Shadow Generation Using Discretized Shadow Volume In Angular Coordinates

Khageshwar Thakur
Lexmark International, Inc.
Lexington, KY 40550
kthakur@lexmark.com

Fuhua (Frank) Cheng
University of Kentucky
Lexington, KY 40506
cheng@cs.uky.edu

Kenjiro T. Miura
Shizuoka University
Hamamatsu, 432-8561 Japan
ktmiura@eng.shizuoka.ac.jp

Abstract

A technique to improve the performance of the shadow-volume method is presented. This technique does not require (1) extensive edge/edge intersection tests and intersection angle computation in shadow polygon construction, or (2) any ray/shadow-polygon intersection tests during scan-conversion.

The first task is achieved by constructing ridge edge (RE) loops, an inexact form of silhouette, instead of the silhouette. The RE loops give us the shadow volume without any expensive computation.

The second task is achieved by discretizing the shadow volume into angular spans. The angular spans, which correspond to scan lines, are stored in a lookup table. This lookup table enables us to mark the pixels that are in shadow directly, without the need of performing any ray/shadow-polygon intersection tests.

In addition, the shadow on an object is determined on a line-by-line basis instead of a pixel-by-pixel basis. The new technique is efficient enough to achieve real time performance, without any special hardware, while being scalable with scene size.

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 viewpoint, but not from the light source. 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.

The problem of shadow generation has been studied for more than thirty years and various methods have been suggested. A good survey of these methods can be found in [5] and [9]. These methods can be classified as *Area Subdivision method, Ray tracing method*, *Depth buffer method* and *Shadow volume method*.

In an area subdivision method [1][6], 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 the viewpoint to remove the hidden faces. Due to the complexity of a good polygon clipper, this method could not gain much popularity.

In ray tracing method, a ray is shot from the viewpoint; 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. This straightforward method can capture shadows in direct, reflected and refracted lighting conditions. Since rays are traced from each visible point it becomes very expensive as the number of objects in the scene grows.

In depth buffer method [8], a depth buffer (with respect to the light source) is maintained to find lit faces. Lit portions of the faces are then rendered from the viewpoint using another depth buffer to remove hidden surfaces. Its simplicity and ease of hardware implementation make this method very popular even with its inherent aliasing problems. Recently, Fernando et al. [4] proposed Adaptive Shadow Maps to reduce aliasing problems in commonly used shadow map methods.

A shadow volume method [3] creates shadow polygons and places them into the scene as invisible surfaces. While rendering the objects, a ray from viewpoint to current pixel is pierced through these invisible surfaces. If the ray intersects the invisible surfaces odd number of times before it reaches a point in the scene, then that point is in shadow. This is also a very popular method because it can be easily integrated with the scan conversion process.

Many variations of this method have been proposed to improve its performance, including a BSP tree based approach [2][7]. None of these methods, however, can completely avoid the two most expensive steps in this process: (1) extensive edge/edge intersection and angle computation for shadow polygon construction, and (2) ray/shadow-polygon intersection tests for each pixel during scan conversion process.

In this paper, we present a technique that avoids these expensive steps and makes it easier to interleave shadow generation with scan conversion process. First, we find edges with exactly one hidden and one visible adjacent face. Such an edge is called a Ridge Edge (RE). As we prove later, REs always form a closed loop. RE loop makes an inexact form of silhouette and represents shadow boundary. Angular representations of shadow boundaries are discretized and stored in a lookup table. The lookup table is made in such a way that there is a direct correspondence between scan lines and lookup table entries. During scan conversion, lengths of shadow are found directly from the table. In this method, the memory and time requirements increase approximately linearly with a small constant factor, as the number of objects increases. Apart from avoiding the two most expensive steps of traditional shadow volume methods, this method also makes it possible to trade between speed and quality. This is particularly useful if real time performance is desired without special hardware support.

The remaining part of the paper is arranged as follows. In Section 2, definitions and basic properties of REs and RE loops are presented. Angular Representation and discretization of Shadow Volumes are presented in Section 3. Issues like Pair Correctness, Depth Information and Run Length Computation are discussed in Sections 4, 5 and 6 respectively. Section 7 summarizes the Shadow Generation Algorithm. Performance, Scalability, Limitations and Special Cases are discussed in sections 8, 9, 10 and 11 respectively. Concluding remarks are given in Section 12.

