

RAY TRACING GEOMETRIC MODELS AND
PARAMETRIC SURFACES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCES
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

154
36-9
1989

By
Veysel İZLER
July 1989

RAY TRACING GEOMETRIC MODELS AND
PARAMETRIC SURFACES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER
ENGINEERING AND INFORMATION SCIENCES
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Veysi İşler
1989

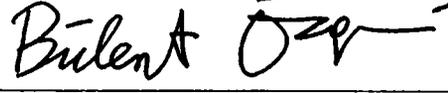

tarafından hazırlanmıştır.

QA
76.9

is4
1989

B 1865

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



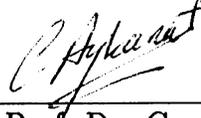
Prof. Dr. Bülent Özgüç (Principal Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Prof. Dr. Mehmet Baray

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



Asst. Prof. Dr. Cevdet Aykanat

Approved for the Institute of Engineering and Sciences:



Prof. Dr. Mehmet Baray, Director of Institute of Engineering and Sciences

ABSTRACT

RAY TRACING GEOMETRIC MODELS AND PARAMETRIC SURFACES

Veysi İşler

M.S. in Computer Engineering and Information Sciences

Supervisor: Prof. Dr. Bülent Özgüç

1989

In many computer graphics applications such as CAD, realistic displays have very important and positive effects on designers using the system. There are several techniques to generate realistic images with the computer. Ray tracing gives the most effective results by simulating the interaction of light with its environment. Furthermore, this technique can be easily adopted to many physical phenomena such as reflection, refraction, shadows, etc. by which the interaction of many different objects with each other could be realistically simulated. However, it may require excessive amount of time to generate an image. In this thesis , we studied the ray tracing algorithm and the speed problem associated with it and several methods developed to overcome this problem. We also implemented a ray tracer system that could be used to model a three dimensional scene and find out the lighting effects on the objects.

iii

Keywords: Ray tracing, Octree , Shading , Realistic Images.

ÖZET

GEOMETRİK MODELLER VE PARAMETRİK YÜZEYLER ÜZERİNDE IŞIN İZLEME

Veysi İşler

Bilgisayar Mühendisliği ve Enformatik Bilimleri Bölümü

Yüksek Lisans

Tez Yöneticisi: Prof. Dr. Bülent Özgüç

1989

Bilgisayarlı bir çok uygulamada gerçeğe uygun görüntüler sıkça kullanılmaktadır. Bu nedenle, bilgisayarda gerçekçi görüntüler elde etmek için çeşitli yöntemler geliştirilmiştir. Işın izleme bunlar arasında en etkili gerçekçi görüntüler elde etmeye yarayan bir yöntemdir. Işın izlemede temel nokta, sunulacak sahnedeki ışık ve modellerin çevreleri ile etkileşimlerinin benzetimi yapılarak yansıma, gölgeleme ve kırılma gibi doğal olayları bilgisayarda hesaplamaktır. Işın izleme metodu bu kadar yararlı olmasına rağmen, bu teknikle elde edilen görüntüler aşırı hesaplama zamanı gerektirmektedir. Bu araştırmada ışın izleme metodu çalışılmış ve bu yöntemin sahip olduğu avantajlar ve dezavantajlar incelenmiştir. Ayrıca, ışın izlemedeki problemleri çözmek için geliştirilen metotlar araştırılıp geliştirilen ışın izleme sisteminde kullanılmıştır.

v

Anahtar kelimeler : Işın İzleme , Sekizli Ağaç, Tonlama, Gerçekçi Görüntü.

ACKNOWLEDGEMENT

I wish to thank very much my supervisor Professor Bülent Özgüç, who has guided and encouraged me during the development of this thesis.

I am grateful to Professor Mehmet Baray and Dr. Cevdet Aykanat for their remarks and comments on the thesis.

I would also like to express my gratitude to Dr. Varol Akman, who provided me with several papers that were very helpful to me.

My sincere thanks are due to Cemil Türün for his technical help.

Finally, I appreciate my colleagues Aydın Açıkgöz and Uğur Güdükbay for their valuable discussions.

TABLE OF CONTENTS

| | | |
|----------|----------------------------------|----------|
| 1 | INTRODUCTION | 1 |
| 2 | THE RAY TRACING ALGORITHM | 4 |
| 2.1 | The Shading Model | 4 |
| 2.1.1 | Ambient Light | 4 |
| 2.1.2 | Diffuse Reflection | 6 |
| 2.1.3 | Specular Reflection | 6 |
| 2.1.4 | Reflection | 6 |
| 2.1.5 | Transparency | 7 |
| 2.1.6 | Shadows | 7 |
| 3 | SPEEDING UP THE ALGORITHM | 9 |
| 3.1 | Bounding Volumes | 10 |
| 3.2 | Spatial Subdivision | 11 |
| 3.3 | A Hybrid Technique | 15 |
| 3.4 | Parallel Ray Tracing | 16 |

| | |
|--|-----------|
| 4 AN IMPLEMENTATION OF SPACE SUBDIVISION USING OCTREE | 18 |
| 4.1 Overview | 18 |
| 4.2 Octree Building and Storage | 21 |
| 4.3 Movement to the Next Box | 23 |
| 5 A RAY TRACER SYSTEM | 25 |
| 5.1 Defining the Scene | 25 |
| 5.1.1 Textual Input | 26 |
| 5.1.2 Interactive Tool | 30 |
| 5.2 Processing: Ray Tracer | 31 |
| 5.3 Displaying the Image | 33 |
| 5.4 Examples and Timing Results | 34 |
| 6 CONCLUSION | 40 |
| APPENDICES | 49 |
| A THE USER'S MANUAL | 47 |
| A.1 The Modules | 47 |
| A.2 The Interface | 48 |

LIST OF FIGURES

| | | |
|-----|--|----|
| 2.1 | Tracing of one ray. | 5 |
| 2.2 | Vectors used in shading computations. | 8 |
| 3.1 | Several types of bounding volumes. | 10 |
| 3.2 | Adaptive space subdivision. | 12 |
| 3.3 | Subdivision of space into equally sized cubes. | 13 |
| 4.1 | Naming the boxes. | 20 |
| 4.2 | Hash table to store the generated boxes. | 22 |
| 5.1 | Four spheres and a triangle. | 36 |
| 5.2 | Fifty spheres. | 36 |
| 5.3 | Two hundred spheres. | 37 |
| 5.4 | Twenty four spheres. | 37 |
| 5.5 | Sphere above chessboard. | 38 |
| 5.6 | Shield and a sphere. | 38 |
| 5.7 | Boxes and superquadrics. | 39 |
| 5.8 | Mirrors and reflections. | 39 |

| | | |
|-----|---------------------------------------|----|
| A.1 | User interface of the system. | 54 |
| A.2 | Selecting objects. | 54 |

LIST OF TABLES

| | | |
|-----|--|----|
| 5.1 | Timing results of figures 5.1, 5.2, 5.3. | 35 |
| 5.2 | Timing results of figure 5.4. | 35 |

1. INTRODUCTION

One of the most important goals in computer graphics is to generate images that appear realistic, that is, images that can deceive a human observer when displayed on a screen. Realistic images are used widely in many computer graphics applications. Some of them are :

- CAD
- Animation and Visualization
- Simulation
- Education
- Robotics
- Architecture
- Inside Decoration
- Advertising
- Reconstruction for Medical and Other Purposes

There are several advanced techniques used to add realism to a computer generated picture. All these techniques involve both hidden-surface and shading computations. The hidden-surface computation determines which parts of object surface are visible, which ones are not. Hidden-surface elimination is essential for realistic display of objects. Once visible surfaces have been identified, for instance, by a hidden-surface method, a shading model is used to

compute the intensities and colors of the surfaces. The shading model does not exactly simulate the behavior of light and surfaces in the real world but only approximates actual conditions. The design of the model is a compromise between precision and computing expense.

The initial approaches to generate realistic images on a computer were primarily hidden-surface removal and shading of surfaces without considering the effects of objects in the environment. However, to obtain more realistic and detailed images, global shading in addition to hidden-surface operation should be performed.

Ray tracing was the first method introduced in order to generate very realistic images by including the effects of shadows and reflections in addition to transparency of neighboring surfaces [43]. The basic idea in ray tracing is to find out the effect of the light source(s) on the objects in the scene. Ray tracing that performs a global shading gives more depth cues than the local shading [19]. This is due to the fact that, the images generated by ray tracing algorithm may contain a number of optical effects such as shadows, reflection, refraction and transparency. That is, both geometric and shading information are calculated for each pixel of the image [23].

Although ray tracing is so useful in generating very realistic images, it has two major drawbacks: one is its computational complexity and the other is the aliasing caused by the inherent point sampling nature of the technique [2]. Due to these difficulties, this powerful technique cannot be included in most interactive systems. Once the time spent is reduced to a reasonable amount, this elegant technique could be widely used in many applications.

Ray tracing algorithms can be practical in a wide variety of applications when the following conditions are satisfied [23] :

- First of all, the rendering time should be independent of the scene complexity. That is, when the number of objects in the scene increases, the time to generate an image should remain close to a constant.
- Secondly, the computation time should be reasonable for each scene so that we do not spend an excessive amount of time to generate an image.

- Thirdly, each generated ray should take almost a constant time independent of the origin or direction of the ray.
- Another important condition is that, the ray tracer should not accept only specific types of geometric objects but must also be extendible to a variety of types easily.
- Finally, the user should not get involved in the construction of the auxiliary data structures used in the algorithm and the algorithm should be amenable to implementation on parallel architectures.

In chapter 2 , a brief description of the algorithm will be presented. Each ray tracing algorithm adopts an appropriate shading model to find a color value on an object's surface. As it is pointed out earlier, there is a trade-off in using a shading model against the time spent on the computer. A simple shading model which gives satisfactory results is also given in this chapter.

Since ray tracing consumes most of the time in testing intersections of the rays with the objects, researchers have attempted to reduce these intersection calculations. In chapter 3, some historical attempts to speed up the algorithm are overviewed.

One of the methods used to speed up the algorithm is to subdivide the space into disjoint volumes. Many variations of this method exist to accomplish this. In chapter 4, a scheme that uses "Octree" hierarchical data structure is presented with its important implementation details.

As one important part of the thesis, we implemented a ray tracing system to generate realistic images. The scene to be processed can be defined by an interactive tool that has several facilities. The tool with the other parts of the system is described in chapter 5. Finally, a conclusion and future directions are given in chapter 6.

