Exploiting Sphere Mapping

By Paul Zimmons and Rui Bastos

12/20/99

Abstract

Sphere mapping is a useful tool for creating realistic computer-generated images. It is currently the only view-dependent texture mapping technique widely supported in hardware. Understanding how sphere mapping relates to other environment mapping techniques such as parabolic mapping and cubic mapping can help promote its use in the appropriate context. After an initial survey of texture mapping and a discussion of the benefits and limitations of various mapping models, sphere mapping and its applications are discussed in depth. The discussion of sphere mapping includes: the simulation of scenes as viewed from a window, lighting models, colorgrams, and reflectance functions.

1. Introduction

Texture and environment mapping are used extensively to help add detail to computer generated images at little expense. Texture mapping refers to the application of an image-based quantity to the surface of an object in order to make the object appear as if the image were painted, projected, or wrapped onto its surface. Environment mapping is similar to texture mapping in that an image is applied to an object. However, the image represents the environment surrounding the object. By using special indexing into the environment image, the surface of the object appears to mirror its outward environment. Environment mapping has the potential to create a significant range of view-dependent effects.

Sphere mapping is a specific type of environment mapping. It is an underutilized, hardware-supported method of image mapping that allows view-dependent effects to be incorporated into computer generated imagery at interactive rates. Although initially designed as an environment mapping technique, sphere mapping actually has broader applications which can improve image quality when view dependency is a factor. For example, sphere mapping can be used in capturing surface-dependent lighting and reflectance characteristics.

After a general overview of texture mapping and various mapping techniques, this paper will focus on sphere mapping and its application to specific phenomena such the simulation of window panes, lighting models, colorgram stickers (surfaces that change color based on viewing angle), and reflectance functions.

1.1 Motivation

The rise of hardware complexity has made new indexing modes possible. As a result, a wider range of visual effects can be simulated. Sphere mapping, which has a hardware basis, can be an attractive option because its view-dependent effects can be produced quickly and inexpensively. Sphere mapping promises to be a valuable tool for increasing realism in interactive computer graphics.

2. Texture Mapping

Sphere mapping is a subset of texture mapping. To understand the nuances of sphere mapping, a general understanding of texture mapping is required. Therefore, a short review of texture mapping is presented to provide context.

Texture mapping refers to the application of an image to a surface in order to make an object appear as if the image were painted, projected, or wrapped onto the surface. Ever since Ed Catmull introduced texture mapping in his 1974 Ph.D. thesis, texture mapping has been recognized as an important method for adding realism to a computer generated image. [7] The simplest form of texture mapping requires a parametric definition of a surface as well as a texture to be applied to that surface. The parameters for the surface are evaluated to give position information (z value). Those same parameters can then be used to index into an image and retrieve color for the portion of the surface being evaluated. For example, if a surface such as $z(x,y) = 2x+3y$ is being evaluated at a point (x, y) to obtain the surface's z value, then the image can be sampled at (x, y) as well to obtain the color with which to draw that portion of the surface. If the index is outside the image, modular indexing or repeating border pixels may be used.

However, most models are not parametric and are composed of triangles. In that case, each triangle vertex has an associated texture coordinate which corresponds to a particular place on the 2D image to be applied to the surface. Usually, the image coordinates are represented as s and t instead of x and y. The texture coordinates are then linearly interpolated across the corners of the triangles. The process is similar to a triangular cookie cutter stamping out areas of the image and pasting it onto the triangle being processed as seen in Figure 1. By sharing texture coordinates with corresponding vertices in adjacent triangles, an image can appear to be continuous across several polygons.

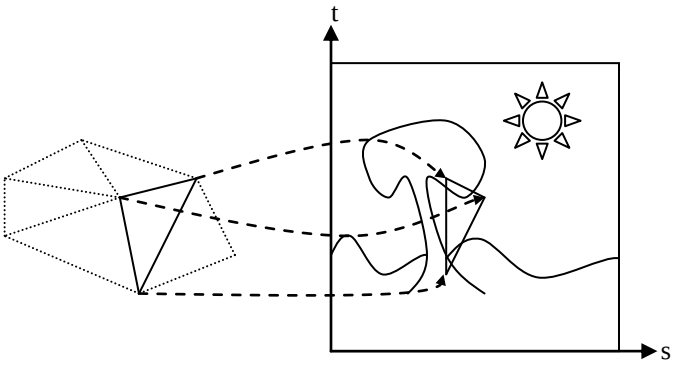
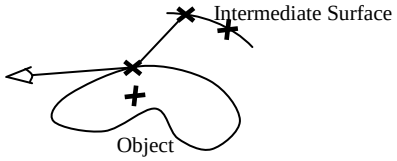
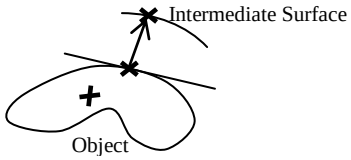


Figure 1: A Triangular Surface and the Corresponding Texture Indexing.

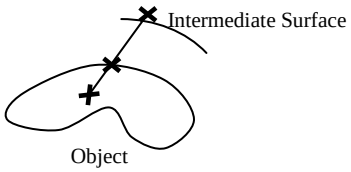
Because the triangles can be of different sizes and orientations, the texture can be predistorted in order to look appropriate on the surface after the texture is mapped. This is similar to film shot under an anamorphic lense. The resulting picture looks appropriate when projected onto the screen, but it is horizontally compressed on the film itself. Instead of performing this predistortion with a specific object in mind, in 1986 Bier and Sloan introduced an approach which mapped the image onto a simple, intermediate surface such as a cylinder or a sphere. [3] Bier and Sloan referred to this step as *S mapping*. This intermediate surface was then mapped onto the object to be texture mapped in a process called *O mapping*. Since the intermediate surface was simple, the generation of texture coordinates for vertices was easier to manage. There are several types of O mappings which determine exactly which texture coordinate to choose for each vertex in the object. All of the following vectors can be used to generate texture coordinates: the vertex's normal, the ray from the object's centroid through the vertex, the intermediate surface's normal, or a reflected ray off the surface from a viewer. Figure 2 shows these four type of mappings from a top-down perspective with a cylinder being used as an intermediate surface.



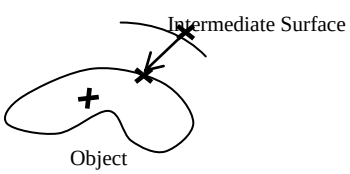
a) Indexing using the Reflected Ray



b) Indexing using the Object Normal



c) Indexing using the Object Centroid



d) Indexing using the Intermediate Surface Normal

Figure 2: Methods of Mapping a Texture onto a Surface. (This figure is derived from Bier and Sloan's original texture mapping paper. [3])

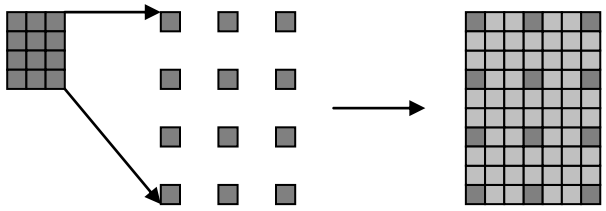
2.1 Filtering

Because the image can be stretched and shrunk as it is applied to the surface, texture filtering is an important part of texture mapping (as well as environment mapping). Filtering refers to the process of interpolating areas of the texture map when a small number of texture map pixels take up a large part of the screen (magnification) or when a large texture maps to a few pixels on the screen (minification).

For example, if the original texture map is 512x512 pixels and it is mapped onto a polygon that takes up a full 1900x1600 pixel screen, the texture is magnified. There is not a one to one ratio between texture map pixels and screen pixels. The texels could be spaced out to fill the screen, but then there would be gaps between the pixels where there is no information in the texture map. To fill in values between these pixels, the known color values must be interpolated to provide information to the polygon on the screen. This interpolation is depicted in Figure 3. The method used to fill in the values between pixels is called filtering. Common filtering methods include linear, quadratic, and cubic interpolations. Linear filtering means that the color between two pixels in the texture map will be some weighted average depending upon where the texture map is being evaluated.

Minification is very similar. However, instead of trying to interpolate colors between textured pixels, too many texture map pixels correspond to the same pixel on the screen. Therefore, some averaging must be done on the texture map pixels to come up with a representative pixel for that area.

