Real-Time Fur with Precomputed Radiance Transfer

John W. Kloetzli, Jr.*
UMBC

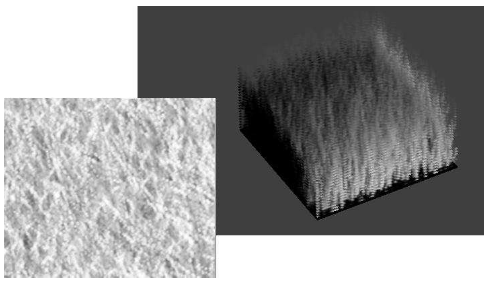
Figure 1: Left: Dense, unlit fur (color set by depth) 55000 hairs, 32 shells. Right: Final render of fur in real-time with lighting.

Abstract

This paper introduces Precomputed Radiance Transfer (PRT) to shell textures in the context of real-time fur rendering. PRT is a method which allows static objects to have global illumination effects such as self shadowing and soft shadows while being rendered in real-time. This is done by precomputing radiance on the surface in a special basis that is chosen to allow reconstruction of correct illumination in arbitrary lighting environments. Shell textures is a technique for rendering complex surface geometry in real-time through the use of concentric 3D *rings* or *shells* around a model. The shells are transparent everywhere except at the intersection of the shell and the microgeometry that is being rendered. We novelly apply these two techniques to the problem of rendering fur to produce real-time rendering of fur with global illumination.

Keywords: real-time, fur, precomputed radiance transfer, PRT, microgeometry, microsurfaces

1 Introduction

Rendering microsurfaces is a difficult task in computer graphics. Because microsurfaces are by definition very high frequency geometry, traditional rasterization or ray tracing techniques bog down to the point of uselesness or are plagued with terrible aliasing artifacts. Surface shading techniques are also not suited to the task because microsurfaces, although very small, are still geometrically visible to the naked eye, which lighting equations alone are unable to capture.

Fur is a perfect example of a microsurface, containing hundreds or thousands of hairs that are very small but certainly individually visible. Still, we need to render fur if we intend to have realistic animals in computer graphics. In addition, we need a very fast way to render it if we intend said animals to be in an interactive application such as a game.

Some of the first techniques used to render fur were based on 3D textures. Kajiya and Kay [1989] described a ray-tracing method which stored microsurface data in a volume and modeled light scattering and attenuation along each ray through the volume. This method produced excellent results, but was far from real-time. In addition, close inspection of the fur revealed what they described as the "painter's illusion": blotches of color that appear to be microgeometry from a distance but up close separate into meaningless blobs. *Hypertexture* [Perlin and Hoffert 1989] is another volume method, but instead of capturing surfaces they record densities between 0 and 1. This gives the effect of 'soft' objects, with which, by modulating these densities using different functions, they could produce a variety of effects, including fur. Although Hypertexture can model many different things effectively, it doesn't produce as realistic fur as other methods. There are implementations of Hypertexture that run in realtime on todays graphics hardware, but none of them attempt to compute inter-object shadows that are needed for realistic microsurfaces.

Another method for fur rendering was proposed by Goldman [1997], who was working on rendering fur for the Disney film *101 Dalmations*. His basic assumption was that the camera was never going to get close enought to a furry object for the geometric properties of the fur to become distinct from the lighting properties. Since it is impossible to explicitly see any geometry in this application, none is modeled. His method described light reflections from coats of fur using a fairly complex (although much less than geometric approaches) purely stochastic lighting model, and thus isn't suitable for applications that want to display geometric fur.

Real-time fur methods have also been previously explored. Gelder and Wilhelms [1997] explore simply drawing lines using standard graphics workstations. Although they didn't get very realistic results, they did show that it was possible to render fur at interactive rates.

