2001

Shadow generation Based on RE loops and their angular representations

Khageshwar Thakur
University of Kentucky, kthakur@lexmark.com

ABSTRACT OF THESIS

SHADOW GENERATION BASED ON RE LOOPS AND THEIR ANGULAR REPRESENTATIONS

The initial attempt was to find efficient technique to identify shadow polygons in the shadow-volume based shadow generation algorithm. It was observed that shadows correspond to loops of ridge edges (REs). By identifying all the non-overlapping RE loops of a 3D object, one finds all the shadow polygons and, consequently, all the shadows it generates on other objects as well as shadows it generates on itself. This, however, requires extensive edge-edge intersection tests.

It was subsequently realized that by storing the angular representations of the RE loops in a look up table, one can avoid the need of decomposing RE loops into non-overlapping loops and, consequently, the need of performing extensive edge-edge intersection tests. Actually, by building the look up table in a way similar to the bucket-sorted edge table of the standard scan-line method, one can use the table in the scan conversion process to mark the pixels that are in shadow directly, without the need of performing any ray-polygon intersection tests as required in the shadow-volume based shadow generation algorithm. Hence, one gets a new shadow generation technique without the need of performing expensive tests.

Keywords: shadow, shadow polygon, ridge edge, ridge edge loop, pseudo intersection point

Khageshwar Thakur
December 10, 2001

SHADOW GENERATION BASED ON RE LOOPS AND THEIR ANGULAR REPRESENTATIONS

By

Khageshwar Thakur

Fuhua (Frank) Cheng
Director of Thesis

Grzegorz W. Wasilkowski
Director of Graduate Studies

December 10, 2001

RULES FOR THE USE OF THESIS

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with the permission of the author, and with the scholarly acknowledgments.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

THESIS

Khageshwar Thakur

The Graduate School

University of Kentucky
2001

SHADOW GENERATION BASED ON RE LOOPS AND THEIR ANGULAR REPRESENTATIONS

THESIS

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science in the
College of Engineering
at the University of Kentucky

By
Khageshwar Thakur
Lexington, Kentucky

Director: Dr. Fuhua (Frank) Cheng, Professor of Computer Science
Lexington, Kentucky
2001

Acknowledgments

I am grateful to Dr. Fuhua Cheng for introducing me to the exciting field of Computer Graphics. His efficient support and guidance through out my Masters program lead to the successful completion of this thesis.

I am thankful to my lab-mates Dr. Jun-Hai Yong, Dr. Pifu Zhang and Shirish Pande for providing an environment conducive to good research. They were always ready to help me.

Table of Contents

List of Figures

List of Files

1	thesis.pdf	.....	604 KB

Chapter 1

Introduction

Shadow generation is a classic problem in computer graphics. The problem is to identify the regions that are in shadow and then modify the illumination accordingly. A region is in shadow if it is visible from the view point, but not from the light source (we assume that there is only one light source in the scene. When there are multiple light sources, we simply classify the region relative to each of them using the same technique). Shadow generation is important in that shadows not only increase realism of a picture, but also provide better understanding of the spatial relationships between objects: if object A casts a shadow on object B, then we know that A is between B and the light source [4][7][8].

1.1 Previous work

The problem of shadow generation has been studied for more than thirty years. Various methods for shadow generation have been suggested. An excellent survey of these methods can be found in [8]. These methods can be classified as *Shadow volume method*, *Area Subdivision method*, *Depth buffer method* and *Ray tracing method*.

In a shadow volume method [7], shadow polygons are generated as preprocessing, each face is marked front facing or back facing. While rendering the objects, each ray from view point is tested for shadow count (front facing polygons - back facing polygons). If the count is 0 then it is not in shadow. Later this method was refined by BSP trees [4]. In this method a binary tree is made as preprocessing which stores all the faces in back-to-front order which can be used for any view point by properly traversing the tree. Most recently, Chrysanthou and Slater [5] proposed a BSP tree based approach, which is a generalization of SVBSP tree approach proposed by Chin and Feiner [4].

In an area subdivision method[10][2], two passes of polygon clipping are used to calculate the region in shadow. In the first pass, a polygon clipper is used to separate the lit portion from dark portion, then in the second pass clipping is done with respect to viewing point to remove the hidden faces.

In depth buffer methods, a depth buffer (with respect to light source) is maintained to find lit faces. Lit portions of the faces are then rendered from the view point.

In a ray tracing method, a ray is shot from the view point, a surface with minimum hit distance is declared as visible. From each visible point a ray is shot to the light source, if

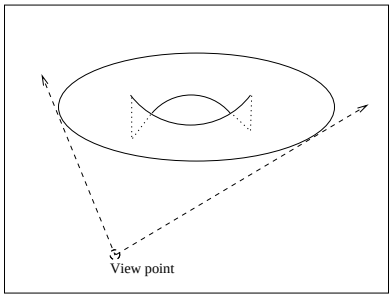

Figure 1.1: Rim and Occluding Contour of a Torus.

it intersects with any other object then the point is in shadow.

The idea of Visible Rim and Occluding Contour has been proposed in Image Processing and also in shadow generation [13][9][14]. An Occluding Contour is the projection of the Visible Rim. A Rim is a set of surface points whose tangent plane contains the viewpoint. A Visible Rim is a set of points on Rim which are visible. Figure 1.1 shows the Occluding Contour by solid line and Rim by Solid and dotted lines. In most cases Occluding Contour contains a set of closed curves and open curves ending in T-junction and cusps. These ending points has been of special interest in constraining the view points in image processing. Seales and Dyer [13] showed that Occluding Contours can also be used for shadow computation. An aspect representation or "asp" is used for all view points. An asp represents the appearance of each face in the scene from all viewing directions. A comparison of the appearances of a face from view point and from light source tells the regions in shadow. This method takes very good advantage of viewpath coherence for animation. This technique takes $\Omega(n^2)$ to $\Omega(n^4)$ time in preprocessing (asp construction) which is the dominant cost, where n is the number of faces in the scene.

All these methods have some advantages and disadvantages, but they have one thing in common, they do not scale well for large scenes. In this thesis, we present a new method for shadow generation for 3D polyhedra. This method, which starts from an idea very similar to Occluding Contours, exploits the fact that (1) the actual shadow is a logical OR of shadow produced by all objects and (2) the angular span of shadow remains the same at all depths for a given light source and a given object. The first fact dictates that we stop our search for shadow when we found one. The second suggests that angular coordinate system is a more natural choice for this problem. Presented method, combined with the above facts, becomes much faster. The preprocessing (lookup table construction) is so fast that it can be done on-line.

1.2 Our work

We start with the observation that shadow produced by an object is bounded by loop of Ridge Edges (REs). These loops are very similar to Rim discussed above. Loops of Ridge Edges in case of smooth surfaces becomes Rim. We define Ridge edge as an edge whose one adjacent face is visible and another is hidden from light source. We will prove that Ridge Edges always form a closed loop. Apparently it has been proved earlier that Rim of a smooth manifold also form a closed loop. Intuitively it seems obvious, but since we could not find any such proof, we can not establish a relationship between these two proofs.

The union of all the Ridge Edge loops is the boundary of the shadow volume. As we show later, these RE loops are very easy to find. We represent RE loops in an angular coordinate system. This means, we can use the same representation to calculate shadow on all objects without any transformation. We make a lookup table to store these representations. While looking up the shadow we stop the first time we find shadow. As seen by the simulation results, this method scales very well for large scenes and is very fast. This algorithm takes less than a second for a scene containing 2000 cubes (in the worst case). In fact the bottle neck is due to the scan conversion process. There are some limitations however, since the output of this method comes in the form of line segments, it is difficult to apply texturing. Also, this is more suitable for cases when light is away from scene.

The contributions of thesis include: (1) establishing a correspondence between shadows and RE loops, (2) developing efficient algorithms to identify and traverse the RE loops, (3) developing an angular representation of the RE loops so that a lookup table of RE loops can be constructed and used in the scan conversion process (using, for instance, z-buffer method) efficiently.

The remaining part of the thesis is arranged as follows. Chapter 2 describes in detail the RE loops traversing techniques, including external loops, internal loops, intersecting/overlapping loops, and loops corresponding to holes. In chapter 3 we demonstrate how to use the results identified in chapter 2 in the shadow generation process. Implementation details, performance, scalability and limitations are discussed in chapter 4. Special cases are addressed in chapter 5. Concluding remarks are given in chapter 6.

