Environment Maps And Their Applications

Wolfgang Heidrich

Max-Planck-Institute for Computer Science
Im Stadtwald, 66123 Saarbrücken, Germany
heidrich@mpi-sb.mpg.de

1 Introduction

In this part of the tutorial we are going to discuss environment maps [1] and their applications to interactive rendering. Environment maps are particular textures that describe for all directions the incoming or outgoing light at one point in space. The main use of these maps is to simulate reflections in curved objects, but they can do much more than that. In particular in hardware-accelerated renderers, environment maps are often used to store precomputed directional information that is too expensive to compute on the fly.

The basic idea of environment maps is that, if a reflecting object is small compared to its distance from the environment, the incoming illumination on the surface really only depends on the direction of the reflected ray. Its origin, that is the actual position on the surface, can be neglected. Therefore, the incoming illumination at the object can be precomputed and stored in a 2-dimensional texture map.

If the parameterization for this texture map is cleverly chosen, the illumination for reflections off the surface can be looked up very efficiently. Of course, the assumption of a small object compared to the environment often does not hold, but environment maps are a good compromise between rendering quality and the need to store the full 4-dimensional radiance field on the surface.

Both offline [8] and interactive, hardware-based renderers [17] have used this approach to simulate mirror reflections, often with amazing results.

In this part, we first discuss the issue of parameterizations (or representations) for environment mapping. In particular, we describe spherical maps, cube maps, and parabolic maps, all of which are supported in the most recent hardware. Following this discussion, we discuss and compare techniques for using environment maps for matte reflections and different reflection models. Finally, we will show some other examples for environment maps including applications for non-photorealistic rendering.

2 Parameterizations for Environment Maps

Since environment maps represent directional information as a 2D texture, it is necessary to decide for a mapping from directions to texture coordinates in order to define a concrete representation. This mapping, which is also called the *parameterization* of the environment map, should fulfill a couple of properties in order to be useful for hardware-accelerated rendering:

- the method for computing the texture coordinates should be simple and efficient, and it should be easy to implement in hardware. This means that complicated and expensive mathematical functions line trigonometric functions should not be necessary.
- for walkthroughs of static environments, it should not be necessary to create a new environment map every frame. This means that
  - the computation of the texture coordinates is possible for all viewing directions.
  - all light directions need to be represented equally well in the environment map. Although some light directions are more important than others for a certain viewing direction, all directions are equally important for a walkthrough, where the viewing direction is not previously known. This property is called the *uniformity* of the parameterization.
- for interaction with dynamic environments, it should be easy and inexpensive to create a new

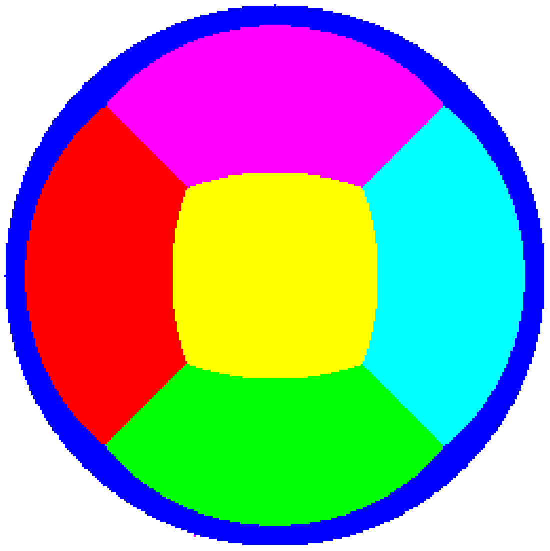
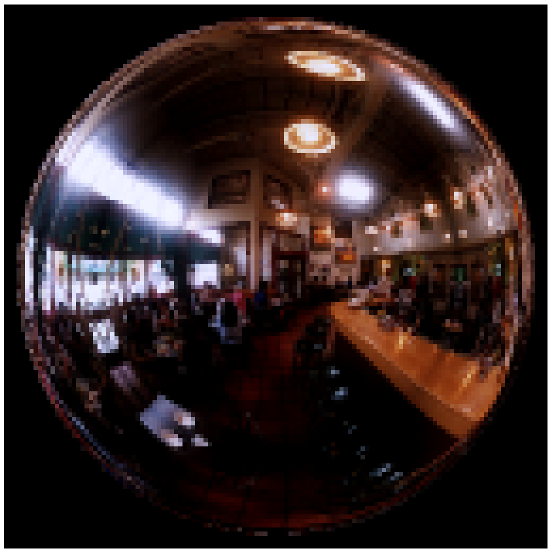

Figure 1: Left: A spherical environment map from the center of a colored cube. Note the bad sampling of the cube face directly in front of the observer (black). Right: a spherical map of a real scene.

environment map from perspective images of the scene (because this is what the hardware can generate).

In the following, we will discuss the three parameterizations for environment maps that have gained some importance in hardware rendering.

2.1 Spherical Maps

The parameterization traditionally used in computer graphics hardware is the *spherical environment map* [7]. It is based on the analogy of a small, perfectly mirroring metal ball centered around the object. The image that an orthographic camera sees when looking at such a ball from a certain viewing direction is the environment map. An example environment map from the center of a colored cube is shown on the left of Figure 1, a map of a real scene is shown on the right.

The major reason why spherical maps are used is that the lookup can be computed efficiently with simple operations in hardware (see Figure 2 for the geometry): for each vertex compute the reflection vector $\vec{r}$ of the per-vertex viewing direction $\vec{v}$. A spherical environment map which has been generated for an orthographic camera pointing into direction $\vec{v}_o$, stores the corresponding radiance information for this direction at the point where the reflective sphere has the normal $\vec{h} := (\vec{v}_o + \vec{r})/||\vec{v}_o + \vec{r}||$. If $\vec{v}_o$ is the negative z-axis in viewing coordinates, then the 2D texture coordinates are simply the x and y components of the normalized halfway vector $\vec{h}$. For environment mapping on a per-vertex basis and a reference viewing direction v_o identical to the negative z-axis in eye space, these texture coordinates are automatically computed by the texture coordinate generation mechanism of OpenGL.

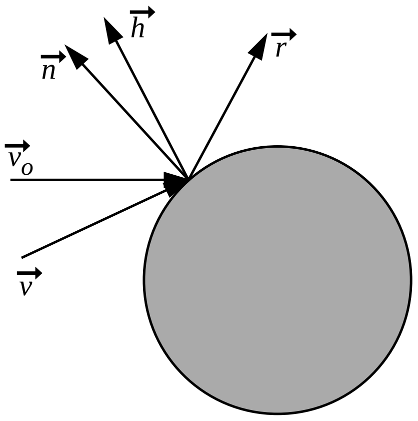

Figure 2: The lookup process in a spherical environment map.

The sampling rate of spherical maps reaches its maximum for directions opposing the viewing direction (that is, objects behind the viewer), and goes towards zero for directions close to the viewing direction, because these correspond to the tangential areas of the virtual metal ball used to generate the map. Because of this singularity in the viewing direction, it is clear that this parameterization is not suitable for viewing directions other than the original one, especially since the automatic texture coordinate mode does not support this case. Thus, maps using this parameterization have to be regenerated for each change of the view point, even if the environment is otherwise static. The creation of a spherical map requires a texture mapping step in which perspective images are warped into the spherical form.

Despite these disadvantages, the spherical map is very useful if the only interaction with the scene is rotating an object in front of the screen. This is the case, for example, in design and CAD applications, and also for some non-photorealistic rendering applications (see Section 4).

2.2 Cube Maps

The second parameterization, *cubical environment maps* or *cube maps* [6, 18] consist of six independent perspective images from the center of a cube through each of its faces. From this description it is clear that the generation of such a map simply consists of rendering the six perspective images. A warping step as required for spherical maps is not necessary. The sampling of these maps is fairly good. It can be shown that the sampling rates for all directions differ by a factor of $3\sqrt{3} \approx 5.2$.

The calculation of the texture coordinates proceeds as follows:

- compute eye-space reflection vector $\vec{r}_e$.
- transform $\vec{r}_e$ to object space, yielding $\vec{r}_o$ (cube faces are aligned with main axes in object space).
- the component of $\vec{r}_o$ with the largest absolute value and the sign of this component determine the cube face. The other two components divided by the largest one are the texture coordinates. For example, if $y = -.7$ is the largest absolute value, then the cube face at $y = -1$ is used, and $s := x/y$ and $t := z/y$ are the texture coordinates within that face.

Obviously, this parameterization is suitable for arbitrary viewing directions, and several current PC graphics boards support it via a specific OpenGL extension. One problem here is the use of six independent textures, which requires some special mechanisms in the texture specification. Also, the separation into six textures may produce seams between the cube faces. In particular, this is the case if mip-mapping is used, because then each face is downsampled individually. It would be possible to overcome these problems by adding a border of several pixels to each of the faces, and to replicate some information from neighboring faces there.

2.3 Parabolic Maps

Parabolic maps [9, 10], sometimes also called *dual paraboloid maps*, are based on an analogy similar to the one used to describe spherical environment maps. Assume that the reflecting object lies at the origin, and that the viewing direction is along the negative z axis. The image seen by an orthographic camera when looking at a metallic, reflecting paraboloid

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2), \quad x^2 + y^2 \leq 1, \qquad (1)$$

contains the information about the hemisphere facing towards the viewer. The complete environment is stored in two separate textures, each containing the information of one hemisphere. The geometry is depicted in Figure 3.

