Environment Mapping

Simone Kriglstein
Günter Wallner

Abstract

In this paper environment mapping is presented. First we will discuss environment mapping in general and the methods of parameterization like, sphere -, cubic –and parabolic maps. For sphere, parabolic and cubic maps the way of indexing is shown. Then ways of prefiltering of environment maps are shown. In detail we present how this can be done with the Phong reflection model and how the Fresnel term can be used. In addition environment mapping with the help of BRDF´s is presented. Also environment mapped bump mapping (EMBM) will be discussed. In this chapter we discuss mainly how the environment is mapped on an object with a structured surface. In the last chapter an introduction into environment mapping with OpenGL is given.

1 Introduction

The aim of the game designer is to design games, which get more and more realistic and that can not be done without environment mapping. It has the appearance for everybody that this method is not so important. However, can you imagine a motor racing game in which the surroundings are not reflected on the car body (in Figure 1.1 you can see, how the surrounding is reflected on a bonnet)
or a hero of an action game which does not have a reflection on the water.
Modern 3D hardware accelerators are supporting a lot of gadgets, such as texture mapping, bump mapping, pixel shading, lighting effects, shadow mapping, T&L, environment mapping and so on. Games which are released know or in future are using these techniques to provide a realistic environment as possible. To get attention, new games have new and improved graphic effects. Things on that we do not concentrate in real life, are missed immediately in games or other interactive applications.
Surfaces that are reflecting the environment, correctly broken shadows and many other small things are making the game experience perfect.

Figure 1.1
Car which reflects the environment

In our paper we will discuss mainly environment mapping. But for what is environment mapping good for ? With the help of environment mapping, an object can reflect the environment. For that texture mapping can be used. But some problems are occurring. The problem how the textures are saved must be solved (i.e. to save memory). The second problem is that the calculation of mapping the 2-dimensional to 3-dimensional coordinates must be possible in a short time, so that these techniques can be used in interactive applications. For parameterization three different kinds are existing. The most famous methods are spherical, cube and parabolic maps. We will discuss each of the methods mentioned above and enumerate the advantages and disadvantages of those parameterizations. Since there are different kinds of surfaces (wood, plastic,...) with different kinds of reflection properties (Figure 1.2 shows a very good example of reflection), these properties should be taken into account when generating an environment map. The reflection properties can be divided into three parameters: diffuse, glossy and mirror. For metallic surfaces a Fresnel term can be used. Those properties are applied to the environment map in a prefiltering step. For glossy surfaces we will explain the widely used Phong model. This model is physically not correct, because it makes some simplifications. But, it is simple and the quality is good enough for most graphic applications.

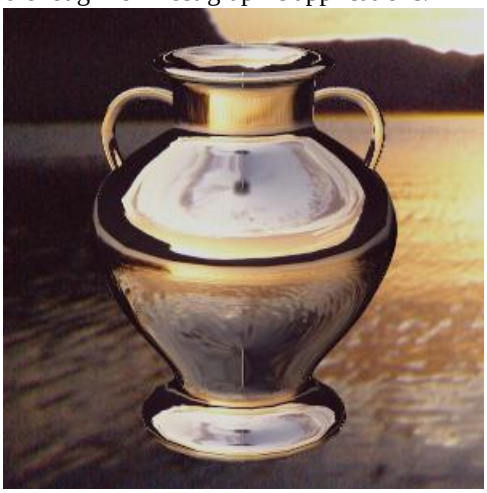

Figure 1.2
Reflection with environment mapping

We will also take a closer look on how to represent BRDF´s with environment mapping. A BRDF describes how much light is reflected when light makes contact with the surface.
In chapter 10 of the paper we will discuss environment mapped bump mapping. These technique allows us to model surfaces with bumps. Examples of use are bark, metal sheet, scratch and so on.

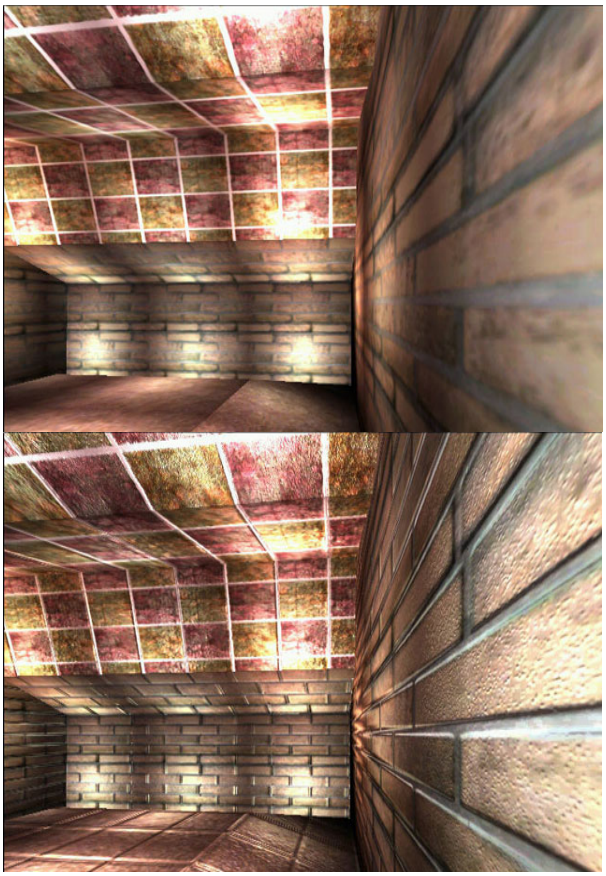

Figure 1.3
Top: without bump mapping
Bottom: with bump mapping

Scenes can be represented more realistically by environment mapped bump mapping (EMBM). This method is particularly used for water effect.
Figure 1.3 shows how a scene looks better when EMBM is used.
The wall looks much more realistic with reflections and structures. The ground was also changed, it has a structure at the lower picture.
In the last chapter environment mapping with OpenGL is presented.

2 Environment Mapping

„ Projection of the whole environment on a texture surface “

For us reflection effects are a normal thing and not many people are thinking about this phenomena.
But games without reflections are boring or bad. Reflections are making the games look realistic and exciting.

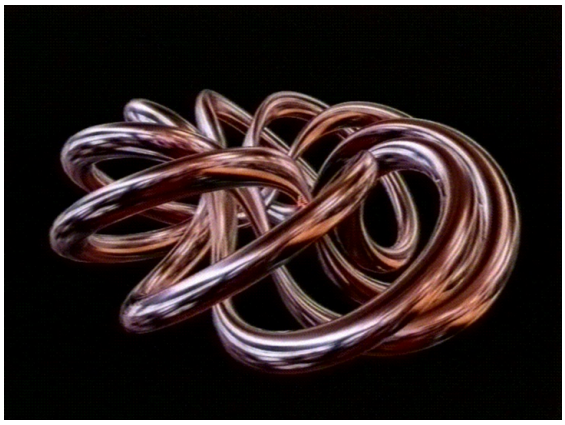

Figure 2.1
Example of reflection with environment map

What exactly is environment mapping now?

Environment mapping [10] is used to reflect the surroundings on an object. Thus, an object must be covered with a texture of the real object surface as well as with the environment texture. A good example of environment mapping is shown in Figure 2.1.

Environment mapping is a separate technique rather than as a category of texture mapping. This method was first developed by Blinn and Newell in 1976 [17].

Environment maps are particular textures that describe for all directions the incoming or outgoing light at one point in space.

