Shadow Volumes

1 Definitions

For a 3D object, by its shadow volume we mean the set of all points that are in its shadow. We are going to represent the boundary of the shadow volume using a polygonal mesh. The most important part of the boundary separates the empty space that is lit from the empty space that is in shadow. It will be representred using *shadow polygons*. By convention, we will orient the shadow polygons so that they appear front facing when looked at from outside of the shadow volume.

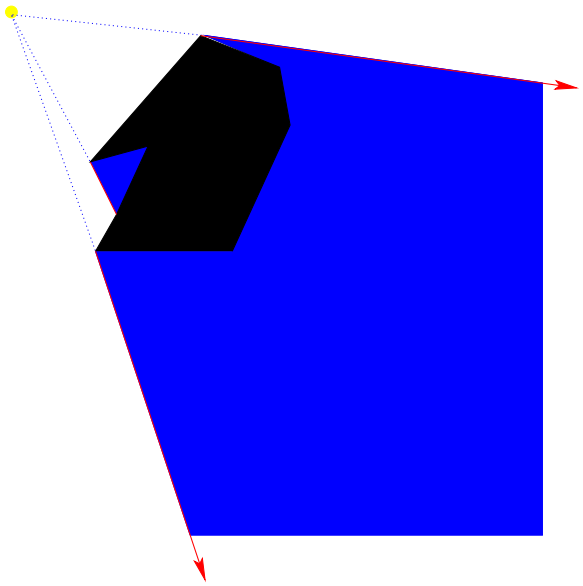

Figure 1: The shadow volume of the black object is shown in blue - it consists of all 3D points that are in its shadow; note that the shadow volume extends to infinity. The 1D counterparts of shadow polygons are shown in red. Formally, they also extend indefinitely.

2 Examples

For simple geometric primitives (like triangles and spheres) the shadow volumes and polygons are also simple.

2.1 Sphere

For a sphere, the shadow volume can be thought of as a cone whose apex is at the light source and which tightly encloses the sphere. This cone can be obtained by 'drawing' all rays originating at the light source and tangent to the sphere. If only points beyond the tangency are drawn, we obtain the surface that separates the lit points from points in shadow of the sphere. The shadow polygons should approximate this surface. Ideally, the shadow polygons should extend indefinitely. A possible way to obtain the shadow polygons is to compute the circle on the sphere which is the locus of all tangency points for rays originating from the light source (this circle is shown in red in the Figure). This circle can be approximated with an n-gon with large enough number of edges. The shadow polygons will be the shadows of the edges of these n-gons. Of course, They will typically be truncated somewhere far away beyond the scene.

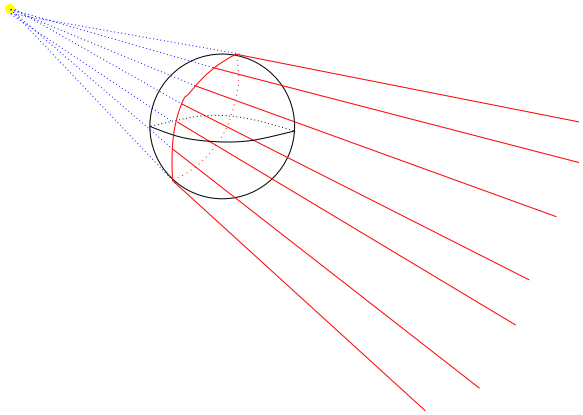

Figure 2: The shadow polygons for a 3D sphere.

2.2 Triangle

For a triangle, the shadow volume is an unbounded prism and the shadow polygons are basically shadows of the triangle's edges.

3 Shadows with Shadow Volumes

Say you are given a number of triangles and spheres in 3D and you want to draw them with shadows. For simplicity, assume that there is just one point light source. The geometric principle behind the shadow volumes is that one can decide whether a visible point p on one of the primitives is in shadow or is lit by counting the intersections of the shadow polygons with the ray originating from the viewpoint and passing through p that occur between the two points.