Chapter 2

Ridge Edge Loops

We will show in this chapter that shadows correspond to ridge edge loops. By identifying all the ridge edge loops of a 3D object, one can find the shadows it generates on other objects as well as shadows it generates on itself.

2.1 Definitions and Basic Properties

To find the outline of an object, it is sufficient to consider only *visible faces* or only *invisible faces*. (A face is defined to be visible if the dot product of its outward normal and a ray from the light source to any point on the face is less than zero.) In fact the edges of the outline are always the edges between visible and invisible faces. Such edges are called *ridge edges* (REs). In this thesis REs have been shown by red lines. REs always form a closed loop (will be proved shortly). Figure 2.1 shows an object with a very simple *RE loop*.

For more complicated objects there can be more than one loop. These loops can be joint (through a vertex) or disjoint, overlapping or completely inside another loop. Figures 2.2-2.4 shows some of these situations.

All the RE loops of a given object are potential source of shadow on all objects. If an RE loop is coiled more than once, we separate the loop into simpler ones via VREPIPs (to be defined later).

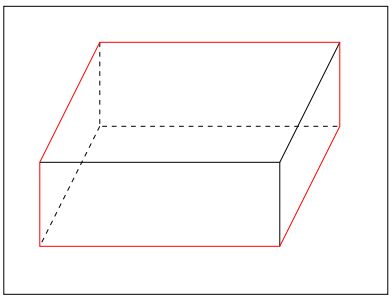

Figure 2.1: An object with a simple RE loop.

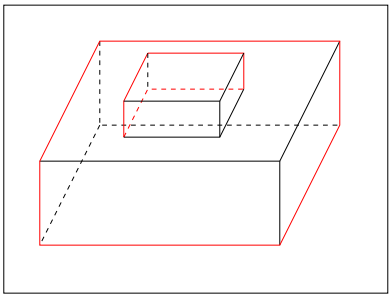

Figure 2.2: Two disjoint loops.

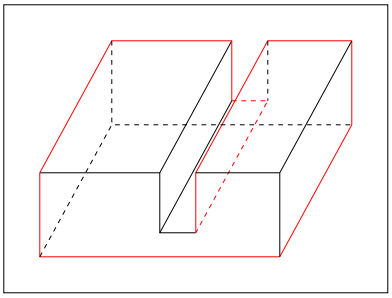

Figure 2.3: Single loop which is coiled twice.

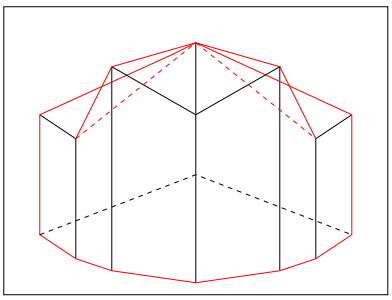

Figure 2.4: A loop coiled thrice. This can also be seen as three simple loops.

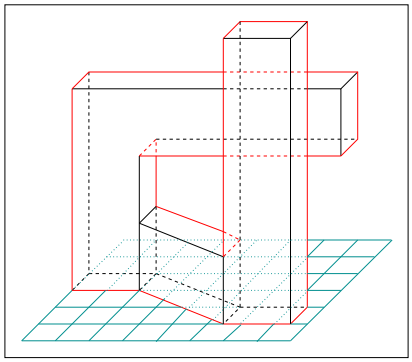

Figure 2.5: RE loop for a more complex object.

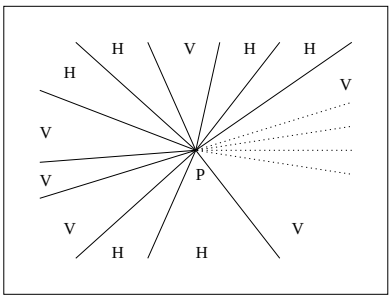

Figure 2.6: Many faces converging at P.

In the following we prove that REs always form a closed loop. Consider a vertex 'p' in Figure 2.6, where a number of faces are converging. Faces are labeled as 'V' or 'H' depending on whether they are visible or hidden. Now let us count the REs converging at 'p'. The edges between two visible (invisible) faces are not REs, so for counting purposes we can merge all the adjacent visible (invisible) faces. Then Figure 2.6 reduces to Figure 2.7. Now we see that any visible (invisible) face has two invisible(visible) neighbors, except for the case when we started with all faces visible (invisible). In any case, a face after merging, will have zero or two neighbors of other kind. So if there are n visible (invisible) faces after merging, there will be $2n$ REs (zero if $n = 1$). Thus at any vertex, only even number of REs can converge. This implies that there can not be any broken RE loop. (If a RE loop is broken at any vertex, there must be an odd number of REs). This completes the proof. Here we assumed that there are no dangling faces in the object and all objects are closed objects. These assumptions dose not pose any serious problem as any open object or an object with dangling faces can be treated as closed objects by introducing a few infinitesimally thin faces.

REs can be classified into two categories, based on the two faces they share. If the visible face is closer to the light source then the Ridge Edge is called a *visible ridge edge* (VRE). Otherwise, the ridge edge is called a *hidden ridge edge* (HRE). In this thesis VREs and

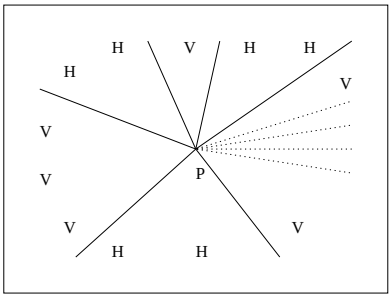

Figure 2.7: After merging.

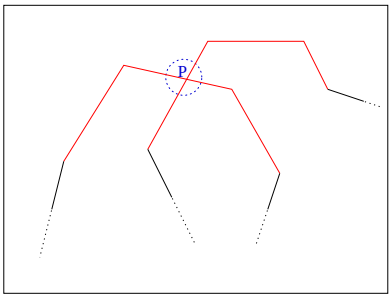

Figure 2.8: Two VREs having a PIP.

HREs have been shown by solid and dashed red lines, respectively. An HRE implies that there is an invisible face between the visible face and the light source. So there must be a shadow on the visible face, or in other words, shadow on the object itself. This classification is more useful when we deal with holes.

If two VRE's projections on a plane perpendicular to the direction of the light intersect, then the points on the VREs corresponding to the intersection point are called *visible ridge edge pseudo intersection points* (VREPIPs). At VREPIPs, two VREs do not really intersect but they seem to be intersecting as seen from the light source. Presence of VREPIPs implies that one visible face partially hides another visible face from the light source. Hence VREPIPs also imply shadow on the object itself. See Figures 2.8-2.9. If we join the VREPIPs in 3D by an imaginary line, VREPIPs will be used to break the coiled loops into simple loops.

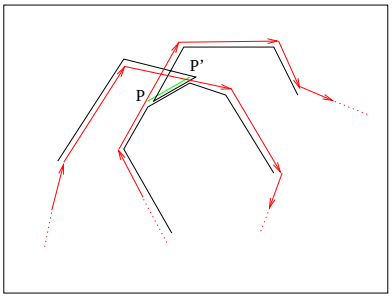

Figure 2.9: Two VREPIPs joined by an imaginary line. Separated simple loop segments are also shown.

2.2 RE Loop Traversing: separating internal and external loops

The role of an RE loop in some cases is quite clear. For instance, the smaller loop in Figure 2 bounds a piece of shadow on the object itself and the bigger loop bounds the object, as seen from the light source. The smaller loop is called an *internal (RE) loop* and the bigger loop is called an *external (RE) loop*. An external loop determines shadows generated on other objects. However, in most of the cases, the role of an RE loop can not be so clearly defined as it might contain RE segments from both groups. It is necessary to separate these segments so that new RE loops with clear roles can be constructed. This process can be accomplished by carefully traversing the RE loops.

We first obtain all the REs and VREPIPs. VREPIPs occur in pairs (see Section 2.4 for a remark). One is near the light source and another is far. An imaginary edge will be used to connect the corresponding VREPIPs. We start traversing the loops from any VRE, using a fixed clockwise or counterclockwise direction (with respect to the outward normals of the corresponding visible faces). When we hit a VREPIP, we split the edges at both VREPIPs. After splitting the edges we move from the current VREPIP to the other VREPIP along the imaginary edge, and then continue the traversing on the new edge that contains the other VREPIP in the same clockwise or counterclockwise direction of its adjacent visible face. See Figure 2.10. The traversing process of the current loop stops when the start point is reached.

