Realistic, Hardware-accelerated Shading and Lighting

Wolfgang Heidrich* Hans-Peter Seidel

Max-Planck-Institute for Computer Science

Abstract

With fast 3D graphics becoming more and more available even on low end platforms, the focus in hardware-accelerated rendering is beginning to shift towards higher quality rendering and additional functionality instead of simply higher performance implementations based on the traditional graphics pipeline.

In this paper we present techniques for realistic shading and lighting using computer graphics hardware. In particular, we discuss multipass methods for high quality local illumination using physically-based reflection models, as well as techniques for the interactive visualization of non-diffuse global illumination solutions. These results are then combined with normal mapping for increasing the visual complexity of rendered images.

Although the techniques presented in this paper work at interactive frame rates on contemporary graphics hardware, we also discuss some modifications of the rendering pipeline that help to further improve both performance and quality of the proposed methods.

CR Categories: I.3.1 [Computer Graphics]: Hardware Architecture—Graphics processors; I.3.3 [Computer Graphics]: Picture/Image Generation—Bitmap and frame buffer operations; I.3.6 [Computer Graphics]: Methodology and Techniques—Standards I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, Shading, Shadowing and Texture

Keywords: reflectance functions, illumination effects, shading, texture mapping, rendering hardware, frame buffer techniques

1 Introduction

Until recently, the major concern in the development of new graphics hardware has been to increase the performance of the traditional rendering pipeline. Today, graphics accelerators with a performance of several million textured, lit triangles per second are within reach even for the low end. As a consequence, we see that the emphasis is beginning to shift away from higher performance towards higher quality renderings and an increased feature set that makes graphics hardware applicable to more demanding applications.

Despite this development, most current graphics hardware still only uses a local Phong illumination model, which was shown to be inappropriate a long time ago [4]. Moreover, techniques for visualizing non-diffuse global illumination are not widely applied, although several researchers have worked on such methods.

In this paper, we present a class of techniques to improve the quality of shading and lighting in hardware-accelerated computer graphics. We start with algorithms for high-quality local illumination using alternative lighting models such as the one by Torrance and Sparrow [39]. This approach is based on an analytic factorization of the respective model into bivariate terms that can be represented as texture maps. We then discuss methods for visualizing non-diffuse global illumination solutions based on environment maps. We introduce both a Fresnel term for simulating reflections in non-metallic objects, as well as a pre-filtering method for environment maps. To this end, we also review an alternative parameterization for environment maps that we recently introduced [20], and that allows us to use one map for all viewing positions and directions. These techniques are finally combined with normal mapping to increase the visual complexity of the scene.

Although the techniques presented here produce renderings at interactive frame rates on contemporary graphics hardware, a more direct support by future hardware could be achieved through some modifications of the rendering pipeline. These will be outlined at the end of this paper.

2 Previous Work

Most of today's graphics hardware uses either the Phong [7], or the Blinn-Phong [4] illumination model. Many other, physically more plausible models have been proposed, but have so far only been used in software rendering systems or by hardware that has programmable shaders, such as [30]. The most important of these models are the ones by Torrance and Sparrow [39] and Cook and Torrance [11]. The Torrance-Sparrow model uses a Gaussian micro facet distribution function and a geometric attenuation factor based on the assumption of v-shaped grooves on the surface. Other distribution functions have been proposed, for example, by Beckmann and Spizzichino [3], while Smith [36] presented a more accurate geometry term under the assumption of a Gaussian facet distribution. He et al. [18] proposed the HTSG model based directly on the Kirchhoff theory of electromagnetic fields. This model is capable of simulating even more physical effects, although at significantly increased computational cost.

In addition to the isotropic models listed above, anisotropic models have also been proposed. Banks [1] described a very simple model based on results from the illumination of lines in 3-space, while Cabral et al. [8] and Poulin and Fournier [32] use simulations of different kinds of micro geometry. Ward [42] modified the Torrance-Sparrow model by using an anisotropic micro facet distribution function.

Several different methods for interactively visualizing non-diffuse global illumination have been suggested in the literature. Environment maps as a means of computing a mirror term were proposed by Blinn [6], while the spherical parameterization used by most graphics hardware today was presented by Haeberli and Segal [16]. Diefenbach [13] demonstrated a multipass method for rendering mirror and glossy reflections on planar surfaces. Ofek and

*Computer Graphics Group, Im Stadtwald, 66123 Saarbrücken, Germany, {heidrich,hpseidel}@mpi-sb.mpg.de

Rappoport [29] interactively compute mirror reflections off curved reflectors by warping the reflected geometry. Miller et al. [28] propose an approach where the glossy reflection on a surface is stored in a compressed light field, and Walter et al. [41] place virtual lights in the scene to simulate glossy reflections. Stürzlinger and Bastos [38] employ multipass rendering to visualize the result of a photon tracing algorithm. Finally, in some work developed in parallel to ours, Bastos et al. [2] use textures and filtering techniques to render reflections in planar objects based on physically correct reflection models, and Cabral et al. [9] propose an environment mapping technique for glossy reflections not unlike ours.

Bump maps, originally introduced by Blinn [5], have recently found their way into hardware-accelerated rendering. Some researchers [14, 26, 31] built or proposed dedicated graphics hardware to implement bump maps, while others [25] use multipass techniques to realize bump maps with traditional graphics hardware. In contrast to bump maps, which define the variation of the surface normal in terms of a bump height, normal maps directly specify the direction of the surface normal. On the one hand, normal maps are less expensive to compute than bump maps, and can also be generated more easily, for example by measurements [33], or surface simplification [10]. On the other hand, normal maps are attached to the underlying geometry, while bump maps can be applied to any surface.

