Homomorphic Factorization of BRDFs for High-Performance Rendering

Michael D. McCool   Jason Ang   Anis Ahmad

Computer Graphics Lab
Department of Computer Science
University of Waterloo

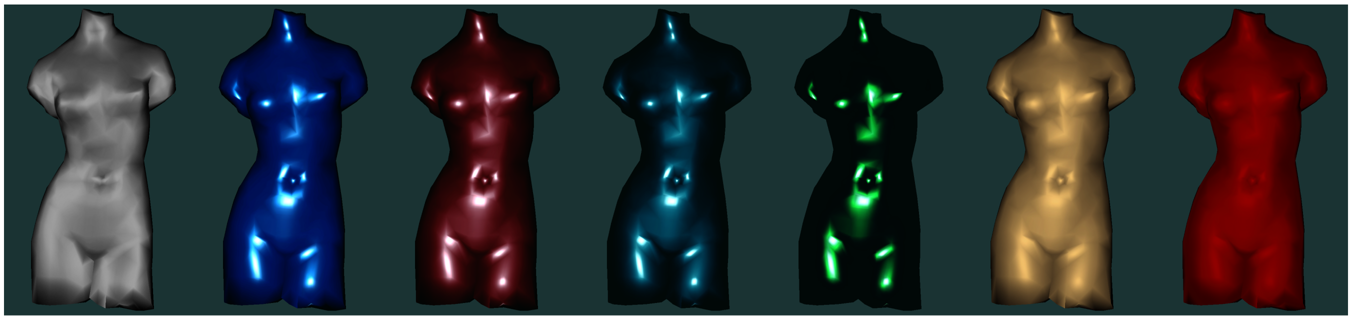

Figure 1: *A model rendered at real-time rates (approximately half the performance of the standard per-vertex lighting model on an NVIDIA GeForce 3) with several BRDFs approximated using the technique in this paper. From left to right: satin (anisotropic Poulin-Fournier model), krylon blue, garnet red, cayman, mystique (Cornell measured data), leather, and velvet (CURET measured data).*

Abstract

A bidirectional reflectance distribution function (BRDF) describes how a material reflects light from its surface. To use arbitrary BRDFs in real-time rendering, a compression technique must be used to represent BRDFs using the available texture-mapping and computational capabilities of an accelerated graphics pipeline. We present a numerical technique, homomorphic factorization, that can decompose arbitrary BRDFs into products of two or more factors of lower dimensionality, each factor dependent on a different interpolated geometric parameter. Compared to an earlier factorization technique based on the singular value decomposition, this new technique generates a factorization with only positive factors (which makes it more suitable for current graphics hardware accelerators), provides control over the smoothness of the result, minimizes relative rather than absolute error, and can deal with scattered, sparse data without a separate resampling and interpolation algorithm.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture.

Keywords: Hardware accelerated rendering and shading.

1 Introduction

The empirical Phong lighting model currently implemented in hardware accelerators cannot capture the subtle differences in reflectance due to different materials. Fortunately, the computational capabilities of existing graphics pipelines can be used to evaluate alternative lighting models. We present a texture-based technique for per-pixel evaluation of physically-based lighting models that can approximate the reflectance models of a wide range of materials using only a few texture lookups and multiplications.

Physical surface reflectance can be modelled locally using a bidirectional reflectance distribution function, or BRDF. Let $\hat{\mathbf{n}}$ be the unit normal at point $\underline{\mathbf{x}}$ on a surface. For a homogeneous surface, the outgoing radiance L_o from point $\underline{\mathbf{x}}$ in direction $\hat{\omega}_o$ can be computed using an integral of the incoming radiance L_i over all incoming directions $\hat{\omega}_i$. The incoming radiance must be weighted by the positive projected surface area $[\hat{\mathbf{n}} \cdot \hat{\omega}_i] = \max(0, \hat{\mathbf{n}} \cdot \hat{\omega}_i)$, the BRDF f, and the solid angle measure σ:

$$L_o(\hat{\omega}_o, \underline{\mathbf{x}}) \quad = \quad \int_\Omega f(\hat{\omega}_o, \hat{\omega}_i)\, L_i(\hat{\omega}_i, \underline{\mathbf{x}})\, [\hat{\omega}_i \cdot \hat{\mathbf{n}}]\, d\sigma(\hat{\omega}_i). \quad (1)$$

While the radiances L_o and L_i are parameterized globally, the BRDF f must be parameterized relative to a local orthonormal surface frame. Such a local surface frame can be generated from the normal $\hat{\mathbf{n}}$ and an orthonormal reference tangent $\hat{\mathbf{t}}$. The last element of the local surface frame, a normalized secondary tangent, can be generated with $\hat{\mathbf{s}} = \hat{\mathbf{n}} \times \hat{\mathbf{t}}$.

When the illumination is from N point sources, the incoming radiance can be modelled with a sum of impulses and the above integral reduces to

$$L_o(\hat{\omega}_o, \underline{\mathbf{x}}) \quad = \quad \sum_{\ell=1}^{N} f(\hat{\omega}_o, \hat{\omega}_i^\ell)\, [\hat{\omega}_i^\ell \cdot \hat{\mathbf{n}}]\, \frac{I_\ell}{r_\ell^2}, \qquad (2)$$

where r_ℓ is the distance to the ℓth light source and I_ℓ is its intensity. We will call this the *general point source local lighting model.*

The parameters to the BRDF, $\hat{\omega}_i$ and $\hat{\omega}_o$, have (in general) four degrees of freedom. For isotropic BRDFs, we can reduce this to three. To get physically-accurate local lighting for point source illumination, we would like to use the above lighting model with the BRDFs of real materials. Unfortunately, analytic models are not always available or practical for BRDFs (often we want to use measured data), and tabulated 3D or 4D data takes too much space to be practical for most real-time applications, especially when many different materials are needed in a scene.

Fortunately, there are several effective BRDF representation and compression techniques. We focus in this paper primarily on real-time rendering, and review several representations suitable for such applications in Section 2.

In this paper a new numerical factorization algorithm and representation for BRDFs is introduced. This factored representation can be used to represent arbitrary BRDFs, including anisotropic BRDFs. Considered as a compression technique, it achieves high compression efficiencies (see Section 5), yet permits decompression (point evaluation) with only a small number of table-lookup and multiplication operations. It also permits simple antialiasing on existing hardware, so surfaces rendered with this representation do not suffer from highlight aliasing. Finally, we can guarantee that the approximation will satisfy Helmholtz reciprocity, an important symmetry property that must be satisfied by physical BRDFs.

