Chapter 16

Texture Shading

Texture mapping provides us with the ability to render surfaces by looking up data indexed by its texture coordinates. The initial application of texture mapping added detail to a surface using the texture image to modify its color. In this chapter, we will study other ways to use the texture map to improve rendering, ways in which the texture map was never intended to be used.

16.1 Bump Mapping

Often the surface detail provided by an ordinary texture map includes shaded relief. Classic examples include the surface of a strawberry or an orange, which do not have much color variation other than due to the lighting of their changing orientation at the mesoscale. An ordinary texture map would create a disagreement between the lighting of the Phong-shaded polygons and the likely different lighting of the texture as recorded by the pixel color values of the texture image.

Bump mapping is a technique designed to resolve the illumination of textured relief with the actual illumination of the model. Bump mapping illuminates a textured surface dynamically when it is rendered time whereas an ordinary texture map of surface relief is stored with static illumination that is mapped to the surface regardless of the surface's lighting configuration. Since we use the surface normal to represent surface orientation, bump mapping uses the texture image to adjust each fragment's surface normal to the orientation of the relief mapped onto the surface. Hence bump mapping does not change the actual position of any point on the surface, but creates the illusion that the surface is raised and lowered by the textured relief.

The texture image used for bump mapping is an image array of the altitudes of the textured relief. Let $B(s,t)$ represent this relief image of altitudes. If the relief were completely flat, then $B(s,t)$ could be set to any single altitude. The relief is represented by changes in these altitudes.

Hence we will use the slopes in the two coordinate directions, computed as the altitude changes over a unit surface distance,

$$B_s(s,t) \approx B(s+1,t) - B(s,t), \quad (16.1)$$
$$B_t(s,t) \approx B(s,t+1) - B(s,t), \quad (16.2)$$

to adjust the surface normal accordingly,

$$\mathbf{n} += B_s(s,t)\mathbf{n} \times \mathbf{p}_t - B_t(s,t)\mathbf{n} \times \mathbf{p}_s. \quad (16.3)$$

3D BUMP MAPPING FIGURE.

The vectors $\mathbf{p}_s, \mathbf{p}_t$ are the derivatives of the fragment's 3-D position (in the same coordinate system that also hold the normal) with respect to the texture coordinates s, t. These can sometimes be provided directly by a parametric model (such as the globe example), but for a general triangle mesh we will have to estimate them from the texture coordinates stored at the vertices.

CITE CG TUTORIAL CHAPTER 8 — Error in dp/ds = B (should be C)

To find $\mathbf{p}_s$ and $\mathbf{p}_t$ we first define

$$\mathbf{p}(s,t) = (x(s,t), y(s,t), z(s,t), \quad (16.4)$$

the 3-D position as a function of texture coordinates s and t, and then differentiate wrt each of these two parameters. Each vertex of a triangle $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v_2}$ contains a 3-D position $\mathbf{p}_i = x_i, y_i, z_i$ as well as (at least) two texture coordintes s_i, t_t for $i = 0 \ldots 2$. We then set plane equations

$$A_x x + B_x s + C_x t + D_x = 0, \quad (16.5)$$
$$A_y y + B_y s + C_y t + D_y = 0, \quad (16.6)$$
$$A_z z + B_z s + C_z t + D_z = 0. \quad (16.7)$$

So A_x, B_x, C_x, D_x describe a plane in the (x, s, t) coordinate system, A_y, B_y, C_y, D_y in the (y, s, t) coordinate system, and A_z, B_z, C_z, D_z in the (z, s, t) coordinate system. For each of these we have

$$(A, B, C) = (\mathbf{v}_0 - \mathbf{v}_1) \times (\mathbf{v}_0 - \mathbf{v}_2), \quad (16.8)$$
$$D = -(A, B, C) \cdot \mathbf{v}_0 \quad (16.9)$$

in their appropriate coordinate systems $(x, s, t), (y, s, t)$ and (z, s, t). Since the cross products producing A_x, A_y and A_z all use only the same s and t coordinates of the three vertices, we can refer to them identically as A. We also notate $\mathbf{B} = (B_x, B_y, B_z)$ and similarly for $\mathbf{C}$ and $\mathbf{D}$. Solving the system (16.5) - (16.7) for x, y and z yields the definition

$$\mathbf{p}(s,t) = \frac{-1}{A}(s\mathbf{B} + t\mathbf{C} + \mathbf{D}), \quad (16.10)$$

and partial differentiation yields

$$\mathbf{p}_s = \frac{-1}{A}\mathbf{C}, \qquad \mathbf{p}_t = \frac{-1}{A}\mathbf{B}. \tag{16.11}$$