The second real-time technique of interest here was introduced in 2001 by Lengyel et al [2001]. The method creates concentric *shells* around the model being rendered, each shell displaying a different part of a volume texture. The shells are transparent except for the precomputed intersection of the shell and the fur volume that they created. When rendered together they create a very convincing furry surface. While this works well when viewed from above, since the shells overlap, creating the illusion of continuous geometry, it doesn't work for vertices near the silhouette of the object since the gaps between adjacent shells become apparent. To remedy this, they add small textures rendered normal to the surface across all the edges in the model, which they fade in as the edge approaches the silhouette. These *fin textures* fill in the gaps in the shell textures on the silhouette, creating a complete geometric render of fur. This technique can be viewed as a form of rendering the Kajiya and Kay [1989] fur model in real-time, but using a simple

*e-mail: jk3@umbc.edu

ambient/diffuse/specular lighting scheme that doesn't capture self shadowing. Real-time lighting of microsurfaces is a difficult problem, but one that must be solved to even approach realistic results. As discussed by Lokovic and Veach [2000], rendering realistic microgeometry such as fur depends heavily on inter-object shadows, namely hair-to-hair shadows.

One method that can be used to perform inter-object shadows with complicated geometry is Deep Shadow Maps [Lokovic and Veach 2000]. Traditional shadow mapping basically renders the scene from the position of each light, storing the depth value for each render into depth-maps. The scene can then be re-rendered from the camera point of view, looking at the position of the first obeject hit for each fragment. If the distance between that object and the light is greater than the corresponding depth-map value, then the surface isn't the first one hit by the light and therefore is in shadow. If the depths match, then the object is the first one seen by the light and should be lit. However, traditional shadow maps are prone to aliasing for high-frequency geometry (unless prohibitively high resolutions are used) and hence are not suitable for microgeometry. Deep shadow maps, on the other hand, store a function at each texel. This function defines the fractional visibility of the surface at all depths, which can be used to render fast self-shadowing of objects. One downside of this method is that the functions must be recomputed if the lighting changes, making deep shadow maps most useful in static lighting situations.

Since asthetically pleasing fur illumination can be obtained in offline rendering [Kajiya and Kay 1989], one possible approach is to precompute the lighting at different points on the object and recreate it at run time to approximate run-time lighting. Precomputed Radiance Transfer [Sloan et al. 2002] allows for this through the use of basis functions designed to efficiently encode the lighting across an object for later reconstruction. Several different basis functions have been explored, including Spherical Harmonics (SH)[Sloan et al. 2002] [Sloan et al. 2003b], Zonal Harmonics (ZH)[Sloan et al. 2005], and Harr Wavelets (HW) [Wang et al. 2005] [Liu et al. 2004]. While SH and ZH are limited to low-frequency lighting, they are easily rotatable and require small texture sizes. The HW basis functions are much more numerous, which allows them to represent all-frequency lighting, but also require more texture space and run-time processing.

Many different variations of PRT have been explored. It has been used to recreate diffuse [Sloan et al. 2002] and specular [Sloan et al. 2002] [Liu et al. 2004] lighting, as well model subsurface scattering [Wang et al. 2005] in real-time. Multiple scales of PRT [Sloan et al. 2003b] can be used to create high-frequency local effects in addition to complete inter-model lighting. The ZH basis functions can be used to represent easily rotated lighting, leading to PRT for deformable objects [Sloan et al. 2005], which is otherwise impossible (since inter-object lighting effects are being precomputed). Usual implementations store lighting on a per-vertex basis, but light-cloud representations [Kristensen et al. 2005] have also been used.

2 Method

Since the principal type of illumination that we are trying to capture is hair-to-hair shadows in a furry surface, we will be using local PRT on a single fur texture. We use the the Spherical Harmonics as basis functions for our application because the procecss of rendering the volume will tend to smooth out any high frequency lighting.

Our method is to combine the successful real-time fur rendering of Leyngel et al [2001] with PRT lighting. Our approach can be summarized as follows:

1. Create a realistic fur texture
2. Compute lighting of the fur using an off-line technique
3. Convert the lighting into a PRT basis
4. Reconstruct the lighting in real-time

2.1 Creating Fur

Several things are necessary to create a good fur model. First, hairs need to start out thick and become thinner as they go out. We model this by choosing a random width for the base of each hair and calculating a linear falloff as the hair goes upward. Second, they need to curl. We model this with a simple parabolic trajectory in a random direction. Third, the textures generated should be antialiased. This is important to prevent *jaggies* from making individual hairs appear to jump around as they go up. We do this by computing the percent of each hair that crosses a horizontal plane at the center of each texel that it passes through. Essentially this is calculating the intersection of the hair and the pieces of shell that it is going to be rendered on.