The rest of the paper is structured as follows: In Section 2 we review previous work in the representation of local illumination for high-performance image synthesis. Then we present our factorization algorithm in Section 3 and results in Section 4.

2 Previous Work

Practical BRDF representations for real-time rendering currently fall into three main categories: basis summation, environment mapping, and factorization. There is, however, a significant amount of overlap among these categories, and often they can be used synergistically.

2.1 Basis Summation

Basis summation approaches represent BRDFs using a sum of simpler functions. For instance, a BRDF might be represented using a sum of Ward [33] or generalized Phong lobes [21]. In general a large number of lobes might be required to approximate a given BRDF to sufficient accuracy, but if the basis function can be implemented efficiently, then this technique can be practical. For instance, the Phong lobes provided in the lighting model for existing graphics pipelines can be used as basis functions [25, 32], and used either to approximate the BRDF itself or the radiance leaving a surface [30]. Since in current hardware accelerators the built-in lighting model is not usually implemented at the per-pixel level, factorization and environment-mapping based approaches (which are both texture-based and so are evaluated per-pixel) are usually more suitable for many applications. Basis representation techniques based on orthonormal expansions [20, 29] are difficult to implement on many current hardware accelerators due to the lack of comprehensive support for signed arithmetic. Signed arithmetic can be simulated on standard graphics systems if necessary, but a general solution is costly [9].

2.2 Environment Mapping

Environment-mapping based approaches prefilter an environment map with the BRDF and can have excellent visual quality, but have to make some assumptions and approximations, which limits their generality. In particular, environment-map based approaches are poor at representing the effects of anisotropic BRDFs. Basis function approaches and environment-map filtering can be combined. For instance, a BRDF can be approximated with a sum of basis functions that are each radially symmetric and so suitable for filtering an environment map [18, 19]. Image-based techniques based on warping have also been used [6]. Environment-map based approaches can also be combined with factorization approaches in various ways [14], most simply by using an environment map for the specular part of the BRDF.

2.3 Factorization

Factorization techniques can deal only with point or directional sources of illumination. In compensation, they can easily handle anisotropic reflectance models and local viewers. They represent BRDFs using lower-dimensional functions (factors) that are multiplied together [10, 14, 17]. For real-time rendering the factors are generally stored in texture maps and the multiplication is done using compositing or multitexturing arithmetic. Computation of the factorization can be done analytically (for specific BRDFs) or numerically. Factorization can be combined with environment mapping, for instance by weighting a (possibly prefiltered) environment-mapped reflection with shadowing, masking and/or Fresnel factors [14].

A numerical approach based on the singular-value decomposition (SVD) builds a series approximation consisting of several two-factor product terms added together [17]:

$$f(\hat{\omega}_o, \hat{\omega}_i) \quad = \quad \sum_{j=1}^{J} u_j(\pi_u(\hat{\omega}_o, \hat{\omega}_i))\; v_j(\pi_v(\hat{\omega}_o, \hat{\omega}_i)). \qquad (3)$$

Even for difficult BRDFs, given enough terms the SVD approach is capable of achieving arbitrary fidelity. However, for many BRDFs the numerical factorization approach is surprisingly effective with even a small number of terms. If the parameterization of the BRDF is chosen carefully, even one term with two factors is visually adequate in many cases of interest: metals, cloth, even anisotropic brushed metal. Unfortunately, as with other expansions relative to orthonormal bases, the SVD expansion requires signed arithmetic to be used in its full generality. Second, in order to get good separability with the SVD approach, a relatively complicated reparameterization of the BRDF is required. Third, before applying the SVD the BRDF must be reconstructed in four dimensions and resampled on a dense regular grid, which is often difficult and awkward. Finally, the SVD approximation minimizes RMS error, which tends to overemphasize the importance of the fit to the BRDF's peak, at the expense of the low-level but visually important base colour.

The algorithm presented here is an improved numerical factorization algorithm for BRDFs. As such, it can be used with arbitrary BRDFs, including anisotropic BRDFs, but is limited to point (or directional) sources of illumination.

The new factored representation presented here is in many ways similar to the SVD representation. However, our new algorithm and separable BRDF representation addresses the shortcomings of the SVD approach: most importantly, it avoids negative numbers, and can use much simpler and more easily computed parameterizations (in fact, *any* parameterization). These properties translate directly into improved performance, while making the technique more flexible and easier to apply in practice.

3 Decomposition Algorithm

We will now present our decomposition algorithm. Section 3.1 and 3.2 explain the use of the logarithmic homomorphism and the basic structure of the factored representation proposed. Sections 3.3 through 3.5 set up the problem numerically as an overconstrained linear system, while Section 3.6 discusses issues involved in robustly solving this system using an iterative technique.

3.1 Logarithmic Transformation

Rather than using a sum as in Equation 3, our approach approximates a BRDF f using a pure product of an arbitrary number of positive factors,

$$f(\hat{\omega}_o, \hat{\omega}_i) \quad \approx \quad \prod_{j=1}^{J} p_j(\pi_j(\hat{\omega}_o, \hat{\omega}_i)), \tag{4}$$

where the p_j are two-dimensional functions (ultimately to be stored in 2D texture maps) and $\pi_j : \mathbb{R}^4 \mapsto \mathbb{R}^2$ are projection functions associated with each map. The (possibly nonlinear) projection functions define the parameterization of the factors with respect to the original parameterization of the BRDF.

Taking the logarithm of both sides of Equation 4 (letting $\tilde{a} = \log a$) results in the the following linear data-fitting problem:

$$\tilde{f}(\hat{\omega}_o, \hat{\omega}_i) \quad \approx \quad \sum_{j=1}^{J} \tilde{p}_j(\pi_j(\hat{\omega}_o, \hat{\omega}_i)). \tag{5}$$

Once we have solved this transformed problem, we can convert the result back to a solution to the original nonlinear problem by exponentiating. The logarithmic/exponential homomorphism $\log : \mathbb{R}^+ \leftrightarrow \mathbb{R}$ simplifies the problem and also ensures that the result will be positive, even though we solve Equation 5 assuming signed arithmetic.

Our numerical approach can determine optimal values for an arbitrary number of factors and can obtain arbitrarily precise approximations with appropriately independent projection functions. As with the SVD, for the best results it is important to pick projection functions π_j that are compatible with the features and directions of greatest variation of the function f.

The mathematics and numerical techniques we will present are similar to algorithms used in computed axial tomography [3]; specifically, filtered backprojection. Filtered backprojection can reconstruct a higher-dimensional function to arbitrary precision given enough independent lower-dimensional projections. However, rather than reconstructing a function from projections, we find a set of projections to best reconstruct a given function. We also do not assume a uniform or dense set of projection functions.

