

Project 4 Help

1 Things to remember

Sunday before the finals week is the **final and absolute** deadline. Projects submitted after the deadline will not be graded.

There is no requirement to use display lists (although you might want to use them for the scene, i.e. spheres and triangles). Shadow polygons will be changing as the light moves and using a fixed display list for them is at least tricky. Recompute those polygons on frame-by-frame basis.

It is essential that you use point light sources located at finite distance from the scene. In the skeleton code, the homogenous coordinate of the light source location is set as 0 (in one of the `glLight*` calls). It has to be 1 for a point light source at a finite distance – homogenous coordinate of 0 indicates a point at infinity (for light source location, it means that the light is coming from a point light source at infinity – the light rays are parallel everywhere).

The input file does not contain the viewpoint. This is fine, you should set up your own ‘reasonable’ view (with which a scene where the cube that is supposed to hold all the objects is visible and covers almost all of the window’s surface). **Make sure that the view direction is the negative z axis.** Note that the skeleton code for the previous project displays a view of our cube with viewing direction along the negative z as the first view (however, that cube is scaled by a factor of .8 by the transformation in the display list).

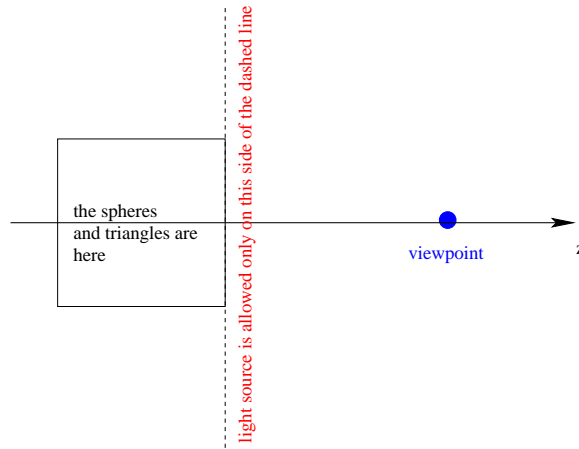


Figure 1: Allowed light source and viewpoint location, relative to the scene (possibly translated and scaled by your transformations).

Remember that the light source is specified in the same coordinate system as the scene. Therefore, all the transformations applied to the spheres and

triangles should be applied to the light source as well. We promise that **the z -coordinate of the light source will be greater than 1** (the original file I posted had this wrong - it's been corrected). This means that the light will be closer along the z -axis to the viewpoint than any object of the scene. In particular the viewpoint will be lit (Figure 1). The light should rotate around the z -axis.

2 Geometry of Shadow Volumes

2.1 Triangles

Shadow polygons of triangles are very easy to compute. It's more difficult to get their orientations right though. But let's put the orientation issue aside for now.

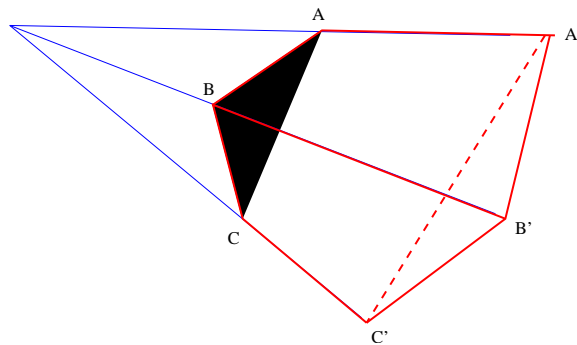


Figure 2: Shadow volume of a triangle ΔABC .

Each triangle will have three shadow polygons (quads). Basically, they will be shadows of the edges of the triangle. Formally, each of them is unbounded. It is possible to specify vertices at infinity in OpenGL (zero homogenous coordinate), but in the project it is equally OK to put them 'far enough', i.e. somewhere far beyond the whole scene. The Figure 2 shows the three shadow polygons for a triangle ΔABC : they are $ABB'A'$, $BCC'B'$ and $CAA'C'$. The primed vertices should be put at least as far from their unprimed counterparts so that their distance is at least as much as the distance between any pair of points on the surfaces forming the scene (basically, we have to make sure that the truncated shadow volumes enclose all the objects that the 'true' unbounded volumes do). Since the whole scene is to fit into the axis-oriented cube of side 2 centered at the origin, that maximum distance is certainly less than the diameter of that cube, i.e. $2\sqrt{3}$. Thus, you can make sure that the primed points are at least this far from their counterparts. Then, you can arrange the three shadow quads into one quad strip. To get the coordinates of the primed vertices, move the unprimed ones away from the light by some large amount (more than $2\sqrt{3}$). Feel free to experiment with vertices at infinity too.