In order to simplify this calculation, we restrict the width of hairs to be at most 1 pixel, limiting the number of adjacent pixels that a hair can intersect to four. Each of these pixels is numbered from the upper left clockwise. These four cases, shown in Figure 2, allow us to approximate the area of the pixel covered by the fur. The formula that we use for this calculation is

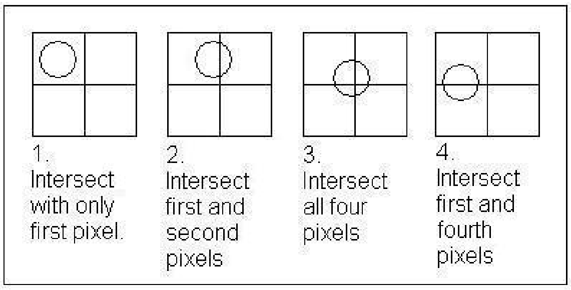
1.
Intersect with only first pixel.
2.
Intersect first and second pixels
3.
Intersect all four pixels
4.
Intersect first and fourth pixels

Figure 2: The four different hair-pixel intersections

Case 1:

$$\text{Pixel 1} = \frac{\text{area of circle}}{\text{area of pixel}}$$

Pixels 2-4 = 0

Case 2:

$$\text{Pixel 1} = \frac{\text{area of (circle}-\text{chord)}}{\text{area of pixel}}$$

$$\text{Pixel 2} = \frac{\text{area of chord}}{\text{area ofpixel}}$$

Pixel 3-4 $= 0$

Case 3:

$$\text{Pixel 1} = \frac{\text{area of (circle}-\text{(right chord + lower chord - wedge))}}{\text{area of pixel}}$$

$$\text{Pixel 2} = \frac{\text{area of (right chord - wedge)}}{\text{area of pixel}}$$

$$\text{Pixel 3} = \frac{\text{area of wedge}}{\text{area of pixel}}$$

$$\text{Pixel 4} = \frac{\text{area of (lower chord - wedge)}}{\text{area of pixel}}$$

Case 2:

Pixel 1 $= \frac{\text{area of (circle}-\text{chord)}}{\text{area of pixel}}$

Pixel 2-3 $= 0$

Pixel 4 $= \frac{\text{area of chord}}{\text{area ofpixel}}$

The area of the circle and pixel are trivial to calculate exactly, but the area of an arbitrary chord and wedge are somewhat more expensive. Correct values involve both an inverse tangent and a square root operation, which become expensive considering that this calculation is being performed many times for each hair in our volume. Our approach is to approximate both of these values, using an approximation from Harris and Stocker [1998] for the chord value and our own approximation for the wedge value. Given radius r and maximum length of radius inside the chord m, the formula for the area of the chord is

$$(1) \quad A = \frac{4}{3} \cdot d \cdot m + \frac{d^3}{4 \cdot m}$$

where $m = \sqrt{d(2r-d)}$. According to Harris, this will have maximum error of .8% from the true chord area. Our approximation of the wedge, given side lengths s_1 and s_2, is

$$(2) \quad A = \frac{1}{4} \cdot \pi \cdot (\frac{s_1+s_2}{2})^2$$

which is calculating a quarter the area of a circle. with radius the average of s_1 and s_2. We have not calculated error bounds on this approximation.

