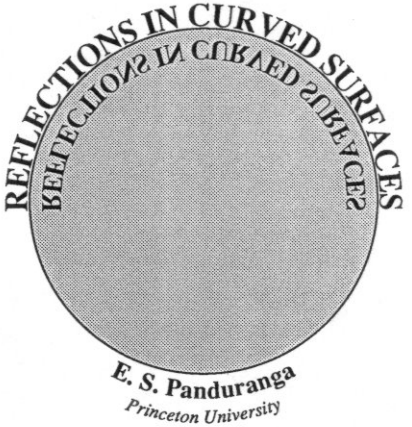
REFLECTIONS IN CURVED SURFACES

E. S. Panduranga

CS-TR-122-87

October 1987



A DISSERTATION
PRESENTED TO THE
FACULTY OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE BY THE
DEPARTMENT OF
COMPUTER SCIENCE

OCTOBER 1987

Dedicated To

My Beloved Mother

E.S.Lokamatha

नहि ज्ञानेन सदृशम् पवित्रमिह विद्यते ।

Nahi jn̈ânâna sadṛisham pavithramiha vidyathâ.
There is nothing in this world as holy as knowledge.
— *Bhagavadgeethä*

To the ocean of knowledge I hope I have added but a small drop.
— E.S.P. (1987)

ACKNOWLEDGMENTS

Words are not enough to express my gratitude to Prof. David Dobkin. Without his constant advice and involvement in the project, this thesis could not have happened. Weekly discussions with him were a great source of knowledge and enthusiasm. He made himself always available at the times of need. He has been my advisor, instructor, friend and philosopher.

I am grateful to Prof. Bernard Chazelle and Prof. Kai Li, who were the reviewers for this dissertation. Each made numerous suggestions that have enhanced the quality of this work. In addition, during the course of my research, I have benefited from timely conversations with each of them and with Prof. William Thurston of the Mathematics Department. I should specially thank Silvio Levy, also of the Mathematics Department, for his help with contour tracing and for fruitful discussions on various topics.

I am indebted to Vijaya Venkatesh and Krishen Kak for non-technical review of this dissertation. Both of them did an excellent job.

I wish to acknowledge the following people for their contributions to the enhancement of this dissertation: Abdou Youssef, William Lin, Arvin Park, K. Balasubramanian, Michael Laszlo, Deborah Silver, Fook-Luen Heng, Eleftherios Koutsofios, Kriton Kyrimis, Steve Friedman, Toshio Nakatani, Diane Souvaine (all my colleagues at Princeton University); G. Baskaran and Prof. Frank Shoemaker of the Physics Department and Prof. Myles of the Electrical Engg. Department (also at Princeton University), and C.P. Lee, Sudangshu Karmakar, and Marvin Israel (all my colleagues at Bell Communications Research). I apologize to those others who have helped me and whose names I may have omitted.

The faculty and staff of the Department of Computer Science provided a friendly atmosphere to conduct the research presented in this dissertation. I am awed by the efficiency and perfection of Sharon Rodgers in taking care of the day to day departmental chores. She as well as Winnie Waring, Gerree Pecht, Grace Kehoe and Eugene Davidson were always helpful.

I must thank my brothers E. S. Dwarakadasa and E. S. P. Das for guidance and advice regarding ways of the world. Special thanks go to my brother E. S. Badarinarayana who looked after me, during the past year, like my mother used to, taking care of all household chores and making sure I ate and slept well during the most strenuous period of thesis writing. I also wish to thank my previous apartment-mates Prakash C. Shrivasthava and Sundaram Thangavelu, who provided me with a homely atmosphere to live in.

I am indebted to my wife, Jayashree, who, in spite of having to live away from me, has kept my spirits up by regularly writing me letters full of love, encouragement, and hopes for our reunion. Last, but not least, I especially owe my sincere gratitude to my parents, who have shaped me into what I am today, and to my brothers, sisters, and their families for their constant support.

This research was supported in part by NSF grants MCS83-03926 and CCR85-05517.

ABSTRACT

Reflections add a new dimension to the realism of computer generated scenes. The only existing method for implementing reflections satisfactorily is ray-tracing. However, it has major drawbacks, the most obvious being its expensive CPU requirements. The objective of this dissertation is to gain an understanding of the nature of reflections, and to develop alternate methods for rendering them by exploiting their geometric coherence.

We develop a disciplined science for characterizing reflections in curved surfaces. There is no hope for closed form solutions to rendering them since they are algebraically intractable. However, we describe three techniques to handle the situation: contour tracing, special purpose numerical methods, and spline approximation.

We extend four existing algorithms to include reflections, namely, the painter's algorithm, the z-buffer algorithm, the scan plane algorithm, and what we call the curvilinear trapezoid method. Methods and scope for parallelizing these algorithms are also discussed. A comparative study of these algorithms is attempted to outline their suitability under various computing environments. The computing environment is thus a significant factor for the choice of an algorithm. Our efforts are focused on scenes composed of spheres but more general scenes are possible.

TABLE OF CONTENTS

1

REFLECTIONS IN CURVED SURFACES

The pursuit of an idea is as exciting as the pursuit of a whale.
— Henry Norris Russell (1877-1957)

1.1 Introduction

Computer-generated pictures are a very powerful means of man-machine communication. They are being used effectively both for forming the input commands to be executed on a computer and also for the feedback of the results of the computations in a form suitable for quick and accurate interpretation by the humans. The adage "One picture is worth a thousand words" continues to hold true in this area.

Reflections in curved surfaces add a new dimension to the realism of computer-generated pictures. They give a powerful tool to the creative artist to produce pictures that look magnificent to the on-looker but have relatively simple underlying structure. Plate 1 shows an example of a picture that was generated with multiple-level reflections. The picture consists of only five simple objects. There are four balls with reflecting surfaces inside a fifth ball whose interior is a perfect mirror.

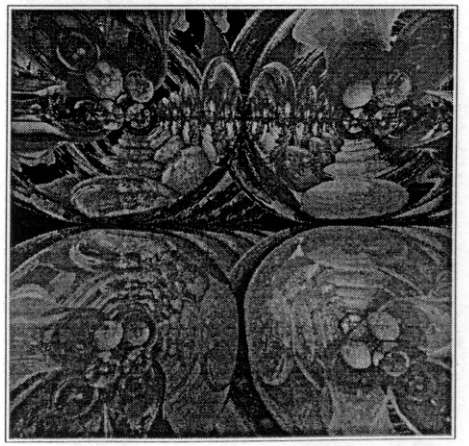


Plate 1 Cosmic Reflections

1.2 Background

Almost in every walk of the modern life we see the influence of computer-generated pictures. The commercials we see on television continue to grow more sophisticated in terms of picture quality and creativity because of the synthetic pictures generated by computers. The making of commercials is a demanding process because the time available for the broadcast is very limited and in that period the maximum possible information has to be relayed to the viewer. This requires the production of good quality pictures of high visual complexity that catch the eye. This has been possible only because of the computer generation of pictures in a way that is hard to achieve by other means. The following are some of the other common applications of high quality synthetic images.

Who does not love cartoon shows on television? In the earlier ages the cartoons were produced by hand. In a process called *cell animation*, the main artist draws certain *key frames* and the assistant artists draw the *in-between* frames. Each of the frames then form one frame in the final movie. Nowadays, even though certain traditional cartoons are still produced by cell animation, many of the modern cartoons are produced with the computer. The main artist draws the key frames – often with the aid of a paint program – and the in-betweening is done by the computer. With the computer taking care of most of the routine tasks, the cartoon can be produced in a fraction of the human time it would otherwise require. Computer-generated movies are produced much in the same fashion except that the key frames may also be produced with the aid of computer modeling and rendering of the scenes involved.

In a construction project it is desirable to have an engineering drawing of the proposed building. By analyzing the drawing, the engineers can pin-point any design flaws before the construction begins. The drawings are also used by the builders for the selection and measurement of the materials used in the construction. With the help of computers, the drawings can be generated quickly and changes applied easily. The design calculations can also be carried out on the same computer which houses the

design drawings. This saves a lot of time and avoids inaccuracies in the design process which would creep in if the calculations were carried out by hand. With advanced technology, the computer can display full color pictures of how the finished building would look from various viewpoints. This gives a chance for the artists, in charge of the aesthetic appearance of the building, to express their opinions. A striking example of this use of pictures generated by computer modeling is in the recent restoration of the Statue of Liberty [FREN86].

Simulation is another field which makes extensive use of computer-generated pictures. In designing a robot, a rocket, an assembly line for the manufacture of a new gadget, etc., it would be very expensive to build and test the final version. A smaller version is built in the laboratory and tests are conducted on it. This is known as simulation or prototyping. An alternative would be to use a computer model of the product and carry out the simulation using the computer. The analysis of the success or failure of the product can be carried out by observing pictures generated by the computer as a result of the simulation. Flight simulation is a very good example of computer simulation where the trainee pilots are taught the controls of an airplane using real time simulation of the runway and the cockpit employing computer-generated pictures. This gives a realistic view to the trainee without any risk to himself, the plane and would-be passengers.

Education is slowly being impacted by the use of computer-generated pictures. For example, the principles of physics and chemistry can be illustrated easily with the aid of computer animation. This enhances the understanding by the pupils and at the same time leads to a reduction in the cost of the materials required for conducting the actual experiments. For this reason more and more graphical tools are being added to the *CAI (computer aided instruction)* courses.

Engineers and scientists often conduct experiments on supercomputers which result in enormous amounts of data. It is virtually impossible for humans to go through these data in their raw form and interpret the results. With the aid of computers,

these data can be displayed as animated multidimensional graphs and can be quickly and easily interpreted.

We encounter computer-generated pictures in other diverse areas such as automobile dashboards, typesetting of books, music-video and other television entertainment, video games, computer art, medical imaging, computer aided design and manufacture, molecular modeling and drug design. Such is their impact on our day to day life.

1.3 The Quest for Realism in Computer Graphics

In computer science, the generation of pictures by the computer is formally associated with the field of computer graphics. Computer graphics, as a science, has focused on algorithms and data-structures for generating synthetic images with realism close to that of nature. It is a fast developing field with many fascinating areas. Every scientist and engineer should have some knowledge of computer graphics.

It is interesting and educating to follow the history of development of computer graphics. In its infancy, computer graphics started out with algorithms for lines [BRES65], conics [PITT67] and polygon filling. These are basically two dimensional (2-D) primitives. Then developed algorithms for three dimensional (3-D) geometric modeling, hidden line/surface elimination [SUTH74], shading, shadow generation [CROW77a, ATHE78], and anti-aliasing (smoothing of silhouettes) [CROW77b, CATM78, GUPT81, TURK82], various illumination models [BLIN77, COOK81] and so forth. The 3-D objects were basically represented by a set of polygons. As a result, most of the hidden surface elimination and other algorithms developed in the early stages focused chiefly on polygons as primitives. Gouraud [GOUR71] and Phong [BUIT75] developed two shading methods by which the resulting pictures looked smooth in spite of the underlying polygonal representation.

In the mid 70's, a fast method for rendering spline patches [CATM74] was introduced and used to represent curved surfaces. During the same period a method for mapping two dimensional textures onto the surfaces of three dimensional objects

was devised. These additions made possible the generation of pictures smoother than those obtained with polygonal approximations.

In the early 80's, a versatile technique [WHIT80] for rendering complex scenes was popularized. This is the "*ray-tracing*" algorithm. The major contribution of ray-tracing is the addition of reflections and refractions to computer-generated scenes. Remarkable pictures can be generated using this algorithm. Many recent researchers have further broadened the scope of ray-tracing [AMAN84, COOK84, GLAS84, HECK84, KAJI82, KAJI83, KAJI84, SEDE84, SPEE85, WYWI86, FUJI86 and JOY86].

Along with these developments authors have reported other schemes to render curved surfaces using their direct representation. Porter [PORT78, PORT79] reports a method for obtaining smoothly shaded spheres. He allows only orthogonal projection. Knowlton and Cherry [KNOW77] describe a method for drawing only the visible portions of spheres used to model molecules. They achieve this by dividing the projections of the spheres into visible curvilinear trapezoids clipped by spheres closer to the eye. The curves of intersection are approximated by circular arcs. Pique [PIQU83] uses z-buffering to allow intersecting molecules. He uses incremental algorithms approximating cosines, to compute shading. Max [MAX79] renders the spheres in perspective. He uses approximate circular arcs to represent arbitrarily oriented ellipses, which are in turn perspective projections of spheres on the display screen.

Spheres are the simplest of a class of curved surfaces called quadrics. Weiss [WEIS66] describes a method for obtaining line drawings of orthographic views of scenes consisting of plane and quadric surfaces. The algorithms of Mahl [MAHL72] can draw filled-in quadric patches. There are techniques available to generate shadows of curved surfaces on curved surfaces [WILL78].

There is a lot of literature available on various topics in computer graphics. It is impossible in this current context to cover all of them in great detail.

1.4 Reflections in Curved Surfaces

Most of the authors mentioned in the previous section focused chiefly on generating pictures with smooth shading and shadows, but left out reflections. In fact, in the context of some of the applications such as molecular modeling, generation of reflections will be confusing to the user rather than serving any useful purpose. In many other applications such as computer art, mathematical modeling, advertisement, etc., reflections in the scenes make them look closer to nature and enhance their realism. The first satisfactory generation of reflections was with ray-tracing. Reflections and refractions add a degree of realism to the computer-generated pictures. Plates 1 and 2 show examples of what can be generated – with the aid of computer graphics – using reflections alone. The study of reflections is the main goal of this dissertation.

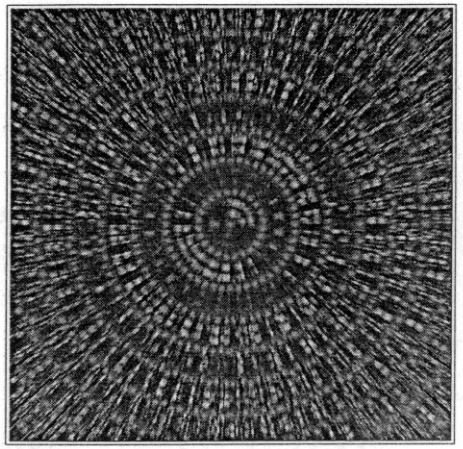


Plate 2 Kaleidoscope

There seems to be a lack of science dealing specifically with the computer generation of reflections. This is in spite of a great deal of research that has been carried out

and recorded in the field of physics. Physics studies reflections both for a theoretical understanding of the nature of light itself and for applications to the effective design of optical instruments. To meet this latter purpose it is sufficient to study reflections of point sources. In computer graphics, there is a need of a knowledge base for studying reflections of more general class of objects. We develop a disciplined approach to the study of reflections and the design of algorithms to generate them efficiently on a computer.

A close look at virtually any computer synthesized picture reveals that smoothly shaded curved surfaces are necessary to achieve realism close to that of nature. This is because many of the objects found in nature as well as those made by humans have curved shapes. Approximation of these shapes with polygons and using smooth shading methods of Gouraud [GOUR71] and Phong [BUIT75] are not always satisfactory. The silhouettes of pictures generated with these methods still show a pronounced polygonal structure. As mentioned above, there exist methods to handle curved surfaces. Our purpose here is to develop techniques to generate *reflections* in curved surfaces.

Plane surfaces are equally important, but the reflections in plane surfaces have been studied extensively and they obey a simple rule, namely "*the distance of the image behind the (plane)mirror is the same as the distance from the object in front of it.*" Further, there is a one-to-one correspondence between an object and its reflection in a plane mirror. Thus, reflection in a plane mirror is just a translation and a rotation. The reflections in curved objects are not so simple and warrant further study and research.

Thus the theme of the thesis is reflections, mainly in curved surfaces. We intend to bring about the same level of understanding for reflections in curved surfaces as that exists for reflections in planar surfaces. We will also develop algorithms and data-structures to generate them efficiently on a computer. We study the following aspects of reflections in curved surfaces: mathematical tractability, physical and geometric

characterization, high-level algorithms and data-structures, primitives or low-level algorithms and data-structures, and hardware implementation issues. We also hope to pave the way for future research in this direction as we think it is a challenging as well as a rewarding area.

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows:

Chapter 2 starts out with a motivation for our current approach to the generation of reflections. It explores previously existing methods and discusses their advantages, disadvantages and scope.

Chapter 3 deals with properties of reflections and the challenges involved in their computer generation.

In chapter 4 algorithms for rendering a basic reflection are discussed. Three methods are outlined, namely the general contour tracing algorithm, special purpose numerical algorithm and spline approximations. The chapter ends with a brief introduction to the high level algorithms which are described in detail in subsequent chapters.

Chapters 5 through 8 expound algorithms for rendering reflections with hidden surfaces removed. These algorithms assume the rendering of a basic reflection as a primitive. Four different algorithms are considered, namely Modified Painter's Algorithm, Modified Z-buffer Algorithm, Scan Plane Method and Curvilinear Trapezoid Method. They are characterized by the underlying hidden surface elimination methods. They also have various levels of coherence. Complexity analysis and parallelism are touched upon.

Chapter 9 compares and contrasts the four algorithms and tabulates the results. The choice of the algorithm is influenced by computer resources required, ease of implementation, flexibility of the algorithm to run in various environments and the quality of the picture generated. The z-buffer algorithm apparently is easy to implement and

uses less resources whereas the curvilinear trapezoid method requires elaborate programming but yields good quality output on various types of output devices. The other two algorithms offer compromises between the two methods.

Chapter 10 reviews the contributions of this work and describes some interesting and challenging open problems.

Appendices 1 and 2 are supplements to the text in chapter 3. They detail some of the characteristics of reflections in curved surfaces.

I do not know what I may appear to the world;
but to myself I seem to have been only like a boy playing on the sea shore,
and diverting myself in now and then finding a smoother pebble
or a prettier shell than the ordinary,
whilst the great ocean of truth lay all undiscovered before me.

— Sir Isaac Newton

2

MOTIVATION

Clay is molded to make a vessel,
but the utility of the vessel lies in the space where there is nothing ···
Thus, taking advantage of what is,
we recognize the utility of what is not.

— Lao Tze (604-531 BC)

In this chapter we define what we want to achieve, namely efficient techniques for generating reflections in curved surfaces. We first explore the work carried out by previous authors to determine what exists and what is lacking. We present a brief survey of two existing schemes and discuss their scope as well as their merits and demerits. The chief purpose of this chapter is to bring out the motivations for the research described in the subsequent chapters.

2.1 We Want to Build a Reflection Engine

Our goal is to devise techniques to efficiently generate pictures of complexity similar to that of Plate 1 (chapter 1). The main focus is on the representation and generation of reflections using a computer. We will describe algorithms that can be implemented in software or hardware or a combination of both.

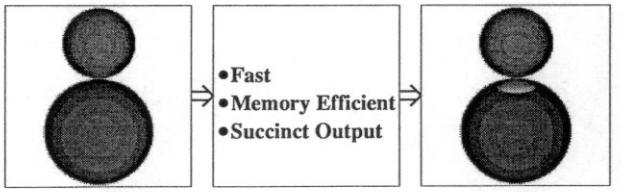


Fig. 2.1 The Reflection Machine

Fig. 2.1 illustrates the above idea. Our engine takes as input the representation of two or more curved surfaces and yields a picture depicting the reflections in the reflecting surfaces in the scene. (We will come back to representation schemes in chapter 4). The algorithms should be fast and memory efficient. In other words they should make the best use of the available resources. The output should be succinct, that is, it should precisely represent the scene and no more or no less. To illustrate, the two balls in the scene on the left of Fig. 2.1 can be compactly represented by their equations and the material properties of their surfaces such as their reflection coefficients. The output scene can in a similar fashion be represented by the same

sphere equations and in addition the contour of the reflection generated. This can be done in a device-independent fashion rather than as a bitmap or a bytemap which is a raster dependent low level description of the final picture desired. (Again, we will have a more detailed discussion of this topic later).

2.2 Current Schemes Are Too Slow

There are two existing methods for generating scenes with reflections. One, the texture mapping scheme, simulates reflections and is limited in scope. The other, ray-tracing, is a simple yet powerful method for rendering scenes with reflections as well as refractions. We will describe each method in order.

2.2.1 Simulated Reflections by Texture Mapping

Catmull introduced this method. In his thesis [CATM74] he mentions that reflections in curved surfaces can be simulated using texture mapping, and the work includes a picture of "142 bottles and glasses" with simulated reflections. A detailed description of the technique can be found in a later article [BLIN76]. The following is a brief outline.

Let us suppose that there is only a single reflecting object in the scene. Further we assume that the other surrounding objects and light sources are at large distances from the reflecting object and occlusions of the environment by parts of the reflecting object itself are neglected. With these simplifying assumptions, to the reflecting object the environment appears as a picture painted on the inside of a large sphere. The object itself is deemed to be positioned at the center of this sphere. Then the environment can be modeled as a two dimensional texture map indexed by polar coordinate angles. This texture is then mapped on to the reflecting object in the following way to simulate reflection of the environment in the object. For each picture element representing a portion of the reflecting surface, the direction of the ray which is reflected to the eye is computed. The polar coordinate angles of this direction are used to index into the

above texture map to determine the reflected light intensity. Both [CATM74] and [BLIN76] have examples of pictures generated with this scheme.

The chief advantage of this method is that it allows the reflecting object to have a complicated surface. The major drawback is that mutually reflecting objects are hard to handle, if not impossible. It is further complicated by multiple reflections. Also, the method assumes that the light sources and the objects being reflected are far away from the reflecting object. Whereas this assumption is plausible for light sources as is frequently encountered in orthogonal projection methods, it is not always true of the reflected objects (Plate 1, chapter 1). This method has some of the other shortcomings in common with the second method, ray-tracing. This latter method is more interesting and is worthy of a broader description.

2.2.2 Ray-tracing

Ray-tracing was popularized in the computer graphics domain by Whitted [WHIT80]. Fig. 2.2 illustrates the basic working of the ray-tracing algorithm. A ray R is traced from the eye E towards a pixel P on the logical display screen. If this ray intersects the surface of an object in the scene at a point O, the color of the material of the surface of this object at O is used to compute the coloring of the pixel P. If the surface at O is a reflecting one, a reflected ray R′ is generated according to the laws of reflection [BORN65] and this reflected ray is traced in turn. If it hits another object at point S, the contribution of the surface at S is suitably scaled with the reflection coefficient of the surface at O and added to the color of P. This is illustrated by the following simplified equation.

$$C_P = C_O + (K_O \times C_S)$$

where C_P is the color attributed to the pixel P, C_O is the color contribution of the surface at O, C_S is the color contribution of the surface at S, and K_O is the reflection coefficient of the surface at O.

Multiple reflections can be easily accomplished by spawning further reflection rays and adding their contribution to the color of the pixel P. Refraction and shadows can

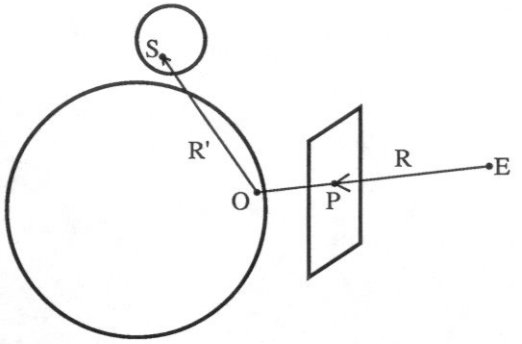


Fig. 2.2 Ray-tracing Basics

be incorporated in a similar fashion by tracing refraction rays and shadow rays. The refraction laws [BORN65] govern the generation of refraction rays and shadow rays are simply rays towards the light source from the points of intersection such as the points O and S in the above example.

The rendering of the scene is complete when rays to all the pixels of the display screen have been traced, including the reflection, refraction and shadow rays.

Ray-tracing has occupied a place in computer graphics as a simple yet powerful method for rendering scenes with realism close to that of nature. It encompasses many aspects of picture generation such as specular reflection, shadows and refraction. This power is derived from the fact that ray-tracing "simulates" nature. As it turns out, simulating nature isn't all that easy and it is a very CPU intensive process. For example, to render a scene with a resolution of 512 × 512 pixels with just one level of reflection, 512 × 512 × 2 = 524,288 rays have to be traced. In a simple ray-tracer, a major portion of the CPU time is spent on computing intersections of these rays with the objects in the scene.