3.1.1 Disadvantages of Approach

There are some potential disadvantages to the logarithmic/exponential homomorphic transformation of the problem.

First of all, if f can be zero for any combination of parameters, then a small bias must be added to f before taking the logarithm. Otherwise, negative infinities will dominate the data-fitting problem. We actually use the transformation

$$\tilde{f} \quad = \quad \log\left(\frac{f + \varepsilon A}{A}\right), \tag{6}$$

where A is the average value of the available BRDF samples (A will equal the best diffuse approximation if the samples are equidistributed). This transformation also maps values of f nearer to A closer to 0 in log space, which leads to better numerical stability when fitting sparse data (most fitters send solutions to zero if no constraints are present). The inverse of this mapping is

$$f \quad = \quad A \exp(\tilde{f}) - \varepsilon A. \tag{7}$$

In practice, we omit the subtraction to avoid negative values in the result. Use of the bias term εA formally invalidates the transformation from Equation 4 to Equation 5, although it still holds approximately (and well enough in practice) if ε is small. We fix $\varepsilon = 10^{-5}$, so the error due to the εA bias is well below the relative error in almost all measured data (note that the bias can be interpreted as an artificial "measurement error" in f). Appropriate modifications must also be made to account for the scale factor A: each of the J terms $\tilde{p}_j$ actually uses the scale factor $\sqrt[J]{A}$, while $\tilde{f}$ uses A. To simplify presentation, we assume in the derivation which follows that the logarithm is actually used.

Secondly, errors made in the linearized fitting problem, if assumed centered around zero, will have a tendency to bias the exponentiated solution towards positive values. However, this bias is consistent in the formal sense that as the accuracy of the approximation improves (for instance, with approximations using a greater number of factors), it will tend to zero. On the other hand, the logarithmic homomorphism discounts large values in the data, which prevents it from overfitting peaks, but also tends to bias the approximation downward, and can cause it to misrepresent the specular color. However, often we will want to separate out the specular peak anyways (to represent it with an environment map, for instance).

Fortunately, for real-time rendering, if we can control the error and bias so they are within the numerical precision of the factor representations and the reconstruction and display process, they will be irrelevant. We can also correct for bias problems (on average) to preserve the energy conservation properties of the BRDF representation. The error of an actual approximation is analyzed in Section 4.3.

Finally, in graphics accelerators that provide only fixed-point per-pixel arithmetic, we may not have enough precision near zero to get an accurate reconstruction of the BRDF, or enough dynamic range. We can rescale the texture maps to make best use of the available precision and can perform the reconstruction computation in logarithmic space if needed; see Section 4.1.

3.1.2 Advantage of Approach

The disadvantages of the homomorphic approach to factorization are offset by a significant advantage: If we solve the linearized problem described in Equation 5 in logarithmic space by minimizing RMS error, we will in fact be minimizing *relative* RMS error for the solution to the original nonlinear problem. This is perceptually desirable, since the eye seems to be roughly sensitive to ratios of intensity, not absolute intensity [7, 13].

3.2 Parameterization

Despite the potential generality of our approach, in this paper we will limit ourselves (mostly) to a three-factor approximate BRDF representation and a very simple parameterization that reuses one of two texture maps:

$$\hat{\mathbf{h}} \quad = \quad \mathrm{norm}(\hat{\omega}_o + \hat{\omega}_i), \tag{8}$$

$$f(\hat{\omega}_o, \hat{\omega}_i) \quad \approx \quad p(\hat{\omega}_o)\, q(\hat{\mathbf{h}})\, p(\hat{\omega}_i). \tag{9}$$

This particular parameterization is based on the assumptions that shadowing, masking, and the microfacet distribution dominate the *variation* of reflectance [1], and that shadowing and masking are independent. Of course other effects contribute to reflectance and can easily be significant: multiple surface and subsurface scattering [15, 24], interference [12], the Fresnel effect (dependent on $(\hat{\omega} \cdot \hat{\mathbf{h}})$, where $\hat{\omega}$ equals $\hat{\omega}_i$ *or* $\hat{\omega}_o$), and changes in the apparent microfacet distribution dependent on masking [5]. Shadowing and masking effects can also violate the assumption of independence when $\hat{\omega}_i \approx \hat{\omega}_o$. However, our numerical technique will use the available degrees of freedom to fit *any* phenomena that are present as well as possible. For instance, a colour shift caused by multiple scattering or interference will automatically be approximated with a function that varies with incident and exitant direction under the chosen parameterization. It is possible that in the future other parameterizations and factorizations might be devised to better capture BRDFs whose dominant variation arises from other mechanisms than we assume, or even that optimized parameterizations could be found automatically. It would be interesting, for instance, to explore the use of additional factors dependent on $(\hat{\omega}_i \cdot \hat{\omega}_o)$ or especially $(\hat{\omega} \cdot \hat{\mathbf{h}})$. The $\hat{\mathbf{d}}$ vector from the Kautz-McCool parameterization [17] or normalized affine combinations of any of the other vectors could also be used.

However, the specific target representation given in Equation 9 has several advantages. The approximation will *always* satisfy Helmholtz reciprocity (trivially, since the representation is symmetric in $\hat{\omega}_o$ and $\hat{\omega}_i$) which is required for physically-based BRDFs. The representation only requires two texture maps, thus minimizing storage costs and permitting many BRDFs to be used in a scene even on low-end systems. Although three factors are required in total, only two factors depend on the direction of the light source, which simplifies implementation on systems that permit only two texture lookups in one pass when multiple light sources are desired. The parameterization is also easy to compute, significantly more so than the parameterization in Kautz and McCool [17], yet the surface-relative parameters are easy to interpolate without singularities. Surfaces dominated by shadowing and masking, as well as those dominated by microfacet distributions, should both be equally well-served by this parameterization, and we feel this covers a large class of interesting materials. Finally, as we discuss in the conclusions, this parameterization is potentially useful for generating random samples of the BRDF for Monte Carlo evaluation of local lighting in offline rendering.

3.3 Data Constraints

Given samples of f, we now need to devise a numerical technique to find p and q. After transforming both sides of Equation 9 by the logarithmic homomorphism, we obtain the following:

$$\tilde{f}(\hat{\omega}_o, \hat{\omega}_i) \quad \approx \quad \tilde{p}(\hat{\omega}_o) + \tilde{q}(\hat{\mathbf{h}}_i) + \tilde{p}(\hat{\omega}_i). \tag{10}$$

Suppose we have a set of data samples $f[k]$ of a BRDF (with no particular structure required) and a set of parameter locations $\hat{\omega}_o[k]$ and $\hat{\omega}_i[k]$ for each of those samples. For each data point we can compute $\hat{\mathbf{h}}[k]$ and then set up a linear constraint equation for it. We will assume that the function $\pi_j \in \{\pi_o, \pi_h, \pi_i\}$ will be used to map directly from $(\hat{\omega}_o, \hat{\omega}_i)$ to the corresponding texture map coordinate (using, for instance, a parabolic map represention of a function over a hemisphere; see Section 4.1). Let π_j^u be the u component of the projection π_j and let π_j^v be the v component.