Environment mapping [2] may be achieved through texture mapping in different ways. One way requires six texture images, each corresponding to a face of a cube, that represent the surrounding environment. At each vertex of a polygon to be environment mapped, a reflection vector from eye off of the surface is computed. This reflection vector indexes one of the six texture images. As long as all the vertices of the polygon generate reflections into the same image, the image is mapped onto the polygon using projective texturing. If a polygon has reflections into more than one face of the cube, then the polygon is subdivided into pieces, each of which generates reflections into only one face. Because a reflection vector is not computed at each pixel, this method is not exact, but the same results are quite convincing when the polygons are small.
Another way is to generate a single texture image of a perfectly reflecting sphere in the environment. This image consists of a circle representing the hemisphere of the environment behind the viewer, surrounded by an annulus representing the hemisphere in front of the viewer.
At each polygon vertex, a texture coordinate generation function generates coordinates that index this texture image, and these are interpolated

across the polygon. If the (normalized) reflection vector at a vertex is

$$r = (x \quad y \quad z) \qquad \text{Equation 1}$$

and

$$m = \sqrt{2(z+1)} \qquad \text{Equation 2}$$

then the generated coordinates are x/m and y/m when the texture image is indexed by coordinates ranging from –1 to 1 (the calculation is diagrammed in Figure 2.2).

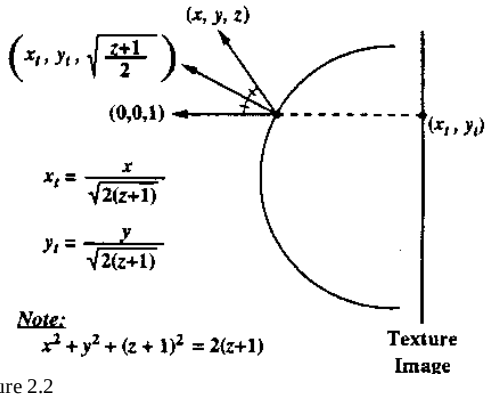

Figure 2.2
Spherical reflection geometry

The distance between viewer and object is important, because this is the reason for distortions. Because there is a singularity in the viewing direction, since all points where the viewing vector is tangential to the sphere show the same point of environment. Moreover, the object can not reflect itself, because the calculations can not be performed in real time. But, it is possible with ray tracing algorithms [16].

There are many different techniques for implementing environment mapping like cube mapping [see Figure 2.3, see also chapter 5], spherical mapping [for more information about this method see chapter 4] and parabolic mapping [see chapter 6].

Figure 2.3
Example for cube environment mapping

3 Parameterization

A parameterization is a mapping from directions to texture coordinates. This mapping should meet some properties in order to be useful in interactive rendering:

- for walkthroughs of static environment, it should not be necessary to create a new map every frame
- it should be easy to create a new environment map from perspective images of the scene
- calculating the texture coordinates should be easy and simple to implement in hardware

Three methods which have gained some importance in hardware accelerated and interactive rendering are: spherical maps, cube maps and parabolic maps which will be explained in the following.

3.1 Spherical Environment Maps

Spherical environment mapping (see [3,4,14]) is based on the analogy of a infinitely small, perfectly mirroring metal ball centered around the object. The environment map is the image that a orthographic camera sees when looking at the ball from a certain viewing direction. A sample of a spherical map is shown in Figure 3.1.1.
This kind of parameterization has two big disadvantages. First, it is not suitable for viewing directions other than the original one and second, it

Figure 3.1.1
Top: Spherical environment map
Bottom: Reflecting Teapot

does not reflect changes in the scene, as they happen.
The major reason why spherical environment mapping is still used is, that the lookup can be calculated efficiently with simple operations in hardware. Also, they are good enough to create cheap static reflections which are in most cases good enough for game reflections. Now, how does this lookup process works ?
In Figure 3.1.2 a image of the lookup process is shown. A spherical environment map which has been generated for a camera pointing in direction v_o, stores the corresponding radiance information at the point where the reflective sphere has the normal h. If v_o is the negative z- Axis then the 2D texture coordinates are the x –and y- axis of the normalized halfway vector h. If the direction is close to the viewing direction distortions are likely to appear, because these directions correspond to the tangential areas of the ball.

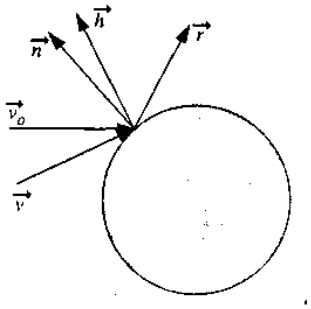

Figure 3.1.2
Lookup process in a spherical environment map

The creation of a spherical map requires a texture mapping step. In that step the perspective images of the environment are warped into the spherical form. Let‘s take a look, how that can be done.

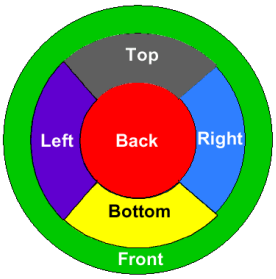

Figure 3.1.3
Spherical environment map shape

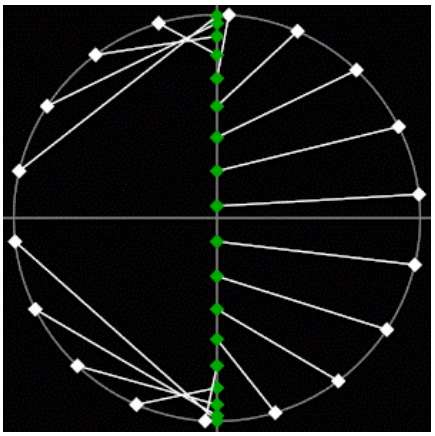

Figure 3.1.4
Cross section of a sphere on which the environment is mapped

First the environment in which the reflective model should be placed must be rendered as viewed from the desired position of the model. This is done by rendering six images in the directions front, back, top, bottom, left and right. After the six images are rendered they are sampled to generate a environment map. Such an environment map is shown in Figure 3.1.3.
Figure 3.1.4 shows the cross section of a sphere on which the environment is mapped. In this example, the eye point is placed at the right hand side of the figure. The white points on the perimeter of the circle are mapped to the green points in the 2D environment map. Each point on the sphere's cross section is connected to a point on the plane by a white line.

3.2 Cube Maps

Cubic environment mapping [6] is the oldest of the environment mapping techniques covered here. This kind of mapping was invented by [1]. Cube Maps or cubical environment maps consist of six independent perspective images from the center of a cube throw each of his faces. Thus, the generation of such a map consists only of rendering the six images. In comparison to spherical environment maps no warping step is required. In Figure 3.2.1 the unfolded cube map is shown and in Figure 3.1.2 how that cube map surrounds an object. A second benefit of cube environment maps is, that they are viewpoint independent and do not need to be recalculated in scenes within a static environment.

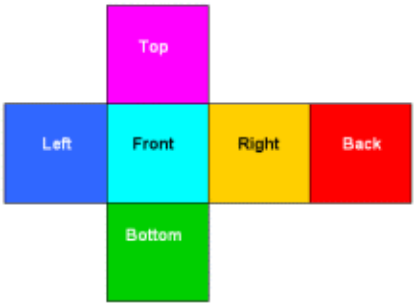

Figure 3.2.1
Unfolded cube environment map

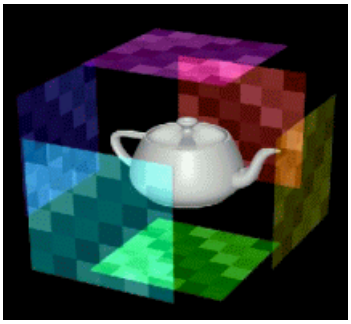

Figure 3.1.2
Cube map surrounds object

Indexing of Cube Maps, in comparison with spherical maps and parabolic maps is simple. As mentioned by Zimmons [18] two different ways of indexing are possibly. The first is based on projected pyramids through pixels and the second is implemented in cube mapping hardware. We will discuss the second method.

