13.6 Environment Mapping

The goal of *environment mapping* is to simulate an object reflecting its surrounding, e.g., a shiny kettle reflecting the kitchen or a well-polished car reflecting the street. As the environment can be seen by reflection off the object, it is said to be mapped onto the object. An approach to environment mapping originally invented by Blinn and Newell [19] in the seventies is still popular today because of its ease of implementation.

The Blinn-Newell method makes clever use of textures, but the basic idea is not hard. An image of the environment (presumed static) is captured in a texture or multiple textures. Subsequently, the particular texture and texture coordinates used to paint a point on the environment-mapped object are determined from the position of the viewer relative to the object.

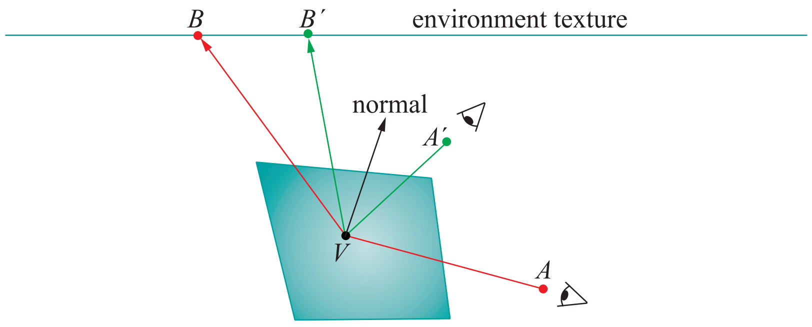

Figure 13.14: Blinn-Newell environment mapping principle: texture coordinates for a vertex V on an environment-mapped surface are obtained from the point on the texture image struck by the reflected ray originating from the eye.

Figure 13.14 illustrates the principle. The texture coordinates at the vertex V of an environment-mapped quad are determined by the point of the environment – more precisely, the corresponding point of the environment texture – seen by the viewer by reflection off the object. For example, when the viewer is at A, V is painted with the color values at B (red in the figure); when she moves to A', those of B' are used (green). The crux of the Blinn-Newell approach then is to *dynamically* compute texture coordinates, based on the laws of reflection, as the viewpoint changes.

OpenGL provides support for two methods of environment mapping: *sphere mapping* and *cube mapping*. Both are based on the Blinn-Newell approach, the difference being in the way that the environment is captured on texture and that texture coordinates are computed. OpenGL provides *automatic texture coordinate generation* for either method. We'll discuss sphere mapping in fair detail.

We'll, however, split our presentation into implementation and theory, as the former is straightforward and what the practitioner needs most to grasp, while the latter is rather more theoretical and demanding.

13.6.1 Sphere Mapping

Getting It to Work

Implementing sphere mapping using OpenGL is simple, as the following program shows.

Experiment 13.12. Run `sphereMapping.cpp`, which shows the scene of a shuttle launch with a reflective rocket cone initially stationary in the sky in front of the rocket. Press the up and down arrow keys to move the cone. As the cone flies down, the reflection on its surface of the launch image changes. Figure 13.15 is a screenshot as it's about to crash to the ground. **End**

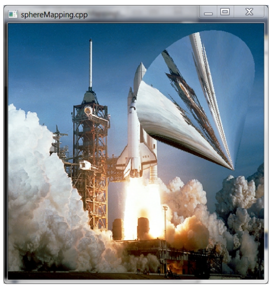

Figure 13.15: Screenshot of `sphereMapping.cpp`.

The two commands

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
```

in the initialization routine of `sphereMapping.cpp` ask OpenGL to use functions from its library to generate the s and t texture coordinates for sphere mapping.

The pair of commands

```
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```

and its inverse

```
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);
```

in the drawing routine, bracketing the drawing of the cone, enable and disable the use of these functions. That's pretty much all there is to implementing a sphere map using OpenGL! Note that at the time sphere mapping is activated the currently bound texture is the launch image, which, of course, is why it is reflected in the cone.