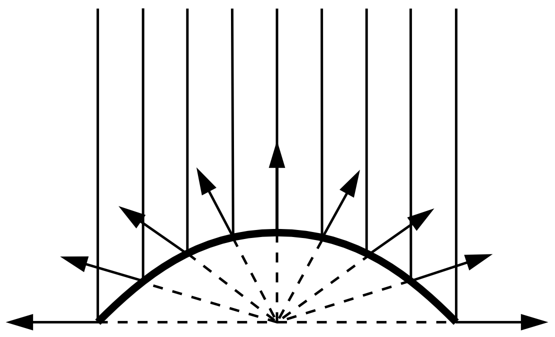

Figure 3: The rays of an orthographic camera reflected off a paraboloid sample a complete hemisphere of directions.

This parameterization has recently been introduced in [15] in a different context. Nayar actually built a lens and camera system that is capable of capturing this sort of image from the real world. Besides ray-tracing and warping of cubical environment maps, this is actually one way of acquiring maps in the proposed format. Since two of these cameras can be attached back to back, it is possible to create full $360°$ images of real world scenes.

The geometry described above has some interesting properties. Firstly, the reflected rays in each point of the paraboloid all originate from a single point, the focal point of the paraboloid, which is also the origin of the coordinate system (see dashed lines in Figure 3). This means that the resulting image can indeed be used as an environment map for an object in the origin. Spherical environment maps do not have this property; the metal spheres used there have to be assumed small.

Secondly, the sampling rate of a parabolic map varies by a factor of 4 over the complete image. This can be shown easily through the following considerations. Firstly, a point on the paraboloid is given as $\vec{f} = (x, y, \frac{1}{2} - \frac{1}{2}(x^2 + y^2))^T$. This is also the vector from the origin to the point on the paraboloid, and the reflection vector $\vec{r}$ for a ray arriving in this point from the viewing direction (see Figure 3). If the cross section of the viewing ray has a differential area dA (which corresponds to the pixel area), then the reflected ray also has this area. The solid angle covered by the pixel is thus the projection of dA onto the unit sphere:

$$\omega(x, y) = \frac{dA}{||\vec{f}(x, y)||^2} \cdot sr. \tag{2}$$

Since all pixels have the same area dA, and $\vec{f}(0, 0) = 1/2$, the sampling rate for $x = 0$ and $y = 0$, that is, for reflection rays $\vec{r} = (0, 0, 1)^T$ pointing back into the direction of the viewer, is $\omega_r = 4sr/m^2 \cdot dA$. Thus, the change in sampling rate over the hemisphere can be expressed as

$$\begin{aligned} \frac{\omega(x, y)}{\omega_r} =& \frac{1}{4||\vec{f}(x, y)||^2} \\ =& \frac{1}{4(x^2 + y^2 + (\frac{1}{2} - \frac{1}{2}(x^2 + y^2))^2)} \end{aligned} \tag{3}$$

Pixels in the outer regions of the map (i.e. with $x^2 + y^2 = 1$) cover only $1/4$ of the solid angle covered by center pixels. This means that directions perpendicular to the viewing direction are sampled at a higher rate than directions parallel to the viewing direction. Depending on how we select mip-map levels, the factor of 4 in the sampling rate corresponds to one or two levels difference, which is quite acceptable. In particular this is somewhat better than the sampling of cubical environment maps. The sampling rates for different parameterizations are compared in Figure 4.

Figure 5 shows the two images comprising a parabolic environment map for the simple scene used in Figure 1. The left image represents the hemisphere facing towards the camera, while the right image represents the hemisphere facing away from it.

2.3.1 Lookups from Arbitrary Viewing Positions

In the following, we describe the math behind the lookup process of a reflection value for an arbitrary viewing position and -direction. We assume that environment maps are specified relative to a coordinate system in which the reflecting object lies at the origin, and the map is generated for a viewing direction (i.e. vector from the object point to the eye point) of $\vec{d}_o = (0, 0, 1)^T$. It is not necessary that this coordinate system represents the object space of the reflecting object, although this would be an obvious choice. However, it is important that the transformation between this space and eye space is a rigid body transformation, as this means that vectors do not have to be normalized after transformation. To simplify the notation, we will in the following use the term “object space” for this space.

In the following, $\vec{v}_e$ denotes the normalized vector from the point on the surface to the eye point in eye space, while the vector $\vec{n}_e = (n_{e,x}, n_{e,y}, n_{e,z})^T$ is the normal of the surface point in eye space. Furthermore, the (affine) model/view matrix is given as $\mathbf{M}$. This means, that the normal in eye space $\vec{n}_e$ is really the transformation $\mathbf{M}^{-T} \cdot \vec{n}_o$ of some normal vector in object space. If $\mathbf{M}$ is a rigid body transformation, and $\vec{n}_o$ is normalized, then so is $\vec{n}_e$. The reflection vector in eye space is then given as

$$\vec{r}_e = 2 < \vec{n}_e, \vec{v}_e > \vec{n}_e - \vec{v}_e. \tag{4}$$

Transforming this vector with the inverse of $\mathbf{M}$ yields the reflection vector in object space:

$$\vec{r}_o = \mathbf{M}^{-1} \cdot \vec{r}_e. \tag{5}$$

The illumination for this vector in object space is stored somewhere in one of the two images. More specifically, if the z component of this vector is positive, the vector is facing towards the viewer, and thus the value is in the first texture image, otherwise it can be found in the second. Let us, for the moment, consider the first case.

$\vec{r}_o$ is the reflection of the constant vector $\vec{d}_o = (0, 0, 1)^T$ at some point (x, y, z) on the paraboloid:

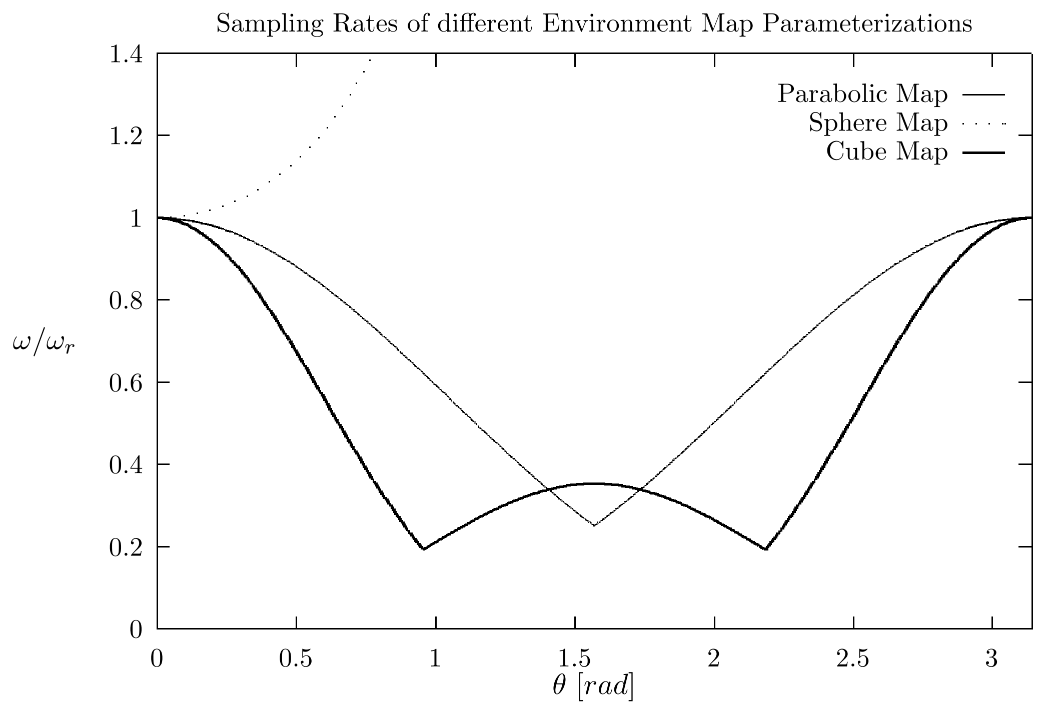

Figure 4: The change of solid angle ω/ω_r covered by a single pixel versus the angle θ between the negative z-axis and the reflected ray. ω_r is the solid angle covered by pixels showing objects directly behind the viewer. The parabolic parameterization proposed here gives the most uniform sampling.

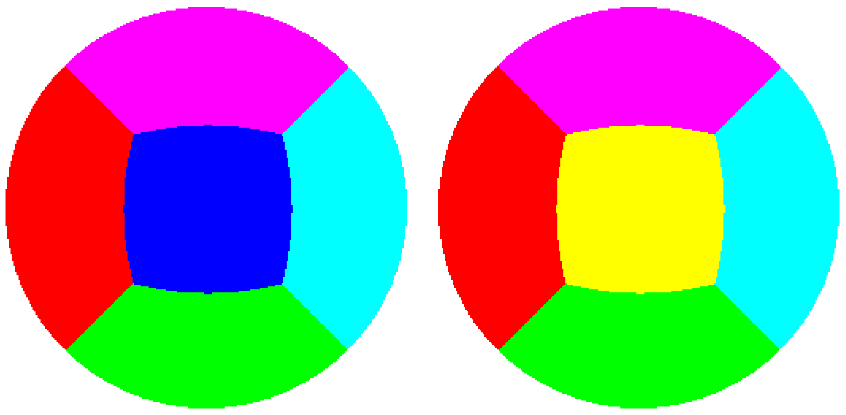

Figure 5: The two textures comprising an environment map for an object in the center of a colored cube.

$$\vec{r}_o = 2 < \vec{n}, \vec{d}_o > \vec{n} - \vec{d}_o, \tag{6}$$

where $\vec{n}$ is the normal at that point of the paraboloid. Due to the formula of the paraboloid from Equation 1, this normal happens to be

$$\vec{n} = \frac{1}{\sqrt{x^2+y^2+1}} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}. \tag{7}$$

The simplicity of this formula is the major reason that the parabolic parameterization can be easily implemented in hardware. It means that an unambiguous representative for the normal direction can be computed by dividing a (not necessarily normalized) normal vector by its z component, which can be implemented as a perspective division. Another way to disambiguate the normal direction would be to normalize the vector, which involves the more expensive computation of an inverse square root. This is the approach taken by spherical maps. The combination of Equations 6 and 7 yields

$$\vec{d_o} + \vec{r_o} = 2 < \vec{n}, \vec{v} > \vec{n} = \left(\begin{array}{c} k \cdot x \\ k \cdot y \\ k \end{array} \right). \qquad (8)$$

for some value k.