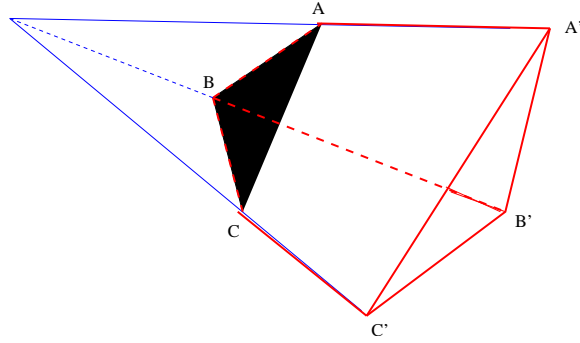


Figure 3: Different orientation of the triangle with respect to the light source.

Orientation is a more complex issue here. We want a triangle to cast the same shadow no matter how its vertices are listed. The problem is that the orientation of the shadow quads relative to the shadow volume will depend on whether the triangle's vertices will be listed clockwise or counterclockwise if looked at from the light source. In Figure 2, the vertices of the quad $ABB'A'$ appear counterclockwise if looked at from outside the shadow volume. On the other hand, if the vertex B is moved to the other side of the interval AC (from the light source's point of view), as in Figure 3, the same quad will appear clockwise when looked at from outside the shadow volume. Notice that the two figures differ in the apparent ordering of vertices of the triangle when looked at from the light source (clockwise in Fig. 2, counterclockwise for Fig. 3). Thus, your code should check in which order the triangle's vertices appear and, according to what it is, switch the orientation of the shadow quads. Of course, this needs to be done on per-triangle basis.

How to decide in which order they appear? For example, take the cross product $\vec{AB} \times \vec{AC}$ and check whether it points to the side of the triangle $\triangle ABC$ where the light is or not. This amounts to testing the sign of the 'mixed' tripple product $(\vec{AB} \times \vec{AC}) \cdot \vec{AL}$, where L is the light source location. You should toggle the orientations of your shadow quads according to the sign of the mixed product.

Use the 'show shadow volumes' option to debug your shadow volume code (e.g. if you use back face culling and the back shadow polygons show instead of the front ones, this means that you have to orient the vertices the other way).

2.2 Spheres

As explained in the shadow volume notes, the shadow volume of a sphere is a part of the solid cone bounded by the surface formed by lines going through the light source and tangent to the sphere. The points that are in shadow are inside that cone and behind the sphere. The shadow polygons should faithfully approximate the part of the cone's surface that is behind the sphere.

How to compute the shadow polygons? Here is a possible way. We start by computing the center and the radius of the circle formed by the tangency points of rays from the light source to the sphere. To do that, we draw a section along a plane through the center of the sphere (call it C) and the light L (Figure 4).

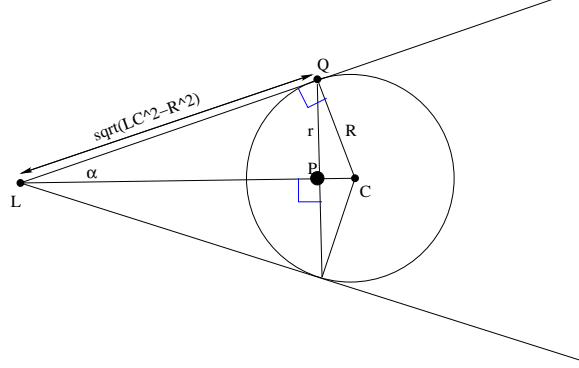


Figure 4: A planar section through the light source and the center of the sphere.

Let's use the notation in the Figure. r is the radius of the circle of tangency points we want to calculate. R is the radius of the sphere. We have:

$$\cos \alpha = \frac{\sqrt{LC^2 - R^2}}{LC} = \frac{PL}{\sqrt{LC^2 - R^2}}.$$

This yields

$$PL = \frac{LC^2 - R^2}{LC}.$$

Having computed PL , we can easily get the radius of the circle of the tangent points:

$$r = \sqrt{LQ^2 - PL^2} = \sqrt{LC^2 - R^2 - PL^2}.$$

Thus far, we have computed lengths of all the intervals. To compute the 3D coordinates of the point P (which is the center of the circle of tangencies), we can simply move from L in the direction of C by the distance equal to the length of LP . This gives the following formula (the fraction is the unit vector in the direction from L to C):

$$P = L + LP * \frac{\vec{LC}}{LC}.$$

Now, we are going to compute two perpendicular vectors \vec{u} and \vec{v} that will start at P and end somewhere at the circle of tangencies (see Figure 5). They both have to be perpendicular to the line through L and P and have length of r . To compute u , one can simply select an arbitrary vector perpendicular to \vec{LC} and scale it to get one of length r . If $\vec{LC} = [x, y, z]$ then examples of perpendicular vectors are $[-y, x, 0]$, $[-z, 0, x]$ and $[0, -z, y]$. Choose one of them.

