Shadow generation based on RE Loops and Their Angular Representations

Khageshwar Thakur and Fuhua (Frank) Cheng
University of Kentucky, Lexington, KY 40506-0046
Kenjiro T. Miura
Shizuoka University, Johoku, Hamamatsu, 432-8561 Japan

Abstract

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 are corresponding 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 techcique without the need of performing expensive tests.

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

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 obviously A is between B and the light source [4][7][9].

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 [16]. These methods can be classified as *shadow volume method*, *area subdivision method*, *depth buffer method* and *ray tracing method*. In a shadow volume method, shadow polygons are generated as preprocessing and then each ray is tested for shadow count [7]. Later this method was refined by BSP trees [4]. 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 [2][10], two passes of polygon clipping are used to calculate the region in shadow In a depth buffer method [15], a depth map from light source is used to generate shadows. In a ray tracing method [1][8], 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 it intersects with any other object then the point is in shadow.

All these methods have some advantages and disadvantages, but they have one thing in common, they do not scale well for large scenes. One reason is that they fail to exploit one or both of the following facts: (1) the actual shadow is a logical OR of shadows produced by all objects and (2) the angular span of a 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 chioce for this problem. In this paper, we present a completely new method for the classic problem of shadow generation for 3D polyhedra. This method, combined with the above facts, becomes much faster.

We start with the observation that shadows produced by an object are bounded by loops of *ridge edges* (REs). A *ridge edge* is an edge whose one adjecent face is visible and another is hidden from the light source. We further prove that ridge edges always form closed loops. 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 construct 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 is a limitation however, so far it works only for a scene confined in one hemisphere of the light source. To enhance this method to handle light-in-scene would be our future work.

The contributions of this paper 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 proposed new shadow generation algorithm can also be used to generate soft shadows and remove hidden edges and surfaces.

The remaining part of the paper is arranged as follows. Section 2 describes in detail of the RE loops traversing techniques, including external loops, internal loops, intersecting/overlapping loops, and loops corresponding to holes. In Section 3 we demonstrate how to use the results identified in Section 3 in the shadow generation process. Results of our test cases are presented in Seciton 4. Concluding remarks are given in Section 5.

2 Identifying Ridge Edge Loops

We will show in this section that shadows are corresponding to ridge edge loops. By (properly) 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. The 3D objects considered in this work are *3-manifolds*. Therefore, each edge is shared by exactly two faces and an object can not have *dangling edges* and *dangling faces*.

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 paper REs have been shown by red lines. REs always form a closed loop (will be proved shortly). Figure 1 shows an object with a very simple *RE loop*. For more complicated objects

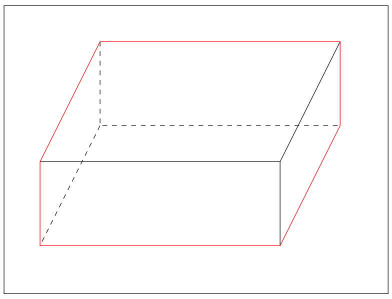


Figure 1: An object with a simple RE loop.

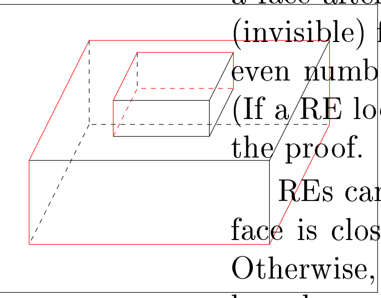
there can be more than one loop. These loops can be joint (through a vertex) or disjoint, overlapping or completely inside another loop. Figures 2-4 shows some of these situations. All the RE loops of a given object are potential source of shadow on all objects. If an RE loop is coiled more than once, we separate the loop into simpler ones via VREPIPs (to be defined later). In the following we prove that REs always form a closed loop.

Consider a vertex 'p' in Figure 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

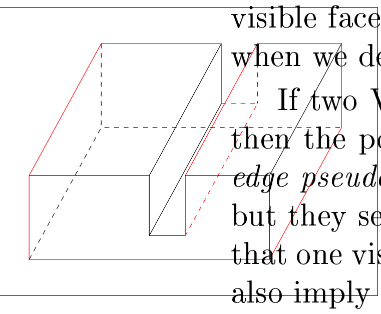
Figure 2: Two disjoint loops.