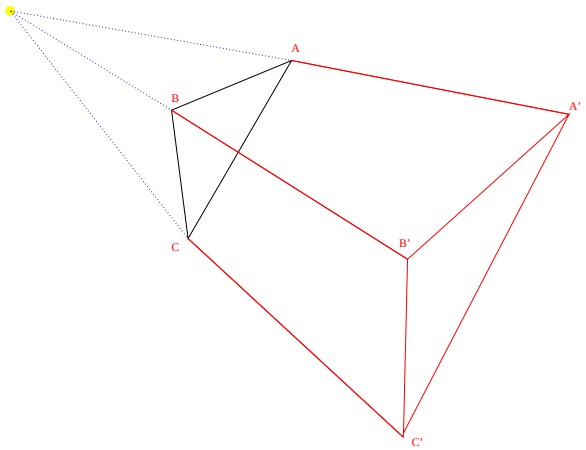

Figure 3: The shadow polygons for a triangle consists of three quads, AA'B'B, BB'C'C and CC'A'A. The points with primes can be obtained bu moving the points without the primes in the direction away from the light. Ideally, they should be moved infinitely far away, but in practice we'll just move them far enough, i.e. so that they are somewhere beyond the scene

Intersections where the ray enters a shadow volume are counted with a '+' sign and the ones where it exits - with '-' sign. If the total count is zero (which means that the ray exits a shadow volume the same number of times as it enters one) then p is lit. If the number of entries is greater than the number of exits (meaning that the sum of signed intersections is positive) then p is in shadow.

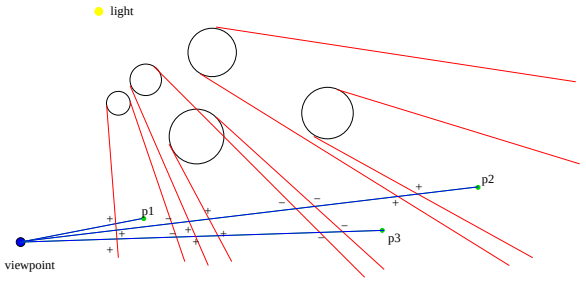

Figure 4: Using shadow volumes to tell shadowed from lit. The shadow polygons are shown in red. The sums of signed intersections for p1,p2 and p3 are 1,2 and 0 (respectively). Therefore, p1 and p2 are in shadow while p3 is lit.

The procedure as describes above works only if the viewpoint is lit. If there exist primitives that cast shadow on the viewpoint, one has to adjust the counting according to how many of them there are. For example, if there is exactly one primitive that casts shadow on the viewpoint, the rays reaching other points from the viewpoint need to exit the shadow volume of that polygon to reach the

lit portion of the space. Therefore, if the count of signed intersections is 0, we are still in shadow. However, if it is -1, we are lit. In general, if the viewpoint is in shadow of N primitives, a point p is lit if and only if its count of signed intersections is $-N$.

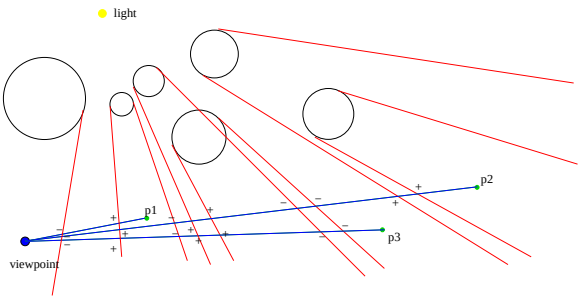

Figure 5: There is one primitive that shadows the viewpoint. The signed intersection counts for p1,p2 and p3 are 0, 1 and -1. Only points whose count is -1 are lit (in this case, just p3).