In summary, this means that x and y, which can be directly mapped to texture coordinates, can be computed by calculating the reflection vector in eye space (Equation 4), transforming it back into object space (Equation 5), adding it to the (constant) vector $\vec{d_o}$ (Equation 8), and finally dividing by the z component of the resulting vector.

The second case, where the z component of the reflection vector in object space is negative, can be handled similarly, except that $-\vec{d}$ has to be used in Equation 8, and that the resulting values are $-x$ and $-y$.

2.3.2 Implementation Using Graphics Hardware

An interesting observation in the above equations is that almost all the required operations are linear. There are two exceptions. The first is the calculation of the reflection vector in eye space (Equation 4), which is quadratic in the components of the normal vector $\vec{n}_e$. The second exception is the division at the end, which can, however, be implemented as a perspective divide.

Given the reflection vector $\vec{r}_e$ in eye coordinates, the transformations for the frontfacing part of the environment can be written in homogeneous coordinates as follows:

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = \mathbf{P} \cdot \mathbf{S} \cdot (\mathbf{M}_l)^{-1} \cdot \begin{bmatrix} r_{e,x} \\ r_{e,y} \\ r_{e,z} \\ 1 \end{bmatrix}, \qquad (9)$$

where

$$\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad (10)$$

is a projective transformation that divides by the z component,

$$\mathbf{S} = \begin{bmatrix} -1 & 0 & 0 & d_{o,x} \\ 0 & -1 & 0 & d_{o,y} \\ 0 & 0 & -1 & d_{o,z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (11)$$

computes $\vec{d_o} - \vec{r_o}$, and $\mathbf{M}_l$ is the linear part of the affine transformation $\mathbf{M}$. Another matrix is required for mapping x and y into the interval $[0,1]$ for the use as texture coordinates:

$$\begin{bmatrix} s \\ t \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} \qquad (12)$$

Similar transformations can be derived for the backfacing parts of the environment. These matrices can be used as texture matrices, if $\vec{r}_e$ is specified as the initial texture coordinate for the vertex. Note that $\vec{r}_e$ changes from vertex to vertex, while the matrices remain constant.

Due to non-linearity, the reflection vector $\vec{r}_e$ either has to be computed in software, or using a special mode for automatic texture coordinate generation. This mode is currently available on almost all contemporary PC boards.

What remains to be done is to combine frontfacing and backfacing regions of the environment into a single image. Using multiple textures, this can be achieved in a single rendering pass. To this end, the backfacing part of the environment map is specified as an RGB texture with the appropriate matrix for the backfacing hemisphere. Then, the frontfacing part is specified as a second texture in RGBA format. The alpha is used to mark pixels inside the circle $x^2 + y^2 \leq 1$ with an alpha value of 1, pixels outside the circle with an alpha value of 0. The blending between the two maps is set up in such a way that the colors of the two textures are blended together using the alpha channel of the frontfacing map.

The important point here is that backfacing vectors $\vec{r}_o$ will result in texture coordinates $x^2 + y^2 > 1$ while the matrix for the frontfacing part is active. These regions fall outside the circular region marked in the frontfacing map, and are thus covered by the backfacing environment map.

Alpha values between 0 and 1 can be used to blend between the two maps if seams are visible due to inconsistent data for the two hemispheres (e.g. if the two hemispheres are recovered from photographs). This would require the pixels outside the circle $x^2 + y^2 \leq 1$ to be valid as in Figure 6. These maps have been generated by extending the domain of the paraboloid (Equation 1) to $[-1, 1]^2$.

If the hardware does not support multiple simultaneous textures, the parameterization can still be applied using a multi-pass method and alpha testing. As above, the pixels of the frontfacing map are marked in the alpha channel. In pseudo-code, the algorithm then works as follows:

```
glAlphaFunc( GL_EQUAL, 1.0 );
glEnable( GL_ALPHA_TEST );
glMatrixMode( GL_TEXTURE );

glBindTexture( GL_TEXTURE_2D,
               frontFacingMap );
glLoadMatrix( frontFacingMatrix );
draw object with $\vec{r}_o$
        as texture coordinate

glBindTexture( GL_TEXTURE_2D,
               backFacingMap );
glLoadMatrix( backFacingMatrix );
draw object with $\vec{r}_o$
        as texture coordinate
```

Figure 7 shows a reflective sphere and torus viewed from different angles with the environment maps from Figure 6.

2.4 Mip-map Level Generation

Anti-aliasing is of particular importance for environment maps, since, depending on the surface geometry, reflections can occur both magnified and minified, and therefore have a large range of possible scales.

In hardware, anti-aliasing of textures is done with mip-mapping [19], or, more recently, with anisotropic filtering methods like footprint assembly [16]. If available with the specific hardware in use, anisotropic filtering is highly recommendable, because most of the time some parts of objects will be seen at grazing angles.

Both for mip-mapping and footprint assembly, it is necessary to compute a hierarchy of texture maps at different resolutions. For normal texture mapping, these are most often generated by simply averaging a 2×2 block of pixels to obtain one pixel value for the next level. For environment maps, however, this is not the right way. Rather than that, each pixel should be weighted by the solid angle it covers to account for the non-uniformity of the used environment map parameterization (e.g. Equation 2 for parabolic maps).

Furthermore, in order to avoid seams for representations that use multiple 2D textures for one environment map, there should be a border that is several pixels wide for each of the textures. This border should replicate information from the other textures in the environment map, so that the mip-mapping uses cross-texture information.

3 Complex Reflection Models and Environment Map Prefiltering

Once an environment map is available, it can be used to add a mirror reflection term to an object. Using multi-pass rendering and alpha blending, this mirror reflection term can be added to local illumination terms that are generated using hardware lighting. In order to incorporate global illumination with other reflection models than a perfect metallic mirror, we need to perform some precomputations, since real-time calculations are typically not possible due to the high computational cost. The two fundamental techniques for using environment maps with more general reflection models are

- Decomposition. The reflection model is decomposed into simpler contributions, which can be treated separately. For example, a reflection model may be separated into a diffuse and a specular term, where the specular term is additionally multiplied with an angular dependent term (Fresnel term).

- Prefiltering. For certain reflection models, the reflection of an environment map can be analytically precomputed and stored into a new map. The latter

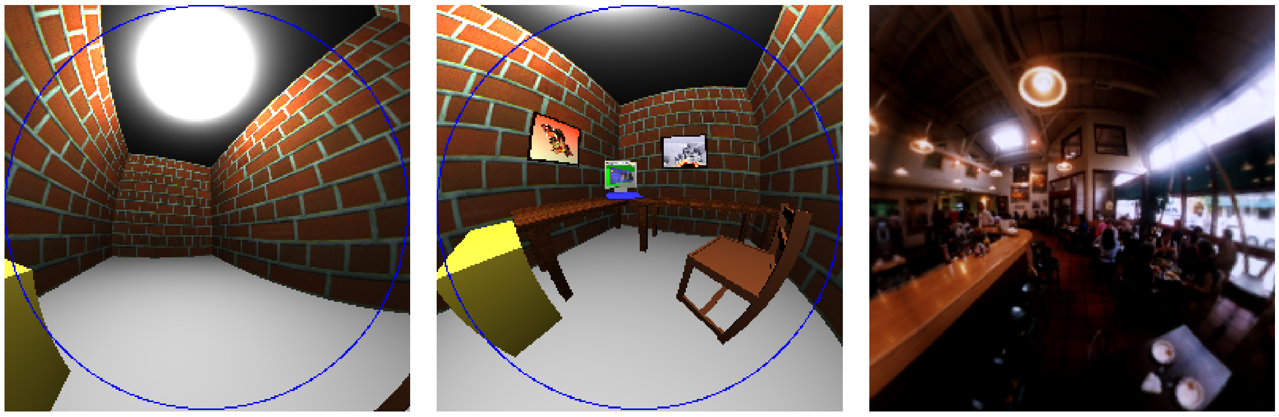
Figure 6: Left/Center: two textures comprising the environment map for an object in the center of an office scene. Right: A resampled image from [7]. The original image was taken with a 180° fisheye lens showing one hemisphere. Then a cubical map was generated by replicating the image for the second hemisphere. From this cubical map, we resampled the map for our parameterization. The map for the second hemisphere would be identical to this one.

is called *prefiltered environment map* or *reflection map*.

In the following we will describe these two techniques and demonstrate some applications for them.

3.1 Decomposition

As stated above, decomposition of a reflection model means separating its terms into simpler expressions that can be handled individually. The most fundamental example is a separation into diffuse and specular contributions. We will show in Section 3.2 that the diffuse term as well as certain specular terms can be treated with prefiltering. Another term could be if we had a reflection model with a term for retro-reflection (light that is reflected back into the direction of incoming light). Also a very interesting example is the factorization of the specular component into a standard environment map and an angular dependent term (Fresnel term), as described in the following.