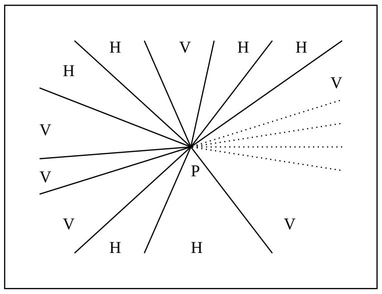
2. Ridge Edge Loops

A *shadow polygon* of an object is a semi-infinite polygon defined by the light source and a *silhouette edge* [3]. A portion of an object edge is a *silhouette edge* if it satisfies the following two conditions: (1) exactly one of its adjacent faces is a visible face [1] and (2) its perspective projection is a bounding edge of the object's perspective projection (with respect to the light source).

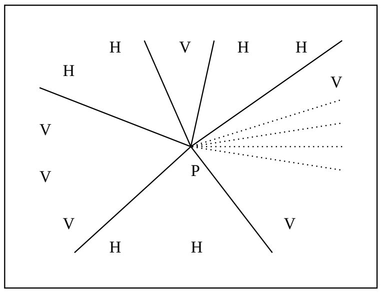
To check for the second condition, one needs to perform extensive edge/edge intersection tests and intersection angle computation. If the volume bounded by the shadow polygons (i.e., the *shadow volume*) is all one is interested in, then collecting object edges satisfying the first condition is sufficient. For future reference, such edges will be called *Ridge Edges* (REs) to distinguish them from the silhouette edges. Note that REs include silhouette edges as subset - a silhouette edge is either an RE or part of an RE. On the other hand, points of an RE are either on the boundary or inside the region bounded by the silhouette edges. Therefore the volume bounded by the semi-infinite polygons defined by the light source and the REs is exactly the same as the volume bounded by the shadow polygons of the object.

[1]A face is called a visible face 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 the following we prove that REs always form a closed loop for a closed object (without dangling faces). Consider a vertex 'p' in Figure 1(a), where a number of faces are converging. Faces are labeled as 'V' or 'H' depending on whether they are visible or hidden from the light source. We start by counting the REs that converge at 'p'. The edges between two visible (hidden) faces are not REs, so for counting purposes we can merge all the adjacent visible (hidden) faces. Then Figure 1(a) reduces to Figure 1(b). Now we see that any visible (hidden) face has two hidden (visible) neighbors, except for the case when we started with all faces visible (hidden). In any case, a face after merging, will have zero or two neighbors of other kind. So if there are n visible (hidden) 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 an RE loop is broken at any vertex, there must be an odd number of REs). This completes the proof.



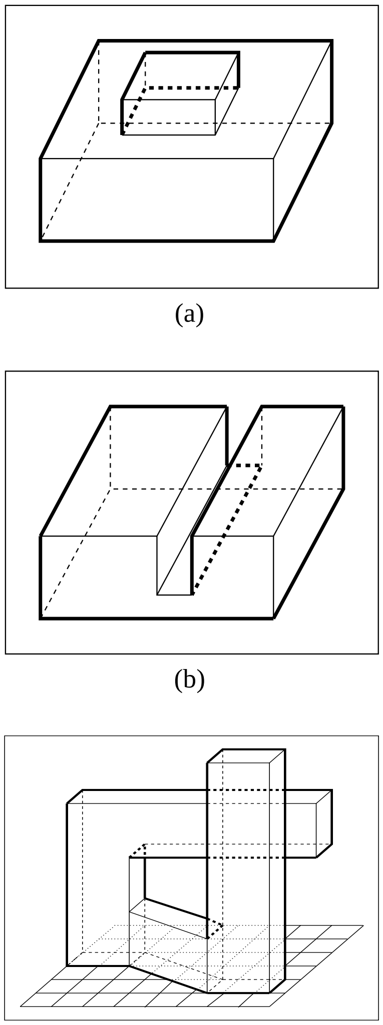
(a)



(b)

Figure 1. (a) Original faces converging at P. (b) Faces after merging.

An object can have more than one RE loop. The RE loops of an object can be connected or disjoint, overlapping or completely inside another loop. The RE loops of a given object represent a potential source of shadow on other objects or the object itself. Figure 2 shows *RE loops* of some objects.


(a)

(b)

(c)

Figure 2. (a) Two disjoint loops (one of them is an internal loop). (b) Single loop which is coiled twice. (c) RE loop for a more complex object. (Ridge Edges are shown by thick lines)

3. Discretization of Shadow Volume in Angular Coordinates

The shadow of an object has angular symmetry: the angle subtended on the light source by the shadow remains the same no matter where the shadow is produced. Hence, using angular coordinates for shadow representation is a natural choice.

We define angular coordinates (r, θ, ϕ) as:

$$r = \sqrt{X^2 + Y^2 + Z^2} \tag{1}$$

$$\theta = tan^{-1}(\frac{Y}{-Z}) \tag{2}$$

$$\phi = tan^{-1}(\frac{X}{-Z}) \tag{3}$$

where X, Y and Z are measured in a coordinate system with origin at the light source and XY-plane parallel to the projection plane.