We start the traversing of the next RE loop at the VREPIP where a new edge is selected, but move in the other direction of the imaginary edge. For example, in Figure 2.9, P would be the start point of the new loop, and we would move toward P' and then travel in the same clockwise or counterclockwise direction on the edge that contains P'. When the traversing of all the RE loops are finished, each RE will be traversed once, but each imaginary edge will be traversed twice, once in each direction. This traversing policy guarantees that one will always get the external loop first, and then the internal loops.

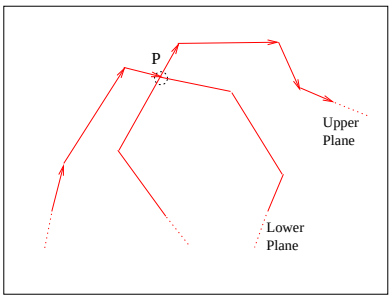

Figure 2.10: Traversing the external loop.

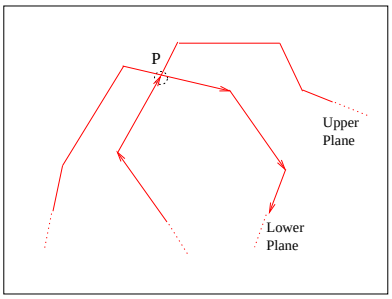

Figure 2.11: Traversing the internal loop.

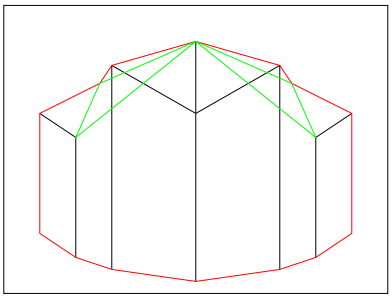

Figure 2.12: Separated internal (green) and external (red) loops of the object shown in Figure 4. Obviously, imaginary connecting lines are not visible.

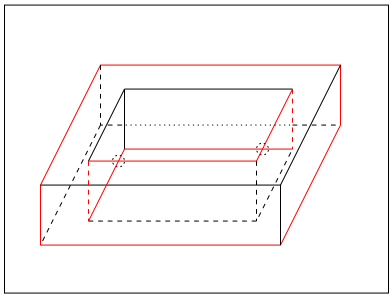

Figure 2.13: Hole's RE loop with VREPIPs. There is an external loop, light source can look through this object.

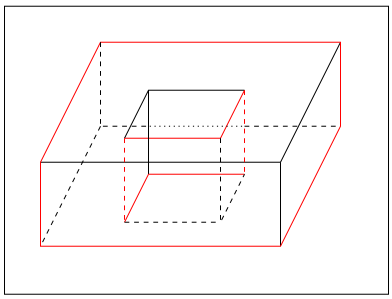

Figure 2.14: Hole's RE loop without VREPIP. Light source can not look through this object.

Figure 2.12 shows the separated loops for the object shown in Figure 2.4. The two internal loops (in green) will cast shadow on the object itself while the outer loop (in red) will cast shadow on other objects.

2.3 Objects with holes

If the object has a hole, there will be another loop of REs at the hole. We separate this loop also into simpler loops by means of VREPIPs. A loop containing HREs will cast shadow on the walls of the hole. The loop containing only VREs will be responsible for shadow on the rest of the world. The light source can look through this loop. See Figure 2.13. It is possible that there is no loop which contains VREs only. This situation is shown in Figure 2.14.

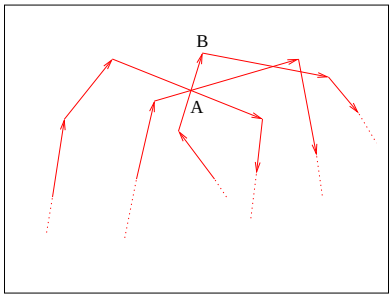

Figure 2.15: Three VREPIPs intersect at the same point.

2.4 Remarks

We make two remarks in this subsection. First, it is possible that the projections of more than two VREPIPs intersect at the same point. In the process of traversing RE loops, if one reaches such a VREPIP, one should move along the imaginary edge from that VREPIP to the VREPIP whose edge is closest to the incoming edge clock wisely, and continue the traversing on the new edge. For example, in Figure 2.15, when the point A is reached, one should continue the traversing process on the edge AB. Here, again, once the traversing of the current RE loop is finished, one should start the traversing of the next RE loop at the VREPIP where the new edge is selected, but move in the opposite direction of the imaginary edge.

Second, it is clear that to decompose RE loops into non-overlapping loops, the major expense is the process of finding the VREPIPs. One needs to perform extensive edge-edge intersection tests to find the VREPIPs.

This step of breaking RE loops into non-overlapping loops is required for the shadow volume based shadow generation algorithm for, otherwise, shadows that are supposed to be generated on an object itself might be generated on other objects, or, one might miss regions that are supposed to be in shadow.

This is an expensive process, not to mention the extensive ray-polygon intersection tests required in the subsequent scan-conversion process.

In the next section, we will show that by representing RE loops in an angular fashion, and storing the angular representations of the RE loops in a look up table, one can avoid the need of breaking RE loops into non-overlapping loops. Actually, the new representation avoids the need of performing ray-polygon intersection either because shadow polygons are no longer needed in the subsequent scan conversion process.

Chapter 3

Storing and Using the RE Loops

This chapter will discuss how we store the RE loops what are the difficulties and how do we solve them.

3.1 Basic idea

We know that the shadow of an object has angular symmetry. That is, the angle subtended on the light source by the shadow remains the same no matter where the shadow is produced. This angle is the same as the angle subtended by the shadow producing object on the light source. See Figure 3.1 where L is the location of a point light source and A is the shadow producing object.

We also know that the boundary of a shadow is determined by RE loops. So if we represent the RE loops in terms of angles, the same representation can be used to mark shadows on all objects. For example, consider a two dimensional case of Figure 3.1. Object 'A' will cast shadow from ϕ_{A+} to ϕ_{A-}. If a polar coordinate system with respect to the light source is used then to know if a point (r, ϕ) is in shadow, we check if ϕ is inside (ϕ_{i+}, ϕ_{i-}) for any object i. If this succeeds, we then compare distances to be sure. In the three dimensional case, we need to consider one more angle θ. To make the comparison process more efficient, a look up table which contains (r, θ, ϕ) representations of all RE loops from all objects will be constructed. The structure of the look-up table is shown in Figure 3.2.

(r, θ, ϕ) are defined as follows:

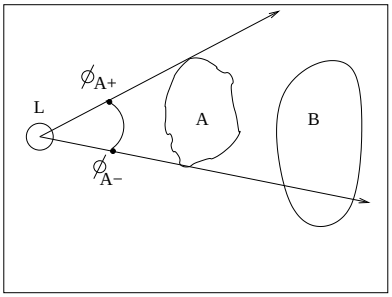

Figure 3.1: Angular span of shadow remains the same for all distances.

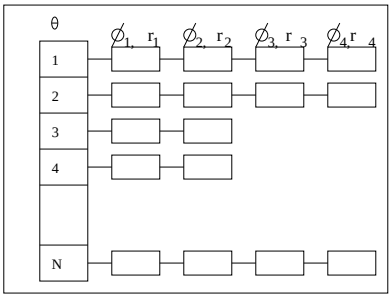

Figure 3.2: Structure of a look up table.

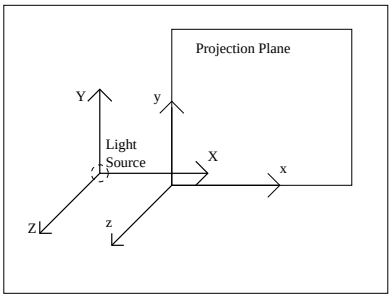

Figure 3.3: Coordinate system fixed to the light source.