3.1.1 Generalized Mirror Reflections using a Fresnel Term

The Fresnel term is a physical term describing the reflectivity of a material depending on its optical density n ("index of refraction") and the angle of incoming light. It is given as

$$F = \frac{(g-c)^2}{2(g+c)^2}\left[1 + \frac{(c(g+c)-1)^2}{(c(g-c)+1)^2}\right], \qquad (13)$$

with $c =< \vec{n}, \vec{v} >$ and $g^2 = n^2 + c^2 - 1$.

A regular environment map without prefiltering describes the incoming illumination at a point in space. If this information is directly used as the outgoing illumination, as is described above, and as it is state of the art for interactive applications, only metallic surfaces can be modeled. This is because for metallic surfaces (surfaces with a high optical density) the Fresnel term is almost constant one, independent of the angle between light direction and surface normal. Thus, for a perfectly smooth (i.e. mirroring) surface, incoming light is reflected in the mirror direction with a constant reflectance.

For non-metallic materials (materials with a small optical density), however, the reflectance strongly depends on the angle of the incoming light. Mirror reflections on these materials should be weighted by the Fresnel term for the angle between the normal and the reflected viewing direction $\vec{r}_v$, which is, of course, the same as the angle between normal and viewing direction $\vec{v}$.

For any given material, the Fresnel term $F(\cos\theta)$ for the mirror direction $\vec{r}_v$ can be stored in a 1-dimensional texture map, and rendered to the framebuffer's alpha channel in a separate rendering pass. The mirror part is then multiplied with this Fresnel term in a second pass, and a third pass is used to add the diffuse part. If we have a reflection model consisting of a mirror component L_m and a diffuse component L_d, this yields an outgoing radiance of $L_o = F \cdot L_m + L_d$.

In addition to simply adding the diffuse part to the

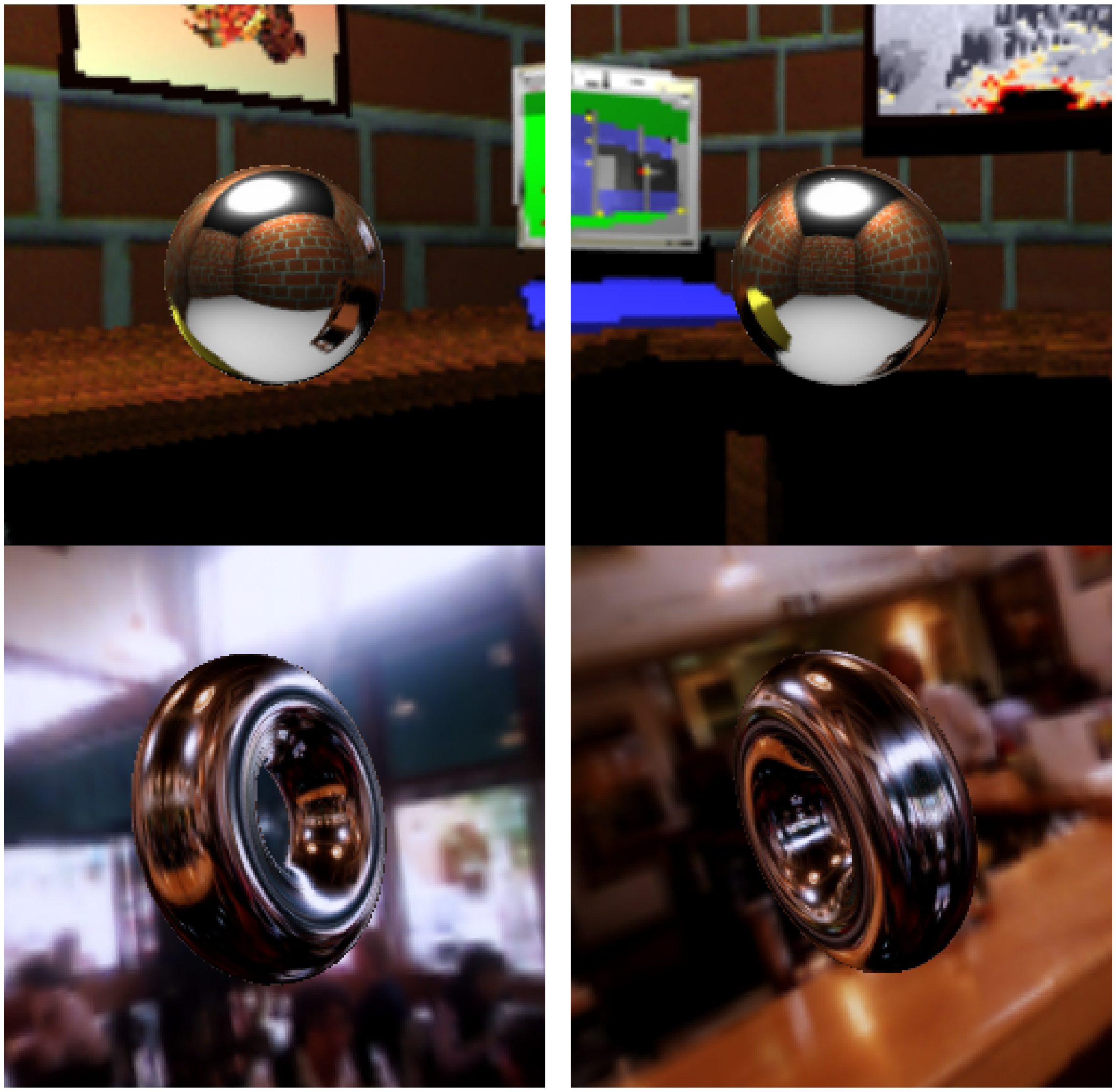

Figure 7: The environment maps from Figure 6 applied to a sphere and a torus, and seen from different viewpoints.

Fresnel-weighted mirror reflection, we can also use the Fresnel term for blending between diffuse and specular: $L_o = F \cdot L_m + (1 - F)L_d$. This allows us to simulate diffuse surfaces with a transparent coating: the mirror term describes the reflection off the coating. Only light not reflected by the coating hits the underlying surface and is there reflected diffusely.

Figure 8 shows images generated using these two approaches. In the top row, the Fresnel-weighted mirror term is shown for indices of refraction of 1.5, 5, and 200. In the center row, a diffuse term is added, and in the bottom row, mirror and diffuse terms are blended using the Fresnel term. Note that for low indices of refraction, the object is only specular for grazing viewing angles, while for a high indices of refraction we get the metal-like reflection known from Figure 7.

3.2 Prefiltered Environment Maps

Generally speaking, prefiltered environment maps capture all the reflected exitant radiance towards all directions $\vec{v}$ from a fixed position $\mathbf{x}$:

$$L_o(\mathbf{x};\vec{v},\vec{n},\vec{t}) = \int_\Omega f_r(\vec{w}(\vec{v},\vec{n},\vec{t}),\vec{w}(\vec{l},\vec{n},\vec{t}))L_i(\mathbf{x};\vec{l}) < \vec{n},\vec{l} > \; d\vec{l}, \tag{14}$$

where $\vec{v}$ is the viewing direction and $\vec{l}$ is the light direction in world-space, $\{\vec{n},\vec{t},\vec{n}\times\vec{t}\}$ is the local coordinate frame of the reflective surface, $\vec{w}(\vec{v},\vec{n},\vec{t})$ represents the

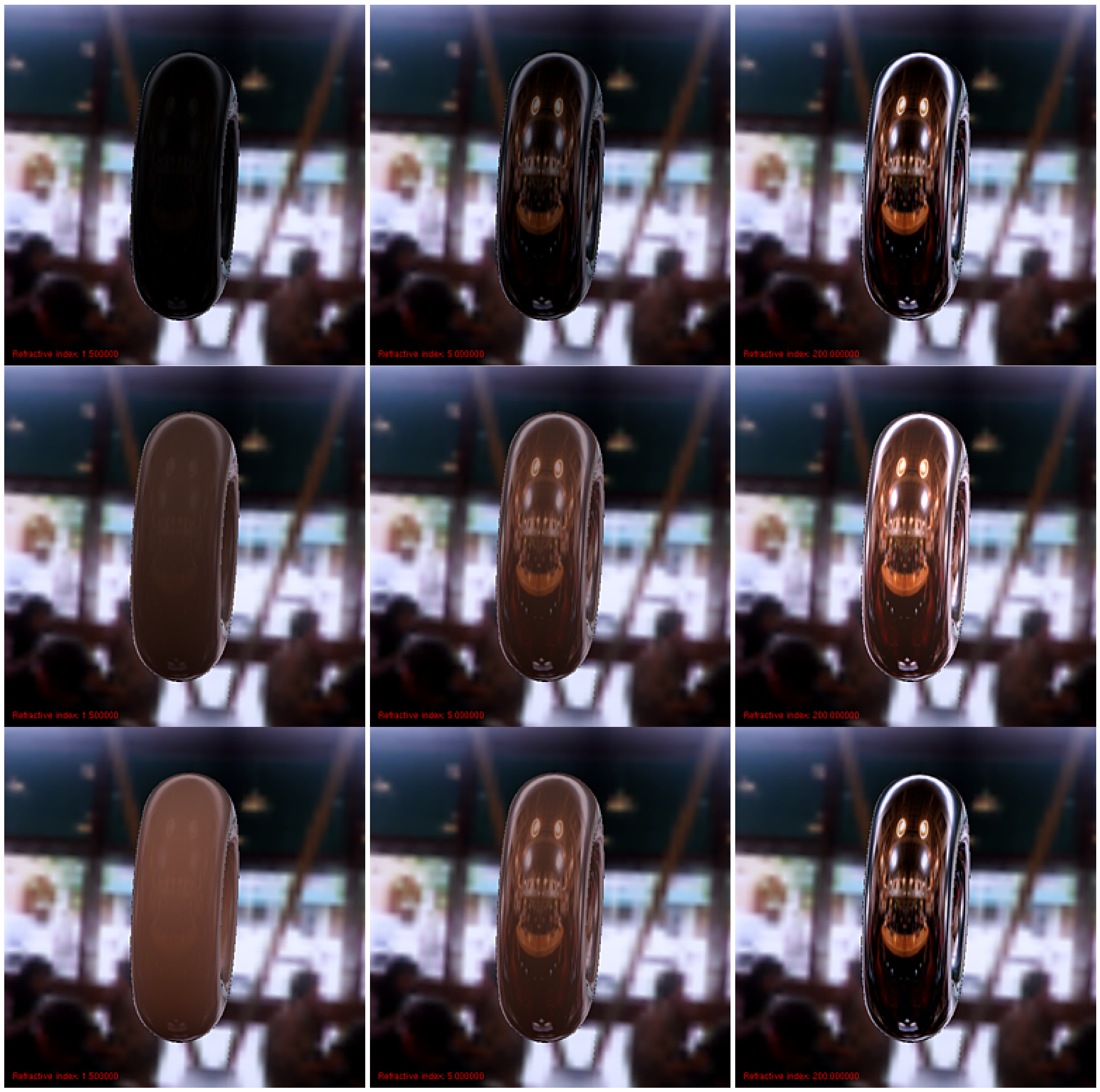
Refractive index: 1.500000
Refractive index: 5.000000
Refractive index: 200.000000
Refractive index: 1.500000
Refractive index: 5.000000
Refractive index: 200.000000
Refractive index: 1.500000
Refractive index: 5.000000
Refractive index: 200.000000