It is no wonder that several researchers have focused on devising methods to speed up the ray-tracing process. These involve such methods as dividing the world into an

octree data-structure [GLAS84, FUJI86], arranging the world as a constructive solid geometry tree [WYWI85, WYWI86] and so on. These techniques take advantage of the image space coherence to reduce the number of intersection calculations to be performed per ray. Other approaches such as working in integer arithmetic for tracing the rays [FUJI86] are used to speed up the tracing process itself.

The conventional methods achieve speed by time-tested methods for incremental calculations exploiting the object space and image space coherence. However, in the field of ray-tracing not much work seems to have been done on exploiting the coherence properties of the objects in the scene to reduce the number of rays traced. Also, a given ray is computed independent of the other rays adjacent to it. It is possible that rays close to each other may intersect the same object. In this case it would be advantageous to trace only one ray and use the information gained to reduce intersection computations for the adjacent rays. A recent publication [SPEE85] has addressed this issue and it reports that not much can be gained with this approach because of the overhead involved in maintaining coherence information.

We would like to take the approach similar to those of the traditional methods, for generating reflections. We try to demonstrate that, making use of coherence, certain simple objects in the scene can be rendered very fast without the use of ray-tracing at all. To illustrate our approach to the situation, we use simple spheres and render them with reflections.

2.3 Motivation for Our Approach

The basic motivation for the current approach came from looking at the multiple reflections of four balls inside a fifth ball as shown in Plate 1 (chapter 1). This picture generated with a Whitted [WHIT80] style ray-tracing software took more than 3000 CPU minutes on a VAX/750. It was generated at a resolution of 512 × 512 and four rays per pixel were traced. It has 10 levels of reflection and includes shadows. Thus, the total number of rays traced is $512 \times 512 \times 4 \times 22 = 23068672 \approx 2.3 \times 10^7$. It occurred to us that there is a definite pattern to the reflections occurring in the picture and,

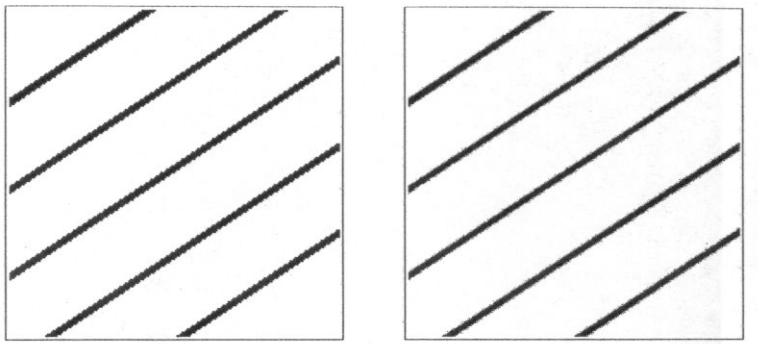
by attacking the problem directly, we might be able to do better. Other motivations to take the direct approach come from the paper on coherent ray-tracing [SPEE85], where the authors have reported that making the ray-tracing coherent does not buy us significantly more power. And hence it is wise to look into other approaches. Though there have been numerous *ad hoc* algorithms in the literature focusing on making ray-tracing itself faster [GLAS84, FUJI86], we think that there is a limit to which such improvements can lead.

When rendering a particular object in the scene, we know exactly what portion of the screen it projects onto. Thus, there is no need to *poke* every point on the screen (as in ray-tracing) to figure out if indeed it represents an object in the scene. Similar arguments hold for reflections of other objects in the scene in the given object. The projections of reflections in an object lie within the silhouette of the reflecting surface.

There are other advantages to using the direct approach. The following are some of them.

Anti-aliasing: We assume that the readers are familiar with the aliasing artifacts [CROW77b] on raster displays. One striking example of the effect aliasing is that a line drawn on the computer display appears to be jagged. Fig. 2.3a shows lines with exaggerated aliasing. The picture was generated at a resolution of 256 x 256 pixels. The aliasing is due to the fact that the pixels on the screen are treated as discrete points. The pixels closest to a given line are lit up and those farther away are turned off. In actual practice, each pixel occupies a finite area of the screen. Hence, turning a pixel off or on is equivalent to effecting the entire area of the pixel, rather than a single point. This leads to the aliasing effects.

One method of achieving anti-aliasing is to let the intensity of each pixel be proportional to the area of the pixel covered by a line passing through it. This is known as *area sampling*. This assumes, of course, that we have multiple gray levels of intensity available for each pixel. Fig. 2.3b shows the lines drawn with this type of anti-aliasing. The pixels each have 256 gray levels.



(a) (b)

Fig. 2.3 (a) Aliasing Effect (b) Anti-aliased Lines

As mentioned above, ray-tracing simulates nature. However, the naive ray-tracer does only a *discrete event simulation*. In simple words, it takes only discrete samples of the scene and this leads to aliasing effects. Each pixel on the display screen covers a certain finite area. The ray-tracer treats the pixel as a point concentrated, say, at the intersection of the grid lines. The color computed for this point is attributed to the whole area of the pixel. In effect, the ray-tracer is sampling the picture at the discrete "points" rather than along the whole pixel area. This is why we termed its approach a discrete event simulation. To avoid aliasing, we need to compute the contribution of a portion of the scene which projects onto the finite area of the pixel and assign the corresponding color to the pixel. This is known as *area sampling*. Some researchers have addressed this issue but cannot claim complete area sampling of the scene [AMAN84, HECK84].

When anti-aliasing a given scene, our method has the knowledge of the silhouettes of all the objects in the scene including reflections, and it is only necessary to anti-alias these silhouettes. To obtain anti-aliasing with ray-tracing, it is necessary to use *ad hoc* methods to detect the *edges* in the scene and then use more rays to get the details

around these edges.

Resolution-independence: Ray-tracing is not *resolution-independent*. If the resolution changes, we have to start all over again to get the output from the input description. There is no resolution-independent processing that can be done once and for all and stored away. The direct approach will be resolution-independent and capable of producing intermediate output independent of resolution. This intermediate output can then be processed, hopefully fast, to get the final image. These advantages manifest vividly as hardware technology progresses making feasible graphics displays of higher and higher resolution. Even today, there are color printers of very high resolution (6' × 6' at 300 dots per inch) which would benefit from fast resolution-independent algorithms. These algorithms, unlike ray-tracing, avoid generation and storage of huge bit maps.

It is conceivable that one can design an algorithm that processes the output of a ray-tracer and yields a bitmap at a higher resolution. However it is well known by information theory that the information content of the image so obtained is no more than that contained in the original lower resolution image. Since the ray-tracer loses some information due to discrete sampling of the scene, to generate a picture at higher resolution, some more rays will have to be traced. This is in contrast to the direct approach which stores all the information about the scene concisely, without loss of any information, and thus the image can be generated at any resolution without any further processing of the input.

Communication Bottleneck: Graphics workstations are getting more and more sophisticated. Today the trend is to have a main processor controlling a special purpose graphics processor. Thus, there is a need for algorithms which achieve the best of both processors. The direct approach is definitely a win here. High level processing can be done on the main processor which breaks a given scene into its primitives and downloads this data onto the graphics processor. The graphics processor will then render the primitives independent of the host processor. The communication time between

the main processor and the graphics processor may be the chief bottleneck of the whole process as evidenced in a recent investigation [PAND85].

As more powerful graphics processors emerge, we need to define more primitives which can be rendered on these processors. This reduces the data that needs to be transmitted from the host to the graphics processor. For example, to display a polygon with a fill-in of its interior, the host processor can generate the line segments that make up the polygon and communicate these to the graphics processor. Alternatively, with today's technology, a polygon filling sub-routine can be defined on the graphics processor as a primitive in either hardware or firmware. The host then needs only to transfer the vertices of the polygon to be drawn. With this approach there will be an order of magnitude reduction in the data that needs to be transferred. Since ray-tracing produces only bitmaps as its output, it cannot take much advantage of the advances in the local *intelligence* of graphics processors.

Also, with the advent of supercomputers, more and more complex problems will be solved and we need graphic outputs to show the data precisely and quickly to the humans. Thus, transfer of huge bit-maps from the supercomputer to the graphics processor is not only a waste of communication time, but also a waste of supercomputer time itself. In many cases it may be unrealistic and outright infeasible. The importance of I/O bandwidth of a computer cannot be overemphasized. Special purpose desktop computers are emerging with the processing power of the fastest general purpose supercomputers such as CRAY/XMP, at a fraction of the price. The major distinguishing feature between the two classes of supercomputers is the I/O bandwidth. Thus, if we can limit the amount of data to be input and output to such computers, we can derive their maximum efficacy.

As technology progresses, computers and communications will merge into a single entity. Computers will be connected together with networks. The networks will be the slower of the two elements. The information receiving terminals will be capable of local processing. Thus, it is advantageous to transmit data as efficiently as possible.

Instead of transmitting huge bit-maps, we should therefore transmit succinct information which can be processed at the receiving end before displaying to the humans.

Storage: Closely connected with the communication and previewing (see next paragraphs) is the question of representation and storage of pictures. With the information age of the future, certain pictures will form the core data base which will be transmitted many more times than others. This gives us another motivation for succinct representation. In computer animation, one may need to store several frames of data and flash them onto the screen at near video rates. Because of the number of frames to be stored for a realistic real time movie, the data has to reside on a mass storage medium such as the magnetic disk. The disks are rather slow devices compared to the CPU, and again compact representation of data is necessary to achieve the required response time.

Device-independence: The representation of the output should be independent of the device on which it is finally going to be displayed. The output of the ray-tracer is a bit-map and is thus suitable only for displaying on raster devices. If, on the other hand, the output is represented in a device-independent manner, it can be then rendered either on a raster device or on a vector device. To illustrate, consider a simple straight line. A line is represented simply by its two end points. If we just store these two end points we can display the line on raster displays using scan conversion algorithms. And the same line can be displayed on a vector device using the vector drawing capabilities of the system. If we stored the bit-map that represented the line, this cannot be done. We propose to develop algorithms that yield the contours of the reflections in a device-independent representation.

One of the applications of computer graphics is to making computer-generated movies for both entertainment and educational purposes. In making a movie, the scene has to be rehearsed several times to suit the script which may change several times over the production cycle. It is very expensive to generate the picture in its final form for this purpose. It is usually quite adequate to draw a very simplified form

of the picture and the artist can easily visualize from this version whether the movie suits the script. This is what we mean by *previewing*.

Device-independent representation allows fast previewing. Since we know the contours of all the reflections, we can just display these contours at their full size for previewing without filling them. To preview a picture for ray-tracing one has to generate it at a much lower resolution. This leads to loss of details and the artist may not be completely satisfied. The direct approach is thus much more pleasing and cost effective in an environment for the production of animated scenes.

Plate 3. Cosmic Reflections
(Single precision used for storing intermediate results)

Round-off Errors: There are round-off considerations with ray-tracing, especially with multiple levels of reflection. Plate 1 (chapter 1) was generated using double precision (64 bits) for all calculations and storage of intermediate results. Plate 3 shows the same picture generated using single precision (32 bits) for storing intermediate

results. The discrepancy in Plate 3 is definitely due to round-off errors. In ray-tracing, with every hop of reflection or refraction, the error accumulates and there is no good mechanism by which we can detect and adjust for this error dynamically. On the other hand, with the direct approach, since all the data is at hand, higher level reflections can be computed at a higher precision if needed.

The above paragraphs serve a dual purpose. They point out the disadvantages of existing methods and at the same time they detail the characteristics of the algorithms that we are searching for. The next chapter outlines the challenges involved in modeling and rendering reflections.

Everything has its beauty but not everyone sees it.
— Confucius (551-479 BC)

3

A REFLECTION ON REFLECTIONS

Man and science are two concave mirrors continually reflecting each other.
— Alekzandr Ivanovich Herzen (1812-1870)

Chapter 2 justified the need for alternate methods to ray-tracing for rendering scenes with reflections. In this chapter we undertake a systematic study of reflections. We enumerate the characteristics of reflections and the issues involved in modeling them and rendering them.

The Challenge

Reflections in curved surfaces have the following properties. Each of these properties offers a certain resistance to the computer rendering of reflections.

[1] They elude analytical methods.

[2] They differ from ordinary objects.

[3] Their colors need to be blended with those of the reflecting objects.

[4] They are constrained to silhouettes of reflecting objects.

[5] Their shape and position are eye-position dependent.

[6] They can be real or virtual.

Let us explore each of the above properties in greater detail.

3.1 Good-bye Analysis

Most of the current knowledge about reflections comes to us from physics – geometric and physical optics. The scope of this dissertation does not permit us to review all results of the physics of reflections, but we shall allude to the relevant ones as needed. In the development of what follows in this chapter, the reader is assumed to be familiar with the fundamentals of optics, such as the laws of reflection [BORN65].

Now You See It, Now You Don't!

Reflections in plane mirrors are well behaved. The reflection in a plane mirror is virtual, is of the same shape and size as the source, and the position of the reflection is such that "*the image behind the mirror is at the same distance from the plane mirror as the object is in front of it.*" Reflections in curved surfaces are very different and

defy analytical methods of characterizing them. In general, the shape of the image of a point source in a curved mirror surface resembles that of a comet. This type of aberration is well known in physics as *coma*. This causes serious problems in the design of optical instruments employing curved elements such as lenses and mirrors and has been studied extensively in physics [BORN65]. Thus, there does not exist a simply definable reflection in a curved surface, not even in a sphere, the simplest of the curved surfaces! Please see appendix 1 for a brief narrative on why it is so.

This is rather disappointing since it removes the possibility of getting an analytical closed form solution to the problem. Readers may wonder at the fact that they do see a reflection in a spherical ball such as a mirrored Christmas ball. However, the image we see in such a ball is a "fuzzy" reflection. The eye acts as an optical integrator and, together with the brain, compensates for the loss of clarity and "makes up" the reflection that we "see."

Study of Ray-tracing Offers a Solution

Now let us stop for a moment and ask ourselves how the pictures in Plate 1 and 2 were obtained. They do seem to have a close resemblance to what we may see in reality. These pictures were obtained with a ray-tracer. Maybe taking a closer look at the ray-tracing process can tell us something.

Recall from chapter 2 how a ray-tracer works to render a reflection. A ray R is traced from the eye E to a pixel P on the logical display screen(Fig. 3.1). If this ray intersects a surface at a point O, the color of the material of the surface at O is used to compute the coloring of the pixel P. If the surface at O is a reflecting one, a reflected ray R′ is generated according to the laws of reflection and this reflected ray is traced in turn. If it hits another object at point S, the contribution of the surface at S is suitably scaled with the reflection coefficient of the surface at O and added to the color of P.

Immediately we can observe the following: Even though there is no clearly defined

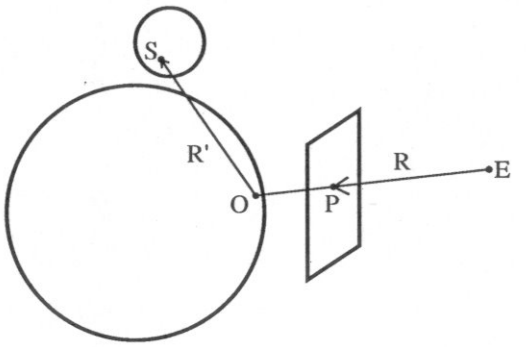


Fig. 3.1 Ray-tracing Review.

virtual image point, there exists a *unique*[1] point O on the surface of the reflecting sphere such that the ray starting from a given point S has to reflect off the surface at this point in order to reach the eye situated in a given position. The following is another way of stating the same idea. Imagine a pencil of rays emanating from a given point S. These rays diverge in different directions after reflection in the reflecting object. If the reflected rays diverge[2], of all the reflected rays, only one ray passes through the eye position E. If we trace this ray backwards to its source, we find that it is incident on the reflecting surface at a single point and this point is the unique point O. As we have seen above, this is what a ray tracer finds, only in the reverse order.

We can extend the above idea further. Consider all the rays originating from points on a given object which reach the eye after reflection in the surface of a convex body. The points of incidence of all such rays, on the reflecting surface, lie within a closed region. There exists a clearly defined boundary of such a region. Fig. 3.2 depicts the reflection of the sphere S in the sphere O. The boundary of the reflection

[1] We wish to emphasize a minor point here. The uniqueness holds true for virtual reflections; it may not be always true for real reflections.

[2] This is the condition for the formation of a virtual image.

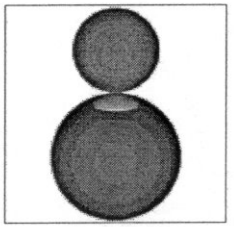


Fig. 3.2 A sphere reflecting another.

is what we call the contour[3]. Our thesis is that if we are able to find this contour *we can then fill in the region enclosed by the contour in a fast and efficient manner.* We say this because we know, for the points in such a region, the exact objects that are involved in the reflection and hence need not trace many rays unlike a ray-tracer. We can use interpolation techniques or other suitable methods to derive efficiency. What we obtain as a result — we cannot claim to be the true reflection — will rival that obtained by ray-tracing.

Even with the above simplification, the geometry of reflections is still hard to tackle analytically. Even for one level of reflection, the analytical method involves the solution of a sixth degree polynomial equation in two independent variables. Appendix 2 details the algebra involved for interested readers. Thus, we are forced to abandon the search for analytic closed form solutions and to look for alternate approaches. Our solutions to the problem are outlined in the next chapter where we address three algorithms for generating the basic reflection of one object in another.

3.2 Reflections Are Not Solid

From a computer graphics point of view it would be ideal if we can consider the reflections as nothing more than ordinary objects so that the conventional hidden surface and rendering algorithms can be applied to the objects and their reflections in

[3] As seen on the display, it is actually the boundary of the projection of the reflection on the display screen.

a uniform way. In the above paragraphs we just saw that such modeling is not possible. In addition, the reflections are distinct in that they do not hide the ordinary objects but only effectively hide themselves and higher level reflections. Thus, conventional hidden surface algorithms would not work. We need to invent new ones or modify the existing ones.

One important observation is to be made here. If there are $\mathbf{n}$ objects in the scene and we wish to see $\mathbf{d}$ levels of reflections the output size is of complexity $O(\mathbf{n^d})$. However, suppose the complexity of a conventional hidden surface algorithm is $O(f(\mathbf{n}))$, where f is a function of $\mathbf{n}$. The hidden surface algorithm for a scene with reflections need not (and should not) be of the complexity $O(f(\mathbf{n^d}))$. This is because the reflections are uniquely determined by the positions of the reflecting objects and the eye. Thus, there are some definite patterns in the scene which the algorithms should exploit to achieve efficiency.

To illustrate, consider a scene made up of two reflecting objects A and B. When solving the hidden surface problem for the base spheres A and B, the reflections in them can be ignored. Similarly, reflections in the sphere B do not hide the reflections in the sphere A, and vice versa. This pattern extends to higher level reflections. Define AB to imply the reflection of A in B and BA to represent the reflection of B in A. Further, ABA represents the reflection of A in B reflected back in A, which is a second level reflection. A string $S_0S_1S_2\cdots S_n$ is then the reflection of S_0 in S_1 in $S_2\cdots$ in S_n. This is an nth level reflection. All the reflections in a scene can thus be represented by a set of strings. There is a one-to-one correspondence between the length of the string and the level of the reflection. The alphabet, of which these strings are made, consists of the letters associated with the individual base objects. Not all such possible strings represent actual reflections. They cannot have any two consecutive letters the same. For instance, the string AA does not make sense, because, according to our definition, it would mean the reflection of A in A. We can group the reflections according to the length of their string representations. Then we can consider the hidden surface

problem for each group separately. Reflections with a given length of string do not affect reflections with a longer string representation.

Another point is that to compute the color contributions of reflections we need to trace information about both reflecting objects and reflected objects. Thus, additional computation is needed. Hilariously, the old Indian adage "One object in the hand is worth two in the mirror" does not apply to curved surface reflections.

3.3 The Problem of Color Blending

Hidden surface algorithms for reflections are further complicated by the problem of properly blending the colors of reflections with those of the reflecting objects.

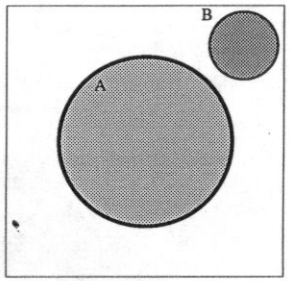


Fig. 3.3a

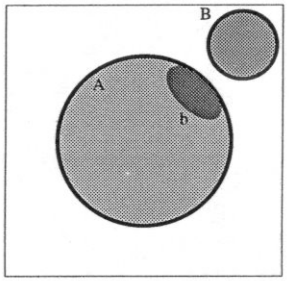


Fig. 3.3b

When a sphere is rendered onto the display screen, the contribution of the color of each point on the sphere replaces the corresponding area of the display which we have termed a pixel. When a reflection in this sphere is rendered, the contribution of the reflection does not replace the pixel color (which represents the base sphere) but adds onto it. For example, in Fig. 3.3a the spheres A and B were rendered onto an initially empty display. In Fig. 3.3b, the reflection b of B in A is rendered which adds to the color of A.

The requirement of color blending for reflections is similar to that demanded by rendering of transparent objects [KAY79], only more pronounced. When a opaque object is rendered, its color simply replaces the background color. When a transparent object is rendered, the background color (of objects previously rendered) is suitably scaled according to the transparency factor and added to the color of the object. When another transparent object is rendered the same scheme can be followed. This does not lead to any problems. On the other hand when a reflection is rendered we have to keep track of whether it hides any other reflections of the same level. The color blending rules are different for regions where the reflection being rendered hides other reflections from those where it does not. In the region where the overlap region between two reflections the color contribution of the reflection which was hidden will have to be some how subtracted out. This adds further complications.

3.4 Clip, Clip, Clip

Reflections are restricted to lie within the silhouettes of the reflecting objects. For example, the reflection b of object B in the reflecting object A in Fig. 3.3b should lie entirely in the circle that represents the silhouette of A. This applies to all levels of reflections. Using our string notation as above, a reflection with the representation $S_0S_1\cdots S_n$ (which is the reflection of S_0 seen in the object S_n after intermediate reflections in all the objects S_1 through S_{n-1}) lies within the silhouette of the reflection $S_1\cdots S_n$ (which is the reflection of level one lower).

This property of reflections is both an advantage and a problem. The problem is that we need to make sure that the reflections do not protrude outside the contours of the reflecting objects. This would require sophisticated clipping methods.

On the other hand, the advantage is that when a reflecting object is rendered, we immediately know that all the reflections in it lie within its contour projected onto the display screen. Thus, knowing the contour of the reflected object gives us the limits within which all the reflections in it lie. Put in another way, we can render all the reflections in an object at the same time as we render the object. As we will see in later

chapters, this does indeed simplify some hidden surface algorithms. Moreover, if an object is completely hidden from view, so are all the reflections in it. If this fact can be used, it saves us a lot of CPU time which we may otherwise spend in rendering them unnecessarily, only to be later obscured. (We may still need to deal with the hidden object since other reflecting objects in the scene may show its reflection in them).

3.5 Eye-position Dependency

Reflections in plane mirrors are independent of the position of the viewer, with regard to both their position and shape. This is an established fact of geometric optics. In contrast to this, the reflections in curved mirrors are eye-position dependent. For example, if one looks at the image of a fluorescent light in an ellipsoidal reflecting surface and moves the eye from one position to another, it can be observed that the image changes its shape as well as its position.

The implications of eye point dependency are not very serious for still-picture generation. However, for making movies with movement of the camera position, each scene may have to be generated afresh. Researchers have tried finding schemes to encode information that does not change from frame to frame [FUCH79, FUCH83, HUBS81] and to use this information in generating consecutive frames of a movie in a fast manner. In this case the scenes consist of objects that do not change their relative positions. The only thing that changes would be the eye position. It would be an interesting research topic to find and exploit the maximum common information contained in scenes with reflecting objects.

3.6 Real vs. Virtual

Reflections can be both real and virtual. Virtual images can only be viewed from the *other side* of the mirror. Real images can be caught on to a screen which lies on the same side of the mirror as the object.

A plane mirror always produces a virtual reflection. A convex reflecting surface behaves in the same manner. However, the reflection in a concave reflecting surface

is virtual or real depending respectively on whether the reflected object is within the focal distance from the mirror or farther from it. A point exactly at the focal distance from the mirror has its reflection formed at an infinite distance from the mirror. This causes complications when an object is partly within the focal distance and partly without. Then part of it forms a virtual image, part of it forms an image at infinity, and the rest of it forms a real image. To avoid this chaos, in this dissertation we consider only virtual images. Furthermore, we restrict our attention to only convex reflecting surfaces.

