Vol. [VOL], No. [ISS]: 1–15

Normal Mapping with Low-Frequency Precomputed Visibility

Michal Iwanicki
CD Projekt RED

Peter-Pike Sloan
Disney Interactive Studios

Abstract. Normal mapping is a common technique used in video games, decoupling surface details stored at high spatial frequencies which are often tiled or repeated, from lighting information that is both unique and stored at a lower sampling rates. This paper presents two techniques that couple normal maps on static geometry with soft shadows from smooth distant lighting in a more efficient manner compared to previous work. In the first technique the visibility function is represented using low-order spherical harmonics, and the product of the Lambertian clamped cosine function and the lighting environment is tabulated in textures. The second technique uses principal component analysis to compress the visibility function, decreasing the data size and increasing the performance. Finally we also examine the efficiency of four common parameterizations for spherical harmonics.

1. Introduction

Generating realistic images of complex scenes at interactive rates is a challenging problem. Games have traditionally used a combination of static light maps and dynamic point or directional light sources. Surfaces are generally textured with both reflectance properties and normal maps [Blinn 78, Peercy et al. 97] that approximate complex surface details. These textures tend to

1086-7651/06 $0.50 per page

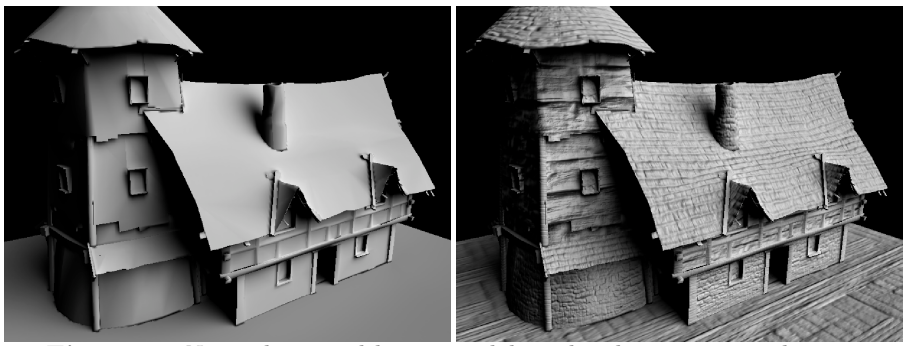


Figure 1: *Normal mapped house model rendered using our technique*

be sampled at high spatial sampling rates and are often layered and tiled to create even higher effective sampling rates. In contrast, precomputed lighting techniques require unique representations of a signal and are stored at much lower sampling rates. It is impractical to sample these unique details at the effective composite sampling rate.

For static lighting these interdependencies have been decoupled both in film [Tabellion and Lamorlette 04] and games [McTaggart 04, Chen 08]. For parameterized lighting, a heavy weight technique that can model fine scale lighting effects [Sloan et al. 03b] has been proposed along with a lighter weight technique [Sloan 06] inspired by techniques used in games [McTaggart 04].

This work focuses on just shadows, not inter-reflections. A simpler precomputed signal is used, visibility instead of a transfer matrix, that reduces storage costs and enables higher frequency lighting compared to previous work [Sloan 06].

2. Background

Decoupling of normal variation from lighting has been done in off-line rendering [Tabellion and Lamorlette 04], and games [McTaggart 04, Good and Taylor 05, Chen 08] for static lighting. Chen [Chen 08] modeled both diffuse and glossy low frequency lighting, while most of the other work, like ours, focused on diffuse lighting.

Precomputed Radiance Transfer (PRT) [Sloan et al. 02] enables interactive rendering of static objects/scenes with complex global illumination effects under dynamic (often distant) illumination. The work on PRT parameterized lighting with low frequency spherical harmonics, as do we, but instead of storing a transfer vector that encodes normal-dependence we simply store the visibility function like Ng et al. [Ng et al. 04]. The reflectance equation can

then be efficiently evaluated for an arbitrary normal using spherical harmonics tripling coefficients analogous to the wavelet triple product approach [Ng et al. 04]. PRT has been extended to handle effects at finer spatial scales in two previous papers: the first approach can model complex local effects but is too heavyweight for games [Sloan et al. 03b], while the second approach was only capable of handling normal variation [Sloan 06]. Unlike our proposed technique, both of these papers modeled direct and indirect light, but at a large storage cost for transfer matrices stored over the surface of the mesh. These matrices can be compressed at vertices [Sloan 06], but still use significantly more memory than our approach.