Figure 8: Top row: Fresnel weighted mirror term. Center row: Fresnel weighted mirror term plus diffuse illumination. Bottom row: Fresnel blending between mirror and diffuse term. The indices of refraction are (from left to right) 1.5, 5, and 200.

viewing direction and $\vec{w}(\vec{l}, \vec{n}, \vec{t})$ the light direction relative to that frame, f_r is the BRDF, which is usually parameterized via a local viewing and light direction.

A prefiltered environment map stores the radiance of light reflected towards the viewing direction $\vec{v}$, which is computed by weighting the incoming light L_i from all directions $\vec{l}$ with the BRDF f_r. Note, that L_i is stored in the unfiltered original environment map. As you can see, in the general case we have a dependence on the viewing direction as well as on the orientation of the reflective surface, i.e. the local coordinate frame $\{\vec{n}, \vec{t}, \vec{n} \times \vec{t}\}$.

This general kind of environment map is five-dimensional. Two dimensions are needed to represent the viewing direction $\vec{v}$ (a unit vector in world coordinates) and three dimensions are necessary to represent the coordinate frame $\{\vec{n}, \vec{t}, \vec{n} \times \vec{t}\}$; e.g. three angles can be used to specify the orientation of an arbitrary coordinate frame.

Of course, five-dimensional textures have enormous memory requirements, which is why the prefiltered environment maps which we will examine drop some dependencies (e.g. the tangent $\vec{t}$) and are often reparameterized (e.g. indexing is not done with the viewing direction $\vec{v}$, but the reflected viewing direction). Because this reduction in dimensionality also removes some to the generality of the approach, the decomposition method is often required to combine several of these simplified models.

If the original environment map is given in a high-dynamic range format such as [4], then the prefiltering technique allows for effects similar to the ones described in [3].

3.2.1 Diffusely Prefiltered Maps

As we have seen, we can combine a mirror reflection term using an environment map with local illumination terms that are generated using hardware lighting. It is also possible to add a diffuse global illumination term through the use of a precomputed texture. For the generation of such a texture, there are two methods. In the first approach, a global illumination algorithm such as Radiosity is used to compute the diffuse global illumination of every surface point.

The second approach is purely image-based, and uses a prefiltered environment map [14, 6]. The environment map used for the mirror term contains information about the incoming radiance $L_i(\mathbf{x}, \vec{l})$, where $\mathbf{x}$ is the point for which the environment map is valid, and $\vec{l}$ the direction of the incoming light. The outgoing radiance for a diffuse BRDF is then:

$$L_o(\mathbf{x}, \vec{n}) = k_d \cdot \int_{\Omega(\vec{n})} L_i(\mathbf{x}, \vec{l}) \cdot \cos(\vec{n}, \vec{l}) d\omega(\vec{l}). \quad (15)$$

Due to the constant BRDF of diffuse surfaces, L_o is only a function of the surface normal $\vec{n}$ and the illumination L_i stored in the environment map, but not of the outgoing direction $\vec{v}$. Thus, it is possible to precompute a map containing the diffuse illumination for all possible surface normals. For this map, like for the mirror map, any parameterization from Section 2 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. Figure 9 shows such a prefiltered map, a torus with diffuse illumination only as well as a torus with diffuse and mirror illumination.

3.2.2 Glossy Prefiltering of Environment Maps

A simplification similar to the one used for diffuse materials is also possible for certain specular reflection models [10, 14], most notably the Phong model. Voorhies et al. [18] used a similar approach to implement Phong shading for directional light sources.

As shown in [13], the Phong BRDF is given by

$$f_r(\mathbf{x}, \vec{l} \rightarrow \vec{v}) = k_s \cdot \frac{<\vec{r}_l, \vec{v}>^{1/r}}{\cos\alpha} = k_s \cdot \frac{<\vec{r}_v, \vec{l}>^{1/r}}{\cos\alpha}, \quad (16)$$

where $\vec{r}_l$, and $\vec{r}_v$ are the reflected light- and viewing directions, respectively, and $\cos\alpha =<\vec{n}, \vec{l}>$. Thus, the specular global illumination using the Phong model is

$$L_o(\mathbf{x}, \vec{r}_v) = k_s \cdot \int_{\Omega(\vec{n})} <\vec{r}_v, \vec{l}>^{1/r} L_i(\mathbf{x}, \vec{l})\, d\omega(\vec{l}), \quad (17)$$

for some roughness value r. This is only a function of the reflection vector $\vec{r}_v$ and the environment map containing the incoming radiance $L_i(\mathbf{x}, \vec{l})$. As for diffuse illumination, it is therefore possible to take a map containing $L_i(\mathbf{x}, \vec{l})$, and generate a filtered map containing the outgoing radiance $L_o(\mathbf{x}, \vec{r}_v)$ for a glossy Phong material.

Figure 10 shows such a map generated from the original cafe environment in Figure 6, as well as a glossy sphere and torus textured with this map.

A Fresnel weighting of these prefiltered environment maps along the lines of Section 3.1.1 is only possible with approximations. The exact Fresnel term for the glossy reflection cannot be used, since this term would have to appear inside the integral of Equation 17. However, for glossy surfaces with a low roughness, the Fresnel term can be assumed constant over the whole specular peak (which is very narrow in this case). Then the Fresnel term can be moved out of the integral, and the same technique as for mirror reflections applies.

The use of a Phong model for the prefiltering is somewhat unsatisfactory, since this is not a physically valid model. However, this method works for all reflection models having lobes that are rotationally symmetric about the reflected viewing direction, and whose shape does not depend on the angle to the surface normal.

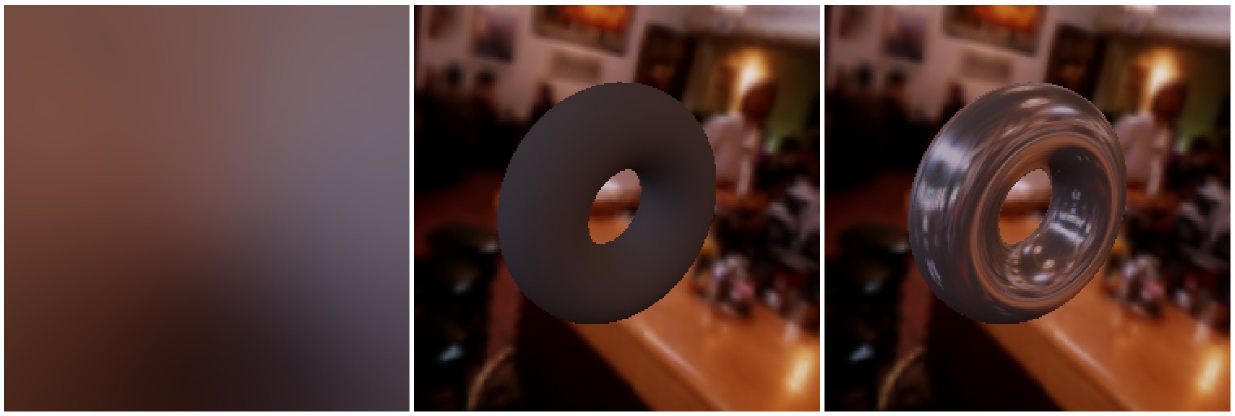

Figure 9: Left: diffusely prefiltered environment map of the cafe scene. Center: diffusely illuminated torus. Right: same torus illuminated with both a diffuse and a mirror term.