3.1 Implementing shadow volumes using Stencil Buffer

Stencil buffer is yet another buffer that you can use in OpenGL. Its most commonly seen function is to restrict drawing to certain (arbitrarily shaped) area/collection of pixels, but its specific meaning can be highly application-dependent. In our case, stencil buffer will be used for counting the intersections of rays with the shadow volumes and restricting drawing to lit pixels.

One can associate stencil test and stencil operations with stencil buffer. The stencil test can is performed at the end of the graphics pipeline, but before the depth test. If it fails, the pixel is not written into the frame buffer. If it passes, the pixel is written (unless it fails the depth test). This test can take the form of any comparison of the value stored in the frame buffer to any fixed pixel-independent reference value. It is also possible to set it to always pass or always fail.

The stencil operations allow to alter the values in the stencil buffer depending on the result of stencil and depth test. More precisely, the programmer can specify an operation performed in the following events:

1. Stencil test fails

2. Stencil test passes and depth test fails

3. Stencil test passes and depth test passes

The available operations include a lazy one that does nothing and increment and decrement operations (these are most important for shadow volumes).

The positive intersections (the points where the ray enters a shadow volume) can be distinguished from negative ones (points where the ray exits a shadow

volume) by properly orienting the shadow polygons and switching between front and back face culling (remember that culling is done before all per-fragment operations, in particular before stencil and depth tests). If all shadow polygons are oriented so that they appear front facing when looked at from outside their shadow volume, the positive intersections happen at shadow polygons that are front facing (since the eye ray enters the shadow volume there, meaning that the eye is outside it). Similarly, the negative intersections happen at shadow polygons that are back-facing. Here is the algorithm (assuming the viewpoint is not in shadow):

1. Clear all buffers (depth, color and stencil).

2. Draw the scene (without shadow polygons) using only ambient lighting. Back face culling can be turned on, the depth test set to 'less' or 'less or equal'.

3. Draw all the shadow polygons. But before you draw them, make the color and depth buffers read only (this means that the shadow volumes will not be dislayed). Also, set the stencil test to always pass, turn on back-face culling and set the stencil operation to increment if both stencil and depth tests pass (if one of the tests fails, just keep the previous value).

4. Draw all the shadow polygons again, but this time with front face culling turned on and with stencil operations set to decrement if the stencil test passes and depth test passes.

5. Draw the scene again, this time with full lighting. Before, turn on stencil test, setting it to pass only if the stencil value is zero. Also, set the depth test to less or equal. And, of course, make the depth and color buffers writeable.

How does it work? Drawing the shadow polygons actually allows to count the intersections of the rays with shadow polygons. In step 3, only front-facing polygons are scan-converted. This causes only the entry points/pluses to be counted. Similarly, in step 4, only back-facing shadow polygons are scan converted and counted in the stencil buffer as negative (since the stencil operation decrements) - thre the points where the ray exits a shadow volume. The counting is done only if the depth test passes. Therefore, we actually count only the intersections that happen between the viewpoint and the point on one of the primitives (which are drawn in step 2) that is visible through a pixel. Thus, just before step 5, the stencil buffer contains the signed intersection counts. The scene is drawn in two passes (step 2 and step 5). In the first pass, we pretend that everything is in shadow (only ambient light is used). In the second pass (step 5) we overdraw the parts of the scene that are lit (because of the stencil test) using full lighting.

The only change we need to make if we wish to handle the case of the viewpoint being in shadow is to clear the stencil buffer to a value different from zero (equal to the number of shadow volumes that contain the viewpoint).