Another popular approach to model complex distant illumination is to use preconvolved environment maps together with precomputed ambient occlusion maps. This method is very cheap, since the preconvolved enviroment map can be represented with a low order SH vector or a low resolution cubemap, and ambient occlusion requires just a single floating point component per shade point. However this method is limited in the sense that directional occlusion is not accounted for and variations in distant illumination do not affect e.g. the shadowing on the surface. A more heavyweight technique [Green et al. 07] precomputes visibility on static scenes at the vertices and uses that to add approximate shadowing to preconvolved environment mapping techniques. Our approach focuses on diffuse reflection and evaluates the product of visibility and lighting more accurately.

The use of directional lightmaps (as in [McTaggart 04, Good and Taylor 05]) addresses the aforementioned problems, however only under very limiting dynamic lighting constraints where a separate set of lightmaps are required for each expected incident lighting scenario.

Instead of using precomputed occlusion information some methods generate it at runtime, combining precomputed occlusion generated for individual parts of the scene [Hill 10]. Since this combination is performed in screen space, such methods cannot afford a higher order SH representation for occlusion, due to memory constraints (additionally, certain simplifications and approximations are required for multiple occlusion). The combinations of these limitations makes these techniques only practical for ambient occlusion rather than directional occlusion. Our method offers correct directional occlusion and normal mapping while still allowing fully dynamic lighting, but at a higher cost. Other work on rendering fully dynamic scenes [Ren et al. 06, Sloan et al. 07] use a similar rendering technique, but compute visibility on-the-fly and do not address compression. We simply store visibility, enabling our techniques to scale to more complex scenes.

Indirect lighting is not modeled in our technique, but direct-to-indirect radiance transfer [Lehtinen et al. 08] could be used to model such effects.

2.1. Spherical Harmonics

Spherical harmonics are the spherical analog of the 1D fourier basis and have been used extensively in computer graphics. The general form is:

$$Y_l^m(\theta,\phi) = K_l^m e^{im\phi} P_l^{|m|}(\cos(\theta)), l \in \mathbb{N}, -l \leq m \leq l \tag{1}$$

where P_l^m are the associated Legendre polynomials and K_l^m are the normalization constants

$$K_l^m = \sqrt{\frac{(2l+1)(l-|m|)!}{4\pi(l+|m|)!}} \tag{2}$$

The above definition forms a complex basis; a real-valued basis is given by the simple transformation

$$y_l^m = \begin{cases} \sqrt{2}K_l^m \cos(m\phi)P_l^m(\cos\theta) & m > 0 \\ \sqrt{2}K_l^m \sin(|m|\phi)P_l^{|m|}(\cos\theta) & m < 0 \\ K_l^m P_l^m(\cos\theta) & m = 0 \end{cases} . \tag{3}$$

Low values of l (called the band index) represent low-frequency basis functions over the sphere. The basis functions for band l reduce to polynomials of order l in the Cartesian coordinates on the sphere, x, y, and z.

Spherical harmonics can only efficiently represent smooth lighting environments. Smooth lighting environments induce low spatial sampling rates, making them more practical for applications like computer games. We use them to represent both the lighting environments and the visibility functions and we direct readers to the [Sloan 08] for a more detailed overview of spherical harmonic properties.

3. Decoupling surface variation from visibility

Let us consider a diffuse-only surface with no inter-reflections. Outgoing radiance for such a surface can be expressed using the reflectance equation:

$$I = \int_\Omega L(\vec{\omega})V(\vec{\omega})H(\vec{\omega})d\vec{\omega} \tag{4}$$

Where I is the outgoing radiance, L represents the distant lighting environment, V is the visibility function and H is the cosine lobe oriented about the surface normal. If all of the terms are expressed in a common basis, the reflectance equation can be solved by computing the tripling coefficients with respect to the basis [Ng et al. 04]. With spherical harmonics, we can further

simplify this computation by first computing the SH product of two of the three terms, followed by a dot product with the third [Ren et al. 06, Sloan 08].

In a typical PRT formulation of direct light, the lighting is projected onto some basis and plugged into the integral. Factoring the unknown lighting projection coefficients out results in a transfer vector that depends on normal variation. We instead project both light and visibility into SH, and effectively tabulate the product of light and the cosine term over the space of normals. This allows us to decouple the smooth visibility function from high frequency normal variations. Furthermore, the normals are completely decoupled from all tabulated data allowing us to apply the usual tricks of tiling/layering normal maps.