To find p and q, we will set up a system of linear constraints relating each sample $f[k]$ to certain texels in p and q, and then solve the resulting linear system in a minimum-residual sense.

Suppose we have computed the projection of a sample into a texture map with coordinates $(u, v) \in \mathbb{R}^2$. Texels for $\tilde{p}$ and $\tilde{q}$ will be laid out on a finite-resolution grid and must be interpolated to get a function over $\mathbb{R}^2$. If we assume texels are located at integers, we can compute bilinear weighting factors to get subpixel precision for the location of the projection $(u_j, v_j) = \pi_j(\hat{\omega}_o, \hat{\omega}_i)$:

$$\begin{aligned} (U_j, V_j) &= (\lfloor u_j \rfloor, \lfloor v_j \rfloor), & (11)\\ \left(\alpha_j^u, \alpha_j^v\right) &= (u_j - U_j, v_j - V_j), & (12)\\ \left(\beta_j^u, \beta_j^v\right) &= \left(1 - \alpha_j^u, 1 - \alpha_j^v\right). & (13) \end{aligned}$$

Given the appropriate bilinear weights for the projection of the sample into the parameter space of each term, and letting π_o, π_h, and π_i be the appropriate projections for the individual terms in Equation 10, a constraint equation for each BRDF sample can be set up in the following form (the index k of the BRDF sample and the parameter vectors has been suppressed for clarity):

$$\begin{aligned} \tilde{f} \;=\;& \beta_o^u \beta_o^v \;\; \tilde{p}[U_o, V_o] \\ +\;& \alpha_o^u \beta_o^v \;\; \tilde{p}[U_o + 1, V_o] \\ +\;& \alpha_o^u \alpha_o^v \;\; \tilde{p}[U_o + 1, V_o + 1] \\ +\;& \beta_o^u \alpha_o^v \;\; \tilde{p}[U_o, V_o + 1] \\ +\;& \beta_h^u \beta_h^v \;\; \tilde{q}[U_h, V_h] \\ +\;& \alpha_h^u \beta_h^v \;\; \tilde{q}[U_h + 1, V_h] \\ +\;& \alpha_h^u \alpha_h^v \;\; \tilde{q}[U_h + 1, V_h + 1] \\ +\;& \beta_h^u \alpha_h^v \;\; \tilde{q}[U_h, V_h + 1] \\ +\;& \beta_i^u \beta_i^v \;\; \tilde{p}[U_i, V_i] \\ +\;& \alpha_i^u \beta_i^v \;\; \tilde{p}[U_i + 1, V_i] \\ +\;& \alpha_i^u \alpha_i^v \;\; \tilde{p}[U_i + 1, V_i + 1] \\ +\;& \beta_i^u \alpha_i^v \;\; \tilde{p}[U_i, V_i + 1]. \end{aligned} \tag{14}$$

If the two projections π_i and π_o map a BRDF data point to the same texels in the texture map for $\tilde{p}$, the corresponding coefficients should be summed.

Bilinear interpolation in log space does *not* correspond exactly to the bilinear interpolation that will be performed later between texels during hardware rendering. We do it here *only* to obtain subpixel precision for the projected location of the BRDF sample.

All constraints can be placed in a matrix A, split into the coefficients on $\tilde{p}$ and on $\tilde{q}$, with the texels in $\tilde{p}$ and $\tilde{q}$ unpacked into a single column vector:

$$\left[\; \tilde{f} \;\right] \;=\; \left[\; \mathsf{A}_p \quad \mathsf{A}_q \;\right] \left[\; \begin{array}{c} \tilde{p} \\ \tilde{q} \end{array} \;\right]. \tag{15}$$

We will write this compactly as

$$\tilde{\mathbf{f}} \;=\; \mathsf{A}\tilde{\mathbf{x}} \tag{16}$$

This linear system may be overconstrained or underconstrained. Either way, we can solve it in a minimum-residual sense in a number of ways.

One approach to solving this system in the least squares sense would be to use a pseudoinverse computed with an SVD [27] (this is different from using the SVD to compute the factorization directly). However, computing the pseudoinverse is not practical, since the matrix is quite large: $K \times 2M$, where K is the number of samples of the BRDF we have and M is the number of samples in each output texture map, which we assume are both the same resolution.

While the matrix A is large, it is also quite sparse. Taking bilinear interpolation into account, there are a maximum of 12 non-zero values in each row of A for our target three-factor parameterization. We solve Equation 16 using an iterative technique described in Section 3.6 that only requires the computation of matrix-vector products. With this approach, we need not even store the matrix explicitly.

Another issue is that we may not have enough data-fitting equations to directly constrain all texels. A typical problem would be gaps in measured BRDF data, since it is hard to measure large incident and exitant polar elevation angles and certain combinations of incident and exitant directions (such as direct retroreflection, with the light and view in the same direction). We might also use a texture-map parameterization that has texels not projected onto by any possible combination of incident and exitant directions (in fact *all* of the hemispherical texture map parameterizations we use have this property). In the latter case, we still need to set unused texels to "reasonable" values to avoid problems during interpolation, particularly if per-vertex normals are used to control shading.

In Section 3.4 we describe a regularization technique that ensures that every texel is constrained and which permits control over the smoothness of the solution. Our regularization technique also automatically interpolates over gaps in the BRDF data and extrapolates out to unused parts of the texture map, finding the smoothest solution consistent with the data. This both fills in holes and avoids interpolation problems during rendering.

We conclude this section with a two-dimensional example. Consider the left image in Figure 2. This is a 2D Perlin-noise function that we wish to approximate with a product of 1D functions. In this case, we use the three projection functions $\pi_x : (x,y) \mapsto x$, $\pi_y : (x,y) \mapsto y$, and $\pi_{x-y} : (x,y) \mapsto x-y$. Taking the logarithm of the input image, setting up constraint equations for each input pixel, solving the resulting linear system in a minimum-residual sense, and then exponentiating this solution results in the two and three-factor approximations shown. Note that the two-factor approximation is set up and solved separately from the three-factor approximation.

