tracing algorithm has outperformed the conventional ray-tracing technique in cutting down the computation time. Results have also shown that the algorithm is extremely fast and effective in achieving the goal of speeding up the rendering process. Obviously, presorting objects into a spatial structure is required so that it can easily and directly be accessed by the path of a propagating ray. Rays are intersected with only a small number of objects regardless of the environmental complexities. The presorting steps are necessary to achieve before applying the conventional ray-tracing because these results are performed only once during the preprocessing stage and are eliminated in some calculations of the ray per loop. The sorting performed during preprocessing also ensures that the closest intersection point along the path of the ray is found first, except for the special case when the ray has multiple intersections with objects in the same spatial box. For this special case, some sorting must still be performed.

The remainder of this chapter presents some of the results of the images synthesis that are generated on Sun workstation and Sequent's Balance 8000. The time required for producing the computer generated-images is measured in minutes approximately and tabulated on the tables described below.

TABLE II

TIMINGS OF IMAGES GENERATION WITHOUT A FLOATING POINT CHIP FOR BOTH SUN WORKSTATION AND BALANCE 8000

Figure	Surfaces	Lights	Shadows	Reflections	Processing time	
					Balance	Sun
2	16	10	190874	135368	135.11	188.86
2	16	8	170282	135368	105.69	140.45

Table II describes the timing of image generation produced on a Sun workstation 3/110 and Sequent's Balance 8000 running with a single processor and without a floating point chip. This table shows that image generation on Balance 8000 is slightly faster than the Sun workstation.

The rendering time of images on a Sun workstation with and without a floating point chip are listed in Table III. From this table, it is obvious that the ray tracing algorithm requires enormous floating point computations. Executing the program with a floating point chip (fpc) will cut down the processing time by a factor of two.

TABLE III

PROCESSING TIME OF IMAGES GENERATED WITH AND WITHOUT A FLOATING POINT CHIP

Figure	Surfaces	Lights	Shadows	Reflections	Processing time	
					fpc	no_fpc
1	63	10	243792	128894	73.00	182.43
2	16	3	78754	0	21.05	54.29

The following tables describe the timings of the images with a variety of rendering process parameters. They are : degree of complexities (the number of light sources, the number of reflected rays, etc.), the size of an image resolution, and the number of surfaces.

TABLE IV

THE GENERATING TIME FOR IMAGES WITH DIFFERENT DEGREES OF COMPLEXITY

Figure	Surfaces	Lights	Shadows	Reflections	Processing time
4	10	6	91003	177220	54.61(reflective)
4	10	6	59768	0	26.99(opaque)

The time of rendering process can be seen to rise as the number of light sources, number of the objects, and the size of image resolution increase. However, the space-tracing is very suitable for complex images, thus, its rendering process slightly increases

TABLE V

THE GENERATING TIME FOR IMAGES WITH VARIOUS NUMBERS OF SURFACES (WITH FPC)

Figure	Surfaces	Lights	Shadows	Reflections	Processing time
1	63	2	24412	128894	24.65
2	16	2	81568	135368	24.95
3	6	3	12480	115154	16.63
4	10	2	28162	177220	20.56
5	49	2	48360	206583	48.68

TABLE VI

THE GENERATING TIME FOR IMAGES WITH DIFFERENT IMAGE RESOLUTION

Figure	Lights	Shadows	Reflections	resolution	Processing time
5	2	48360	206583	512x512	48.68
5	2	48476	226075	640x640	52.55

from one image to another. The factors which control the time taken for processing are :

- the number of surfaces,
- the number of light sources,
- the degree of complexity of the image, and
- the size of the image resolution.

As the number of surfaces increase, the ray has to be checked for possible intersection with more objects, hence the processing time increases. Since space-tracing works well with large numbers of objects, the rendering process will increase slightly.

As a matter of fact, the researchers could speed up the rendering process by a factor of n, if they sample the image with "n" number of pixels per ray instead of one pixel per ray. This speed improvement can be seen in Table VII. However, the image quality as produced by this technique is poor, when compared with the other method (i.e one pixel per ray). The image generated by this method consists of distortion effects, such as jaggies at the edge of surfaces. It is obvious that the image quality becomes worse as the number of pixels per sample increase.