Now, a reader watching the cone as it zooms down may be wondering how authentic actually is the reflection. Good question, and it leads us to investigate how OpenGL computes sphere-mapped texture coordinates.

How It Works

This part is fairly mathematical. If your interest is practical and limited to using the technique, you can safely skip it and jump to the part on preparing the environment texture.

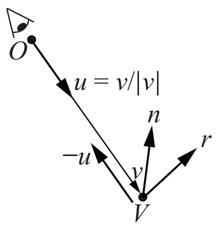

Figure 13.16: The vectors involved in generating texture coordinates.

Here's how sphere-mapped texture coordinates are generated at a vertex V. See Figure 13.16. The unit vector u from the eye (the origin O in OpenGL) toward V is $v/|v|$, where v is the position vector of V, assuming, of course, that $v \neq 0$. The unit eye direction vector from V then is $-u$. The unit normal n at V is user-provided.

OpenGL computes the reflection vector r, the unit vector in the direction that a hypothetical ray from the eye is reflected at V, with the help of the following equation (obtained by replacing light direction vector l with eye direction vector $-u$ in the formula of Exercise 11.4):

$$r = u - 2(n \cdot u)n$$

Suppose, then, OpenGL finds that $r = (r_x, r_y, r_z)$. Computed next is the quantity

$$m = 2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$$

Finally, the texture coordinates at V are calculated as

$$s = \frac{r_x}{m} + \frac{1}{2} \quad \text{and} \quad t = \frac{r_y}{m} + \frac{1}{2}$$

Whew!

If we parse the expressions for s and t carefully, though, it'll not be hard to understand the game plan. Using the expression for m above, write

$$s = \frac{1}{2}\frac{r_x}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2} \quad \text{and} \quad t = \frac{1}{2}\frac{r_y}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2}$$

or

$$s = \frac{1}{2}R_x + \frac{1}{2} \quad \text{and} \quad t = \frac{1}{2}R_y + \frac{1}{2} \tag{13.5}$$

where the variables

$$R_x = \frac{r_x}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} \quad \text{and} \quad R_y = \frac{r_y}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} \tag{13.6}$$

Once we understand what the mapping

$$(r_x, r_y, r_z) \mapsto (R_x, R_y)$$

does geometrically the rest will be straightforward.

The reflection vector $r = (r_x, r_y, r_z)$ is the position vector of some point, say P, on the unit sphere S centered at the origin. See Figure 13.17(a). Now, the position vector of P with respect to the *south pole* $(0, 0, -1)$ of S is $r' = (r_x, r_y, r_z + 1)$. And r' normalized is the vector

$$r'' = \frac{1}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}}(r_x, r_y, r_z{+}1) = \left(R_x, R_y, \frac{r_z + 1}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}}\right) \tag{13.7}$$

In fact, r'' itself is the position vector, with respect to the south pole, of the point Q of intersection of the line from the south pole to P with the

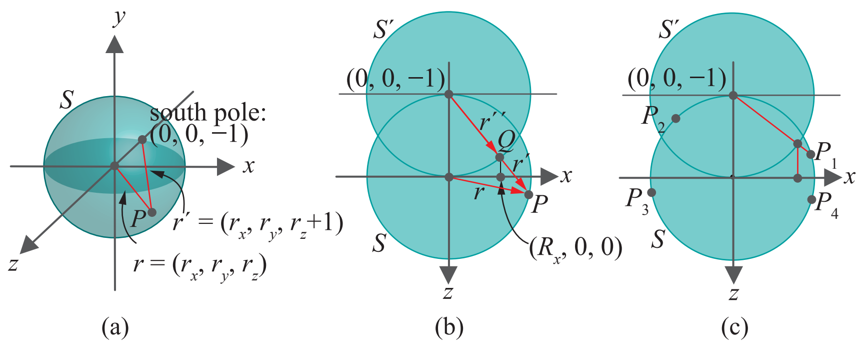

Figure 13.17: Determining R_x from r.

unit sphere S' centered at the pole. S' is not drawn in Figure 13.17(a), but Figure 13.17(b) shows both S and S' in section along the xz-plane (for this particular drawing we assume that P lies on this section). R_x being the x-value of Q by (13.7), the projection of Q on the x-axis in Figure 13.17(b) is $(R_x, 0, 0)$ (Q's y-value R_y is 0, of course, as it's on the xz-plane). Here's an exercise to reinforce your understanding of the preceding construction to find R_x from r.