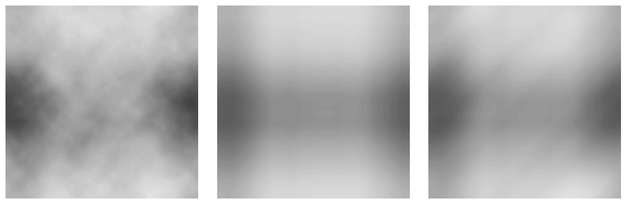
Figure 2: *A 2D example. Left to right: input function, separable approximation with two 1D factors (based on the projections π_x and π_y), separable approximation with three 1D factors (based on the projections π_x, π_y, and π_{x-y}).*

3.4 Smoothness Constraints

Regularization adds additional equations to the system both to make sure that every texel has an equation that constrains it, and to control the smoothness of the solution. To accomplish these dual goals, we will add a constraint that equates the Laplacian of the BRDF reconstruction everywhere to zero. By weighting these new constraints by a factor λ, we can control how important smoothness is to the result. The smoothness constraint also encourages the solution to interpolate over any gaps in the data without introducing discontinuities at the boundaries of the gap.

Ideally, we would want to apply the smoothing operator to $\tilde{f}$. Unfortunately, evaluating a 4D Laplacian operator would be expensive. Instead, we can project the operator down to the texture spaces and apply it in 2D. The linear projection of a Laplacian operator is a Laplacian operator, so this is exactly equivalent to applying the Laplacian operator in the 4D space. In practice, we don't bother doing a projection (and ignore the fact that the projection operators may be nonlinear, and that the projections are not orthogonal to each other) but just apply the 2D Laplacian masks shown in Figure 3 to each pixel in the textures $\tilde{p}$ and $\tilde{q}$, then set the result to zero. Let each row of L_p and L_q correspond to the application of

$$\begin{array}{ccc}
\boxed{\begin{array}{rr} \mathbf{2} & -1 \\ -1 & 0 \end{array}} &
\boxed{\begin{array}{rrr} -1 & \mathbf{3} & -1 \\ 0 & -1 & 0 \end{array}} &
\boxed{\begin{array}{rr} -1 & \mathbf{2} \\ 0 & -1 \end{array}} \\
\boxed{\begin{array}{rr} -1 & 0 \\ \mathbf{3} & -1 \\ -1 & 0 \end{array}} &
\boxed{\begin{array}{rrr} 0 & -1 & 0 \\ -1 & \mathbf{4} & -1 \\ 0 & -1 & 0 \end{array}} &
\boxed{\begin{array}{rr} 0 & -1 \\ -1 & \mathbf{3} \\ 0 & -1 \end{array}} \\
\boxed{\begin{array}{rr} -1 & 0 \\ \mathbf{2} & -1 \end{array}} &
\boxed{\begin{array}{rrr} 0 & -1 & 0 \\ -1 & \mathbf{3} & -1 \end{array}} &
\boxed{\begin{array}{rr} 0 & -1 \\ -1 & \mathbf{2} \end{array}}
\end{array}$$

Figure 3: Masks used to constrain the smoothness of the reconstruction. The outputs of these masks (position shown in boldface), multiplied by the smoothness factor λ, are set equal to zero.

the Laplacian operator to every pixel of the corresponding texture, including those not mapped to by any BRDF sample. Then the augmented system of equations is:

$$\begin{bmatrix} \tilde{f} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathsf{A}_p & \mathsf{A}_q \\ \lambda\mathsf{L}_p & \mathbf{0} \\ \mathbf{0} & \lambda\mathsf{L}_q \end{bmatrix} \begin{bmatrix} \tilde{p} \\ \tilde{q} \end{bmatrix}. \qquad (17)$$

We can write this compactly as

$$\tilde{\mathbf{g}} = \mathsf{B}_\lambda \tilde{\mathbf{x}} \qquad (18)$$

Note that setting $\lambda = 0$ and solving this system using a minimum-residual solver will solve the original set of equations, although inefficiently. A larger λ will result in a smoother solution.

3.5 Weighting

If further control over the relative importance of data fitting and smoothness is desired, individual data constraint equations (i.e. both the BRDF data point and the coefficients that relate texture values to it) can be scaled up or down.

Let $\mathsf{W} = \mathrm{diag}\,(w[k])$ be a diagonal matrix of such weights; then the following system gives independent control of data point weighting and smoothness:

$$\begin{bmatrix} \mathsf{W}\tilde{f} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathsf{WA}_p & \mathsf{WA}_q \\ \lambda\mathsf{L}_p & \mathbf{0} \\ \mathbf{0} & \lambda\mathsf{L}_q \end{bmatrix} \begin{bmatrix} \tilde{p} \\ \tilde{q} \end{bmatrix} \qquad (19)$$

We do not consider weighting further in this paper, instead we have focused on solving the unweighted problem. However, weighting of data points should be useful if measurement or simulation error can be estimated for each data point. In that case, weighting each data point by the reciprocal of its expected variance would be a good strategy.

3.6 Iterative Solution of Sparse System

To find a factorization it is necessary to solve the sparse, overconstrained linear system given in Equation 18. Ideally we want to find a solution that minimizes the residual error.

Since the matrix is large and sparse, we used an iterative technique that avoids modifying the matrix and instead depends only on matrix-vector products. To solve our systems, we used the quasi-minimal residual (QMR) algorithm [11], a modified version of the conjugate-gradient algorithm, as implemented in the publically-available IML++ library from the NIST.

Although it is not necessary, for simplicity we explicitly represented the sparse matrix using SparseLib++, which comes with IML++. Since the coefficient matrix depends only on the sampling geometry and is never modified, a more efficient implementation would form the required matrix-vector products on the fly. Such an algorithm would require storage only for the original BRDF data and the result.

Two other techniques can be used to reduce computation time in practice: starting with a good initial guess, and solving the problem at a sequence of increasing scales. Our solver implements both these techniques. An initial guess can be obtained by using simple averaging during projection. We also solved the system over a sequence of increasingly higher resolution grids progressing from coarse to fine, bilinearly interpolating each stage's solution up to get an initial guess for the next finer grid.

4 Results

Section 4.1 discusses some issues that arise in hardware accelerated rendering implementations. Section 4.2 presents some examples and Section 4.3 analyses the error in the factored approximation of some test BRDFs.

4.1 Hardware Accelerated Rendering

Graphics accelerators are basically specialized SIMD/vector computers capable of general (but precision-limited) numerical computations [8, 23, 31]. To map our representation onto an accelerator, we need to understand both the computation required and the capabilities of the target. For our target, we will consider the hardware abstraction presented by standard OpenGL (with multitexturing) as well as the computational model presented by the NVIDIA register combiner extension. Our implementation also uses the NVIDIA vertex shader extension [22].