2. THE RAY TRACING ALGORITHM

In a naive ray tracing algorithm, a ray is shot for each pixel from the view point into the three dimensional space as seen in Figure 2.1. Each object is tested to find the first surface point hit by the ray. The color intensity at the intersection point is computed and returned as the value of the corresponding pixel. In order to compute the color intensity at the intersection point, the ray is then reflected from this surface point to determine whether the reflected ray hits a surface point or a light source. If the reflected ray ends at a light source, highlights or bright spots are seen on this surface. If the reflected ray hits another surface of an object, the color intensity at the new intersected point is also taken into account. This gives reflection of a surface on another. When the object is transparent, the ray is divided into two components and each ray is traced individually.

As explained above, the color value at the intersection point gives the color value of the pixel associated with the intersecting ray. Therefore, having found the intersection point, the color at that point should be calculated according to a shading model. A simple one is given in the next section.

2.1 The Shading Model

2.1.1 Ambient Light

Initially the surface has ambient color which is a result of the uniform ambient light emitted by the surrounding objects. That is, a surface can still be visible even if it is not exposed directly to any light source. In this case, the surface

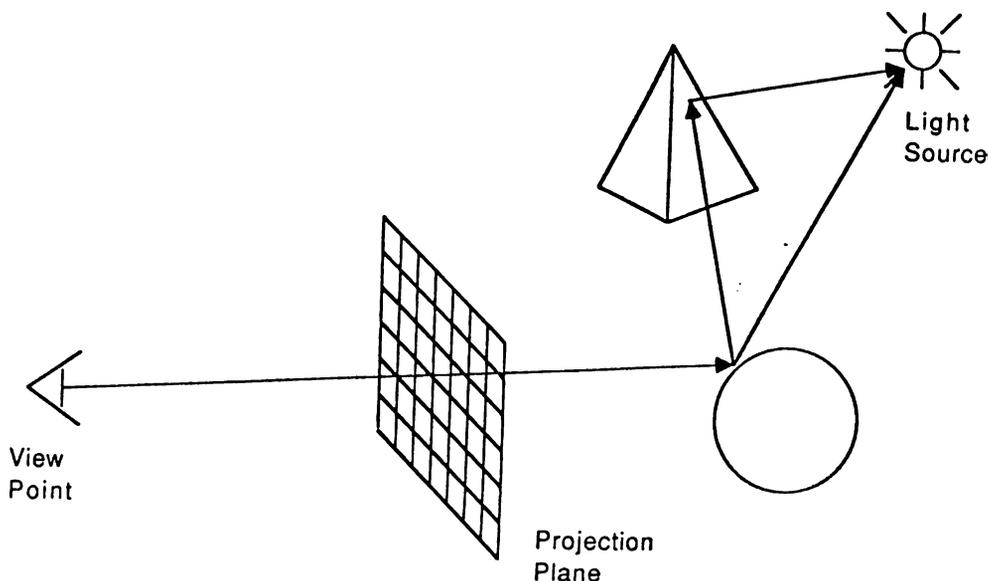


Figure 2.1: Tracing of one ray.

is illuminated by the objects in its vicinity. Ambient color behaves similarly regardless of the viewing direction . We can express the intensity at a point on the surface of an object as

$$Color_p = k_d Color_a$$

where k_d is the coefficient of reflection, $Color_a$ is the ambient light intensity. k_d takes values between 0 and 1. It is 1 for highly reflective surfaces. Unfortunately, ambient color alone does not give satisfactory results. Therefore, the effect of the light source(s) on the surface according to the orientation of the surface should also be considered in the computation of shading. That is, the diffuse reflections and the specular reflections add very much to the realism of the image. These computations use the reflection, normal and other vectors as seen in Figure 2.2. N is the unit vector normal to the point being shaded. L is the unit vector from the point to the light source. R is the unit vector in the reflection direction. The angle between R and N is equal to the angle between V and N . V is the unit vector from the viewing point to the point on the surface.

2.1.2 Diffuse Reflection

Diffuse reflection computation is based on the Lambert cosine law, which states that the intensity of the reflected light depends on the cosine of the angle between the normal of the surface and the ray to the light [33]. The cosine of this angle is the dot product of two unit vectors in the light and normal vector directions.

Diffuse reflection is computed as

$$Color_p = \frac{k_d Color_l}{d + d_0} (N.L)$$

where $Color_l$ is the intensity of the light source. d represents the distance from a light source to the point being shaded and d_0 is a constant to prevent denominator from approaching zero.

2.1.3 Specular Reflection

Highlights are seen from the view point when incidence light ray is at a certain angle and surface is shiny. The highlights (specular reflection) can be modeled as

$$Color_p = \frac{k_s Color_l}{d + d_0} (V.R)^n$$

where n is a constant related to the surface optical property. It is zero if the surface is dull and very large if the surface is a perfect mirror. k_s is a constant for specular reflection depending on the surface property.

2.1.4 Reflection

In order to simulate the reflection of surrounding objects on the point being shaded, a reflection ray is sent from this point and this ray is tested with the objects to find any intersection. If this ray hits any object, the color intensity at the intersected point contributes to our shading computation as follows:

$$Color_p = Color_p + k_r Color_r$$

where $Color_p$ was the intensity computed previously for the point being shaded. $Color_r$ is the intensity at the intersection point. k_r is a constant that is related to the surface property. It is coefficient of reflection.

2.1.5 Transparency

If the shaded object is transparent, the reflections from the objects behind it should also be considered. This reflection contributes to the shading computation as

$$Color_p = (1 - r)Color_t + rColor_b$$

$Color_t$ is the total intensity at the surface point after summing the intensities of the ambient light, the diffuse reflection and the specular reflection. $Color_b$ is the intensity of the surface point behind the transparent object. r is a constant that is related to the transparency of the object. It is 0 if the object is opaque. In other words, the ray that hits a surface continues traveling through the transparent object until it intersects another object. The intensity at the intersection point is taken to contribute to the transparent object. This ray could also be refracted.

2.1.6 Shadows

Shadows that give very strong depth cues to the image can be obtained while finding out the diffuse and specular reflections. The regions of a surface are in shadow if the light sources are blocked by any opaque or semi-transparent object in the scene. This is found out by sending a ray from the point on the surface towards the light sources and testing for intersection of the ray with an object before the light source. If there is any intersection both diffuse and specular reflections become zero.

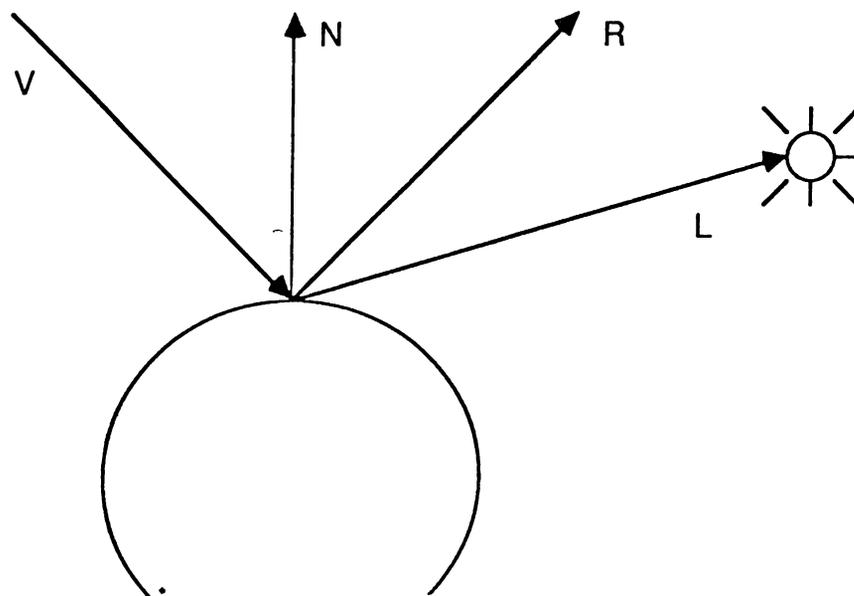


Figure 2.2: Vectors used in shading computations.

3. SPEEDING UP THE ALGORITHM

As mentioned above, the major drawback which prevents ray tracing from being attractive for interactive systems is its computational complexity due to many intersection tests between rays and objects. Whitted has estimated that up to 95% of the time is spent during these intersection tests [43]. They take too much time since all of the objects in the scene have to be tested to find the nearest intersection point with the ray, requiring intensive floating point operations. In order to reduce the processing time in the ray tracing algorithm, the computation for intersection tests should be decreased. There are two basic approaches to do this.

- First, an intersection test should be simple to compute. That is, it should take minimum number of computer cycles. The initial attempts to speed up the ray tracing were based on this approach. To make the intersection tests simple to compute, either intersection tests are made efficient [17,19,29,34,38,39] or bounding volumes explained in the next section are used.
- Second, the number of objects to be tested for intersections should be as few as possible. Not all of the objects in the scene should be tested for intersection with the traced ray as in the naive ray tracing algorithm. Only objects that are highly possible for intersection should be tested. In other words, the objects on the ray's path should be considered for intersection tests. Several methods have been developed to achieve this. They are discussed in the next sections.

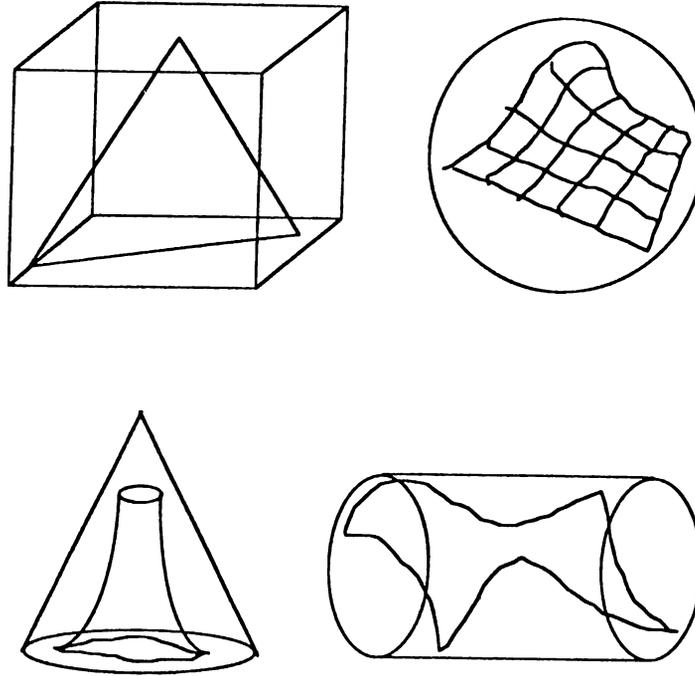


Figure 3.1: Several types of bounding volumes.