Exercise 13.16. For each point $P_i, 1 \leq i \leq 4$, in Figure 13.17(c), use a ruler and pencil to draw the corresponding point $(R_x, 0, 0)$ on the x-axis.

Part answer: Red lines indicate the construction for P_1.

The reader may now agree that, at least as P varies over the xz-section of S, $(R_x, 0, 0)$ varies between $(-1, 0, 0)$ and $(1, 0, 0)$ and, correspondingly, R_x between -1 and 1. Moreover, the closer P gets to the south pole the closer is R_x to -1 or 1, depending on which side of the pole P is. However, P should never be *at* the south pole, for, otherwise, the construction to determine R_x breaks down. It follows that R_x itself reaches neither value -1 nor 1. In fact, considering now all of the sphere S, not just its xz-section, it's not hard to see that R_x varies over the open interval $(-1, 1)$ as P varies over S minus its south pole.

The mapping from P to R_y is similar. Therefore, as P moves over S minus its south pole, (R_x, R_y) moves within the interior of the square $[-1, 1]\times[-1, 1]$. For an even better understanding, let's determine analytically the dependence of (R_x, R_y) on P.

Choose a Z in $-1 < Z \leq 1$. The plane $z = Z$ intersects S in a latitudinal circle

$$x^2 + y^2 + Z^2 = 1 \quad \text{or} \quad x^2 + y^2 = 1 - Z^2$$

Now, from (13.6) we have that

$$R_x^2 + R_y^2 = \frac{r_x^2 + r_y^2}{r_x^2 + r_y^2 + (r_z + 1)^2}$$

Therefore, if P lies on the latitudinal circle $x^2 + y^2 = 1 - Z^2$, so that $r_x^2 + r_y^2 = 1 - Z^2$ and $r_z = Z$, then the preceding equation says that (R_x, R_y) lies on the circle

$$R_x^2 + R_y^2 = \frac{1 - Z^2}{1 - Z^2 + (1 + Z)^2} = \frac{1 - Z^2}{2 + 2Z} \tag{13.8}$$

Now, we can see how (R_x, R_y) varies with $r = (r_x, r_y, r_z)$ as we had set out to. In fact, we'll draw a picture. See the two diagrams on the left of Figure 13.18.

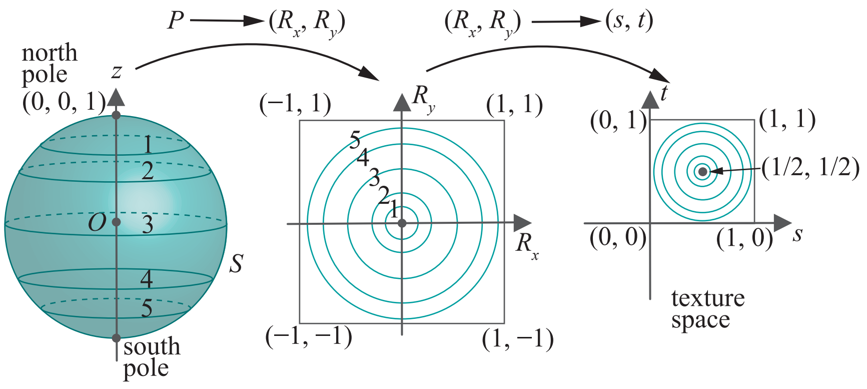

Figure 13.18: The maps $P \mapsto (R_x, R_y)$ and $(R_x, R_y) \mapsto (s, t)$.