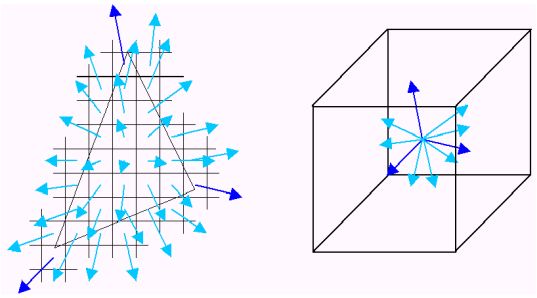

Figure 3.2.3
Fragments of a polygon along with their reflected vectors (left). The representation of those reflected vectors inside the cube map (right)

First, the reflection vector relative to the eye is calculated per vertex of a given polygon. This reflection vector is in object- space and has three components (r_x,r_y,r_z). This handles the corners of the polygon (dark blue vectors in Figure 3.2.3) but leaves the middle undefined. The middle of the polygon is filled by performing an interpolation of the reflective vector across the polygon. Note, that such a interpolation instead of recalculation introduces errors. This interpolation can be performed incrementally from one fragment to the next. Those vectors are shown light blue in Figure 3.. Once a fragment has been assigned to a reflective vector, the texel (defined through 3 coordinates) which gives the fragment it´s color must be determined. How can that be done ? The first step is to find out which component is the largest; x, y or z. This component and the sign of the component defines the cube face. The coordinates for that faces can be calculated with the following equations.

$$s = \frac{\frac{sc}{|m|}+1}{2} \qquad t = \frac{\frac{tc}{|m|}+1}{2}$$ Equation 3

s and t are the texture coordinates. m is the component with the largest value. The values for sc, tc and m can be looked up in the Table 1

Major axis direction	m	sc	tc
+rx	r_x	$-r_z$	$-r_y$
-rx	r_x	$+r_z$	$-r_y$
+ry	r_y	$+r_x$	$+r_z$
-ry	r_y	$+r_x$	$-r_z$
+rz	r_z	$+r_x$	$-r_y$
-rz	r_z	$-r_x$	$-r_y$

Table 1

For a better understanding a small example. We want to calculate the (s,t) coordinates for the reflection vector (0.2, 0.4, 0.3). In this example the r_y component is the largest (positive sign). In the table we take a look at the $+r_y$ row. In that row sc uses $+r_x$ (this is because the growth of positive x agrees with the growth of positive u on the $+r_y$ face [see Figure 3.2.4]) and tc uses $+r_z$.

Cube environment mapping is already supported by DirectX 7 and OpenGL.

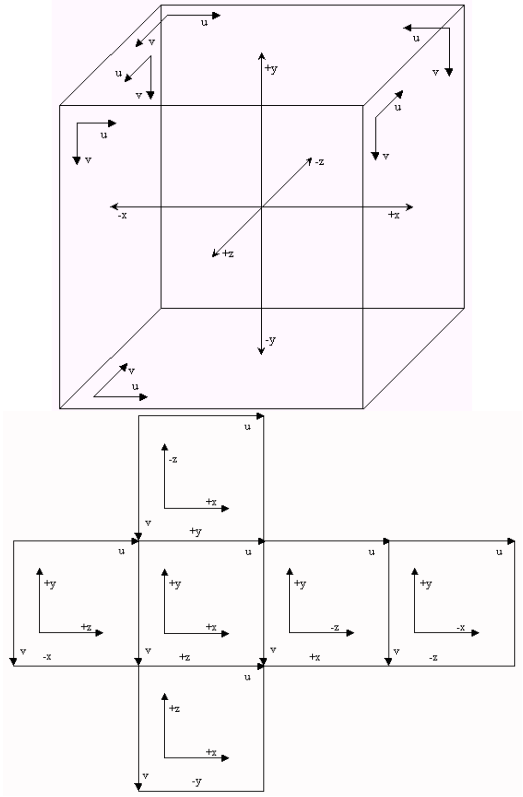

Figure 3.2.4
Coordinate system for a cube map (Renderman standard)

3.3 Parabolic Maps

Dual paraboloid environment mapping was invented by [3,4]. They are based on the same analogy used to describe spherical environment maps. But here the hole environment is stored in two different textures. Each texture contains the information of one hemisphere. The geometry behind this, is shown in Figure 3.3.1. It can be shown that it uses less than one third of the pixels that cube maps do. However, about 25% of the pixels are not used.

Another big advantage is that it can be used very efficiently in hardware.
In comparison with spherical environment map the problem of distortion is better solved. Nevertheless, it does not reach the quality of cube maps in that case. In Figure 3.3.2 you can see a typical parabolic map. And in Figure 3.3.3 a torus rendered with the help of this map is shown.

The indexing of such a map is a little bit complicated. For an exact explanation see [3,4] and [18]. We will now only give a short overview of the indexing method. The indexing scheme is nearly linear, so it allows a significant speed up with the help of hardware support.
The map for dual paraboloid mapping is made from the point of view of the object being environment mapped and considers the object as the origin. The paraboloid map assumes that the map is created from a view looking down the positive z axis. The „front“ map is on the positive z axis while the „back“ map is facing negative z.
The generation of the (x,y) values for the parabola corresponds to the following matrix equation:

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = P * S * M^{-1} * \begin{bmatrix} r_{e,x} \\ r_{e,y} \\ r_{e,z} \\ 1 \end{bmatrix}$$

Equation 4

Where M is a linear transformation mapping the environment map into eye space. The inverse of M thus maps the reflection vector r_e back into object space. When the viewer views the object, a vector v_e is generated for the point being environment mapped from the surface to the eye. Next the normal n_e at that point is calculated.
For a view vector and a normal, both in eye space, the reflected vector can be computed in eye space by calculating $r_e = 2(n_e.v_e)n_e - v_e$, the standard mirror reflection equation.
S performs the addition of the object space reflected vector from the viewing vector d_o. S is defined as follows:

$$S = \begin{bmatrix} -1 & 0 & 0 & d_{o,x} \\ 0 & -1 & 0 & d_{o,y} \\ 0 & 0 & -1 & d_{o,z} \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Equation 5

Note, that this definition of S is not the same as in [3,4], because the reverse reflection vector equation has been used. P is defined as

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Equation 6

In order to generate appropriate texture coordinates, a final matrix multiplication must be performed.

$$\begin{bmatrix} s \\ t \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix}$$

Equation 7

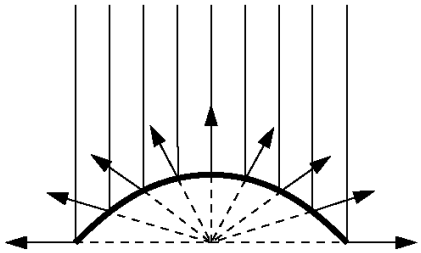

Figure 3.3.1
The rays of an orthographic camera reflected of a paraboloid.