This coordinate system is easily invertible. So it is easier to transform to and from the Cartesian coordinate system used in input data and scan conversion. In addition, any line parallel to the x-axis is a constant-θ line and any line parallel to the y-axis is a constant-ϕ line for fixed z. This enables us to easily integrate the lookup process with the scan conversion process.

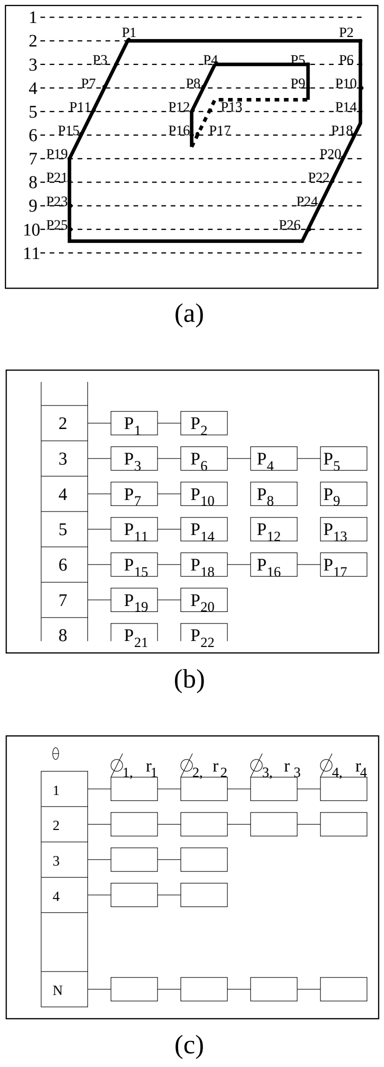
RE loops are conceptually sliced along θ. Each slice (which corresponds to one θ value) represents a span of shadow along ϕ. To do this, the range of θ (which is $\pm\frac{\pi}{2}$) is expanded to a large integral range say 0-600. This is essentially a discretization of shadow volume. Figure 3(a) shows an illustration of discretized RE loops of object shown in Figure 2(a). The integral range of expansion or the discretization parameter determines the speed, quality and storage requirement. Larger range takes more storage, more time and produces better quality output. Smaller range is faster and takes less memory but produces aliasing. Since the screen resolution is limited, there is an upper limit on the discretization parameter. Beyond this limit, there is no perceivable difference in output quality. Therefore, for optimal results, discretization parameter must be selected at this limit. This limit depends on the actual range of theta considering all objects in the scene.

At the limiting range there is a unique θ value for each scan line. For example, if the light source is far away from the scene then number of scan lines (or height in pixels) in the scene is a good estimate for the range. If the light source is close to scene then this range (R) can be estimated by this formula:

$$R = Z_f(tan(\theta_{max}) - tan(\theta_{min})) \tag{4}$$

where Z_f is the Z-coordinate of point farthest from light source and θ_{max} and θ_{min} are the maximum and minimum θ values in the scene. This range is computed at the set up time based on the view volume. In practice, once the range is determined, discretization of volume is merely an integer multiplication of θ values.

For easy computation of shadow, discretized shadow volumes in the form of angular spans, are stored in a lookup table index by θ. The structure of the lookup table is shown in



1
2
3
4
5
6
7
8
9
10
11
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
P11
P12
P13
P14
P15
P16
P17
P18
P19
P20
P21
P22
P23
P24
P25
P26
(a)
2 P_1 P_2
3 P_3 P_6 P_4 P_5
4 P_7 P_{10} P_8 P_9
5 P_{11} P_{14} P_{12} P_{13}
6 P_{15} P_{18} P_{16} P_{17}
7 P_{19} P_{20}
8 P_{21} P_{22}
(b)
θ
$\phi_1,\ r_1$ $\phi_2,\ r_2$ $\phi_3,\ r_3$ $\phi_4,\ r_4$
1
2
3
4
N
(c)

Figure 3. (a) RE loops of the object in Figure 2(a), P_i are points on the loops. (b) Representation of the RE loops in a. (c) General structure of a lookup table.

Figure 3(b). Figure 3(c) shows a general structure of lookup table. 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. 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(b), 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.

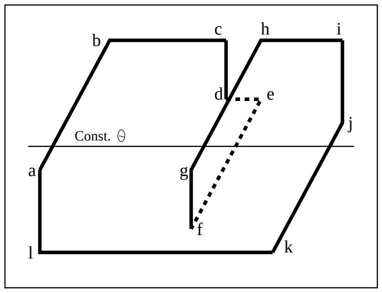
To check if a point (x, y, z) is in shadow, it is first converted to angular coordinates (r, θ, ϕ). An integral equivalent of θ is obtained based on the range defined above. Then the shadow is found from the lookup table entry at θ.