Figure 3: Single loop which is coiled twice.

for counting purposes we can merge all the adjacent visible (invisible) faces. Then Figure 6 reduces to Figure 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.



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 paper VREs and 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 8-9. If we join the VREPIPs in 3D by an imaginary line, VREPIPs will be used to break the coiled loops into simple loops.

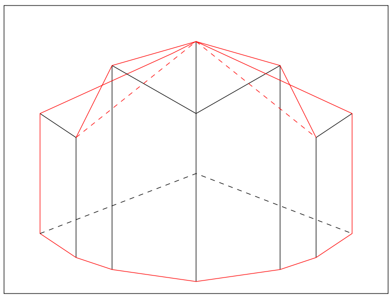


Figure 4: A loop coiled thrice. This can also be seen as three simple loops.

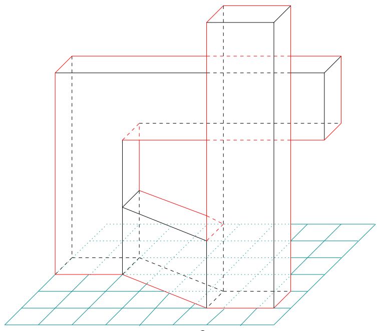


Figure 5: RE loop for a more complex object.

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 imaginery 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 10. The traversing process of the current loop stops when the start point is reached.

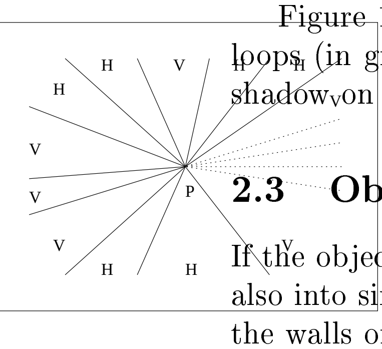
We start the traversing of the next RE loop at the VREPIP where a new edge is selected, but move in the other direction of the imaginary edge. For example, in Figure 9, P would be the start point of the new loop, and we would move toward P' and then travel in the same clockwise or counterclockwise direction on the edge that contains P'. When the traversing of

Figure 6: Many faces converging at P.

Figure 7: After merging.

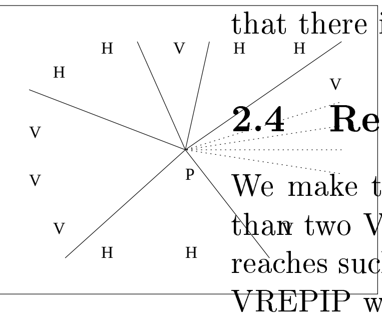
all the RE loops are finished, each RE will be traversed once, but each imaginary edge will be traversed twice, once in each diection. This traversing policy guarantees that one will always get the external loop first, and then the internal loops.

Figure 12 shows the separated loops for the object shown in Figure 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 13. It is possible that there is no loop which contains VREs only. This situation is shown in Figure 14.



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 clockwisely, and continue the traversing on the new edge. For example, in Figure 15, when the point A is reached, one should continue

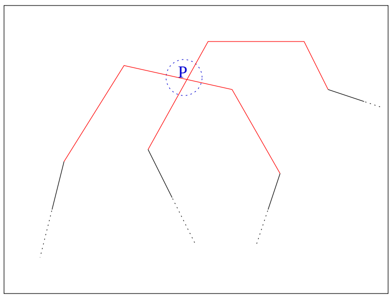


Figure 8: Two VREs having a PIP.

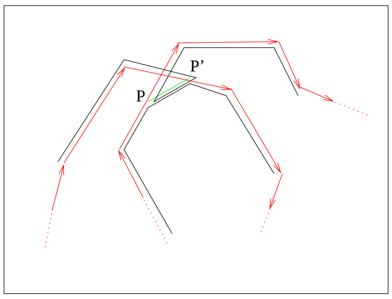


Figure 9: Two VREPIPs joined by an imaginary line. Separated simple loop segments are also shown.

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.