TABLE VII

THE GENERATING TIME FOR IMAGES WITH DIFFERENT NUMBERS OF SAMPLING PIXELS PER RAY

Figure	Lights	Shadows	Reflections	pixels/ray	Processing time
4	6	91003	177220	1	54.61
4	6	22616	44238	4	13.55

The load of the system could be controlled by adjusting the H parameter of packets. The results of Figure 2 and Figure 4 are tabulated in Table VIII.

TABLE VIII

THE GENERATING TIME FOR IMAGES PRODUCED IN STATIC AND DYNAMIC SCHEDULING WITH DIFFERENT VALUES OF H (4 PIXELS PER RAY)

H	Processing time			
	Figure2		Figure4	
	static	dynamic	static	dynamic
2	3.90	3.81	3.85	3.70
4	3.95	3.83	3.88	3.76
8	4.02	3.87	4.04	3.79
16	4.20	3.90	4.07	3.84
32	4.40	4.03	4.33	4.17
64	6.67	6.50	5.77	5.72
128	10.25	10.03	9.32	9.17
256	14.45	14.45	15.40	15.40

The workloads are evenly distributed among processors when the H-value is very small because there is a small delay time between one task (packet) and another. In other words, each packet requires approximately the same amount of processing time. The performance of balancing system as tabulated in Table VIII can be depicted in Figure 24.

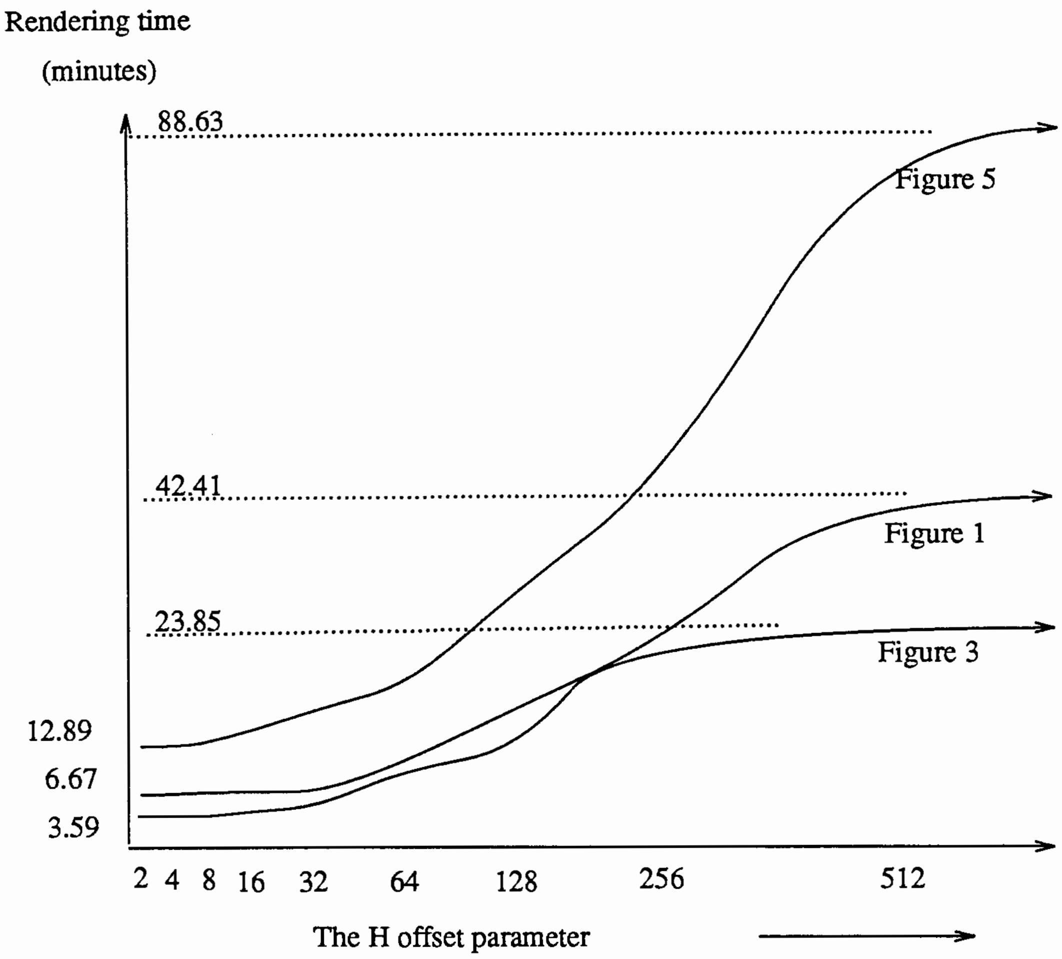


Figure 24. The curves show the range performance of the system.