4. Correctness of Pairing

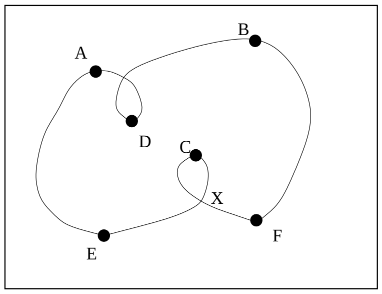
The correctness of the above scheme relies on the correctness of pairing. As pairs mark the beginning and end of shadows, both elements of a pair must come from the same RE loop. For example, it will be incorrect to have list elements P_3, P_4, P_5, P_6 in the second entry of Figure 3(b). To prevent this from happening we must append RE loops in the lookup 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 2(b). Its RE loop looks like the one shown in Figure 4(a). 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 4(b). The problem is solved by the following scheme. Identify all local peaks in the loop (points A, B and C in Figure 4(b)). Process edges adjacent to the same local peak together, one peak at a time, going as much down as possible each time. Groups without crossover (e.g. ones starting at A & B) are merged side by side. Groups with crossover (like the one starting at C) are inserted appropriately in the lookup table. For example, in Figure 4(b) we pick point A and process edges from 'A' to 'E' and 'A' to 'D'. Then we pick point 'B' and process edges from 'B' to 'D' and 'B' to 'F'. Then we pick point 'C' and process edges from 'C' to 'E' and 'C' to 'F' (this time we see a crossover of edges at X). Two groups starting at A and B are merged side by side while the group starting at C is switched between C and X then inserted into the lookup table. The final pairing result is shown in Figure 4(c). As indicated by arrows we see that points are paired correctly.

5. Depth Information

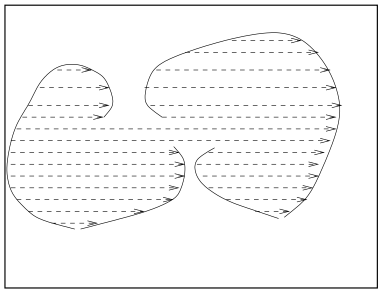
So far the lookup table contains correct information about shadow boundaries only. Since RE loops are not planar it is possible that a line joining two points of an RE loop may not pass entirely through the object. In other words, RE loops can not capture depth information about bumps or cavities in the object. To have an exact depth map we also include *hidden edges* in our lookup table. A *hidden edge* is



(a)



(b)



(c)

Figure 4. (a) RE loop of the object shown in Figure 3(b). (b) A coil in general. (c) Appending two pairs of branches starting from peaks.

one whose both adjacent faces are hidden from light source. 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 lookup table. For example, when we insert hidden edges of the object in Figure 2(a) into the lookup table of Figure 3(b), the table contains the whole picture as in Figure 5.

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. Note that inserting (in duplicate) visible edges (with both adjacent faces visible) will work as well.

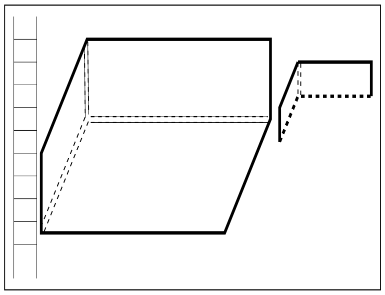


Figure 5. Complete representation of the object in Figure 2(a).

6. Calculation of Maximum Run Length

To find if a point (x, y, z) is in shadow, it is converted to angular coordinates (r, θ, ϕ). If there is no entry in the lookup table at θ then the point is not in shadow. If the bucket at θ is not empty then (ϕ, r) is compared with all the pairs ($\phi_i r_i$, $\phi_j r_j$) in the list. If there is a pair such that $\phi_a \leq \phi \leq \phi_b$ and ((r_a or r_b) $\leq r$) then the point in question is in shadow. If in shadow, the extent of shadow can be calculated from ϕ_b. If not in shadow, then the minimum distance of next shadow can be calculated from the smallest of ϕ_j (which is greater than ϕ). Thus a Run Length of shadow is obtained. This Run Length is good for current θ. For planes not parallel to XY-plane, θ will change with x. Therefore, the above Run Length may not be valid for entire scan line. Depending on the orientation of the plane, there is a maximum limit on the Run Length which is calculated as follows.

The Maximum Run Length (MRL) is the distance on a scan line for which θ stays the same.