3.1 Bounding Volumes

Some simple mathematically defined objects such as spheres, rectangular boxes or cones can be tested for intersection with few number of operations [13]. The complex objects are surrounded by these simple objects (bounding volumes) as in Figure 3.1 and intersections are first tested with the bounding volumes instead of the complex objects . When the ray intersects the bounding volume of an object, tests are carried out for the complex object as well. Obviously, the advantage of using bounding volumes is to eliminate the intersection test with a complex object once its bounding volume is not intersected with the ray. Its disadvantage is the extra time spent in testing the bounding box if the object itself has a possible intersection. It should be noted that the bounding volumes are not mutually exclusive and thus a ray might be tested for an intersection with more than one object. This is another drawback of the bounding volumes, since an intersection test for a complex object may take excessive time. When this type of test is carried out more than once for a ray, it will be even worse.

When there is a large number of objects in the scene, even the tests for

the bounding volumes can take an enormous amount of time. By forming a hierarchy of bounding volumes, a number of tests can be avoided once a bounding volume that surrounds some other bounding volumes is not hit by the ray. Several neighboring objects form one level of the hierarchy. The other drawback of this method is that these hierarchies are difficult to generate and manually generated ones can be poor. That is, they may not be helpful in speeding the intersection operation. Goldsmith has proposed methods for the evaluation of these hierarchies in approximate number of intersection calculations required and for automatic generation of good hierarchies [14].

The bounding volumes used can be spheres, rectangular boxes, polyhedrons, parallel slabs, cones, or surfaces of revolution. The bounding volume chosen for each object in the scene can be different to enclose the object more tightly. This may be needed in order not to test more than one bounding volume for a ray.

3.2 Spatial Subdivision

A different approach to improve the efficiency of ray tracing is called space subdivision [10,23]. The 3-D space that contains the objects is subdivided into disjoint rectangular boxes so that each box contains a small number of objects. A ray travels through the 3-D space by means of these boxes. A ray that enters a box on its way is tested for intersection with only those objects in the box. If there are more than one intersecting object, the nearest point is found and returned. If no object is hit, the ray moves to the next compartment (box) to find the nearest intersection there. This is repeated until an intersection point is found or the ray leaves the largest box that contains all of the objects. It is necessary, in this case, to build an auxiliary data structure to store the disjoint volumes with the objects attached to them [36,37].

This preprocessing will require a considerable amount of time and memory as a price for the speedup in the algorithm. It is, however, worth using the space subdivision particularly when the scene contains many objects, since

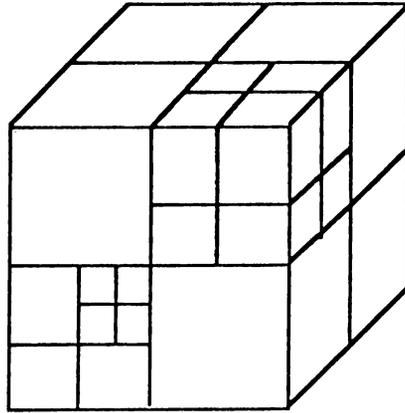


Figure 3.2: Adaptive space subdivision.

this data structure is constructed only once at the beginning and is used during the ray tracing algorithm. The number of rays traced depends both on the resolution of the generated image and the number of objects in the scene. The auxiliary data structure helps to minimize the time complexity of the algorithm by considering only those objects on the ray's way.

There are several techniques that utilize space coherence. They basically differ in the auxiliary data structures used in the subdivision process, and the manner used to pass from one volume to another.

In some ray tracing schemes that utilize the spatial coherence, the space subdivision process is based on the octree spatial representation. An octree is a hierarchical data structure organized such that each node can point to one parent node and eight leaf nodes. Figure 3.2 shows this type of subdivision of the space. In the spatial subdivision ray tracing algorithm, each node of the octree corresponds to a region of the three dimensional space [11,12,14]. The octree building starts by finding a box that includes all of the objects in the scene. A given box is subdivided into eight equally sized boxes according to a subdivision criterion. These boxes are disjoint and do not overlap as the bounding volumes might do. Each of the generated boxes are examined to

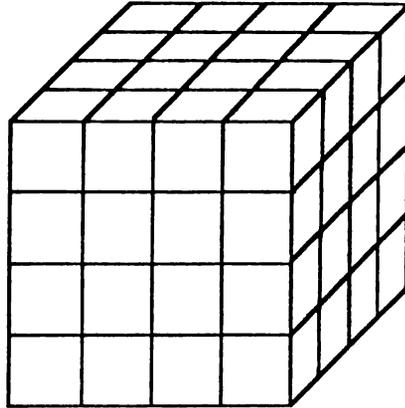


Figure 3.3: Subdivision of space into equally sized cubes.

find which objects of the parent node are included by each child node. The child nodes are subdivided if the subdivision criterion is satisfied. This is carried out recursively for each generated box.

The subdivision criteria may be based on the number of objects in the box, the size of the box, the density ratio of total volume that is enclosed by all objects in the scene to the volume of the box. When the criterion for the number of objects in a given box is very large, each object in the scene is tested for intersection for all rays as in the naive algorithm. No speedup will be achieved, on the contrary the time and the memory will be wasted for the octree data structure. On the other hand, if the number of objects is one for the criterion, there will be many boxes in the structure and the overhead for traveling through the 3-D space may increase.

Kaplan used a data structure which he calls BSP (Binary Space Partitioning) tree to decompose the three-dimensional space into rectangular regions dynamically [23]. BSP is very similar to octree structure in that it also divides the space adaptively. The information is stored as a binary tree (a tree where each non-terminal node can have exactly two child nodes) whose non-leaf nodes are called slicing nodes, and whose leaf nodes are called box

nodes and termination nodes. Each slicing node contains the identification of a slicing plane, which divides all of space into two infinite subspaces. The slicing planes are always aligned with two of the cartesian coordinate axes of the space that contains the objects. The child nodes of a slicing node can be either other slicing, termination nodes or box nodes. A termination node denotes a subspace which is out of the three-dimensional space that does not contain any objects. A box node, on the other hand, is described by the slicing nodes that are traversed to reach it. They denote a subspace containing at least one object. BSP actually encodes the octree in the form of a binary space partitioning tree and it is traversed to find the node containing a given point.

The other spatial subdivision technique for ray tracing is based on the decomposition of the 3-D space into equally sized cubes [10]. Figure 3.3 contains a scene decomposed into equally sized volumes. The size of the cubes determines the number of objects in each cube. Therefore, an optimal cube size must be considered such that the overhead for moving through the boxes should not exceed the time gained in testing intersections.

Fujimoto proposed a scheme that imposes an auxiliary structure called SEADS (Spatially Enumerated Auxiliary Data Structure) on objects in the scene [10]. This structure uses a high level of object coherency. He also developed a traversing tool that fits in well with SEADS to take advantage of the coherency in a very efficient way based on incremental integer logic. This method, called 3DDDA (3-D Digital Differential Analyzer), is a three-dimensional form of the two-dimensional digital-differential analyzer algorithm commonly used for line drawing in raster graphics system. The major advantage of this scheme is related to the manner to travel through 3-D space containing the objects. 3DDDA does not require floating-point multiplications or divisions in order to pass from one subspace (voxel) to the next while looking for intersections, once a preprocessing for the ray has been performed. Fujimoto states that an order of magnitude improvement in ray tracking speed over the octree methods has been achieved. It is also possible to improve the performance of octree traversal by utilizing the 3DDDA method to traverse horizontally in the octree, but vertical level changes must

be traversed as usual.

3.3 A Hybrid Technique

Recently, Glassner has presented techniques for ray tracing of animated scenes efficiently [13]. In his technique, he renders static 4-D objects in spacetime instead of rendering dynamically moving 3-D objects in space. He uses 4-dimensional analogues familiar to 3-dimensional ray-tracing techniques. Additionally, he performs a hybrid adaptive space subdivision and bounding volume technique for generating good, non-overlapping hierarchies of bounding volumes. The quality of the hierarchy and its non-overlapping property is an advantage over the previous algorithms, because it reduces the number of ray-object intersections that must be computed.

The procedure to create such a hierarchy starts by finding a box that encloses all of the objects in the scene, including light sources and the view point. The algorithm then subdivides the space adaptively as in the octree method. The subdivision that is based on a given criterion is performed for each box recursively. The recursion is terminated when no boxes need to be subdivided.

As returning from the recursive calls made by the space subdivision process, the bounding volume hierarchy is constructed. Each box is examined, and a bounding volume is defined that encloses all the objects included within that box. The defined bounding volume must not intersect any other box. That is, it is clipped by the space subdivision box.

At the end of this process, a tree of bounding volumes that has both the nonoverlapping hierarchy of the space subdivision technique and the tight bounds of the bounding volume technique is constructed. Thus, the new hierarchy has the advantages of both approaches while avoiding their drawbacks.

3.4 Parallel Ray Tracing

One other approach that is useful to speed up the algorithm is to use several processing elements running in parallel. Since the rays are traced independently from each other, the algorithm can be easily parallelized by distributing the computation related to different rays.

The simplest way to parallelize the algorithm is to partition the image space into several rectangular regions and to compute pixel values for each disjoint region in parallel. The image can be partitioned in several ways: the regions may be obtained by simply dividing the image space into equal sized rectangles. Each rectangular area is computed on different processing elements and the generated images are joined as a single image. A disadvantage of dividing the image space in this way is related to the distribution of computation load to different processing elements unevenly. That is, some processing elements may complete their tasks much earlier than others, since less objects are contained in the viewing volume associated with the region. The other approach to obtain the subimages divides the image space adaptively in order to distribute the tasks evenly. The subimages obtained in this way may be of different sizes but they should require approximately the same computation time so that no processing element is idle for a long time, while others are busy. In this case, we may achieve a speedup that is close to the number of processing elements running in parallel.

Another parallel ray tracing is essentially based on the spatial subdivision mentioned earlier [9]. The 3-D space containing the objects is subdivided into several disjoint volumes. The computation in each volume is carried out on a different processing element. The ray that travels through 3-D space to find an intersection passes from one processing element to another via messages. Each processing element contains the information about the volume assigned to it. A suitable architecture to accomplish this can be three dimensional array processor. In this architecture, each processing element is connected to 6 neighboring processing elements in order to pass messages which consist of information about the rays. On the other hand, hypercube architecture has potential to perform this task as 3-D array processor. A hypercube of

dimension d contains 2^d processors [35]. Assume the processors are labeled $0, 1, \dots, 2^d - 1$. Two processors i and j are directly connected iff the binary representations of i and j differ in exactly one bit. Its advantages in this context may come from its recursive definition and embedding the 3-D array processor architecture.

Both of the above ideas can be combined to reach at a more efficient utilization of processing elements [6,9,30]. The 3-D space is again decomposed into disjoint volumes which are assigned to different processing elements. In this case, several rays are traced independently in parallel by subdividing the image space as well. A more detailed parallel ray tracing can be seen in [6,9].

Müller has attempted to ray trace movies by distributing the frames to different Workstations connected through a network [26]. They generated a 5-minutes ray traced animation within 2 months without boring the users of the Workstations by efficiently using the network.