$r = \sqrt{X^2 + Y^2 + Z^2}$
$\theta = tan^{-1}(\frac{Y}{-Z})$
$\phi = tan^{-1}(\frac{X}{-Z})$

X, Y and Z are measured in a coordinate system fixed with the light source which is parallel to the projection plane. Note that this is not the general spherical coordinate system. It can only represent points in one hemisphere uniquely. However, this definition of (r, θ, ϕ) has some properties we wanted. Any line parallel to the x-axis of the projection plane is a constant-θ line and any line parallel to the y-axis is a constant-ϕ line. See Figures 3.3 and 3.4. To find the (r, θ, ϕ) representation of an RE loop, we first find the intersection points of the loop with constant θ-planes. We then find the ϕ and r values of the intersection points. A constant θ-plane is a plane that passes through the light source, origin of the coordinate system, and is parallel to the x-axis.

This approach enables us to combine the look up process with the scan conversion process. When we scan convert a polygon, we first find the corresponding θ for the current scan line (y) then we find the spans (ϕ_{i+}, ϕ_{i-}) from the look up table entry at θ. Of course the above definition of θ will be a floating point ranging from -$\pi/2$ to +$\pi/2$. We scale it to a large range say 0-500 and then discretize it so that it can be used as an index. Choice of this range is a matter of a trade-off between quality and speed. Figures 3.5 and 3.6 illustrate two RE loops and their representations in the look up table. The RE loops are due to the

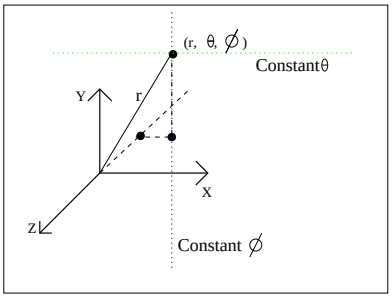

Figure 3.4: Constant-θ and ϕ lines are parallel to the X and Y axes, respectively.

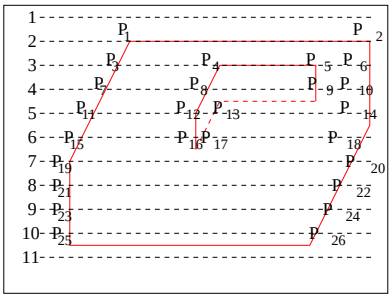

Figure 3.5: RE loops of the object in Figure 2.2. P_i are points on the loops.

object shown in Figures 2.2. Several things are to be noted here. The above loops range from θ=2 to θ=10. θ=0 in the table will correspond to a point in an RE loop which is a global (considering all loops of all objects) minimum in θ and θ=N will correspond to a point in an RE loop which is a global maximum in θ. Since all RE loops are potential source of shadow on all objects, we put representations of all loops in our table in any order. Every two consecutive pairs of ϕ and r in a table entry marks the beginning and end of shadow due to one loop for the corresponding θ. For example, in the entry $\theta = 4$ of Figure 3.6, the first two pairs of ϕ and r mark the beginning and end of shadow due to the external RE loop and the third and the fourth pairs of ϕ and r are the beginning and end of the shadow due to the internal RE loop.

3.2 Correctness of Pairing

The correctness of the above scheme relies on the correctness of pairing. As pairs mark the beginning and end of shadows, both elements of a pair must come from the same RE loop. For example, it will be incorrect to have list elements P_3, P_4, P_5, P_6 in the second entry of Figure 3.6. To prevent this from happening we must append RE loops in the look up table one at a time (in any order).

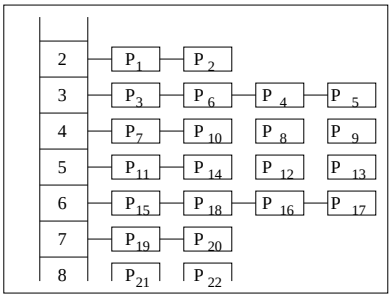

Figure 3.6: Representation of the RE loops in Figure 2.2.

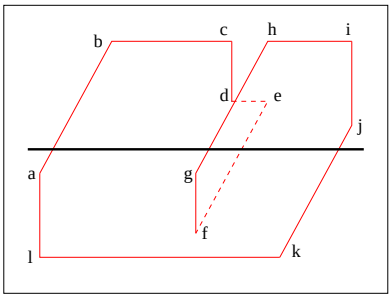

Figure 3.7: RE loop of the object shown in Figure 2.3.

Sometimes a single loop can contribute to more than one pair at the same θ. Again we must ensure correct pairing. Consider the object of Figure 2.3. Its RE loop looks like the one shown in Figure 3.7. In this case edge (a,b) must be paired with edge (e,f) and edge (g,h) must be paired with edge (j, k) for the shown constant-θ line. If we incorrectly pair (a,b) with (g,h) and (e,f) with (j,k) then the region from (g,h) to (e,f) will be interpreted as a hole, which is not the case. The cause of this problem is coiling of loop. A coil in general will look like the case shown in Figure 3.8. The problem is solved by the following scheme. Identify all local peaks in the loop (points A and B in Figure 3.8). Process edges adjacent to the same local peak together, one peak at a time, going as much down as possible each time. Two groups are merged into one group when they both reach the same local minimum. So in Figure 3.8 if we first picked 'A' then we will process edges from 'A' to 'C' and 'A' to 'D'. Then we pick point 'B' and process edges from 'B' to 'C' and from 'B' to 'D'. Both processes stop when the height of 'C' is reached. After that point, the edge from 'A' to 'D' is processed with the edge from 'B' to 'D'. The result is shown in Figure 3.9. We see as indicated by arrows that points are paired correctly. We will always avoid coiling by not allowing us to traverse an RE loop in the upward direction.

If two local peaks of the same height are adjacent to each other, the one on the left is ignored, we only consider the one on the right. The edge between them is also ignored. In this case, the other adjacent edge of the ignored local peak is considered as an adjacent edge of the kept local peak. For example, in Figure 3.12, only vertices i and c are considered

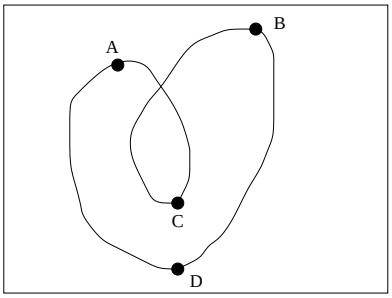

Figure 3.8: A coil in general.

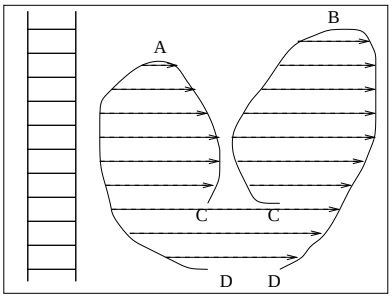

Figure 3.9: Appending two pairs of branches starting from peaks.

as local peaks, and edge (a, b) is considered as an adjacent edge of c and edge (g, h) is considered as an adjacent edge of i.

Improper pairing can also happen due to a hole or a hole-like situation, as shown in Figures 3.10 - 3.12. As we can see, if we follow the above scheme in these cases we will end up doing improper pairing as indicated above. To handle this situation we use the fact that a shadow is bounded by REs with opposite directions. Specifically, if the indices to faces are in clockwise order with respect to the outward normals, then all the RE loops will be clockwise as seen from the light source. So, left boundary must be going up and right boundary must be going down. However, inner boundaries of a hole or a hole-like situation is bounded by directions down and up from left to right. While following our aforesaid scheme we get points 'A' and 'B' as our starting points for Figure 3.10 and Figure 3.12. When we process branch pairs starting at 'B', once we reach the region bounded by edges (C, E) and (D, F), we know from their directions that they correspond to a hole or a hole-like situation. We insert these branch pairs instead of appending them. Any ambiguity is broken by these direction constraints.

We summarize our scheme for correct pairing here.

1. In any order but process one loop at a time. This takes care of wrong pairing due to two different loops.

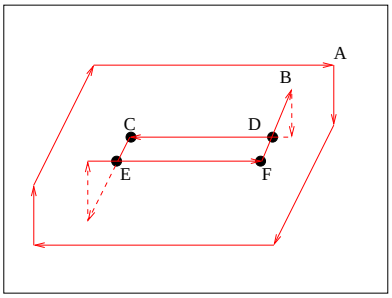

Figure 3.10: RE loops of an object with a hole.