3 Alternative Lighting Models for Local Illumination

In this section we describe techniques for applying physically accurate reflection models to the computation of local illumination in hardware-based rendering. Rather than replacing the standard Phong model by another single, fixed model, we seek a method that allows us to utilize a wide variety of different models so that the most appropriate model can be chosen for each application.

To achieve this flexibility without introducing procedural shading, a sample-based representation of the BRDF seems most promising. However, a faithful sampling of 3D isotropic or 4D anisotropic BRDFs requires too much storage to be useful on contemporary graphics hardware. Wavelets or spherical harmonics could be used to store this data more compactly, but these representations do not easily lend themselves to hardware implementations.

3.1 Isotropic Models

We propose a different approach. It turns out that most lighting models in computer graphics can be factored into independent components that only depend on one or two angles. Consider, for example the model by Torrance and Sparrow [39]:

$$f_r(\vec{l} \to \vec{v}) = \frac{F \cdot G \cdot D}{\pi \cdot \cos\alpha \cdot \cos\beta}, \tag{1}$$

where f_r is the BRDF, α is the angle between the surface normal $\vec{n}$ and the vector $\vec{l}$ pointing towards the light source, while β is the angle between $\vec{n}$ and the viewing direction $\vec{v}$. The geometry is depicted in Figure 1.

For a fixed index of refraction, the Fresnel term F in Equation 1 only depends on the angle θ between the light direction $\vec{l}$ and the micro facet normal $\vec{h}$, which is the halfway vector between $\vec{l}$ and $\vec{v}$. Thus, the Fresnel term can be seen as a univariate function $F(\cos\theta)$.

The micro facet distribution function D, which defines the percentage of facets oriented in direction $\vec{h}$, depends on the angle δ between $\vec{h}$ and the surface normal $\vec{n}$, as well as a roughness parameter.

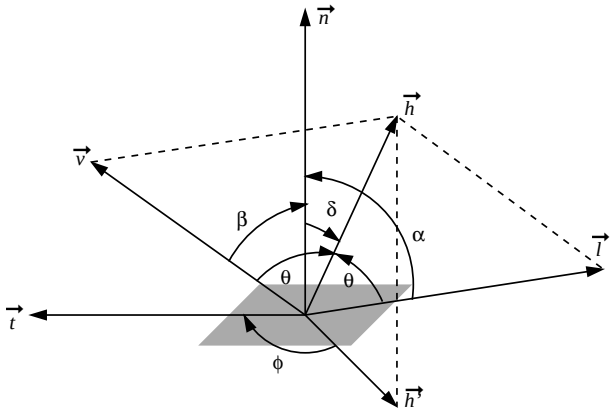

Figure 1: The local geometry of reflection at a rough surface.

This is true for all widely used choices of distribution functions, including a Gaussian distribution of δ or of the surface height, as well as the distribution by Beckmann [3]. Since the roughness is generally assumed to be constant for a given surface, this is again a univariate function $D(\cos\delta)$.

Finally, when using the geometry term G proposed by Smith [36], which describes the shadowing and masking of light for surfaces with a Gaussian micro facet distribution, this term is a bivariate function $G(\cos\alpha, \cos\beta)$.

The contribution of a single point- or directional light source with intensity I_i to the intensity of the surface is given as $I_o = f_r(\vec{l} \to \vec{v}) \cos\alpha \cdot I_i$. The term $f_r(\mathbf{x}, \vec{l} \to \vec{v}) \cos\alpha$ can be split into two bivariate parts $F(\cos\theta) \cdot D(\cos\delta)$ and $G(\cos\alpha, \cos\beta)/(\pi \cdot \cos\beta)$, which are then stored in two independent 2-dimensional lookup tables.

Regular 2D texture mapping can be used to implement the lookup process. If all vectors are normalized, the texture coordinates are simple dot products between the surface normal, the viewing and light directions, and the micro facet normal. These vectors and their dot products can be computed in software and assigned as texture coordinates to each vertex of the object.

The interpolation of these texture coordinates across a polygon corresponds to a linear interpolation of the vectors without renormalization. Since the reflection model itself is highly nonlinear, this is much better than simple Gouraud shading, but not as good as evaluating the illumination in every pixel (Phong shading). The interpolation of normals without renormalization is commonly known as *fast Phong shading*.

This method for looking up the illumination in two separate 2-dimensional textures requires either a single rendering pass with two simultaneous textures, or two separate rendering passes with one texture each in order to render specular reflections on an object. If two passes are used, their results are multiplied using alpha blending. A third rendering pass with hardware lighting (or a third simultaneous texture) is applied for adding a diffuse term.

If the light and viewing directions are assumed to be constant, that is, if a directional light and an orthographic camera are assumed, the computation of the texture coordinates can even be done in hardware. To this end, light and viewing direction as well as the halfway vector between them are used as row vectors in the texture matrix for the two textures:

$$\begin{bmatrix} 0 & 0 & 0 & \cos\theta \\ h_x & h_y & h_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta \\ \cos\delta \\ 0 \\ 1 \end{bmatrix} \tag{2}$$

$$\begin{bmatrix} l_x & l_y & l_z & 0 \\ v_x & v_y & v_z & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \\ n_z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha \\ \cos\beta \\ 0 \\ 1 \end{bmatrix} \quad (3)$$

Figure 2 shows a torus rendered with two different roughness settings using this technique. The assumption of an orthographic camera for lighting purposes is quite common in hardware-accelerated rendering, since it saves the normalization of the viewing vector for each vertex. APIs like OpenGL have a separate mode for applications where this simplification cannot be used, and the viewing direction has to be computed for every vertex. This case is called a *local viewer*.