4. AN IMPLEMENTATION OF SPACE SUBDIVISION USING OCTREE

4.1 Overview

The space subdivision method can be broken up into two different steps; preprocessing and ray casting. We first build an auxiliary data structure that will be then used while traveling through the 3-D space. The auxiliary data structure will contain the information that will allow ray-environment intersections to be computed as quickly as possible. The 3-D space which contains all of the objects in the scene is divided into a hierarchical structure of cubic boxes aligned with the cartesian coordinate system. In the algorithm that uses space coherence, the only difference is in the intersection routine to find the first object hit by the ray, if there is any. The new intersection routine will be as follows:

- Find a point along a given ray in the first box it intersects. If the ray is originated inside the root box that encloses all of the boxes, the starting position is the point looked for. Otherwise, the ray that is originated outside the root is tested for intersection with this largest box. The point to be returned is the one a little further from the intersection point which is in the first box on the path of the ray. If the ray does not hit the root box, the next ray is shut from the viewpoint and no intersecting object is returned.
- Having found a point, find the box id that is associated with that point. This step may take most of the time for traveling through the boxes.

Therefore, the data structure should be designed so that we can access a box in the hierarchy for a given box name quickly.

- At this step, only the objects in the currently visited box are tested for intersection. Since each box contains few number of objects, we spent a little and approximately constant time in each box. If the ray hits more than one object in the same box, the nearest intersection point is returned. If the ray does not hit any object in this box, we move to the next box along the ray and perform intersection tests with the objects in this box. This process is repeated until an intersection is found or the ray leaves the root box.

Basically, there are three operations to be performed for the above algorithm. They are:

- Given a point in space, find the box and its data. Since space is divided dynamically and unevenly, this cannot be performed simply by indexing into a three-dimensional table of box references.
- Given a ray that originates within a given box, find the next intersected box, if there is. Otherwise, return a message signaling that it will leave the largest box.
- Given a box that describes a subspace of the scene, obtain a list of all objects whose surfaces intersect that box. Only the objects in this list will be tested for intersection with each ray that passes through the box.

The new algorithm after subdividing the 3-D space into cubic boxes will refer to the data base frequently. Therefore, it is important for the sake of the speed to organize the data structure so that it will be easy to access the information contained in it. We used a hierarchical data structure called octree to store this information. Octree structure consists of nodes that represent a subspace of the scene. A node is a leaf node if it is not subdivided any more. Each non-leaf node has eight children each of which describe a subspace of the parent node. The volumes described by children nodes are

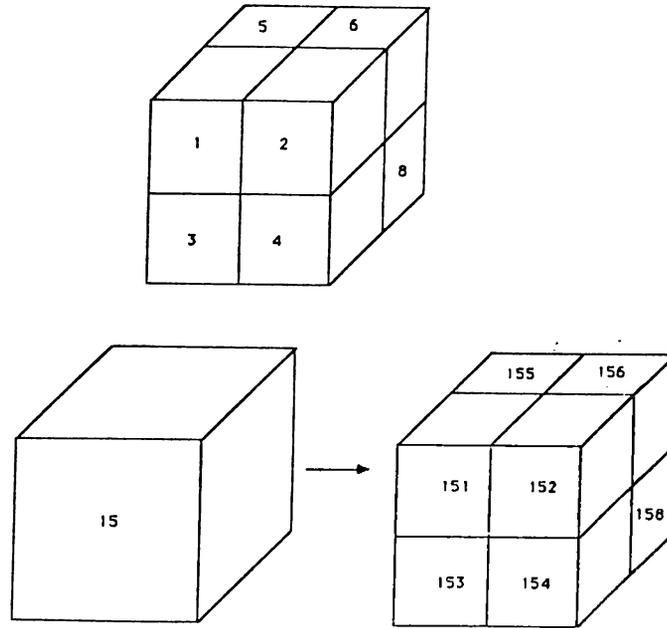


Figure 4.1: Naming the boxes.

disjoint and are equal in size. The root node which encloses all of the objects in the scene is labeled node 1. When we subdivide a node, it passes its name as a prefix to all its children, which are numbered 1 through 8, as shown in Figure 4.1. Thus the eight children of the root node are nodes 11 through 18. The children of node 15 are nodes 151 through 158, and so on. Now, we should answer the second question in the previous section. That is, how could we find the address of the box given its label ?

We can accomplish this in two extreme ways. In the first way, we could build a table with an entry for every possible node name that contains that node's address. Obviously, this possibility will require large amount of memory. This is due to the fact that not all possible nodes need to be created. Instead, the nodes of octree are created dynamically when needed. If we subdivide the root node twice, the maximum possible box name is 188. For example, we may not need to create nodes 151-158 if node 15 does not satisfy a subdivision criteria. On the other hand, this scheme would have the advantage of extreme speed in finding the address of a node for a given name. In the second way, we could construct the hierarchy by using linked lists. This time, each node would have eight pointers to its children and each time we would search the tree from beginning to find a node address of a given node

name. This scheme requires less memory than the first one but searching for a node slows down the operation and may come up with great overhead.

4.2 Octree Building and Storage

We mixed these two methods using a hashing scheme in the same way as Glassner did [12]. In this scheme, we have a hash table to hold pointers to a structure. This structure contains addresses of eight children of a subdivided node and the name of their parent node as shown in Figure 4.2. The construction of the data structure is as follows:

1. Find a cubic box, called root, which contains all of the objects in the scene. A box is defined by its center, size, a flag and a list of objects whose surfaces intersects with this box. Flag is set to zero for a leaf box.
2. If the root box contains objects more than a specified number, go to the next step to subdivide it into disjoint volumes. Otherwise, this is the leaf node. No further actions are to be taken for this box.
3. Using the name of current box that is to be subdivided, compute a hash function. We use a very simple hash function which is the node name modulo tablesize. Let **index** be the computed hash function.
4. The **index** is the location where we want to put the consecutive addresses of eight children and their parent name. If this location is not empty, the collided node names form a linked list. That is, by simply following the linked list, the new structure is appended to the end of this list. If this location does not point to any structure, it will contain the address of the new structure.
5. Now, for each generated child, find the center and the objects the surfaces of which intersect the box represented by the child. The size of the child node is the half of the parent's size. While determining the sizes of generated boxes, the minimum size of the boxes is stored in a

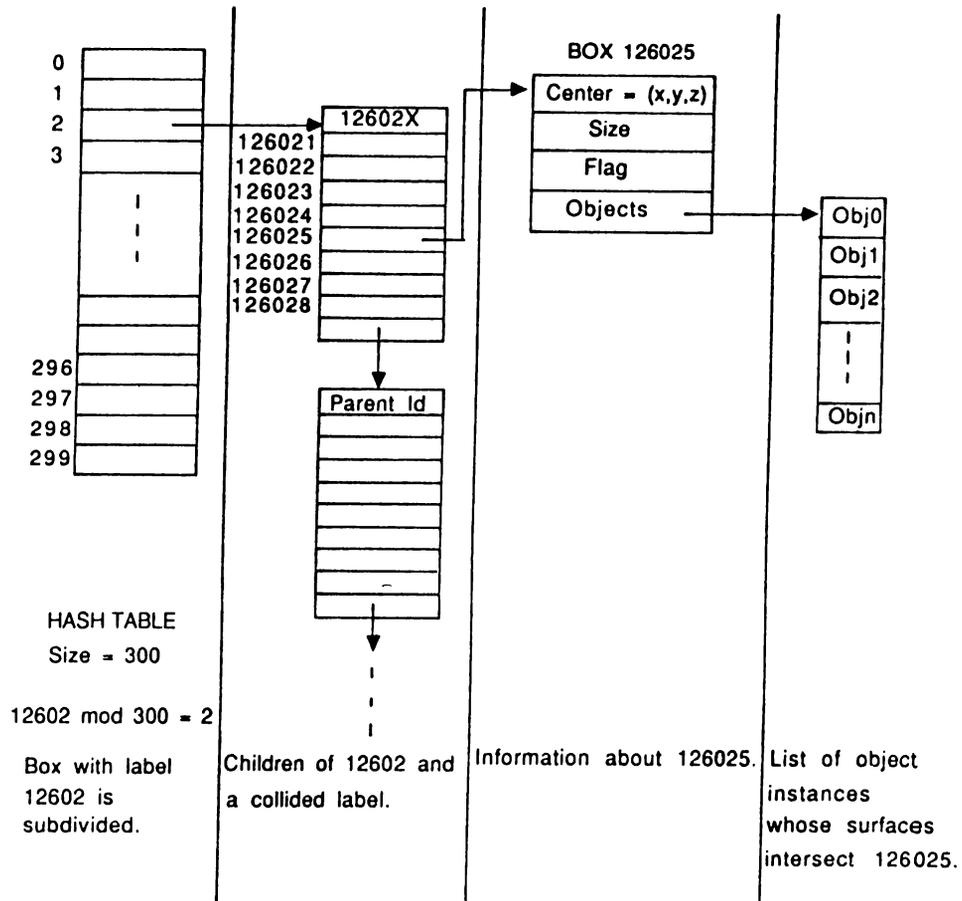


Figure 4.2: Hash table to store the generated boxes.

global variable called **Minlen**. **Minlen** is to be used later for moving from a box to another.

6. If the number of objects pointed by a child exceed a specified value, Its flag is set to a non-zero value. Each child node is subdivided as its parent if the flag is non-zero. That is, we go to the step 3 for each child with a non-zero flag.

4.3 Movement to the Next Box

Whenever no object in a box intersects the ray being traced, the next box, if any, on the ray's path should be determined and this time the intersection tests are carried out with the objects in this box. Movement to the next box continues until an intersection point is found, or until the ray leaves the root box. There are two issues involved in this operation: First, because the space is dynamically decomposed when we build the octree, we do not know how large (or small) any box in space is with the exception of the current one. That is, how far should we move in order to guarantee that we are in the next box, but not in another. Secondly, movement to the next box should be fast enough so that we do not lose what we gain by decomposing the 3-D space into boxes.

The essence of the box-movement algorithm is to find a point that is guaranteed to be in the next box whatever its size. This point is then used to derive a box name and the address of this box along with the information contained in it.