For every problem there is a simple solution that won't work.
— H.L. Mencken

No problem is so formidable that you can't just walk away from it.
— C. Schulz

4

PRIMITIVES

Whenever there is a hard job to be done I assign it to a lazy man;
he is sure to find an easy way of doing it.
— Walter. P. Chrysler

I have yet to see any problem, however complicated,
which, when you looked at it in the right way,
did not become still more complicated.
— Poul Anderson

In this chapter we develop algorithms and data-structures for rendering a basic reflection of one object in another. These techniques will be used as primitives by higher-level algorithms, described in the next four chapters, to build up a complex scene consisting of several objects, with hidden surface elimination.

We briefly survey computational models suitable for graphics algorithms in general. In addition, we discuss representation schemes for computer models of objects in the scene and comment on the representation suitable for generating reflections. The output representation should reflect the arguments put forth in chapter 2 for device independence, resolution independence, and succinctness.

4.1 Computational Models

We do not intend to exhaust all the models that are available in the field today. Our interest is mainly in an informal qualitative understanding of the models to aid in the efficient design of our algorithms. A model abstracts certain relevant features common to various computers while ignoring others. It helps us to compare two algorithms without having to actually implement and execute them on a real computer system.

The models reflect the computer architectures of the present and the near future. We are going to discuss three models, namely, the conventional model, the graphics workstation model and the parallel model. There may not be a unique algorithm which is suitable across all models. In the same vein, a given architecture may be suitable for certain specific algorithms. Thus, the computing environment is a significant factor in the selection of algorithms.

Conventional Model

The conventional model used in computation theory has been the RAM model [AHO74]. In this model, a single processor executes a program which reads its input data, performs operations on this data and writes out the results. The random access memory, available to the processor to store intermediate results and data, is assumed to be a contiguous array of cells with no limit on the size and number of cells. This model

is suitable for analyzing theoretical complexity of algorithms that run on a single processor. Given that the parallel processing is still a young field, this model encompasses most of the current computer architectures. The model is realistic for programs which are CPU-bound when the problems they handle fit in the main memory of the computer.

Graphics Workstation Model

As display technology is evolving, the trend is to move away from conventional ASCII terminals towards more sophisticated bit-mapped and byte-mapped displays. This allows for both graphics and text to be intermixed. With multiple windowing systems, there may be several activities going on simultaneously. The display needs extensive graphics capabilities for both input and output. It may thus become a bottleneck. In such systems, response time is very critical for good user interface. If the user has to wait hundreds of milliseconds before he can see the result of his action of typing in a key or moving a mouse, his productivity is hindered and the very purpose of using graphics displays may be defeated. At the same time, the main processor cannot be too involved in managing the graphics display as it needs to work on processing "real" data. To solve this problem, graphics workstations employ a dedicated graphics processor with perhaps limited memory and lacking other complex functional capabilities of the main processor, but very fast in accessing the display buffer and rendering graphics primitives. The model here is a main processor communicating with a graphics processor. The main CPU computes the primitives, which the scene to be displayed is made of, and transmits these primitives to the graphics processor which then renders them. Thus the main CPU is free of the small details of rendering and the graphics processor functions efficiently to fulfill its critical role. The authors of [FOLE82] use the notion of a display processor unit or DPU which controls the computer display terminal. The graphics processor of our model plays a similar role but has a more generic scope that covers a broader range of output devices including computer display terminals, digital film recorders etc.

In such a model, there are two issues involved which are closely related. One is the communication channel and the other is the data that needs to be transmitted. The communication channel may very well become the bottleneck. An extreme case of the situation is when a supercomputer transmits data to a smaller computer over a telephone line. In the recent past, the rate of increase in the speed of communication links has not kept up pace with the increase in the processor speeds. There has been the advent of optic fibers which can be used as high bandwidth channels for point-to-point communication. However, this technology has not yet been widely applied. Thus, to make best use of both the processors, we need to make the algorithms CPU bound rather than communication bound. This calls for efficient data representation. There are two approaches to this situation. One is to compress the data on the main processor and transmit the compressed data. The graphics processor will then decompress the data and use it. The effectiveness of this method is perhaps limited. The preferred method would be to define better primitives which encode the data in a compact manner. To illustrate, consider a sphere. It can be defined in terms of its constituent primitives, namely the line segments that make up its image. One can then transmit these line segments to the graphics processor which then renders these line segments. If the sphere is $\mathbf{n}$ pixels tall, then we need to transmit $2 \times \mathbf{n}$ pieces of data – two end points of each line. On the other hand if we define a sphere itself to be a primitive, we need to transmit just two pieces of data, namely, the center and the radius of the sphere. Thus more and more work is being distributed to the graphics processors and the main CPU is off-loaded. In a way it is equivalent to achieving parallelism.

We will be aiming at developing our algorithms with this model in mind. The previous model is going to be outdated and the next model is still not fully developed.

Parallel Model

To get supercomputer performance in a cost effective manner, several existing cheap processors are connected in parallel. With such a multiprocessor, it is possible to

effectively break down a given problem into smaller pieces and hand each piece to a different processor. The processors render their respective pieces in parallel. Thus, if there are **P** processors, the algorithm will *theoretically* run in **1/P** of the time it would take one processor to process the entire problem single-handed. Of course, not all problems can be broken down effectively into smaller equal pieces and the actual speed up may be less than the theoretical maximum possible.

In the light of this model, we try to analyze our algorithms for parallel implementation. Parallel processing is still an infant field but it seems to hold promise in the near future. It makes people think ahead of the current technology: "If enough processing power were available what could be achieved?" Thinking about parallelizing an algorithm can reveal its inner structure in greater glory. It seems that parallel processing is a must for graphics and image processing as well as computer vision. In fact, the way in which the human visual system works may be a parallel process. For example, when we look at a person's face for just a fraction of a second, we can immediately recognize the person. The performance of the eye-brain system is incredible given that the present day computers have not come anywhere close to this response time in recognizing images.

4.2 Representation Schemes

Here again, we will touch on simple schemes. We are interested only in the representation of three-dimensional surfaces and will not deal with other representations.

Polygonal Approximation

The input can be represented as a collection of polygons approximating the surface of the given object. The hidden surface algorithms that have evolved from the early stages of computer graphics have focused on scenes composed of polygons. Thus, the chief advantage of this scheme is that the hidden surface computations can be applied

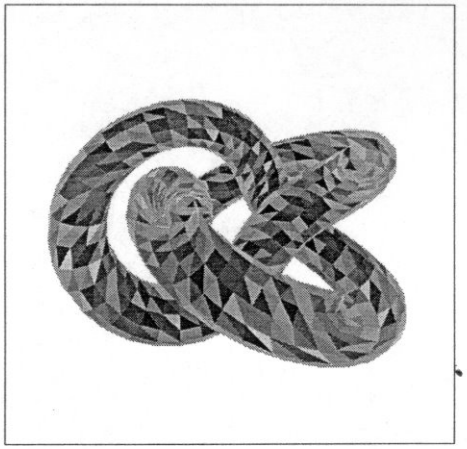


Plate 4. Knot approximated with polygons.

to polyhedral objects and curved objects uniformly. This works quite well for displaying many a number of curved surfaces because of the works of Gouraud [GOUR71] and Phong [BUIT75]. Plate 4 shows a three-dimensional knot. The triangles which make up its surface are deliberately shown to reveal the underlying polygonal structure. Plate 5 shows the same picture smoothly shaded using a breed between Gouraud and Phong shading techniques.

The output generated from the polygonal representation of the input can also be a set of polygons, with hidden polygons removed and partly visible polygons subdivided into smaller polygons to show only completely visible parts. The main CPU can compute the visible polygons and then the graphics processor can render them with or without smooth shading. The pictures in Plates 4 and 5 were generated in this way. The hidden surface calculations were executed on a VAX 11/750 and the resulting triangles were transmitted to an Ikonas graphics processor which rendered them.

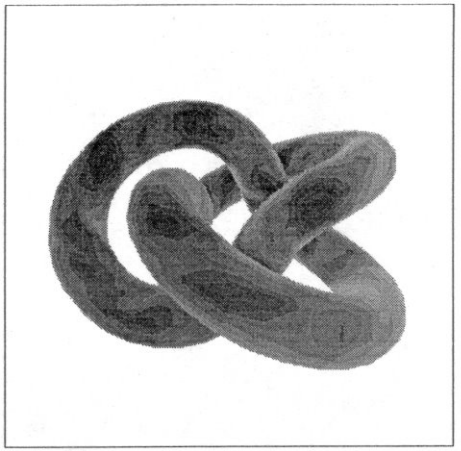


Plate 5. Same knot smoothly shaded.

The main disadvantage of this representation is that it requires lot of data to represent curved surfaces. The picture in Plate 6 required several thousands of triangles to represent the scene. Another disadvantage is that in spite of the smooth shading, the silhouettes of the objects still show pronounced polygonal edges (Plates 4 and 5). Above all, reflections in curved surfaces cannot be satisfactorily produced with this kind of representation either for input or output.

Closed-form Equations

For a surface, such as that of a sphere or a cone, the input can be represented directly by its analytic equation. Using this representation, an image of a sphere that closely resembles a realistic spherical ball can be produced (Plate 7). This form of representation is a must for obtaining satisfactory reflections of curved surfaces in curved surfaces.

The chief advantage of this method is that the output closely represents the input.

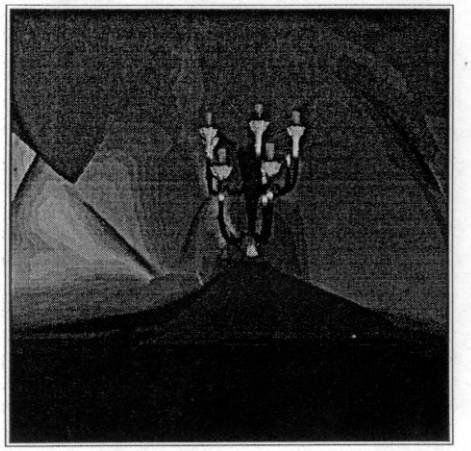


Plate 6. Cave with a candelabrum.

The major disadvantage may be that higher order equations cannot be rendered efficiently, and numerical methods may have to be adopted which may be slower. (This was perhaps one of the other reasons that lead authors to consider polygonal approximation). This is especially so if two or more surfaces in the scene intersect. We need to compute and show the intersection curve with effective hidden surface elimination.

Spline Approximation

There are no effective methods of handling surfaces of high order equations directly. Hence we need to sample points on the surfaces and fit spline patches to these points. This method, though may not be one hundred percent accurate, is much better than the polygonal representation, and is quite satisfactory. It is easier to render higher order equations using this method than using the equations directly. With fast subdivision techniques for spline patches [CATM74], it works out fairly cheap in terms of CPU requirements for rendering them. However, to render reflections faithfully it is

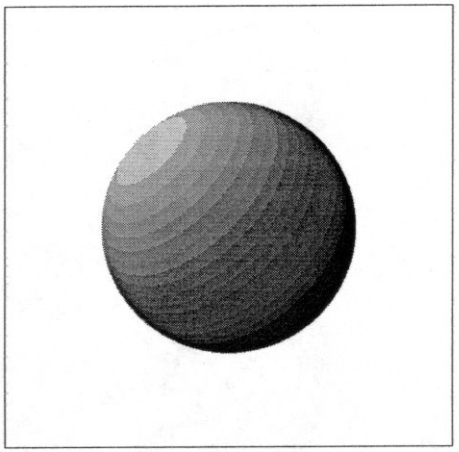


Plate 7. Smoothly shaded sphere.

necessary to use a breed between this and the previous representation. The equations are used for reflecting surfaces, and the spline representation is used for the reflected object surfaces. The actual method of rendering is outlined in a following section.

The output or image representation should be as compact as possible. One of the methods would be to define primitives that can be handled by a graphics device. The image of a sphere can be represented by a circle or an ellipse which is its projection on the image plane. The graphics processor can then render the image using interpolation schemes if necessary. As discussed in the previous chapter, we need to represent the reflections by their contours. Contours of reflections in curved surfaces cannot be found analytically (see appendix 3). We can compute sample points on the contour and represent them by these points. The points can then be joined by straight line segments – yielding a polygonal approximation, or we can fit a splinegon [SOUV86] to them. Another scheme would be to represent the output in terms of three dimensional

spline patches fitted to key points on the reflected surface. Both of these methods are satisfactory compared to the output of a ray-tracer which yields bit-maps. Bit-maps, as we know, are not device independent and resolution independent. Of course, the final output to be displayed on a raster device is a bit-map. We postpone converting to a bit-map as late as possible. This reduces communication bandwidth required between the main processor and the graphics processor. And it means less storage on the main processor's disk for storing away the computed image permanently. The disk I/O is one of the bottlenecks as was shown in [PAND85]. Thus, we are not reducing the response time by computing the picture afresh from the output representation every time we need to display it.

4.3 Low-level Algorithms

As mentioned in the previous chapter, analytical solutions for obtaining reflections in curved surfaces are not feasible. In this section we explore three alternate techniques to render a basic reflection of one object in another. We consider three algorithms:

[1] Contour Tracing Algorithm

[2] Numerical Algorithm

[3] Spline Approximation

4.3.1 Contour Tracing Algorithm

A contour can be described as the map of a function from an **n**-dimensional space to a **k**-dimensional space, where $\mathbf{n} > \mathbf{k}$. The contour of the reflection of one sphere in another can be treated as a map from $\mathbf{R^n}$ to $\mathbf{R^{n-1}}$. The following paragraphs outline how the contour of the reflection of a sphere in another can be generated using a general-purpose contour tracing algorithm [DOBK86].

The scope of this dissertation does not permit for a detailed description of the contour tracing algorithm. Briefly, a point on the contour is located and the algorithm proceeds by finding successive points on the contour. These points are found by taking

small steps along the tangent to the curve and searching in a direction perpendicular to the tangent to "get back" onto the curve. The algorithm described in [DOBK86] is robust in the sense that it does not accumulate round-off errors and has a well defined integer test for detecting loops.

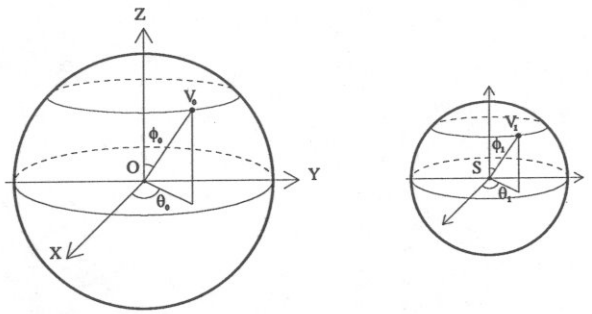


Fig. 4.1 Sphere S is to be reflected in O.

With the contour tracing algorithm we can obtain reflections in arbitrary surfaces, provided that we are able to describe the problem as a map from $\mathbf{R^n}$ to $\mathbf{R^{n-1}}$. To illustrate the technique, consider two spheres O and S as in Fig. 4.1. The sphere S is to be reflected in the sphere O. Assume that the sphere O is centered at the origin and the eye is at a given position E (not shown in Fig 4.1). Our problem space or parameter space will have a dimension of $\mathbf{n} = 4$: two for each sphere. The four independent parameters are the polar angles θ_0, θ_1, and azimuth angles ϕ_0 and ϕ_1. A point V_0 on the reflecting sphere O is parameterized by (θ_0,ϕ_0) and has cartesian coordinates (X_0, Y_0, Z_0). Similarly, a point V_1 on the reflected sphere S is parameterized by (θ_1,ϕ_1) and has cartesian coordinates (X_1, Y_1, Z_1). The solution space consists of three conditions:

$$(\vec{V_1} - \vec{V_0}) \perp (\vec{V_1} - \vec{S}) \tag{1}$$

$$det(\vec{V_0}, (\vec{V_1} - \vec{V_1}), (\vec{V_0} - \vec{E})) = 0 \tag{2}$$

$$\angle((\vec{V_1} - \vec{V_0}), \vec{V_0}) = \angle(\vec{V_0}, (\vec{V_0} - \vec{E})) \tag{3}$$

where $\vec{V_0}$, $\vec{V_1}$, $\vec{S}$ and $\vec{E}$ represent radius vectors from the origin O respectively to point V_0 on the reflecting sphere, point V_1 and center S of the reflected sphere S, and the eye position E. Condition 1 says that the reflected ray is tangential to the sphere S. Conditions 2 and 3 respectively define the first and second laws of reflection (see also appendices 2 and 3).

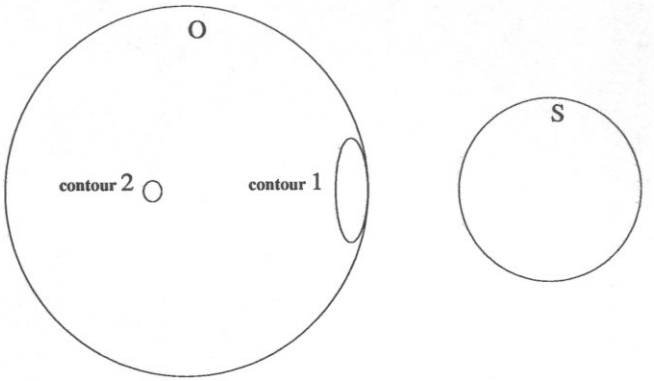


Fig. 4.2 Reflections obtained by contour tracing.

Thus the parameter space is of dimension 4 and the solution space is of dimension 3. The starting point on the contour can be located using a numerical method (see next section). Fig. 4.2 shows a picture obtained by using the above conditions as input to the contour tracing program. An orthogonal projection scheme was used. The two spheres project onto two circles O and S. Contour 1 represents the reflection of S in the convex (outer) surface of O. Contour 2 is the hypothetical reflection of S in the concave (inner) surface of O, which would have formed if the sphere O was translucent and effects of refraction could be neglected. The sphere O has unit radius and the

radius of S is 0.5 units. The eye position for the picture is at three units from the center of the sphere O in a direction perpendicular to the plane of projection.

The output of the contour tracing algorithm is a set of points on the contour. These points can be joined by straight line segments (as was done in Fig. 4.2). Alternatively, a bicubic spline can be fitted to pass through the points. One important observation is that the points generated by the contour tracer algorithm are device independent. The algorithm of [DOBK86] generates more points where the curvature of the contour is pronounced and fewer but enough of points where the contour is smoother.

4.3.2 Numerical Algorithm

The contour tracing algorithm described above is a general purpose contour tracer. It is powerful enough to compute general contours. But, it has certain overhead due to its general nature. If we are considering only a specific type of surface it may be worthwhile to find a special purpose contour tracer, geared to the surface in question, without the extra overhead. In this section we describe a numerical technique to generate specifically the reflection of one sphere in another.

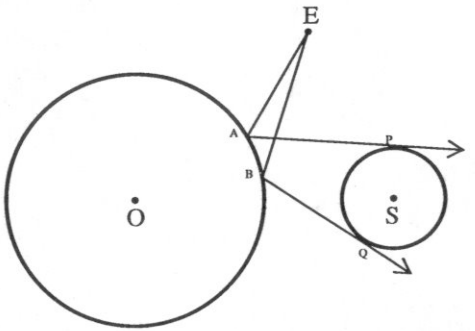


Fig. 4.3 A circle reflecting another.

First, we consider a two-dimensional version of the problem. The results can be easily extended to the three-dimensional case because of the first law of reflection, namely, "the incident ray, the reflected ray and the normal to the surface at the point of incidence, all lie in the same plane."

In Fig. 4.3 there are two circles centered at O and S. The eye is at the position E. The circle S is to be reflected in circle O. The problem is to find points A and B on the circumference of the circle O which represent the contour of the reflection of S in O. The ray EA (EB) is reflected by the circle O to the ray AP (BQ) which is tangential to the circle S. Appendix 2 deals with the difficulties of analytical approaches to solving this problem.

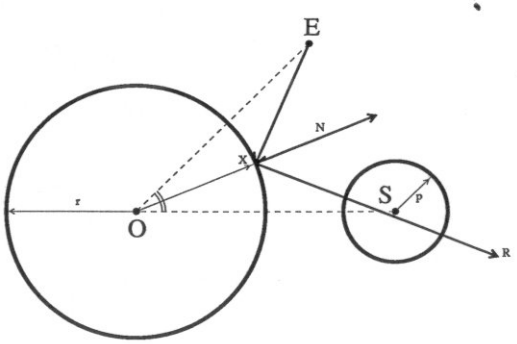


Fig. 4.4 Numerical Solution

To solve the problem numerically, we start out with an initial guess for the solution. For a naive guess we can take the point of intersection of the circle O with the line OX which bisects ∠EOS (Fig. 4.4). More intelligent guesses are possible; but, for illustration purposes, this guess is adequate. Consider the ray EX and the direction XR it takes after reflection in the circle O. We can compute the direction XR simply by using the reflection laws, since we know the normal at X (appendix 2). We compute the algebraic perpendicular distance from the point S to the line XR. For the

line XR to be tangential to the circle S, this distance should equal p, the radius of the circle S. (For the reflected ray corresponding to the point B in Fig. 4.3 this distance will be $-p$.) If the computed distance is not equal to p we make an adjustment to the current guess either by a binary search technique or by a root-finding method such as the Newton-Raphson method [MARO82]. The new guess yields us a point X′ and reflected ray X′R′. We iterate until the tangency condition is satisfied within the precision required. When this happens the solution will converge to the point A.

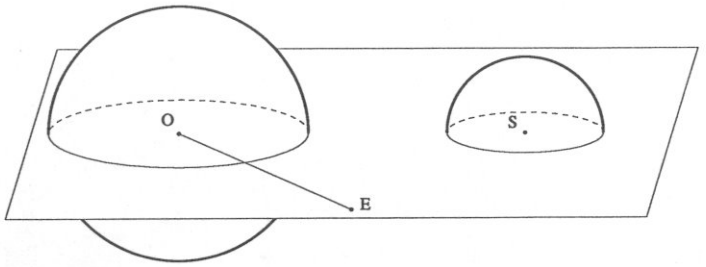


Fig. 4.5 Plane through the eye and the center of reflecting sphere O.

Now let us see how this extends to the three-dimensional case. We have spheres O and S and an eye-position E (Fig. 4.5). Consider the plane SEO. This plane intersects the spheres O and S in two circles. The normal to the reflecting sphere O at any point on the intersection circle also lies in the plane since the origin O is in the plane. If the rays that emanate from the eye lie in this plane, the reflected rays are restricted to lie in the same plane because of the first law of reflection. Thus the problem reduces to that in a plane and thus can be solved using the above method for the two-dimensional case. We thus get two points on the required contour. The plane above can be imagined to be rotated about a line through EO, sweeping the sphere S in small increments, until it becomes tangential to the sphere on either side. For each position of the plane we solve the two dimensional problem and get two more points on the contour. The collection of all such points yields the required contour.

Again, it is to be noted that this algorithm can be carried out in a device independent manner. Also, this yields a three dimensional contour as opposed to the projection of the contour in the contour tracing algorithm above. This has to be projected in a suitable manner to obtain the final picture.

4.3.3 Spline Approximation

Catmull introduced an algorithm for fast subdivision of splines [CATM74]. The motivation for this section comes from his work. Each object in the scene can be approximated by bicubic spline patches. A bicubic spline patch has a set of control points or vertices. The points[1] corresponding to the reflections of these vertices can be computed using a numerical technique very similar to that in the previous section. The reflections of the control points define another spline patch. We can render the latter spline patch using subdivision techniques [CATM74], to yield the image of the reflection.

Spline approximation yields pictures which look smoother than polygonal approximation. It has the further advantage that it can treat color, normals, x-y-z values and any other relevant parameters uniformly. One can obtain colors by subdivision to carry out a Gouraud [GOUR71] type shading and then use the normals to effect specular shading. The hybrid of the two shading schemes will yield a better picture. Similar techniques can be used for texture mapping.

We know that four points are sufficient to represent an ellipse by splines. Similarly six points are sufficient to represent a sphere with eight patches which share common vertices. As discussed above, we obtain the points corresponding to the reflection of the control points in the reflecting sphere. Then we form spline patches of these reflected control points. To obtain the reflections of the said points the reflecting sphere must be represented by its closed form equation. If the reflections obtained are not smooth enough it may be necessary to subdivide the patches before carrying out the reflection computations. This yields more key points and hence more accuracy.