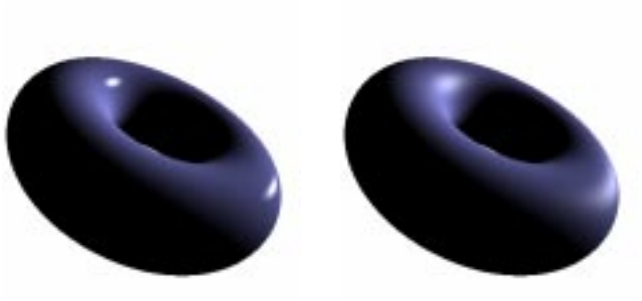

Figure 2: A torus rendered with the Torrance-Sparrow reflection model and two different settings for the surface roughness.

We would like to note that the use of textures for representing the lighting model introduces an approximation error: while the term $F \cdot D$ is bounded by the interval $[0, 1]$, the second term $G/(\pi \cdot \cos\beta)$ exhibits a singularity for grazing viewing directions ($\cos\beta \rightarrow 0$). Since graphics hardware typically uses a fixed-point representation of textures, the texture values are clamped to the range $[0, 1]$. When these clamped values are used for the illumination process, areas around the grazing angles can be rendered too dark, especially if the surface is very shiny. This artifact can be reduced by dividing the values stored in the texture by a constant which is later multiplied back onto the final result. In practice, however, these artifacts are hardly noticeable.

The same methods can be applied to all kinds of variations of the Torrance-Sparrow model, using different distribution functions and geometry terms, or the approximations proposed in [34]. With varying numbers of terms and rendering passes, it is also possible to come up with similar factorizations for all kinds of other models. For example the Phong, Blinn-Phong and Cosine Lobe models can all be rendered in a single pass with a single texture, which can even already account for an ambient and a diffuse term in addition to the specular one.

3.2 Anisotropy

Although the treatment of anisotropic materials is somewhat harder, similar factorization techniques can be applied here. For anisotropic models, the micro facet distribution function and the geometrical attenuation factor also depend on the angle ϕ between the facet normal and a reference direction in the tangent plane. This reference direction is given in the form of a tangent vector $\vec{t}$.

For example, the elliptical Gaussian model [42] introduces an anisotropic facet distribution function specified as the product of two independent Gaussian functions, one in the direction of $\vec{t}$, and one in the direction of the binormal $\vec{n} \times \vec{t}$. This makes D a bivariate function in the angles δ and ϕ. Consequently, the texture coordinates can be computed in software in much the same way as described above for isotropic materials. This also holds for the other anisotropic models in computer graphics literature.

Since anisotropic models depend on both a normal and a tangent per vertex, the texture coordinates cannot be generated with the help of a texture matrix, even if light and viewing directions are assumed to be constant. This is due to the fact that the anisotropic term can usually not be factored into a term that only depends on the surface normal, and one that only depends on the tangent.

One exception to this rule is the model by Banks [1], which is mentioned here despite the fact that it is an *ad-hoc* model which is not based on physical considerations. Banks defines the reflection off an anisotropic surface as

$$I_o = \cos\alpha \cdot (k_d < \vec{n'}, \vec{l} > + k_s < \vec{n'}, \vec{h} >^{1/r}) \cdot I_i, \quad (4)$$

where $\vec{n'}$ is the projection of the light vector $\vec{l}$ into the plane perpendicular to the tangent vector $\vec{t}$. This vector is then used as a shading normal for a Blinn-Phong lighting model with diffuse and specular coefficients k_d and k_s, and surface roughness r. In [37], it has been pointed out that this Phong term is really only a function of the two angles between the tangent and the light direction, as well as the tangent and the viewing direction. This fact was used for the illumination of lines in [37].

Applied to anisotropic reflection models, this means that this Phong term can be looked up from a 2-dimensional texture, if the tangent $\vec{t}$ is specified as a texture coordinate, and the texture matrix is set up as in Equation 3. The additional term $\cos\alpha$ in Equation 4 is computed by hardware lighting with a directional light source and a purely diffuse material, so that the Banks model can be rendered with one texture and one pass per light source. Figure 3 shows two images rendered with this reflection model.

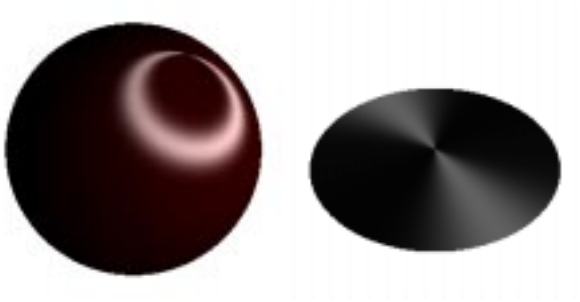

Figure 3: Sphere and disk rendered with Banks' anisotropic reflection model.

4 Visualizing Global Illumination with Environment Maps

The presented techniques for applying alternative reflection models to local illumination computations can significantly increase the realism of synthetic images. However, true photorealism is only possible if global effects are also considered. Since texture mapping techniques for diffuse illumination are widely known and applied, we concentrate on non-diffuse global illumination, in particular mirror- and glossy reflection.