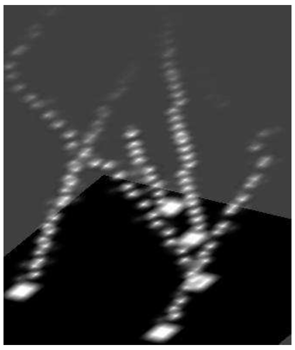
Figure 3: Closeup of several procedurally generated hairs

The last major feature that we address is the number and type of the hairs that we model. Both Kajiya [1989] and Gelder [1997] note the importance of including two layers of fur in their models. The undercoat is made of short hairs that form a dense layer around the object, while the overcoat is composed of longer, thicker hairs. Our approach renders both of these layers by changing the range of widths that each hair starts with. The undercoat will start with a very small width, and will peter out quickly as it goes upward. Longer hairs start with a thicker base and will reach a much greater distance from the base before thinning out. The inset in Figure 1 shows an example of our fur texture without any lighting. Color is set by depth, so darker areas are undercoat, while light areas are overcoat. Figure 1 shows a picture of our complete fur volume with a single low-frequency light. Figure 3 shows a close up of a few hairs.

2.2 Off-line Rendering

Our lighting method is based upon the lighting model for hair described developed by Kajiya et al[1989]. Their method was composed of 1) calculating the attenuation of scene lighting to each voxel on a fur volume, 2) modeling the microgeometry of fur through a lighting model, and 3) calculating the attenuation of the light from the surface to the eye. Our current method only does step 1, which effectivly means that we are calculating the sum illuminance to each point in the volume. Step 2 in their method converts this value into sum luminous exitance, a step which we push back from the precomputation to the reconstruction. This gives us the ability to use heterogeneous lighting models that vary over position and/or time. Our current implementation, however, doesn't take advantage of this freedom and simply displays the light reaching each voxel without any further computation. We perform Step 3 of their method at runtime through alpha blending on graphics hardware. Note that our use of the radiometric terms illuminance and luminous exitance are not strictly correct because our lighting calculations are not limited to the hemisphere above a surface, but are summed across the entire surrounding sphere.

The formula for attenuation of light along a ray $\mathbf{e}$ + $t\mathbf{d}$ through a volume is

$$(3) \quad T = e^{-r \int_{t_{end}}^{t_{start}} \rho(\mathbf{e}+t\mathbf{d})}$$

where T is the transparancy of the surface at a particular voxel and ρ is the 3D density function. However, since we are working with a 'texel' model as defined by Kajiya [1989] the microsurface version of the formula is what we need. It is given by

$$(4) \quad T' = e^{-r \Sigma_{t_{end}}^{t_{start}} \rho(\mathbf{e}+t\mathbf{d})}$$

This version is exactly the same as (3) except the integral is replaced with a sum to take into account the fact that line integrals through surfaces go to zero, which is not what we want. Instead, we sum the contribution of each surface along the ray. Our implementation of this lighting calculation takes the the vector $\mathbf{d}$ and the 'texel' volume data and marches a ray from the center of each voxel in the direction of $\mathbf{d}$ until it leaves the volume, adding the density of each voxel hit scaled by the length of ray that passed through it. Note that this is in effect placing the light at ∞ in the direction defined by the vector $\mathbf{d}$.

2.3 PRT Basis Functions

The Spherical Harmonics (SH) are an infinite set of spherical functions that form an orthonormal basis over the sphere. The SH function with parameters $l : l \in \mathbb{N}$ and $m : -l \geq m \leq l$ is defined by

$$(5) \quad Y_l^m(\theta,\psi) = \sqrt{\frac{2l+1}{4\pi} \cdot \frac{(l-m)!}{(l+m)!}} \cdot e^{im\psi} \cdot P_l^m(cos\theta)$$

where P_l^m are the associated Legendre polynomials. This formulation isn't quite what we want for this application, however, because it will contain imaginary numbers if $m \neq 0$. We can remedy this by constructing the following piecewise defined function to take care of those cases:

$$(6) \quad Y_l^m(\theta,\psi) = \begin{cases} \sqrt{2} \cdot K_l^m \cdot cos(m\psi) \cdot P_l^m \cdot (cos\theta), & if\ m > 0 \\ \sqrt{2} \cdot K_l^m \cdot sin(-m\psi) \cdot P_l^{-m} \cdot cos(\theta), & if\ m < 0 \\ K_l^0 \cdot P_l^0 \cdot cos(\theta), & if\ m = 0 \end{cases}$$