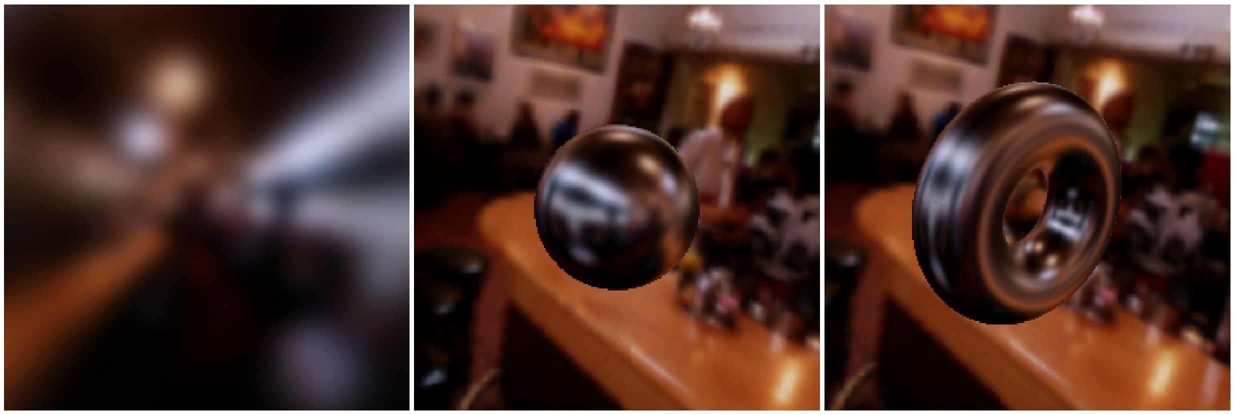

Figure 10: A prefiltered version of the map with a roughness of 0.01, and application of this map to a reflective sphere and torus.

3.2.3 Approximations of General Isotropic BRDFs

Based on this concept, Kautz and McCool [11] extended the Phong environment maps idea to other isotropic BRDFs by approximating them with a special class of BRDFs:

$$f_r(\vec{v},\vec{l}) \quad := \quad p(<\vec{n},\vec{r}_v(\vec{n})>,<\vec{r}_v(\vec{n}),\vec{l}>),$$

where p is an approximation to a given isotropic BRDF, which is not only isotropic, but also radially symmetric about $\vec{r}_v(\vec{n}) = 2(\vec{n}\cdot\vec{v})\vec{n}-\vec{v}$, and therefore only depends on two parameters.

Now consider Equation 14 using this reflectance function:

$$L_o(\mathbf{x};\vec{v},\vec{n},\vec{t}) = \int_{\Omega(\vec{n})} p(<\vec{n},\vec{r}_v(\vec{n})>,<\vec{r}_v(\vec{n}),\vec{l}>) \cdot L_i(\mathbf{x};\vec{l}) <\vec{n},\vec{l}> \; d\omega(\vec{l}). \tag{18}$$

The authors then make the assumption that the used BRDF is fairly specular, i.e. the BRDF close to zero almost everywhere, except for $\vec{r}_v(\vec{n}) \approx \vec{l}$. Using this assumption they reason that $<\vec{n},\vec{r}_v(\vec{n})> \approx <\vec{n},\vec{l}>$. Now the equation can be reparameterized and rewritten the following way:

$$L_o(\mathbf{x};\vec{r}_v,<\vec{n},\vec{r}_v>) = <\vec{n},\vec{r}_v> \int_{\Omega(\vec{n})} p(<\vec{n},\vec{r}_v>,<\vec{r}_v,\vec{l}>) \cdot L_i(\mathbf{x};\vec{l}) \, d\omega(\vec{l}), \tag{19}$$

which is three dimensional. The third dimension is used to vary the diameter of the lobe with the angle between reflection vector and surface normal. This way, it is possible to have materials that are almost mirror-like at grazing viewing angles, while they are matte if looked at perpendicularly. This is a behavior that can be seen quite often with real materials.

In addition to this, Kautz and McCool also proposed an approximation technique that generates a BRDF with rotationally symmetric lobes from an arbitrary BRDF. This is done by averaging the lobes for different viewing directions.

This technique has the advantage that it can use approximations of arbitrary isotropic BRDFs and achieves interactive frame rates. Off-specular peaks can also be incorporated into this technique. An additional Fresnel factor like Miller [14] and Heidrich [10] proposed is not needed because it can be incorporated into the dependency on the viewing angle, i.e. the third dimension of the map. On the down side, 3D textures are quite space consuming and are not supported by most current low-end hardware.

Depending on the BRDF, the quality of the approximation varies. For higher quality approximations Kautz and McCool also propose to use a multilobe approximation, which basically results in several prefiltered environment maps which have to be added.

For instance, if a BRDF is to be used, which is based on several separate surface phenomena (e.g. has retro-reflections, diffuse reflections, and glossy reflections) each part has to be approximated separately, since no radially symmetric approximation can be found for the whole BRDF. This again means a decomposition of the reflection model into several parts.

3.2.4 Warping for Environment Maps with Isotropic BRDFs

A different technique which makes similar assumptions (isotropic and radially symmetric BRDF) was presented by Cabral et al. [2]. They prefilter an environment map for different fixed viewing directions, resulting in view-dependent, spherical environment maps. An alternative to the prefiltering process is to take photographs from different viewing directions of a sphere made of the same material one would like to represent.

In contrast to the previous approach, this is actually a four-dimensional environment map

$$L_o(\mathbf{x};\vec{v},\vec{n}) = \int_{\Omega(\vec{n})} p(<\vec{n},\vec{r}_v>,<\vec{r}_v,\vec{l}>) \cdot L_i(\mathbf{x};\vec{l}) <\vec{n},\vec{l}> \; d\omega(\vec{l}), \tag{20}$$

but the two dimensions representing the viewing direction $\vec{v}$ are only sampled very coarsely. A different two dimensional spherical map is extracted from this four-dimensional map for every new viewpoint. This map corresponds to one specific viewing direction and is generated using warping. The new view-dependent environment map is then applied to an object. The warping compensates for the undersampled viewing directions, and minimizes the visible artifacts. Although the warping requires high-end graphics hardware to achieve interactive frame rates, the final rendering can be done with standard sphere mapping, which is major the reason for generating the intermediate spherical map.

Warping is done based on an assumption what the central reflection direction of the BRDF is (the reflected viewing direction and the surface normal are mentioned as examples in [2]). For example, if a specular highlight is assumed, then the warping is performed such that the location of the highlights are located in the same position after warping to the destination direction. The assumption of a single, predominant reflection direction fails for BRDFs that have off-specular reflections like strong diffuse components or retro-reflection. Similarly, since radially symmetric BRDFs are used, this method has the same difficulties with complex BRDFs as the previous method. To overcome these problems, the method can be combined with a decomposition approach.

As mentioned before the generated two dimensional environment map is view-dependent, so the reflective object needs to be viewed with an orthographic projection or otherwise the reflections are incorrect, since the reflection directions are computed based on an infinite viewer. For example, if the material contains a strongly varying Fresnel term, it cannot be represented in this form, because the spherical map does not depend on the angle between normal and light direction.

3.2.5 Hardware Accelerated Prefiltering

For interactive applications it would be nice if environment map prefiltering could be done on the fly.

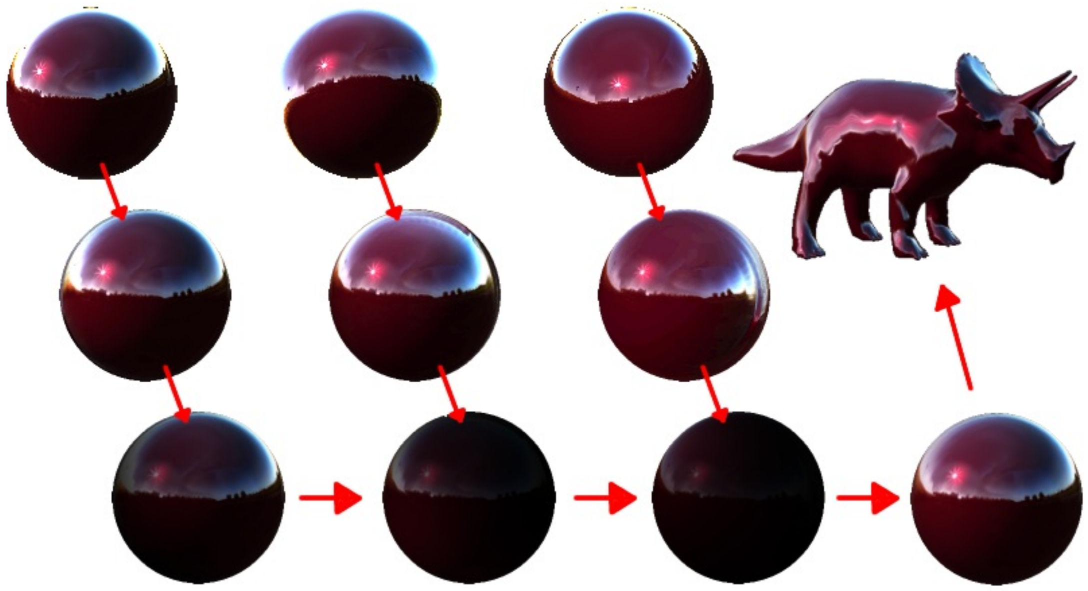

Figure 11: Warping of the environment maps is done by taking three source maps corresponding to close-by viewing directions (top row). Each source map is then warped to the desired viewing direction by assuming a mirror reflection for the feature correspondences (i.e. matching the highlights, center row). Finally, all warped maps are weighted by considering how similar the viewing direction of the corresponding source map is to the desired viewing direction (bottom row). After summing up the contributions, a new spherical map is obtained that can be applied to an object. Figure courtesy of Marc Olano.

This means that if the scene changes, glossy reflections change accordingly. Here, we will describe a method to perform hardware-accelerated Phong filtering [12] of a given environment map similar to Section 3.2.2.