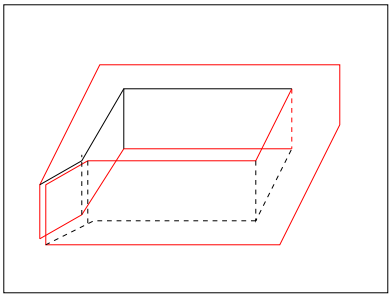

Figure 3.11: A hole-like situation in an object.

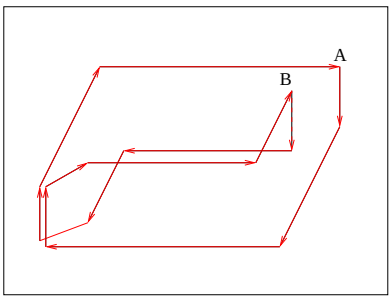

Figure 3.12: RE loop of object shown in Figure 3.11.

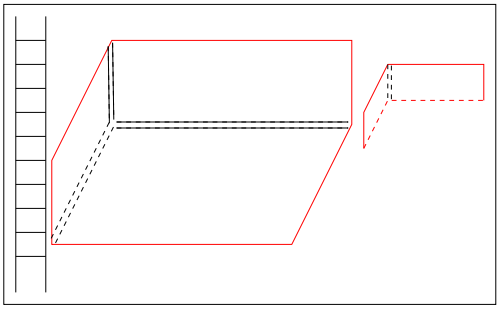

Figure 3.13: Complete representation of the object in Figure 2.2.

2. Identify local peaks in the loop. Divide the loop into branch pairs starting at these peaks. Process them one at a time. This takes care of wrong pairing due to coils.

3. Be wary of direction constraints while performing the above steps. This takes care of wrong pairing due to a hole or a hole-like situation.

3.3 Depth Information

So far our look up table contains correct information about shadow boundaries only. It has depth information of RE loops only. Since RE loops are not planer it is possible that a line joining two points of an RE loop may not pass entirely through the object. In other words, RE loops can not capture depth information about bumps or cavities in the object. To have an exact depth map we also include *hidden edges* in our look up table. A *hidden edge* is one whose adjacent faces are both hidden. Since a hidden edge does not mark the beginning or end of a piece of shadow, we insert them in duplicate to preserve the meaning of the look up table. For example, when we insert hidden edges of the object in Figure 2.2 into the look up table of Figure 3.6, the table contains the whole picture as in Figure 3.13. Now we can find the depth of a shadow producing object at any ϕ inside a pair using simple trigonometry as we know (r, ϕ) of the end points. We can even do a linear interpolation if the spanning angles are small. This is true because now every line between any two points given by pairs must entirely pass through the object.

Inserting hidden edges is not difficult. We maintain a graph like data structure for RE loops where each vertex of the RE loops also contains information about hidden edges incident to it. Hidden edges subsequently connect to other hidden edges. While appending/inserting an RE loop we also insert hidden edges at appropriate locations. Hidden edges have no direction so any ambiguity in placing them is resolved by their ϕ-values only. We can follow any commonly used graph traversing technique to make sure that we insert all hidden edges once and only once (in duplicate).

3.4 Using the look up table

Now that we have a look up table, we can very easily determine if a point is in shadow while doing scan conversion. We simply find out (r, θ, ϕ) of the point in question and look up in the table entry θ. If we find a pair (ϕ_1, ϕ_2) such that $\phi_1 \leq \phi \leq \phi_2$ then we find depth r(ϕ) from (r_1, r_2) and compare it with r-value of the point in question. If this test fails we go to the next pair. We stop when we find a match or we reach the end of the list. In fact the scan conversion process does not have to inquire for each pixel. The look up procedure can return a span of angles for an asked θ-value. For a given scan-line of a plane, we find coordinates (r, θ, ϕ) of starting point, we check the status of the point whether it is in shadow, then we find how long this status is valid. To do this, we find the next (nearest in ϕ) table entry. There is a problem however, as we reach this point on the scan-line, the depth can change and so can θ. It means that, our predicted span is no longer correct. To solve this problem, we find the distance we can go along scan-line without changing θ. Once this distance expires we make the lookup call again with a new θ value. We find this distance with the help of coefficients of the plane as follows.

The equation of plane is, aX + bY + cZ + d = 0
So, a + c$\frac{dZ}{dX}$ = 0 (for fixed Y)
Or, dX = $-\frac{c}{a}dZ$ (1)
But, tan(θ)= $-\frac{Y}{Z}$
So, dZ = $\frac{Z^2 sec^2\theta d\theta}{Y}$
But we have scaled and quantized theta, so we can assume maximum permissible $\Delta\theta$ for a scan-line is 1. With this, equation (1) becomes,
dX = $-\frac{cZ^2 sec^2\theta}{aY}$
This is the maximum distance that we can go safely with our prediction. As a first approximation, we use the current value of θ, Y and Z in above equation. As we saw in our implementation, this approximation is enough.

Chapter 4

Implementation

In this chapter we present the pseudo-codes of important steps. Then we discuss issues like performance, scalability and limitations.

4.1 Steps for Shadow Generation

We assume that objects are represented as polygons. We are given the indices to vertices in a fixed order (clockwise or anti-clockwise). For the simplicity of further discussion assume that vertices are in clockwise order. Edges are represented as vertex pair e(v_i, v_j) (preserving clockwise order). This section also assumes that light source is in the same hemisphere as viewpoint. More general case of light in scene will be discussed later. These pseudocods aren't necessarily the fastest, but they are intended to be easy to understand.

4.1.1 Identifying Visible/Hidden faces

For each object
    For each face of object
        Find the outward normal $\mathbf{N}$ of face.
        Find the direction $\mathbf{L}$ of light source from any vertex of face.
        Find the dot product $\mathbf{N} \cdot \mathbf{L}$.
        If ($\mathbf{N} \cdot \mathbf{L} > 0$)
          face = Visible
        Else
          face = Hidden.

4.1.2 Identifying Ridge/Hidden edges

For each object
    For each face of object
        If face is visible
          put all edges of this face in visible edge list VE.

Else
put all edges of this face in hidden edge list HE.
For all edges e(v_i, v_j) $\in$ HE
If e(v_j, v_i) $\in$ VE
transfer e(v_i, v_j) from HE to RE (Ridge Edge).
Rearrange elements of RE such that
$\forall e_i, e_j \in RE, e_i(v_2) = e_j(v_1)$

4.1.3 Building Look Up Table

For each object
For all RE loops found for an object
Convert all vertices to r, θ and ϕ representation.
Identify vertices with local peaks in θ.
Starting from these peaks append all the points of edges
in the look up table till a minimum in θ is reached.
Insert the hidden edges in duplicate in the look up table.
Do a pair wise sorting (in terms of ϕ) of all entries of the look up table.

4.1.4 Scan conversion of polygons

While scan converting for each scan line a quarry is made at the first point say (x, y, z) whether that point is in shadow. The callee returns the answer TRUE/FALSE and also resets a variable nextX. The caller then knows that the current state will remain valid till nextX. That is, it makes the next quarry only when it reaches nextX. The function which handles above quarry does the following:

Convert input x, y, z to r, θ and ϕ
If table entry at θ exists
nextX = x
For all pairs (ϕ_i, ϕ_j) of table entry
If $(\phi_i \leq \phi \leq \phi_j)$
nextX = equivalentOfX(ϕ_j)
return TRUE
Else if($\phi < \phi_i$)
tempX = equivalentOfX(ϕ_i)
If $(tempX > nextX)$
nextX = tempX
return FALSE
Else
nextX = END
return FALSE

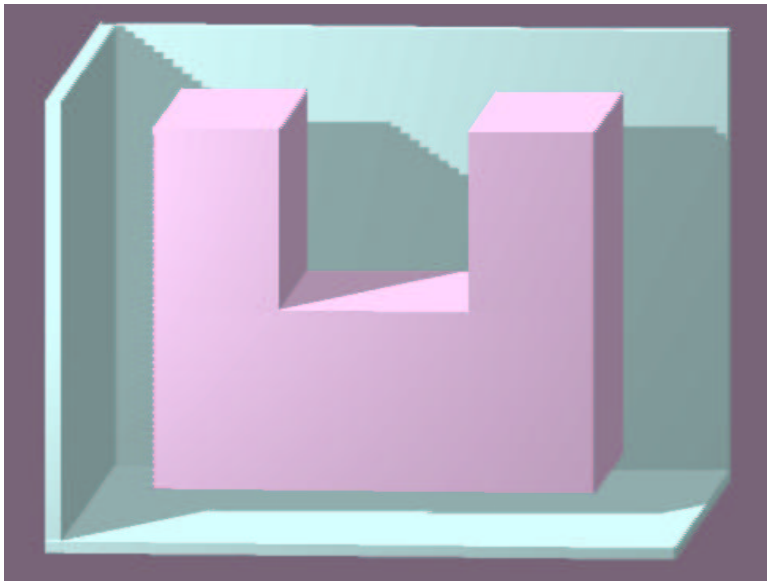

Figure 4.1: Self Shadow, Low resolution.