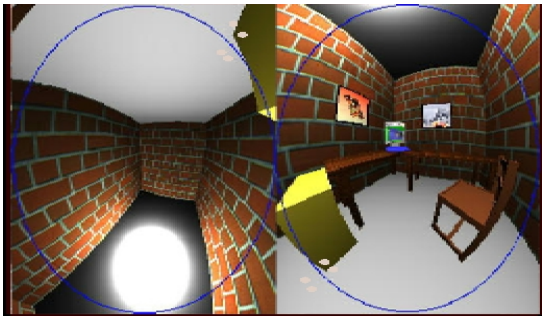

Figure 3.3.2
Parabolic map

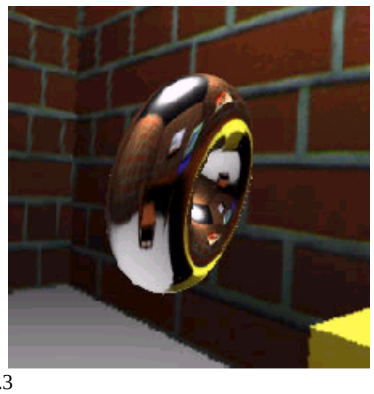

Figure 3.3.3
Rendered torus that uses this environment map.

4 BRDF

We explain BRDFs because we need it for prefiltering of environment maps. A BRDF (for detailed information see [5,19]) describes how much light is reflected, when the light makes

contact with a certain material. The degree to which light is reflected depends on the viewer and the light position (relative to the surface normal). So, a BRDF must capture the view- and light dependent nature of reflected light. Thus, a BRDF is a function of incoming light direction and outgoing view direction relative to a local orientation at the light interacting point.

A BRDF can be written as addition of three components BRDF = Diffuse + Glossy + Mirror (see Figure 4.1)

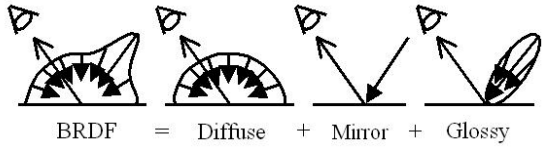

Figure 4.1
BRDF Term

A general BRDF can be written as

$$BRDF_{\lambda}(\theta_i,\phi_i,\theta_o,\phi_o,u,v) \qquad \text{Equation 8}$$

λ indicates that the BRDF depends on the wavelength. θ_i,ϕ_i are representing the incoming light direction in spherical coordinates and the parameters θ_u,ϕ_u the outgoing light direction (also in spherical coordinates). Last but not least the parameters u and v represent the surface position in texture space. Sometimes this two parameters are not included, then the BRDF is called position invariant. In that case the properties of reflection do not vary on spatial positions. Thus, that is only valid for homogenous materials. We will take a closer look at position invariant BRDFs. For simplicity the λ can be omitted. But keep in mind that the BRDF must be calculated for each of the three color channels (RGB).

Since a BRDF measures how light is reflected, we must know how much light arrives at a surface position. Irradiance is measured in energy per area (Watts/m^2). So, it is not really satisfactory to talk about the light incoming from a single direction. It is better to take the neighborhood into account.

With that knowledge we can come to the exact definition of a BRDF. A BRDF is given by

$$BRDF = \frac{L_o}{E_i} \qquad \text{Equation 9}$$

L_o is the radiance from the surface in direction w_o and the irradiance arriving from direction w_i is called E_i.
Taking some physical observations into account the equation above is written as

$$BRDF = \frac{L_o}{L_i \cos(\theta_i) dw_i} \qquad \text{Equation 10}$$

BRDFs can be divided into two classes

- isotropic
- anisotropic

The term isotropic is used to describe BRDFs that represent reflectance properties that are invariant with respect to rotation of the surface around the surface normal vector (i.e. smooth plastic). Anisotropic is the same for rotational variant reflectance properties (i.e. velvet).
These two classes have two important properties. First, the property of reciprocity, which only means, that if the incoming and outgoing direction is changed the BRDF keeps the same. This property is shown in Figure 4.2. The second property, the conservation of energy, simple is: the quantity of light reflected can´t be higher than the quantity of incoming light, which is shown in Figure 4.3.

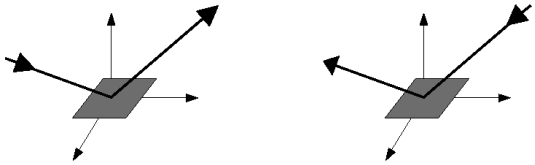

Figure 4.2
Reciprocity

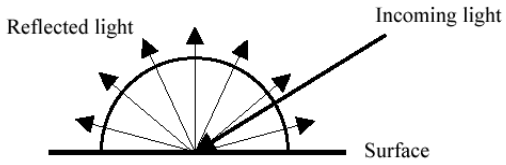

Figure 4.3
Conservation of energy

That means the sum over all outgoing directions of a BRDF must be less than one. When we consider a continuous hemisphere, this is mathematically written as

$$\int_{\Omega} BRDF_{\lambda}(\theta_i,\phi_i,\theta_o,\phi_o)\cos\theta_o dw_o \le 1 \qquad \text{Equation 11}$$

Now, we will define a general lighting equation that expresses how to use BRDFs for computing the illumination produced at a surface point. Suppose we have a scene and we are trying to determine the illumination of a surface point as seen by the observer . In the real world, the whole environment surrounding an object in the scene has an effect on the illumination of every surface point.
The quantity of light reflected into the direction of the observer is a function of all the incoming light and the BRDF at this point. The radiance of outgoing light L_o is given by

$$L_o = \int_{\Omega} L_{oduetoi}(w_i, w_o) dw_i \qquad \text{Equation 12}$$

where

$$L_{oduetoi}(w_i, w_o) \qquad \text{Equation 13}$$

represents the irradiance reflected in direction w_o from direction w_i. Ω represents the hemisphere of incoming light directions. Often it is better to think about things in a discrete space. In such a case the

equation above becomes a sum over a finite set of incoming directions.
For each incoming direction w_i the amount of reflected light depends on the BRDF, which leads us to

$$L_{oduetoi} = BRDF(\theta_i, \phi_i, \theta_o, \phi_o)E_i \quad \text{Equation 13a}$$

where E_i is irradiance incoming from direction w_i. In order to make the amount of light relative to the surface element the light must be "spread out". If we take this into account following replacement can be made.

$$E_i = L_i \cos\theta_i dw_i \quad \text{Equation 14}$$

The same replacing we have done the get Equation 10. For interactive computer graphics not the hole hemisphere is taken into account because it can not be computed fast enough. So interactive applications are using only a small number of point light sources to calculate the illumination of a surface.

For example, suppose we have a scene with n light sources. In this case, the local illumination of a surface is given by

$$L_o = \sum_{j=1}^{n} BRDF(\theta_i^j, \phi_i^j, \theta_o, \phi_o)G_i^j \cos\theta_i^j$$

Equation 15

Where G describes the radiant intensity for a light source.
This is the general BRDF lighting equation for n light sources and can be used for prefiltering of environment maps.

5 Reflections with environment mapping

5.1 Prefiltered environment maps

The idea behind prefiltered environment maps is to apply the reflection step to an environment map of incident light, resulting in an map representing exitant light (see [11]). That means, that an entry in the environment map does not contain the incident light from a certain direction, but the exitant light that results from the incoming light in a global illumination simulation. This approach is useful for non specular reflection. The Phong reflection model, as example, will be described later [see glossy prefiltering of environment maps, chapter 5.3].
The prefiltering process can be thought of applying a (BDRF dependent) kernel filter to an unfiltered source map.
The general kind of an environment map is five-dimensional. Two dimensions are representing the viewing direction v and three dimensions represent the coordinate frame of the reflective surface {n,t,nxt}. Because five- dimensional textures have enormous memory requirements some dependencies are dropped.