To store the factors, texture map representations parameterized by a direction on the hemisphere are needed that permit good interpolation. We recommend performing the factorization using a parabolic mapping [16] of the relevant unit vectors. Parabolic maps work on all hardware accelerators that correctly implement projective texture coordinate transformation and interpolation, but the resulting 2D images can be resampled into cube maps if the hardware supports them.

Substituting our BRDF approximation into the general point source lighting model in Equation 2, we obtain

$$L_o(\hat{\omega}, \underline{\mathbf{x}}) \quad = \quad \alpha p'(\hat{\omega}_o) \sum_{\ell=1}^{N} q'(\hat{\mathbf{h}}^\ell) p'(\hat{\omega}_i^\ell) [\hat{\omega}_i^\ell \cdot \hat{\mathbf{n}}] \, \frac{I_\ell}{r_\ell^2}, \quad (20)$$

where p' and q' are normalized versions of the functions p and q and α is a factor to correct for this normalization.

To map this onto standard OpenGL, we can implement the product inside the summation with compositing (combining the $p'(\hat{\omega}_i^\ell)$ factor with standard Lambertian lighting on one pass, then multiplying by $q'(\hat{\mathbf{h}}^\ell)$ on a following pass), compute the summation over the light sources using the accumulation buffer, then complete the final multiplication by $\alpha p'(\hat{\omega}_o)$ with another rendering and compositing pass. In total, such an implementation will require rendering the geometry $2N+1$ times. On standard OpenGL, we will also have to set up the texture coordinates on the host by computing $\hat{\omega}_o$, then $\hat{\mathbf{h}}^\ell$ and $\hat{\omega}_i^\ell$ for each light source (in model coordinates), then finally projecting $\hat{\omega}_o$, $\hat{\mathbf{h}}^\ell$, and $\hat{\omega}_i^\ell$ onto the local surface frame defined at every vertex of each model to be rendered.

Some savings can be made in special circumstances. When both the viewer and the ℓth light source are at infinity and fixed, $\hat{\omega}_o$, $\hat{\omega}_i^\ell$, and the half-vector $\hat{\mathbf{h}}^\ell$ are global constants. However, it is still necessary to project these vectors into the local surface frames of objects that move relative to the viewer and the light source.

On accelerators that support multitexturing with at least two texture units, we can do the multiplications inside the summation in Equation 20 in a single pass and accumulate the results in the framebuffer using compositing. This is possible, for instance, on the GeForce 2. A final compositing pass is then needed to multiply by the common view-dependent factor. Besides requiring only $N+1$ passes, this approach avoids the use of the accumulation buffer.

In next generation systems that support additional multitexturing units, we will be able to do several light sources and/or materials in a single pass, then accumulate multiple passes into the framebuffer using compositing operations. We can apply the view-dependent term either at the end or during each pass. If we assume the latter (which should better preserve precision and is more flexible, potentially supporting, for instance, per-pixel masking of multiple materials onto one surface primitive) and assume we have enough per-pixel shading resources to compute R light sources in a single pass, we will need only N/R passes.

The GeForce 3 permits simple computations to be performed at every vertex by downloading a program to a vertex shader unit [22, 28]. This capability has been exploited in our implementation to compute the required texture coordinates from $\hat{\omega}_o$, $\hat{\mathbf{h}}^\ell$, and $\hat{\omega}_i^\ell$ on the graphics accelerator, so vertices and normals can be stored in vertex arrays or display lists and the host can be completely freed from the need to update texture coordinates. Using the three lookup factorization above and an extra specular map (four texture lookups total), the performance of a single-light BRDF-based reflectance model on a GeForce 3 is just less than half that of rendering the same model with simple per-vertex diffuse lighting and Gouraud shading.

Precision and dynamic range are potential problems. On current accelerators BRDFs with large dynamic ranges may be hard to approximate without contouring. This is yet another reason to clamp the specular peak before factorization and approximate it separately. However, on future systems supporting full dependent texturing, we could store the texture maps in logarithmic form and perform the multiplications using addition in logarithmic space. After $\log f$ has been reconstructed, it can be exponentiated using another lookup table that can also include a "soft clip". This will increase accuracy near zero yet will avoid undue clamping of intermediate results. Floating point numbers can be considered piecewise-linear approximations to logarithms [4], so this will work in general for extending dynamic range and low-end precision on fixed-point hardware.

4.2 Examples

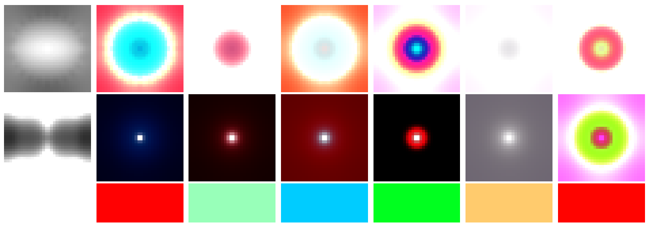

Figure 4: *Texture maps for various BRDFs at* 32×32 *resolution, as used for hardware-accelerated rendering using the parabolic representation. Top to bottom:* p'*,* q'*, and the normalization correction colour* α*. Materials are given in the same order as Figure 1.*

Some examples of texture maps generated using the homomorphic factorization algorithm are shown in Figure 4. For these examples, the specular highlight was removed (by clamping) for very shiny BRDFs, like cayman and mystique, and the factored representation used only for the directional diffuse component. The remaining specular reflectance was accounted for using an additional map dependent on $\hat{\mathbf{h}}$, although an environment map and/or a more complex factorization (including, for instance, a Fresnel term) could have been used instead.

We have implemented demonstration programs on the NVIDIA GeForce 2 and GeForce 3. On the GeForce 3, using both local lights and viewers (but no attenuation), we have obtained rates of over 5.1Mtris/second using triangle strips stored in vertex arrays. We could probably improve on this significantly by using compiled display lists and/or specializations of our vertex shader for directional lights and viewers. For comparison, when using standard OpenGL lighting (diffuse only) with our test models we obtain a rate of 10.6Mtris/second, or 13.3Mtris/second if we use optimized vertex-shader diffuse lighting. Images generated using the texture maps in Figure 4 are shown in Figure 1 and Figure 5.

Our approximation is visually compared with a rendering using an actual BRDF in Figure 5. We used for this error analysis the analytic Poulin-Fournier anisotropic reflectance model [26], which does not take multiple scattering into account but which does account for different effective normal distributions due to shadowing and masking.