Let

$$aX + bY + cZ + d = 0 \tag{5}$$

be the equation of the plane being scan converted, then for fixed Y we have

$$a + c\frac{dZ}{dX} = 0$$

Or,

$$dX = -\frac{c}{a}dZ \tag{6}$$

Since

$$tan(\theta) = -\frac{Y}{Z}$$

It follows that

$$dZ = \frac{Z^2 sec^2\theta d\theta}{Y}$$

With θ being scaled and quantized, we can assume maximum permissible $\Delta\theta$ for a scan-line is 1. With this, equation (5) becomes,

$$dX = -\frac{cZ^2 sec^2\theta}{aY} \tag{7}$$

dX is the Maximum Run Length.

7. The Shadow Generation Algorithm

The Shadow Generation process can be summarized as follows.
1. Build the Lookup Table:
1.1 Find REs
1.2 Connect REs to make RE loops
1.3 Get angular coordinates of all vertices on RE loops
1.4 Identify vertices with local peaks in θ.
1.5 Starting from peaks, append all the points of edges to the lookup table until a minimum in θ is reached.
1.6 Insert hidden edges (in duplicate) in the lookup table.
1.7 Do a pairwise sorting (in terms of ϕ) for all entries of the lookup table.
2. While scan conversion, for each scan line, a query is made at the first point say (x, y, z) to check if that point is in shadow. The function which handles the above query does the following:

```
Convert input x, y, z to r, $\theta$ and $\phi$
If table entry at $\theta$ exists
  nextX = END
  For all pairs ($\phi_i, \phi_j$) of table entry
      If ($\phi_i \leq \phi \leq \phi_j$ AND $r_i \leq r$)
        nextX = xEquivalentOf($\phi_j$)
        return TRUE
      Else if($\phi < \phi_i$)
        tempX = xEquivalentOf($\phi_i$)
        If ($tempX < nextX$)
          nextX = tempX
  return FALSE
Else
  nextX = END
  return FALSE
```

Thus callee returns an answer TRUE/FALSE and also resets a variable *nextX*. The current state of shadow will remain valid till *nextX* for current θ.
3. Calculate MRL.
4. Depending on the return value, create or don't create shadow up to nextX or x+MRL which ever comes first.
5. Make the next query for shadow.

As evident from the above code fragment; the search for shadow stops the very first time an instance of shadow is found. This comes from the fact that the actual shadow is a logical OR of shadows due to all objects. It is needless to search for the entire list. This makes the lookup process much faster, specially in dense scenes. It contrasts with traditional Shadow Volume methods where each shadow polygon must be intercepted and counted.

8. Performance

This algorithm was implemented on a Silicon Graphics machine (O2 @180MHz) using X-windows as the graphics system. Figures 6-9 show some of the test cases.

As one can see in figure 6(a), the very low resolution in θ causes aliasing. With the increase of resolution the quality improves as in figure 6(b). But increasing the resolution further does not make any perceivable improvement in quality as there is an upper limit posed by screen resolution(see figure 7(a)). We also implemented our algorithm on a Windows machine with Pentium 4 processor @1.3GHz (Metrox Millennium G450 32 MB card) using OpenGL as graphics system. We achieved a frame rate of (approximately) 29 frames per second for a scene containing 540 polygons and a moving light source.

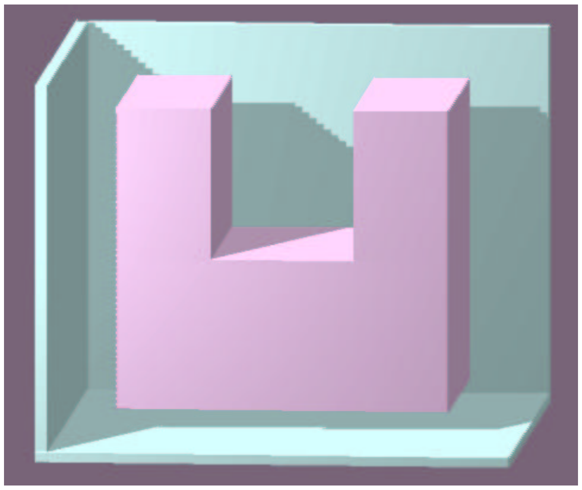
The lookup table construction time is very small and individual lookup time is even smaller (about $10\mu s$ on SGI machines). Since the lookup table is made only once, total shadow generation time depends on how many lookup calls are made. The number of lookup calls depend on the size of polygons and 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 requiring more lookup calls. Above all, z-buffer method for scan conversion was found to be the major bottle neck. So the performance could be improved further by using hardware graphic pipelines for z-buffer scan conversion.