In a prefiltered environment map, every texel is a weighted sum of all pixels in a source environment map. This means, we can think of the filtering process as applying a (BRDF-dependent) filter kernel to some unfiltered source map. We would like to map this filtering operation to the operations provided by a graphics hardware pipeline. The OpenGL imaging subset only supports shift-invariant two dimensional filters of certain sizes, and we would like to use this feature to perform the filtering. Hence, for hardware accelerated prefiltering we have to choose an environment map technique that uses only two dimensional environment maps with a BRDF which results in a shift-invariant filter over the hemisphere, and an environment map representation that keeps the filter shift-invariant.

The Phong model has a shift-invariant filter kernel over the hemisphere, since its cosine lobe is constant for all reflected viewing directions $\vec{r}_v$. It is also radially symmetric about $\vec{r}_v$. The filter size can also be decreased if smaller values are clamped to zero (this will be necessary due to the restricted filter size of the graphics hardware). The filter shape is obviously circular, since it is radially symmetric. Therefore Phong environment maps fulfill the necessary requirements for hardware accelerated prefiltering. We still need to find an environment map representation that maps the shift-invariant circular filter kernel on the hemisphere to a shift-invariant circular filter kernel in texture space.

It turns out that the parabolic maps come close to this desired property. A circular filter kernel which is mapped from the parabolic environment map back to the hemisphere is also (almost) circular. A distortion occurs depending on the radius and the position of the filter. To visualize the distortion, we project a circular filter kernel with a radius of r ($r = 1$ is half the width of the parabolic map) from the parabolic map back to the sphere and measure the error; see right side of Figure 12. The resulting distortion of the circle is given on the left side of the same figure. It depends on the radius of the filter kernel and also on the distance d of the filter's center from the parabolic map's center (i.e. the center of the front- or backfacing paraboloid). The distortion goes up to 25% for large radii, but in these cases the prefiltered environment map will be very blurry, so

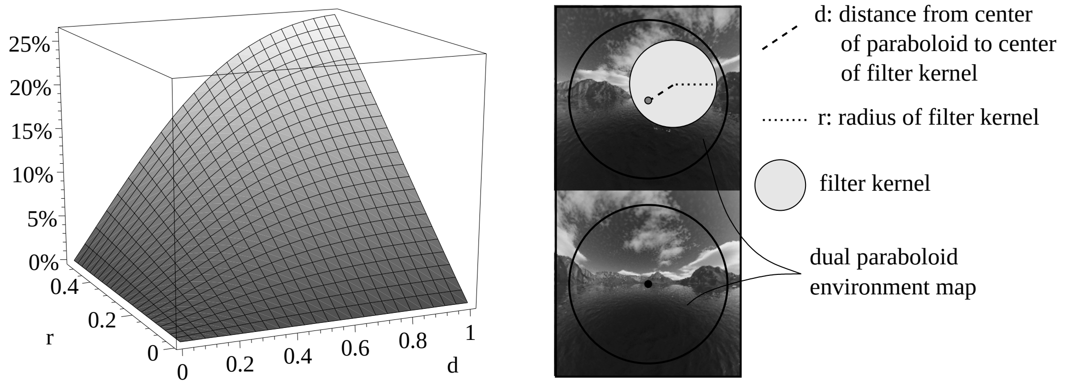

Figure 12: Distortion of a circle when projected from a paraboloid map back to the sphere.

that the distortion will not lead to visible errors. For smaller radii the distortion remains fairly small.

Unfortunately a shift-invariant filter kernel on the sphere does still not completely map to a shift-invariant filter in the parabolic space. Besides the slight distortion, the size of the filter kernel varies with the distance d to the center of the map. The ratio between the smallest filter radius and largest filter radius is $1 : 2$, since the ratio for the areas is 1:4, as shown in Section 2.3. To adjust for this, we generate two prefiltered environment maps, one with the smallest (yields map *S*) and one with the largest necessary filter size (yields map *L*). Then we blend between both prefiltered environments. The value with which we need to blend between both maps is different for different pixels in the parabolic environment map, but it depends only on the distance d and is always d^2. For a pixel in the center of the paraboloid this means that we use 0% of map *L* and a 100% of map *S*; for a pixel with distance $d = 0.5$ to the center of the parabolic map, we use 25% of map *L* and 75% of map *S*, and so on.

The actual algorithm is fairly simple. First we create a mipmap of the parabolic environment map, then we load the environment map (plus the mipmap) into texture memory. The user has to specify the Phong exponent to be used and a limit when BRDF values from the Phong model can be clamped to zero, which is used to restrict the kernel size in the first place. Then we compute the two necessary filter radii, r_s for the small filter and r_l for the large filter. Now we get to the actual rendering part:

```
1. Set the camera to an orthographic projection (so that we can draw the environment map seen from the top).
2. Draw alpha texture with $d^2$ to alpha channel
3. For both radii $r_s$ and $r_l$:
4.     While $r_s$ (resp. $r_l$) $<$ hardware supported filter size:
5.         Divide $r_s$ (resp. $r_l$) by 2. Double the shrink factor.
6.     Draw environment map shrinked by the shrink factor (uses mip-mapping).
7.     Sample Phong model into the filter.
8.     Filter the environment map with it (OpenGL convolution).
9.     Store it again as texture map (RGB$\alpha$ texture).
10. Draw environment map S.
11. Blend environment map L with it (using $d^2$).
12. Store again as a texture map.
13. Set up real camera.
14. Draw reflective object with generated environment map.
```

One problem arises when the center of the filter kernel is close to the border of the environment map. Part of the filter kernel will be outside the actual environment map, thus including values from outside the environment map. This can be solved by including a large border in the environment map.

Some renderings that were done using an SGI O2 at interactive rates, are depicted in Figure 14. All the renderings were done with parabolic environment maps

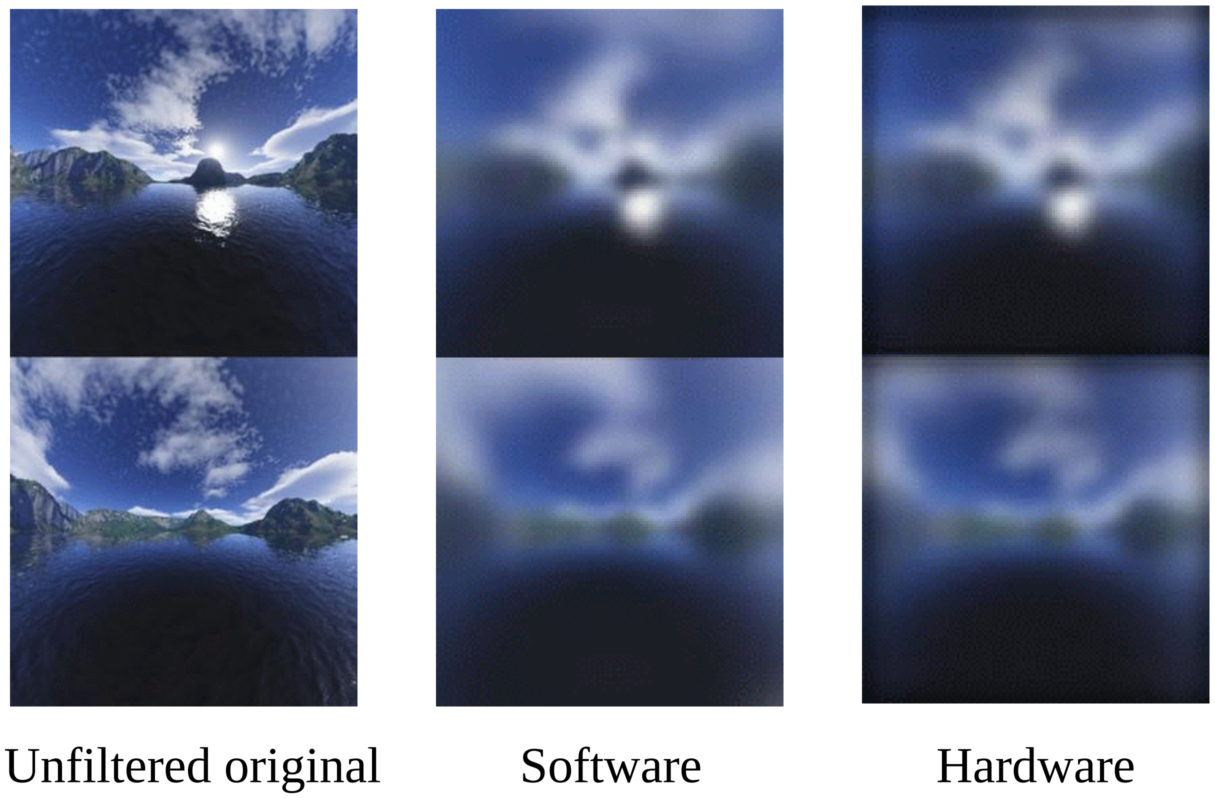

Figure 13: Comparison of the different filtering methods. Filtering was done with the Phong model and an exponent of 100. From left to right: Unfiltered, the classic method, our new hierarchical method, the hardware accelerated method. The original environment map is 128×256 pixels in size with a border of 16 pixels.