A texture can be magnified and then minified to its original size, and it would look exactly like the original image. Unfortunately, the converse is not true. A minified texture destroys information which cannot be recovered when the texture is then magnified.



a b c

Figure 3: Texture Magnification and Interpolation.
a) Original Texture, b) Magnified without Interpolation, c) Magnification with Interpolation

2.2 Environment Mapping

Environment mapping is a subset of texture mapping and is similar in that an image is also applied to a surface. However, environment mapping uses a significantly different indexing scheme than texture mapping. The image used for environment mapping is taken near the center of the object, and the contents of the image are of the environment surrounding the object. By using special indexing into this image, the surface being mapped appears to mirror its outward environment. The indexing method used in environment mapping involves taking the viewer's eye, calculating a vector reflected off the surface from the viewer's eye, and then using that reflected vector as a method of indexing into the image. [4] Figure 2a illustrates the mapping process used in environment mapping.

Sampling is especially important in environment mapping because the creation of the map involves nonuniform sampling of the environment. Storing the world as a 2D texture implies mapping the colors of a set of 3D points in the environment surrounding an object to a two dimensional image. Any predistortion or oversampling will result in uneven coverage of some areas of the environment since the image is stored as a regular, two-dimensional array. In addition, a two-dimensional map of the environment will involve projecting the world onto some intermediate surface which is then mapped into two dimensions. Each step involves distortion of the original environment.

Because the map being applied to an object is distorted and the mapping process itself involves distortion, filtering environment maps is more difficult than normal texture maps. In order to overcome the nonlinear sampling used to construct the environment maps, the maps require nonlinear filtering to reconstruct the environment information onto the mapped object. However, current graphics hardware does not support nonlinear filtering of arbitrary areas; therefore, environment mapped surfaces contain artifacts and/or excess distortion. Linear vs. nonlinear filtering is discussed in more detail in section 2.4.1. Most of the efforts of researchers exploring environment mapping have been directed toward finding ways to reduce the distortion involved while still reaping the benefits of two-dimensional image maps and linear hardware filtering.

2.3 Limitations of Environment Mapping

Although sampling and distortion are very important issues in environment mapping, the following are some significant limitations:

- Only perfectly specular (mirror like) attributes can be modeled because the environment map only uses the reflected (mirror) direction for indexing;
- The object should be convex because the environment map does not capture object self-interreflection;
- Interreflections between two environment mapped objects (or the object and itself) require multiple passes through the graphics pipeline;
- Mapping across a polygon assumes that the interpolation of texture coordinates between the vertices is linear which is usually not the case in environment mapping.

The fundamental limitation of all environment mapping techniques is that they treat the environment as being infinitely far away from the viewer. Since these mappings use 2D maps of the world, they do not contain depth information. Therefore, their approximation to the environment quickly degrades as the object moves away from the center of projection. If depth is associated with each pixel in the environment map, properties such as occlusion, depth of field, and parallax of the reflected world can be properly approximated. Attempts at solving these depth related problems for the case of planar reflections has been investigated by Bastos and Stuerzlinger. [1]

Despite the difficulties inherent in environment mapping, it remains an effective tool for approximating reflective objects. The next section will discuss the various methods for implementing environment mapping in terms of storage, indexing, filtering, and limitations.

3 Ideal, Cubic, Dual Paraboloid, and Sphere Environment Mappings

This section discusses four types of mapping schemes for environment mapping and the advantages and disadvantages of each one.

3.1 Ideal Mapping

All other environment mappings discussed in this section are an approximation to an ideal mapping. An ideal, 2D, environment mapping would sample the same solid angle of environmental information in every viewing direction. The only surface that would enable the storage of this kind of mapping without distortion would be a sphere. A sphere is an ideal basis for environment map sampling since a solid angle subtends an equal amount of the environment in every direction. This kind of environment map could be stored as a list of coordinates on the sphere along with an associated color, (x, y, z, r, g, b) or (θ, ϕ, r, g, b). The sampling points can be determined by several methods: recursive subdivision of the surface of the sphere, hexagonal tiling of the sphere, or the distribution of random points on the sphere through repulsion simulation. The underlying concept is that every point has the same distance to each of its neighboring points on the sphere so that there is a uniform sampling of the sphere's surface. The color of each point corresponds to the color of a surface hit by a ray shooting out from the center of the sphere through the (x, y, z) coordinate and intersecting the environment. Unfortunately, a limitation of this type of spherical map is that it is not amenable to hardware. Instead, a spatial lookup in software is needed to get information from nearby points for filtering. While an ideal mapping uses a sphere, it is not the same as the sphere mapping that will be discussed in Section 3.4.

3.2 Cubic Mapping

To approximate the ideal map, Greene projected the environment onto the sides of a cube. [10] The cubic mapping approach stores the environment as six, 2D images of the faces of a cube. Cubic mapping has the advantage of coming as close to a sphere as possible while still utilizing the convenience of linear interpolation and planar rectangular image maps. Indexing is relatively straightforward as the reflected vector for a pixel can be mapped to a single face of the cube. However, references to the environment maps across cube edges can potentially reference distant portions of memory and lead to memory thrashing. Filtering can also become complicated due to the discontinuities in the storage of image maps. For example, a texture filter across edges of the cube can reference pixels that may also be distant in memory. Cubic mapping remains an important software technique and is useful in deriving maps for other environment map images such as the ones discussed below. Recently, hardware has become available to perform this type of mapping on a per fragment basis.

3.3 Dual Paraboloid Mapping

Dual paraboloid mapping was proposed and implemented by Heidrich and Siedel in 1998 as a low distortion environment mapping technique. [11] The environment is stored as two orthographic images of the world as reflected from an ellipsoid. Each image covers a 180 degree field of view. Both images together provide a complete mapping of the world. The sampling distortion per solid angle is at least 20% lower than cubic mapping. The images are indexed using a series of matrix transformations beginning with a reflected vector on the surface of the object in eye-space. This reflected vector is then transformed into the corresponding normal on the paraboloid; and finally, the point referenced by the normal is converted into texture coordinates. The map is chosen depending on the viewer's eye in relation to the coordinate system of the maps, and an alpha channel is used to determine which image contributes to the surface color for a pixel. A limitation of dual paraboloid mapping is that it is discontinuous at the edges of the map and requires extra filtering to compensate. Filtering across multiple pixels may also be complicated due to the separation of the world into two distinct maps.

3.4 Sphere Mapping

Sphere mapping is a method of storing the entire environment within a single image map. As with dual paraboloid mapping and cubic mapping, the image is an orthographic picture of the environment. Intuitively, the map is indexed based on θ and ϕ from the eye relative to the object's surface as shown in Figure 4.

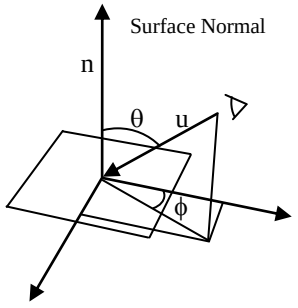


Figure 4: The Eye Drives the Indexing in Sphere Mapping.

3.4.1 Sphere Map Indexing

The indexing scheme for sphere mapping is somewhat complex, so it will be reviewed in detail. The following discussion is based on material from the work of Hoff and Blythe. [12, 6] For sphere

mapping to work, there must be an observer, an object (composed of vertices), and normals at the vertices. Since sphere mapping is an eye-space, mapping technique, the vectors should be calculated relative to the eye. The first step in sphere mapping is to convert a vertex normal (in object coordinates) into eye coordinates. This transformed normal vector is called **n**'. The viewing vector from the eye is represented as a vector **u** which is directed out from the eye towards each vertex being processed. Since environment mapping relies on the reflected vector from the eye, it must be computed (in eye space) from the vectors **u** and **n'**. The standard equation for computing a reflected vector around a normal can be understood through Figure 5.

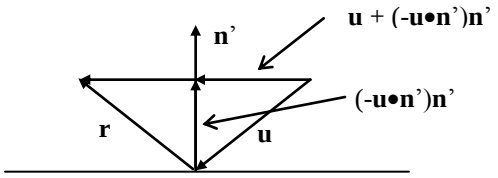


Figure 5: Calculating the Reflection Vector.