Figure 24 shows us that balancing workloads among processors is one of the most important steps in utilizing the multiprocessing power.

The performance of a single processor, static scheduling and dynamic scheduling is listed in Table IX. The results indicate that dynamic scheduling method is more efficient than static scheduling. Dynamic scheduling improves the rendering process by a factor of "(N-1)" (number of processors) but for static scheduling the speed-up is slightly less.

Finally, the researchers could produce a computer generated-image within a few minutes. In fact, the researchers could improve further the rendering process by increasing the number of processors before the intercommunication bottleneck becomes a

TABLE IX

TIMINGS OF IMAGES PERFORMANCE FOR STATIC AND DYNAMIC SCHEDULING
(4 PIXELS PER RAY, H = 2, AND THE NUMBER OF PROCESSORS, P = 7)

Fig	Light	Shadow	Reflect	Process time			Factor	
				p=1	stat	dyna	stat	dyna
1	10	61042	32306	42.41	6.76	6.27	6.27	6.76
2	8	37148	32306	26.75	3.90	3.87	6.86	6.91
3	10	9514	28727	23.85	3.93	3.42	6.07	6.97
4	6	22616	44238	26.04	4.07	3.95	6.40	6.59
5	6	41285	51504	54.77	8.17	7.89	6.70	6.94

problem. The speed performance also depends on the intercommunication between the global bus and processors, or between processors themself. It means that after it reaches the maximum performance, it begins to decrease due to the intercommunication bottleneck between the global bus and processors, as depicted in Figure 25.

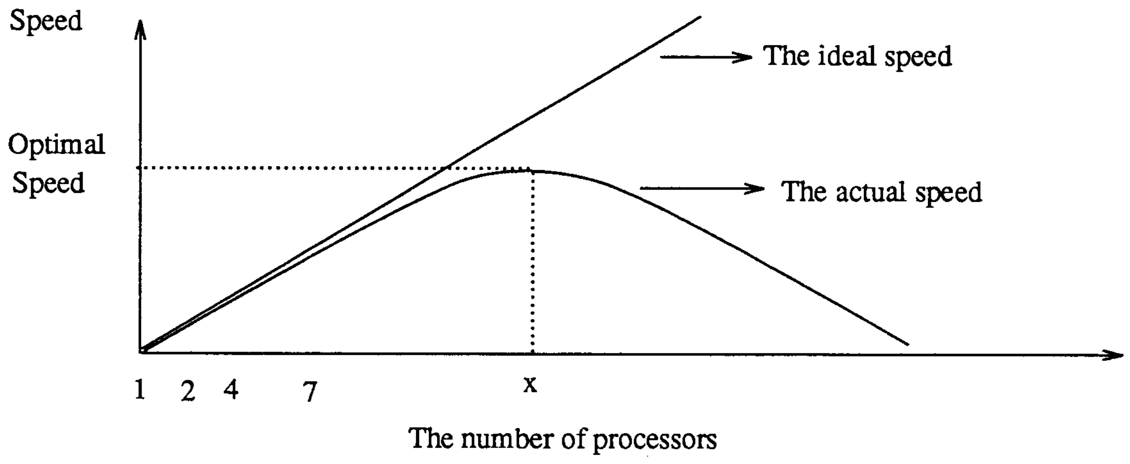


Figure 25. The performance of parallelism. Where x is the optimal number of processors.

Unfortunately, at the present time, Sequent's Balance 8000 is running with eight processors and without floating point accelerator chips. The researchers could not improve the rendering performance linearly by increasing the number of processors because of Amdahl's law.

CHAPTER VII

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

A simple ray-tracing algorithm has been developed and implemented. The algorithm works for any geometric surface and multiple light sources. The concept of space-tracing and a bounding volume are applied in this algorithm which speeds up the rendering process. An octree structure and rectangular parallelepipeds are chosen in representing the environment.