Keep in mind that r is P's position vector, the latter varying over S. Each latitudinal circle on S (now drawn upright at left with the north pole at the top to better see these circles) maps to a circle centered at the origin and in the square $[-1, 1] \times [-1, 1]$ in R_xR_y-space (drawn in the middle). In particular, the north pole maps to the origin, and latitudinal circles from the north pole downward map to increasingly larger circles inside $[-1, 1] \times [-1, 1]$. Five pairs of corresponding circles have been drawn and labeled similarly in the two diagrams. As the latitudinal circles approach the south pole, the mapped circles draw nearer and nearer to the containing square.

Exercise 13.17. What is the radius of the circle to which the equator maps? What are the radii of the images of the latitudinal circles 45°N and 60°S?
Hint: Equation (13.8) gives the radius of the circle in R_xR_y-space, which is the image of the latitudinal circle at $z = Z$. For example, the latitudinal circle 45°N has z-value $\sin 45° = 1/\sqrt{2}$, so plug $Z = 1/\sqrt{2}$ into (13.8) to find the radius of its mapped circle in R_xR_y-space.

Exercise 13.18. How do longitudinal great circles on S map?
Hint: Straight lines through the origin in R_xR_y-space ….

The final transformation from R_xR_y-space to st-space (texture space) is simple. See the rightmost two diagrams of Figure 13.18. (R_x, R_y) is mapped to $(\frac{1}{2}R_x + \frac{1}{2}, \frac{1}{2}R_y + \frac{1}{2})$ via Equations (13.5), which linearly transform the square $[-1, 1] \times [-1, 1]$ in R_xR_y-space to the unit square $[0, 1] \times [0, 1]$ in texture space. The images in texture space of the five circles in R_xR_y-space are shown as well in the rightmost diagram.

Bottom Line

Time for a wrap-up in plain English. If vertex V were on a perfect mirror and the environment around it arranged along the unit sphere S centered at V, then the eye would see the point P where S is intersected by the reflection of the line of sight at V (see Figure 13.19). However, OpenGL's only knowledge of the environment is from a user-provided texture occupying a unit square in texture space. So what it does is this: if the eye wants to see the point P in the spherical environment, OpenGL shows it instead the point (s, t) in texture space to which P is mapped as described above by $P \mapsto (R_x, R_y) \mapsto (s, t)$.

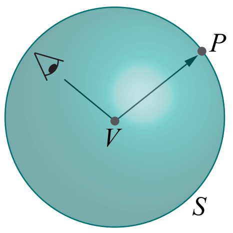

Figure 13.19: The environment S around a perfect mirror vertex V.

The calculations above tell exactly what happens in physical terms. If the eye asks to see the north pole of the environment, then it's shown instead the center of the texture. As the eye travels to see points farther and farther from the north pole, it's shown points farther and farther from the center of the texture. Precisely, latitudinal circles in the environment are replaced for viewing by circles in the texture centered at its middle.

Preparing the Environment Texture

Given this sphere-mapped scheme to present the environment to the viewer via a texture, what is the right way to prepare the texture? Practically speaking, how then should one photograph the environment in order to create the texture image? Comparing the left and right diagrams of Figure 13.18 suggests an answer. The camera should be located at the origin O pointing up the z-axis toward the north pole and have a *very* wide-angle lens; in fact, it would be helpful if the field of view were nearly 360°! Of course, this is impossible, but a fairly wide-angle picture taken with a camera located in the vicinity of the object to be environment mapped, focused up the z-axis of world space, should be good.

Remark 13.4. Since the texel used depends only on the value of the reflection vector at a vertex, and not the vertex's location, reflections in parallel directions appear the same at all vertices. Practically, this means that the environment-mapped object should be small compared to its surroundings for authenticity.

Remark 13.5. Some practitioners advocate the application of filters to the texture prior to sphere mapping. For example, NeHe [102] suggests using the *spherizing* filter (available, e.g., in Adobe's Photoshop software).