$$\text{where } K_l^m = \sqrt{\frac{2l+1}{4\pi} \cdot \frac{(l-m)!}{(l+m)!}}.$$

The SH functions are applicable to our problem because they allow us to approximate arbitrary spherical functions. A sum of an unbounded number of SH basis functions with the correct coefficients can approximate a function to any level of accuracy. In addition, the frequency of the activity in a SH function is preportional to the order (see Figure 4), so if we are interested only in a rough, low frequency approximation we only need to compute a fixed number of low-order SH functions to use as a basis.

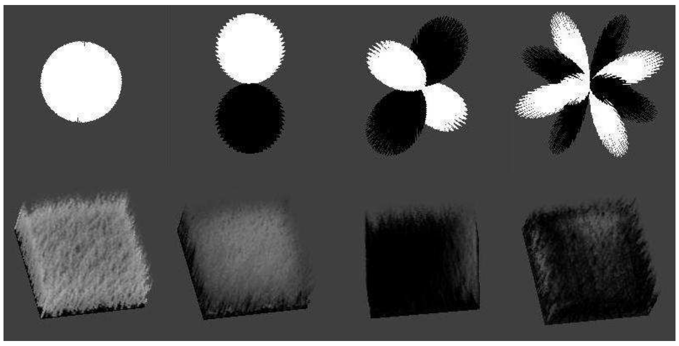

Figure 4: Top: SH functions with (l,m) left to right of (0,0), (1,0), (2,1), and (4, -3). Bottom: Fur volume lit with the SH function above it. Brightness of lighting is scaled to enable comparison.

PRT [Sloan et al. 2002] takes these ideas and applies them to the problem of real-time shadows. If we can calculate our lighting of a complex object based upon SH basis function lighting, then we can approximate any lighting environment by simply multiplying each of our pre-computed lighting models by the correct scalar coefficient and summing the results together. This requires that we be able to encode lighting into a SH basis, but it also requires that we are able to approximate our target lighting environment in terms of the SH basis functions that we have chosen, which the properties of SH let us do fairly easily.

Our implementation starts by choosing the 5 lowest order SH functions, which comes out to 25 distinct functions. We treat them as spherical lights by using the infinite sphere with brightness scaled according to the functions and compute lighting of our fur volume in those environments, using the fur lighting algorithm discussed above. To do this, we first compute a set of rays that are uniformly distributed over the sphere, which we use to sample all the different SH functions. We then compute the lighting for every point in the fur volume for each sample of each SH function, storing the results in textures. Since each of our SH functions must have its own unique space for storing the results of the lighting computation, we need 25 textures for each shell. This gives us 32 x 25 = 800 single-channel textures. In practice we pack 4 SH function lighting results into a four-channel texture, giving us 32 x 7 = 224 different textures.

2.4 Reconstruction

Reconstruction of lighting in an arbitrary surface requires several steps. They can be summarized as

1. Generate a spherical map of the target lighting environment
2. Project that map into our SH basis
3. Reconstruct the lighting for that projection using our pre-computed lighting maps

There are many different ways to capture the target lighting of a scene. It is possible to render a cubemap from the perspective of the object, and sample the cubemap to create a spherical representation, but this is outside the scope of our project. We use the simple method of using a spherical function that produces one low-frequency lobe of light, which we then rotate on a frame-by-frame basis. This representation is passed on to the next stage of reconstruction as the target lighting environment. Note that we could create any functional lighting environment to use without changing any other pieces of code. The function that we use is based on the function described by Green [2003] in his overview of the PRT method.