Recent developments in hardware, such as cheap memory, faster computer, and multiprocessor systems have made the ray-tracing algorithm more an available, viable, and desirable choice for a wide variety of image generation applications. In the near future, the attributions of hardware and software indicate that the algorithm holds out promise for practice. Although implementation of the algorithm is in its simplest form, it illustrates the excellence of the technique. New development methods are necessary to render highly sophisticated images in a reasonable amount of time on currently available microcomputers. Once the algorithm becomes reasonably fast (within a few minutes), it can be interfaced with the present graphic packages to produce animation effects.

The researchers have proposed a parallel processing system for images synthesis using the ray-tracing technique. The parallelism is based on the image (screen) division. The researchers divide the active-region on the image plane into smaller packets and distribute these packets among the processors. This scheme allows the calculation of pixels within the active-region simultaneously in the parallel environment. However, if the computation of propagating rays on a particular packet (processor) through the environment were substantially more intensive than other packets, then this screen division method does not significantly improve upon the single processor because most of the rendering process will be dominated by that particular processor. The tightly coupled architecture (Sequent's Balance Computer System) is suitable for implementing the algorithm discussed above in a multiprocessing system because the process can be performed by the same operations repeatedly on large collections of data. This can be done by creating multiple, identical processes and assigning a portion of the data to each process. An Almost linear speed-up can be obtained on a Balance multiprocessor system, if the tasks evenly among the processors are distributed. This linearity is a function of distribution of the workloads among the processors and the number of processors are used. In other words, the system should run with as much balance as possible so that researchers can utilize the full power of the multiprocessing system with an exception that there is no intercommunication bottleneck. This is almost true for "Balance 8000" with eight processors because it has a high speed common bus.

7.2 FUTURE WORK

So far, the ray tracing algorithm has not been implemented at Portland State University. Therefore, a simple, straight forward ray tracing algorithm has been implemented. Undoubtedly, there is no "optimal" space-tracing method for all applications. There are still many openings in research areas that have to be developed. The following research areas seem to be particularly promising :

1. The algorithm can be used to incorporate a wide range of primitive geometric types and should be easily extensible to new types.

2. The algorithm can be improved in the areas of primary and secondary ray intersection. They involve the intersection of rays with objects along the rays' paths.

3. The algorithm can be developed to make it more suitable for implementing on advanced special hardware. This special hardware purpose could have the following characteristics :

a. The computational time should be fairly predictable and relatively constant over a wide variety of scene complexities.

b. Its operations should be relatively easy to implement in hardware.

4. The basic operation of tracing a ray in an environment is fundamental to many applications outside of image generation. Therefore, this technique can be developed for applications in other fields. For example, in geometric modeling, it provides a technique for performing various analyses on objects defined through constructive solid geometry operation. It can also be used for penetration analyses, interference checking, etc. Some of these applications can directly benefit from this method, while some will require their integration into other algorithms.

5. The algorithm can also be developed to make it more suitable for implementing on advanced architecture. The algorithm can be tailored to a particular parallel architecture so that it can optimize the processing system and deliver good results.

REFERENCES

[1] Cook, Robert L. and Torrance, Kenneth E., "A Reflection Model for Computer Graphics", Computer Graphics 15, 3 (August 1981), 307-316.

[2] Whitted, Turner., "An Improved Illumination Model for Shaded Display", Comm. ACM 23, 6 (June 1980), 343-349.

[3] Clark, J.H., "Hierarchical Geometric Models for Visible Surface Algorithms ", Comm. ACM 19, 10 (October 1976), 547-554.

[4] Appel, Arthur., "Some Techniques for Shading Machine Rendering of Solids", AFIPS Spring Joint Computer Conference, 1968, 37-45.

[5] Goldstein, R.A. and Nagel, R., "3-D Visual Simulation", Simulation, Vol. 16, January 1971, 25-31.

[6] Bui-Tuong Phong., "Illumination for Computer Generated Pictures", Comm. ACM 18, 6 (June 1975), 311-317.

[7] Rubin, Steven M. and Whitted, Turner., "A 3-Dimensional Representation for Fast Rendering of Complex Scenes", Computer Graphics 1980, 110-116.

[8] Hanrahan, Pat., "Ray Tracing Algebraic Surfaces", Computer Graphics, July 1983, 83-90.

[9a] Hall, Roy A. and Greenberg, Donald P., "A Testbed for Realistic Image Synthesis", IEEE CG&A, November 1983, 10-20.