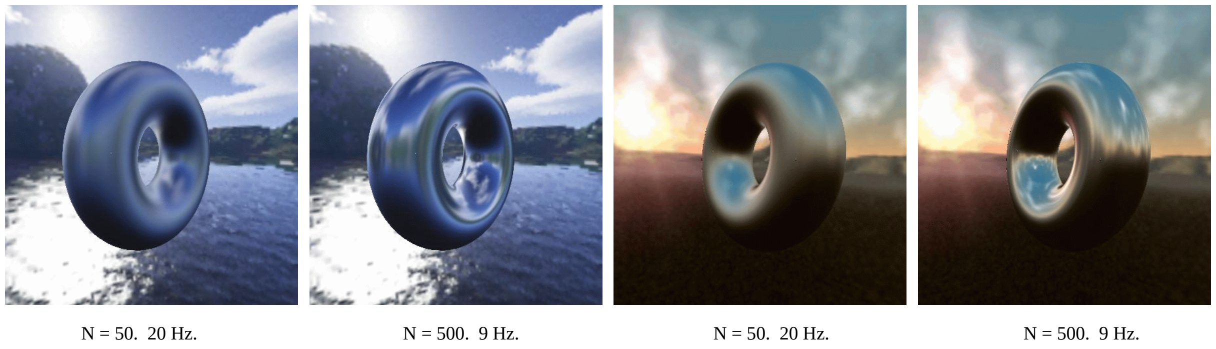

Figure 14: Two scenes rendered with a glossy reflective torus (SGI O2). Filtering is done with the Phong model (exponent of 50 and 500) for every frame, but interactive rates are still achieved. The original environment maps are 512×1024 pixels in size with a border of 64 pixels.

with 512×1024 pixels. The border was 64 pixels in each direction (for each face). The maximum filter size we used was 7 (larger filter sizes considerably degrade the convolution speed on an O2). The following timings were done with the same setting (reflective sphere, 2592 triangles):

exponent	filter size		shrink factor		fps HW
	small	large	small	large	
10	174	260	32	64	25
50	78	136	16	32	20
100	56	100	8	16	16
250	36	66	8	16	16
500	26	48	4	8	9

Please note that filtering was performed for *every* frame, even though the Phong exponent did not change. We have included timings for the hardware convolution, the filter sizes that would have been required (the BRDF clamp value was set to 0.1) and the necessary shrink factor to get filter sizes with a maximum size of 7. You can see that for small Phong exponents hardware prefiltering is very fast. For larger Phong exponents the rendering speed is slower, because filtering then is done with a larger environment map. For a visual comparison please see Figure 13. Due to the way convolutions work in OpenGL, the hardware method generates dark borders, but this does not pose a problem since these are not used for rendering (they replicate information present in the other hemisphere). Figure 14 shows renderings with different environment maps and different Phong exponents; they all run at interactive rates.

Using the same arguments as in Section 3.2.2, we can not only use Phong materials for this hardware prefiltering, but any BRDF with radially symmetric lobes.

4 Non-Photorealistic Rendering

In addition to photorealistic rendering of reflections with various realistic materials, environment maps can also be used to achieve non-photorealistic effects that can be useful for illustration purposes. Figure 15 shows some examples for this. Since in many applications of non-photorealistic rendering, the scene consists of a single object that is rotated relative to the viewer and the environment, spherical environment maps are sufficient. The top row of Figure 15 shows these maps, and examples for rendering results are shown in the bottom row.

The first example is simply a sketch of a window frame and is inspired by a RenderMan shader. Although the image only consists of two colors, black and white, the shape of the object can be gathered quite well, especially when interacting with the program.

The second example shows a cold-warm shading with a very simple, hand-drawn environment map. This method has been described by Gooch et al [5]. The environment map merely consists of two triangles, one blue, one yellow, and one white circle. The resulting image has been blurred with fairly large blur filter. The cold color blue looks like a shadowed region, while the warm color yellow, including the white "highlight" look like parts lit by the sun.

The third column shows an environment mapping technique, which is also from Gooch et al [5], and is designed to render silhouettes black and everything else white. This is achieved by having a thin ring of black pixels in the outer regions of the environment map. Silhouette areas with a small curvature will have a fairly wide black stripe, while areas with a strong curvature have a very narrow one. In this case it is particularly important to use a mip-mapped environment map, because otherwise the high-curvature regions will not have a continuous line. In order to get a silhouette that is independent of curvature, we can also provide a hand-crafted hierarchy of mip-map levels, where the surrounding black border has the same pixel width in all levels. Since the mip-map level is chosen such that the pixel area in the level approximately corresponds to the pixel area in the final image, this will produce a border of approximately constant width (this works best if the hardware supports anisotropic filtering).

The final example shows a simple environment map that simulates hatching in regions close to the silhouettes. Obviously, much more artistic versions of such maps are possible.

5 Acknowledgments

In this document, we used the cafe environment map from Paul Haeberli, as well as environment maps from Jonas Jax and Jeff Heath. Figure 11 was provided by Marc Olano.

References

[1] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19:542–546, 1976.

[2] Brian Cabral, Marc Olano, and Paul Nemec. Reflection space image based rendering. In *Computer Graphics (SIGGRAPH ’99 Proceedings)*, pages 165–170, August 1999.

[3] Paul E. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Computer Graphics (SIGGRAPH ’98 Proceedings)*, pages 189–198, July 1998.

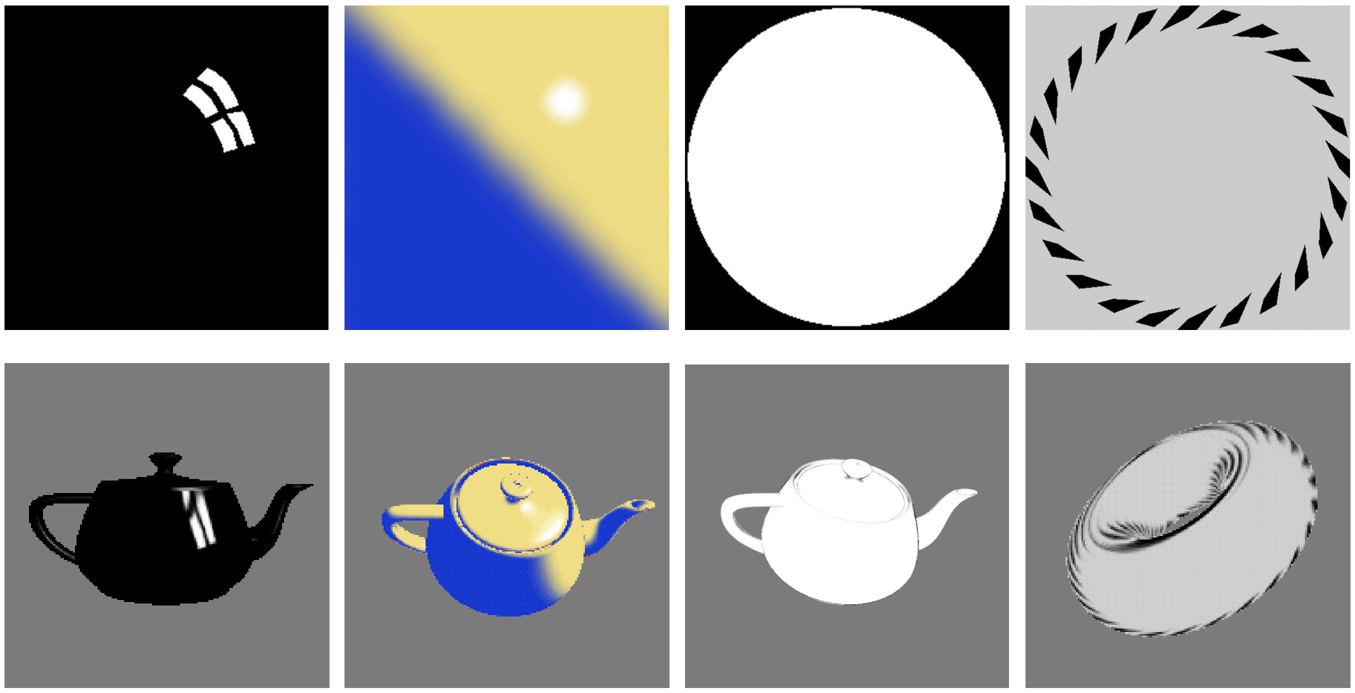

Figure 15: Several spherical environment maps for non-photorealistic rendering (top) applied to geometry (bottom).

[4] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 369–378, August 1997.

[5] Bruce Gooch, Peter-Pike Sloan, Amy Gooch, Peter Shirley, and Richard Riesenfeld. Interactive technical illustration. In *ACM Symposium on Interactive 3D Graphics*, pages 31–38, 1999.

[6] Ned Greene. Applications of world projections. In *Proceedings of Graphics Interface '86*, pages 108–114, May 1986.

[7] Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, pages 259–266, June 1993.

[8] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 289–298, August 1990.

[9] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, 1998.

[10] Wolfgang Heidrich and Hans-Peter Seidel. Realistic, hardware-accelerated shading and lighting. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999.

[11] Jan Kautz and Michael McCool. Approximation of glossy reflection with prefiltered environment maps. In *Proc. of Graphics Interface*, May 2000.

[12] Jan Kautz, Pere-Pau Vázquez, Wolfgang Heidrich, and Hans-Peter Seidel. Unified approach to prefiltered environment maps. In *submitted*, 2000.

[13] Robert R. Lewis. Making shaders more physically plausible. In *Fourth Eurographics Workshop on Rendering*, pages 47–62, June 1993.

[14] G. Miller and R. Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. In *SIGGRAPH '84 Course Notes – Advanced Computer Graphics Animation*, July 1984.

[15] Shree K. Nayar. Catadioptric omnidirectional camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 482–488, June 1997.

[16] Andreas Schilling, Günter Knittel, and Wolfgang Straßer. Texram: A smart memory for texturing. *IEEE Computer Graphics and Applications*, 16(3):32–41, May 1996.

[17] Mark Segal and Kurt Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*, 1998.

[18] D. Voorhies and J. Foran. Reflection Vector Shading Hardware. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 163–166, July 1994.

[19] Lance Williams. Pyramidal parametrics. In *Computer Graphics (SIGGRAPH '83 Proceedings)*, pages 1–11, July 1983.