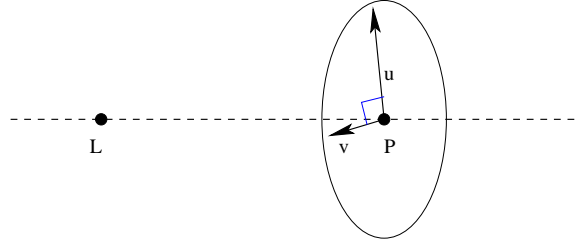


Figure 5: \vec{u} and \vec{v}

For numerical accuracy reasons, it's safest to skip the smallest coordinate so that the vector you choose is long. Then, scale your vector to obtain one (this will be the \vec{u} vector) of length r . To get the other vector \vec{v} , you can take the cross product of \vec{u} and \vec{LC} and scale the result.

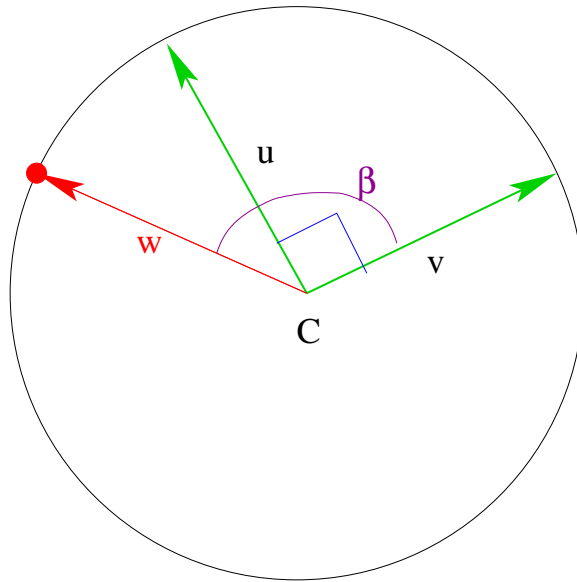


Figure 6: Vectors \vec{u} , \vec{v} and the circle of tangency points.

Now, if you look at what is going on in the plane that the circle of tangencies belongs to, you will see things arranged as in Figure 6. In particular, you can compute the vector w (starting at the center, ending at the circle and forming the angle of β with \vec{v}) using the formula

$$w_\beta = \cos \beta * \vec{v} + \sin \beta * \vec{u}.$$

In particular, you can find vertices of a regular N -gon approximating the circle: just take the points of the above form for angles $k * \frac{2\pi}{N}$ for $k = 0, 1, 2, \dots, N - 1$.

The coordinates of the k -th point can be computed as

$$Q_k = P + w_k * \frac{2\pi}{N}.$$

From now, proceed as in the case of triangles. The shadow polygons are basically shadows of the intervals joining the consecutive Q_k points on the circle. Figure 7 shows these points and intervals for $N = 6$ (in your code, you will probably use something like 32 or 64 to get good accuracy). The shadow quads are shadows of these intervals. Their vertices are the Q_k 's and the points you can obtain by translating them far away ($2\sqrt{3}$ at least) from the light.

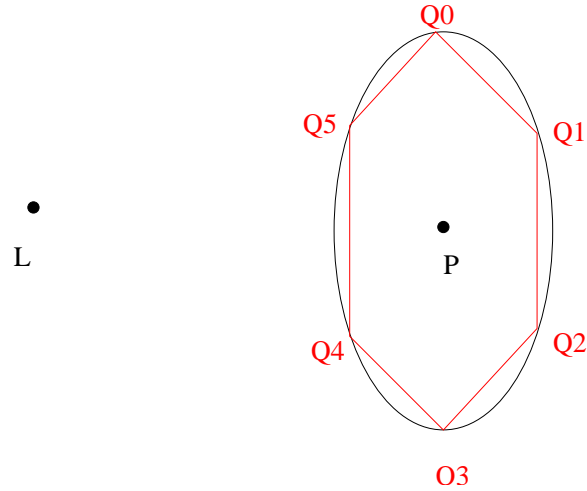


Figure 7: Approximating the circle of tangencies with a regular polygon (here: hexagon).