4.2 Performance

This algorithm was implemented on a Silicon Graphics machine (IP 32 @180MHz) using X-windows as the graphics system. The Z-buffer method was used for the scan conversion process. Figures 4.1-4.8 show some of the test cases.

The following table presents a comparison of performances for the presented test cases.

Figure	#Polygons	θ-Range	Table construction time (ms)	#Lookup calls	Look up time (ms)
4.1	26	0-099	8	18061	100
4.2	26	0-299	9	18454	102
4.3	26	0-599	25	16229	106
4.4	38	0-599	40	16834	116
4.5	330	0-599	44	21765	135
4.6	330	0-599	54	31552	184
4.7	330	0-599	15	25271	131
4.8	750	0-599	9	27953	134

All times are CPU time measured in millisecond. Number of ridge edges and complexity of shadow changes with change in light positions and viewing angles. In the table above what you see is the average over a few viewing angles and light positions. The table construction time starts when we read the data file and time stops when the lookup table is ready. The look up time listed in the last column is the total time elapsed in all look up

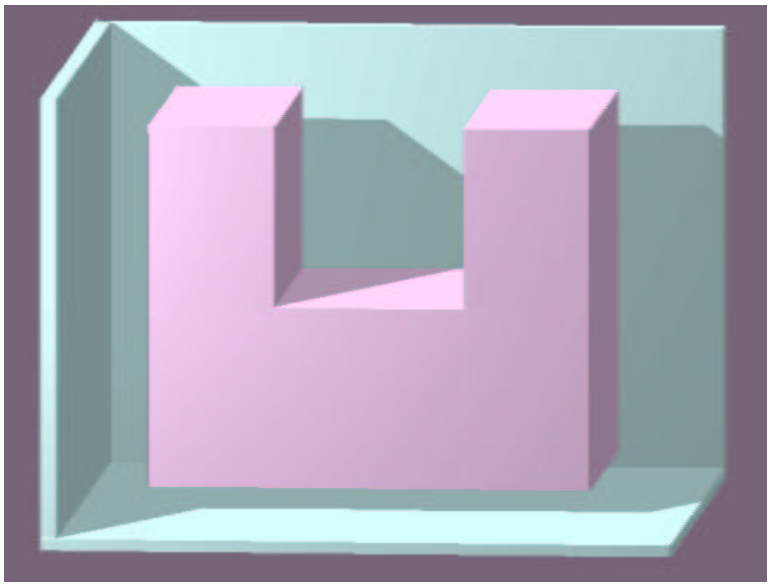

Figure 4.2: Self Shadow, Med resolution.

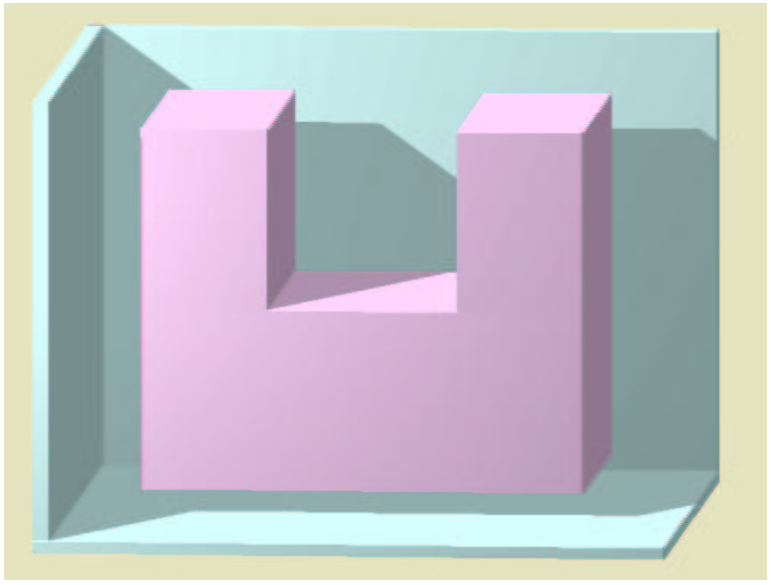

Figure 4.3: Self Shadow, Very high resolution.

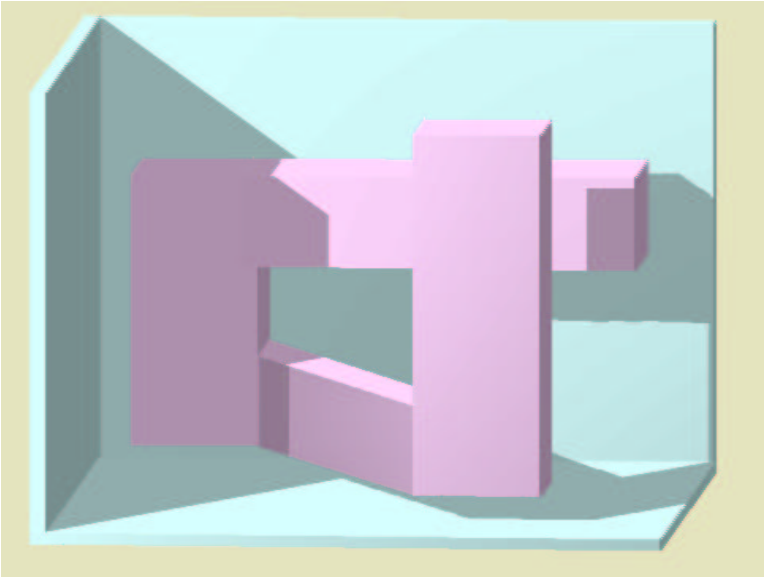

Figure 4.4: More complex object.

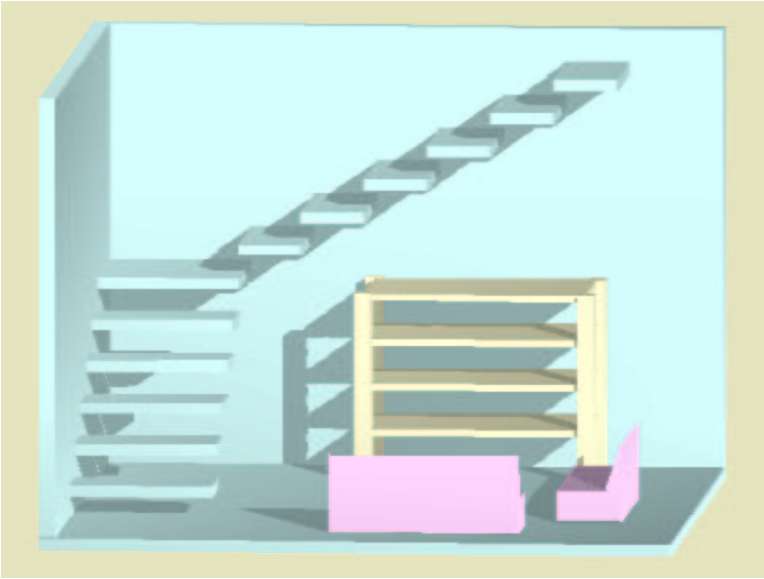

Figure 4.5: One light source.