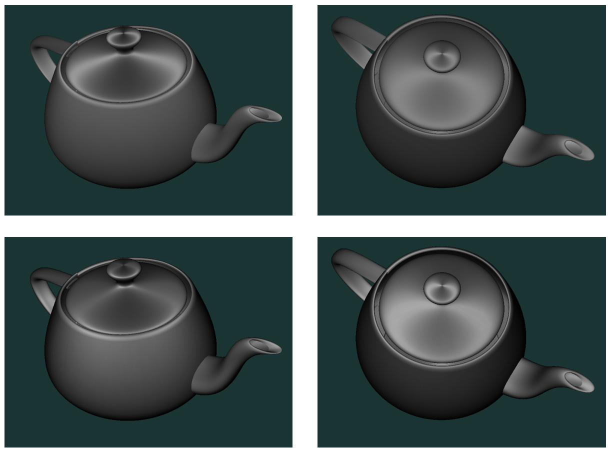

Figure 5: *Top: using actual BRDF values at the vertices of a highly tesselated model rendered with Gouraud shading. Bottom: approximation using the $p(\hat{\omega}_o)q(\hat{\mathbf{h}})p(\hat{\omega}_i)$ factorization.*

4.3 Error Analysis

Error measures were computed by subtracting the values of the original BRDF from the approximation evaluated at the same sample locations. For our tests, we sampled the original BRDF in such a way that sample positions were evenly distributed over both incoming and outgoing hemispheres. Results are summarized in Table 1. We have computed several error metrics, including both absolute, square, and relative error metrics.

The maximum error can be quite high. We expected this, since the logarithmic transformation and the smoothing will both tend to discount the largest peak in the data. As mentioned before, this technique seems to be best for the directional-diffuse components of BRDFs; it tends to damp out specular highlights and/or give them the same color as the base surface.

Compression ratios are also given. These assume that a BRDF sample takes up the same amount of space as a texel, and that textures are stored as square images using parabolic maps (which wastes some space). The base sampling rate given would be required for rendering from a 4D table. Factors could also be stored in compressed form if the hardware permits it (for instance, using JPEG compression). Under the parameterization we tested, isotropic BRDFs result in radially symmetric texture maps, so more compression could be obtained if only the radial variation of these maps were stored (we can also exploit this symmetry during factorization itself).

BRDF:	Cyl16a	Cyl16	Cyl32
Samples	57,600	57,600	57,600
Resolution	$2 \times 16 \times 16$	$2 \times 16 \times 16$	$2 \times 32 \times 32$
Compression	113:1	113:1	28:1
Avg BRDF	0.3855	0.3855	0.3855
Avg Approx	0.3463	0.3905	0.3818
Bias	-0.039	0.0050	-0.0037
Avg Abs Error	0.1089	0.06350	0.06145
Max Abs Error	1.0628	0.9385	0.9168
Avg Rel Error	0.4446	0.2484	0.2334
Max Rel Error	20.91	24.32	24.38
RMS Error	0.1481	0.09105	0.09002
SNR	9.08dB	13.30dB	13.40dB

Table 1: *Error analysis for the anisotropic BRDF approximation shown in 5. Cyl16a is a naïve (averaged projection) approximation, Cyl16 and Cyl32 are approximations found via homomorphic factorization.*

5 Conclusions

An approximate BRDF representation has been presented that is suitable for real-time hardware acceleration of per-pixel local illumination of anisotropic materials from a point source. The representation of a BRDF can be stored in two 2D moderate-resolution texture maps or one cube map, and so a scene can contain models with many different BRDFs. The factorization algorithm has several advantages over the singular value decomposition, including n-way factorization, automatic gap-filling, direct compatibility with sparse, scattered BRDF data, minimization of relative rather than absolute error, support for arbitrary parameterizations, and production of purely positive factors.

Some other applications and extensions are possible.

First of all, this representation is potentially useful for Monte-Carlo importance sampling in offline renderers. When an incoming direction is fixed, either $q(\hat{\mathbf{h}})$ or $p(\hat{\omega})$ can be used as a probability density to efficiently generate outgoing samples in directions in which the BRDF is large. This can be used to reduce variance even if the true BRDF is then evaluated to weight the sample relative to the factored approximation. See Ashikhmin and Shirley for a similar technique using an analytic model [2]. They also discuss how to correct the measure for samples generated with respect to $q(\hat{\mathbf{h}})$.

Secondly, since the technique extends naturally to representations with more than three factors, it might be possible to find other projection functions that better capture other reflectance phenomena than shadowing/masking and microfacet distributions. A 1D "Fresnel" term dependent on $(\hat{\omega} \cdot \hat{\mathbf{h}})$ would be particularly useful.

Third, it should be noted that if bump-mapping or normal mapping is used, or a very fine mesh of polygons with small normal deviations, we don't have to worry so much about interpolation between vertices and can use maps that depend on 1D parameters. One particularly interesting parameterization for such a purpose would be

$$f(\hat{\omega}_o, \hat{\omega}_i) \quad \approx \quad p(\hat{\mathbf{h}} \cdot \hat{\omega}, \hat{\mathbf{h}} \cdot \hat{\mathbf{n}})\, q(\hat{\omega}_i \cdot \hat{\mathbf{n}}, \hat{\omega}_o \cdot \hat{\mathbf{n}}). \qquad (21)$$

Parameterizing a single map with $\hat{\omega}_i \cdot \hat{\mathbf{h}}$ and $\hat{\omega}_o \cdot \hat{\mathbf{h}}$ permits correlated effects between these two parameters to be captured when $\hat{\omega}_i \approx \hat{\omega}_o$.

This representation would require only two texture maps and two lookups, both lookups dependent on parameters which are dot products of the normal (looked up previously in a normal map) with vectors that could be interpolated from vertices. Such a computation should be well within the capabilities of next-generation hardware supporting dependent texturing. Unfortunately, this parameterization is only appropriate for isotropic BRDFs. “Frame mapping”, using anisotropic BRDFs, could however be accomplished using the original parameterization along with multiple maps for normals and tangents at every point on the surface (adding significantly to the cost, but not entirely unreasonable).

Finally, a host of other shading techniques can be employed, such as rendering a “reflectance ball” to get an environment map that can be used in turn for more efficient rendering of complex geometry with many light sources (assuming an orthographic view approximation and isotropic BRDFs), using alpha textures to blend between multiple materials on the same surface (again well within the capabilities of next-generation hardware, even in a single pass, even when combined with normal mapping), and use of environment maps for the specular term.