Our approach is based on environment maps, because they offer a good compromise between rendering quality and storage requirements. With environment maps, 2-dimensional textures instead of the full 4-dimensional radiance field [28] can be used to store the illumination.

4.1 View-independent Environment Maps

The first step for using environment maps is the choice of an appropriate parameterization. The spherical parameterization [16] used in most of today's graphics hardware is based on the simple analogy of a infinitely small, perfectly mirroring ball centered around the object. The environment map is the image that an orthographic camera sees when looking at this ball along the negative z-axis. The sampling rate of this map is maximal for directions opposing the viewing direction (that is, objects behind the viewer), and goes towards zero for directions close to the viewing direction. Moreover, 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 the environment.

With these properties, it is clear that this parameterization is not suitable for viewing directions other than the original one. Maps using this parameterization have to be regenerated for each change of the view point, even if the environment is otherwise static. The major reason why spherical maps are used anyway, is that the lookup can be computed efficiently with simple operations in hardware: the texture coordinates for a given reflection direction are simply the x and y components of the normalized halfway vector between the reflection vector and the z-axis, which acts as a reference viewing direction (see Figure 4). For orthographic cameras, this halfway vector simplifies to the surface normal.

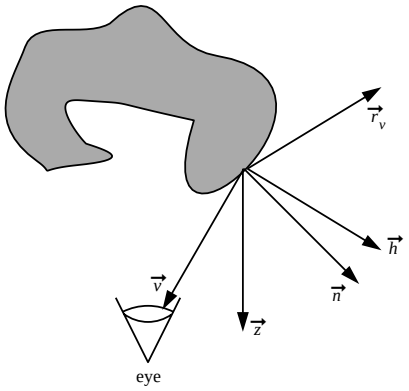

Figure 4: Spherical environment maps are indexed by the x and y components of the halfway vector $\vec{h}$ between the viewing direction $\vec{v}$ and the z-axis.

A parameterization which avoids the problems of spherical maps is cubical environment mapping, which consist of 6 independent perspective images from the center of a cube through each of its faces. The sampling of these maps is fairly good, as the sampling rates for the directions differ by a factor of $3\sqrt{3} \approx 5.2$. However, although hardware implementations of this parameterization have been proposed [40], these are not available at the moment. The reason for this is probably that the handling of six independent textures poses problems, and that anti-aliasing across the image borders is difficult.

We use a different parameterization that is both view independent and easy to implement on current and future hardware, and that we first described in [20]. A detailed description of its properties can be found in [19]. The parameterization is based on an analogy similar to the one used to describe spherical maps. Assume that the reflecting object lies in the origin, and that the viewing direction is along the negative z axis. The image seen by an orthographic camera when looking at the paraboloid

$$f(x,y) = \frac{1}{2} - \frac{1}{2}(x^2 + y^2), \quad x^2 + y^2 \leq 1 \tag{5}$$

contains the information about the hemisphere facing towards the viewer (see Figure 5). The complete environment is stored in two separate textures, each containing the information of one hemisphere.

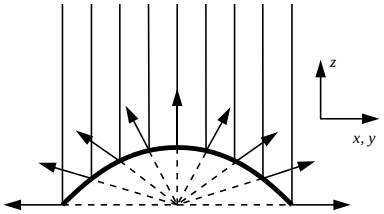

Figure 5: The reflections off a paraboloid can be used to parameterize the incoming light from a hemisphere of directions.

One useful property of this parameterization is a very uniform sampling of the hemisphere of directions. The differential solid angle covered by a pixel at location (x, y) is given as

$$d\omega(x,y) = \frac{dA}{|(x,y,f(x,y))^T|^2} \cdot sr, \tag{6}$$

where dA is the differential area of the pixel. From this formula, it is easy to derive the fact that the sampling rate only varies by a factor of 4, which is even better than for cubical maps [19].

Another big advantage of this parameterization is that it can be used very efficiently in hardware implementations. Since the normal of the paraboloid in Equation 5 is the vector $[x, y, 1, 0]^T$, the texture coordinates for this parameterization are given as h_x/h_z and h_y/h_z. Thus, if the reflection vector $\vec{r}_v$ in eye space is specified as the initial set of texture coordinates, the coordinates needed for the environment lookup can be computed using a projective texture matrix:

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{M}^{-1} \cdot \begin{bmatrix} r_{v,x} \\ r_{v,y} \\ r_{v,z} \\ 1 \end{bmatrix}, \tag{7}$$

where $\mathbf{M}$ is a linear transformation mapping the environment map space into eye space. The environment map space is the space in which the environment is defined, that is, the one, in which the paraboloid is given by Equation 5. The inverse of $\mathbf{M}$ thus maps the $\vec{r}_v$ back into this space. Then the matrix $\mathbf{S}$ adds the vector $[0, 0, 1, 0]^T$ to compute the halfway vector $\vec{h}$, and finally $\mathbf{P}$ copies the z-component into the homogeneous component w to perform the perspective division.

In order to specify $\vec{r}_v$ as the initial set of texture coordinates, this vector has to be computed per vertex. This can be achieved either in software, or by a hardware extension allowing for the automatic computation of these texture coordinates, which we proposed in [20]. Kilgard [23] has implemented this extension for Mesa and the new drivers for the Riva TNT graphics chip.

What remains to be done is to combine frontfacing and backfacing regions of the environment into a single image. To this end, we mark those pixels inside the circle $x^2 + y^2 \leq 1$ of one of the two maps with an alpha value of 1, the others with 0. The second map does not need to contain an alpha channel. Then, with either a single rendering pass and two texture maps, or two separate rendering passes, the object is rendered with the two different maps applied, and the alpha channel of the first map is used to select which map should be visible for each pixel.