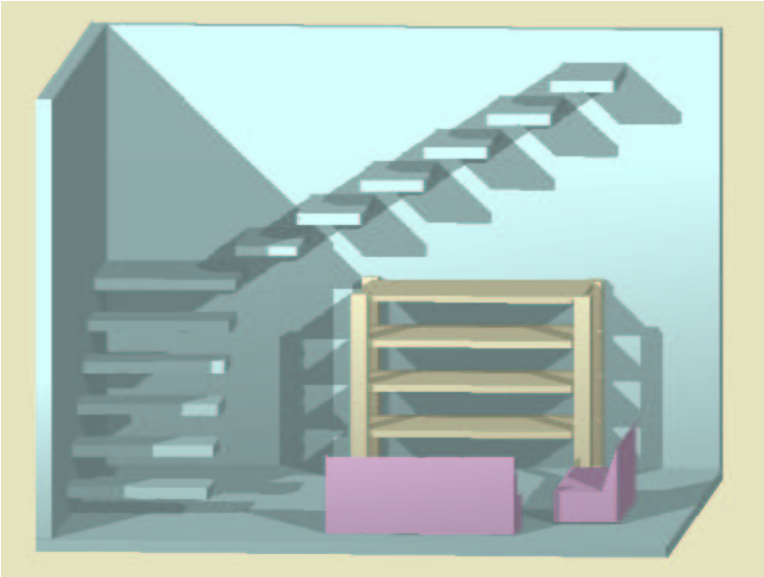

Figure 4.6: Two light sources.

Figure 4.7: Light in the center of the room.

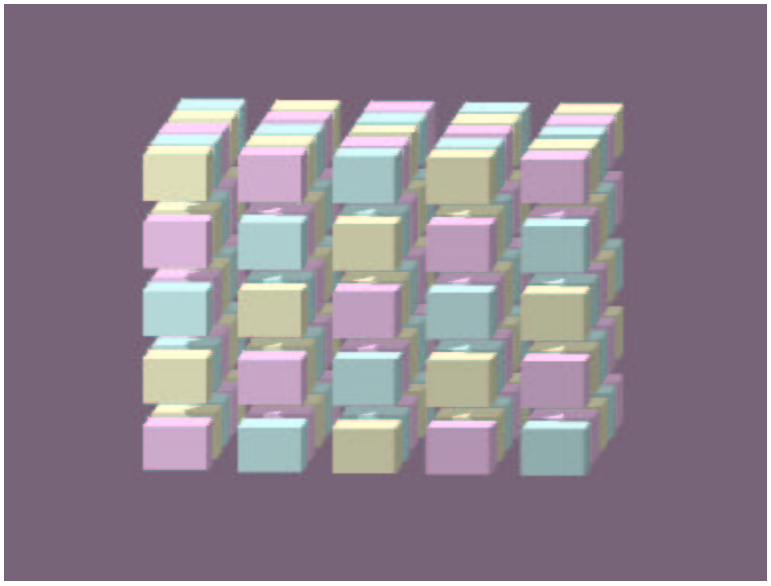

Figure 4.8: Simulation test case.

calls. As one can see in figure 4.1, the very low resolution in θ causes aliasing. By making it a little more expansive the quality improves much more as in figure 4.2. But increasing the resolution further does not make any perceivable improvement (see figure 4.3). Thus we see that there is an optimum value for resolution in θ. As we implemented the z-buffer method for scan conversion in software and realizing that this is the bottle-neck, performance will be much better if we use hardware graphic pipelines for z-buffer scan conversion.

4.3 Scalability

We did a simulation to test the scalability of our method. We assume that the number of elementary objects in a scene is a representative of complexity of the scene. We picked a cube as our elementary object so the data can be generated automatically. We experimented with a large number of cubes. Figure 4.8, shows the generated test case when the number of cube is 125 (#Polygons = 750). Figure 4.9 shows how much time is taken in lookup table construction and Lookup procedure[1] as the number of polygons grows. Figure 4.10 shows how much memory is occupied by lookup table[2]. In this simulation, the total volume of the scene was kept constant. This means that as the number of objects grew their dimensions became smaller. This was done so that we do not have to do any clipping.

[1]This time does not include time spent in other procedures like scene generation, scan-conversion etc.

[2]This does not include other memory requirements like a placeholder for scene itself, z-buffer for scan-conversion, RE loops etc.

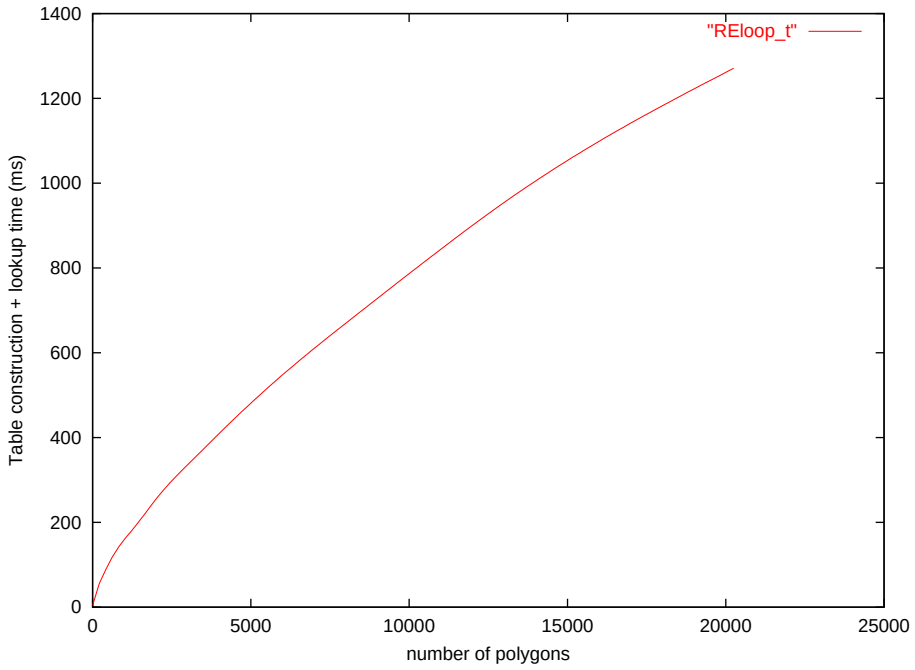

Figure 4.9: Simulation Result: Time taken.

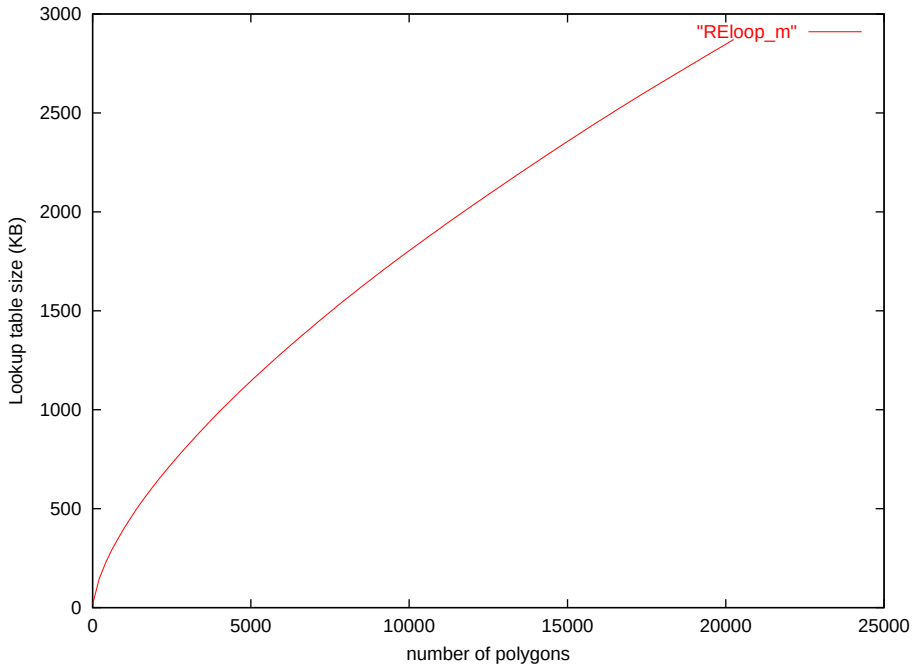

Figure 4.10: Simulation Result: Memory taken.

The best thing would be to get the performance in big-O notation. But there are so many variables that affect the speed that we end up doing too many approximations to get a result in closed form. This makes the result useless. We like to be rather sloppy and claim that "our method is very scalable as costs does not even grow linearly with scene size" as one can see from the plots. The speed of our method depends more on the total area of polygons rather than the number of polygons. This can be understood by considering the fact that table construction time is very small and individual look up time is even smaller (about $10\mu s$). Since the lookup table is made only once, total time depends upon how many look up calls we make. The number of lookup calls depend on the size of polygons and their slope. If the Z coordinate changes very rapidly with X for a given scan-line then θ will change many times for the same scan-line and we have to make more lookup calls. In the introduction, we claimed that our algorithm takes less than a second for a scene containing 2000 cubes "in the worst case". This is based on the fact that our simulation generates cubes in back to front order, i.e. we scan-convert faces of all the cubes leading to a maximum number of look up calls. The best case would be when the objects are already sorted in front to back order.