From the diagram we can see that by reversing **u** and using the **u**+(-**u**•**n**')**n**' vector twice, the reflected vector is obtained. In other words,

$$\mathbf{r} = -\mathbf{u} + 2(\mathbf{u}+(-\mathbf{u}\bullet\mathbf{n}')\mathbf{n}') = \mathbf{u} - 2(\mathbf{u}\bullet\mathbf{n}')\mathbf{n}'. \quad \text{(Equation 1)}$$

Once the reflected vector is computed, it is mapped to texture coordinates on the sphere map image. The coordinates for the image are as in Figure 6.

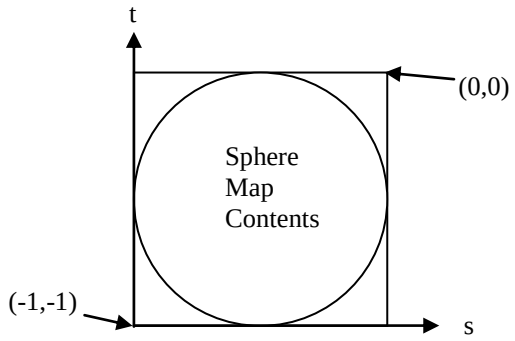


Figure 6: The Coordinate System for the Sphere Map.

Meanwhile, the coordinates of the sphere which is being mapped are given in Figure 7.

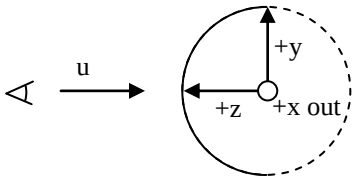


Figure 7: The 3D Sphere Used in Sphere Mapping.

The relation between the sphere and the sphere map involves mapping all the directions emanating from the center of the sphere onto a disk map. The hemisphere facing the viewer is mapped to the center of the disk, whereas the hemisphere facing away from the viewer is mapped to the rest of the map. In this way, the entire environment can be displayed in a single image map. Figure 8 makes this relationship apparent:

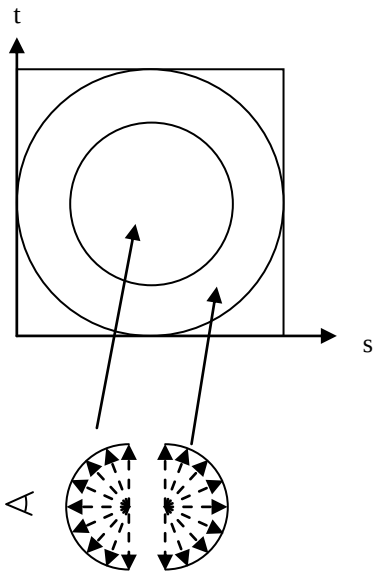


Figure 8: Reflected Vectors for the Whole Sphere Compose the Sphere Map.

From previous figures (Figures 7 and 4), the **u** direction from the eye to the vertex and the +**z** direction of the sphere are aligned before indexing is performed. Because this orientation is constant relative to the viewer, sphere maps are view dependent. The vertex normal, transformed into eye space, is used in conjunction with the **u** vector to compute the reflected vector in eye space according to Equation 1. The vectors **u**, **n**, and **r** are all coplanar. The coordinates of the reflected vector are then used to generate (s,t) coordinates. When the reflected vector, normal, and eye vector are all coincident, the location (0.5, 0.5) in the sphere map image is indexed. When the normal is perpendicular to **u**, **r** is antiparallel to **u** and references a pixel on the outermost ring of the sphere map. The orientation of the vectors relative to each other is given in Figure 9 below.

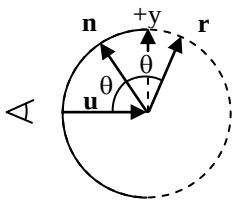


Figure 9: The Reflected, Normal, and Eye Vectors Relative to the Sphere.

Relative to the coordinate system defined for the sphere in sphere mapping, **u** is (0, 0, -1). **N**, the normal to the vector being processed, has coordinates of (n_x, n_y, n_z). Since z is on the sphere, the normal's coordinates can also be given implicitely as (n_x, n_y, $\sqrt{1-n_x^2-n_y^2}$). The reflected vector is then

$$r = \begin{pmatrix} r_x \\ r_y \\ r_z \end{pmatrix} = u - 2(u \bullet n)n = \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} - 2\left(\begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix} \bullet \begin{pmatrix} n_x \\ n_y \\ \sqrt{1-n_x^2-n_y^2} \end{pmatrix} \right) \begin{pmatrix} n_x \\ n_y \\ \sqrt{1-n_x^2-n_y^2} \end{pmatrix}$$

$$= 2\sqrt{1 - {n_x}^2 - {n_y}^2} \begin{pmatrix} n_x \\ n_y \\ n_z - 1 \end{pmatrix}.$$

The above formula for the reflected vector has added a scaling factor to the size of the normal. This will be normalized out later in the calculation. Using this relation, **n** can be phrased in terms of **r**. Next, **n** can be phrased in terms of n_x and n_y which are directly related to s and t. Finally, the conversion from **r** to (s, t) values will be complete. Continuing on from the previous step,

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} = \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix} \Big/ 2\sqrt{1 - {n_x}^2 - {n_y}^2}\ .$$

So **n** will always be in the direction (r_x, r_y, r_z+1), but its length will change with n_x and n_y. Since the $2\sqrt{1 - {n_x}^2 - {n_y}^2}$ term is just scaling the vector, it can be ignored; and **n** can be renormalized. After renormalizing, the unit normal in terms of **r** is

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \Big/ \left\| \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \right\| = \begin{pmatrix} r_x \\ r_y \\ r_z + 1 \end{pmatrix} \Big/ \sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}\ .$$

Since the n_x and n_y components of the normal vector are the only ones that correspond to the sphere map, only n_x and n_y coordinates will be considered from here on. In terms of x and y, the normal is

$$\begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} 2s - 1 \\ 2t - 1 \end{pmatrix} = \begin{pmatrix} \dfrac{r_x}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} \\ \dfrac{r_y}{\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} \end{pmatrix}.$$

In the above formula, (n_x, n_y) = (2s-1, 2t-1) because the sphere ranges from (-1, -1) to (1, 1) in x and y, while the sphere map ranges from (0, 0) to (1, 1) in its own coordinate system of s and t. Finally, the (s, t) texture coordinates in the sphere map can be expressed as

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \dfrac{r_x}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \dfrac{1}{2} \\ \dfrac{r_y}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \dfrac{1}{2} \end{pmatrix}. \text{ (Equation 2)}$$

With exception to the outermost ring of pixels, adjacent pixels in the environment always map to adjacent pixels in memory. Unfortunately, sphere mapping suffers from a singularity at the outermost circumference of the map where all pixels on that circumference represent the same reflected vector. In addition, the mapping is not linear, so filtering can become complicated since current filtering in hardware is linear across pixels. This problem is discussed in the following section.

3.4.2 Linear vs. Nonlinear Filtering in Sphere Maps

Because spheres are nonlinear objects, sphere maps are nonlinear mappings. Although the indexing scheme presented earlier is based on the reflected vector, it is also possible to make the nonlinearity more apparent by interpreting the indexing of sphere maps as occurring in a (r, θ, ϕ) space. As seen in Figure 4, the θ and ϕ are derived from the eye vector, **u**, relative to the tangent plane and surface normal, **n**, at the point being sphere mapped. The radius, r, in this case will always be 1 since the vectors

used in sphere mapping are all normalized. This allows the coordinates to retain two degrees of freedom in θ and ϕ. The sphere map coordinates, in this configuration, are shown in Figure 10.

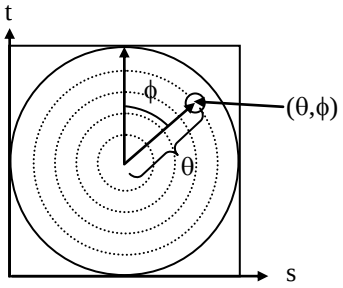


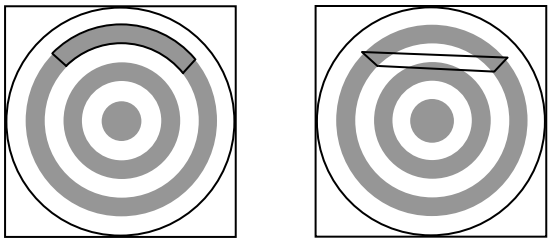
Figure 10: Spherical Coordinates for Sphere Mapping.