A point on a ray can be defined by a parameter t . The value of t increases as we move away from the origin, where t has the value 0. The parametric line equations give us the point $P = (x, y, z)$ corresponding to parameter t as below:

$$x = x_s + tx_r$$

$$y = y_s + ty_r$$

$$z = z_s + tz_r$$

where (x_s, y_s, z_s) is the starting position of the ray, (x_r, y_r, z_r) is the movement in each direction. Given a box definition and a ray that passes through the box, we can find the maximum value of t which may attain in this box. Let this parameter be t_{max} which will give the intersection point when the ray leaves the box. We can compute t_{max} by intersecting the ray with the six planes that bound the current box. Two of these intersections give us bounds on t for x -planes, two others for y -planes and the remaining two for z -planes. Since each plane is parallel to two of the three coordinate axes,

it is inexpensive to intersect a ray with one of these "simple" planes. Let the bounds be denoted by $tx_{min}, tx_{max}, ty_{min}, ty_{max}, tz_{min}, tz_{max}$. For example, tx_{min} is the minimum, tx_{max} is the maximum of t values that give the intersection points with x -planes. The other four are used similarly. Note that the points describing the intersections of the ray with the planes of the box may lie far outside the value of the box itself. But certainly some values of t will hold for all three ranges: these are the values of t inside the box. The intersection of three ranges ($tx_{min}..tx_{max}, ty_{min}..ty_{max}, tz_{min}..tz_{max}$) gives the values of t that the ray may take while it is inside the box. The value of t_{max} is the minimum of tx_{max}, ty_{max} and tz_{max} .

Now, we will use the variable **Minlen** to find a point in the next box. **Minlen** was the size of the smallest box. Having found the parameter t_{max} , we compute $P = (x, y, z)$ using above equation. We find the point within the next box along the ray by merely moving perpendicularly to the planes by a distance **Minlen** if their t values are equal to t_{max} . If the point of intersection is on an edge, that is, when two of the parameters are equal to t_{max} , we must travel perpendicularly to both faces sharing that edge, and similarly we must travel in three directions if t_{max} is on a corner of the box. We simply increment x, y or z component of the point P by **Minlen**, if the corresponding t value is equal to t_{max} .

Now, using this point, we should access the box and its information. We start by checking whether the point is in root or not. If the point is outside the root box, we return and report this. If it is in the root box that has label 1, we find the children of the root box by using the hash function. Next, we can decide which one of eight children contains this point. The same process is applied recursively for the child node that includes that point until a box that has a zero flag is reached. Flag was set to zero for the leaf boxes.

5. A RAY TRACER SYSTEM

A ray tracing system is designed and implemented in C programming language [24] on SUN¹ Workstations running under UNIX² operating system. The system has three major parts:

- Create a scene with objects provided by the system.
- Process the defined scene to obtain a realistic image that consists of RGB values.
- Display the generated image.

The system mainly can be used to create scenes containing 3-D objects and then find out the effect of light sources and the objects on each other. Each part is to be explained in detail in the following sections :

5.1 Defining the Scene

This is the first step in generating a realistic image. User is provided with several types of objects to model the scene. In addition to the definition of objects to be included in the scene, the user can give a number of parameters such as point light sources, viewpoint, origin of the scene etc. so that the processing is carried out with these parameters, otherwise, which would be assigned default values. The mentioned input can be given in two different ways :

¹SUN Workstation is a registered trademark of Sun Microsystems, Incorporated.

²UNIX is a registered trademark of AT&T Bell Laboratories.

- A text file can be prepared with definitions of objects and scene parameters using a text editor.
- The scene can be defined using an interactive tool that gives a friendly environment to the user.

5.1.1 Textual Input

When a person chooses the first way to define a scene, he must describe the scene according to a given syntax. UNIX tools LEX and YACC have been used to parse and process the user input [41]. The file consists of three kinds of information :

- Scene Parameters,
- Object Definitions,
- Material Properties.

Scene Parameters

Scene parameters are related to the things in the scene other than the objects. Each parameter declaration is started by a keyword that consists of uppercase letters. A set of values follows this keyword. The parameter declarations are given in the following form :

`VIEWPOINT x y z`

This is the place in three-dimensional space where the eye of the observer is located.

`ORIGIN x y z`

This is the point in three-dimensional space where the eye looks at.

UPVECTOR x y z

This is used to describe the orientation of the user. (x, y, z) is a normal vector that indicates the viewing direction.

RASTER width height

It is the resolution of the screen. "width" and "height" are the size of the screen in terms of number of pixels.

VIEWPORT width height

This gives the window size that the user can see.

RDEPTH n

This is used for the termination condition of the recursive call to the shading routine. It actually gives interreflections between objects. If n is 0, no reflections exist on the objects.

IMAGEFILE file name

The computed RGB values are written on the given file name.

LIGHT brightness

x y z

This is to define a point light source anywhere in the three-dimensional space. "brightness" gives the intensity of the point light source and ranges from 0 to 1. It is 0 if no illumination is done by the source. Next triple gives the position of the point light source.

Object Definitions

Next comes the definition of objects in the scene. There are five types of objects definable by the system. They are sphere, superquadric, triangle, rectangle and box. Their textual descriptions are as follows :

Each object description is started by a keyword followed by a surface number and the parameters to define the object. Surface number is an integer that refers to a surface definition used for shading computations.

Sphere has two parameters which are center and radius of the sphere.

```
SPHERE  surface-number
radius
x y z   /* center of the sphere */
```

Note that comments can be written as in the C language to make the description more understandable.

A box can be defined by its center and the size information from the center. Boxes are always aligned with the coordinate axes.

```
BOX      surface-number
xc yc zc /* center of the box */
xs ys zs /* size of the box   */
```

Superquadrics in the context of this system can be defined as boxes whose corners are rounded. Therefore, a superquadric is defined similarly to a box with an additional parameter called "Power". "Power" gives the degree of roundness of the corners. Its definition is as follows :

```
SUPERQUADRIC  surface-number
power
xc yc zc  /* center of the superquadric */
xs ys zs  /* size as in box definition   */
```

A triangle in three dimensional space is defined by its three corners. the points of the corners should be given in the counterclockwise direction that is very important in shading computations. The syntax for a triangle declaration is :

```
TRIANGLE    surface-number
x1 y1 z1
x2 y2 z2
x3 y3 z3
```

Similar to a triangle definition, a rectangle is defined by its corners. Again, the corners should be given in counterclockwise direction. Its syntax is :

```
RECTANGLE   surface-number
x1 y1 z1
x2 y2 z2
x3 y3 z3
x4 y4 z4
```

Material Properties

A surface description that is referred by the objects serves the shading computations in all steps. A surface description can be given as follows :

```
SURFACE     surface-number
r1 g1 b1    /* ambient color      */
r2 g2 b2    /* diffuse reflection */
r3 g3 b3    /* specular reflection */
k           /* constant related to the surface property */
r           /* reflectivity coefficient */
t           /* transparency coefficient */
```

5.1.2 Interactive Tool

The other way to specify a scene is to use an interactive tool. The user can give the description of a scene using mouse, menus etc. in a user friendly environment. This tool projects the three-dimensional world onto the two-dimensional screen to provide user with an easy user interface when entering three-dimensional points.

Using this tool, the scene description mainly involves selecting one of the object types and specifying the size and other parameters related to the selected object type by the mouse, panel and other windowing system elements. A more detailed information on the interactive tool is given in the appendix as the user's manual.

After completing the description of the scene, the system converts it into the format given in the previous section and writes the textual description on a text file. Therefore, this interactive tool is nothing but a shell that generates the description on a text file as an output. The textual description of a scene is useful in the sense that the ray tracer can be portable to any computer system.

B-Spline Surfaces

The other advantage of the interactive tool is that user can generate free-form surfaces other than five primitive object types. A free-form surface can be created and placed into an appropriate location in the scene. B-Spline method is used by the system to find out the surface to be included in the scene description. The surface generated is then triangulized and written in the known format on the output file as a collection of triangle primitives.

Since objects with complex shapes occur frequently in our three-dimensional world, special techniques to model them properly are needed [31]. Although these objects can be approximated with arbitrarily fine precision as plane-faced polyhedra, such representations are bulky and intractable. For example, a polyhedral approximation of a hat might contain 1000 faces and would be

difficult to generate and to modify. We need a more direct representation of shapes, easy both to the computer and to the person trying to manipulate the shapes. Bézier and B-Spline are the two methods frequently used to generate curves and surfaces of 3-D. They are similar to each other in that a set of blending functions is used to combine the effects of the control points. The key difference lies in the formulation of the blending functions [3,18,31].

5.2 Processing: Ray Tracer

This part accepts a textual scene description as input that was explained in the previous section and it generates an image containing several optical effects for the sake of realism using ray tracing algorithm. There are three major data structures used by this module. They are used to store the objects, the surfaces and the light point light sources in the scene. The objects are stored in an array that has elements of the following structure:

```
OBJECT TYPE
OBJECT ID
SURFACE NUMBER
POINTER TO AN OBJECT INSTANCE
```

OBJECT TYPE indicates one of the five primitive objects. For example, OBJECT TYPE for a sphere is 0. It is used to call intersection and normal routines related to OBJECT TYPE. OBJECT ID indicates the object instance in the scene. It is used to access the object variables in intersection and normal routines. SURFACE NUMBER refers to a surface definition that gives the object material property. The last entry is an address to an object instance. For example, the address in this entry may contain a sphere definition as below:

```
RADIUS
X Y Z
```

The surface definition are stored in an array of the following structure:

```
AR AG AB      /* Ambient color components */
DR DG DB      /* Diffuse color components */
SR SG SB      /* Specular color components */
COEF          /* Surface coefficient      */
REFL          /* Reflection coefficient   */
TRANSP        /* Transparency coefficient  */
```

Point light sources are simply stored in an array of lights. Each light is defined by

```
BRIGHTNESS
X Y Z
```

Each object type definable by the system has its own intersection function. This gives a great flexibility for adding new objects to the system. That is, if a new object type is to be added to the system, the following should be available for the new object type:

- Object Definition
- A function that allocates space for an instance of the new object type.
- A function that finds out the intersection point on the surface with a ray. This is used during intersection tests.
- A function that returns the normal vector at a given point on the object surface. This is used for shading computations.

Additionally, the intersection calculations become more efficient since each function is implemented for a specific object type.

The shading model used by the system is similar to the one given in the second chapter. Ambient color that is stored in surface definition is the initial color for the point being shaded. The diffuse and specular reflections are calculated by using the entries in the surface structure. The level of reflection is the terminating criterion for the recursive call when the reflected

ray hits another object other than light source. Transparency of an object is simulated by the given calculation model. That is, the ray is divided into two components and one of them continues traveling through the transparent object until it hits another object at its back. After calculating the intensity at the hit point, the intensity at the point of the transparent object is found out by the given formula. In addition to these optical effects, shadows that give very strong depth cues may exist if the light is blocked by opaque or semi-transparent objects.

This part of the system has three different versions. The first version of the system is based on the naive ray tracing algorithm. That is, all objects in the scene are tested to find the first intersection with a traced ray. As it is very obvious, the time spent increases with the number of objects in the scene. For example, when we have 10,000 objects in the scene, processing may take even days on a mini computer.

In the second version of the ray tracing system, the intersection tests are not carried out with all objects in the scene but the spatial coherence technique is used to perform intersection tests only for the objects that are on the path of the ray. Therefore, the space containing the objects is subdivided using the octree representation. The second version of the program is capable of generating the images much faster than the first version that does not utilize the space coherence.