[9b] Hall, Roy A., "A Methodology for Realistic Image Synthesis", Masters Thesis, Cornell University, 1983.

[10] Kajiya, James T., "New Procedures for Ray Tracing Procedurally Defined Objects", ACM Trans. Graphics 2, 3 (July 1983), 161-181.

[11] Weghorst, H., Hooper, G. and Greenberg, D. P., "Improved Computational Methods for Ray Tracing", ACM Trans. Graphics 3, 1 (January 1984), 52-69.

[12a] Van Wijk, Jarke J., "Ray Tracing Objects Defined By Sweeping Planar Cubic Splines", ACM Trans. Graphics 3, 3 (July 1984), 223-237.

[12b] Van Wijk, Jarke J., "Ray Tracing Objects Defined By Sweeping A Sphere", EUROGRAPHICS 1984, 73-82.

[13] Glassner, A. S., "Space Subdivision for Fast Ray Tracing", IEEE CG&A, October 1984, 15-22.

[14] Doctor, L. and Torborg, J., "Display Techniques for Octree-Encoded Objects ", IEEE CG&A, (July 1981), 29-38.

[15] Meager, D., "Geometric Modelling Using Octtree Encoding", Computer Graphics and Image Processing, Vol 19, 2 (1982), 129-147.

[16] Tanimoto, S. L. and Jackins, C. L., "Octtrees and Their Use In Representing Three-Dimensional Objects", Computer Graphics and Image Processing, Vol 14, 3 (1980), 249-270.

[17] Gargantini, D., "Linear Octtrees for Fast Processing of Three-Dimensional Objects", Computer Graphics and Image Processing, Vol 20, 4 (1982), 265-274.

[18] Osterhaug, A., "Guide To Parallel Programming On Sequent Computer Systems," 2nd ed., Prentice Hall, New Jersey, 1989.

[19] Foley, J. D. and Van Dam, Andries., "Fundamental of Interactive Computer Graphics," Addison-Wesley, 1982.

[20] Sutherland, I. E., Sproull, R. F. and Schumacker, R. A., " A Characteristic of Ten Hidden Surface Algorithms," ACM Computing Survey 6, 1 (May 1974), 1-55.

[21] Sproull, R. and Newman, R., "Principles of Interactive Computer Graphics," 2nd ed., McGraw-Hill, New York, 1979.

[22] Gouraud, H., "Continuous Shading of Curved Surface," IEEE Transactions on Computers 20, 6 (June 1971), 623-628.

[23] Lane, J. M. and Carpenter, L. C., "A Generalized Scan Line Algorithm for Computer Display of Parametrically Defined Surface," Computer and Image Processing, 11, 1979, 290-297.

[24] Amanatides, John., "Ray Tracing With Cones", Proc. SIGGRAPH 1984, July 1984, 129-136.

[25] Ballard, Dana H., "Strip Trees: A Hierarchical Representation of Curves", Comm. ACM 24, 5 (May 1981), 310-321.

[26] Aono, M. and T. Kunii., "Botanical Tree Image Generation", IEEE CG&A, Vol. 4, 5 (May 1984), 10-34.

[27] Atherton, P., Weiler, K. and Greenberg, D., "Polygon Shadow Generation", Proc. SIGGRAPH 1978, August 1978, 275-281.

[28] Atherton, Peter., "A Scan-Line Hidden Surface Removal Procedure for Constructive Solid Geometry", Proc. SIGGRAPH 1983, Vol. 17, 3 (July 1983), 73-82.

[29] Barr, Alan H., "Global and Local Deformations of Solid Primitives", Proc. SIGGRAPH 1984, July 1984, 21-30.

[30] Blinn, J.F. and Newell, M.E., "Texture and Reflection In Computer Generated Images", Comm. ACM 19, 10 (1976), 542-547.

[31] Blinn, J.F., "Models of Light Reflection for Computer Synthesized Pictures", Proc. SIGGRAPH 1977, Vol. 11, 2 (1977), 192-198.

[32] Blinn, J.F., "Computer Display of Curved Surfaces", Ph.D. Thesis, Univ. of Utah, Salt Lake City, Utah, 1978.

[33] Blinn, J.F., "Simulation of Wrinkled Surfaces", Proc. SIGGRAPH 1978, 286-292.