Once the target lighting is available in SH coordinates, we sample it with our pre-computed uniformly distributed list of rays (the same one which we used to approximate each SH basis function). Projecting the spherical environment $env(\theta,\psi)$ into the SH basis $y_l^m(\theta,\psi)$ is then a simple weighted average over the sample rays, given as

$$(7) \quad c_{m,l} = \frac{1}{rays} \cdot \sum_{rays}^{0} env(r_i) \cdot Y_l^m(r_i)$$

where $c_{m,l}$ is the coefficient for the corresponding SH basis function and r_i is the ith sample ray in spherical coordinates. Computing this for each of our 25 basis functions gives a 25 component vector. If you know that the lighting of the object will only change rotationally, then it is possible to precompute the lighting in the beginning and simply rotate it to whatever position it is in for subsequent frames. Our current implementation doesn't perform efficient SH rotation since our goal is to be able to put a furry object into an arbitrarily changing lighting environment.

The last step in reconstruction is to multiply each of the $c_{m,l}$ coefficients by the corresponding SH basis functions's lighting map for each pixel on each shell. We perform this computation in a fragment shader in graphics hardware by passing the seven four-channel textures containing the lighting information and the seven four-component vectors containing the coefficient information and performing a 25-component dot product. The three extra spots in the last vector can be used to store any other uniform information that needs to be sent to the shader.

3 Results

Figure 5 shows several unique lighting environments captured in real-time a few seconds apart. We ran all our tests on an Athlon 64 3000+ computer with 512 MB of ram and a GeForce 6800 SLI setup. For a test volume of 64x64x32 we were able to achieve 40FPS at a resolution of 1280x1024.

The results clearly show the largest drawback of our method to be how performance scales in an application. Since we are computing a 25-component dot product for each fragment, things slow down quickly when high resolutions and large areas of fur are used. This is a much worse problem with our method than it is with other PRT methods since most other methods are performing the dot product for each vertex, not fragment. There are several ways that could speed up our method, which we discuss in Future Work.

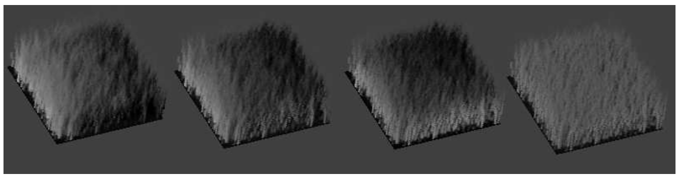

Figure 5: Four real-time views taken as a single light is rotated around the front of the volume and up.

It is worth mentioning that our implementation has a small difficiency in the way in which it calculates the lighting, making the light appear slightly brighter from one side of the fur when compared to the same light on the other side. As of the writing of this paper, this issue has not been fixed.

4 Future Work

Several elements of our implementation could be extended to reproduce a more complete version of either PRT or fur rendering. An improved lighting model would make the fur look much better. Such a model could be based upon the Kajiya [1989] frame bundle technique, or something else. An interesting approach would be to store this information and pass it to the shader at runtime, which would allow varying lighting models across the surface and/or a single set of precomputed lighting maps for different styles of fur. Such a technuque could vastly improve the realistic look of the fur, and allow much more flexibility.

Two things that were done by Leyngal et al [2001] that could be modified to work in our renderer were varying the color of fur over the surface (ours is uniform) and creating 'fin' textures to add substance when the shells are nearly parallel with the viewing direction. The dog model in their paper is a good example of the type of effects that color variation can display, and combined with the realistic lighting model that we present, could go a long way toward photorealistic fur in real-time. The texture fins are really a necessary part of a fur shell texture implementation, but our model adds the extra complexity of having to perform PRT on the fin textures as well as the shell textures. We plan to add fins to our model shortly.

Another major piece that we didn't implement would be placing the fur over a surface. Both Leyngal [**?**] and Kajiya [1989] perform this step with their fur volumes, and both achieved visual success with their respective methods. Leyngal had the advantage of being able to perform this mapping onto the 3D model automatically with the use of Lapped Textures [Praun et al. 2000], but both experimented with 'combing functions' that provided local changes to simulate irregular areas of fur. Adding this ability to our method would be an interesting project, and it is one which we plan to pursue. The ultimate test would be to render something like the Kajiya [1989] teddy bear in realtime.