4 Larger triangle models, shadow volumes and silhouette edges

In principle, the approach as discussed above can be applied to any collection of triangles, including a large triangulated model. However, this might produce very complex collection of shadow polygons, much larger than necessary. Many 3D models are manifold, in particular have two triangles incident upon each edge. For such meshes, typically a lot of immediate cancellations occur when counting the intersections with shadow polygons. The following figure shows a 2D picture showing what is going on. The dotted red lines separate shadows cast by adjacent triangles. When the blue ray intersects one of the dotted lines, it actually intersects shadow polygons of two triangles, exiting the shadow of one of the two and, at the same time, entering the shadow of the other one. Thus, these intersections don't contribute anything to the final count of signed intersections (since they are be counted with opposite signs). The situation is different, however, for the solid red lines: there, the ray enters shadows of two triangles simultanously (this happens at the two leftmost intersections) or exits two simultanously (the rightmost two). In the 3D case, the solid lines correspond to *silhouette edges*, i.e. edges for which one of the incident triangles faces the light and the other one - faces away from the light. In an implementation of shadow volumes for manifold surfaces, one can skip all shadow polygons corresponding to non-silhouette edges (they cancel with each other anyway when doing signed intersection counting). This allows to considerably decrease the number of shadow polygons and speed up the whole process.

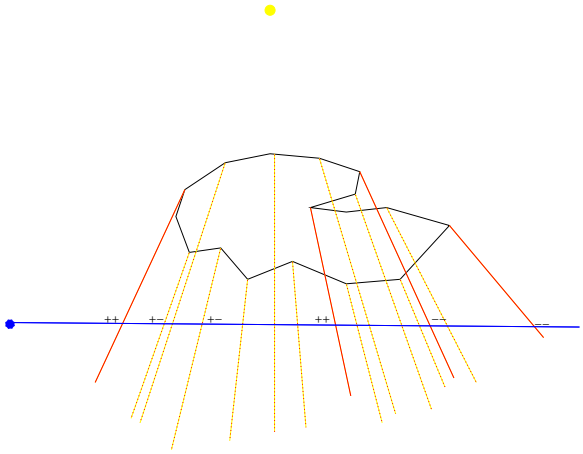

Figure 6: Intersections with shadow polygons in the manifold mesh case.

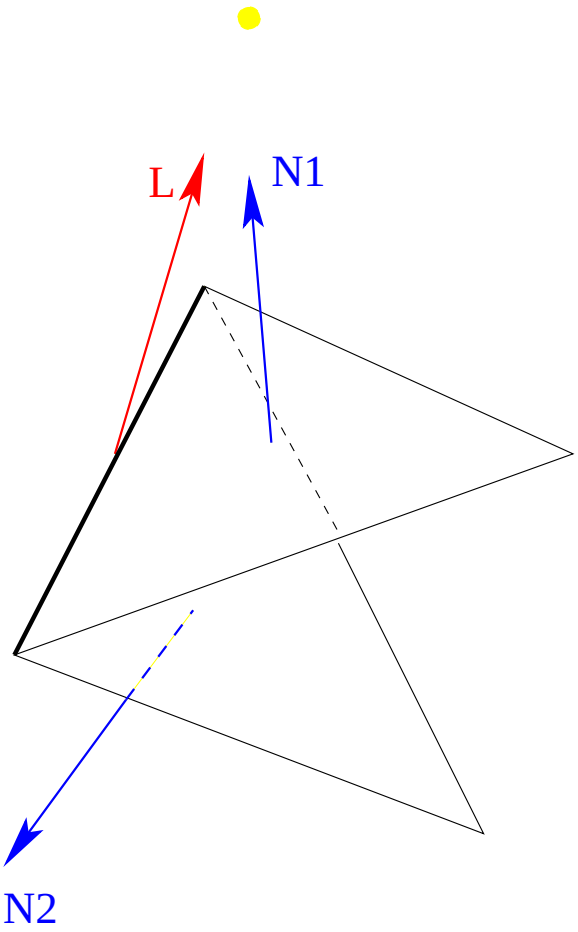

Figure 7: An edge is a silhouette edge if one of its incident triangles faces the light and the other one faces away from the light. Using vector algebra, one can check verify that by checking the signs of the dot products of the light vector L with the normals of the incident triangles, $N1$ and $N2$. If the two dot products have different signs, the edge is a silhouette edge. Otherwise, it is not.