9. Scalability

To test the scalability of this method a simulation was done. Assuming that the number of elementary objects in a scene is a representative of complexity of the scene, a large number of cubes were rendered. Cube was picked as elementary object so the data can be generated automatically. (This experiment was done on a Silicon Graphics machine mentioned above.) Figure 9(b), shows the generated test case when the number of cube is 125 (#Polygons = 750). Figure 10(a) shows how much time is taken in lookup table construction and all Lookup calls[2] as the number of polygons grows. Figure 10(b) shows how much memory is taken by lookup table[3]. In this simulation, the total volume of the scene was kept constant. This means that as the number

[2]This time does not include the time spent on other procedures like scene generation, scan-conversion etc.

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


(a)

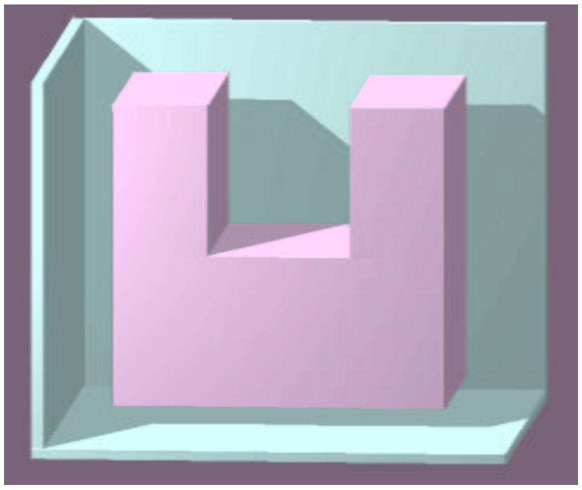
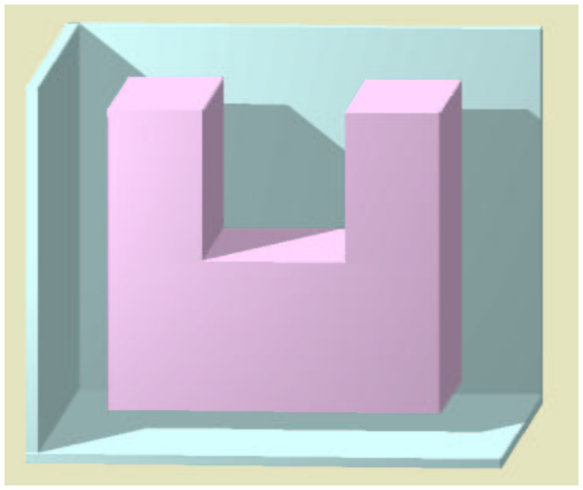

(b)

Figure 6. (a) Low resolution (θ = 0-99) & (b) medium resolution (θ = 0-299).


(a)

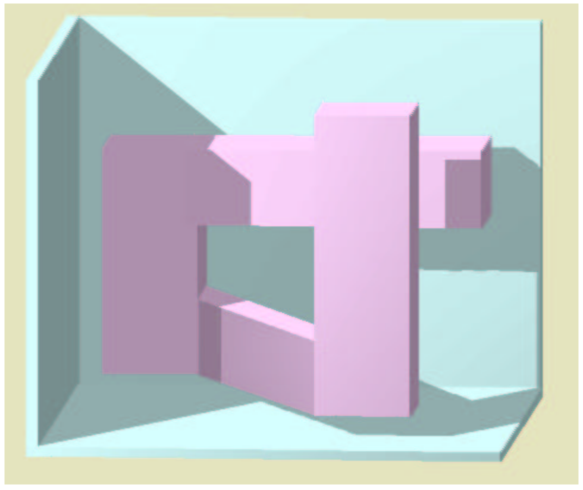
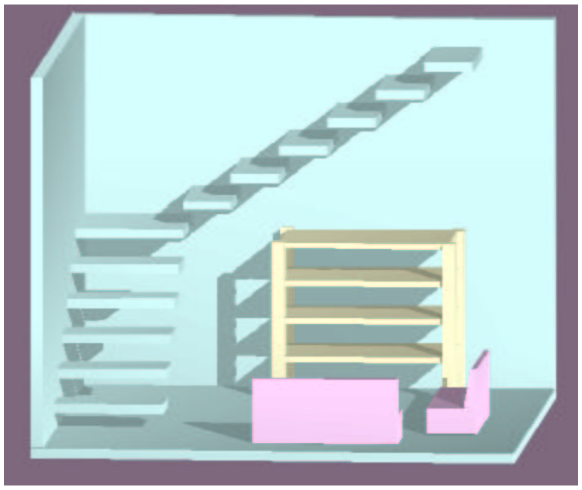

(b)

Figure 7. (a) Very high resolution (θ = 0-599) & (b) more complex object (θ = 0-599).