For a given lighting environment, we need to evaluate $L \times H$ for any possible normal, which we tabulate in textures - absorbing the cosine term into the lighting allows us to pre-compute the SH products that are normally expensive to evaluate in shaders. This will also aid in compressing the signal, which we will discuss later.

The reflectance equation can then be evaluated by simply taking a dot product of the SH coefficients for visibility and the $L \times H$ coefficients looked up in the precomputed texture. Figure 2 outlines the dataflow of our algorithm.

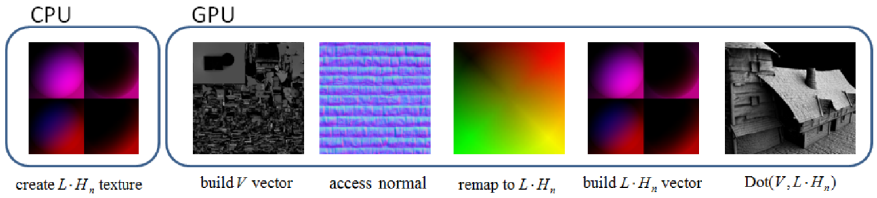


Figure 2: *Stages of algorithm. The first stage is the only one on the CPU.*

3.1. Storing V information

As we apply our technique on rigid models, we can precompute visibility functions V across the model. We store them similarly to lightmaps: a surface parametrization of the model is created together with the texture storing an SH-encoded visibility function at each texel. V coefficents can be stored as floating-point values, but this leads to high memory usage. Alternatively, other more memory-efficient storage schemes [Ko et al. 08] can be applied.

Figure 3: *Sample texture storing V coefficients.*

3.2. Storing $L \times H$ Information

As stated before, $L \times H$ coefficients should be stored for all possible orientations of a surface normal. The natural solution for such look-up table is a cube map texture. It allows us to directly use the normal vector to obtain the coefficients. As the function stored in the textures varies quite slowly, cube map do not need to be large. See the appendix for a comparison of efficiency of representing smooth signals with several parameterizations.

Using a cube map has two drawbacks. Given O SH visibility, $3O^2$ values must be tabulated for every normal, making cube maps impractical on DX9-class hardware since they have at most 16 textures available for a single draw call and often only 8. The visibility functions require $\frac{O^2}{4}$ textures[1] and textures are commonly used for normal maps and albedo. The second drawback is that on DX9-class hardware cube maps do not interpolate across cube faces, causing visible discontinuities that are particularly objectionable at low resolutions. These are not issues on DX10-class hardware, as it natively supports texture arrays [Blythe 06] and also filters across cube faces.

On DX9 hardware we store $L \times H$ in dual paraboloid parametrization [Heidrich and Seidel 98], where we use a mapping from the disk to the square [Shirley and Chiu 97] to guarantee $C0$ continuity across the equator. We use one texture per color channel, and pack 4 values per texel, laying the spheres out in the horizontal axis of the texture. A resolution of 16-to-32 tends to be adequate based on our empirical analysis.

3.3. Dynamic Lighting Environments

One of the most important properties of PRT is its ability to render the same model under totally different lighting conditions. This is still possible with

[1]One large texture could be used if an atlas is built, but that would generate spurious dependent texture operations.

our method. All that is required is a separate set of $L \times H$ look-up textures for each lighting environment we want to use. As the look-up textures contain SH coefficients, they can be blended to produce smooth transitions. For cases where lighting does not change drastically between consecutive frames (for example, in lighting of outdoor scenes), blending can be performed on the CPU and uploaded to the GPU at a lower temporal frequency, saving GPU pixel shading power.

The symmetries of spherical harmonics can be exploited to efficiently build our lookup textures. Given the product matrix of the lighting environment (number of rows is 9 to represent the clamped cosine function, number of columns is determined by the visibility order) one simply needs to scale the rows of the matrix by the SH coefficients of a clamped cosine function oriented with each texel's normal. This can be done efficiently using CPU vector instruction sets. The tabulated normals can be precomputed for a single hemisphere and negating a SH function in Z simply changes the sign of the basis functions where m is odd. This can be done in the unrolled vector code that combines the columns. 90 degree Z rotations are also efficient, if only a quarter of the hemisphere was to be used i.e., on the side faces of a cube map.