The last version is the parallel ray tracing algorithm. It partitions the image space into four equal sized rectangles and generates pixel values for each of them on a different file. It is just a simulation of the parallel algorithm using UNIX's "fork" and "wait" system calls [42]. No actual speedup is achieved, since only one processor is used.

5.3 Displaying the Image

We compute a triple (RGB) for each pixel of the image after tracing a ray. When we compute RGB values in shading routine, we assume a linear intensity response. That is, pixel of a value of 127, 127, 127 has the half intensity

of a pixel value of 255, 255, 255. However, the response of typical video color monitors and of the human visual system is non-linear. Thus, displaying of images in a linear format results in effective intensity quantization at a much lower resolution than the available 256 resolution per color. That is, the true colors will not be perceived by the human eye, because of the non-linearity in the monitor. Therefore, it is necessary to correct the computed values so that the generated picture appears more realistic to a human observer.

A function called gamma correction is used for this purpose [16]. It is an exponential function of the form:

$$lookupvalue = intensity^{1.0/gamma}$$

Gamma represents the nonlinearity of the monitor. Generally monitors have a gamma value that is in the range 2.0 to 3.0. If gamma is equal to 1, the device is a linear one. An incorrect value results in incorrect image contrast and chromaticity shifts. If the gamma is too small, the contrast is increased and the colors approach to the primaries.

The RGB values should be corrected by the above function before the image is displayed or stored to a file.

5.4 Examples and Timing Results

In this section, several images generated by our system are presented in Figures 5.1, 5.2, 5.3, 5.4, 5.5, 5.6, 5.7, 5.8. We also compare the time spent in the first and second version of the algorithm for some images.

Table 1 contains the clock timings for rendering three images given in Figures 5.1, 5.2, 5.3. Actually, there may be several other influences on the rendering time other than the complexity of the algorithm. These are the programming style, the code optimization, the processor speed, etc. As it is clear from these measurements, when the number of objects in the scene increases, the ratio of the naive technique to the fast one gets larger and larger. This is due to the fact that a great amount of time is wasted for the intersection calculations in the naive ray tracing system. The ratio approaches

| Figures | No. of Objects in the scene | Naive (in minutes) | Fast |
|---------|--------------------------------|-----------------------|-------|
| 5.1 | 5 | 05:06 | 08:50 |
| 5.2 | 50 | 54:26 | 25:40 |
| 5.3 | 200 | 123:20 | 40:52 |

Table 5.1: Timing results of figures 5.1, 5.2, 5.3.

| No. of Objects in boxes | Time (in minutes) |
|----------------------------|----------------------|
| 2 | 22:21 |
| 6 | 17:52 |
| 8 | 28:36 |

Table 5.2: Timing results of figure 5.4.

one for the scenes containing few number of objects [21]. In higher speed models, we note that the timings are dependent on the criteria used to subdivide the space. For example, when the space is divided into very small boxes that contain only one object, the overhead for traveling through the boxes may approach to the time saved for intersection tests. That is, the ray may frequently pass through many empty volumes wasting a considerable amount of time. Table 2 shows timings for the image in Figure 5.4 generated using the octree auxiliary data structure for three different terminating criteria values.



Figure 5.1: Four spheres and a triangle.



Figure 5.2: Fifty spheres.



Figure 5.3: Two hundred spheres.



Figure 5.4: Twenty four spheres.

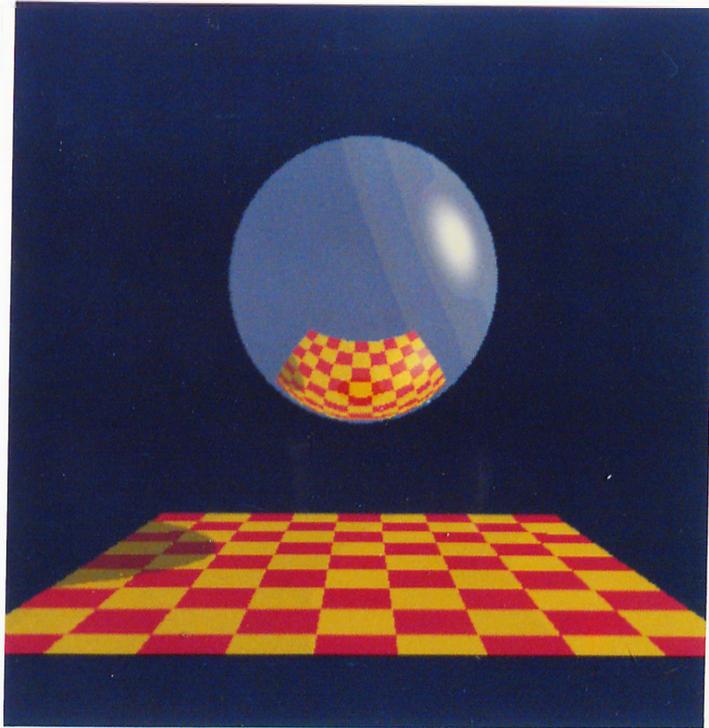


Figure 5.5: Sphere above chessboard.



Figure 5.6: Shield and a sphere.

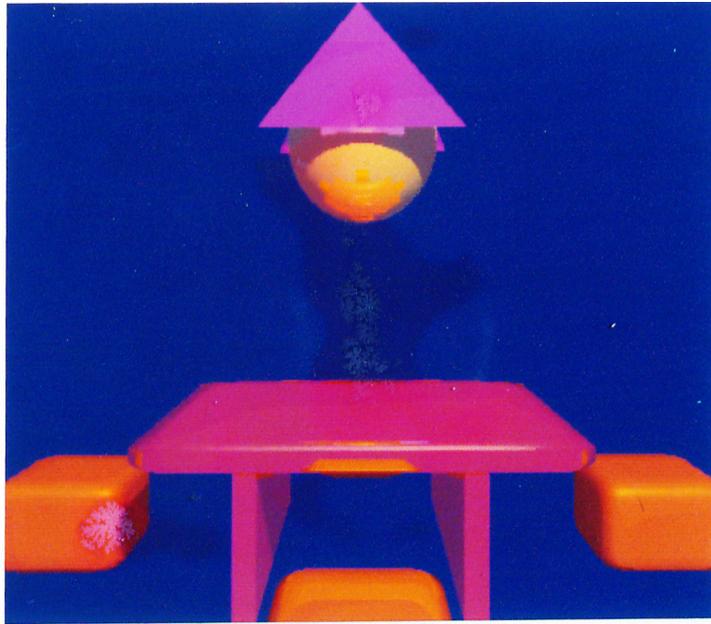


Figure 5.7: Boxes and superquadrics.

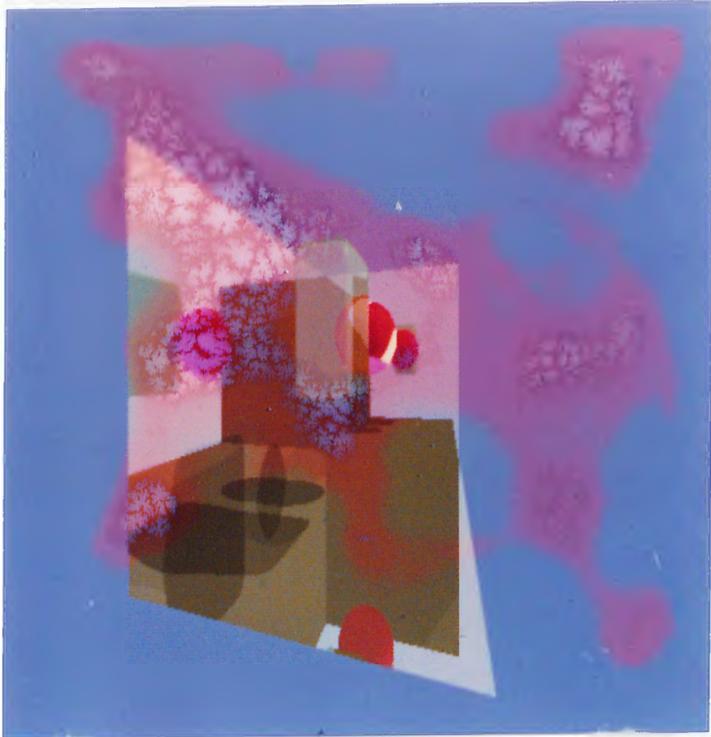


Figure 5.8: Mirrors and reflections.

6. CONCLUSION

Computer generated images that appear realistic to a human observer have been one of the most important goals in Computer Graphics. Ray tracing algorithm with this respect is the most popular method for realistic image synthesis. It is in the class of image generation algorithms, called global shading, that provide the most realistic images by considering the optical effects such as shadowing and reflection from the surfaces in the environment. However, it requires a tremendous amount of time to generate an image. Several methods have been developed to overcome the time problem [6,9,10,11,12,13,23,28,29]. In this thesis, we investigated these methods and used two of them, namely space subdivision and parallelism, in the implementation of the ray tracing system. The space subdivision method saves considerable amount of time when the scene is complex. But the speedup achieved is still not enough when many images are to be generated. The parallel version of the space subdivision method may be the best solution for the time problem. As the first conclusion, the parallelism is essential for the interactive realistic image generation.

Another method in the global shading class has been introduced later than ray tracing as radiosity [7,15,20]. This method can simulate the global illumination effect more accurately than ray tracing can do. Although there has been several attempts to obtain more detailed images using ray tracing as in [1,5,8,22], no one could consider the interaction between diffuse surfaces. Another advantage of radiosity method is that the resultant surface intensities are independent of the viewer position. This allows efficient rendering of dynamic sequences. As the next conclusion, ray tracing should utilize some ideas in this method, namely radiosity to increase the accuracy in simulating

the global illumination effects.

Finally, we should also take into account the texture mapping that is used to cover over a surface with texture [4], in order to provide more detailed images. Texture mapping is basically the method of wallpapering the polygons in the scene. One of the future directions in realistic image generation is to develop methods for ray tracing texture mapped surfaces in a reasonable amount of time.