Each point on the sphere map can be represented by a (θ, ϕ) coordinate where 0≤θ≤180° and 0≤ϕ≤360°. R is implied to be 1. The (θ, ϕ) coordinate mapping can be related back to the normal (s, t) coordinates through the following formulas:

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} 1/2 + 1/2\sin(\phi)\sin(\theta/2) \\ 1/2 + 1/2\cos(\phi)\sin(\theta/2) \end{pmatrix}. \quad \text{(Equation 3)}$$

The mapping is nonlinear in both s and t. The indexing is based on mapping θ from 0 to ½ based on the sine of the angle and then taking that vector and rotating it in the plane by ϕ. Since the 'origin' for the sphere is at (½, ½), that offset must be added to any values computed.

Since the map is indexed in spherical coordinates, filtering should ideally occur in spherical coordinates as well. However, current filtering in hardware is linear across pixels; so nonlinear filtering presents a problem. [13] Intuitively, the problem can be understood through Figure 11. The coordinate system for sphere maps is non-Euclidean. Therefore, lines in the sphere map's space correspond to arcs in the sphere map's texture image. Near the center of the map, the approximation between the linear and nonlinear mappings is acceptably close. This is important because the center of the map is oriented towards the viewer. The best approximation occurs where the viewer is most likely to look, the center of the screen. As the texture coordinates move closer to the edge of the map, the approximation breaks down as demonstrated in Figure 11.



a b

Figure 11: Accurate Nonlinear Filtering (a) vs. Inaccurate Linear Filtering (b).

Ideally, the area described by the four vertices in Figure 11 would describe an area of similar color to the viewer (Figure 11a). However, linear interpolation on the sphere map would cause incorrect striping to occur with the given map (Figure 11b). When sphere mapped polygons take up a large portion of the screen space, large differences in the reflected vector can make meshing artifacts noticeable in the image as well. Sphere mapping is still effective, though, because it is rare to have large planar portions of a reflective object take up most of the screen. If this situation does occur often, however; then other techniques for creating mirrors are usually employed. For example, one technique is rotating the camera about the plane of the mirror, capturing an image of the camera's view, and using portions of that as a texture map to apply to the mirror's surface. Linear interpolation inherently limits sphere mapping to applications where an individual polygon does not cover a large area of the sphere map.

3.4.3 View Dependence in Sphere Mapping

An interesting feature of sphere mapping as opposed to cubic mapping and parabolic mapping is that, while cubic and parabolic mapping are view independent, sphere mapping is view dependent. View dependence is a problem for environment mapping using sphere mapping because the mapping loses validity as it deviates from the map's center of projection. The loss of validity occurs because traditional environment mapping is calculated in object space while sphere mapping is calculated in screen space. In sphere mapping, the current eye position rendered on the screen is used for indexing. Although the view dependence makes sphere mapping an odd form of environmental mapping, it enables properties dependent on the viewing direction relative to the surface to be modeled at interactive frame rates.

This concludes the discussion on the theoretical aspects of sphere mapping. Subsequent sections will deal with its practical application.

4. Creating Windows with Sphere Maps

This section highlights a practical application of the view dependence inherent in sphere mapping to the simulation of a window as seen from inside a room. The view dependence of sphere mapping allows the image to shift with respect to changes in the viewer's position. An added feature of sphere mapping is that the texture matrix can be used to add some view *independence* to the map. The texture matrix functions as a transformation matrix for textures. It can scale, skew, rotate, or even add perspective to textures. As applied to windows, the texture matrix is used to prevent the sphere map from tilting and moving with the viewer's eye. The texture matrix is key to adding view independent properties to sphere mapping.

The idea is to replace the drawing of objects outside of the window with a sphere map. An overview of the situation is shown in Figure 12 from a top-down perspective. Since the sphere map is view dependent, it will change what it displays based on the orientation of the viewer relative to the window. Unfortunately, since sphere mapping aligns the texture in screen space, as the viewer tilts with respect to the window, the sphere map will also tilt. In order to keep the map fixed with respect to the window, the signed angle between the up vector for the world and the up vector for the viewer can be computed in screen space. This angle is then used to cancel out the rotation of the sphere map with respect to the window which normally occurs when orienting the texture with respect to the viewer.

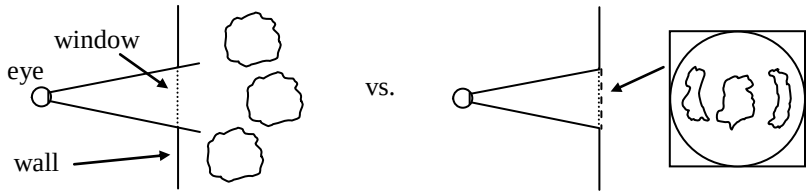


Figure 12: Using Geometry Outside a Window
Versus Simulating the View Outside a Window with a Sphere Map.

Although the screen space rotation can be cancelled out with the texture matrix, rotation of the map in other dimensions is harder to overcome. Since the indexing is based on the eye space reflected vector, as the user changes viewing direction, the contents of the window will change. Figure 13 shows two views of a simulated window using sphere mapping. This rotational dependence is counter to a real window which remains stable with viewer rotation and only changes contents based on changes in viewer position. Hence, windows are only positionally dependent.

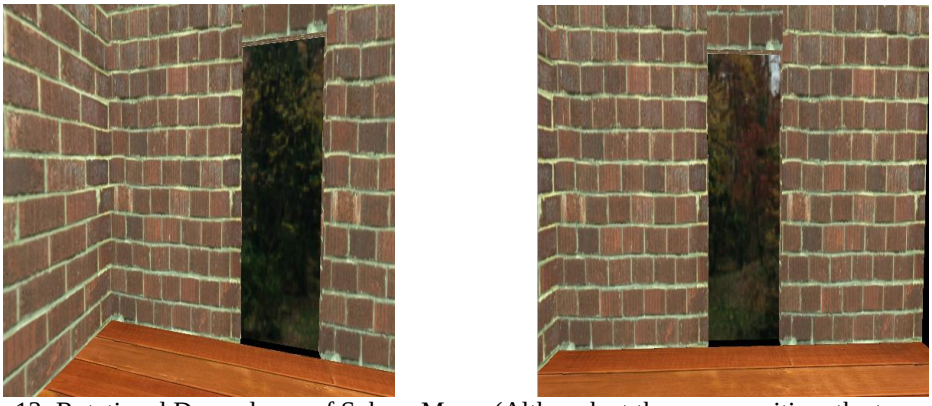


Figure 13: Rotational Dependence of Sphere Maps. (Although at the same position, the two views show different sphere map contents due to rotation.)

One technique to make sphere maps rotationally independent is to invert the rotational part of the object to eye matrix (called the model/view matrix) and apply that to the texture matrix. Under this scheme, the sphere map coordinates are converted into object space and then applied to the object. This approach does not work though; and Equation 2, repeated below, reveals why.

$$\begin{pmatrix} s \\ t \end{pmatrix} = \begin{pmatrix} \dfrac{r_x}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \dfrac{1}{2} \\ \dfrac{r_y}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \dfrac{1}{2} \end{pmatrix} \quad \text{(Equation 2)}$$

In hardware, the texture matrix operates *after* the coordinates in Equation 2 have been created. By that time, any sense of the original eye space reflected vector has been lost due to division, powers, and adding.

Another approach at achieving view independence is changing the texture coordinate generation mode to one that can take advantage of inverting the rotational part of the model/view matrix. Such a texture generation mode is `GL_REFLECTION_MAP_EXT` in OpenGL. This mapping mode calculates an eye-space, reflected vector and delivers it to the texture matrix in its original form (no division, etc.). Applying the inverse of the rotational part of the model/view matrix with these texture coordinates does provide a rotationally independent version of the mapping. However, the application to the sphere has been lost since there has been no division or square root operation. Other types of mappings such as cube mapping and dual paraboloid mapping, which rely more directly on the direction information provided by the reflected vector, are able to use this technique.