(a)

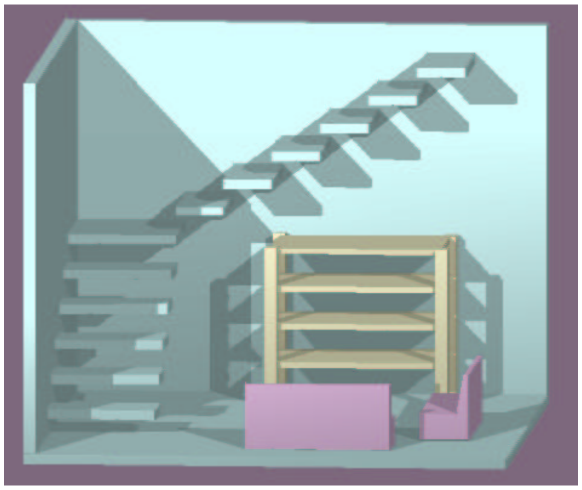
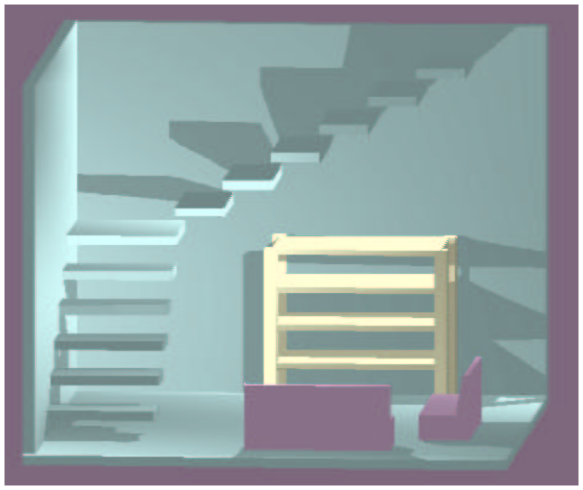

(b)

Figure 8. Multiple objects with (a) one light source & (b) two light sources (θ = 0-599).


(a)

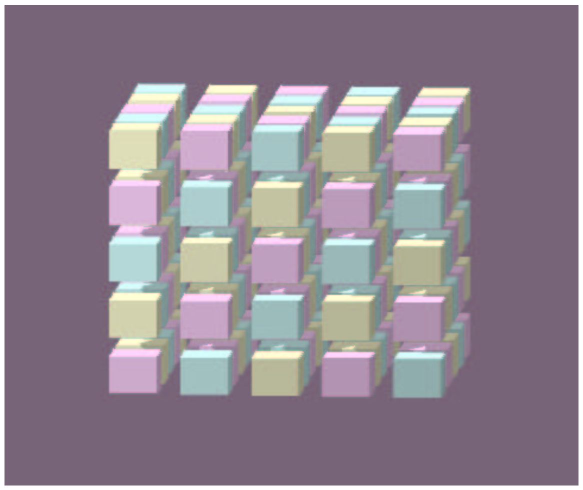
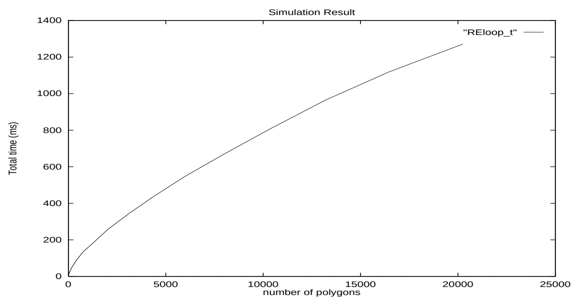
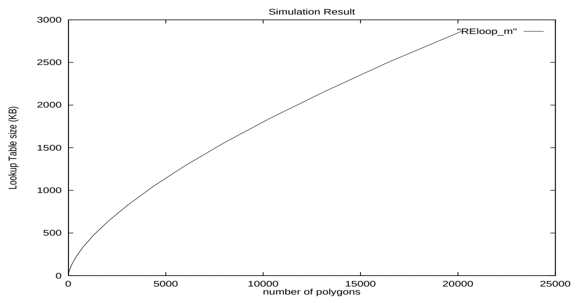

(b)

Figure 9. (a) Light in the center of the room & (b) simulation test case (θ = 0-599).

of objects grew their dimensions became smaller. This was done to avoid the need for any clipping. As one can see



(a)



(b)

Figure 10. Simulation Results: (a) Time taken. (b) Memory taken.

from the plots, for large number of objects, both storage and processing time grows approximately O(N) where N is the number of objects in the scene. Thus its storage requirement compares with BSP tree method, which also requires O(N) in storage. But it does better on processing time which for BSP tree method is O(N^2).