Figure 6 shows the two images comprising an environment map in this parameterization, as well as two images rendered with these

maps under different viewing directions. The environment maps were generated using a ray-caster. The marked circular regions contain the information for the two hemispheres of directions. The regions outside the circles are, strictly speaking, not part of the environment map, but are useful for avoiding seams between the two hemispheres of directions, as well as for generating mipmaps of the environment. These parts have been generated by extending the paraboloid from Equation 5 to the domain $[-1,1]^2$.

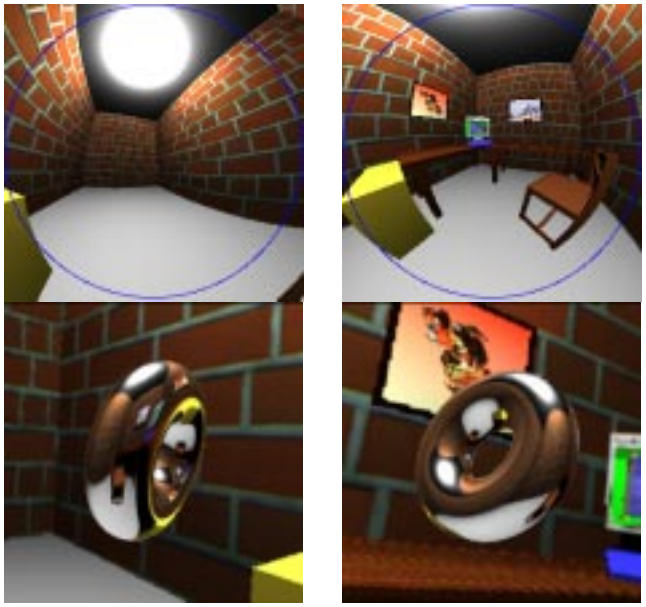

Figure 6: Top: two parabolic images comprising one environment map. Bottom: rendering of a torus using this environment map.

4.2 Mip-map Level Generation for Parabolic Maps

Anti-aliasing of parabolic environment maps can be performed using any of the known prefiltering algorithms, such as mip-mapping [43]. For correct prefiltering, the front- and backfacing maps need to contain valid information for the whole domain $[-1,1]^2$, as in Figure 6.

The next level in the mip-map hierarchy is then generated by computing a weighted sum for each 2×2 block of texels. The weight for each texel is proportional to the solid angle it covers (Equation 6). The sum of these solid angles is the solid angle covered by the texel in the new mip-map level, and is used as a weight for the next iteration step.

Mip-mapping is, of course, an isotropic filtering technique, and therefore produces excessive blurring for grazing viewing angles. These problems, are, however in no way specific to environment maps. By weighting each pixel with its solid angle, filtering is correct within the limits of the mip-mapping approach, since each texel in the map is correctly anti-aliased, and each pixel on the object is textured by exactly one hemispherical map.

4.3 Mirror and Diffuse Terms with Environment Maps

Once an environment map is given in the parabolic parameterization, it can be used to add a mirror reflection term to an object. Due to the view-independent nature of this parameterization, one map suffices for all possible viewing positions and directions. Using multi-pass rendering and either alpha blending or an accumulation buffer [17], it is possible to add a diffuse global illumination term through the use of a precomputed texture. Two methods exist for the generation of such a texture. One way is, that a global illumination algorithm such as Radiosity is used to compute the diffuse global illumination in every surface point.

The second approach is purely image-based, and was proposed by Greene [15]. 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. This information can be used to prefilter the environment map to represent the diffuse reflection of an object for all possible surface normals. Like regular environment maps, this texture is only valid for one point in space, but can be used as an approximation for nearby points.

4.4 Fresnel Term

A regular environment map without prefiltering describes the incoming illumination in a point in space. If this information is directly used as the outgoing illumination, as with regular environment mapping, only metallic surfaces can be modeled. This is because for metallic surfaces (surfaces with a high index of refraction) the Fresnel term is almost 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 index of refraction), 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 viewing direction $\vec{v}$.

Similar to the techniques for local illumination presented in Section 3, the Fresnel term $F(\cos\theta)$ for the mirror direction $\vec{r}_v$ can be stored in a texture map. Since here only the Fresnel term is required, a 1-dimensional texture map suffices for this purpose. This Fresnel term is rendered to the framebuffer's alpha channel in a separate rendering pass. The mirror part is then multiplied with this term in a second pass, and a third pass is used to add the diffuse part. This yields an outgoing radiance of $L_o = F \cdot L_m + L_d$, where L_m is the contribution of the mirror term, while L_d is the contribution due to diffuse reflections.

In addition to simply adding the diffuse part to the 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 7 shows images generated using these two approaches. In the top row, the diffuse term is simply added to the Fresnel-weighted mirror term (the glossy reflection is zero). For a refractive index of 1.5 (left), which approximately corresponds to glass, the object is only specular for grazing viewing angles, while for a high index of refraction (200, right image), which is typical for metals, the whole object is highly specular.

The bottom row of Figure 7 shows two images generated with the second approach. For a low index of refraction, the specular term is again high only for grazing angles, but in contrast to the image above, the diffuse part fades out for these angles. For a high index of refraction, which, as pointed out above, corresponds to metal, the diffuse part is practically zero everywhere, so that the object is a perfect mirror for all directions.