In order to achieve a truly rotationally independent version of a sphere map, the `GL_REFLECTION_MAP_EXT` texture coordinate generation mode should be used. However, the texture matrix must be modified to add ½ to s and t in accordance with Equation 2. This addition can be easily performed. In addition, the texture coordinates must be divided by $2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}$. Unfortunately, there is no method of performing this operation in the texture matrix. In order to scale the coordinate properly, the whole indexing process has to be replicated in software. An untoward effect of using software in this instance is that it nullifies any hardware speed up.

A benefit of using sphere mapping is that the outdoor image automatically scales with respect to the viewer's distance from the window. This occurs because the difference between the reflected vectors increases as the viewer approaches the window causing more of the sphere map to be referenced in a given area. This inherent scaling mimicks the properties of a real window. However, because the image is still two dimensional, there is no perspective foreshortening of the scenery as the viewer approaches the window. Another problem with the sphere mapping approach is that there is no parallax as the viewer moves relative to the window. One possible solution to this problem would include the use of the texture matrix in conjunction with *several* sphere maps. There would be one sphere map for each distinct depth

layer, and the texture matrix would scale each so that each layer moved more or less depending on its distance from the viewer. Another solution would be the inclusion of depth information with the sphere map which would allow image warping to be applied to the sphere map. However, both approaches are costly and require multiple passes through the graphics pipeline to work correctly. In the final analysis, sphere mapping provides a rough approximation with little performance penalty.

5. Lighting Models with Sphere Maps

Lighting with sphere mapping is based upon replacing the normal environment map with a map that shows how light would be attenuated under a specific lighting model. These maps are usually used to add the color of the light to the texture already applied to an object. This texture-based lighting method also produces highlights appropriate for the object's location and orientation relative to the lights and the viewer in the environment. Since sphere mapping is indexed using the reflected ray from the viewer's direction, it is a good candidate for such view-dependent surface characteristics. Because the reflected ray retrieves a single value from the mirror direction, sphere mapping is most suited to simulating the specular aspects of lighting models; and analysis is limited to the specular component of the lighting models. The first model to be explored under sphere mapping will be the Phong lighting model.

5.1 Phong Lighting

The Phong model by Bui Tong Phong was first introduced in 1975 as a means of approximating the intensity of light reflecting off a smooth surface (such as plastic). [14] The model can be understood through Figure 14.

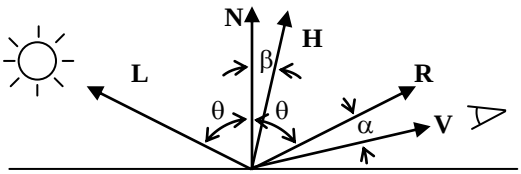


Figure 14: The Phong Lighting Model.

Since sphere maps are intended to emulate the specular nature of lighting and reflection models, sphere mapping only applies to the specular term for Phong lighting. Phong defines the specular term as $I_s = k_s * (\mathbf{R} \bullet \mathbf{V})^n$. In this formula, k_s is the specular coefficient of the surface that indicates (roughly) how much of the total incoming light escapes from the surface to the viewer. **R** is the perfect mirror direction from the light source (in the **L** direction). **V** is the direction towards the viewer. The I_s term uses a dot product raised to a power to show that the highlight on a surface is dependent on how close the vector **R** is to vector **V**. In the diagram in Figure 14, α refers to the angle between **R** and **V**. Since **R** and **V** are unit vectors, $\mathbf{R} \bullet \mathbf{V} = \cos(\alpha)$, Phong's specular term can be rewritten as

$$I_s = k_s (R \cdot V)^n = k_s \cos^n(\alpha) \quad \text{(Equation 4)}$$

The exponentiation of the dot product indicates how sharply the highlight will fall. If n is large the highlight will be sharp and if n is close to 1 the highlight will fade slowly.

5.1.1 $(N \bullet H)^n$ vs. $(R \bullet V)^n$

An approximation to the Phong Lighting model is the use of a halfway vector, **H**, which is the average of the **L** and **V** vectors. Instead of using the dot product of **R** and **V** to calculate the highlight value, the dot product of **H** and **N** can be used to create the highlight. However, the angle between **R** and **V**, α, is twice as large as the angle β between **H** and **N** so n must be chosen differently under this formulation to achieve the same highlight appearance. [15]

The new n, n', can be determined by looking at the equality:

$$k_s (N \cdot H)^{n'} = k_s (R \cdot V)^n .$$

Substituting the definition of **N•H** as |**N**||**H**|cos(β) = cos(β) and **R•V** as |**R**||**V**|cos(α) = cos(α) = cos(2β) results in:

$$k_s(\cos(\beta))^{n'} = k_s(\cos(2\beta))^n .$$

Substituting cosine's double angle identity,

$$k_s(\cos(\beta))^{n'} = k_s(2\cos^2(\beta)-1)^n .$$

Simplifying and taking the logarithm of both sides defines the new n' as

$$n' = n * \frac{\log\left(2*(N\cdot H)^2 - 1\right)}{\log(N\cdot H)} .$$ (Equation 4)

This new n' exponent can then be used in the **N•H** formulation of the Phong model to achieve the same highlight.

5.2 Simple Phong Lighting with Sphere Mapping

Specular lighting (highlights) is a view-dependent effect and, as such, it is appropriate for implementation with sphere mapping. The idea is to store the specular light distribution for a canonical highlight in a sphere map, and then use the texture mapping capabilities of sphere mapping to look up the corresponding highlight value for each vertex and normal.

Highlights for non-vertex points are also referenced from the sphere map as discussed in section 3.4.1. Note that the resultant shading on the primitive is non-linear in contrast to the usual per-vertex Phong. Additionally, for meshes with shared normals, the sphere-mapped Phong lighting provides smooth shading across the mesh which is not achieved with per-vertex Phong lighting due to the Gouraud or bilinear shading. However, sphere-mapped Phong lighting is still an approximation. The lighting is not computed per pixel (i.e., the normals are not interpolated). The scheme relies on the Phong map capturing the nonlinearity of the highlight and approximates the actual distribution described by the Phong lighting model. Given that Phong lighting is not physically based, the per pixel approximation has no effect on the actual realism of the highlights. The smoothness and superior performance of sphere-mapped Phong lighting are enough to justify it over regular bilinear interpolated per vertex Phong lighting. The next section, 5.2.1, discusses the implementation of sphere mapping for the particular case of the light source at the eye's position (similar to a miner's hat); and section 5.2.2 discusses the generalization to arbitrary light source positions.

5.2.1 Light at the Eye's Position (Miner's Hat)

If the light (**L**) used in the Phong model is at the eye, then the reflected vector of the light and the eye coincide. In this special case, the sphere map contains the light intensity distribution as calculated by the Phong model. Since sphere mapping occurs in eye space, the eye will always be in the same direction, (0,0,-1), and only the reflected vector varies. To derive texel values in the sphere map, the location of each texel in the map is converted to a normal vector using the relation (s, t) = (x, y, $\sqrt{1-x^2-y^2}$). Given the constant viewing vector, this normal vector produces a reflected vector per texel. Each reflected vector is used together with the viewing vector to evaluate Equation 4 and to produce the texel highlight value. These calculations are performed as a preprocessing step to compute the sphere map. Once the map is created, the indexing and modulation occur at interactive rates. An example of such a map is shown in Figure 15.

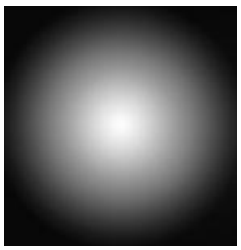


Figure 15: A Phong Light Map.

Once the map in Figure 15 has been calculated, it is then used to add the color of the highlight to the color of the surface giving the surface characteristic Phong highlights. The main problem with this method is that the light is fixed at the viewer's head similar to a miner's hat. Figure 16 shows the result of using a sphere-mapped highlight. A more flexible approach would allow the light *and* the viewer to move relative to the object and have these changes reflected in the sphere map.

Figure 16: Using a Map Similar to Figure 15 to Create Highlights on an Object.
(Notice that the highlight moves with the viewer.)