As mentioned above a BRDF f_r is needed to create a prefiltered environment map which will be applied to the original map. The incoming light Li(x,l) from all directions l can be viewed as the original map. To store the radiance of the reflected light the incoming light must be weighted with the BRDF f_r. This considerations will lead us to the following equation, where w(v,n,t) represents the viewing direction and w(l,n,t) the light direction relative to the frame. The reflected light towards all viewing directions is captured by the prefiltered environment map from a fixed position x.

$$L(x;\vec{v},\vec{n},\vec{t}) = \int_{\Omega} f_r(\vec{w}(\vec{v},\vec{n},\vec{t}),\vec{w}(\vec{l},\vec{n},\vec{t}))L_i(x;\vec{l}) < \vec{n},\vec{l} > d\vec{l}$$

Equation 16

5.2 Diffuse prefiltering of environment maps

At a diffuse object, light has the same radiance into all directions when the light meets the surface of the object. This reflection is independent from the view direction, but the angle of the lightbeam is important.

One possible approach is image- based and uses a prefiltered environment map. This approach was mentioned by [1].
For such a map any parameterization (explained before) can be used. The only difference is that diffusely prefiltered maps are always referenced via the normal of a vertex in environment map space, instead of via the reflection vector.

5.3 Glossy prefiltering of environment maps

There are different methods to handle this, but here we will take a closer look at the Phong reflection model.
The Phong model [15], introduced by B.-T. Phong in 1975, is a linear combination of three components: diffuse, specular and ambient. The ambient part models the reflection of light which arrives at the surface from all directions, after being bounced around the scene in multiple reflections. The diffuse reflection models the reflection from non- shiny surfaces. A perfectly diffuse reflecting material reflects light equally in all directions. The third component, and for glossy reflections the most important, models the reflection from mirror like surfaces.

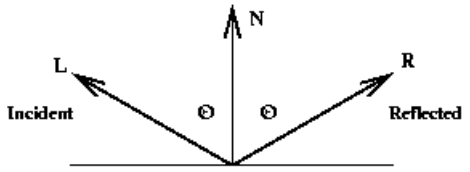

Figure 5.3.1
Reflection of a perfect mirror

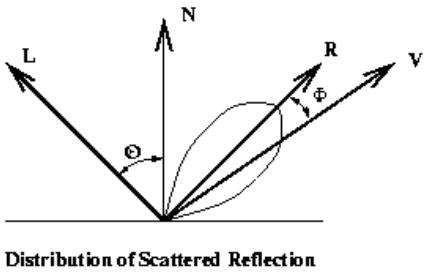

Figure 5.3.2
Reflection of a non perfect mirror

A perfect mirror [see Figure 5.3.1] will reflect light arriving at the surface at an angle of incidence θ (to the normal) at an angle of θ (to the normal) in the same plane as the incident light. Thus, only a viewer on the reflected ray can see the reflected light. In practice no surface is a perfect mirror which leads us to the picture shown in Figure 5.3.2.

The specular contribution is given by a function depending on the angle between the viewing direction and the mirror direction and from n, an index that simulates the roughness of the surface.
Small integer values of n are simulating less glossy surfaces and a large value for n simulates a glossy surface. For a given n a reflection lobe is generated, where the thickness of the lobe is a function of the roughness [see Figure 5.3.3]
The specular reflection term produces a so- called highlight. A highlight is a reflection of the incoming light spread over an area of the surface.
The Phong- model is not a physically correct model because it makes some simplifications, like all geometry except the surface normal are ignored (that means light sources and the viewer are located in infinity). But, because of these simplification the Phong model is a very-liked reflection model and the realism provided by the model is sufficient for many applications.
For a Phong prefiltered environment maps the general Equation 16 becomes

$$L_{phong}(x;\vec{v},\vec{n},\vec{t}) = \int_{\Omega} k_s \frac{<\vec{r}_v(\vec{n}),\vec{l}>^N}{<\vec{n},\vec{l}>} L_i(x;\vec{l}) <\vec{n},\vec{l}> d\vec{l}$$

Equation 17

The integral is an integral over a hemisphere of all directions.
The BRDF f_r has been replaced by the Phong BRDF (shown by Lewis[12])

$$f_r(\vec{v},\vec{l}) := k_s \frac{<\vec{r}_v(\vec{n}),\vec{l}>^N}{<\vec{n},\vec{l}>}$$

Equation 18

the rest is the same as in equation 16.
The two new parameters k_s and N in the equation of the Phong BRDF are used to control the shape and size of the lobe. If we take a closer look at equation 17 we will see that we can make some simplifications. The factor k_s is independent from l, so we can take k_s in front of the integral. The term <n,l> can be cut and the tangent t is not used and can be discarded. Instead of indexing the environment map with v and n it can be reparameterized, so that it is directly indexed by r_v, the reflection vector. After all these simplifications we get a 2- dimensional environment map.
Some samples of Phong environment maps are shown in Figure 5.3.4.

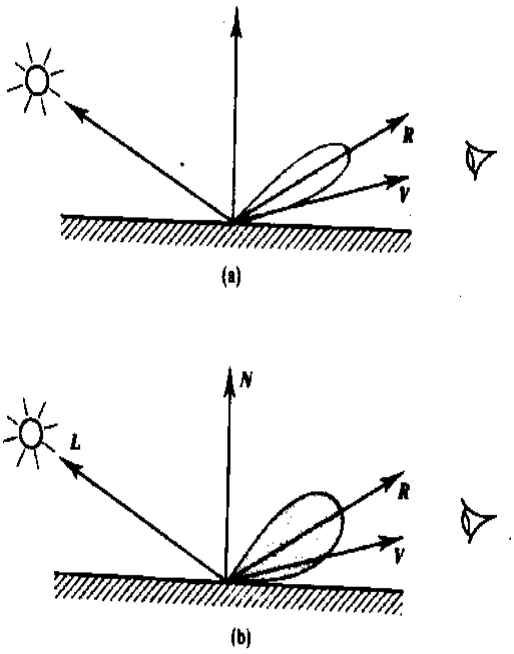

Figure 5.3.3
(a) surface with a large n (b) surface with a small n

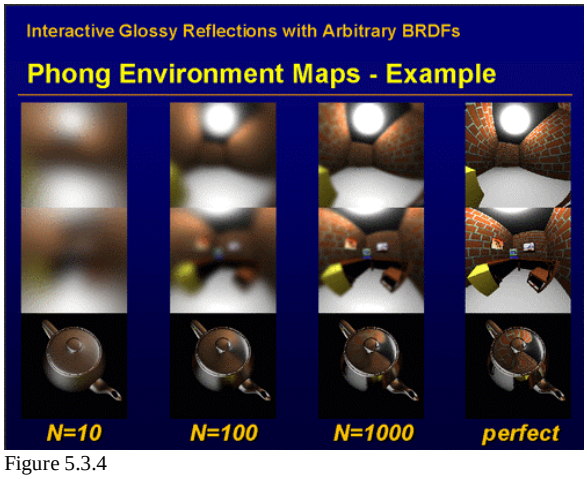

Figure 5.3.4
Phong environment maps with different N
from left to right: N=10, N=100, N=1000, N→∞

To avoid some of the inadequacies of the original Phong model the Phong model can be extended with a Fresnel term, which modulates the fraction of reflected light depending on the incident angle of light, and a weighted sum of a diffuse and a Phong environment map.

In 1982 Cook and Torrance extended the Phong model. The improved model still separates the

reflected light into a diffuse and specular component, and the improvements only concentrate on the specular component. The diffuse part is calculated in the same way as before. The Cook, Torrance model is most successful in rendering shiny- metallic like surfaces.
But the simpler Phong model is today still as popular as in previous days.

$$F = \frac{(g-c)^2}{2(g+c)^2}\left[1+\frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right]$$
$$c = <\vec{n},\vec{v}>,\ g^2 = n^2 + c^2 - 1$$

Equation 19

n is the index of refraction and describes the optical density. For metallic surfaces the index of refraction is very high, so that the Fresnel term is almost one, independent of the angle between the surface normal n and the light direction. Thus, for metallic surfaces the incoming illumination (stored in the environment map) can be used directly as the outgoing illumination. However, for non- metallic surfaces that approach can´t be used. Because for such materials the reflection depends on the angle of the incoming light. For such materials the mirror reflection should be weighted by the Fresnel term for the angle between the surface normal and the reflected viewing direction r_v. As we know the angle of incidence equals the angle of reflection.