[1] on the surface of the reflecting object

It is interesting to note that the spline approximation can also be used for previewing. Projections of the spline patches are splinegons [SOUV86]. For fast previewing we can show only the outlines of these splinegons rather than filling them. The output is also device independent.

The contour tracing algorithm and the numerical algorithm discussed above are both "boundary" algorithms in the sense that they obtain the boundary or contour of the reflection. The region enclosed by the contour has to be filled suitably to obtain the end result. These algorithms may be expected to run a little faster. In contrast, the spline approximation method finds many interior points and hence we would have a more faithful reproduction of the scene. One may imagine combining the two types of algorithms and deriving the best of both. For example, we can use the numerical algorithm to find the silhouette quickly and use only the visible spline patches to build up the interior.

All generalizations are false, including this one.
— Alexander Chase

Generalization is necessary to the advancement of knowledge;
but particularity is indispensable to the creations of the imagination.
— Lord Macaulay (1800-1859)

5

MODIFIED PAINTER'S ALGORITHM

Every good painter invents a new way of painting.
— Aldous Huxley (1809-1894)

High-level Algorithms

In the previous chapter we described three algorithms which showed how the basic reflection of one object in another can be generated. In this and the following three chapters we describe four high-level algorithms which make use of one or more low-level algorithms as a primitive to assemble mutual reflections of multiple objects and blend them together with hidden surfaces eliminated. The algorithms we are going to discuss are:

1. Modified Painter's Algorithm
2. Modified Z-buffer Algorithm
3. Scan Plane Method
4. Curvilinear Trapezoid Method

These four algorithms are characterized by the underlying hidden surface techniques used. They exploit various levels of coherence in the scene and have different time and space complexities. Furthermore, they shine under different operating environments and produce different forms of output. In this chapter we study the first of these.

5.1 Introduction

The painter's algorithm is a hidden surface algorithm which exploits object to object coherence. It is simple to conceive and easy to implement without fancy hardware requirements.

The painter's algorithm is a special case of a class of algorithms called list-priority algorithms [SUTH74]. It works only for scenes consisting of non-intersecting convex objects. There is a further condition that no three objects in the scene hide each other as, for example, in Fig. 5.1. These are not formidable restrictions since in many cases the non-convex objects in the scene can be broken down into a collection of convex objects. Intersecting objects can in a similar fashion be broken down into

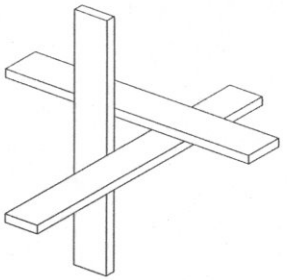


Fig. 5.1 Three convex objects cyclically occluding each other.

non-intersecting convex objects. Likewise, if three or more objects hide each other, they can be decomposed into smaller objects without leading to any ambiguity. The latter needs to be done only once per scene, as a pre-process independent of the view point. If the scene consists entirely of non-intersecting spheres the ambiguity problem does not arise since no three such spheres can hide each other in any position.

The technique underlying the painter's algorithm is quite similar to that employed by oil painting artists [ROGE85, HEAR86]. In creating the art work the artist proceeds by first painting the entire canvas with the background color. Then the most distant objects are painted, followed by painting over them the nearer objects in a back to front order. Each new object painted covers the objects behind it in part or full.

The painter's algorithm, as referred to in computer graphics, follows a very similar procedure to produce a view of a scene consisting of non-intersecting convex objects. The display screen can be thought of as an electronic canvas on which Mr. Computer, the artist, paints the picture. The objects farther away from the view point are painted first and then the nearer objects are painted in order. As mentioned above, the algorithm is a special case of list-priority algorithms. In general objects farther away from the viewpoint have a lower priority than that of objects nearer to the viewpoint. A list is formed of objects sorted according to their relative priorities. The objects

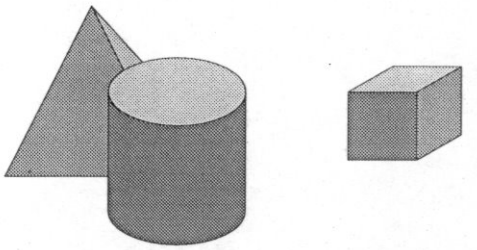


Fig. 5.2 The cube can be drawn independent of the other objects.

are then taken one at a time from the list in reverse priority order and painted onto the screen. The hidden surface removal is automatic since the objects closer to the viewpoint overwrite the ones farther away and thus hide the invisible parts of those objects. The ordering induced by the priority list is only partial since the objects that do not overlap for a given view of the scene can be painted in any order. For example, in Fig. 5.2 the cylinder hides the tetrahedron, but the cube does not hide any other objects and is not hidden by any. Hence the tetrahedron has to be painted before the cylinder, whereas the cube can be painted anytime independent of the tetrahedron and the cylinder.

When only spheres are involved in the scene, determining the object priorities becomes fairly simple. All we need to do is to determine the length of the tangent from the viewpoint to each of the spheres and sort the spheres based on this length. The sorted list thus obtained corresponds to the priority scheme as described above. For orthogonal projection the viewpoint is effectively at infinity. In this case we just need to sort the spheres according to their shortest distances from the view-plane in the direction of viewing. In simple terms, if the viewing direction is along the z-axis, the length to be considered is the z-coordinate of the center of each sphere less its radius.

The painter's algorithm is both an object space and an image space algorithm. The sorting of the objects takes place in object space without reference to the resolution and other physical characteristics of the device. However, the actual hidden surface removal is carried out by painting the objects in the image space from back to front.

The algorithm is appealing because it is simple in concept and easy to implement. Another advantage is that transparency can be easily incorporated by partially overwriting the background objects, instead of complete replacement. It is efficient when the scene consists of relatively few objects since it does not take much time to sort a small number of objects[1]. Object to object clipping is not required and thus clipping time is saved. The objects are rendered one at a time and all the objects are rendered. When there are only a few objects in the scene, the expected overlap may be small and there is not much work lost in rendering all the objects. And, since the objects are rendered one at a time, they can be rendered on a fast graphics processor with perhaps limited space resources [PAND85]. If the objects themselves are defined as primitives –e.g. spheres– the amount of data to be transferred to the graphics processor will be small. This is helpful when the communication between the host and the graphics processor is a bottleneck [PAND85].

On the other hand, if there are too many objects in the scene, the sorting process involved may be a bottleneck. Also, when the number of objects is large – thus increasing the depth complexity of the scene – many of the objects may not contribute significantly to the final image and thus the work done in rendering them goes to waste. Thus the algorithm will not be in tune with the complexity of the picture generated.

It is not a formidable restriction to keep the number of reflecting objects in a scene small. On the other hand, it is better to limit it, since, if all the objects in the scene are reflective, it may result in a very complicated picture. Indeed, the complexity of

[1] Actually breaking up ambiguities could be the bottleneck, but it is unlikely to exist if the number of objects in the scene is small. Furthermore, as we stated earlier, if our scenes consist entirely of non-intersecting spheres, there can be no ambiguities at all.

the picture may overshadow the underlying model of the scene and may fail to convey what is being depicted to the onlooker. For example, the picture in Plate 1 is described as just five spheres reflecting each other, but the image is quite complex.

In the following paragraphs we extend the painter's algorithm to handle reflections. First we discuss why an extension is required and then proceed to detail the modified algorithm. To keep the description of the algorithm simple, we assume that the scene consists only of spheres. This need not be the case in general.

Conventional Painter's Algorithm is Inadequate

When a sphere is rendered onto the display screen, the contribution of the color of each point on the sphere replaces the corresponding pixel value of the frame buffer. When a reflection in this sphere is rendered, the contribution of the reflection does not replace the pixel value (which represents the base sphere) but adds onto it. For example in Fig. 5.3a the spheres A and B were rendered onto an initially empty frame buffer. In Fig. 5.3b the reflection b of B in A is rendered which adds to the image of A.

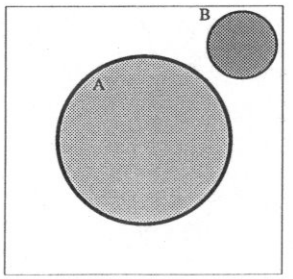


Fig. 5.3a

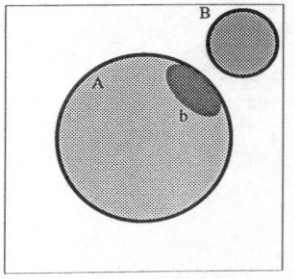


Fig. 5.3b

The painter's algorithm as described above cannot handle reflections because of the following complication. Reflections need to be added onto the images of the base spheres. When there is a need to overwrite a reflection by another reflection of the same level in the same base sphere, there is no way to extract the written value of the hidden reflection. This causes problems. For example, in Fig. 5.4a A is the base sphere which was rendered first. In Fig. 5.4b the reflection b of B (not shown) in A was rendered and merged with A. The shaded area in Fig. 5.4b now represents A + b, the base sphere A and b, the added component of the reflection of B. Then when the same level reflection c of C in A is rendered (Fig. 5.4c) and merged with the existing picture in the frame buffer the common area in the overlap region of b and c (shaded dark) will now represent A + b + c. But only one of the reflections is visible in this region. Assuming the reflection of c is visible, the shaded area should then represent A + c and not A + b + c. In order to do this we need to somehow extract A from A + b (assuming we know the area of overlap) and then add c. This is not possible with a simple painter's algorithm.

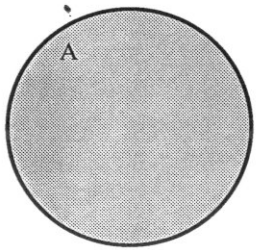


Fig. 5.4a

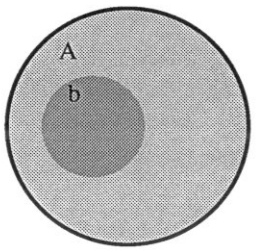


Fig. 5.4b

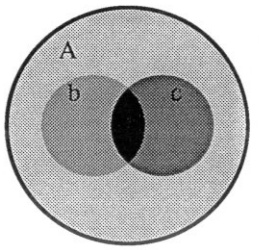


Fig. 5.4c

5.2 Modified Painter's Algorithm

We solve the above problem by modifying the painter's algorithm. To simplify the discussion further, let us term the base spheres as reflections of level zero. To render a k-th level reflection we proceed as follows. First clear the area of the display screen occupied by the reflection. This can easily be done since we can compute the contour of the reflection as outlined in chapter 3. Then recursively render all possible reflections of the next higher level (k+1). Then add in the color due to the current k-th reflection.

To illustrate, let us consider how this works as applied to the above example. Let us assume that the base sphere A is to be rendered with the reflections b and c, and there are no higher level reflections. To render A which is of reflection level 0, we first clear the area occupied by A. Assuming that we started out with an empty frame buffer, Fig.5.5a depicts the situation. The dotted outline of A is not rendered but is shown in the Fig. 5.5a only to indicate that the area enclosed was cleared. We have to render next higher level reflections in A which are b and c. The reflection c has higher priority over b and hence b should be rendered first. To render b, we clear the area represented by it. The situation is as shown in Fig. 5.5b. The frame buffer is still empty. Since there are no higher level reflections, the color contribution due to b is simply added in. This results in Fig. 5.5c. Now to render c, we first clear the area occupied by c on the screen. This clears the overlap region of b and c (Fig. 5.5d). Then the contribution of c itself is added in (Fig. 5.5e). In the overlap region of b and c we now correctly have just c. Now, the higher order reflections b and c finished, we recurse back to the top level and add in the color due to A itself. The final picture (Fig. 5.5f) has the correct blend of colors in all regions.

The careful reader can observe that the recursion step of the algorithm can be applied to both the reflections and the base spheres in a uniform way. Of course, the lower level rendering algorithms may be (and should be) different for at least the base spheres as a class and the higher order reflections as another.

The above represents the basic step of the modified painter's algorithm. It ap-

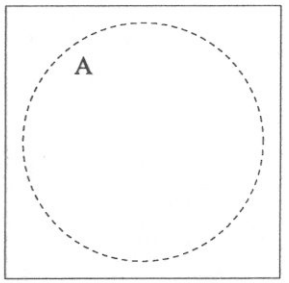


Fig. 5.5a The area occupied by base sphere A is cleared first.

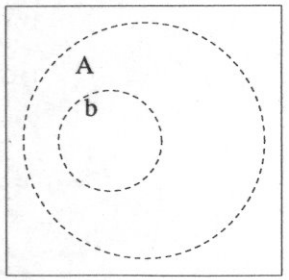


Fig. 5.5b The area occupied by reflection b of B in A is cleared.

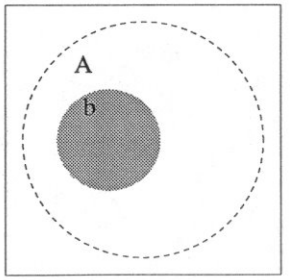


Fig. 5.5c The reflection b of B in A is rendered.

pears that painters some time use part of this modified technique [LI87]. That is, when painting a foreground object, the area represented by it is first cleared to the background color and the object then painted on.

The following section gives the overall description of the algorithm.

5.3 Detailed Description of the Algorithm

Given **n** non-intersecting spheres with reflecting surfaces, the aim is to produce a two-dimensional projection of the scene with **d** levels of reflections.

For the sake of uniformity, let us term the base spheres as reflections of level 0.

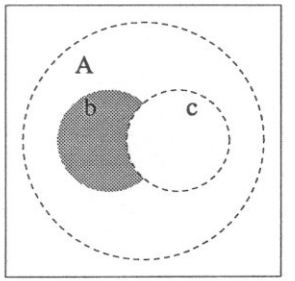


Fig. 5.5d The area occupied by the reflection c of C in A is cleared. This clears the overlap region.

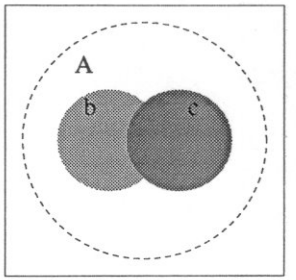


Fig. 5.5e The reflection c of C in A is rendered. The overlap region correctly has only c.

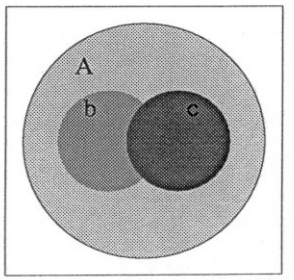


Fig. 5.5f The base sphere A is now rendered and added on. The overlap region has the correct blend of colors.

The algorithm starts out first by sorting all the base spheres according to their priority as in the classical scheme. Then the spheres (and the reflections in them) are rendered in a back to front order. The rendering is different from the classical algorithm as outlined above in that the spheres and the reflections in them are rendered recursively. The following pseudocode outlines the recursive rendering procedure.

render(SPHERE, REFLNLIST, **k**)

/* SPHERE is the sphere being reflected; REFLNLIST is the ordered list of spheres in which the sphere SPHERE gets reflected. At the top recursion level REFLNLIST is empty. **k** is the reflection level and also equals the number of elements in the REFLNLIST

*/

[1] Clear the area represented by the **k**-th reflection of the sphere SPHERE in the spheres in the REFLNLIST.

[2] If **k** equals the maximum level of reflection desired, proceed to step 5.

[3] Sort all other spheres according to the distance from the center of the sphere SPHERE and form a list.

[4] For each sphere in the above sorted list, recursively

render(sphere, REFLNLIST+SPHERE, **k**+1)

[5] Add in the color of the **k**-th reflection of the sphere SPHERE.

The algorithm terminates when all the base spheres are rendered.

The recursive rendering procedure above seems to indicate that there is a need to sort $O(\mathbf{n})$ spheres at each level of the recursion. By clever pre-processing this may be avoided. Observe that the sorting is based on the distance from the center of the particular sphere in consideration. As a preprocessing step to the algorithm, a look-up table is created. The table consists of **n** entries, one for each sphere. Corresponding to a given sphere, the entry in the table is a list of all other spheres sorted based on the distance from the center of that sphere. Once the look-up table is set up, the table entries can be retrieved to get the required sorted list in constant time.

The elegance of the modified painter's algorithm lies in that it can be implemented using almost any raster device without any fancy additional hardware.

5.4 Complexity Analysis

5.4.1 Time Complexity

There are $O(\mathbf{n^d})$ reflections and each reflection is rendered exactly once. Thus the time require for rendering is $O(\mathbf{n^d})$. We need not count the time it takes to merge the higher level reflections with the lower level ones since it is accounted for by step 5 of the recursive rendering procedure. There is a sorting step required for each of the reflections of level $(\mathbf{d-1})$ or less. This sorting step takes $O(\mathbf{n}\ \textit{log}\ \mathbf{n})$ time. Thus the effective time complexity of the algorithm is $O(\mathbf{n^d \bullet n}\ \textit{log}\ \mathbf{n})$. Thus the sorting supersedes the rendering time in complexity.

With the look-up table approach, as discussed at the end of the previous section, the sorting step need not be carried out during the recursive rendering process. To look for an entry in the table takes only constant time. The preprocessing step for building the look-up table takes $O(\mathbf{n^2}\ \textit{log}\ \mathbf{n})$ time, as it involves sorting $O(\mathbf{n})$ spheres **n** times. Thus, effective time complexity reduces to $O(\mathbf{n^d + n^2}\ \textit{log}\ \mathbf{n})$. The first term in this expression supersedes the second for values of **d** greater than 3. Thus for large values of d the time complexity is just $O(\mathbf{n^d})$.

5.4.2 Space Complexity

The algorithm needs to recurse **d** levels. At each recursion level it may need to store the sorted order of the $O(\mathbf{n})$ spheres. Hence the space complexity is $O(\mathbf{n \bullet d})$.

For the look-up table, storage required is for **n** lists each of length $O(\mathbf{n})$. This drives the space complexity to be $O(\mathbf{n^2})$.

The above description of the algorithm assumes the availability of primitives for drawing the basic spheres and the reflections of any level.

5.5 Parallelism

The algorithm seems to be inherently sequential since the primitives are generated and rendered one after another in a sequence. Nonetheless, there are two methods of

achieving parallelism. The first of these methods is a trivial one that can be applied to almost any graphics algorithm, and the second exploits the way in which the modified painter's algorithm works.

The idea behind the first method is to divide the display screen into a number of smaller regions and hand each region to a different processor. Each processor runs the modified painter's algorithm on the entire list of input spheres. Many of the spheres may lie entirely in only one region. A processor need not render the spheres and reflections which lie entirely outside its region. Spheres which partly overlap a region need to be clipped. Clever load balancing may be required if the densities of the spheres across the regions are drastically different. For an average scene we can assume that each region has approximately the same number of spheres.

Another possible way of parallelizing the modified painter's algorithm is suggested by observing the way in which the primitives are rendered. It appears that the rendering follows a tree like path. At the root of the tree are the base spheres. The children of the root are the first level reflections; the descendants of the first level reflections are the second level reflections, and so on. The first level reflections are to be resolved before the base sphere is rendered. In a similar fashion, all the second level reflections in a given first level reflection have to resolved before the latter. This pattern extends to the higher level reflections.

With the above observation, parallelism can be achieved as follows. We arrange processors in a tree like form closely resembling the rendering tree. Each processor has its own frame buffer. The root processor renders the base sphere. In parallel, each of its children renders a first level reflection, each of the second level children renders a second level reflection, and so on. The results are then merged into a single frame buffer in "bottom up" fashion.

The above process is further clarified by an example. Suppose that there are three spheres A,B and C. Assume that the number of reflection levels required is two. Then we need a tree structure with a root, two children and four leaves as in Fig. 5.6.

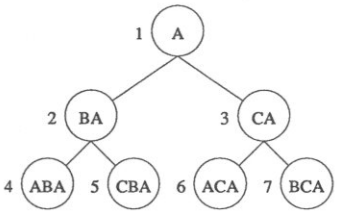


Fig. 5.6 Tree structure of the algorithm.

The processor at the root node 1 renders the base sphere A into its own frame buffer. In parallel the children nodes 2 and 3 respectively render the first level reflections in A, namely, BA and CA. The leaves (4,5) and (6,7) render the second level reflections, namely (ABA,CBA) and (ACA,BCA). Then the reflections ABA and CBA are merged into one frame buffer. During the merging process, hidden surface elimination takes place according to the priority scheme. The contents of the resulting frame buffer are added onto the contents of the frame buffer of the processor 2. The reflections ACA and BCA are similarly merged into another frame buffer and added onto the the frame buffer of processor 3. The frame buffers of processors 2 and 3 are then merged and added onto that of processor 1. The frame buffer of the root node 1 acts as the display frame buffer or the master frame buffer. After the merger takes place in the display frame buffer, the processors are ready to render the next base sphere (B or C) and the reflections in it. It should be emphasized that before proceeding to the next sphere, the frame buffers of all the nodes, except that of the root node, have to cleared.

The above description of the parallelized version of the modified painter's algorithm follows the logic of sequential version very closely. The processors need not be hardwired in any fashion. The processors can be reconfigured to handle any number of spheres and any reflection level. This may again require careful load balancing and scheduling techniques. Furthermore, since the reflections in spheres get smaller and smaller with each level, slower processors and smaller frame buffers can be assigned to

the processors towards the base of the tree, whereas processors at the top of the tree need to be faster and should have larger frame buffers.

5.6 Remarks

The modified painter's algorithm is simple in concept, easy to implement, and requires no special purpose hardware. It is suitable for a quick view on a raster display device when the scene consists of few objects and the depth complexity is small.

To iterate is human; to recurse, divine.
— L. Peter Deutsch

6

MODIFIED Z-BUFFER METHOD

A moment's insight is sometimes worth a life's experience.
— Oliver Wendell Holmes (1809-1894)

6.1 Introduction

Z-buffering is a hidden surface removal algorithm introduced by Catmull [CATM74]. It is characterized by simplicity and ease of implementation. Today it is available in hardware/firmware [IRIS86, ADAG82] for fast routine graphics. Z-buffering is an image space algorithm in that the hidden surface elimination takes place after the objects in the scene are projected and rendered onto the image plane. The chief appeal of the algorithm is its linear time and space complexity.

The image displayed on a raster device is stored in a digital memory called the *frame buffer*. This buffer is arranged as a two dimensional array addressed directly in (integer) x-y or pixel coordinates. Each memory location in the frame buffer corresponds to a rectangular area on the physical display device called a pixel. The value stored in the memory is a measure of the intensity to be displayed at the corresponding pixel location. To display an object on a raster, it is projected onto a plane representing the display device and the pixels in the projected area are suitably modified to represent the color and intensity of the object at those pixels. This process is generally known as scan conversion. The frame buffer is accessed in its entirety once per video refresh cycle by the display circuitry. The computer or the graphics processor accesses it only when it needs to update or alter the picture.

The display device presents a two-dimensional surface for viewing the picture. To represent a three-dimensional scene effectively the objects in the scene are to be projected onto the display screen with hidden surfaces eliminated. One of the methods for achieving this is to sort the objects in the scene according to their distance from the eye and draw them back to front. Sorting can be a slow process when there are many objects in a scene.

Z-buffering is a method for avoiding the sorting process. It requires another memory buffer of the same size and configuration as that of the frame buffer and is

addressed by the same pixel coordinate system. This buffer is called the *z-buffer* or the depth buffer. The z-value is the distance between the viewer and the object in the scene. Initially the frame buffer is cleared to the background color and every location in the z-buffer is initialized to the largest value of z possible.

When a pixel value needs to be modified to represent a point of an object in the scene, the z value of the point is compared with the z value stored in the corresponding location in the z-buffer. If the stored value is greater, the pixel is updated with the color and intensity of the point and the z-buffer location is updated with the new z value. The effect of these operations is to yield a final picture with all the hidden surfaces removed. This is easy to see because if a point P hides another point Q, P will overwrite Q if P was rendered later than Q since the z value for P will be smaller than that for Q. On the other hand if P was rendered first, Q will not overwrite it for the same reason.