5.2.2 View Independent Phong Lighting

To achieve the effect of the light not being fixed at the eye's location, sphere map indexing can be modified using the texture matrix and special environment mapping indexing methods. By computing the proper texture matrix before indexing begins, the light can be moved away from the eye for the general Phong lighting model. One approach to removing the view dependence from sphere maps in OpenGL is to take the inverse of the rotational part of the model/view matrix (the current viewer transformation matrix) and place this inverted 3x3 matrix into the texture matrix. In conjunction with this change to the texture matrix, the texture generation method is altered to compute the eye-space, reflected vector (in OpenGL this is referred to as `GL_REFLECTION_MAP_EXT`). By combining these techniques, the object space reflected vectors can be used for indexing. Figure 17 illustrates indexing with the inverted matrix and reflected vector.

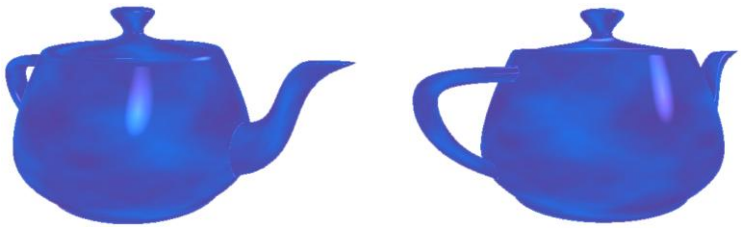


Figure 17: Using the Object Space Reflected Vector to Create Highlights.
(The highlight follows the object more realistically than with the normal sphere map indexing.)

Using object-space reflected vectors means that as the viewer's head changes relative to the orientation of the object, the contents of the environment map change. However, if the viewer rotates about a particular point, the environment mapped object looks the same. Unfortunately, there is still a deficit to this kind of mapping, the problem of scale. As the viewer moves either closer or farther from the Phong-mapped object, the highlight will grow either smaller or larger, respectively. The growth of the highlight is directly proportional to the change in the reflected vectors as the viewer moves either closer to or farther from the sphere-mapped object. Applying a uniform scale in the texture matrix only works for symmetric situations (such as looking at the center of a symmetrical object like a sphere) and does not properly account for the changes at oblique viewing angles. If a nonuniform scaling operation could be performed for each vertex, then this anomaly could be eliminated.

6. Colorgrams

Colorgrams are silvery-surfaced stickers which have the property of reflecting different wavelengths of light based on the viewing angle. Some types of gift wrapping paper also use this eye-

catching effect. This color view dependence can be modeled with sphere mapping with no changes to the texture matrix. The sphere map does not model the outside world at all. Instead, the map consists of rainbow-like rings or arcs of color corresponding to different shading characteristics at different viewing angles. The contents of the map could be derived either programmatically or from measured data. However, there are also many designs that can be created ad hoc in the sphere map which still look acceptable. Figure 18 shows an example of one such map. The surface can then be replaced with the sphere map contents or blended with the surface color or combined with planar reflections to achieve different effects.

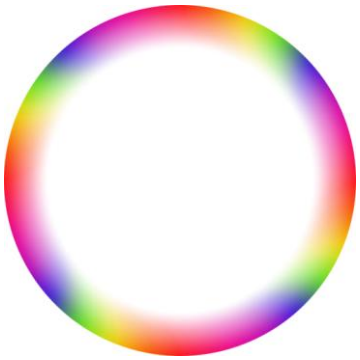


Figure 18: An Empirical Sphere Map for Colorgrams.

An easy way to design such maps is to use a Photoshop design tool called Gradient Designer. The plug-in can create radial gradients easily and blur the results to provide smooth color transitions. Applying another Photoshop filter allows the map to be twisted so that the map is not completely symmetrical (i.e. looking at the material from the left produces different colors than observing the material from the right). In this way, a degree of anisotropy can be modeled in the reflections.

The map contains a large portion of white, especially in the center of the map. This is due to the way the sphere map will be applied to the surface. The map in Figure 18 was designed to be blended with the material being mapped. When a surface is blended with a white texture map in OpenGL, there is no effect on the underlying surface color. Since the center of the map is mostly white, when the viewer looks normal to the surface, the underlying surface color is apparent. As the viewer looks at a more grazing angle at the surface, the rainbow colors toward the edge of the map are blended with the surface to give rise to a colorgram effect. The application of this effect on an object can be seen in Figure 19.

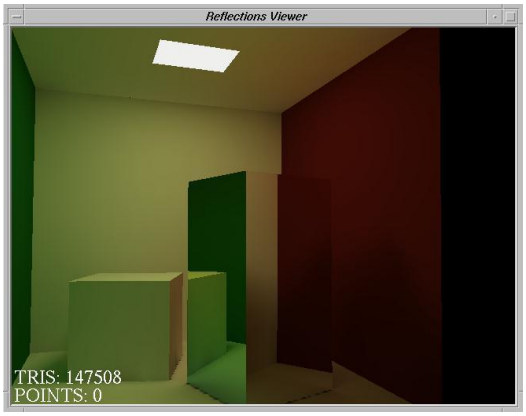


Figure 19: Combining the Sphere Map with Blending.

In Figure 19, the closer box is sphere mapped with the image from Figure 18. The front face of the box shows no color changes because it is relatively normal to the current viewing position. However, the left side shows slight color shifts because it is approaching perpendicular to the current viewing position.

6.1 Color Dependency and Environment Reflection

If the color view dependency is to be incorporated into an environment reflecting object such as the one in Figure 19, there are two options. The first is to use a two-pass method involving the view independent sphere map and a normal sphere map. The view independent sphere map is applied first, and then the colored sphere map is blended onto the object. The two maps cannot be combined because one is view independent while the second is view dependent. The second option involves planar reflections. Planar reflection involves reflecting the camera about a flat surface of the object (usually a triangle) and creating a texture with that camera view. This is repeated for each planar surface of the object. The textures on the object are then blended with the colored sphere map.

The planar reflection method is more accurate than the view independent sphere map because it is generated per frame and samples the environment more finely than a sphere map. However, it is much more expensive. If the object being mapped has a significant amount of curvature or is relatively small in eye space, then view independent sphere mapping would provide higher performance.

6.2 Simulation of Flecks

Colorgram materials often contain groups of reflective components as shown in Figure 20. Sometimes the material contains the components in patterns to create a reflective design on the surface. A random orientation gives the surface a sparkling effect as the light scatters. This section will discuss randomly oriented surface elements.

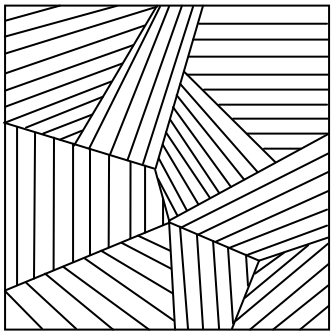


Figure 20: Groups of Randomly Oriented Reflective Flecks in a Colorgram Material.

The fleck effect can be modeled by tessellating the surface so that each fleck has one or more corresponding polygons representing it on the surface. The texture coordinates (generated by the colored sphere map) can be randomly offset using the texture matrix before being used to index into the sphere map. The major drawback of this technique is the large amount of tessellation and coordinate manipulation required to achieve the full-scale effect. Precomputing offsets and tiling groups of clumps are possible methods that can increase performance.

7. Simulation of Reflectance Functions with Sphere Maps

Section 5 discussed the use of sphere maps to simulate the Phong lighting model. This section concentrates on reflectance models and presents methods developed for simulating the Cook-Torrance reflection model as well as bidirectional reflectance functions derived from measured data. The idea is similar to Section 5. The computation of the reflectance function is precomputed and stored in the sphere map. The precomputed values are then indexed by hardware at runtime in the sphere map.

7.1 Cook-Torrance

The Cook-Torrance model highlights the view dependent capabilities of sphere maps to convey the wavelength dependence of the reflected light. The Cook-Torrance model was introduced in 1982 as a formulation for a physically-based, illumination model for computer graphics. [8] For comparison, the Phong model is empirical, which means that a surface may look pleasing but is not physically plausible.

The Cook and Torrance model is based on physical principles. It has the desirable properties of energy conservation and wavelength dependence (color shift). The Cook Torrance model was originally designed for the simulation of polished metal surfaces and includes a Fresnel term which changes the highlight color based on the angle the viewer makes with the surface. Unfortunately, the full Cook-Torrance model cannot be simulated by sphere maps alone. This is because sphere maps only contain data relating to ideal reflections off a surface. Hence, the specular aspects of the model will only be discussed.