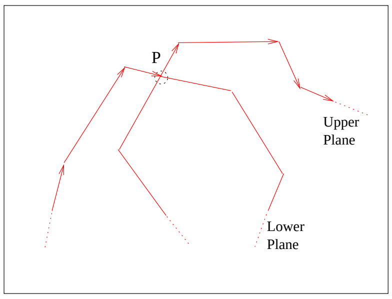


Figure 10: Traversing the external loop.

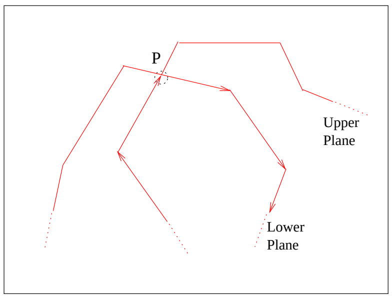


Figure 11: Traversing the internal loop.

3 Storing and Using the RE Loops

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 16 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 16. Object 'A' will cast shadow from ϕ_{A+} to ϕ_{A-}. If a polar coordiante 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 17. (r, θ, ϕ) are defined as follows:

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

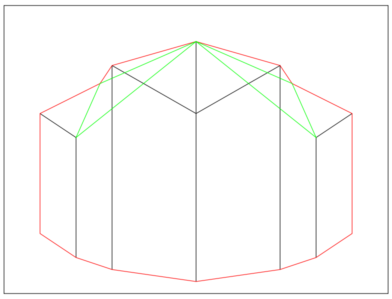


Figure 12: Separated internal (green) and external (red) loops of the object shown in Figure 4. Obviously, imaginary connecting lines are not visible.

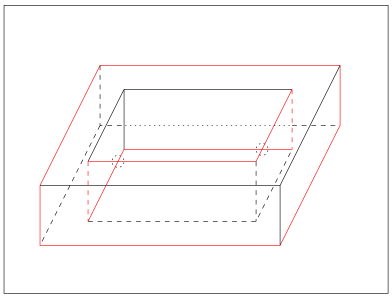


Figure 13: Hole's RE loop with VREPIPs. There is an external loop, light source can look through this object.

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-axix of the projection plane is a constant-θ line and any line parallel to the y-axis is a constant-ϕ line. See Figures 18 and 19. 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 20 and 21 illustrate two RE loops and their representations in the look up table. The RE loops are due to the object shown in Figures 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 corresponds 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

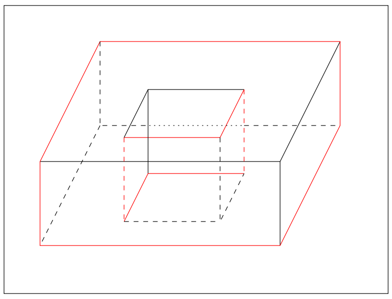


Figure 14: Hole's RE loop without VREPIP. Light source can not look through this object.

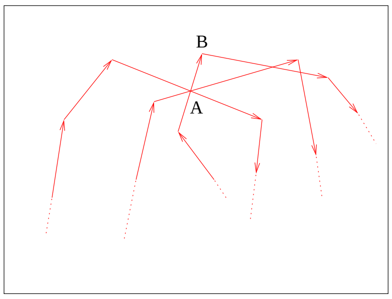


Figure 15: Three VREPIPs intersect at the same point.

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 21, 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 21. To prevent this from happening we must append RE loops in the look up table one at a time (in any order).

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 3. Its RE loop looks like the one shown in Figure 22.

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 23. The problem is solved by the following scheme. Identify all local peaks in the loop (points A and B in Figure 23). 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

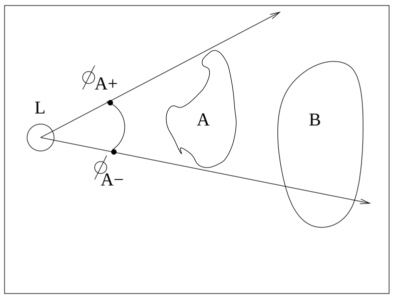


Figure 16: Angular span of shadow remains the same for all distances.