Z-buffering has the disadvantage that it requires large amounts of memory for effective hidden surface elimination. However, this memory need not be accessed by the video display circuitry, and hence the main CPU need not contend for its access. Also, anti-aliasing is difficult with z-buffering. Further processing along the lines of a-buffering [CARP84] is required to incorporate anti-aliasing. Despite these disadvantages it is simple to implement in hardware/firmware as it is executed at the pixel level. The main advantage which stems from it is that the objects need not be sorted and hence can be drawn in any order. In the light of the graphics workstation model as discussed in chapter 4, this gives the additional advantage that the graphics processor can handle one primitive at a time. Avoiding sorting, the time complexity of the algorithm is linear in the number of objects in the scene. Furthermore, since each object can be drawn independent of the others, parallelism is easier to achieve. This prompts us to investigate the extension of this algorithm to render reflections.

Deficiency of the Conventional Method

When a sphere is rendered into the frame buffer, the contribution of the color of each point on the sphere replaces the corresponding pixel value of the frame buffer. When a reflection in this sphere is rendered, the contribution of the reflection does not replace the pixel value (which represents the base sphere) but adds onto it. For example in Fig. 6.1a the spheres A and B were rendered onto an initially empty frame buffer. In Fig. 6.1b the reflection b of B in A is rendered which adds to the image of A.

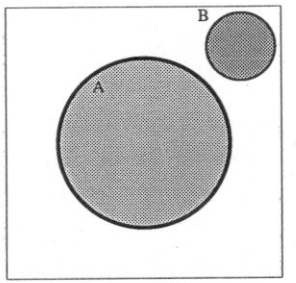


Fig. 6.1a

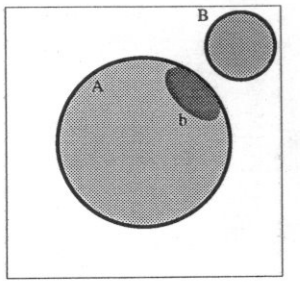


Fig. 6.1b

It is easy to see why the conventional z-buffer is inadequate for rendering scenes with reflections. The major problems are:

1. There is no meaningful way in which we can assign unique z values to the reflections which are virtual (for convex non-intersecting spheres).
2. Reflections are added onto the images of the base spheres. When there is a need to overwrite a reflection by another reflection of the same level in the same base sphere, there is no way to extract the written value of the hidden reflection. This causes problems. For example, in Fig. 6.2a A is the base sphere which was rendered first. In Fig. 6.2b the reflection b of B (not shown) in A was rendered and merged with A (assuming that

we somehow overcame problem 1 above). The shaded area in Fig. 6.2b now represents A + b, the base sphere A and b, the added component of the reflection of B. Then when the same level reflection c of C in A is rendered (Fig. 6.2c) and merged with the existing picture in the frame buffer the common area in the overlap region of b and c (shaded dark) will now represent A + b + c. But only one of the reflections is visible in this region. Assuming the reflection of c is visible, the shaded area should then represent A + c and not A + b + c. In order to do this we need to somehow extract A from A + b (assuming we know the area of overlap) and then add c. This is not possible with a simple z-buffer.

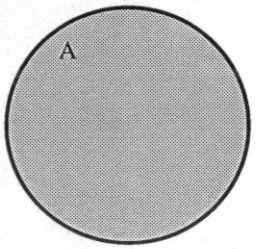


Fig. 6.2a

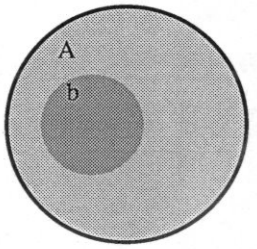


Fig. 6.2b

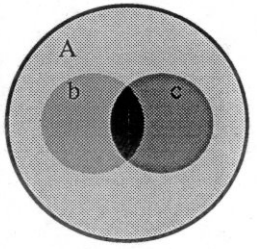


Fig. 6.2c

6.2 Modified Z-buffer Method

To avoid the above problems, what we do is to have additional z-buffers and frame buffers. The base sphere A is first rendered into the first frame buffer (Fig. 6.3a). The z-buffer is updated in the conventional way. The reflection b of B in A is rendered into an initialized second frame buffer z-buffer combination (Fig. 6.3b). The z values used for rendering b is taken as the distance from the center of the sphere A rather than from the eye. The reflection c of C in A is rendered into this second frame buffer containing b. Since C is closer to A than B the image c replaces b in the overlap

region, using the z values from the center of A as before. There is no merging in the second frame buffer – just overwriting. The second frame buffer with the reflections b and c is then merged into the first frame buffer adding the result to the base sphere A. The overlap region now correctly represents A + c. It is trivial to prove that this process yields a valid view of the scene.

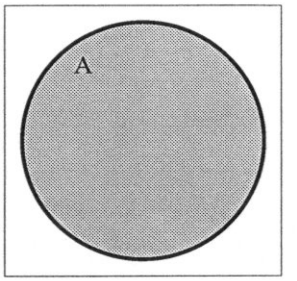


Fig. 6.3a Frame Buffer 1. The base sphere A is rendered first.

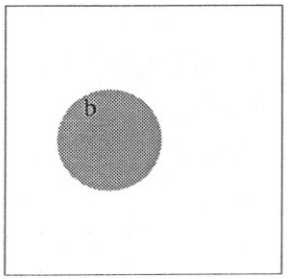


Fig. 6.3b Frame Buffer 2. The reflection b of B in A is rendered.

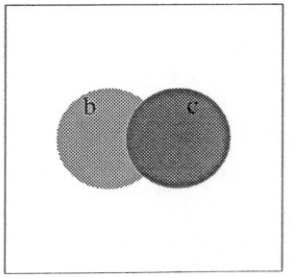


Fig. 6.3c Frame Buffer 2. The reflection c of C in A is rendered. This replaces b in the overlap region.

The above represents the basic step of the modified z-buffer algorithm. Note that the reflections in a sphere lie entirely within its projection on the frame buffer. Thus,

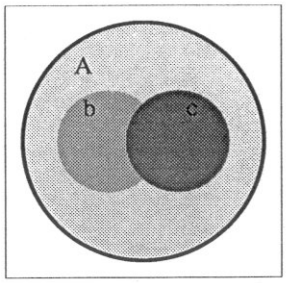


Fig. 6.3d Frame Buffer 1. The results of Frame Buffer 2 are added to the base sphere A in Frame Buffer 1. The overlap region now correctly has A + c.

the second frame buffer and z-buffer only need be as large as the projection of the base sphere being rendered. We will discuss the requirements for the second frame buffer and z-buffer at the end of the next section.

6.3 Detailed Description of the Algorithm

Given **n** non-intersecting spheres with reflecting surfaces, the aim is to produce a two-dimensional projection of the scene with **d** levels of reflections.

For the sake of uniformity, let us term the base spheres as reflections of level 0. For a reflection of level **k**, the algorithm proceeds by first rendering the $(\mathbf{k}+1)$-th level reflections in an appropriately sized frame buffer and z-buffer combination. The z values used for the $(\mathbf{k}+1)$-th reflections are distances from the center of the sphere at the **k**-th level. These $(\mathbf{k}+1)$-th reflections are simply added to the **k**-th reflection and the z values for the former are just discarded. Once this merger takes place, the $(\mathbf{k}+1)$-th reflections are no longer needed.

Thus, we start by rendering the **d**-th level reflections on one of the spheres and build it up to the 0-th level. We do this for every sphere. When all the 0-th level reflections have been rendered the algorithm terminates. The rendering is recursive

as in the case of the modified painter's algorithm (chapter 5), but is simplified further because of the absence of sorting.

Analysis shows that we need at most three physical z-buffers and frame buffers. We need one to accumulate the overall picture, i.e., the 0 level reflections. One buffer is needed each for the **k**-th level and $(\mathbf{k}+1)$-th level reflections. These frame buffers can be as large as the screen buffer or can be of smaller sizes. This is because the reflections are always smaller than the reflecting spheres. Thus, the smaller frame buffers can be allocated and de-allocated as required. It may be significant to note that the secondary frame buffers and z buffers need not contend with the display frame buffer for bus access. They need to exist only in the memory accessed by the graphics processor.

In order for the algorithm to be efficient, we require that the following operations can be carried out as fast as possible.

1. Clear an area of the frame buffer or the z-buffer.
2. Merge two frame buffers, with some arbitrary functions applied.

With the emergence of special purpose graphics processors coming into the market [ASAL86, CARI86, SHIR86] this is very much in sight. For example, clearing the area of a buffer can be achieved at video rates. There are existing graphics systems [ADAG82] capable of doing this, though they can handle only rectangular areas. Merging two frame buffers or parts of the same frame buffer has been implemented by a process called *bitblt*. Again, the current implementations handle rectangular areas only and the functions available are restricted to boolean operations.

6.4 Complexity Analysis

6.4.1 Time Complexity

There are $O(\mathbf{n^d})$ reflections and each reflection is rendered exactly once. There is a sub-linear amount of work done in merging the higher level reflections with the lower

level ones. Thus, the time complexity of the algorithm is $O(n^d)$ which is simply linear in the output size.

6.4.2 Space Complexity

At any given time we need to recurse far enough to get the parameters of at most the d-th level reflection. This may require a build up of a recursion tree that is **d-1** deep. Hence the space complexity is just $O(d + n)$.

Thus the space requirements of the algorithm are very minimal and the time complexity is linear. Hence this algorithm seems to be very attractive. However, these results need to be supported by evidence of practical implementation as the constants involved may be quite large. This is because we need to render each and every reflection.

The above description of the algorithm assumes that we have primitives for drawing the basic spheres and the reflections of any level.

6.5 Parallelism

There are two inherent ways the algorithm is amenable to parallel implementation:

1. When drawing the k-th level reflections, each reflection can be assigned to a different processor and these processors can render the reflections in parallel. A controlling processor then merges the reflections. This assumes that all the processors can access the frame buffer and the z-buffers simultaneously.
2. Another approach would be to give each processor its own memory in which it renders a given sphere and all the reflections in that sphere. The final results are then accumulated in a frame buffer. If the projections of the spheres are small enough, this approach is feasible as the individual frame buffers that each of the processors need to have will be quite small. This is usually the case with molecular modeling where there are many spheres of small sizes. For this kind of parallelism to be useful, the communication time for transmitting the frame buffers should not be significant compared to the rendering time. This can be true if the number of spheres is large or the reflection level is large. In either case the number of reflections to be rendered per sphere will be large and each processor will then spend considerable amount of time rendering compared to the time for communication.

In addition to the above two methods, one of the methods outlined in the previous chapter also works. This method is the division of the display screen into regions and handing each processor a separate region. As mentioned earlier, it does not take advantage of the special structure of the algorithm. Hence one of the above two methods may be more suitable.

6.6 Remarks

The modified z-buffer algorithm described in this chapter is an image space algorithm. The output of the algorithm is a bitmap. It requires extra memory for the z-buffer to carry out effective hidden surface elimination. It has minimal time and space complexity. It is fairly easy to implement. Please see [PARK80] for a further discussion of architectures for parallel implementation of the algorithm.

7

SCAN PLANE ALGORITHM

··· like the statistician who was drowned in a lake of average depth six inches.

— Anonymous

7.1 Introduction

Raster devices such as the raster graphics display terminals and printers often obtain their input and update their output in a scan line by scan line form following a certain repeatable pattern. The pattern is usually to sweep from top to bottom generating each scan line, and within each scan line the horizontal sweep is from left to right. Consequently the memory buffer used for holding the input data is arranged such that there is a one-to-one correspondence between the memory architecture and the display screen. The memory is usually cheaper to access in blocks of scan lines. In such a case, it is worthwhile to generate the graphics data in scan line order.

Especially in the case of printers, which are relatively slow devices, it is helpful and, in fact, necessary that the output comes in a scan line fashion since the printer head or the paper can not move in an arbitrary fashion. Hence we need scan line or scan plane algorithms. With present day technology, some color printers take several minutes to print a color picture and may take several passes for each of the primary colors. It is necessary to have a microcomputer supervising such printers all the time. Cost considerations for the dedicated microcomputer may limit its memory and CPU capabilities. Reliability is important here since, if there is trouble during the printing process, the printing will have to be started all over again. Thus, reliability is a tradeoff for complex CPU and memory functions. With this scenario, we need an algorithm that is simple, does not take too much memory, and generates its output in a scan line fashion. For the printers, the speed of the algorithm is not critical, just enough to be able to keep in pace with the printer. But for photographic printing and raster displays the algorithm should yield itself to implementation on a suitably fast scale. Such an algorithm is a scan line or scan plane algorithm. The output is generated in the same order as it is utilized for printing.

It is also to be noted that the output is to be generated with hidden surfaces already eliminated as opposed to the painter's algorithm or the z-buffer algorithm where the physical image space is used to eliminate the hidden surfaces. Devices

which write directly onto photographic film write each pixel in an irreversible way. Thus it is desirable to have each pixel written only once. Otherwise, a pixel written more than once will appear brighter in comparison with the other pixels. In a printing process one cannot afford to waste ink for overwriting pixels or time in moving the printing head or the paper to achieve the hidden surface elimination.

The scan line algorithm generates its output in a scan line order. At each scan line the algorithm maintains information about all the objects in the scene which intersect the scan line and hence the algorithm can effectively carry out the hidden surface elimination. In this chapter we describe how the scan line algorithm can be extended with clever modifications to handle reflections. We shall use spheres as basic objects to demonstrate the techniques involved but it is not a restriction of the algorithm.

7.2 Scan Plane Algorithm

The traditional scan line algorithm used for rendering scenes consisting of polygons [SUTH74] has been extended to handle molecular models which are represented as a collection of opaque spheres [PORT79]. The spheres are sorted according to their maximum Y and grouped into lists of spheres which start at the same Y scan line. Then as each scan line is processed, the spheres active at the scan line are sorted back to front and the visible sections of spheres are drawn. As we proceed to the next scan line, new spheres may become active and old spheres my drop out. Thus we update the active list at every scan line. Actually, making use of scan line coherence much of the sorting and book keeping tasks may be reduced.

When rendering spheres with reflections we should give careful consideration to the reflections. The reflections exist only within the silhouette of the reflecting base objects. The reflections add onto the colors of the base objects. In a sense, reflections act as objects in that they hide parts of other reflections. But they don't hide the base spheres in which they form. Also, if possible, we do not want to store all the reflections. There are $O(\mathbf{n}^{\mathbf{d}})$ reflections and storing them all at once in the computer memory may be costly. We avoid this latter problem in the following way. Whenever

a sphere becomes active, we generate a list of all the next level reflections in it and merge them with the Y-bucket list. The same thing happens when a reflection becomes active. All the next level reflections in it are added onto the list.

The following section gives the formal description of the algorithm.

7.3 Detailed Description of the Algorithm

Given **n** non-intersecting spheres with reflecting surfaces, the aim is to produce a two-dimensional projection of the scene with **d** levels of reflections.

For the sake of uniformity, let us term the base spheres as reflections of level 0. First bucket-sort all the base spheres based on their maximum Y value. In each bucket we sort them from left to right based on their X-intersections with the scan line. We start with the first non-empty bucket and make the spheres in the bucket active. When each sphere changes from being inactive to active, all the next level reflections in it are added onto the Y-buckets. If any of the reflections were added to the current bucket, we must add them to the current active list. To keep the active list sorted in X, we compute the X-intersections of the reflections just added and merge them into the active list based on these intersections. For the reflections added to the current active list, their next level reflections have to be added to the buckets again and the process is repeated until either no reflection is added to the current bucket or the maximum number of reflection levels is reached, whichever happens first.

At the current scan line we keep a list of sorted boundary points. As we process the scan lines in order, we need to change the sort order only when a new sphere/reflection changes state from being active to inactive or vice versa or edge crossings occur. By making clever use of this fact we can save much of the time required to keep the lists sorted and other book keeping work. It would be nice if we can eliminate reflections or spheres that are not visible at all. This would need additional processing which may not be justifiable.

We need to sort the spheres according to their depth within each scan line. As

noted earlier, this sorting need not be done for every scan line. The sorting order changes only when edge crossings occur or reflections change status from being active to inactive or vice versa. Because of this, two successive scan lines will have a lot of information in common. Making use of this coherence, the algorithm can be speeded up. We must somehow keep track of reflections which overwrite others and which just add up. This again has to be determined and updated only when edge crossings occur or when reflections change status. The reader is referred to [SUTH74] for a fundamental discussion of making use of coherence.

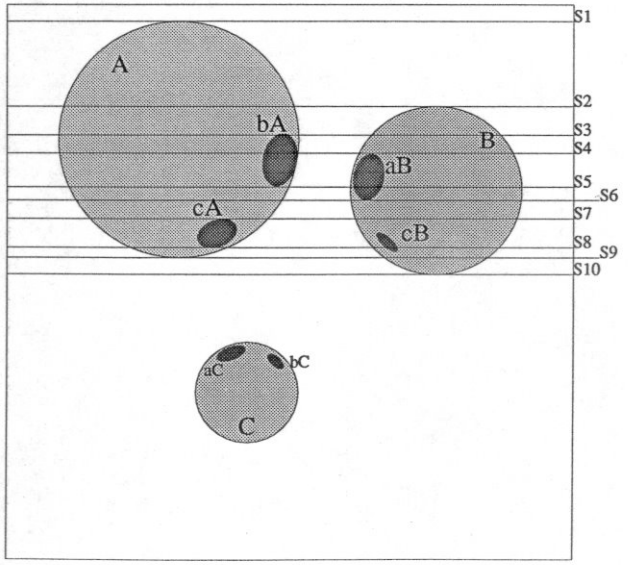


Fig. 7.1

Fig. 7.1 illustrates the working of the algorithm. For the sake of simplicity only three non-intersecting spheres, A,B,C, are shown. Initially the spheres are sorted into the Y-buckets. Sphere A becomes active at the scan line S1. At this point, the

reflections bA of B in A and cA of C in A are merged with the Y-bucket list. These reflections don't become active at the scan line S1 and hence the algorithm continues to render the scan line with only sphere A being active. At scan line S2, the sphere B becomes active and is added to the active list. At the same time the reflections aB and cB are merged with the Y-bucket list.

At scan line S3, the reflection bA becomes active and is merged with the active list. Similarly aB is merged with the active list at scan line S4. At scan line S5 the reflection bA becomes inactive and is taken off the active list. Likewise aB, A and B become inactive respectively at the scan lines S5, S9 and S10. The reflection cA enters the active list at scan line S7 and leaves it at scan line S8. The scan lines at which other reflections enter and leave the active list are not shown for the sake of clarity.

With this approach high complexity pictures can be generated. Sorting will not be a major problem as noted. We need to verify our claims by implementation of the algorithm.

7.4 Complexity Analysis

7.4.1 Time Complexity

Initial bucket sorting is linear in the number of input spheres. All the reflections are processed at least once. Hence the algorithm is at best linear in the size of the output which is $O(\mathbf{n^d})$. For complex scenes with lot of spheres, it can be made sub-linear if spheres which are totally invisible are eliminated completely. Potentially, all the reflections may be active at one of the scan lines. In this case the algorithm has to sort the reflections and hence the time complexity is $O(\mathbf{n^d} log \mathbf{n^d})$. But in the average case, not all the reflections will touch a single scan line and we can expect a better performance. The sorting step dominates all other steps of the algorithm.

7.4.2 Space Complexity

Again, in the case where all the reflections may be active at a single scan line, we need to store all the reflections at once. Hence the worst case space complexity will

be $O(n^d)$. In the average case, the space complexity will be less for the same reasons as in the previous paragraph.

7.5 Parallelism

Scan line algorithm is basically an image space algorithm. There is no easy way to divide the data and distribute them among several processors. However, one way to achieve parallelism would be to divide the display screen into areas and hand each area to a different processor. Each processor renders the scene that falls in its region. The resulting outputs will be merged into a single frame buffer when all the processors complete rendering their share of the scene.

7.6 Remarks

The scan plane algorithm is suitable for printers, film recorders and other raster display devices. It is fairly easy to achieve anti-aliasing with this algorithm. It is moderately easy to implement since it has been in existence from the early stages of computer graphics for rendering various primitives such as polygons.

8

CURVILINEAR TRAPEZOID METHOD

The Temple of Science is a multi-faceted building.
— Albert Einstein (1879-1955)

8.1 Introduction

The previous three chapters discussed hidden surface algorithms which are inherently raster-device oriented. In fact, the painter's algorithm and the z-buffer algorithm make use of the frame buffer to effect hidden surface elimination. The scan-plane method, as noted, can be made to produce device independent output. Yet, it is more suitable for scan-line oriented devices. In this chapter we investigate an algorithm capable of producing device-independent output which can be rendered on raster or vector devices.

Algorithms described so far carried out hidden surface elimination at the pixel level or scan line level. The current algorithm does it at a higher level. It is suitable for writing directly onto photographic film. The film recorder is driven by a microcomputer attached to a host computer. The host computer does the hidden surface elimination and the microcomputer does the filling in and shading. The output can be previewed on a vector or raster device before committing it to film.

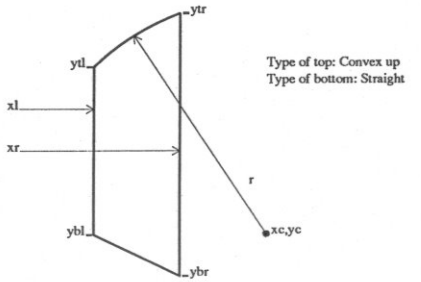


Fig. 8.1 A curvilinear trapezoid and its parameters.

8.2 Curvilinear Trapezoid Method

The idea of using curvilinear trapezoids for hidden surface elimination was first introduced by Knowlton and Cherry [KNOW77]. The algorithm is "$2\frac{1}{2}$D" in that most

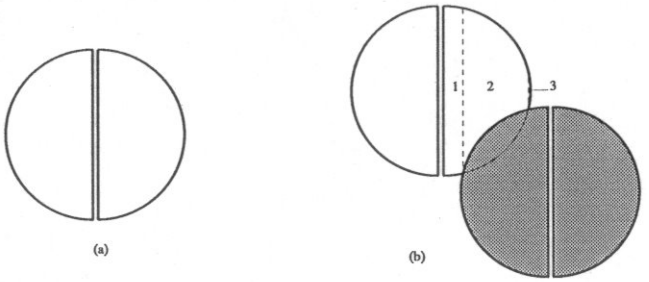


Fig. 8.2 (a) Initial trapezoids of a sphere. (b) Subdivision of one sphere when another occludes it and the resulting visible trapezoids.

of the hidden surface calculations are carried out in the projected image space rather than the object space. The precision of the computation and of the data stored is still at object resolution. The basic algorithm [KNOW77] for computing visible portions of scenes consisting of spheres works in the following way. Each visible region in the scene is represented as a curvilinear trapezoid consisting of two vertical straight edges and the other two sides being straight lines or monotonic concave or convex curves. Fig. 8.1 shows a sample curvilinear trapezoid. The vertical edges can potentially be degenerate. For example, a sphere is represented initially as two trapezoids, each with a degenerate vertical edge, as shown in Fig 8.2a. If a sphere being currently processed occludes part of another sphere already processed, the trapezoids of the latter sphere are subdivided further to represent the visible portions, and the invisible portions are discarded(Fig. 8.2b). The process is repeated until all the spheres are processed. The collection of visible trapezoids is then rendered.

The algorithm as described in [KNOW77] and [MAX79] is capable of handling intersecting spheres. In this case both the spheres hide each other partly and share the intersection curve. This means that the order in which the two spheres are processed does not matter. However, as new spheres are added, some of the earlier trapezoids

may be completely eliminated. If we are considering only non-intersecting spheres, we can avoid this by initially sorting the spheres and processing them in a front-to-back order. In that way the new spheres added do not occlude the earlier ones and the spheres entirely invisible may be completely eliminated. For our current purposes we will assume that we have non-intersecting spheres in the scene. When the algorithm terminates, the visible patches remaining in the scene are rendered to represent the scene.

The algorithm as applied to rendering spheres with reflections follows the same steps as in [KNOW77] except for the following:

[1] The [KNOW77] algorithm uses elliptic or circular arcs to represent the curved edges of the picture. These arcs may not be adequate to represent edges of reflections. Hence we need to approximate these with higher order polynomials. A better method would be to approximate the arcs with splines having enough number of control points.

[2] Each output trapezoid is no longer a simple trapezoid. Reflections do not hide base spheres, but base spheres do hide other elements in the scene. However, this may not be as serious a problem as was in the case of the painter's algorithm or the other algorithms. Since we know exactly which areas are visible, we can render them one after another in the proper order, adding to the frame buffer. Here again we can make use of coherence properties. For each sphere, all the reflections in it lie within its silhouette.