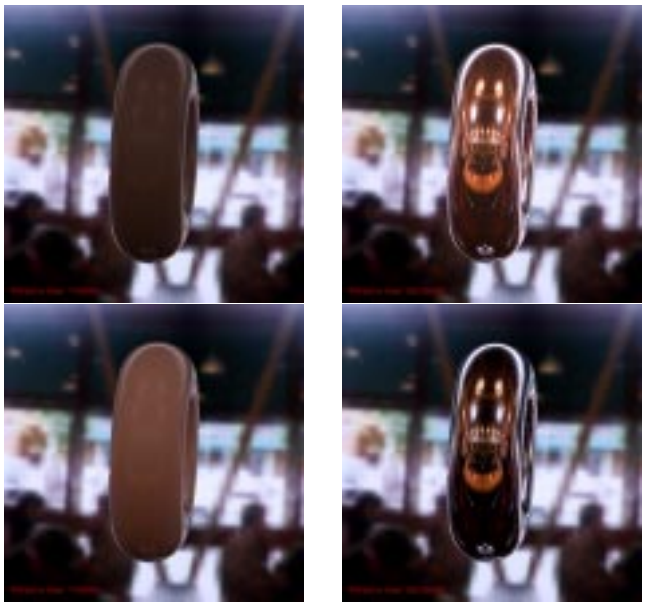

Figure 7: Mirror and diffuse reflections weighted by a Fresnel term for a varying index of refraction. Top: constant diffuse coefficient, bottom: diffuse reflection fading out with the Fresnel term.

Figure 8: Top: original parabolic map used in this figure and Figure 7, as well as prefiltered map with a roughness of 0.01. Bottom: application of the filtered map to a reflective torus (left) and a transmissive rectangle (right).

4.5 Precomputed Glossy Reflection and Transmission

We would now like to extend the concept of environment maps to glossy reflections. The idea is similar to the diffuse prefiltering proposed by Greene [15] and the approach by Voorhies and Foran [40] to use environment maps to generate Phong highlights from directional light sources. These two ideas can be combined to precompute an environment map containing the glossy reflection of an object with a Phong material. With this concept, effects similar to the ones presented by Debevec [12] are possible in real time.

As shown in [24], the Phong BRDF is given by

$$f_r(\vec{l} \rightarrow \vec{v}) = k_s \cdot \frac{<\vec{r}_l, \vec{v}>^{1/r}}{\cos\alpha} = k_s \cdot \frac{<\vec{r}_v, \vec{l}>^{1/r}}{\cos\alpha}, \quad (8)$$

where $\vec{r}_l$, and $\vec{r}_v$ are the reflected light- and viewing directions, respectively.

Thus, the specular global illumination using the Phong model is

$$L_o(\vec{r}_v) = k_s \cdot \int_{\Omega(\vec{n})} <\vec{r}_v, \vec{l}>^{1/r} \; L_i(\vec{l}) \; d\omega(\vec{l}), \quad (9)$$

which is only a function of the reflection vector $\vec{r}_v$ and the environment map containing the *incoming* radiance $L_i(\vec{l})$. Therefore, it is possible to take a map containing $L_i(\vec{l})$, and generate a filtered map containing the *outgoing* radiance for a glossy Phong material. Since this filtering is relatively expensive, it cannot be redone for every frame in an interactive application. Thus, it is important to use a view-independent parameterization such as our parabolic maps.

Figure 8 shows such a map generated from the original environment in Figure 6, as well as a glossy sphere textured with this map. The same technique can be applied to simulate glossy transmission on thin surfaces. This is also depicted in Figure 8.

If the original environment map is given in a high-dynamic range format, then this prefiltering technique allows for effects similar to the ones described by Debevec [12]. Although reflections of an object onto itself cannot be modeled by environment maps, the renderings are quite convincing, considering that these images can be rendered at frame rates of about 20Hz even on low end workstations such as an SGI O2.

5 Normal Maps

Bump maps are becoming popular for hardware-accelerated rendering, because they allow us to increase the visual complexity of a scene without requiring excessive amounts of geometric detail.

Normal maps can be used for achieving the same goal, and have the advantage that the expensive operations (computing the local surface normal by transforming the bump into the local coordinate frame) have already been performed in a preprocessing stage. All that remains to be done is to use the precomputed normals for shading each pixel. Another advantage of normal maps is that recently methods have shown up for measuring them directly [33], or for generating them as a by-product of mesh simplification [10]. Although we only handle normal maps here, some of these results could also be useful for implementations of bump maps, as will be discussed in Section 6.

In this section, we first describe how normal maps can be lit according to the Blinn-Phong illumination model using a set of so-called *imaging operations*. These have been formally defined in OpenGL version 1.2 [35], and it can therefore be expected that they will be available on a wide variety of future graphics hardware.

Afterwards, we discuss how the techniques for other local illumination models from Section 3, as well as the environment mapping techniques from Section 4 can be used together with normal maps. This part relies on the *pixel texture* extension that is currently only available from Silicon Graphics [22]. It has been shown elsewhere [21], that this extension is so powerful and versatile, that it might be implemented by other vendors in future systems.

The methods described here assume an orthographic camera and directional light sources. The artifacts introduced by these assumptions are usually barely noticeable for surfaces with bump maps, because the additional detail hides much of them.

5.1 Normal Maps with local Blinn-Phong Illumination

Among many other features (see [35] for details), the imaging operations make it possible to apply a 4×4 matrix to each pixel in an image, as it is transferred to or from the frame buffer or to texture RAM. Following this color matrix transformation, a lookup table may be applied to each individual color component.