[34] Blinn, J.F., "A Scan Line Algorithm for Displaying Parametrically Defined Surfaces", Proc. SIGGRAPH 1979 (Special Issue), August 1979.

[35] Blinn, J.F., "A Generalization of Algebraic Surface Drawing", ACM Trans. Graphics 1, 3 (1982), 235-256.

[36] Blinn, J. F. and Newman, R., "Light Reflection Function for Simulation of Clouds and Dusty Surface," Computer Graphics 16, 3 (July 1982), 21-29.

[37] Bouknight, W.J. and K.C.Kelley., "An Algorithm for Half-Tone Representation of Three-Dimensional Objects", Proc. AFIPS JSCC, Vol. 36, 1980, 1-10.

[38] Carpenter, Loren., "The A-Buffer, An Anti-Aliased Hidden Surface Method", Proc. SIGGRAPH 1984, 103-108.

[39] Catmull, Edwin., "Computer Display of Curved Surfaces", Proc. IEEE Conf. Computer Graphics, Pattern Recognition, and Data Structures, 1975, 11-17.

[40] Catmull, Edwin., "A Hidden-Surface Algorithm With Anti-Aliasing", Proc. SIGGRAPH 1978, Vol. 12, 3 (July 1978), 6-10.

[41] Catmull, Edwin. and Alvy, Ray Smith., "3-D Transformations of Images in Scanline Order", Computer Graphics 1980, 279-285.

[42] Catmull, Edwin., "An Analytic Visible Surface Algorithm for Independent Pixel Processing", Proc. SIGGRAPH 1984, July 1984, 109-115.

[43] Clark, James H., "A Fast Algorithm for Rendering Parametric Surfaces", Proc. SIGGRAPH 1979 (Special Issue), August 1979.

[44] Clark, James H., "Design Surface in 3-D", Comm. ACM 19, 8 (August 1976), 454-460.

[45] Cook, Robert L., "A Reflectance Model for Realistic Image Synthesis", Masters thesis, Cornell University, 1981.

[46] Cook, R.L., Porter, T. and Carpenter, L., "Distributed Ray Tracing", Computer Graphics, July 1984, 137-146.

[47] Cook, Robert L., "Shade Trees", Computer Graphics, July 1984, 223-231.

[48] Crow, Franklin C., "Shadow Algorithms for Computer Graphics", Computer Graphics 11, 2 (July 1977), 242-248.

[49] Crow, Franklin C., "The Aliasing Problem In Computer-Generated Shaded Images", Comm. ACM 20, 11 (November 1977), 799-805.

[50] Crow, Franklin C., The Use of Grayscale for Improved Raster Display of Vectors and Characters", Computer Graphics 12, 3 (1978), 1-5.

[51] Crow, Franklin C., "A Comparison of Anti-Aliasing Techniques", IEEE CG&A, January 1981, 40-48.

[52] Crow, Franklin C., "Computational Issues in Rendering Anti-Aliased detail", Proc. COMPCON Spring 82, 238-244.

[53] Crow, Franklin C., "A More Flexible Image Generation Environment", Computer Graphics 16, 3 (1982), 9-18.

[54] Duff, Tom., "Smoothly Shaded Renderings of Polyhedral Objects on Raster displays", Computer Graphics 1979, 270-275.

[55] Evans, R., "An Introduction to Color," John Wiley & Sons, New York, 1984.

[56] Fournier, Alain, Don Fussell, and Loren Carpenter., "Computer Rendering of Stochastic Models", Comm. ACM 25, 6 (June 1982), 371-384.

[57] Fuchs, H., Kedem, Z., and Naylor, B., "On Visible Surface Generation By A Priori Tree Structures", Computer Graphics, July 1980, 124-133.

[58] Fuchs, H., Abram, G.D., and Grant, E.D., "Near Real-Time Shaded Display of Rigid Objects", Computer Graphics 1983, 65-73.

[59] Fujimoto, Akira. and Kansei, Iwata., "Jag-Free Images on Raster Displays", IEEE CG&A, December 1983, 26-34.

[60] Gardner, G.Y., "Simulation of Natural Scenes Using Textured Quadratic Surfaces", Computer Graphics 18, 3 (July 1984), 11-20.

[61] Goral, C.M., Torrence, K.E and Greenberg, D.P, "Modeling The Interaction of Light Between Diffuse Surfaces", Computer Graphics 18, 3 (July 1984), 213-222.