Since we restrict our curves to be monotonically convex or concave, we can represent the trapezoids by splinegons [SOUV86]. We can cut down on the number of subdivisions by observing the fact that reflections do not hide base spheres. When a reflection is added onto a base sphere, we do not need to subdivide the base sphere. However, when a base sphere is occluded by another, we need to subdivide both the base sphere and the reflections in it. Also, we need to carry the information about the base sphere when rendering the reflections in it. Both these can be easily carried out if we have a

tree like data structure for each sphere and the reflections in its surface. Each node of such a tree (Fig. 8.3) contains information about a reflection and a linked list of visible patches. The root of the tree represents the base sphere. The children of the root are the first-level reflections in the base sphere. The children of each node represent the next level reflections in the parent node. When a node needs to be subdivided, all its children nodes should be tested for associated further subdivisions. But when a child node is being subdivided, its parent node need not be subdivided. When rendering, the whole tree is rendered by traversing the tree in postfix, infix or prefix order, and adding the contribution of each visible patch.

To illustrate, let us assume that we want to represent a sphere A in which we see the reflections of spheres B and C as in figure 8.3. The root node consists of the label A to represent the sphere A and a list of visible patches 1 and 2 (Fig. 8.3a). When the reflection b of B in A is added (Fig. 8.3b), the base sphere A need not be subdivided further and hence we just add the node b as a child of A with its list of visible patches consisting of trapezoids 3 and 4. Upon the addition of reflection c of C in A (Fig. 8.3c), the sphere A again stays put while the patch 4 of b has to be subdivided into patches 7,8 and 9. Thus the list of the node b is altered and a new node c is added as its sibling with a list of visible patches in it consisting of patches 5 and 6. To render the sphere A with its reflections, we can traverse the tree in, say, the prefix order, and render the patches 1,2,3,7,8,9,5 and 6 in that order. And when rendering a patch in the visible list of a node, we need to have the information about the spheres that were encountered along the path from the root node to this node. Since this information can be obtained when traversing the tree, we need not store the information about every sphere involved in a reflection at all nodes.

8.3 Detailed Description of the Algorithm

The spheres are sorted to form a list in front-to-back order. Starting with an empty forest we build a tree for each sphere in the list. The root node of a tree represents a base sphere and its children nodes represent reflections in that base sphere. Each

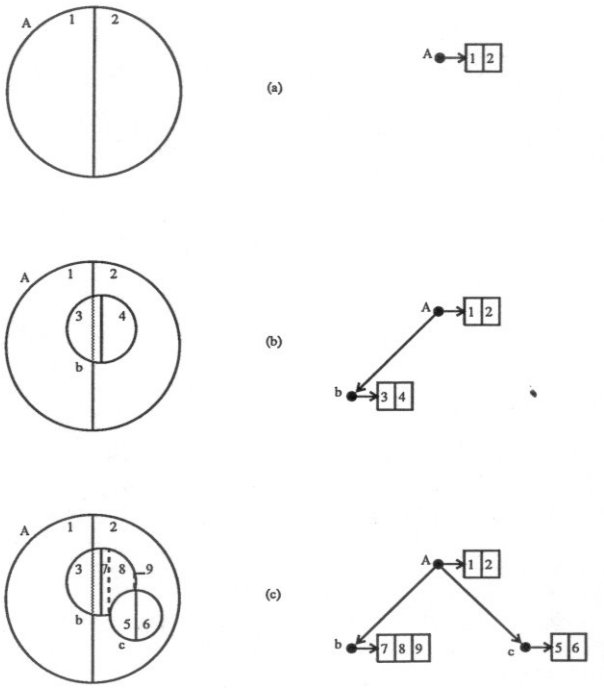


Fig. 8.3

node points to a list of visible patches. We process each sphere in the sorted list and at the same time process the reflections in it.

When a sphere is first considered, it is represented as a bi-trapezoid. It does not hide any of the ones already added, since we process the spheres in front-to-back order. Hence we need not alter the visible list so far. We divide the trapezoids into visible portions. For this purpose we need to clip only against the base spheres that were

processed earlier and hence the higher-level reflections in these spheres can be ignored. In other words, we need to worry only about the root nodes of the trees in the forest built so far. We create a new tree with the root pointing to the list of visible patches in the current sphere. We then generate the next level reflections recursively and add children nodes to the root node which point to the corresponding visible patches. The forest data structure thus generated can be used to generate all the visible patches.

To render the scene from the forest data structure generated, we traverse each tree in the forest in prefix order and render each visible patch encountered. When a patch is rendered, it is just added to the frame buffer. In theory, the patches can be rendered in any order since hidden surface elimination has already been carried out.

This algorithm can be implemented using a general purpose graphics device. It requires no extra hardware. As mentioned earlier it is ideally suited for a microprocessor based graphics system for raster display or for film recorders. This suits the graphics workstation model discussed in chapter 4.

8.4 Complexity Analysis

8.4.1 Time Complexity

There is an initial sorting of the spheres which takes $O(\mathbf{n}log\mathbf{n})$ time. The major time of the algorithm will be spent in subdividing the trapezoids to find the visible patches. This amounts to clipping each sphere or a reflection to those which have already been processed.

When a base sphere is processed, it needs to be clipped against the other base spheres already considered, which are at most $O(\mathbf{n})$ in number. Thus the processing time for $\mathbf{n}$ base spheres is $O(\mathbf{n^2})$.

The $O(\mathbf{n})$ first-level reflections in a given base sphere are required to be clipped against $O(\mathbf{n})$ first level reflections in that base sphere as well as $O(\mathbf{n})$ base spheres. Thus for processing all $O(\mathbf{n^2})$ first level reflections we need time proportional to $O(\mathbf{n^2} \bullet (\mathbf{n} + \mathbf{n})) = O(\mathbf{n^3})$.

Continuing the analysis this way we find that the second level reflections require $O(\mathbf{n^4})$ processing, third level reflections require $O(\mathbf{n^5})$ processing, and so on. Thus the total time required for processing $\mathbf{d}$ levels of reflections is given by

$$O(\mathbf{n^2} + \mathbf{n^3} + \mathbf{n^4} \cdots + \mathbf{n}^{d+2}) = O(\mathbf{d} \bullet \mathbf{n}^{d+2})$$

Thus the time complexity of the algorithm is $O(\mathbf{d} \bullet \mathbf{n}^{d+2})$.

8.4.2 Space Complexity

Space complexity of the algorithm can be deduced from an analysis close to that above for time complexity. Each clipping operation will give rise to one or more new patches. If we assume that there is a maximum limit to the number of patches generated per clipping operation, the space complexity of the algorithm will be identical to the time complexity, that is $O(\mathbf{d} \bullet \mathbf{n}^{d+2})$.

The space complexity can be reduced by an orderly sequence of processing the spheres and the reflections. We have already seen that the front-to-back ordering of the spheres reduces some amount of work. We also know that when a sphere is processed, all the reflections lie within its silhouette. With these two pieces of knowledge let us see how the space complexity can be diminished. We clip a sphere in the sorted list against the $O(\mathbf{n})$ spheres occluding it. This requires $O(\mathbf{n})$ space. After we clip the base sphere we consider the first level reflections in it. We also process the first level reflections in a "front-to-back" order. A given first level reflection has to be clipped to the $O(\mathbf{n})$ base spheres and $O(\mathbf{n})$ first level reflections in the sphere being processed. This needs $O(2\mathbf{n})$ space. Before we proceed to another first level reflection, we recursively continue to process the next level reflection. It is easy to extend the analysis to reveal that the second level reflection can be processed with a space of $O(3\mathbf{n})$. In general, a $\mathbf{k}$th level reflection will need $O((\mathbf{k}+1)\mathbf{n})$ space. We proceed to the next base sphere only when the current base sphere is finished. The same is true of the higher level reflections. After all the first-level reflections in the current sphere have been processed, they can be written out to the output and the memory used for

them can be released. We can argue the same way about the second level reflections in a given first level reflection, and so on, upto the **d**th level reflections. Thus, the space complexity of the algorithm is just $O(\mathbf{n}+2\mathbf{n}+3\mathbf{n}\cdots(\mathbf{d}+1)\mathbf{n}) = O(\mathbf{d^2n})$. If a look-up table is maintained to obtain the sorted list of spheres for each level of reflection, as was discussed in the case of the modified painter's algorithm, the space complexity for storing the table will be $O(\mathbf{n^2})$. Thus the effective space complexity of the algorithm is $O(\mathbf{n(d^2+n)})$.

8.5 Parallelism

The hidden surface elimination part of the algorithm seems to be sequential at the outset. However, a slightly modified version of the algorithm may lead to a certain level of parallelism. Instead of processing each sphere and the reflections in it at the same time, we can process all the base spheres first and then add on the reflections one level at a time. The base spheres will be processed sequentially by a single processor. Then each sphere can be handed to a different processor. These processors will then compute the visible patches for the first level reflections. The second and higher-level reflections can similarly be assigned to separate processors.

Once the forest data structure is created, the individual patches can be rendered in any order. Thus we can assign each patch to a different processor. This assumes that each processor has access to the frame buffer. This may cause bus contention. As an alternative, we can give each processor its own memory buffer in which it renders a given patch and a central processor then merges all the buffers together to get a final picture.

8.6 Remarks

The curvilinear trapezoid method performs extensive object to object clipping and hence requires elaborate programming effort to implement it. The rewards are that the output is device-independent and anti-aliasing is easy to achieve since there are no overlapping primitives in the final result.

9

ALGORITHMS IN PERSPECTIVE

It isn't that they can't see the solution.
It is that they can't see the problem.

— Gilbert Keith Chesterton (1874-1936)

We have thus far described four algorithms to render scenes with reflections and hidden surface elimination. They share some common characteristics and differ in others. In the following paragraphs we undertake to study them in perspective and bring out their similarities and contrasts. At the same time, we comment on which of the goals set forward in chapter 2 are accomplished by each of the algorithms. We are also interested in comparing our algorithms with ray-tracing since we presented these algorithms as alternatives to it.

For the sake of brevity, we shall term the modified painter's algorithm MPA, the modified z-buffer method MZM, the scan plane algorithm SPA, the curvilinear trapezoid method CTM, and ray-tracing RT.

Tables 9.1 through 9.3 below give a bird's-eye view of the algorithm comparisons. They list the main features of interest.

Algorithm Complexity		
Algorithm	Time Complexity	Space Complexity
MPA	$O(\mathbf{n^d \bullet n}\ log\ \mathbf{n})$	$O(\mathbf{n \bullet d})$
MZM	$O(\mathbf{n^d})$	$O(\mathbf{n + d + R^2})$
SPA	$O(\mathbf{n^d}\ log\ \mathbf{n^d})$	$O(\mathbf{n^d})$
CTM	$O(\mathbf{d \bullet n^{d+2}})$	$O(\mathbf{d \bullet n^{d+2}})$
RT	$O(\mathbf{n^d \bullet R^2})$	$O(\mathbf{n + d})$

Table 9.1

9.1 Algorithm Complexity

9.1.1 Time Complexity

Please refer to Table 9.1. In this table **n** represents the number of objects in the scene, **d** is the number of reflection levels required and **R** is the resolution of the (raster) display device on which the final output will be displayed.

The time complexity is useful for studying the asymptotic behavior of the algorithms when the input consists of a large number of objects or the number of reflection levels required is big or both. The same is true of space complexity of the algorithms.

When there are only a few objects in the scene, the constants of proportionality in the complexity measure play an important role. These constants depend to some extent on the implementation details such as the speed of the computer, the efficiency of the program etc., and so are not shown in the tables. It is sufficient for us to talk about the constants in a qualitative sense, in terms of small, moderate or huge constants.

Table 9.1 says that MZM is the winner regarding the asymptotic time complexity of the algorithms. In the light of the discussion above, this may be a little deceiving when there the number of objects in the scene is small. The MZM has a large constant because every entity in the scene is rendered. The same is true of the MPA. The RT has the same order of complexity as the MZM but it has a multiplier $\mathbf{R^2}$ where $\mathbf{R}$ is the resolution of the display device. This is because of the fact that the RT has to trace at least one ray per pixel. For a display of 512×512 resolution this constant is $512 \times 512 = 262144 = 2^{18} \approx 10^5$. For scenes with reflections, we can expect the number of objects in the scene to be small. As we mentioned in the earlier chapters, a scene with just a few objects can be quite complicated (Plate 2).

The MZM and the RT achieve their complexity because they avoid sorting which is an $O(\mathbf{p}\ log\ \mathbf{p})$ process, where $\mathbf{p}$ is the number of entities involved in sorting.

9.1.2 Space Complexity

The RT has the best space complexity. For this algorithm it is enough to keep track of the $\mathbf{n}$ objects in the scene and information about one ray for each of the $\mathbf{d}$ reflection levels. Hence its space complexity is $O(\mathbf{n + d})$. The MZM uses z-buffers which are as large as the frame buffer used for the display. This accounts for the additional term $O(\mathbf{R^2})$ in its space complexity expression.

Again, as in the case of time complexity, the impact of the space complexity measure of the algorithm depends heavily on the number of objects in the scene as well as the computing environment available.

The complexities shown for MPA are when no pre-processing is carried out to form a look-up table. With table look-up its time complexity is reduced to $O(\mathbf{n^d} + \mathbf{n^2} log \mathbf{n})$ whereas the space complexity increases to $O(\mathbf{n^2})$.

Algorithm Features			
Algorithm	Coherence	Implementation	Parallelism
MPA	Area + Object	Easy	Moderate
MZM	Area + Object	Moderate	Moderate
SPA	Scan Line	Moderate	Little
CTM	Area	Elaborate	Moderate
RT	None	Easy	Inherent

Table 9.2

9.2 Algorithm Features

9.2.1 Coherence

The algorithms make use of various levels of coherence. Coherence, as defined in chapter 4, is the extent to which the scene is locally constant. There are various types of coherence [SUTH74] possible in a given environment. By exploiting coherence the algorithms can save work.

The MPA and the MZM make use of area coherence. When rendering an element of the scene (an object or a reflection), the algorithms have to work only on the projected area, and in that projected area they have to render only that particular element. This is what we call area coherence. The CTM also has area coherence, but to a finer degree. It renders only visible patches of the scene and each patch occupies a certain area of the projected scene and does not depend on other patches. It seems that the MPA and the MZM have better coherence than the CTM. However, the former algorithms have to do more work rendering the objects. They render each and every object regardless of whether it is visible or not, and the hidden surface elimination process takes place at the same time as rendering. The CTM, on the other hand,

renders only the visible patches. The hidden surface elimination is done at a higher level before the rendering begins.

The MPA and the MZM also have what we call object-to-object coherence. When rendering an object with these algorithms, the object need not be clipped explicitly to other objects in the scene. With the MPA, the clipping is done implicitly by overwriting the background elements with the foreground objects. The MZM does it in a similar fashion, at a pixel level, using the z-buffer.

The SPA has scan-line coherence. The information about the segments of entities visible from one scan-line to the next does not change very much. We would like to term this scan-line-to-scan-line coherence. Also along a scan line, between two edge crossings (along a visible segment), the visibility information does not change from pixel to pixel. This is scan-line coherence. The SPA makes use of both these types of coherence.

We listed the RT as having no coherence at all. This is true only of a naive ray-tracing algorithm. In recent years authors have reported enhancements to the naive ray-tracing algorithm which make use primarily of object space coherence [GLAS86, FUJI86]. Other methods make use of object coherence [AMAN84, HECK84, SPEE85, JOY86], but these methods are not as robust as our algorithms in making use of the coherence.

9.2.2 Implementation

Ease of implementation is a big factor in deciding the utility value of an algorithm. This is especially so in today's software environments. It is conceivable that the useful life cycle of a software system is only a few years. Moreover, machine time has become less expensive compared to the software implementor's time. Thus, if an algorithm is easy to implement, it is likely to be implemented even in spite of its poor CPU performance. This is perhaps the reason why we see so many versions of ray-tracing programs floating around, even though the algorithm is slow to execute. The RT, at least in its basic form, is very easy to implement.

Among the algorithms we have developed, the MPA is easy to implement. It is conceptually simple to understand. The MZM is moderately easy if z-buffering is available in hardware. Otherwise, the z-buffering can be simulated in software if enough memory is available on the host processor. If this is not the case, other more complicated methods have to be adopted.

The SPA is only moderately easy since there is a need to keep track of a fair amount of information from one scan line to the next. The CTM requires elaborate programming to implement computation of the visible patches and clipping.

9.2.3 Parallelism

Only the RT is inherently parallel. This is because each pixel can be treated independent of the others. In fact, in the naive version of the algorithm, each pixel *is* processed separately from the others. A fair amount of parallelism can be achieved with little programming effort. In fact, various parts of the scene can be rendered on different machines and combined together. However, each processor needs to have the global data about all the objects in the scene. The algorithm can be easily implemented on a parallel machine with shared memory architecture such as the Firefly.

The MPA and the MZM are moderately parallel. Both of these algorithms combine hidden surface elimination with the rendering process. The CTM is also moderately parallel. This algorithm computes the visible patches first and then renders them. The hidden surface elimination part of the algorithm can be parallelized to a certain degree. The visible patches can then be rendered in parallel independent of each other. The SPA has little parallelism.

9.3 Output Characteristics

9.3.1 Anti-aliasing

For applications such as movie making and art, where a high quality picture is desired, anti-aliasing is a feature to look for in an algorithm. In the final production stage,

Output Characteristics			
Algorithm	Anti-aliasing	Device-independence	Resolution-independence
MPA	Moderate	No	No
MZM	No	No	No
SPA	Yes	No	No
CTM	Yes	Yes	Yes
RT	No	No	No

Table 9.3a

though time is always critical, a good picture is worth considerably more than a number of bad ones. A desirable feature of an algorithm would therefore be the ability to produce pictures fast without anti-aliasing and on demand yield good quality anti-aliasing with a little additional work.

With the MPA anti-aliasing is easy since we know the borders of entities being drawn. There is, however, a problem when the projections of two objects intersect on the image plane. For example, when a sphere hides another sphere, at the intersection point of the projections of the two spheres the pixel represents part of both the spheres. Though it is easy to compute how much of the pixel is occupied by the closer of the two spheres, there is no information available, at the time of rendering it, about the farther sphere. Hence at the intersection points the anti-aliasing will not be very good.

The MZM does not have anti-aliasing capabilities at all. To achieve anti-aliasing oversampling methods similar to A-buffering [CARP84] may have to be adopted.

It is easy to achieve anti-aliasing with the SPA and the CTM. With the SPA, we need to anti-alias only at the edge crossings on each scan line and when objects change from being visible to invisible and vice versa. The CTM produces visible patches which are effectively non-overlapping in the image plane. Thus the problems with the MPA do not occur.

Ray-tracing is, in a sense, similar to the z-buffering algorithm in the anti-aliasing context. The simple version of the algorithm does not perform anti-aliasing, as there is no knowledge of the edges in the scene. There are various enhancements to achieve anti-

aliasing with the RT. The distributed ray-tracing [COOK84] technique uses multiple rays per pixel. This only adds to the already expensive CPU requirements of the RT. Cone tracing [AMAN84] and beam tracing [HECK84] achieve some what better degree of anti-aliasing with lesser CPU requirements.

9.3.2 Device-independence

There are primarily two kinds of devices, raster devices and vector devices. In addition to this broad classification, there are other variations in device characteristics. For example, on most of the raster display devices individual pixel elements can be accessed randomly and pixels can be overwritten. Printers, on the other hand, provide access to the pixels on a scan-line-by-scan-line basis only. Also, there is a material cost incurred for overwriting with a printer. Thus, a printer can be considered as a raster device with certain restrictions. Likewise, plotters can be thought of as vector devices with penalties for overwriting. Film recorders are also raster devices with random access capabilities, but they cannot be overwritten at all since writing onto photographic film is an irreversible process. Table 9.3b tabulates specific devices the algorithms are suited for.

Environments Suitable					
Algorithm	Raster Device	Vector Device	Printer	Plotter	Film Recorder
MPA	Yes	No	No	No	No
MZM	Yes	No	No	No	No
SPA	Yes	No	Yes	No	Yes
CTM	Yes	Yes	Yes	Yes	Yes
RT	Yes	No	No	No	No

Table 9.3b

If the output from an algorithm is in a form that can be displayed on all devices, then it is termed device-independent. Also, the output can be rendered to its full glory with filling-in and shading, or only the contours of the primitives can be drawn to yield a fast preview of the picture. This also depends on the output representation scheme.

The MPA is suitable for a quick display of the scene on raster devices when there are only a few objects in the scene. The intermediate output of the MPA, after the objects in the scene have been sorted out and the painting order determined, can be thought of as a device-independent output representation. But there is a fair amount of redundancy in this output data. And, it uses the image space to generate the final scene from this data, since, the actual hidden surface elimination is carried out in the image space. Thus, it cannot be used to display a line drawing of the scene, with hidden lines eliminated. For the same reason, it is not suitable for display on printers, plotters, and for directly writing onto photographic film.

The MZM shares the same characteristic as the MPA in that both use the image space for hidden surface elimination. The MZM is more specific, since it needs pixel based raster device for this purpose. The MZM is viable for display on raster devices when a z-buffer is available in hardware or can be software simulated on the availability of enough memory. The MZM, like the MPA, is not suitable for printers and film recorders. Also, it does not produce any intermediate output. It yields only a raster output and hence cannot drive vector devices or plotters.

The SPA is very much suitable for printing on scan-line oriented printers as well as for raster displays. It can also drive film recorders if the CPU performance is satisfactory. It cannot drive vector devices and plotters. However, it is capable of producing intermediate output with slight modifications.

The CTM is the best among the algorithms in this characteristic. The CTM does produce device-independent output in its most concise form. There are no redundancies in the output data. The output very closely resembles the final scene. The output is also compact and takes less space for long term storage. It is thus suitable for transmission over tight communication channels. The output of the CTM can be displayed on raster devices, printers and film recorders, using scan-conversion fill algorithms to render the visible patches generated by the CTM. The patches can also be rendered on vector devices and plotters by rendering only the borders of the patches for a quick

preview. Thus the CTM is versatile across various environments.

The RT produces only raster output. Hence it is suitable for displaying on raster devices. However, it may not be suitable for printing devices, either because of the size of the raster required to be stored and transmitted, or because of CPU requirements if the RT directly drives a printer. The same holds true for film recorders.

9.3.3 Resolution-independence

Scalability is another flexibility desirable in the output format. If the output can be scaled, it can be displayed without regard to the resolution of the display device. Also, in a windowing environment this is an added advantage, since, the scene can be displayed on various sized windows.

Only the CTM produces resolution-independent output. The others produce bitmaps which can be scaled to a limited extent, using scaling algorithms that apply to rasters.

9.4 Remarks

From the above analysis it is apparent that the CTM is advantageous on many counts. However, the MPA and the MZM are easy to implement and perhaps should be tried out first by enthusiasts. The SPA is a compromise between ease of implementation and performance. The course of implementation may be to first implement the RT, then in order the MPA, the MZM, the SPA, and finally the CTM.

10

FINAL THOUGHTS

This is not the end.
It is not even the beginning of the end.
But it is, perhaps, the end of the beginning.

— Sir Winston Churchill (1874-1965)

I know you believe you understand what you think I said,
but I'm not sure you realize that what you heard is not what I meant.

— Anonymous

10.1 Results

Let us summarize the results of the research presented in this dissertation. The goal was to undertake a disciplined study for characterizing reflections in curved surfaces and to device alternate algorithms to ray-tracing for rendering reflections. Ray-tracing, as we discussed, is the only existing method to satisfactorily generate reflections, but suffers from several drawbacks.

We have developed a fairly good understanding of how reflections are generated. At the outset it was not clear as to how the reflections in curved surfaces could be modeled. Then we looked at the physics of reflections and how the ray-tracer generated reflections. The knowledge derived from this study culminated in three algorithms to generate the basic reflection of one object in another. These three algorithms are respectively the contour tracing algorithm, special purpose numerical algorithm and spline approximation algorithm. Two of these algorithms yield the boundary representation or the contour of the reflection and the other gives spline patches approximating the reflection. We have implemented the first of these algorithms and have tools to implement the other two algorithms. A sample picture obtained with the contour tracing algorithm was presented in chapter 4.