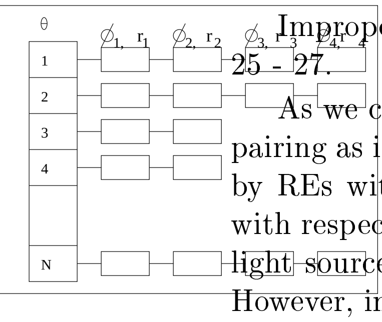


Figure 17: Structure of a look up table.

one group when they both reach the same local minimum. So in Figure 23 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 24.

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 27, only vertices i and c are considered 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 25 - 27.

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

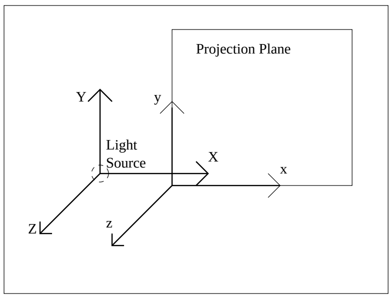


Figure 18: Coordinate system fixed to the light source.

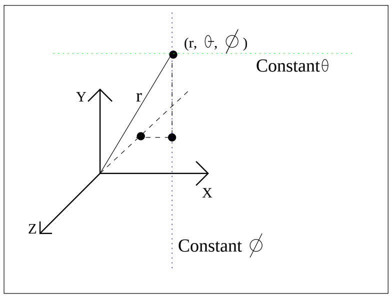


Figure 19: Constant-θ and ϕ lines are parallel to the X and Y axes, respectively.

starting points for Figure 25 and Figure 27. 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.

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

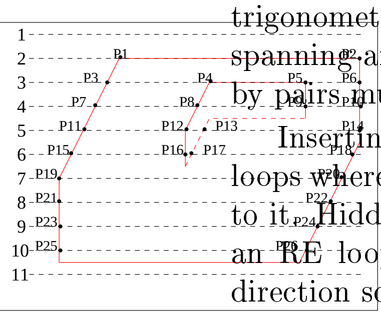


Figure 20: RE loops of the object in Figure 2. P_i are points on the loops.

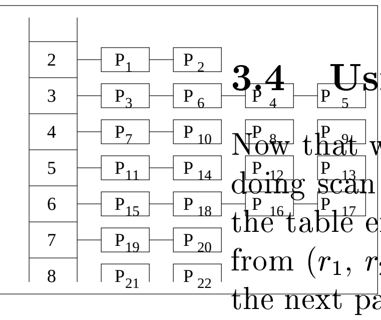


Figure 21: Representation of the RE loops in Figure 2.

table. For example, when we insert hidden edges of the object in Figure 2 into the look up table of Figure 21, the table contains the whole picture as in Figure 28.

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

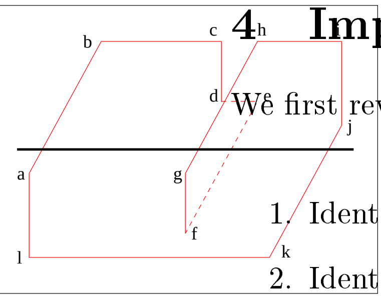


Figure 22: RE loop of the object shown in Figure 3.

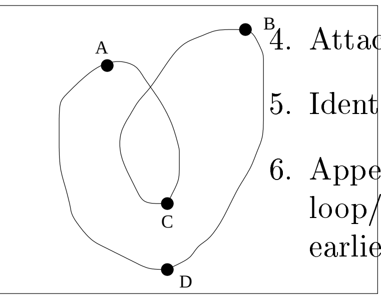


Figure 23: A coil in general.

a span of angles for an asked θ-value.

4 Implementation

We first review the steps involved in this algorithm.

1. Identify Visible/Hidden faces.
2. Identify Ridge/Hidden edges.
3. Construct an ordered circularly linked list of Ridge Edges (RE loop).
4. Attach Hidden edges to the vertices of the above list.
5. Identify vertices which are local peaks.
6. Append/Insert angular representation of all points (scaled and quantized in θ) of RE loop/Hidden edge starting from these peaks. Observe pair correctness scheme discussed earlier.
7. Use the above prepared look up table while doing scan conversion.

Figure 24: Appending two pairs of branches starting from peaks.