[62] Gouraud, Henri., "Computer Display of Curved Surfaces", Ph.D. dissertation, Univ. of Utah, Salt Lake City, 1971.

[63] Hecht, Eugene and Zajac, Alfred. "Optics", Addison-Wesley Publishing Company, 1979.

[64] Heckbert, P. and Hanrahan, P., "Beam Tracing Polygonal Objects", Computer Graphics 1984, 119-128.

[65] Kajiya, James T. and M. Ullner., "Filtering High Quality Text for Displaying on Raster Scan Devices", Computer Graphics 15, 3 (1981), 7-15.

[66] Kajiya, James T., "Ray Tracing Parametric Patches", Computer Graphics 1982, 245-254.

[67] Kajiya, James T. and Von Herzen, Brian P., " Ray Tracing Volume Densities", Computer Graphics, July 1984, 165-174.

[68] Kay, Douglas S., "Transparency, Refraction, and Ray Tracing for Computer Synthesized Images", M.S. Thesis, Cornell University, January 1979.

[69] Kay, Douglas S. and Donald Greenberg., "Transparency for Computer Synthesized Images", Computer Graphics 1979, 158-164.

[70] Lane, J.M., Carpenter, L.C., Whitted, T., and Blinn, J.F., "Scan Line Methods for Displaying Parametrically Defined Surfaces", Comm. ACM 23, 1 (January 1980), 23-34.

[71] Liang, You-Dong and Brian A. Barsky., "An Analysis and Algorithm for Polygon Clipping", Comm. ACM 26, 11 (November 1983), 868-877.

[72] Meyer, Gary W. and Greenberg, Donald P., "Perceptual Color Spaces for Computer Graphics", Computer Graphics, August 1980, 254-261.

[73] Meyer, Gary W., "Colorimetry and Computer Graphics", Cornell University, Program for Computer Graphics, Report 83-1, April 1983.

[74] Miller, Gene S. and C. Robert Hoffman., "Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments", SIGGRAPH 1984 Course Notes: Advanced Computer Graphics Animation.

[75] Newell, J., Newell, R., and Sancha, T., "A Solution To the Hidden Surface Problem", Proc. ACM Nat'l Conf., 1972, 443-450.

[76] Newell, Martin E. and Carlo H. Sequin., "The Inside Story on Self-Intersecting Polygons", Lambda, Second Quarter, 1980, 20-24.

[77] Norm, D., Kirkpatrick, D. G., and Walsh, J. P., "Hierarchical Approaches to Hidden Surface Intersection Testing," Proceeding of Graphics Interface 1982, May 1982, 49-56.

[78] Porter, Thomas and Duff, Tom., "Compositing Digital Images", Computer Graphics 1984, 253-259.

[79] Potmesil, Michael and Chakravarty, Indranil., "Synthetic Image Generation With A Lens and Aperture Camera Model", ACM Trans. on Graphics 1, 2 (April 1982), 85-108.

[80] Roth, Scott D., "Ray Casting for Modeling Solids", Computer Graphics and Image Processing 18, 1982, 109-144.

[81] Samet, H., "The Quadtree and Related Hierarchical Data Structures", ACM Computing Surveys, (June 1984), 187-260.

[82] Sederberg, Thomas W. and Anderson, David C., "Ray Tracing Steiner Patches", Computer Graphics, July 1984, 159-164.

[83] Smith, Alvy Ray., "Incremental Rendering of Textures in Perspective", SIGGRAPH 1980 Course Notes: Computer Animation.

[84] Steinberg, Herbert A., "A smooth Surface Based on Bi-Quadratic Patches", IEEE CG&A, September 1984, 20-23.

[85] Sutherland, I. E., Sproull, R. F. and Schumacker, R. A., "Sorting and The Hidden Surface Problem," In Proc, AFIPS 1973 National Computer Cong., Vol 42, 1973.

[86] Sutherland, I. E., "Reentrant Polygon Clipping," Comm. ACM 17, 1, 1974, 32-34.

[87] Verbeck, Channing P. and Donald P. Greenberg., "A Comprehensive Light-Source Description for Computer Graphics", IEEE CG&A, July 1984, 66-75.

[88] Warnock, John E., "A Hidden Surface Algorithm for Computer Generated Half-Tone Picture Representation", Tech. report 4-15, Univ. of Utah, Salt Lake City, June 1969.