With these two mechanisms and a given normal map in the form of a color coded image, the map can be lit in two rendering passes. First, a color matrix is specified, which maps the normal from object space into eye space and then computes the diffuse illumination (which is essentially given by the dot product with the light direction). When the normal image is now loaded into texture RAM, the lighting computations are performed. Afterwards the loaded, lit texture is applied to the object using texture mapping.

A similar second rendering pass draws the specular part. This time, however, the matrix computes the dot product between normal and the halfway vector $\vec{h}$ from Figure 1. The exponentiation by $1/r$, where r is the surface roughness, is performed by a color lookup table.

Figure 9 shows two images where one polygon is rendered with this technique. On the left side, a simple exponential wave function is used as a normal map. The normal map for the image on the right side has been measured from a piece of wallpaper with the approach presented in [33].

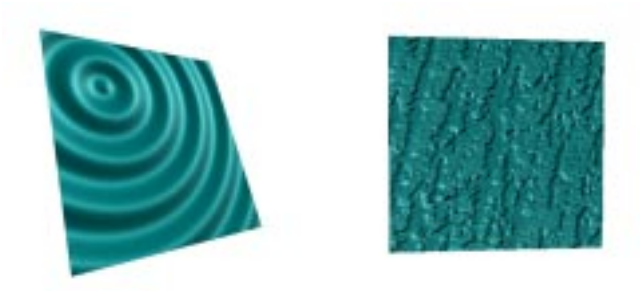

Figure 9: Two Phong-lit normal maps. The right one has been measured from a piece of wallpaper using the approach presented in [33].

5.2 Normal Maps with Other Reflection Models and Environment Maps

The techniques from Sections 3 and 4 could also be applied to normal maps, if the texture lookup could be performed per pixel instead of only per vertex. This can be achieved using the pixel texture extension from Silicon Graphics. It adds an additional stage to the rendering pipeline directly after the color matrix and the color table described above. This stage allows to interpret each of the color components as a texture coordinate pointing into a 1-, 2-, 3-, or 4-dimensional texture (see [22] for details).

This leads to the following algorithm for applying alternative lighting models:

- Render the object with the color coded normal map as a texture, thereby marking each visible pixel in the stencil buffer.
- Copy the frame buffer to main memory. This yields an image which contains the normal vector for each pixel.
- For each pass described in Section 3, set up the color matrix and blending as required by the respective illumination model, and copy the normal image from main memory into the frame buffer. Copy only those pixels marked in the stencil buffer.

For environment maps, the situation is somewhat more difficult, because the parabolic parameterization requires the use of projective textures, which are currently not supported by the pixel texture extension. As has been pointed out in [21], this limitation also prevents the use of pixel textures in other interesting applications, such as shadow maps. Hopefully this restriction will be removed in the future. Until then, pixel textures can still be used for environment mapping from a fixed viewing direction, which of course defeats the point of introducing the parabolic parameterization.

The images in Figure 10 were generated using the pixel texture extension and a single, view-dependent environment map.

Figure 10: Combination of environment mapping and normal mapping. Left: environment map only. Right: Local Phong illumination plus environment map.

6 Discussion and Conclusions

The methods described in this paper provide means for generating realistic shading and lighting effects with computer graphics hardware. We have concentrated on concepts that can be implemented on current graphics hardware. All the techniques presented here run at frame rates of 15-20Hz on an SGI O2 (except for the ones requiring pixel textures, which the O2 does not support), and > 20 Hz on an SGI Octane MXE.

To conclude, we would now like to mention some issues which we deem important for the design of future graphics hardware:

The number of required rendering passes is reduced dramatically if the hardware has support for multiple textures. This is a feature which is beginning to appear on PC boards, and will probably be universally available soon.

Pixel textures are a very valuable tool and should be available on more platforms (see [21] for other applications than presented here). They should also support projective textures.

The easiest way to directly support our techniques in future hardware is to add more automatic texture generation modes. For local illumination with isotropic lighting models, the cosines between surface normal, facet normal, light direction and viewing direction are required. Except for orthographic viewing and directional light sources, these angles need to be computed in software, which requires the transformation of all vectors into eye space. For hardware lighting, these vectors are available in eye space anyway, so if the hardware could generate these cosine values automatically, this would both improve the performance and simplify the implementation. Automatic texture coordinate generation is also useful for generating the reflection vector required for the parabolic environment map parameterization. As pointed out above, this extension is already available in some implementations of OpenGL [23].

A more ambitious change to the rendering pipeline would be to replace the traditional hardware Phong lighting with a customizable sampling-based approach. In this case, the programmer would specify the material as a set of two or three 2-dimensional tables that can be downloaded to the hardware. The geometry processing hardware can then compute the indices into these tables for multiple light sources at the same time.

Although we have only discussed normal maps, the shading and lighting techniques presented here could also be applied to bump mapping hardware, by allowing an additional texture access after the normal for a pixel has been computed. This would, in particular, be a transparent implementation of environment maps for bump mapping hardware.

7 Acknowledgments

This work was done while the authors worked at the University of Erlangen. The cafe environment map used in this paper is a resampled version of the spherical map used in [16]. We would like to thank Mark Kilgard for implementing the extension for parabolic environment maps for Mesa and Riva TNT.

References

[1] D. C. Banks. Illumination in diverse codimensions. In *Computer Graphics (Proceedings of SIGGRAPH '94)*, pages 327–334, July 1994.