Acknowledgements

This research was sponsored by research grants from the National Science and Engineering Research Council of Canada (NSERC) and the Centre for Information Technology of Ontario (CITO). We also gratefully acknowledge NVIDIA’s hardware donations and early access to prototypes of their next-generation systems. Our original real-time prototype was based on code originally written by Chris Wynn and Jan Kautz. The anisotropic Poulin-Fournier model was evaluated using code (bv) written by Szymon Rusinkiewicz. Measured BRDF datasets identified by “Cornell” were obtained from the Cornell University Program of Computer Graphics. Measured BRDF datasets identified by “CURET” were obtained from the Columbia-Utrecht Reflectance and Texture Database. Code for the QMR algorithm and the IML++ and SparseLib++ packages were obtained from the National Institute of Standards and Technology (NIST) in the United States.

References

[1] M. Ashikhmin, S. Premože, and P. Shirley. A Microfacet-Based BRDF Generator. *SIGGRAPH*, pp. 65–74, 2000.

[2] M. Ashikhmin and P. Shirley. An anisotropic Phong BRDF model. *Journal of Graphics Tools*, 5(2), pp. 25–32, 2000.

[3] H. H. Barrett and W. Swindell. *Radiological Imaging: The Theory of Image Formation, Detection, and Processing*. Academic Press, 1981.

[4] J. Blinn. Jim Blinn’s Corner: Floating-Point Tricks. *IEEE Computer Graphics & Applications*, 17(4), July–August 1997.

[5] B. Cabral, N. Max, and R. Springmeyer. Bidirectional Reflection Functions From Surface Bump Maps. *SIGGRAPH*, pp. 273–281, 1987.

[6] B. Cabral, M. Olano, and P. Nemec. Reflection Space Image Based Rendering. *SIGGRAPH*, pp. 165–170, 1999.

[7] Commission Internationale de l’Éclairage. *CIE Colorimetry Standard*. Technical report, Central Bureau of the CIE, 1986.

[8] P. Diefenbach. *Pipeline Rendering: Interaction and Realism through Hardware-Based Multi-pass Rendering*. PhD thesis, Department of Computer and Information Science, 1996.

[9] C. W. Everitt. High-Quality, Hardware-Accelerated Per-Pixel Illumination for Consumer Class OpenGL Hardware. Master’s thesis, Dept. of Computational Engineering, Mississippi State University, 2000.

[10] A. Fournier. Separating Reflection Functions for Linear Radiosity. *Rendering Techniques ’95 (Eurographics Workshop on Rendering)*, pp. 383–392. Springer, 1995.

[11] R. W. Freund and N. M. Nachtigal. A Quasi-Minimal Residual Method for Non-Hermitian Linear Systems. *Numerische Mathematik*, 60, pp. 315–339, 1991.

[12] J. Gondek, G. Meyer, and J. Newman. Wavelength Dependent Reflectance Functions. *SIGGRAPH*, pp. 213–220, 1994.

[13] R. Hall. *Illumination and Color in Computer Generated Imagery*. Springer-Verlag, 1989.

[14] W. Heidrich and H.-P. Seidel. Realistic, Hardware-Accelerated Shading and Lighting. *SIGGRAPH*, pp. 171–178, 1999.

[15] W. Heidrich, K. Daubert, J. Kautz, and H.-P. Seidel. Illuminating Micro Geometry Based on Precomputed Visibility. *SIGGRAPH*, pp. 455–464, 2000.

[16] W. Heidrich and H.-P. Seidel. View-Independent Environment Maps. *Eurographics/SIGGRAPH Workshop on Graphics Hardware*, pp. 39–45, 1998.

[17] J. Kautz and M. D. McCool. Interactive Rendering with Arbitrary BRDFs using Separable Approximations. *Rendering Techniques ’99 (Eurographics Workshop on Rendering)*, pp. 281–292, Springer, 1999.

[18] J. Kautz and M. D. McCool. Approximation of Glossy Reflection with Prefiltered Environment Maps. *Graphics Interface*, pp. 119-126, 2000.

[19] J. Kautz, P.-P. Vázquez, W. Heidrich, and H.-P. Seidel. A Unified Approach to Prefiltered Environment Maps. *Rendering Techniques ’00 (Eurographics Workshop on Rendering)*, pp. 185–196. Springer, 2000.

[20] J. Koenderink, A. van Doorn, and M. Stavridi. Bidirectional Reflection Distribution Function Expressed in Terms of Surface Scattering Modes. *European Conference on Computer Vision*, pp. 28–39, 1996.

[21] E. Lafortune, S.-C. Foo, K. Torrance, and D. Greenberg. Non-Linear Approximation of Reflectance Functions. *SIGGRAPH*, pp. 117–126, 1997.

[22] E. Lindholm, M. Kilgard, and H. Moreton. User-Programmable Vertex Engine. *SIGGRAPH*, 2001.

[23] M. Peercy, M. Olano, J. Airey, and J. Ungar. Interactive Multi-Pass Programmable Shading. *SIGGRAPH*, pp. 425–432, 2000.

[24] M. Pharr and P. Hanrahan. Monte Carlo Evaluation of Non-Linear Scattering Equations for Subsurface Reflection. *SIGGRAPH*, pp. 75–84, 2000.

[25] B.-T. Phong. Illumination for Computer Generated Pictures. *Communications of the ACM*, 18(6), pp. 311–317, June 1975.

[26] P. Poulin and A. Fournier. A Model for Anisotropic Reflection. *SIGGRAPH*, pp. 273–282, 1990.

[27] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992.

[28] K. Proudfoot, W. R. Mark, S. Tzvetkov, and P. Hanrahan. A Real-Time Procedural Shading System for Programmable Graphics Hardware . *SIGGRAPH*, 2001.

[29] P. Schröder and W. Sweldens. Spherical Wavelets: Efficiently Representing Functions on the Sphere. *SIGGRAPH*, pp. 161–172, 1995.

[30] W. Stürzlinger and R. Bastos. Interactive Rendering of Globally Illuminated Glossy Scenes. *Rendering Techniques ’97 (Eurographics Workshop on Rendering)*, pp. 93–102. Springer, 1997.

[31] C. Trendall and A. J. Stewart. General Calculations using Graphics Hardware, with Applications to Interactive Caustics. *Rendering Techniques ’00 (Eurographics Workshop on Rendering)*, pp. 287–298. Springer, 2000.

[32] B. Walter, G. Alppay, E. LaFortune, S. Fernandez, and D. Greenberg. Fitting Virtual Lights for Non-diffuse Walkthroughs. *SIGGRAPH*, pp. 45–48, 1997.

[33] G. Ward. Measuring and Modeling Anisotropic Reflection. *SIGGRAPH*, pp. 265–272, 1992.