There are also several directions that the project could be extended in terms of PRT. We noticed while working with this project that the higher level SH basis functions produced minimal values in the lighting textures. It would be worth investigating the effect of reducing the number of basis functions used on the realism of the fur. It could be the case that the 4th order SH functions are not necessary for a visually appealing fur rendering, which would significantly reduce the processing required per-pixel. This would enable much more scalability than in most other applications of PRT.

It would also be interesting to try a set of basis functions other than the SH ones that we use. Investigating the results with Zonal Harmonic or Wavelet basis functions would be an interesting endeavour. Specificaly, the ZH basis functions seem to be an interesting choice given that they are easily rotatable and can be more efficient in texture space when used correctly [Sloan et al. 2005]. This would speed up rendering a furry object through pasting a small precomputed fur volume over a 3D surface since the ZH coefficients could easily be rotated for each position on the surface.

References

GELDER, A. V., AND WILHELMS, J. 1997. An interactive fur modeling technique. In *Proceedings of Graphics Interface 1997*, Graphics Interface.

GOLDMAN, D. B. 1997. Fake fur rendering. In *Proceedings of SIGGRAPH 1997*, ACM Press / ACM SIGGRAPH, ACM.

HARRIS, J. W., AND STOCKER, H. 1998. *Handbook of Mathematics and Computational Science*. New York: Springer-Verlag.

KAJIYA, J. T., AND KAY, T. L. 1989. Rendering fur with three dimensional textures. In *ACM SIGGRAPH Computer Graphics Vol. 23, Issue 3*, ACM SIGGRAPH.

KAUTZ, J., SLOAN, P.-P., AND SNYDER, J. 2002. Fast, arbitrary brdf shading for low-frequency lighting using spherical harmonics. In *Thirteenth Eurographics Workshop of Rendering*, The Eurographics Association, The Eurographics Association.

KRISTENSEN, A. W., AKENINE-MOLER, T., AND JENSEN, H. W. 2005. Precomputed local radiance transfer for real-time lighting design. In *Proceedings of SIGGRAPH 2005*, ACM Press / ACM SIGGRAPH, ACM.

LENGYEL, J., PRAUN, E., FINKELSTEIN, A., , AND HOPPE, H. 2001. Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press / ACM SIGGRAPH, ACM.

LIU, X., SLOAN, P.-P., SHUM, H.-Y., AND SNYDER, J. 2004. All-frequency precomputed radiance transfer for glossy objects. In *Eurographics Symposium on Rendering 2004*, The Eurographics Association, The Eurographics Association.

LOKOVIC, T., AND VEACH, E. 2000. Deep shadow maps. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, ACM.

PERLIN, K., AND HOFFERT, E. M. 1989. Hypertexture. In *ACM SIGGRAPH Computer Graphics Vol. 23, Issue 3*, ACM SIGGRAPH.

PRAUN, E., FINKELSTEIN, A., , AND HOPPE, H. 2000. Lapped textures. In *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH, ACM.

R.GREEN. 2003. Spherical harmonic lighting: The gritty details. In *Proceedings of GDC, 2003*, Game Developers Conference.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environemnts. In *Proceedings of SIGGRAPH 2002*, ACM Press / ACM SIGGRAPH, ACM.

SLOAN, P.-P., HALL, J., AND SNYDER, J. H. J. 2003. Clustered principal components for precomputed radiance transfer. In *Proceedings of SIGGRAPH 2003*, ACM Press / ACM SIGGRAPH, ACM.

SLOAN, P.-P., LIU, X., SHUM, H.-Y., AND SNYDER, J. 2003. Bi-scale radiance transfer. In *Proceedings of SIGGRAPH 2003*, ACM Press / ACM SIGGRAPH, ACM.

SLOAN, P.-P., LUNA, B., AND SNYDER, J. 2005. Local, deformable precomputed radiance transfer. In *Proceedings of SIGGRAPH 2005*, ACM Press / ACM SIGGRAPH, ACM.

WANG, R., TRAN, J., AND LUEBKE, D. 2005. All-frequency interactive relighting of translucecnt objects with single and multiple scattering. In *Proceedings of SIGGRAPH 2005*, ACM Press / ACM SIGGRAPH, ACM.