[89] Warn, David R., "Lighting Controls for Synthetic Images", Computer Graphics, July 1983, 13-21.

[90] Watkins, G.S., "A Real-Time Visible Surface Algorithm", Ph.D. dissertation and tech. report UTECH-CSC-70-101, Univ. of Utah, Salt Lake City, June 1970.

[91] Weiler, Kevin, and Atherton, Peter., "Hidden Surface Removal Using Polygon Area Sorting", Computer Graphics 1977, 214-222.

[92] Weiman, Carl F.R., "Continuous Anti-Aliasing Rotation and Zoom of Raster Images", Computer Graphics 1980, 286-293.

[93] Whitted, Turner., "Anti-Aliased Line Drawing Using Brush Extrusion", Computer Graphics 17, 3 (July 1983), 151-156.

[94] Whitted, Turner and David M. Weimer., "A Software Testbed for The Development of 3d Raster Graphics Systems", ACM Trans. Graphics 1, 1 (January 1982), 43-58.

[95] Williams, Lance., "Casting Curved Shadows on Curved Surfaces", Computer Graphics 12, 3 (August 1978), 270-274.

[96] Williams, Lance., "Pyramidal Parametrics", Computer Graphics 17, 3 (July 1983), 1-12.

[97] Wyszecki, G. and Stiles, W.S., "Colour Science", 2nd ed. John Wiley and Sons, 1982.

APPENDIX

Rasterfile is a typical Sun's file format for displaying raster images. It is composed of three parts as depicted in Figure 26.

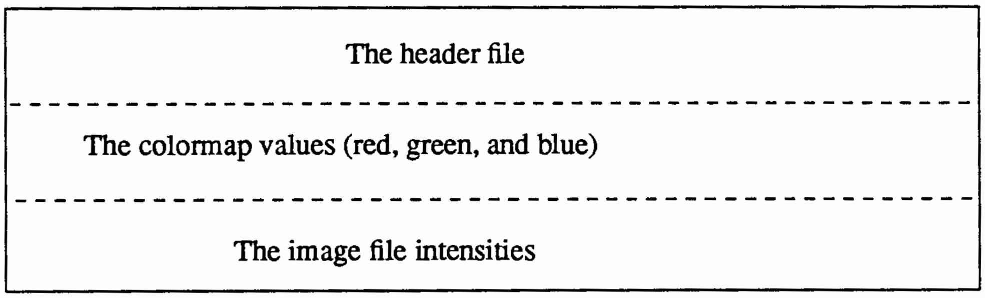

The header file
The colormap values (red, green, and blue)
The image file intensities

Figure 26. A typical rasterfile format.

The header file is consists of eight integers. A set of colormap is made of the value of red, green, and blue components of the color. The image file stored a line at a time in increasing high (y) order. Each line of the image is rounded up to a multiple of 16 bits.

The header file is made of the following structure.

```
struct rasterfile {
        int    ras_magic;
        int    ras_width;
        int    ras_height;
        int    ras_depth;
        int    ras_length;
        int    ras_type;
        int    ras_maptype;
        int    ras_maplength;
};
```

The details of header file is explained as follows :

- The ras_magic is an integer (magic) number.

• The ras_width and ras_height fields represent the image's width (Xres) and height (Yres) in pixels.

• The ras_depth field corresponds to the frame buffer (colormap). Its value is either 1, or 8. If the value is equal to 1, it means that the file will represent white and black images. Otherwise, it is color images which have a maximum 256 colors at a time.

• The ras_length field indicates the length in bytes of the image data or it is equal to a product of the ras_width and the ras_height fields.

• The ras_type indicates the type of image file.

• The ras_maptype and ras_maplength fields represent type and length in bytes of the colormap values, respectively.

An example of the header file is described as follows :

```
raster.ras_magic        = RAS_MAGIC;       /*magic number*/
raster.ras_width        = Xres;            /*width of image*/
raster.ras_height       = Yres;            /*height of image*/
raster.ras_depth        = 8;               /*depth of pixel*/
raster.ras_length       = Xres * Yres;     /*length of image*/
raster.ras_type         = RT_STANDARD;     /*type of file */
raster.ras_maptype      = RMT_EQUAL_RGB;   /*type of colormap*/
raster.ras_maplength    = 3 * 256;         /*length of colormap*/
```