Note that we do not need to compute any of the D terms. Furthermore since we only need the directions of $\mathbf{p}_s$ and $\mathbf{p}_t$, then can be replaced by $-\mathbf{C}$ and $-\mathbf{B}$, yielding

$$\mathbf{p}_s \propto (s_1 - s_2)\mathbf{p}_0 - (s_0 - s_2)\mathbf{p}_1 + (s_0 - s_1)\mathbf{p}_2, \tag{16.12}$$

$$\mathbf{p}_t \propto (t_1 - t_2)\mathbf{p}_0 - (t_0 - t_2)\mathbf{p}_1 + (t_0 - t_1)\mathbf{p}_2 \tag{16.13}$$

16.2 Normal Mapping

16.3 Environment Mapping

Simulating the reflection of an object off a shiny surface is challenging for a real-time rasterization-based system. The reflected object must be rendered following the same steps as the shiny object, and if the reflected object is itself shiny and reflecting yet another object, this would add yet further rendering to a SIMD-based rasterization system designed to run approximately the same amount of computation per pixel.

When looking in a mirror on the wall, one sees a virtual image reflecting the objects in the room including themselves. We can simulate a planar reflection simply by reproducing the reflected geometry in the room on the other side of the mirror. One could even simulate the minor distortions of a curved mirror by transforming the reflected geometry, but this would get quite complicated for reflections off an arbitrary object.

Sphere Mapping

The environment map is a method for capturing what could be reflected in an environment off of an object. One common representation of an environment map is the which is the image of a gazing ball reflecting its environment. Gazing balls are often found in gardens, so gardeners can see if anyone (or anything) is approaching from any direction. The gazing ball is just a reflective sphere, but is capable of reflecting light from almost all directions toward a single viewer.

Consider the image of an orthographic projection of a gazing ball reflecting its environment. Then the center of the ball's image reflects objects in the direction of the viewer. The annular region of the image about halfway between the center and edge of the ball's projection reflects objects to the left, right, above and below the gazing ball. Beyond this annular region between it and the circular silhouette of the ball's projection is a larger annular region that reflects objects *behind* the gazing ball. All of these

reflections are distorted, and the ones from objects behind the ball can be stretched quite thin. Nevertheless, all of the environment, other than what gazing ball occludes from the viewer, is reflected toward the viewer.

The environment uses the image of a gazing ball as a texture map, and then assigns texture coordinates to a shiny object based on what direction should be reflected at that point. The gazing ball is a sphere, and a useful property of a sphere is that any possible surface normal orientation can be found on its surface. This is also true for any closed smooth surface but easiest to find on the sphere. Given a surface normal $\mathbf{n}$, the point on a unit sphere $\mathbf{x}$ with that surface normal has the same coordinates as the surface normal, $\mathbf{x} = \mathbf{n}$.

We can simulate what is reflected at a point on an arbitrary reflecting surface with surface normal $\mathbf{n}$ by finding out what is reflected by gazing ball at the point with the same surface normal. We store as a texture image the image of the gazing ball reflecting its environment, projected as a disk extending across the texture image. Given a fragment with surface normal $\mathbf{n}$ in viewing coordinates, we find its position on the gazing ball's projection in the texture image by setting texture coordinates

$$s = \tfrac{1}{2}n_x + \tfrac{1}{2}, \qquad t = \tfrac{1}{2}n_y + \tfrac{1}{2}. \tag{16.14}$$

What is reflected by the gazing ball with that normal is what would be reflected by the surface with that normal. The spheremap image only shows the front hemisphere of the gazing ball sphere, but this represents all of the front facing surface normals which is only what we could possibly see on a closed object.

Such environment mapping is an approximation. The environment reflected is based on what is loaded into the texture image, and may not contain actual objects in the scene. Furthermore, the environment map is representing the reflections of points on a sphere of some radius, and these reflections can be slightly different than what would be reflected by that surface normal at a different location on the target reflective object's surface.

Often the reflective object is removed from the scene and the scene is rendered into a spheremap. Hence, the reflective object itself will not appear in the environment map and so will not reflect itself. Furthermore, the scene will be reflected as it was when the environment map was created, and not how it might be currently configured.

Several methods are available to create a sphere map. One can remove the reflective object and render the scene viewed from its center. A single rendering toward the original viewer with a nearly 180° field of view could generate the needed information, but often it is better to create separate renderings on the sides of a cube (as is done for the cube map in Section **??**) and project these onto a reflective sphere.

Environment Mapped Reflections Actual Reflections

Figure 16.1: The reflections approximated by environment mapping do not include self-reflections of the reflective object.