We have outlined four algorithms which render reflections with hidden surface elimination. These algorithms make use of one of those mentioned above as a primitive. We developed these algorithms as alternatives to ray-tracing, which has several disadvantages as discussed in chapter 2. We have placed emphasis on the speed of the algorithms as well as their flexibility in terms of output representation. Our algorithms are better in many aspects than ray-tracing and not too bad in others for most practical applications. Chapter 9 gave a comparative study of these algorithms as well as ray-tracing.

10.2 Future Work and Open Problems

The next logical step would be to implement one of the four algorithms presented. As mentioned at the end of chapter 9 the modified painter's algorithm or the modified

z-buffer method should be tried out as a start. Comparison of the performance of these algorithms with that of ray-tracing should give a fair idea of their utility value. We expect big gains if the output resolution desired is large.

We would like to extend the results obtained in this dissertation to more general surfaces. We have focused on spheres as our primitive objects. Spheres are the simplest of curved surfaces and belong to a class of surfaces called quadric surfaces. Other surfaces of this class are ellipsoids, cylinders, cones, hyperboloids and paraboloids. Some of these surfaces have finite extent and others have infinite extent. Immediate extensions to the algorithms should span at least the general quadric surfaces with finite extent.

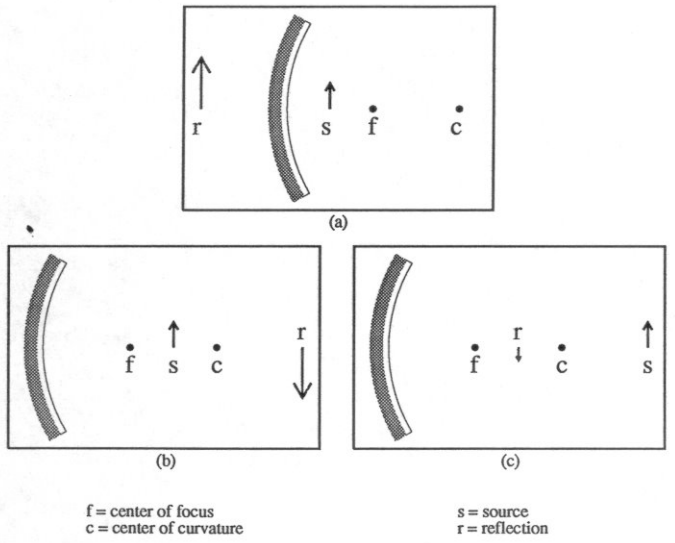


Fig. 10.1 (a) Virtual image. (b) Real image (magnified). (c) Real image (diminished).

One interesting open problem is to generate reflections in concave surfaces. Plane sections of concave surfaces have what is known as a *center of focus* (Fig. 10.1). The distance between the center focus and the reflecting surface is known as the *focal distance* or *focal length*. For circular sections, the focal distance is equal to half the radius of curvature. If the distance **u** between the object being reflected and the reflecting surface is less than the focal distance, the image formed will be virtual (Fig. 10.1a). If it is greater than the focal distance but less than twice the focal distance, the image formed will be real and will lie at a distance greater than twice the focal distance (Fig. 10.1b). Likewise, if **u** is greater than twice the focal length, the image will be formed at a distance between focal length and twice the focal length (Fig. 10.1c). An object at infinity is focused at the center of focus and naturally the image of an object at the center of focus is formed at infinity. One can very well imagine an object enclosing the center of focus such that part of the object is within the focal distance and part of it is not. Then the image will be not only discontinuous but also consist of both virtual and real parts.

One plausible solution to handle the above situation is as follows: Let us suppose that we have a technique for rendering real reflections. We can approximate the reflected object by set of spline patches. The spline patches can be subdivided easily to form smaller patches. We can thus subdivide the object to separate parts of the object which fall within the focal distance from those which do not. Then we can render reflections of the former as virtual reflections and the latter as real reflections. Part of the object which falls right at the focal distance may be clipped to avoid problems with infinite images.

Another possible extension would be to render scenes with refraction. Projecting from the experience with rendering reflections, we expect to run into similar problems such as the various aberrations. With refraction we have the additional complexity of *chromatic aberration*. This is due to the fact the material of the refracting body offers different refractive indices for different wave lengths of light. In spite of these

problems we can expect to get pictures close to that of a ray-traced picture.

Transparency can be considered as a weaker form of refraction. We can define transparency as the translucence of the objects neglecting refraction effects. This should be easier to achieve than full fledged refraction. Having transparency in the scene adds to its realism without the complexity imposed by refraction.

It would be interesting to attempt hardware implementation of the algorithms. One specific requirement for hardware implementation is that the algorithm should have simple repeatable structure. Thinking along these lines may yield a new algorithm which can be easily implemented in hardware but may be slow in an equivalent software implementation. Parallel processing is another good direction to investigate.

In essence, we have just scratched the surface of the shell of the research egg. There are a great many possibilities for further exploration.

Production of commercials and computer generated movies are some of the areas for the application of our results. Realizing imaginary mathematical objects akin to Plate 2 on computer displays which involve reflections would be another application. Other areas of interest are visual special effects, arts and computer vision. Perhaps new avenues will open up in the area of computer vision for recognizing objects from their reflections. As demonstrated by eminent researchers in this area, vision requires some sort of knowledge about the model of the object being recognized. Perhaps our techniques can be used to generate those required models. A remote application of this would be the following scenario: If, in a self driven robot automobile, it is required to monitor both the forward field and the rear field, we can mount a mirror to monitor the rear field. The mirror will be a convex mirror to cover a larger field of vision. In all automobiles the rear view mirror is convex for the same reason. The use of the mirror is justified if the visual sensor is expensive and a second sensor cannot be mounted. We can visualize the results of our current research applied to this situation either directly or indirectly. For example, the model of the reflected object can be generated using contour tracing techniques and this contour of the model of the reflection may

be sufficient to give clues regarding the actual object being recognized.

We hope that we have paved the way for future research in the direction set out in this dissertation. The techniques developed should have a continued effect on future researchers in the design of both algorithms and input-output representations.

One small step for man, one big step for mankind.
— Neil A. Armstrong

APPENDIX 1

PHYSICS OF REFLECTIONS

All that glitters may not be gold,
but at least it contains free electrons.
— John Desmond Bernal (1901-1971)

This appendix gives an example to hint at the nature of reflections in curved surfaces. Physics literature has established the various aberrations that occur due to the curvature mirror surfaces. Here we illustrate just one of them, the *coma*.

We can think of physics as the science of experimenting about the laws of nature and fitting the results so obtained to an empirical formula. The same can be said about the physics of reflections. It is important to know about the physics of reflections because it tells us the nature of reflections and also gives clues as to how to derive the position and shape of a reflection. For example, the reflection in a plane mirror is virtual, is of the same shape and size as the source and the position of the reflection is such that "*the image behind the mirror is at the same distance from the plane mirror as the object is in front of it.*"

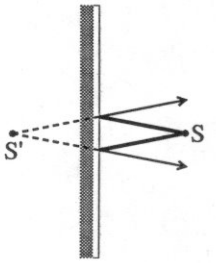


Fig. A1.1a Virtual Image

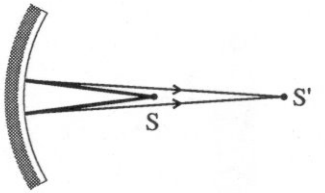


Fig. A1.1b Real Image

Physics tells us that for a point light source the position of the image formed is the point from which the reflected rays of light appear to emerge (diverge) or the point at which the light rays converge. In the former case we have a virtual image (Fig. A1.1a), and in the latter a real image(Fig. A1.1b).

Another important result physics has given us is that in order to obtain the reflection of a point source of light we need to trace two or more rays. The intersection of the reflected rays (produced backwards if necessary) then gives us the position of the image (Fig. A1.1a). In the case of curved mirrors, it is necessary that both the

source and the image screen (on which the real image is caught) or the eye (for viewing virtual images) be on or close to the optical axis of the mirror. Then *paraxial* rays, i.e. rays that deviate very little from the optical axis of the reflecting surface, are to be traced to obtain the position of the image (Fig. A1.1b).

Suppose that for a given position of the eye E, a given point source of light S, and a reflecting surface O, we can trace a ray R emanating from S that reaches the eye after reflection in O(Fig. A1.2). The reflected ray R′ determines the direction along which the image of S will lie. To determine the position of the image, we take another ray A very close to the ray R. The intersection of the reflected rays A′ and R′ yields the approximate position of the image. The exact position of the image is obtained as the intersection point in the limit the ray A tends to the ray R, provided such a limit exists. This forms the fundamental principle on which are based most of the lens and mirror equations in geometric (planar) optics.

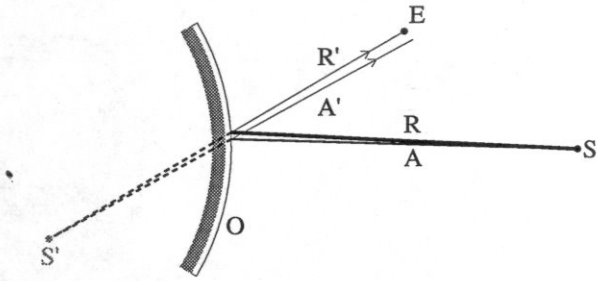


Fig. A1.2 Approximate image position.

Before we dive into the question of deriving the equation of the reflection of an object in a sphere, let us see what the above observation implies. From mathematics we know that a limit point is defined to exist only if the limit point is the same when we allow the ray A to converge to ray R from any direction. In the case of a plane mirror, such a limit point exists. It is a well known law from physics which we would like to mention here as a theorem.

Theorem 1. *The image behind a plane mirror is at the same distance as the object is in front of it. Conversely, if the image conforms to this rule, the mirror must be plane.*

It is important to recognize that the image in a plane mirror is independent of the eye position. Hence it is very straightforward to characterize and derive the reflection of an object in a plane mirror. It is just the "reflection" of the object in a mathematical sense.

However, in the case of curved mirror surfaces such a limit point does not always exist. This can easily be shown by a simple example, involving a sphere which is the simplest of the curved surfaces. First we state when a limit point can exist. Limit point exists precisely when the eye and the source lie on the same axial line passing through the center of the sphere. It ceases to exist when the eye and the image are far removed, subtending a large angle at the center of curvature. The following figures illustrate why this is so.

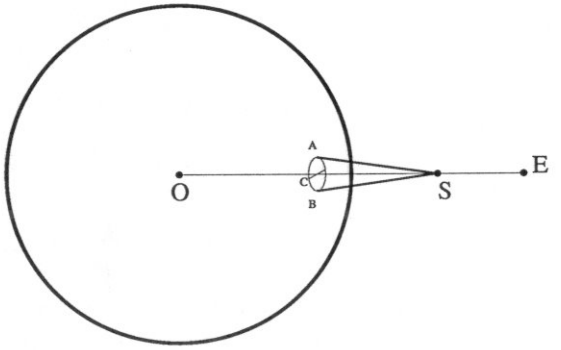


Fig. A1.3 Paraxial view point.

In Fig. A1.3, the eye E lies along the axial line joining the center of the sphere O

to the source S. Consider two rays SA and SB in the plane SAB. They see a certain curvature offered by the spherical surface. This same curvature is offered in any plane containing the center of the sphere and the source S and the eye E, such as for example the plane SCE. Hence the image of S in the sphere O exists as a limit point in the above sense. Its existence can be justified by symmetry arguments. The image position in this case is still independent of the eye position, provided the eye lies along the axis. And it is given by the well know formula $1/u + 1/v = 2/r$, where u is the distance from the mirror surface to the source S, v is the (algebraic) distance from the mirror surface to the image point and r is the radius of curvature of the surface at the point of reflection.

To illustrate the second case, let us take an extreme situation. In Fig. A1.4a, the point S lies close to the tangent from the eye to the reflecting sphere. The circle ACBD highlights the point of tangency which is at its center. In the line of the tangent, the sphere presents a curvature close to zero. Thus the surface acts as a plane mirror in the direction of the tangent and the corresponding image i.e., the limit point of intersection of the rays EA and EB as A and B converge to the point of tangency, will be found at S′. However, in a direction perpendicular to the tangent the sphere presents its "full" curvature and thus the limit point will be different for the intersection of the rays EC and ED. We shall not derive where the limit point falls but rather appeal to Theorem 1. In Fig. A1.4b the situation is depicted as seen from the eye position and also as a projection on a plane containing the tangent line and the center of the sphere O.

In general, the shape of the image of a point source will resemble a comet and hence the aberration caused due to extra-axial points is called *coma*. This causes serious problems in the design of optical instruments employing curved elements such as lenses and mirrors and has been studied extensively in physics [BORN65]. Thus there does not exist a simply definable reflection in a curved surface! This is rather disappointing from the point of getting a clean solution to the problem. Readers may object to this by saying that they do see a reflection in a spherical ball such as a

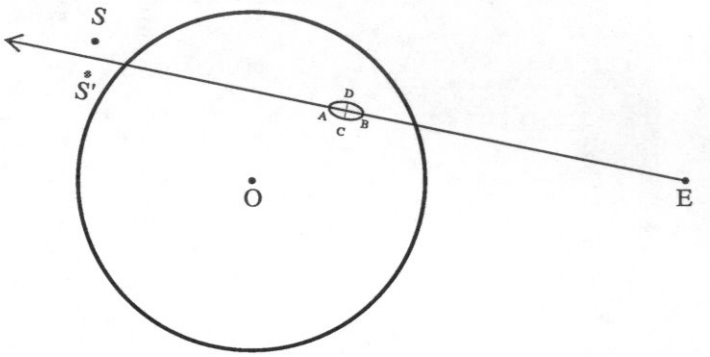


Fig. A1.4a In the tangent line the sphere acts as a plane mirror.

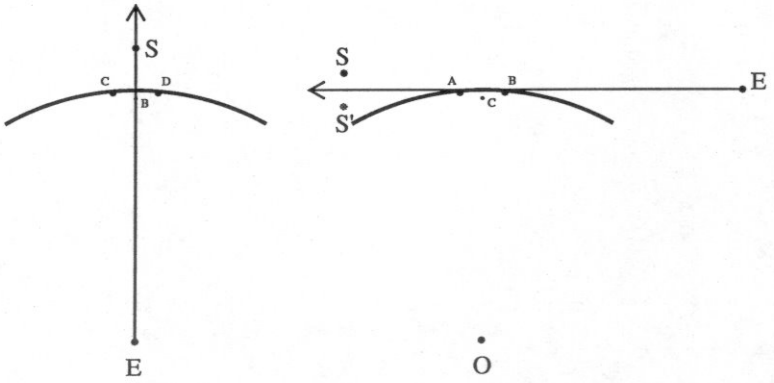


Fig. A1.4b Eye's view of the situation in Fig. A1.4a.

mirrored Christmas ball. However, the image we see in such balls is not the "true" mathematical reflection. The eye acts as an optical integrator and together with the brain "makes up" the reflection that we "see."

APPENDIX 2

GEOMETRY OF REFLECTIONS

Let no one ignorant of geometry enter my door.
— Plato (429-347 BC)

In this appendix we outline the algebra involved in solving for reflections analytically. The outcome is a negative result showing the impossibility of closed form solution.

Problem Statement

Appendix 1 dealt with the impossibility of modeling reflections in curved surfaces with simple mathematical formulae. In chapter 3 we said that we can approximate the reflections by finding the contours within which fall all the rays emanating from the eye that intersect the reflected object after reflection in the surface of the reflecting object. With this preamble the problem to be solved can be stated as:

> Given a reflecting object O, another object S to be reflected, and a given eye position E, find on the surface of O the contour of the reflection of S in O.

Simplified 2-D case

In what follows, we consider rays as emanating from the eye and reaching the reflected object after reflection in the reflecting surface — in the spirit of ray-tracing.

Let us consider the 2-D version of the above problem limited to simple objects and reflecting surfaces. In Fig. A2.1a, the reflecting surface is a circle O and the object to be reflected is another circle centered at S and the eye is at the position E. We can think of the rays AP and BQ as tangents to the reflected object S and after reflection in the surface of O they pass through the eye. In other words, points A and B on the circle O represent the contour of the reflection of S in O.

We will now show that even this seemingly simple problem of finding the required two points by analytic means is algebraically quite complicated and this serves as the motivation to abandon the analytic approach to the solution and to look for numerical algorithms.

The problem can be attempted from several different approaches. The natural one to take is to simply apply the principles of geometric optics, which are stated as follows for reflections.

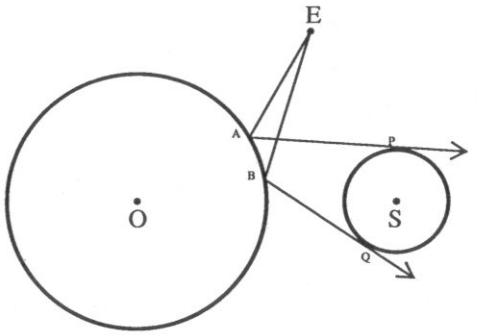


Fig. A2.1a A circle reflecting another.

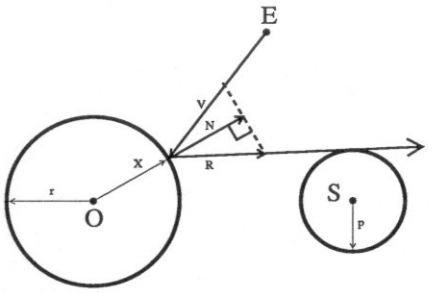


Fig. A2.1b

a. The incident ray, the reflected ray and the normal to the surface at the point of reflection all lie in the same plane.

b. The angle of incidence is equal to the angle of reflection.

Principle **a** is redundant to the current problem since we are dealing in 2-D. However, it is very important in the 3-D case. In addition to the above two conditions, we can add a further condition relevant to the current problem, namely,

c. The reflected ray is tangential to the reflected object S.

The above conditions suffice to characterize both the contour points A and B. Without loss of generality, we make the simplification that the reflecting object O is centered at the origin. A point X on the circle O (Fig. 2.1b). satisfies the equation

$$\vec{X} \bullet \vec{X} = r^2$$

where $\vec{X}$ represents the vector from the origin O to the point X, r is the radius of the circle O and $\bullet$ represents the scalar product of vectors. Let $\vec{V}$ be the vector from the eye to the point X, and $\vec{N}$ be the normal vector to the circle O at the point X. We can take $\vec{N} = \vec{X}$ for the circle centered at the origin. Then the reflected ray direction $\vec{R}$ is given by [WHIT80]

$$\vec{R} = \vec{V'} + 2\vec{N}$$

where

$$\vec{V'} = \frac{\vec{V}}{|\vec{V} \bullet \vec{N}|}$$

The parametric equation of the reflected ray can be written as

$$\vec{W}(t) = \vec{X} + t \times \vec{R}$$

where $\vec{W}(t)$ is a general point on the reflected ray and t is the independent scalar parameter in the range $-\infty \leq t \leq \infty$. The intersection points of this ray with the reflected circle S satisfy the equation:

$$(\vec{W}(t) - \vec{S}) \bullet (\vec{W}(t) - \vec{S}) - p^2 = 0$$

where p is the radius of the circle S. The above equation is a quadratic equation in t and the condition for tangency is that there be only one solution for t. In other words the discriminant of the equation should be set to zero. This yields an equation, with t eliminated, which has to be solved for X.

We attempted to solve this problem using MAPLE (a symbolic manipulation language) [CHAR85] and the results were surprising. It gave us a sixth degree equation

in x and y, the components of the vector $\vec{X}$. It is well known [HERS75, JACO74] that a general algebraic equation of degree greater than five is not solvable in closed form as a function of its coefficients using radicals. What we have is a sixth degree equation in x and y and the solution seems hopeless. While there are Galois-theoretic techniques [BAJA84] to determine whether a particular form of algebraic equation is solvable by radicals in closed form, the results are likely to be unsatisfying. Without going into the details, we strongly feel that the particular equation at hand is not analytically solvable. We say this because the existence of closed form solution depends on the symmetries that exist in the problem and apparently the solutions are not symmetric. The truth of this can be easily verified along the lines of [BAJA84]. Even if the problem were solvable in closed form, we may still need to find numeric solutions as the degree of the equations grow with the increase in the number of levels of reflection.

Further analysis showed that the sixth degree equation was obtained because there are indeed six solutions to the problem with the conditions used in the course of deriving the equations as above. There are only two physically realizable solutions but the other four solutions are geometrically possible. In Fig. A2.2a and A2.2b the points A′ and A″ also satisfy the conditions of the problem. However, solution A′ is not physically possible because the ray EA′ cannot penetrate the (opaque) sphere and in Fig. A2.2b the reflected ray at A″ has to be produced backwards to meet the reflected object S and thus represents an imaginary ray. In addition to these two solutions two more solutions B′ and B″ are possible which are counterparts of the solution B.

Thus additional conditions are required to sift out the two required solutions. These conditions take the form of constraints, namely,

1. The angle between the incident ray and the normal is acute.
2. The parameter t at the point of tangency to the reflected circle is positive.

But, constraints can only clip off some of the unwanted solutions *after* the system of equations has been solved. Thus we still need to solve the sixth degree equation, which is not analytically possible. Looking at the six solutions reveals that they are

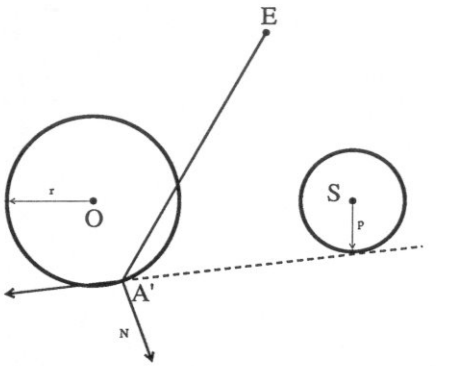


Fig. A2.2a

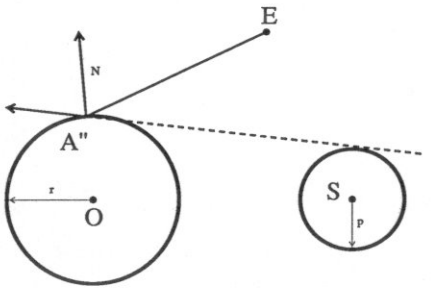


Fig. A2.2b

not symmetric about any axis and intuitively the equation is irreducible to factors of lesser degree equations.

We took other approaches to solving the problem, but all of them led to sixth degree equations of one sort or another. One of the methods worth mentioning is as follows: Fermat's principle states that a light ray takes the (locally) minimum possible amount of time in traveling from one point to the other. Thus if we take all possible

paths from a source point S to the eye E, with an intermediate point being a general point X on the reflecting circle, the length of the path will be the least when EX lies along the reflection of the ray SX in the circle. Thus we can pose the problem as that of minimizing the sum of the distances SX and XE with the constraint that the point X lines on the circle O. However, this does not simplify the problem to any degree since there are three local minima possible.

The problem being such in 2-D, it is worse in 3-D. Thus we are forced to find other than closed form solutions. Such solutions are discussed in chapter 4.

If you do not expect the unexpected, you will not find it;
for it is hard to be sought out and difficult.
— Heraclitus of Ephesus (550-475 BC)

BIBLIOGRAPHY

[ADAG82] *ADAGE RDS 3000 User's Guide,* ADAGE, INC., April 1982.

[AHO74] AHO, A. V., J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms,* Addison-Wesley Publishing Company, 1974.

[AMAN84] AMANATIDES, J., "Ray-tracing With Cones," *Computer Graphics,* **18(3),** July 1984, pp. 129-135.

[ASAL86] ASAL, M. ET AL., "The Texas Instruments 34010 Graphics System Processor," *IEEE CG&A,* **6(10),** October 1986, pp. 24-39.

[ATHE78] ATHERTON, P., K. WEILER AND D. GREENBERG, "Polygon Shadow Generation," *Computer Graphics,* **12(3),** August 1978, pp. 275-281.

[BAJA84] BAJAJ, C., "The Algebraic Degree of Geometric Optimization Problems," **CSD-TR-496,** Purdue University, October 1984.

[BARR81] BARR, A. H., "Super Quadrics and Angle Preserving Transformations," *IEEE Computer Graphics and Applications,* **1(1),** January 1981, pp. 11-23.