4.4 Limitations

While very promising in general, this method also have some limitations. In this section we discuss and propose solutions for some of these. This can be a good starting point for further research.

4.4.1 Good but unpredictable speed

This method, while performs very well in general, it can not be guaranteed for all scene. The performance depends on many factors like slope of polygons, position of light source and viewer etc. This is not a problem for non-real time systems. For real time systems, one solution might be to use some buffer and allow more time for expensive situations. Another solution would be to compromise with quality when ever needed to reach the target speed.

4.4.2 Not suitable for Texturing

To make the scenes more realistic texturing is always a good tool. However this method cannot support texturing in real time. The shadow output of this method is essentially line segments and applying texture for each line would be very expensive. One way to get textures would be to combine several scan lines to get a patch of area and then apply texture. Possibly this can also be made real time if we use specialized hardware for texturing part.

4.4.3 Not suitable for Light in scene

Due to many new tests introduced in look up procedure the speed reduces. But this does not pose a big problem as this kind of cases are rare.

Chapter 5

Special cases

In this chapter we are going to discuss how we handle special cases like more than one light source and light in the scene.

5.1 More than one light sources

This case is easier to handle as we just make one look up table for each light source. When a quarry for shadow is made we look in all tables for the corresponding θ and return the number of shadows instead of just TRUE or FALSE. The look up procedure obviously becomes a bit slower. Further, $nextX$ value set by look up procedure is the smallest of $nextX$ value obtained from all the tables. This means the scan conversion procedure will make more look up calls. This leads to another reduction in speed. We experimented with two light sources and found that overall speed reduces by $5 - 10\%$. Once again giving a sharp estimate is not possible as the speed will depend on scene. Figure 4.6, shows one result.

5.2 Light in scene

To handle this case we can proceed in the same way as described before while keeping track of objects in two hemispheres (separated by XY plane, in a coordinate system fixed with

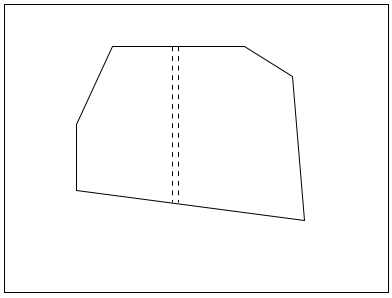

Figure 5.1: Segmenting RE loop

light source) separately. However situation becomes more complex as objects may go across the hemispheres. We divide the RE loops into two whenever they cross the XY plane. To do this we introduce two new vertices for each RE crossing the XY plane and connect them as shown by dashed lines in figure 5.1. During the scan conversion we also split the plane which crosses XY plane. These two steps essentially reduces the problem to a case when light is out of scene. There is a further problem however. Now the light source is at zero distance from objects and RE loops. This means we do have $\theta = \pm\frac{\pi}{2}$ or at least close to it. At this point $tan(\theta)$ will blow up. We must remain careful for this. Further, as we discretize and index our look up table based upon θ we need to have infinitely many table entries for exact result, which is obviously not possible. We can settle for very large number of table entries. But it is still wasteful, since we do not need extremely high resolution near $\theta = 0$. What we really do is to have a look up table with resolution varying with theta. All these combined together gives a slow but correct result. An example is shown in figure 4.7.

Chapter 6

Conclusion

An initial attempt to find efficient technique to identify shadow polygons turns into the process of developing a new shadow method for 3D polyhedra. The new method is based on building a look up table of the RE loops so that one can use the table in the scan conversion process to mark the pixels that are in shadow directly. The new approach avoids the need of performing ray-polygon intersection tests required in the classical shadow volume based approach. Since the RE loops are not required to be decomposed into non-overlapping loops, the new approach does not require edge-edge intersection tests to identify VREPIPs either. Several test cases were generated which gave satisfactory results. While correct for a case with light in scene this method is more suitable for cases with light away from scene if real time performance is sought. This and some other limitations open space for further research. This method can also be used in association with available graphics hardware for z-buffer and texturing to get much faster and realistic results.

Bibliography

[1] Appel, A., Some Techniques for Shading Machine Renderings of Solids, *Proc. AFIPS JSCC*, Vol. 32, 1968, 37-45.

[2] Atherton, P., Weiler, K. and Greenberg, D., Polygon Shadow Generation, *Computer Graphics* (*Proc. SIGGRAPH*) 12,3 (August, 1978), 275-281.

[3] Bouknight, J. and Kelley, K., An algorithm for Producing Half-Tone Computer Graphics Presentations with Shadows and Movablve Light Sources, *AFIPS Conf. Proc.*, Vol. 36, 1970, 1-10.

[4] Chin, N. and Feiner, S., Near Real-Time Shadow Generation Using BSP Trees, *Computer Graphics* (*Proc. SIGGRAPH '89*) 23,3(July 1989), 99-106.

[5] Chrysanthou, Y. and Slater, M., Shadow Volume BSP Trees for Computation of Shadows in Dynamic Scenes, *Proc. 1995 Symposium on Interactive 3D Graphics, SI3D '95,* April 9-12, 1995, Monterey, CA, ACM, 45-50.

[6] Cohen, M.F. and Donald, D.P., The HEMI-CUBE: A Radiosity solution for Complex Environments, *Computer Graphics* (*Proc. SIGGRAPH '85*) 19,3 (July 1985), 31-40.

[7] Crow, F.C., Shadow Algorithms for Computer Graphics, *Computer Graphics* (*Proc. SIGGRAPH '77*) 11,3(August 1977), 242-248.

[8] Foley, J.D., van Dam, A., Feiner, S.K. and Hughes, J.F., Computer Graphics, Principles and practice, 2nd Edition in C, Addison-Wesley, Reading, Mass., 1997, 614.

[9] Kutulakos, K. N., Exploring Three-Dimensional Objects by Controlling the Point of Observation, *Ph.D. thesis*, 1994.

[10] Nishita, T. and Nakamae, E., An Algorithm for Half-Tone Representation of Three-Dimensional Objects, *Information Processing in Japan*, 14 (1974), 93-99.

[11] Nishita, T. and Nakamae, E., Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection, *Computer Graphics* (*Proc. SIGGRAPH '85*) 19,3(July 1985), 23-30.

[12] Petrovic, L., Fujito, B., Williams, L. and Finkelstein, A., Shadows for Cel Animation, *Computer Graphics* (*Proc. SIGGRAPH '00*) 34,2(July 2000), 511-516.

[13] Seales, W. B. and Dyer, C. R., Shaded Rendering and Shadow Computation for Polyhederal Animation, *Proc. Graphics Interface '90, Canadian Image Proc. and Pattern Recog Soc.*, 1990, 9-16.

[14] Seales, W. B. and Dyer, C. R., Constrained Viewpoint from Occluding Contour, *Proc. IEEE Workshop on Directions in Automated CAD-based Vision.* , June 1991, 54-63.

[15] Segal, M., Korobkin, C., van Widenfelt, R., Foran, J. and Haeberli, P., Fast Shadows and Lighting Effects Using Texture Mapping, *Computer Graphics* (*Proc. SIGGRAPH '92*) 26,2(August 1992), 249-252.

[16] Thibault, W.C. and Naylor, B.F., Set Operations on Polyhedra Using Binary Space Partitioning Trees, *Computer Graphics* (*Proc. SIGGRAPH '87*) 21,4 (July 1987), 153-162.

[17] Williams, L., Casting Curved Shadows on Curved Surfaces, *Computer Graphics* (*Proc. SIGGRAPH*) 12,3 (August, 1978), 270-274.

[18] Woo, A., Poulin, P., and Fournier, A., A Survey of Shadow Algorithms, *IEEE Computer Graphics & Applications* 10 (November, 1990), 13-32.

Vita

Khageshwar was born in 1976 in Katihar, a small town in India. He received his M.S. in Physics from Indian Institute of Technology, Kanpur, India. At the time of writing this thesis he worked at Lexmark International, Inc. Lexington, Kentucky.