7.1.1 Mathematical Formulation of the Cook-Torrance Model

The formula for Cook-Torrance is:

$$Energy = sR_s + dR_d \quad s + d = 1 \text{ . (Equation 5)}$$

Basically, Equation 4 defines the outgoing energy from a surface as a convex combination of specular and diffuse terms. Sphere maps model the R_s aspect of the model. The R_d term combines both the diffuse and ambient characteristics of the surface.

The formula Cook and Torrance used for the specular term (R_s) is given here:

$$R_s = (DG)\left(\frac{F}{\pi(N \cdot L)(N \cdot V)}\right). \quad \text{(Equation 6)}$$

An important part of the Cook-Torrance model is the calculation of a Fresnel term, called F, in the lighting calculations. This term introduces color and reflected energy shifts dependent on the properties of the surface relative to the viewer. When the viewer is looking normal to the surface, most of the intrinsic surface color is seen. As the viewer approaches a grazing angle, the surface acts more like a mirror (like most surfaces) and the reflected image becomes much more dominant. However, this reflected image is still modulated by the surface properties. Hence as the viewer nears grazing angle, the surface properties can favor certain wavelengths of light. Evidence of F in the sphere map for the model can be seen as changes in color and brightness in the reflectance map towards the edges. Bastos et al. calculated a reflectance map for copper under the Cook-Torrance model for increasing realism in interactive walkthroughs. [2] The result is shown in Figure 21.

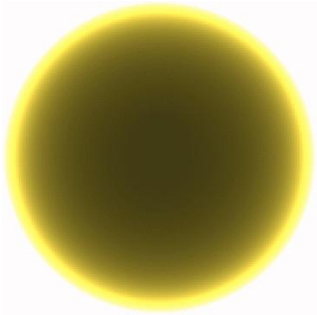


Figure 21: The Reflectance Map for Copper Showing the Color Shift of the Material Based on Changes in View as Calculated by the Cook-Torrance Model. (From [2])

Cook-Torrance also includes a distribution term (D) which describes statistically how the light will scatter when it hits the surface. When more light is scattered off the surface, the D term becomes larger. Variations in the D term cannot be modeled solely by a sphere map. This is because sphere maps only index in the mirror direction. Sphere mapping assumes there is no perturbation of the rays off the surface. To achieve different D values, the sphere map can be applied first. Then, some blurring of the applied map is performed; or, alternatively, jittering of the viewpoint can be applied. Blurring the map after applying it to the object performs an averaging of nearby points which is similar to averaging nearby rays. Jittering the viewpoint around the object and accumulating the images operates on the same principle. Each accumulation of jitter averages nearby points and hence nearby rays. Both of these methods have a significant performance penalty, but blurring through convolution is supported in hardware on some systems.

The G term describes how much of the reflected light escapes the surface. G depends on the normal (**N**), the vector to the light (**L**), the eye vector (**V**), and the halfway vector (**H**). **H** is the average of **L** and **V**. There are several terms which govern G which will be presented only briefly because they reduce to 1 under the reflectance model used for sphere mapping. G is formulated as:

$$G = \min(1, G_s, G_m)$$
$$G_s = \frac{2(N \cdot H)(N \cdot L)}{V \cdot H}$$
$$G_m = \frac{2(N \cdot H)(N \cdot V)}{V \cdot H}. \quad \text{(Equation 7)}$$

However, the sphere map model of Cook-Torrance is a *reflectance model* because the sphere map stores the values for reflected vectors. The reflected vectors used in the sphere map are derived from the eye. Since sphere maps handle reflectance instead of lighting, there is no real **L** vector under this formulation. The **V** vector drives a reflected vector into the scene. The sphere map model uses **V**=**L**; and thus, **N**=**H**. Using these substitutions,

$$G = \min(1, G_s, G_m)$$
$$G_s = \frac{2(N \cdot N)(N \cdot L)}{V \cdot H} = 2$$
$$G_m = \frac{2(N \cdot N)(N \cdot V)}{V \cdot H} = 2. \quad \text{(Equation 8)}$$

The G term is thus 1 when cast as a reflectance model and therefore factors out of Cook-Torrance as a reflection model.

Also, as a reflection model, the denominator of R_s scales the sphere map (notice that **N•L** = **N•V** in this reflectance model).

The modulation of the surface through the model can be appreciated most directly through changing the viewpoint of the objects. Two different screen shots are included to show the view dependent aspects of the Cook-Torrance model under BRDF mapping.

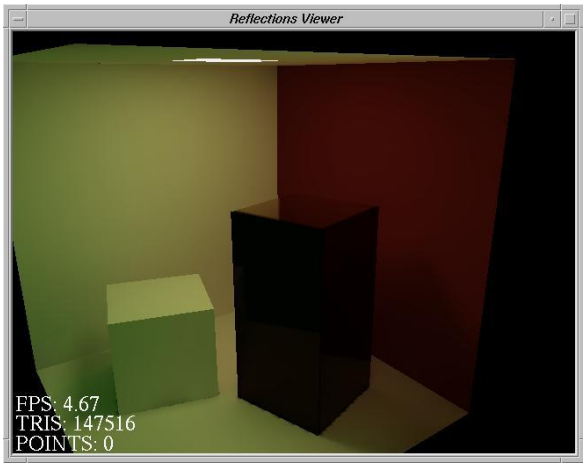


Figure 22: Cook-Torrance as a Reflection Model Using Sphere Maps at a Non-Grazing View.

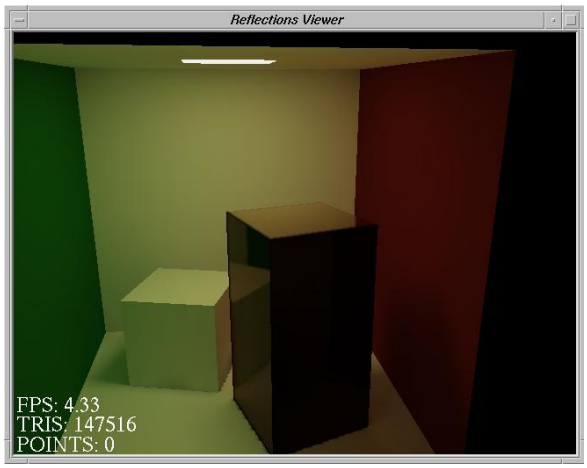


Figure 23: Cook-Torrance as a Reflection Model at a Near Grazing Angle. (Note the color shift of the reflected image on the side of the box.)

As you can see in Figure 22, when the near face of the box is facing the viewer, it is black. This is because there are no diffuse properties set for the box, and the specular component is very low due to the non-grazing situation. However, in Figure 23 the viewer is at a grazing angle and the mirror-like properties of the surface take effect. The color of the reflected image also takes on different shades depending on the viewer's angle with the surface. This change in color corresponds to the dominance of the Fresnel term in calculating the reflected light.

8. Using Measured BRDF Data

All of the sphere mapping techniques presented thus far use empirical methods or values calculated from computational models of how light interacts with surfaces. These images may be convincing but are still approximations to real surfaces. This section discusses how data measured from surfaces can be used to create sphere maps. The data discussed in this section was downloaded from Cornell University's Light Measurement Lab.

8.1 Data Format

Cornell University's Light Measurement Lab uses several instruments to help characterize the properties of samples of surfaces. [9] They offer the full BRDF energy distribution for several sample surfaces. The data from Cornell's lab is taken at 8 different incoming angles (10°, 20°, 30°, ... , 80°), and data is collected over the quarter sphere around those angles. Figure 24 illustrates the organization of the measuring device and the data gathered.

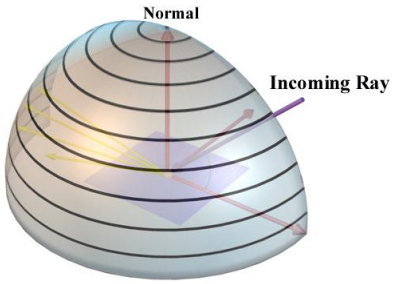


Figure 24: The Domain of Data Capture for the Cornell Data for an Incoming Ray.