[BARR84] BARR, A. H., "Global and Local Deformations of Solid Primitives," *Computer Graphics,* **18(3),** July 1984, pp. 21-30.

[BARS84] BARSKY, B. A., "A Description and Evaluation of Various 3-D Models," *IEEE CG&A,* **4(1),** January 1984, pp. 38-52.

[BENT86] BENTLEY, J., "Birth of a Cruncher," *Communications of the ACM,* **29(12),** December 1986, pp. 1155-1161.

[BISW85] BISWAS, S. N. AND B. B. CHAUDHURI, "On the Generation of Discrete Circular Objects and Their Properties," *Computer Vision, Graphics and Image Processing,* **32,** 1985, pp. 158-170.

[BLIN76] BLINN, J. F. AND M. E. NEWELL, "Texture and Reflection in Computer Generated Images," *Communications of the ACM,* **19(10),** November 1976, pp. 542-546.

[BLIN77] BLINN, J. F., "Models of Light Reflection for Computer Synthesized Pictures," *Computer Graphics,* **11(2),** Summer 1977, pp. 192-198.

[BLIN78] BLINN, J. F., "Simulation of Wrinkled Surfaces," *Computer Graphics,* **12(3),** August 1978, pp. 286-292.

[BLIN82] BLINN, J. F., "A Generalization of Algebraic Surface drawing," *ACM TOG,* **1(3),** July 1982, pp. 235-256.

[BOIN81] BOINODIRIS, S., "Hidden Surface Elimination for Complex Graphical Scenes," *Computer Graphics,* **14(4),** March 1981, pp. 153-167.

[BOOT82] BOOTH, K. AND S. A. MCKAY, "Techniques for Frame Buffer Animation," *Proceedings of Graphics Interface '82,* May 1982, pp. 213-220.

[BORN65] BORN, M. AND E. WOLF, *Principles of Optics,* Pergamon Press, 1965.

[BRES65] BRESENHAM, J. E., "Algorithm for Computer Control of a Digital Plotter," *IBM Systems Journal,* **4(1),** 1965, pp. 25-30.

[BROW84] BROWN, M. H. AND R. SEDGEWICK, "A System for Algorithm Animation," *Computer Graphics,* **18(3),** July 1984, pp. 177-186.

[BUIT75] BUI-TUONG, PHONG, "Illumination for Computer Generated Pictures," *Communications of the ACM,* **18(6),** June 1975, pp. 311-317.

[CARI86] CARINALL, C. AND J. BLAIR, "National's Advanced Graphics Chip Set for High-performance Graphics," *IEEE CG&A,* **6(10),** October 1986, pp. 40-48.

[CARP84] CARPENTER, L., "The A Buffer, An Antialiased Hidden Surface Method," *Computer Graphics,* **18(3),** July 1984, pp. 103-108.

[CATM74] CATMULL, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces,* Ph. D. Thesis, Dept. of Computer Science, University of Utah, 1974.

[CATM78] CATMULL, E., "A Hidden Surface Algorithm With Antialiasing," *Computer Graphics,* **12(3),** August 1978, pp. 6-11.

[CATM79] CATMULL, E., "A Tutorial on Compensation Tables," *Computer Graphics,* **13(2),** August 1979, pp. 1-7.

[CATM80] CATMULL, E. AND A. R. SMITH, "3D Transformations of Images in Scan Line Order," *Computer Graphics,* **14(3),** July 1980, pp. 279-285.

[CHAN81] CHANG, P. AND R. JAIN, "A Multi-processor System for Hidden Surface Removal," *Computer Graphics,* **15(4),** December 1981, pp. 405-436.

[CHAR85] CHAR, B. W. ET AL., *Maple User's Guide,* WATCOM Publications Limited, 1985.

[COOK81] COOK, R. L. AND K. E. TORRANCE, "A Reflectance Model for Computer Graphics," *Computer Graphics,* **15(3),** August 1981, pp. 307-316.

[COOK84] COOK, R. L., T. PORTER AND L. CARPENTER, "Distributed Ray Tracing," *Computer Graphics,* **18(3),** July 1984, pp. 137-145.

[CROC84] CROCKER, G. A., "Invisibility Coherence for Faster Scan Line Hidden Surface Algorithms," *Computer Graphics,* **18(3),** July 1984, pp. 96-102.

[CROW77a] CROW, F., "Shadow Algorithms for Computer Graphics," *Computer Graphics,* **11(2),** Summer 1977, pp. 242-247.

[CROW77b] CROW, F., "The Aliasing Problem in Computer Shaded Images," *Communications of the ACM,* **20(11),** November 1977, pp. 799-805.

[CSUR79] CSURI, C. ET AL., "Towards an Interactive, High Visual Complexity Animation System," *Computer Graphics,* **13(2),** August 1979, pp. 289-299.

[DIPP84] DIPPE, M. AND J. SWENSON, "An Adaptive Sub-division Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics,* **18(3),** July 1984, pp. 148-158.

[DOBK83] DOBKIN, D. P. AND D. G. KIRKPATRICK, "Fast Detection of Polyhedral Intersection," *Theoretical Computer Science,* **27,** 1983, pp. 241-253.

[DOBK85a] DOBKIN, D. P. AND D. L. SOUVAINE, "Computational Geometry - A User's Guide," *Princeton University Technical Report* **CS-TR-334-85,** February 1985.

[DOBK85b] DOBKIN, D. P., "Geometric Complexity and Computer Graphics," *Unpublished Manuscript,* July 1985.

[DOBK85c] DOBKIN, D. P. AND I. M. MUNRO, "Efficient Uses of the Past," *Journal of Algorithms,* **6(4),** December 1985, pp. 455-465.

[DOBK86] DOBKIN, D. P., W. P. THURSTON AND A. R. WILKS, "Robust Contour Tracing," *Princeton University Technical Report* **CS-TR-054-86,** September, 1986.

[DUBR83] DUBRULLE, A. A., "A Class of Numerical Methods for the Computation of Pythagorean Sums," *IBM Journal of Research and Development,* **27(6),** November 1983, pp. 582-589.

[DUFF79] DUFF, T., "Smoothly Shaded Renderings of Polyhedral Objects," *Computer Graphics,* **13(2),** August 1979, pp. 270-275.

[DUNH81] DUNHAM, D. AND J. LINDGRAM, "Creating Repeating Hyperbolic Patterns," *Computer Graphics,* **15(3),** August 1981, pp. 215-223.

[FANT86] FANT, K. M., "A Non-aliasing Real Time Spatial Transform Technique," *IEEE CG&A,* **6(1),** January 1986, pp. 71-80.

[FARB86] FARBOOSH, H. AND G. SCHRACK, "CNS-HLS Mapping Using Fuzzy Sets," *IEEE CG&A,* **6(6),** June 1986, pp. 28-35.

[FARO85] FAROUKI, R. T. AND J. K. HINDS, "A Hierarchy of Geometric Forms," *IEEE CG&A,* **5(5),** May 1985, pp. 51-78.

[FEIB80] FEIBUSH, E. A., M. LEVOY AND R. L. COOK, "Synthetic Texture Using Digital Filters," *Computer Graphics,* **14(3),** July 1980, pp. 294-301.

[FIEL83] FIELD, D. E., *Algorithms for Drawing Simple Geometric Objects on Raster Devices,* Ph. D. Thesis, Princeton University, 1983.

[FOLE82] FOLEY, J. D. AND A. VAN DAM, *Fundamentals of Interactive Computer Graphics,* Addison Wesley Publishing Company, 1982.

[FRAN78] FRANKLIN, W. R. AND H. R. LEWIS, "3-D Graphics Display of Discrete Spatial Data by Prism Maps," *Computer Graphics,* **12(3),** August 1978, pp. 70-75.

[FRAN81] FRANKLIN, W. R., "An Exact Hidden Sphere Algorithm That Operates in Linear Time," *Computer Graphics and Image Processing,* **15(4),** April 1981, pp. 364-379.

[FREN86] FRENKEL, K. A., "Computers, Complexity and the Statue of Liberty Restoration," *Communications of the ACM,* **29(4),** April 1986, pp. 284-296.

[FREU86] FREUND, D. D., "An Interactive Procedure for Constructing Line and Circle Tangencies," *IEEE CG&A,* **6(4),** April 1986, pp. 59-63.

[FUCH79] FUCHS, H. AND Z. M. KEDEM, "Predetermining Visibility Priority in 3-D Scenes," *Computer Graphics,* **13(2),** August 1979, pp. 175-181.

[FUCH83] FUCHS, H., G. D. ABRAHAM AND E. D. GRANT, "Near Real Time Shaded Display of Rigid Bodies," *Computer Graphics,* **17(3),** July 1983, pp. 65-69.

[FUJI86] FUJIMOTO, A., T. TANAKA AND K. IWATA, "ARTS: Accelerated Ray Tracing System," *IEEE CG&A,* **6(4),** April 1986, pp. 15-26.

[GALL73] GALL, L., *Classical Galois Theory,* Chelsea Publishing Company, 1973.

[GLAS84] GLASSNER, A. S., "Space Subdivision for Fast Ray-tracing," *IEEE CG&A,* **4(10),** October 1984, pp. 15-22.

[GOLD83a] GOLDMAN, R. N., "Quadrics of Revolution," *IEEE CG&A,* **3(3),** March 1983, pp. 68-76.

[GOLD83b] GOLDMAN, R. N., "Two Approaches to a Computer Model for Quadric Surfaces," *IEEE CG&A,* **3(9),** September 1983, pp. 21-24.

[GOUR71] GOURAUD, H., "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers,* **C-20(6),** June 1971, pp. 623-628.

[GREE86] GREEN, N. AND P. S. HECKBERT, "Creating Raster Omnimax Images from Multiple Perspective Views," *IEEE CG&A,* **6(6),** June 1986, pp. 21-27.

[GUPT81] GUPTA, S. AND R. F. SPROULL, "Filtering Edges for Gray-scale Displays," *Computer Graphics,* **15(3),** August 1981, pp. 1-5.

[HEAR86] HEARN, D. AND P. M. BAKER, *Computer Graphics,* Prentice Hall, 1986.

[HECK84] HECKBERT, P. S. AND P. HANRAHAN, "Beam Tracing Polygonal Objects," *Computer Graphics,* **18(3),** July 1984, pp. 119-127.

[HERB78] HERBISON-EVANS, D., "NUDES 2: A Numeric Utility Displaying Ellipsoid Solids," *Computer Graphics,* **12(3),** August 1978, pp. 354-356.

[HERB80] HERBISON-EVANS, D., "Rapid Raster Ellipsoid Shading," *Computer Graphics,* **13(4),** February 1980, pp. 355-361.

[HERS75] HERSTEIN, I. N., *Topics in Algebra,* John Wiley and Sons, 1975.

[HUBS81] HUBSCHMAN, H. AND S. ZUCKER, "Frame to Frame Coherence and the Hidden Surface Computation: Constraints for a Complex World," *ACM TOG,* **1(2),** April 1981, pp. 129-162.

[HYMA65] HYMAN, R. AND B. ANDERSON, "Solving Problems," *International Science and Technology,* September 1965, pp. 36-41.

[IMAI85] IMAI, H., M. IRI AND K. MUROTA, "Voronoi Diagrams in The Laguerre Geometry and it Applications," *SIAM Journal of Computing.,* **14(1),** February 1985, pp. 93-105.

[IRIS86] *IRIS User's Guide,* **V 3.0,** 1986, Silicon Graphics, Inc.

[JACO74] JACOBSON, N., *Basic Algebra I,* W.H.Freeman and Company, 1974.

[JOBL78] JOBLOVE, G. H. AND D. GREENBERG, "Color Spaces for Computer Graphics," *Computer Graphics,* **12(3),** August 1978, pp. 20-25.

[JOY86] JOY, K. I. AND M. N. BHETANABHOTLA, "Ray-tracing Parametric Surface Patches Utilizing Numerical Techniques and Ray Coherence," *Computer Graphics,* **20(4),** August 1986, pp. 279-285.

[KAJI82] KAJIYA, J. T., "Ray Tracing Parametric Patches," *Computer Graphics,* **16(3),** July 1982, pp. 245-254.

[KAJI83] KAJIYA, J. T., "New Techniques for Ray Tracing Procedurally Defined Objects," *Computer Graphics,* **17(3),** July 1983, pp. 91-102.

[KAJI84] KAJIYA, J. T. AND B. P. VON HERZEN, "Ray-tracing Volume Densities," *Computer Graphics,* **18(3),** July 1984, pp. 165-174.

[KAY79] KAY, D. S. AND D. GREENBEG, "Transparency for Computer Synthesized Images," *Computer Graphics,* **13(2),** August 1979, pp. 158-164.

[KNOW77] KNOWLTON, K. AND L. CHERRY, "ATOMS–A Three-D Opaque Molecule System–for Color Pictures of Space-filling Ball-and-stick Models," *Computers and Chemistry,* **1,** 1977, pp. 161-166.

[KNOW81] KNOWLTON, K., "Computer-Aided Definition, Manipulation and Depiction of Objects Composed of Spheres," *Computer Graphics,* **15(4),** December 1981, pp. 352-375.

[KOLA84] KOLATA, G., "Esoteric Math Has Practical Result," *Science,* August 1984, pp. 494-495.

[LEVI79] LEVIN, J. Z., "Mathematical Models for Determining the Intersections of Quadric Surfaces," *Computer Graphics and Image Processing,* **11,** 1979, pp. 73-97.

[LI87] LI, K., "Private Communication," March 1987.

[LIEB78] LIEBERMAN, H., "How to Color in a Coloring Book," *Computer Graphics,* **12(3),** August 1978, pp. 111-116.

[MAHL72] MAHL, R., "Visible Surface Algorithms for Quadric Patches," *IEEE Transactions on Computers,* **21(1),** January 1972, pp. 1-4.

[MAND82] MANDELBROT, B. B., *The Fractal Geometry of Nature,* W. H. Freeman Press, 1982.

[MARO82] MARON, M. J., *Numerical Analysis: A Practical Approach,* McMillan Publishing Co., 1982.

[MARS80] MARSHAL, R., R. WILSON AND W. CARLSON, "Procedure Models for Generating 3-D Terrains," *Computer Graphics,* **14(3),** July 1980, pp. 154-162.

[MAX79] MAX, N. L., "ATOMLLL:- Atoms with Shading and Highlights," *Computer Graphics,* **13(2),** August 1979, pp. 165-173.

[MAX81] MAX, N. L., "Vectorized Procedural Models for Natural Terrain: Waves and Islands in the Sun Set," *Computer Graphics,* **15(3),** August 1981, pp. 317-324.

[MAX83] MAX, N. L., "Computer Representation of Molecular Surfaces," *IEEE CG&A,* **3(8),** August 1983, pp. 21-29.

[MOLE83] MOLER, C. AND D. MORRISON, "Replacing Square Roots by Pythagorean Sums," *IBM Journal of Research and Development,* **27(6),** November 1983, pp. 577-581.

[MORA81] MORAVEC, H. P., "3D Graphics and The Wave Theory," *Computer Graphics,* **15(3),** August 1981, pp. 289-296.

[PACH84] PACHNER, J., *Handbook of Numerical Analysis Applications,* McGraw-Hill Book Company, 1984.

[PAND85] PANDURANGA, E. S., "A Bit-sliced Microprocessor Based Graphics System," Computer Science Research Seminar, Princeton University, January 1985.

[PARK80] PARKE, F. I., "Simulation and Expected Performance Analysis of Multiple Processor Z-buffer Systems," *Computer Graphics,* **14(3),** July 1980, pp. 48-56.

[PATN86] PATNAM, L. K. AND P. A. SUBRAMANIAN, "Boolean Operations on N-dimensional Objects," *IEEE CG&A,* **6(6),** June 1986, pp. 43-51.

[PHIL84] PHILLIPS, M. B. AND G. M. ODELL, "An Algorithm for Locating and Displaying the Intersection of Two Arbitrary Surfaces," *IEEE CG&A,* **4(9),** September 1984, pp. 48-58.

[PILL80] PILLER, E. AND H. WIDNER, "Real-time Raster Scan Unit with Improved Picture Quality," *Computer Graphics,* **14(1&2),** July 1980, pp. 15-38.

[PIQU83] PIQUE, M. E., "Fast 3-D Display of Space-filling Molecular Models," *Technical Report* **83-004,** Department of Computer Science, University of North Carolina, 1983.

[PITT67] PITTEWAY, M. L. V., "Algorithm for Drawing Ellipses and Hyperbolae With a Digital Plotter," *Computer Journal,* **10(3),** November 1967, pp. 282-289.

[PITT81] PITTEWAY, M. L. V., "On Filtering Edges for Grey-scale Displays," *Computer Graphics,* **15(4),** December 1981, pp. 322-326.

[PORT78] PORTER, T. K., "Spherical Shading," *Computer Graphics,* **12(3),** August 1978, pp. 282-285.

[PORT79] PORTER, T. K., "The Shaded Surface Display of Large Molecules," *Computer Graphics,* **13(2),** August 1979, pp. 234-236.

[POTM81] POTMESIL, M. AND I. CHAKRAVARTHY, "A Lens and Aperture Camera Model for Synthetic Image Generation," *Computer Graphics,* **15(3),** August 1981, pp. 297-305.

[PROS82] PROSSER, C. J. AND A. C. KILGOUR, "Generating Conic Sections Using Integer Arithmetic," *Report* **CSC/82/R1,** University of Glasgow, Department of Computer Science, March 1982.

[ROGE85] ROGERS, D. F., *Procedural Elements for Computer Graphics,* McGraw Hill, 1985.

[ROTE82] ROTENBERG, A., "Johnny – An Algorithm for Reading Orthographic Drawings," *Engineering Design Graphics Journal,* Winter 1982, pp. 10-17.

[ROTH82] ROTH, S. D., "Ray Casting for Modeling Solids," *Computer Graphics and Image Processing,* **18,** 1982, pp. 109-144.

[RUBI80] RUBIN, S. M. AND T. WHITTED, "A 3-dimensional Representation for Fast Rendering of Complex Scenes," *Computer Graphics,* **14(3),** July 1980, pp. 110-116.

[SAIL82] SAILLEN, P., "Hidden Lines Elimination for Parameterized Surfaces," *Engineering Design Graphics Journal,* Winter 1982, pp. 18-23.

[SALE86] SALESIN, D. AND R. BARZEL, "Two bit Graphics," *IEEE CG&A,* **6(6),** June 1986, pp. 36-42.

[SARR83] SARRAGA, R. F., "Algebraic Methods for Intersections of Quadric Surfaces in GMSOLID," *Computer Vision, Graphics and Image Processing,* **22,** 1983, pp. 222-238.

[SECH81] SECHREST, S. AND D. P. GREENBERG, "A Visible Polygon Reconstruction Algorithm," *Computer Graphics,* **15(3),** August 1981, pp. 17-27.

[SEDE84] SEDERBERG, T. W. AND D. C. ANDERSON, "Ray-tracing of Steiner Patches," *Computer Graphics,* **18(3),** July 1984, pp. 159-164.

[SEDE85] SEDERBERG, T. W. AND D. C. ANDERSON, "Steiner Surface Patches," *IEEE CG&A,* **5(5),** May 1985, pp. 23-36.

[SEDE86] SEDERBERG, T. W. AND R. N. GOLDMAN, "Algebraic Geometry for Computer Aided Design," *IEEE CG&A,* **6(6),** June 1986, pp. 52-59.

[SEQU85] SEQUIN, C. H. AND P. R. WENSLEY, "Visible Surface Return at Object Resolution," *IEEE CG&A,* **5(5),** May 1985, pp. 37-50.

[SHAR85] SHARIR, M., "Intersection and Closest Pair Problems for a Set of Planar Disks," *SIAM Journal of Computing,* **14(2),** May 1985, pp. 448-468.

[SHIR86] SHIRES, G., "A New VLSI Graphics Coprocessor: The Intel 82786," *IEEE CG&A,* **6(10),** October 1986, pp. 49-55.

[SHOU79] SHOUP, R. G., "Color Table Animations," *Computer Graphics,* **13(2),** August 1979, pp. 8-13.

[SINH80] SINHA, R. M. K. AND A. RAMAN, "A Modular Data Terminal for Indian Languages," *Computer Graphics,* **14(1&2),** July 1980, pp. 39-72.

[SMIT78] SMITH, A. R., "Color Gamut Transform Pairs," *Computer Graphics,* **12(3),** August 1978, pp. 12-19.

[SMIT84] SMITH, A. R., "Plants, Fractals and Formal Languages," *Computer Graphics,* **18(3),** July 1984, pp. 1-10.

[SOUV86] SOUVAINE, D. L., *Computational Geometry in a Curved World,* Ph. D. thesis, Princeton University, 1986.

[SPEE85] SPEER, L. R., T. D. DEROSE AND B. A. BARSKY, "A Theoretical and Empirical Analysis of Coherent Raytracing," *Proceedings of Graphics Interface '85,* May 1985, pp. 11-25.

[STAU86] STAUDHAMMER, J. AND A. KHURANA, "Display of Molecular Models With Interactive Computer Graphics," *IEEE CG&A,* **6(1),** January 1986, pp. 26-31.

[SUTH74] SUTHERLAND, I. E., R. F. SPROULL AND R. A. SCHUMAKER, "A Characterization of Ten Hidden Surface Algorithms," *Computing Surveys,* **6(1),** March 1974, pp. 1-55.

[TAMM84] TAMMINEN, M. AND H. SAMET, "Efficient Octree Conversion by Connectivity Labeling," *Computer Graphics,* **18(3),** July 1984, pp. 43-51.

[THAL86] THALMANN, M. N. AND D. THALMANN, "Special Cinematographics Effects With Virtual Movie Cameras," *IEEE CG&A,* **6(4),** April 1986, pp. 43-50.

[THUR84] THURSTON, W. P., J. R. WEEKS, "The Mathematics of Three Dimensional Manifolds," *Scientific American,* July 1984, pp. 108-120.

[TORI86] TORIYA, H., T. SATOH, K. UEDA AND H. CHIYOKURA, "Undo Redo Operations for Solid Modeling," *IEEE CG&A,* **6(4),** April 1986, pp. 35-42.

[TURK82] TURKOWSKI, K., "Anti-aliasing Through the Use of Coordinate Transformations," *ACM TOG,* **1(3),** July 1982, pp. 215-234.

[VANA84] VAN AKEN, J. R., "An Efficient Ellipse Drawing Algorithm," *IEEE CG&A,* **4(9),** Sept. 1984, pp. 24-35.

[VANW84] VAN WIJK, J. J. AND F. W. JANSEN, "Realism in Raster Graphics," *Computers and Graphics,* **8(2),** 1984, pp. 217-219.

[WEIM80] WEIMAN, C. F. R., "Continuous Anti-aliased Rotation and Zoom of Raster Images," *Computer Graphics,* **14(3),** July 1980, pp. 286-293.

[WEIN81] WEINBERG, R., "Parallel Processing Image Synthesis and Anti-aliasing," *Computer Graphics,* **15(3),** August 1981, pp. 55-62.

[WEIS66] WEISS, R. A., "BE VISION, A Package of IBM 7090 FORTRAN Programs to Draw Orthographic Views of Combinations of Plane and Quadric Surfaces," *JACM,* **13(2),** April 1966, pp. 194-204.

[WHIT80] WHITTED, T., "An Improved Illumination Model for Shaded Display," *Communications of the ACM,* **23(6),** June 1980, pp. 343-349.

[WHIT82] WHITTED, T. AND D. M. WEIMER, "A Software Testbed for The Development of 3D Raster Graphics System," *ACM TOG,* **1(1),** January 1982, pp. 43-58.

[WILL78] WILLIAMS, L., "Casting Curved Shadows on Curved Surfaces," *Computer Graphics,* **12(3),** August 1978, pp. 270-274.

[WRIG79] WRIGHT, T., "ISOPROF–An Algorithm for Plotting Iso-valued Surfaces of a Function of Three Variables," *Computer Graphics,* **13(2),** August 1979, pp. 182-189.

[WYWI85] WYWILL, G. AND T. L. KUNII, "A Functional Model for Constructive Solid Geometry," *The Visual Computer,* **1(3),** 1985, pp. 3-14.

[WYWI86] WYWILL, G., T. L. KUNII AND Y. SHIRAI, "Space Subdivision for Ray-tracing in CSG," *IEEE CG&A,* **6(4),** April 1986, pp. 28-34.