The texture image pixel with texture coordinates $(s,t) \in [0,1]^2$ corresponds to the point on the reflecting sphere with coordinates

$$x = 2s - 1, \qquad y = 2t - 1 \qquad z = \sqrt{1 - x^2 - y^2}, \tag{16.15}$$

as well as its normal. This point and its self-same normal reflect the view vector $(0,0,1)$ into the unit direction

$$\begin{aligned} \mathbf{r} &= 2((x,y,z)\cdot(0,0,1))(x,y,z) - (0,0,1), && (16.16)\\ &= 2z(x,y,z) - (0,0,1), && (16.17)\\ &= (2xz, 2yz, 2z^2 - 1). && (16.18) \end{aligned}$$

Real-world sphere map environments can be captured by photographing a gazing ball, often referred to in this context as a . It is best to use a zoom lens to photograph the gazing ball from a distance to more closely approximate an orthographic projection, while filling as much of the frame as possible to best sample the reflected environment. Photographing from a distance also minimizes the gazing ball reflection of the photographer.

Cube Mapping

labelsec:cubemap The sphere map has several limitations as an environment map. One is that the spheremap provides reflections from only one point of view. Another is the uneven sampling that distorts and undersamples objects reflected near its silhouette, which interferes with the idea of moving a spheremap's samples to change its viewpoint.

The cube map records the environment surrounding a point by projecting its view onto the six square sides of the cube.

We use a reflection vector $\mathbf{r}$, computed as usual by reflecting the view vector about the surface normal, to index into the cube map. We use the maximum magnitude component of $\mathbf{r}$ to decide which of the six texture

images, $+x, -x, +y, -y, +z$ or $-z$, to use. We then divide the other two coordinates by the maximum magnitude coordinate to find the position in the corresponding texture map as

$$(s,t) = \left(\frac{1}{2},\frac{1}{2}\right) + \frac{1}{2}\begin{cases} \left(\dfrac{r_z}{|r_x|},\dfrac{r_y}{|r_x|}\right) & \text{if } |r_x| > \max(|r_y|,|r_z|), \\ \left(\dfrac{r_x}{|r_y|},\dfrac{r_z}{|r_y|}\right) & \text{if } |r_y| \geq \max(|r_x|,|r_z|), \\ \left(\dfrac{r_x}{|r_z|},\dfrac{r_y}{|r_z|}\right) & \text{otherwise.} \end{cases} \tag{16.19}$$

16.4 Shading with the Environment Map

We can use the environment map to fix a problem of interpolating highlights. Recall that Gouraud's interpolation of per-vertex lighting colors works well for diffuse shading but can miss specular highlight peaks that occur between vertices. Per-fragment lighting using Phong's interpolation of surface normals fixes this problem, but at the expense of the per-fragment inverse square root needed to unitize the linearly interpolated normal. The environment map can be used instead to reproduce the highlight gleam even if it appears on a face between vertices.

We can shade an object using only environment mapping, for example with a sphere map. We first set up a sphere at the approximate position and size of the target object, illuminated by the light sources. (This approximation works best for distant isotropic or directional light sources, and with no attenuation in the shader). We render the sphere using the object's (shading function, resulting in the image of a sphere made of the object's material illuminated by the object's lighting.

We then render the object with lighting and shading disabled except for environment mapping in decal mode (each fragment displayed the its corresponding environment map color). We use the sphere map rendered as described above, such that points on the surface with a given normal will appear as on the point on the sphere with that normal. Instead of interpolating color values, which can obscure intervening specular highlights, gleams are properly reproduced as they are indexed from the environment map by the interpolation of texture coordinates generated from surface normals.

This approach has several additional benefits. The environment map can be blurred to antialias the shading, such as the glistening that can occur from small highlights. Furthermore, since the shading is precomputed and stored in the environment map, more sophisticated shading can be applied than might be ordinarily allowed for real-time rendering.

16.5 Shading with the Texture Map

We can also use ordinary texture mapping to illuminate an object. Consider this simplified Phong shading model consisting of ambient, diffuse and specular terms

$$\mathbf{c} = k_a \mathbf{c}_a + k_d \mathbf{c}_d \mathbf{n} \cdot \mathbf{l} + k_s \mathbf{c}_s (\mathbf{v} \cdot \mathbf{r})^n. \tag{16.20}$$

We can reproduce this shading using the texture map. We first precompute a texture using the function

$$\text{texture}(s,t) = k_a \mathbf{c}_a + k_d \mathbf{c}_d s k_s \mathbf{c}_s t^n. \tag{16.21}$$

We then render the object with lighting and shading disabled and texturing enabled (in decal mode), assigning to each vertex the texture coordinates

$$s = \mathbf{n} \cdot \mathbf{l}, \qquad t = \mathbf{v} \cdot \mathbf{r}. \tag{16.22}$$

Texture coordinates are interpolated and sample shading between vertices. This technique can miss specular highlights that occur between vertices, unlike shading with the environment map. However, environment map shading was limited to a single vector input (usually the surface normal) whereas this approach can provide a wider variety of inputs (usually the results of dot products).

SKIN

HAIR