REFERENCES

- [1] Amanatides, J., "Ray Tracing with Cones," *ACM Computer Graphics*, Vol. 18, No. 3, (July 1984), pp. 129-135.
- [2] Amanatides, J., "Realism in Computer Graphics : A Survey," *IEEE CG & A*, (January 1987), pp. 44-56.
- [3] Barsky, B. A., "A Description and Evaluation of Various 3-D Models," *IEEE CG&A*, (January 1984).
- [4] Blinn, J. F. and Newell, M. E., "Texture and Reflection in Computer Generated Images," *Communications of the ACM*, Vol. 19, No. 10, (October 1976), pp. 542-547.
- [5] Bouville, C., Brusq, R., Dubois, J. L., Marchal, I., "Generating High Quality Pictures by Ray Tracing", *Computer Graphics Forum*, 4 (1985) pp. 87-99.
- [6] Cleary, J. G., Wyvill, B. M., Birtwistle, G. M., Vatti, R., "Multiprocessor Ray Tracing," *Computer Graphics Forum*, 5 (1986) pp. 3-12.
- [7] Cohen, M. F. and Greenberg, D. P., "A Radiosity Solution For Complex Environments," *ACM Computer Graphics (Proc. SIGGRAPH)*, Vol. 19, No. 3, (July 1985), pp. 31-40.
- [8] Cook, R. L., Porter, T., Carpenter, L., "Distributed Ray Tracing," *ACM Computer Graphics*, Vol. 18, No. 3, (July 1984), pp. 129-135.
- [9] Dippé, M. and Swensen, J., "An adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *ACM Computer Graphics*, Vol. 18, No. 3, (July 1984), pp. 149-158.

- [10] Fujimoto, A., Tanaka, T. and Iwata, K., "ARTS: Accelerated Ray-Tracing System," *IEEE CG&A*, (April 1985), pp. 16-26.
- [11] Geoff, W., "Space Division for Ray tracing in CSG," *IEEE CG&A*, (April 1986), pp. 28-34.
- [12] Glassner, A. S., "Space Subdivision for Fast Ray Tracing," *IEEE CG&A*, (Oct. 1984), pp. 15-22.
- [13] Glassner, A. S., "Spacetime Ray Tracing for Animation," *IEEE CG&A*, (March 1988), pp. 60-70.
- [14] Goldsmith, J., Salmon, J., " Automatic Creation of Object Hierarchies for Ray Tracing," *IEEE CG&A*, (May 1987), pp. 14-20.
- [15] Goral, C. M., Torrance, K. E., Greenberg D. P. and Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces," *ACM Computer Graphics*, (1984), pp. 213-222.
- [16] Hall, R., "Color Reproduction and Illumination Models," *Techniques for Computer Graphics*, Springer-Verlag, (1987), pp. 195-238.
- [17] Hanrahan, P., "Using Caching and Breadth-First Search to Speed up," *Proceedings : Graphics and Vision Interface '86* , Canadian Information Society, Toronto, (1986), pp. 56-61.
- [18] Hearn, D. and Baker, M. P., *Computer Graphics*, Printice-Hall, Inc., Englewood Cliffs, NJ (1986).
- [19] Heckbert, P. S., Hanrahan, P., "Beam Tracing Polygonal Objects," *ACM Computer Graphics*, Vol. 18, No. 3 (July 1984), pp. 119-127.
- [20] Immel, D. S., Cohen, M. F., Greenberg, D. P., "A Radiosity Method For Non-Diffuse Environments," *ACM Computer Graphics (Proc. SIG-GRAPH)*, (1986), pp. 133-142.
- [21] İşler, V. and Özgüç, B., "Ray Tracing Geometrical Models," *Proceedings of the Fourth International Symposium on Computer and Information Sciences*, Çeşme, Turkey (October 1989, to appear).

- [22] Kajiya, J. T. and Herzen, B. P. V., "Ray Tracing Volume Densities," *ACM Computer Graphics*, Vol. 18, No. 3, (July 1984), pp. 165-174.
- [23] Kaplan, M. R., "The Use of Spatial Coherence in Ray Tracing," *Techniques for Computer Graphics*, Springer-Verlag, pp. 173-193 (1987).
- [24] Kernighan, B. W., Ritchie, D. M., *The C Programming Language*, Prentice Hall, Inc., Englewood Cliffs, NJ (1978).
- [25] Kuchkuda, R., "An Introduction to Ray Tracing," *Theoretical Foundations of Computer Graphics and CAD. Edited by R.A. Earnshaw*, NATO ASI Series. Vol. F40, Springer-Verlag Berlin Heidelberg 1988, pp 1039-1060.
- [26] Leister, W., Maus, T., Müller, H., Neidecker, B., Schmitt, A., Achim, S., "Occursus Cum Novo : Computer Animation by Raytracing in a Network," Technical Report, Universität Karlsruhe , Fakultät Für Informatik, November 1987.
- [27] Levner, G., Tassinari, P. and Marini, D. "A Simple, General Method for Ray Tracing Bicubic Surfaces," *Theoretical Foundations of Computer Graphics and CAD. Edited by R.A. Earnshaw*, NATO ASI Series. Vol. F40, Springer-Verlag Berlin Heidelberg 1988, pp 805-820.
- [28] Müller, H., "Ray Tracing Complex Scenes By Grids," Technical Report, Universität Karlsruhe , Fakultät Für Informatik, December 1985.
- [29] Müller, H. and Hagen, H., "Trading Speed Against Space in Ray Tracing Free Form Surfaces," Technical Report, Universität Karlsruhe , Fakultät Für Informatik, May 1986.
- [30] Nemoto, K. and Omachi, T., "An Adaptive Subdivision by Sliding Boundary Surfaces," *Proceedings : Graphics and Vision Interface'86*, Canadian Information Society, Toronto, (1986), pp. 43-48.
- [31] Newman, W. and Sproull, R., *Principles of Interactive Computer Graphics*, McGraw-Hill, (1981), second edition.

- [32] Özgüç, B., "Thoughts on User Interface Design for Multi Window Environments," *Proceedings of the Second International Symposium on Computer and Information Sciences*, Istanbul, Turkey (1987), pp. 477-488.
- [33] Phong, B. T., "Illumination for Computer Generated Pictures," *Communication of the ACM*, Vol. 18, No. 6, (June 1975), pp. 311-317.
- [34] Pulleyblank, R., and Kapenga, J., "The feasibility of a VLSI Chip for Ray Tracing Bicubic Patches," *IEEE CG&A*, (March 1987), pp. 33-44.
- [35] Sahni, S. and Ranka., S., *Hypercube Algorithms for Image Processing and Pattern Recognition*, (to be published), (1989).
- [36] Samet, H. and Webber, R. E., "Hierarchical Data Structures and Algorithms for Computer Graphics , Part I: Fundamentals," *IEEE CG&A*, (May 1988), pp. 48-68.
- [37] Samet, H. and Webber, R. E., "Hierarchical Data Structures and Algorithms for Computer Graphics , Part II: Applications," *IEEE CG&A*, (July 1988), pp. 59-75.
- [38] Schmitt, A., Müller, H. and Leister, W., "Ray Tracing Algorithms - Theory and Practice," *Theoretical Foundations of Computer Graphics and CAD. Edited by R.A. Earnshaw*, NATO ASI Series. Vol. F40, Springer-Verlag Berlin Heidelberg 1988, pp 997-1030.
- [39] Sederberg, T. W. and Anderson, D. C., "Ray Tracing of Steiner Pathces," *ACM Computer Graphics*, Vol. 18, No. 3, (july 1984), pp. 159-164.
- [40] Sun Microsystems, *Sun View Programmer's Guide*, Mountain View, CA (1986).
- [41] Sun Microsystems, *Programming Utilities for the Sun Workstation*, Mountain View, CA (1986).
- [42] Sun Microsystems, *UNIX Interface Reference Manual*, Mountain View, CA (1986).

- [43] Whitted, T., "An Improved Illumination Model for Shaded Display,"
Communications of the ACM, 23 (June 1980), pp. 343-349.

A. THE USER'S MANUAL

This system is mainly used to model 3-D scenes, to find the interaction of objects with the light sources and to display the resultant image that appears realistic.

To provide the users with a friendly environment, SUNVIEW has been used for creating panels, menus, buttons, etc. [40]. More information about multi window environment can be found in [32]. The modules that comprise the system and the user interface are given in detail in the following two sections.

A.1 The Modules

The system can be broken up into two different subsystems: the interactive tool and the ray tracer. The first one is used to model a 3-D scene as an input to the ray tracer and to display the realistic image that contains some optical effects. The second subsystem generates the realistic image that is to be displayed by the interactive tool. Each subsystem consists of several modules and header files written in the C programming language. The description and comments about the modules are given at the beginning of the files containing the modules. The modules are compiled and linked using the UNIX's *makefile* facility.

The interactive tool contains the following files :

`coarse.c` `conls.c` `disp.c` `dither.c`

| | | | |
|------------|---------|-----------|-----------|
| imsl.c | model.c | object.c | parmenu.c |
| pmain.c | pnot.c | surface.c | trans.c |
| writedef.c | | | |

| | | | |
|---------|----------|--------|---------|
| pcons.h | pdef.h | pext.h | pfunc.h |
| plib.h | pthead.h | | |

| | | | |
|----------|------------|------------|--------------|
| box.icon | eye.icon | light.icon | psource.icon |
| rec.icon | sph.icon | spl.icon | sup.icon |
| tri.icon | viewp.icon | | |

The ray tracer contains the following files :

| | | | |
|----------------|---------------|--------------|----------|
| box.c | createstr.c | fintersect.c | fmain.c |
| input_lex1.c | input_yacc1.c | intersect.c | light.c |
| light2.c | main.c | nextbox.c | objbox.c |
| objend.c | outputp.c | pfmain.c | pmain.c |
| poutputp.c | raymath.c | rectangle.c | setup.c |
| shade.c | shade2.c | sphere.c | stubs.c |
| superquadric.c | triangle.c | viewing.c | |

| | | | |
|-------------|--------------|------------|-------------|
| constants.h | constants1.h | funcdefs.h | funcdefs1.h |
| globalvar.h | globalvar1.h | maindecl.h | maindecl1.h |
| typedefs.h | typedefs1.h | y.tab.h | |

| | | | |
|--------------|---------------|-------------|--------------|
| input_yacc.y | input_yacc1.y | input_lex.l | input_lex1.l |
|--------------|---------------|-------------|--------------|

A.2 The Interface

The user interface for the system consists of four subwindows. The first one is a panel that contains several buttons, two text items to enter full path name of a file and a message item to give warnings and messages. The second one is

a canvas to draw the scene elements onto the screen. The third subwindow is used for different purposes at different times. For example, eight icons appear on this window when the "Objects" menu item under "Define Scene" button is selected. After selecting one of these icons, this subwindow then displays various parameters related to the selected icon that can be modified. The fourth subwindow is used to enter color information about surfaces. Below, the way how these subwindows are used is given in detail.

There are eight buttons in the main panel as shown in Figure A.1. Some of them cause menus to be popped up when selected. The function of each button in the panel is explained in the following sections.