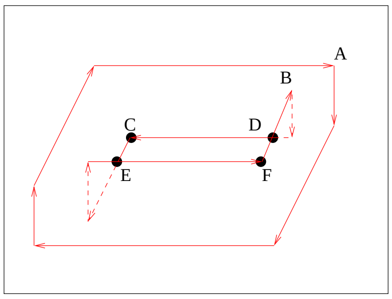


Figure 25: RE loops of an object with a hole.

This method is implemented on a Silicon Graphics machine using X-windows as the graphics system. The Z-buffer method is used for the scan conversion process. Figures on color plate show some of the test cases.

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

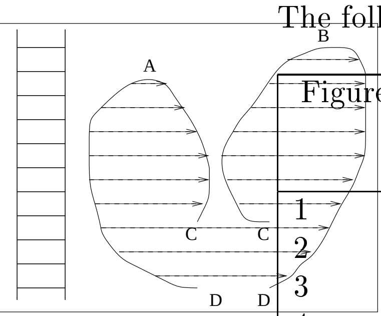


Figure	#Polygons	θ-Range	Table construction time (ms)	#Lookup calls	Look up time (ms)
1	26	0-599	25	16229	106
2	330	0-599	44	21765	135
3	38	0-599	40	16834	116
4	330	0-599	54	31552	184

All times are CPU time measured in milli seconds. 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 calls.

We did a simulation to test the efficieny and scalability of our method. We assume that complexity of a scene is related to the number of elementary objects in 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 29 shows the result.

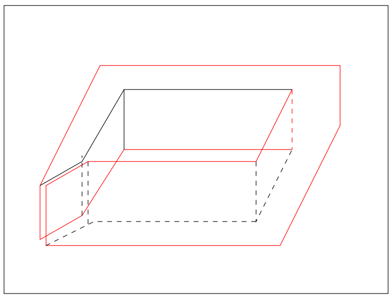


Figure 26: A hole-like situation in an object.

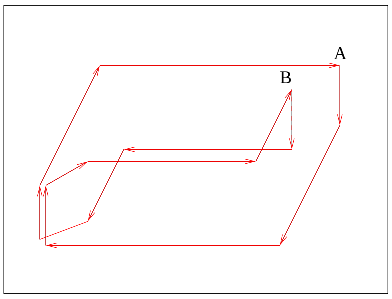


Figure 27: RE loop of object shown in Figure 26.

For comparison purposes, we also plotted the performance of BSP tree algorithm reported in Chrysanthou[1995]. This algorithm was implemented on SUN SparcStation 2. Assuming its clock speed to be 50MHz, our system is approximately 3.5 times faster. So for fairness, we devide the time reported in Chrysanthou[1995] by 3.5 before comparison. Of course a truly fair comparison is impossible since the performance depends on other system parameters also like memory etc. performance will also vary with different data set.

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 cost does not even grow linearly with scene size" as one can see from the plot. 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 rapidely with X for a given scanline then θ will change many times for the same scanline 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 scanconvert 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.

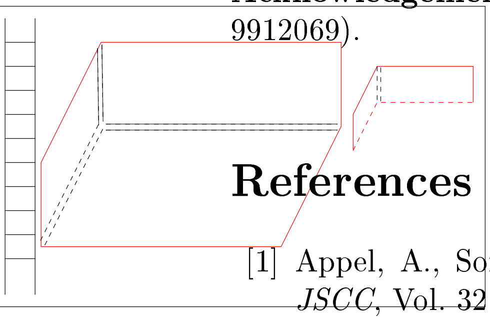
Figure 28: Complete representation of the object in Figure 2.

5 Conclusion

An initial attemp 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 required edge-edge intersection tests to identify VREPIPs either.

The current approach can not handle the case when the light source is inside the view volume yet. This will be a future research topic.

Acknowledgement Research work of the authors is partially supported by NSF (DMI-9912069).



References

[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.

Figure 29: Simulation Result.

[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] Goldstein, R. and R. Nagel, 3-D Visual Simulation, *Simulation*, Jan. 1971, 25-31.

[9] 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.

[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] 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.

[14] 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.

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

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