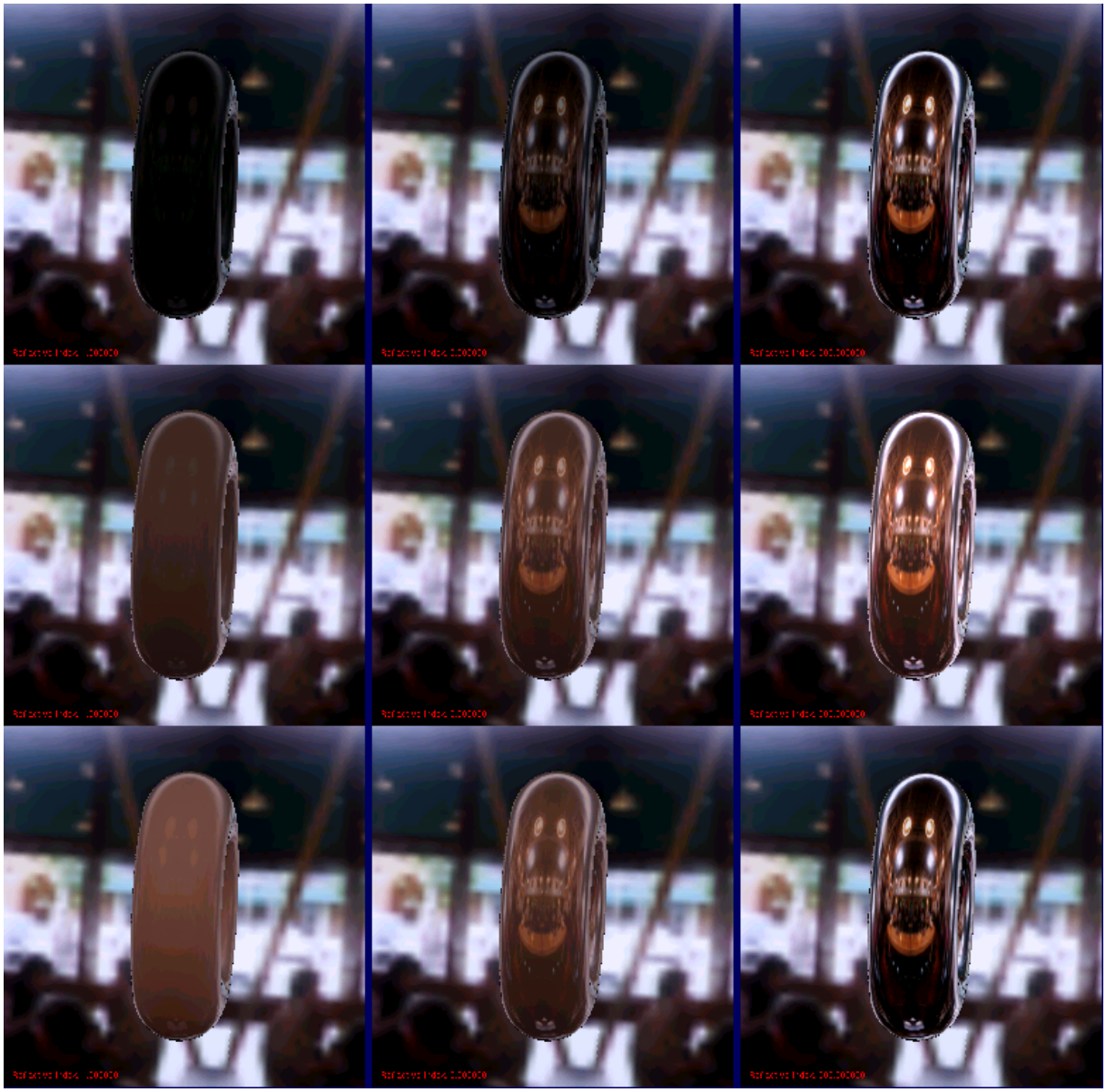

Figure 5.4
Top row: Fresnel weighted mirror term
Center row: Fresnel weighted mirror term plus diffuse illumination
Bottom row: Fresnel blending between mirror and diffuse term

6 Fresnel Term

The Fresnel term is a physical term which describes the reflectivity of a material depending on it´s optical density and the angle of the incoming light.

Thus, the angle, mentioned one sentence above, is the same as the angle between the surface normal and the viewing direction v.

Under the presupposition that the material of a object does not change (that means that n is constant) the Fresnel term for reflected viewing direction can be stored in a 1- dimensional environment map. The hole term now does only depend on the viewing direction, so that this assumption is possible. The mirror part (L_{mirror}) of the given surface is than multiplied with the appropriate Fresnel Term. Last but not least the diffuse part ($L_{diffuse}$) of the material is added. Thus, the outgoing radiance is

$$L_{out} = F \cdot L_{Mirror} + L_{Diffuse} \qquad \text{Equation 20}$$

With the help of the Fresnel Term, materials with a transparent coating can be simulated.

$$L_{out} = F \cdot L_{Mirror} + (1-F) L_{Diffuse} \qquad \text{Equation 21}$$

The term above only means that only the light which is not reflected by the surface hits the surface below and is there diffusely reflected.
In Figure 5.4 some images are shown which are illustrating the theoretical stuff above.

7 Environment mapped bump mapping (EMBM)

Bump mapping (developed by Blinn (1978) [17]) is an elegant method that enables a surface to appear wrinkled or dimpled without the need to model complicated geometry. In Figure 7.1 you can see the differences between a smooth and a bumpy surface.

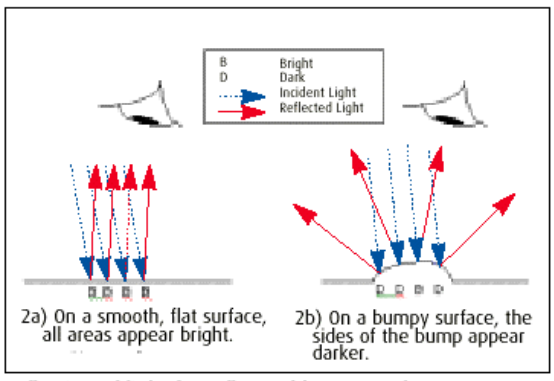

Figure 7.1
Reflections of a smooth surface (left) and of a bumpy surface (right)

With environment mapped bump mapping reflective surfaces can be created.
We need three different textures. A „normal“ texture, a so called bump map and an environment map. See Figure 7.2
A bump map has a value for each texel, which defines an appropriate value in the environment map. A texel (short for texture element) is the smallest element (pixel) of a texture on an object in three dimensional space.

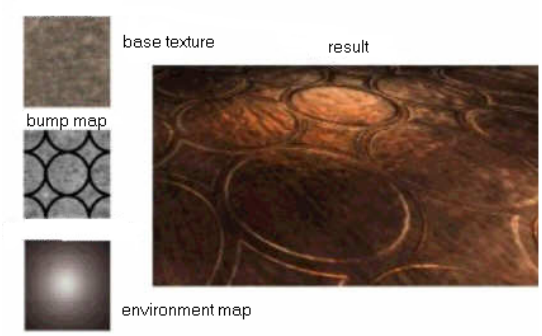

Figure 7.2
The three textures that are needed for EMBM

A very good example which shows the differences between environment mapped bump mapping and non EMBM can you see in Figure 8.1.