10. Future work

Being the first of its kind, this method provides a good ground for future research and development. Some of them are described here.

One drawback of this method is that its storage requirement (like other Shadow Volume methods) grows with the number of objects. However, this method may become more amenable in storage management than its peers. As pointed out earlier, the actual shadow is a logical OR of shadows from all objects. Using this fact greatly reduces the number of lookup calls and so the processing time. Similarly, this concept can also be applied to storage. That is, if there is one entry in the lookup table that encompasses other entry then we need to use only one of them. This way the lookup table will eventually become saturated and will not grow any further. This will require some extra search at the lookup table construction time but it will pay off in storage and also in shadow generation as there are fewer elements to look for.

In this method the speed depends on many factors like slope of polygons, position of light source and viewer etc. causing a non-uniform speed. For moving scenes, it might be a good idea to add a variable time delay to get a smooth output.

To make the scenes more realistic texturing is a good option. The shadow output of this method is essentially line segments which makes texture application difficult. One easy way to get textures would be to combine several scan lines to get a patch of area and then apply texture. Possibly textured shadow can also be made real time if we use specialized hardware for texturing part.

11. Special cases

11.1. More than one light source

This is handled by making one lookup table for each light source. When a query 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. Further, $nextX$ value set by lookup procedure is the smallest of $nextX$ values obtained from all the tables. This means the scan conversion procedure will make more lookup calls. These lead to a slight reduction in speed. We experimented with two light sources and found that overall time increases by about 10%. Figure 8(b), shows a scene with two light sources.

11.2. Light source in the 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 light source) separately.

We split the RE loops whenever they cross the XY-plane. During the scan conversion we also split the polygons crossing XY-plane. These two steps essentially reduces the problem to a case when the light source is out of the scene. As from Equation 4, a large discretization parameter is needed near $\theta = \pm\frac{\pi}{2}$. A variable resolution lookup table is made for this situation. Lower resolution is used near $\theta = 0$ and a higher resolution is used near $\theta = \pm\frac{\pi}{2}$. An example is shown in figure 9(a).

12. Conclusion

This paper presents an improved shadow volume method for shadow generation which can achieve real time performance. This technique identifies and uses the boundary of shadow rather than interior of shadow. It exploits the angular symmetry of shadow by building a lookup table which contains angular representations of shadow boundaries or RE loops. As the actual shadow is a logical OR of shadows due to all objects, searching the entire table for shadow is not needed. This makes the use of lookup table more appropriate. Several test cases including multiple light sources and light source in the scene are generated. This technique can also be used in association with specialized graphics hardware for z-buffer to get even faster results. Simulation results indicate the suitability of this method for larger scenes. For low cost systems like a game box, where a fast moving scene is desired without a high performance processor, this method allows some trade offs between speed and quality.

References

[1] P. Atherton, K. Weiler, and D. Greenberg. Polygon shadow generation. In *Proceedings of SIGGRAPH 1978*, Computer Graphics Proceedings, Annual Conference Series, pages 275–281. ACM, ACM Press / ACM SIGGRAPH, 1978.

[2] N. Chin and S. Feiner. Near real-time shadow generation using bsp trees. In *Proceedings of SIGGRAPH 1989*, Computer Graphics Proceedings, Annual Conference Series, pages 99–106. ACM, ACM Press / ACM SIGGRAPH, 1989.

[3] F. C. Crow. Shadow algorithms for computer graphics. In *Proceedings of SIGGRAPH 1977*, Computer Graphics Proceedings, Annual Conference Series, pages 242–248. ACM, ACM Press / ACM SIGGRAPH, 1977.

[4] R. Fernando, S. Fernandez, K. Bala, and D. P. Greenberg. Adaptive shadow maps. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 387–390. ACM, ACM Press / ACM SIGGRAPH, 2001.

[5] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and practice*. Addison-Wesley, 1990.

[6] T. Nishita and E. Nakamae. An algorithm for half-tone representation of three-dimensional objects. In *Information Processing in Japan*, pages 93–99, 1974.

[7] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In *Proceedings of SIGGRAPH 1987*, Computer Graphics Proceedings, Annual Conference Series, pages 153–162. ACM, ACM Press / ACM SIGGRAPH, 1987.

[8] L. Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH 1978*, Computer Graphics Proceedings, Annual Conference Series, pages 270–274. ACM, ACM Press / ACM SIGGRAPH, 1978.

[9] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. In *IEEE Computer Graphics & Applications*, pages 13–32. IEEE, 1990.