[2] R. Bastos, K. Hoff, W. Wynn, and A. Lastra. Increased photorealism for interactive architectural walkthroughs. In *Symposium on Interactive 3D Graphics*. ACM Siggraph, 1999.

[3] P. Beckmann and A. Spizzichino. *The Scattering of Electromagnetic Waves from Rough Surfaces*. McMillan, 1963.

[4] J. F. Blinn. Models of light reflection for computer synthesized pictures. In *Computer Graphics (SIGGRAPH '77 Proceedings)*, pages 192–198, July 1977.

[5] J. F. Blinn. Simulation of wrinkled surfaces. In *Computer Graphics (SIGGRAPH '78 Proceedings)*, pages 286–292, August 1978.

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

[7] P. Bui-Tuong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, June 1975.

[8] B. Cabral, N. Max, and R. Springmeyer. Bidirectional reflection functions from surface bump maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, pages 273–281, July 1987.

[9] B. Cabral, M. Olano, and P. Nemec. Reflection space image based rendering. In *Computer Graphics (SIGGRAPH '99 Proceedings)*, August 1999.

[10] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 115–122, July 1998.

[11] R. L. Cook and K. E. Torrance. A reflectance model for computer graphics. In *Computer Graphics (SIGGRAPH '81 Proceedings)*, pages 307–316, August 1981.

[12] P. 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.

[13] P. J. Diefenbach. *Pipeline Rendering: Interaction and Realism Through Hardware-based Multi-Pass Rendering*. PhD thesis, University of Pennsylvania, 1996.

[14] I. Ernst, H. Rüsseler, H. Schulz, and O. Wittig. Gouraud bump mapping. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 47–54, 1998.

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

[16] P. Haeberli and M. Segal. Texture mapping as A fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, pages 259–266, June 1993.

[17] P. E. Haeberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, pages 309–318, August 1990.

[18] X. D. He, K. E. Torrance, F. X. Sillion, and D. P. Greenberg. A comprehensive physical model for light reflection. In *Computer Graphics (SIGGRAPH '91 Proceedings)*, pages 175–186, July 1991.

[19] W. Heidrich. *High-quality Shading and Lighting for Hardware-accelerated Rendering*. PhD thesis, University of Erlangen-Nürnberg, April 1999.

[20] W. Heidrich and H.-P. Seidel. View-independent environment maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pages 39–45, 1998.

[21] W. Heidrich, R. Westermann, H.-P. Seidel, and Th. Ertl. Applications of pixel textures in visualization and realistic image synthesis. In *Symposium on Interactive 3D Graphics*, 1999. (Accepted for publication).

[22] Silicon Graphics Inc. *Pixel Texture Extension*, December 1996. Specification document, available from http://www.opengl.org.

[23] M. Kilgard. Personal communication, April 1999.

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

[25] T. McReynolds, D. Blythe, B. Grantham, and S. Nelson. Advanced graphics programming techniques using OpenGL. In *Siggraph 1998 Course Notes*, July 1998.

[26] G. Miller, M. Halstead, and M. Clifton. On-the-fly texture computation for real-time surface shading. *IEEE Computer Graphics & Applications*, 18(2):44–58, March–April 1998.

[27] G. Miller and C. Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *ACM SIGGRAPH '84 Course Notes - Advanced Computer Graphics Animation*, July 1984.

[28] G. Miller, S. Rubin, and D. Ponceleon. Lazy decompression of surface light fields for precomputed global illumnation. In *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop)*, pages 281–292, March 1998.

[29] E. Ofek and A. Rappoport. Interactive reflections on curved objects. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 333–342, July 1998.

[30] M. Olano and A. Lastra. A shading language on graphics hardware: The PixelFlow shading system. In *Computer Graphics (SIGGRAPH '98 Proceedings)*, pages 159–168, July 1998.

[31] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 303–306, August 1997.

[32] P. Poulin and A. Fournier. A model for anisotropic reflection. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 273–282, August 1990.

[33] H. Rushmeier, G. Taubin, and A. Guéziec. Applying shape from lighting variation to bump map capture. In *Rendering Techniques '97 (Proceedings of Eurographics Rendering Workshop)*, pages 35–44, June 1997.

[34] C. Schlick. A customizable reflectance model for everyday rendering. In *Fourth Eurographics Workshop on Rendering*, pages 73–83, June 1993.

[35] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 1.2)*, 1998.

[36] B. G. Smith. Geometrical shadowing of a random rough surface. *IEEE Transactions on Antennas and Propagation*, 15(5):668–671, September 1967.

[37] D. Stalling, M. Zöckler, and H.-C. Hege. Fast display of illuminated field lines. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):118–128, 1997.

[38] W. Stürzlinger and R. Bastos. Interactive rendering of globally illuminated glossy scenes. In *Rendering Techniques '97*, pages 93–102, 1997.

[39] K. E. Torrance and E. M. Sparrow. Theory for off-specular reflection from roughened surfaces. *Journal of the Optical Society of America*, 57(9):1105–1114, September 1967.

[40] D. Voorhies and J. Foran. Reflection vector shading hardware. In *Computer Graphics (SIGGRAPH '94 Proceedings)*, pages 163–166, July 1994.

[41] B. Walter, G. Alppay, E. Lafortune, S. Fernandez, and D. P. Greenberg. Fitting virtual lights for non-diffuse walkthroughs. *Computer Graphics (SIGGRAPH '97 Proceedings)*, pages 45–48, August 1997.

[42] G. J. Ward. Measuring and modeling anisotropic reflection. *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 265–273, July 1992.

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