- **GENERAL** : This button is used to select the actions that are generic in the sense that they can be selected at any time while using the system. Currently, one menu item is contained in the menu related to this button. More can be appended to provide user with other facilities. The function of this menu item is given below :
 - **Operating System**: This creates a window and enables user to work at the operating system level.
- **DEFINE SCENE**: This button is used to define the scene to be ray traced. A scene can be defined by one or more objects with their material properties and scene parameters such as light sources, viewing direction, etc. Objects and scene parameters can be given by selecting "Objects" menu item, while material properties can be entered by "Surface" menu item. Each is going to be described below.
 - **Objects**: After selecting this menu item, the panel on the left (third subwindow) contains 8 choices indicated by icons as in Figure A.2. At the same time, three dimensional coordinate axes and y-z plane as a reference is drawn on the canvas where 3-D points are given for the objects, light sources and view point. In other words, 3-D space is projected on the 2-D screen to facilitate for 3-D input. The first six icons are to define objects to be included in the scene while the last two icons are to give other scene parameters.

The object definition starts by selecting the corresponding icon and inputting the necessary size and position information about the selected object. Every object should refer to a number called "Surface number" that must be defined to denote the material properties of the object referring to it.

A point in 3-D space is entered by first pointing at x-z plane and pressing a mouse button at that point, then dragging either up or down to enter the y-coordinate. In other words, first x,z components are given, next the y component of the point is denoted by releasing the mouse button. The y-component is the difference between the heights of releasing point and pressing point.

Each icon has a function that will be described below:

- * Triangle : Three points are given successively for the corners of the triangular plane in space. Before that, the referred surface number should be selected from the panel in the third subwindow. To put other objects into scene after one or more triangular planes are defined, "Objects" menu item should be selected.
- * Rectangle : Rectangular surface is defined similarly, but with four points for the corners of the rectangle.
- * Sphere : First, the center is given as a 3-D point. Next, the radius is entered by pressing a mouse button for a second time. The distance between center and the last position gives the radius of the sphere.
- * Superquadric : Superquadric in the context of this system is a box with rounded corners. The center is entered simply as a 3-D point. The size information is given by dragging the mouse in x,y and z directions. A box that encloses the superquadric is drawn when the button is released. "Power" is the parameter that denotes the roundness of the superquadric. It shows up in the panel of the third subwindow and it takes a default value if the user does not specify its value.
- * Box : A box is defined by its center and size in three directions, namely x,y, and z. The surface associated with the box should

be given as in other object definitions.

- * **B-Spline** : B-Spline surfaces are defined by a set of control points and the order of continuity that gives the smoothness of the generated surface. Before entering the control points, the user should specify the order of continuity, the number of control and generated points in the panel of the third subwindow. Otherwise, default values shown in the panel are assumed. The control points are entered by selecting 3-D points that form a network of points in space. After entering the complete network of points, the user should wait for a while until the surface is obtained and drawn as wire-frame. User can proceed to define the rest of scene as in the previous object definitions.
- * **View Point** : Each scene definition contains exactly one viewer position. It is selected as if a 3-D point is entered. When the view point is given, two other parameters, namely "Recursive Depth" and "Raster Size" are specified in the panel of the third subwindow. Recursive depth is used for terminating criterion which determines the level of reflections on the surfaces. Raster size is the size of the screen where the image is to be displayed.
- * **Light Source** : A scene may contain a number of light sources to illuminate the environment. A light source has a position and brightness. Its position is given as 3-D point, and brightness is specified in the panel of the third subwindow.
- **Surface** : Each object refers to a surface by its number. Surface definition gives the material property of the object with its color. Surface definition consists of the following information.
 - * Ambient Color
 - * Diffuse Color
 - * Specular Color
 - * Surface Coefficient
 - * Reflection

* Transparency

The color values are given using the panel in the third subwindow. There are 16 predefined colors displayed in the small squares. User can select one of these and load into ambient, diffuse or specular boxes that are above squares. To select a color, the square is pointed and the mouse button is pressed. To load a color from current selected square, point at one of ambient, specular or diffuse boxes and press a mouse button. The color of a selected square can be changed by sliding the red, green or blue components. The background color is taken from the last square in the panel.

The last three parameters are entered from the panel in the third subwindow. They assume default values for the parameters, if no value is specified. For more information about these parameters, refer to chapter 2.

- PROCESS : This button is used to process the defined scene in three different ways. The menu items related to this button are explained below:
 - Naive : This choice calls the naive version of the ray tracing algorithm. That is, all of the objects in the scene are compared to find the first intersection point, if any. Naturally, it is very slow when the number of objects is large. It is included into the system to be able to compare the timing results with other versions.
 - Fast : This calls the fast algorithm that uses spatial coherence. The space is subdivided into disjoint volumes and only the objects on the path of the ray are tested for intersection. The reader is referred to the related chapter for other details.
 - Parallel : This choice calls the parallel version of the ray tracing algorithm. It only simulates the parallelism using some operating system functions. It divides the image space into four equal sized rectangles and generates pixel values for each of them on a different file.
- DISPLAY : This is used to display the generated image on the screen. Due to the limited size of the color table in our system, we could not

manage to display an image with more than 256 different color values. For instance, a generated image on the average contains 4000 different RGB values depending on the object colors in the scene. Therefore, we had to map the 4000 different RGB values to the 256 available colors reducing the quality of the display. There are several ways to accomplish this:

- Scan : One way to do this is to map several close RGB values to one entry of the table. A hashing function is used to find an entry for an RGB value in the color table. The RGB values at the calculated entry are compared to the ones of the pixel being scanned. If they are almost the same, the new pixel takes its color value from this entry by simply pointing to it. Otherwise, an empty entry in the color table, if available, is used for this new value.
- Dither : This choice calls the dither procedure that reduces the number of colors to at most 256 through multiplying the pixel values by a dither matrix.
- Coarse : This is similar to the previous one, but it forms boundaries around objects. It is slower than dither procedure.
- Gray : This is called to generate a gray level image from the color one. It can be useful on the systems with limited colormap size.
- IMAGE : This button is used to save or to load the displayed image in different formats which are machine dependent.
 - Save Raster : This saves the canvas in raster format that is usually used on SUN Workstations.
 - Load Raster : This loads the image saved in raster format. It is very fast compared to other display functions.
 - Save Targa : This is used to save in a format that can be displayed on the machine with "Targa" board.
- CLEAR : To clear the canvas.
- QUIT : To exit from the ray tracing system.

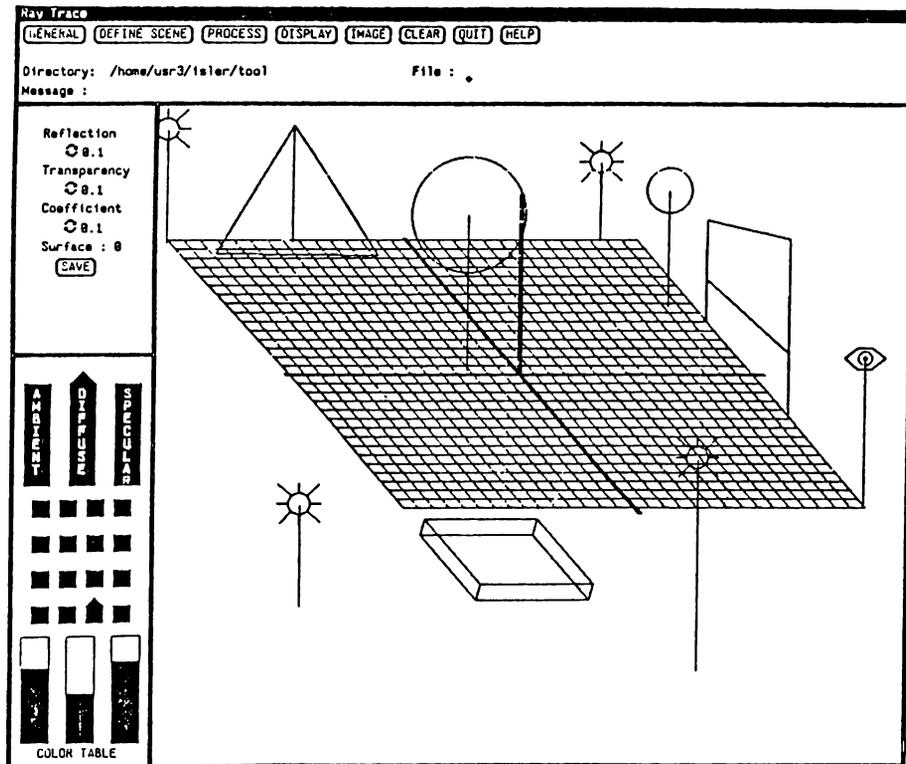


Figure A.1: User interface of the system.

- **HELP** : To display help texts about the system.

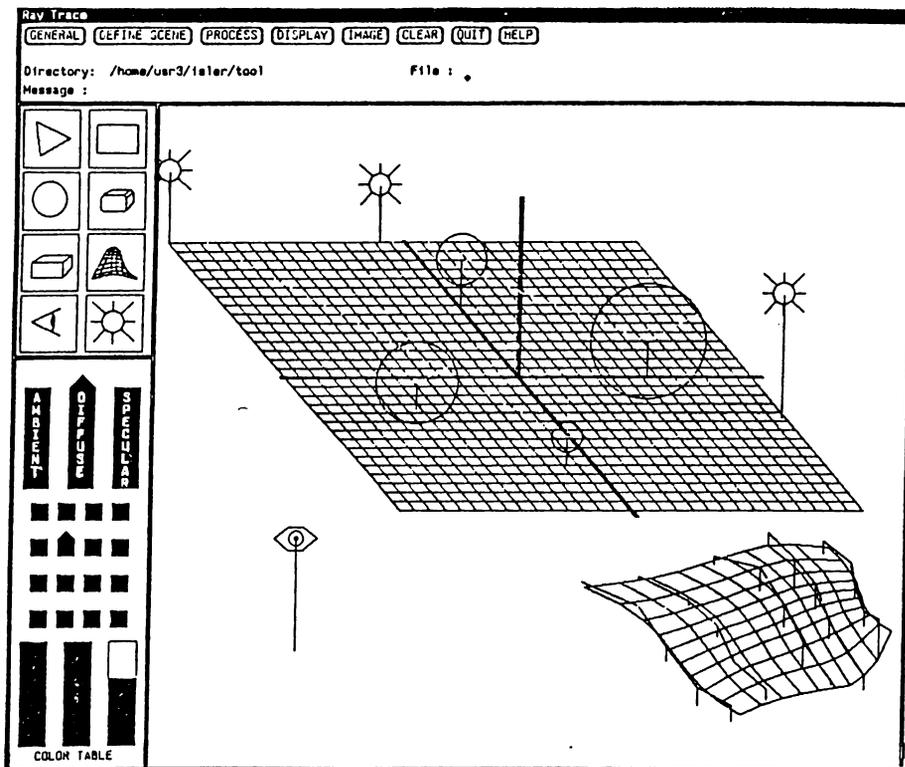


Figure A.2: Selecting objects.