Example [9]:
An environment map is lied onto a sphere and a light source should illuminate the sphere from the right (see Figure 7.3). For that reason an appropriate environment map is produced, which handles the tasks of a light map.

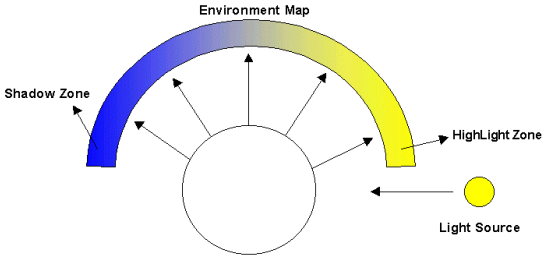

Figure 7.3
Sphere with a light source placed on the right

Without bumps the sphere would be illuminated equally. But, if you lay a bump map over the object, the texels are handled with the bumps on a different place at the environment map to consider the bend (see Figure 7.4).
One side of the object is closer to the light source, so the environment map is brighter on the right side. For each pixel, these information's are located in the bump map.

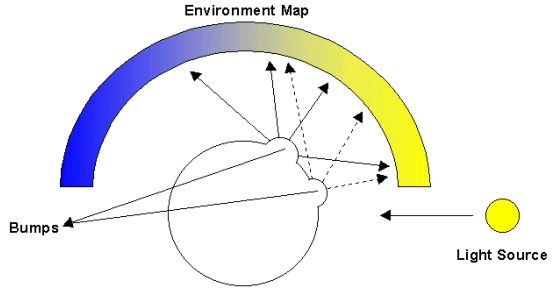

Figure 7.4
Indexing of environment map

EMBM can be used in two different ways, for bump mapping with a light map (see Figure 7.5) or for real environment mapped bump mapping (see Figure 7.6). The second method can be used for reflection- effects (i.e. water).

Figure 7.5
Water with the help of bump mapping (picture taken from the game Expandable)

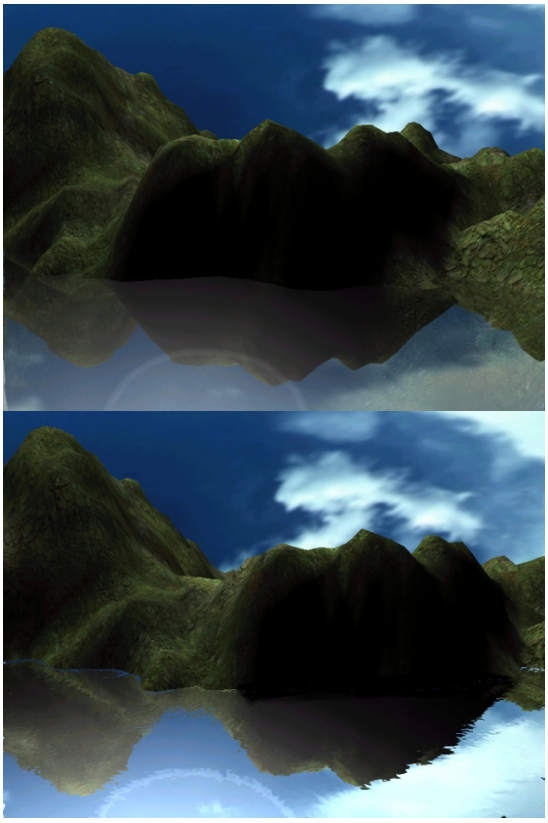
Figure 7.6
Example for real environmental bump mapping
Top: without EMBM
Bottom: with EMBM

More complicated and more hardware- intensive are dynamically light sources, because the light map must be recalculated every time when the light- direction changes.

With cube environment maps the same effects are possible, but it is more expensive.
Cube environment mapped bump mapping makes environment mapping possible in all directions. Non- cube environment mapped bump mapping is limited to few directions (because of distortions).

8 OpenGL

8.1 Sphere Mapping

For a detailed OpenGL specification see [13].
Sphere Environment Mapping is a quick way to add a reflection to a metallic or reflective object in your scene. To use sphere mapping in OpenGL, the following steps must be performed:

1. Bind the texture containing the sphere map.
2. Set sphere mapping texture coordinate generation.
3. Enable texture coordinate generation.
4. Draw the object, providing correct normals on a per-face or per-vertex basis.

We will explain the theoretical stuff above based on an example [8].
For environment mapping we need a texture for the object. We have to load an image file and transform it to an OpenGL texture.
First we create some space to store intermediate image data and load the image into that temporary storage. Where `texture_file` identifies the file of the image which is used for the texture. Note that `texture_file` must be from type `*FILE`.

```
AUX_RGBImageRec *localTexture[1];
localTexture[0] =
auxDIBImageLoad(texture_file);
```

Now space for the texture must be generated. Also, you must tell OpenGL that it is a 2-dimensional texture. After this two steps are performed, the texture can be generated.

```
glGenTextures(1,&textures[texID]);
glBindTexture(GL_TEXTURE_2D,
textures[texID]);
glTexImage2D(GL_TEXTURE_2D, 0, 3,
localTexture[0]->sizeX,
localTexture[0]->sizeY, 0, GL_RGB,
GL_UNSIGNED_BYTE,
localTexture[0]->data);
```

`texID` identifies the texture. Each texture must have an seperate identifier.

With the aid of `glTexEnvf` the texture environment parameters are set. `GL_TEXTURE_ENV` and `GL_TEXTURE_ENV_MODE` must not be altered. Instead of `GL_DECAL`, `GL_BLEND` or `GL_MODULATE` can be used.

Figure 8.1
Top: without EMBM
Bottom: with EMBM
(screenshots taken from Slave Zero)

```
glTexEnvf(GL_TEXTURE_ENV,GL_TEXTUR
E_ENV_MODE,GL_DECAL);
glTexGeni(GL_S,GL_TEXTURE_GEN_MODE
,GL_SPHERE_MAP);
glTexGeni(GL_T,GL_TEXTURE_GEN_MODE
,GL_SPHERE_MAP);
```

The two `glTexGeni` calls are setting the texture generation mode for S and T to sphere mapping. The texture coordinates S, T, R & Q relate in a way to object coordinates x, y, z and w. If you are using a one-dimensional texture (1D) you will use the S coordinate. If your texture is two dimensional, you will use the S & T coordinates.
The next two calls to `glTexGen` very important, as they are the ones that create the Environmental Mapping effect. This
function is responsible for automatically creating texture coordinates, so instead of calling `glTexCoord` to assign texture coordinates, they are automatically assigned by OpenGL. The first parameter specifies the axis that we want our coordinates to be created for. GL_S is used for the x-axis and GL_T is used for the y-axis. The R and Q coordinates are usually ignored. The Q coordinate can be used for advanced texture

mapping extensions, and the R coordinate may become useful once 3D texture mapping has been added to OpenGL, but for now the R & Q coordinates can be ignored. The S coordinate runs horizontally across the face of our polygon, the T coordinate runs vertically across the face of our polygon.
The combination of `GL_TEXTURE_GEN_MODE` with `GL_SPHERE_MAP` creates the appropriate texture coordinates for the environmental mapping effect.