An incoming ray of a single wavelength is aimed at a target of the material under study. The material scatters the incoming ray into all directions. Only those on half of the hemisphere are recorded (the

other half is assumed to be symmetric). About 178 points are sampled on the quarter sphere, and the energy returned by the surface at that direction and particular wavelength are recorded. Data is gathered for 30 wavelengths resulting in ~1500 total sample points (8 quarter sphere's worth) for a complete characterization surface under many viewing conditions. The data is represented in a Matlab .mat workspace file. The actual data shown here was part of a Matlab file containing the data for Ford's Mystique Laquer.

As an example of what the data reveals, Figure 25 shows the data (in false color) for an incoming direction of 50°:

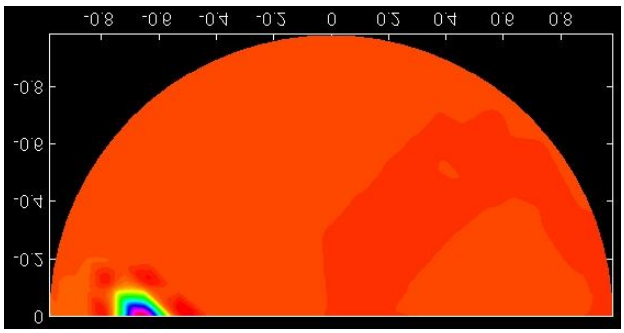


Figure 25: A False Color Representation of the Cornell Reflectance Data for a Material for an Incoming Angle of 50°.

Figure 25 is actually a projection of the quarter sphere data onto the xy plane and colored by reflected energy values. The highlight corresponds to the highest energy in the reflected direction of the incoming light (a little less than 50°). For creating a sphere map, the reflected direction (mirror direction) energy is recorded for each of the incoming directions. Then these results are concatenated into a single row of pixels which corresponds to the reflected data at all the incoming directions. Figure 26 explains this process:

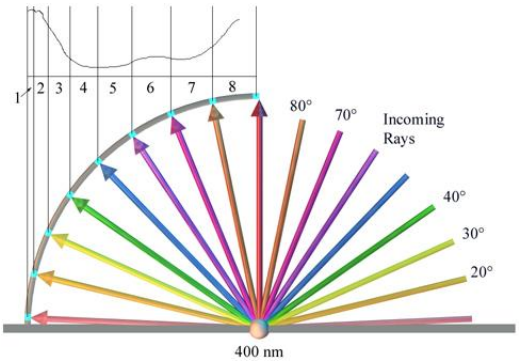


Figure 26: The Method of Constructing the Reflected Energy for a Surface with BRDF Data.

Data was pieced together using the strongest reflected energy for each wavelength. The strongest reflected energy was assumed to be in the mirror direction. Recalling that data is garnered from 8 distinct incoming directions of light, there are 8 corresponding mirror reflections. So each incoming direction contributes to $1/8^{th}$ of the single line of pixels describing the surface. Each different shaded arrow in Figure 26 signals a different data set for the same wavelength. The number of sample points varies because the measuring device itself shadows part of the data generated. Therefore, holes may appear in parts of the data set at certain incoming angles. The piecing together of data is repeated for 750nm, 550nm, and 400nm or red, green, and blue respectively. Since the data is in Matlab format, a Matlab script can be used to get the pieces of data needed. After the data has been collected for red, green, and blue, Photoshop can then be used to join the data together into a single map. A map for Mystique laquer was created with this process. However, any material with BRDF data could be formulated into a sphere map.

9. Conclusion

This paper has described the general process of environment mapping with a special concentration on sphere mapping. An overview of texture mapping provided the context for the discussion of several environment mapping techniques. Ideal, sphere, cubic, and dual paraboloid mappings were summarized in terms of their indexing methods, storage, benefits, and limitations. The indexing method used for sphere mapping was explored in detail because of its nonintuitive mechanics. Since linear interpolation can lead to very large distortions in the area defined by the texture coordinates generated under sphere mapping, the difference between linear and nonlinear interpolation was also clarified. Additionally, equations for mapping reflected directions in (θ, ϕ) space to (s, t) coordinates were derived. These equations make the creation of sphere maps related to different lighting models easier to generate.

After the properties of sphere maps had been explored, other applications of sphere maps were discussed. Lighting models reformulated as reflection models proved to be good candidates for sphere maps. Sphere maps also were effective at simulating colorgrams resulting in interesting color surface effects. The colorgram maps used anisotropic sphere maps (seen as non-symmetric sphere maps) and blending to create purely view dependent color shifts. General BRDF data was also determined to be amenable to sphere mapping. The method described in Section 8 can be used to generate isotropic BRDF maps.

Items with little view dependence, like windows, proved much more difficult to simulate under sphere mapping. Unless the OpenGL texture matrix operation is altered, cube maps or dual paraboloid maps would make better candidates.

Future applications of sphere mapping (and environment mapping in general) will most likely start by combining view dependent and view independent effects since multiple textures and novel coordinate generation techniques are already beginning to be supported in commodity hardware. An object could use a view independent mapping for reflection, but rely on sphere mapping for purely view dependent properties such as wavelength shifts. Also, environment maps could benefit from including depth information about the environment. By including depth values, perspective foreshortening and parallax effects could be achieved with greater realism. Until textures with depth are supported in hardware, layered sphere maps could provide an avenue of simulating environments that can be segmented into distinct depth layers (such as near and far field). Finally, fast dynamic generation of environment maps could provide another avenue for more realistic object and environment interaction.

Bibliography

1. Rui Bastos and Wolfgang Stuerzlinger, "Forward Mapped Planar Reflections," Univ. of North Carolina Technical Report, 1998, TR98-026. ftp://ftp.cs.unc.edu/pub/publications/techreports/98-026.ps.Z.

2. Rui Bastos, Kenneth Hoff, William Wynn, and Anselmo Lastra, "Increased Photorealism for Interactive Architectural Walkthroughs," Proc. of the 1999 Symposium on Interactive 3D Graphics, 1999, pp. 183-190.

3. Eric A. Bier and Kenneth R. Sloan, Jr. "Two-Part Texture Mappings," IEEE CG&A, Vol. 6, No. 9, Sept. 1986, pp. 40-53.

4. James F. Blinn and Martin E. Newell, "Texture and Reflection in Computer Generated Images," Comm. ACM, Vol. 19, No. 10, Oct. 1976, pp. 542-547.

5. James F. Blinn, "Simulation of Wrinkled Surfaces," Computer Graphics (Proc. SIGGRAPH 78), Vol. 12, No. 3, Aug. 1978, pp. 286-292.

6. David Blythe et al. "Lighting and Shading Techniques for Interactive Applications, " SIGGRAPH 99 Course Notes, Course #12, Aug. 1999.

7. Ed Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces," PhD Thesis, Univ. of Utah, Dec. 1974.

8. Robert L. Cook and Kenneth E. Torrance, "A Reflectance Model for Computer Graphics," Computer Graphics (Proc. SIGGRAPH 81), Vol. 15, No. 3, Aug. 1981, pp. 244-253.

9. Donald P. Greenberg et al., "A Framework for Realistic Image Synthesis," Computer Graphics (Proc. SIGGRAPH 97), Vol. 33, No. 3, Aug. 1997. pp. 477-494.

10. Ned Greene, "Environment Mapping and Other Applications of World Projections," IEEE CG&A, Vol. 6, No. 11, Nov. 1986, pp. 21-29.

11. Wolfgang Heidrich and Hans-Peter Seidel, "View-Independent Environment Maps," Proc. SIGGRAPH/Eurographics Workshop on Graphics Hardware, 1998. http://www9.informatik.uni-erlangen.de/eng/research/rendering/envmap.

12. Kenneth E. Hoff, "Derivation of OpenGL Sphere-Mapping, Automatic, Texture-coordinate Generation for use in Environment Mapping," http://www.cs.unc.edu/~hoff/techrep/spheremap.html.

13. Mark J. Kilgard, "Real-time Environment Reflections with OpenGL," Game Developer's Conference. 1999. http://www.nvidia.com/Marketing/Developer/DevRel.nsf/TechnicalPresentationsFrame?OpenPage.

14. Bui-T. Phong, "Illumination for Computer Generated Pictures," Comm. ACM, Vol. 6, No. 18, June 1975, pp. 311–317.

15. Alan Watt, *Advanced Rendering and Animation Techniques*, Addison-Wesley Publishers, 1992.