Before creating any of the objects a call to `glBindTexture` is done in order to specify that the following objects will use the specific texture. A call to `glEnable` with a parameter of `GL_TEXTURE_2D` is also done to enable 2D texturing.

```
glBindTexture(GL_TEXTURE_2D,textur
es[0]) ;
glEnable(GL_TEXTURE_2D) ;
glEnable(GL_TEXTURE_GEN_S) ;
glEnable(GL_TEXTURE_GEN_T) ;

// create here an object
// this object will be environment
// mapped
// with the texture specified
// in glBindTexture

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

8.2 Cube Mapping

We will know describe how cube mapping is done with OpenGL (see [7]). OpenGL V1.2 has a new extension, called `EXT_texture_cube_map` which provides a new texture generation scheme for cube map textures. For cube mapping the texture is a set of six 2D images representing the faces of a cube. Cube map texturing requires the ability to access these six images at once. Note that the six images must be quadratic. This feature is supported only be newest hardware, whereas sphere mapping is supported by older hardware too.

We can divide cube maps into static and dynamic cube maps. Dynamic means that the cube map texture is rerendered every frame. Thus, we can represent an environment which changes. A dynamic cube map texture is generated the following way. First, the six images from the point of view of the reflective object are rendered. Each image belongs to one of the faces. In OpenGL the `glCopySubTexImage2D` command copies each rendered cube face into the cube map. Now, the scene can normally be rendered using the dynamic cube map when rendering the reflective object. Such dynamic cube maps are significantly more expensive than static ones (means, that when the view changes the texture is not updated), but it can still be done at interactive rates.

The OpenGL cube map extension adds a new target, called `GL_TEXTURE_CUBE_MAP_EXT`. This enumerant is passed to `glBindTexture`, `glTexParameter`, `glEnable` and `glDisable` when using cube map textures. The texture cube map extension makes a distinction between the cube map "texture as a whole" and the six texture images. So, the above target is not used for `glTexImage2D` and related commands, which are used for 2D textures. The six image targets are:

```
GL_TEXTURE_CUBE_MAP_POSITIVE_X_EXT
GL_TEXTURE_CUBE_MAP_NEGATIVE_X_EXT
GL_TEXTURE_CUBE_MAP_POSITIVE_Y_EXT
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y_EXT
GL_TEXTURE_CUBE_MAP_POSITIVE_Z_EXT
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z_EXT
```

Which of the six targets is used to render a point on an object can be seen in Table 1 in the cube map section.

These cube map "texture image" targets are passed to commands such as `glTexImage2D`, `glCopyTexImage2D` and so on. As mentioned above, the cube map images must always have square dimension to form a cube. That means, that they must have the same dimension and the same width and height. To check if the texturing will work, there is a special target for cube map textures called `GL_PROXY_TEXTURE_CUBE_MAP_EXT`. Since every face must have the same size we do not need such a target for every image. `GL_MAX_CUBE_MAP_TEXTURE_SIZE_EXT` indicates the maximum cube map texture size that is supported by the OpenGL implementation.

How the images of a cube map texture are set is shown in following code sequence.

```
GLubyte face[6][64][64][3];
for (i=0; i<6; i++)
{
      glTexImage2D(GL_TEXTURE_CUBE
      _MAP_POSITIVE_X_EXT + i,
      0,
      GL_RGB8,
      64,
      64,
      0,
      GL_RGB,
      GL_UNSIGNED_BYTE,
      &face[i][0][0][0]);
}
```

In the above example each of the faces is 64x64 RGB image. The six image targets have one advantage. The "texture image" targets are ordered in a sequential way, so the six images can be simply set in a loop, by adding one to the target. This example is realized without mipmaps. To establish

mipmaps instead of `GL_TEXTURE_2D`, `gluBuild2DMipmaps` is used.

To use cube map textures in an application it must be enabled. This is done as follows:
`glEnable(GL_TEXTURE_CUBE_MAP_EXT)`
And with following instruction the extension can be disabled:
`glDisable(GL_TEXTURE_CUBE_MAP_EXT)`

Two new texture coordinate generation modes have been added to OpenGL. The two modes are generating the eye-space reflection vector or normal vector in the (s,t,r) coordinates. Here an example for a reflection map:

```
glTexGenfv(GL_S,
GL_TEXTURE_GEN_MODE,
GL_REFLECTION_MAP_EXT);

glTexGenfv(GL_T,
GL_TEXTURE_GEN_MODE,
GL_REFLECTION_MAP_EXT);

glTexGenfv(GL_R,
GL_TEXTURE_GEN_MODE,
GL_REFLECTION_MAP_EXT);

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

For a normal map the `GL_REFLECTION_MAP_EXT` enumerant is simple changed into `GL_NORMAL_MAP_EXT`. These two modes work only correctly if per-vertex normals are supplied.

9 Conclusion

We have presented how environment maps can be parameterized. For each parameterization we showed how the indexing is done. As shown in the OpenGL chapter most of this parameterizations can be practically used for 3D graphic programming.
Also, prefiltering of environment maps, like diffuse and glossy prefiltering was shown. As example for glossy prefiltering the widely used Phong model was explained. Note, that there are many other models which are more physically accurate, but are not so often used because they are more complicated.
In chapter 10 the fundamentals of environmental mapped bump mapping was discussed. Based on examples we have shown that this method is preferred in games nowadays.

10 References

[1] Greene Ned. Applications of world projections. IEEE Computer Graphics and Applications, pages 21-29, August 1986

[2] Haeberli Paul, Segal Mark, Texture Mapping as a Fundamental Drawing Primitive. In Fourth Eurographics Workshop on Rendering, pages 259-264, 1993

[3] Heidrich Wolfgang, Seidel Hans-Peter. Realisic, Hardware-accelerated Shading and Lighting. In Computer Graphics (Proceedings Annual Conference Series), pages 174-176, 1999.

[4] Heidrich, Wolfgang. Interactive Display of global Illumination Solutions for Non-Diffuse Environments, Eurographics, pages 2-9, 2000.

[5] http://www.nvidia.com /Marketing/Developer/DevRel.nsf/bookmark/B0C2609E737F9C278825698A0002B42F

[6] http://www.nvidia.com /Marketing/Developer/DevRel.nsf/bookmark/EC702BA2A8D553048825686600832F80

[7] http://www.nvidia.com /Marketing/Developer/DevRel.nsf/bookmark/B4AFCEDB4AF0B84A8825681E0076ADB6

[8] http://www.dev-gallery.com /programming/opengl/env_mapping/env_map1.htm

[9] http://www.3dconcept.ch /artikel/bump/index.html

[10] http://www.3dconcept.ch /artikel/environment/

[11] Kautz Jan, Vázquez Pere-Pau, Heidrich Wolfgang, Seidel Hans-Peter. A Unified Approach to Prefiltered Environment Maps.

[12] Lewis Robert R. Making shaders more physically plausible. In Computer Graphics (Eurographics Conference Issue), pages 1-13, June 1993.

[13] Mark Segal, Kurt Akeley, The OpenGL Graphics System: A specification, 1999 http://trant.sgi.com/opengl/docs/Specs/glspec1.1/glspec.html

[14] Mizutani Yoshihiro and Reindel Kurt. Environment Mapping Algorithms http://home.san.rr.com/thereindels/Mapping/Mapping.html

[15] Watt Alan, 3D Computer Graphics, second edition, Addison-Wesley, page 96-100, 1993.

[16] Watt Alan, 3D Computer Graphics, second edition, Addison-Wesley, page 267, 1993.

[17] Watt Alan, 3D Computer Graphics, second edition, Addison-Wesley, page 225-262, 1993.

[18] Zimmons Paul. Spherical, Cubic, and Parabolic Environment Mappings. Dezember 1999

[19] Zimmons Paul. Exploring BRDF Mapping. CS236 Project, Spring 1999 http://www.cs.unc.edu/~zimmons/cs236/BRDFmap.html