4. Compressed Rendering Technique

Unlike work on completely dynamic scenes [Ren et al. 06, Sloan et al. 07], we would like to compress the visibility functions to reduce storage. We use the principal component analysis (PCA) compression scheme. PCA treats a signal as a linear combination of basis vectors and a signal mean. Instead of storing a spatially varying signal, we need only store a set of basis vectors and spatially varying basis function weights. As such, we trade a higher-order SH representation with a lower-dimensional PCA basis representation, and substituting this PCA visibility representation in (4) yields:

$$I = \int_{\Omega} L(\vec{\omega}) \left(\sum_{n=1}^{M} w_n V_n(\vec{\omega}) + V_e(\vec{\omega}) \right) H(\vec{\omega}) d\vec{\omega} \tag{5}$$

where the sum is over the M PCA basis vectors V_n, V_e is the mean visibility vector and w_n are the spatially varying PCA projection coefficients. The sum and addition can be factored out due the linearity of integration, resulting in the following equation:

$$I = \sum_{n=1}^{M} w_n \left(\int_{\Omega} L(\vec{\omega}) V_n(\vec{\omega}) H(\vec{\omega}) d\vec{\omega} \right) + \int_{\Omega} L(\vec{\omega}) V_e(\vec{\omega}) H(\vec{\omega}) d\vec{\omega} \ . \tag{6}$$

In the compressed case, our textures store these spatially varying PCA

weights w_n. Additionally, for each visibility basis vector, we generate a separate texture that is indexed by the surface normal. These textures contain the triple product integral of lighting L, the clamped cosine lobe H (oriented about the normal), and the corresponding visibility basis function V. We note that these integrals result in color outputs, not SH vectors. They can be interpreted as environment maps occluded by the visibility function defined by a given PCA basis vector, convolved with a clamped cosine kernel.

An alternative approach requires storing just the PCA weights w_n and off-loading more computation to the shaders at run-time: we can compute a product of V_n and L for each basis vector (resulting in an SH vector for each color channel of the lighting environment), pass them to the shader, and perform the final convolution against $H(\vec{\omega})$ directly in the shader[2]. Visibility (in the form of weights for the basis vectors) can also be stored at the vertices of a mesh, instead of texels, marking a fairly simple extension of previous work [Sloan et al. 03a].

Following the evolution of previous work, we experimented with clustered PCA representations but our initial experiments illustrated objectionable artifacts at cluster boundaries. Addressing these artifacts is left for future work.

5. Results and discussion

Below are performance measurements for the house scene. Times are in milliseconds, spent rendering the image using given technique, per frame

Unc6	Unc4	Unc3	PCA4	PCA8
8.19 ms	3.33 ms	2.75 ms	1.40 ms	2.08 ms

PCA16	PCA24	SHLmap	Preconv	SH2
3.61 ms	5.56 ms	0.846 ms	0.648 ms	0.607 ms

The visibility textures are 512×512 and we render into a 1280×800 framebuffer. The numbers are GPU time, measured on an NVIDIA GeForce GTX 295. The main cost is texture lookup, since the technique performs a dependent lookup based on the normal, stressing the texture cache. In case of the PCA compressed version, this cost could be reduced by using the variation of the method that store just the weights, as described earlier.

Compressed results are not sensitive to the lighting order: increasing the lighting order just requires more PCA vectors to get an accurate result. Based on our experiments, 12 PCA vectors yields pleasing results for 4^{th} order SH lighting, and 16 to 24 vectors seem necessary for 6^{th} order SH lighting. If

[2]The Lambertian clamped cosine lobe is simple enough to be generated analytically in the shader, where this convolution boils down to a dot product in SH

visibility is used for something else [Green et al. 07], the uncompressed solution would be more efficient, otherwise the compressed representation is preferable.

The table also contains the performance data for three other methods: SH lightmaps (storing 3^{rd} order SH lighting; see the SHLmap column), using preconvolved environment map scaled by scalar AO (Preconv column), and performing multiplication of the lighting environment and the visibility function followed by a convolution against the clamped cosine kernel in the shader (this requires no precomputation of $L \times H_n$ textures, however is only practical for SH orders up to 2; see the SH2 column).

The images below show the result of lighting the house model using techniques from the table.

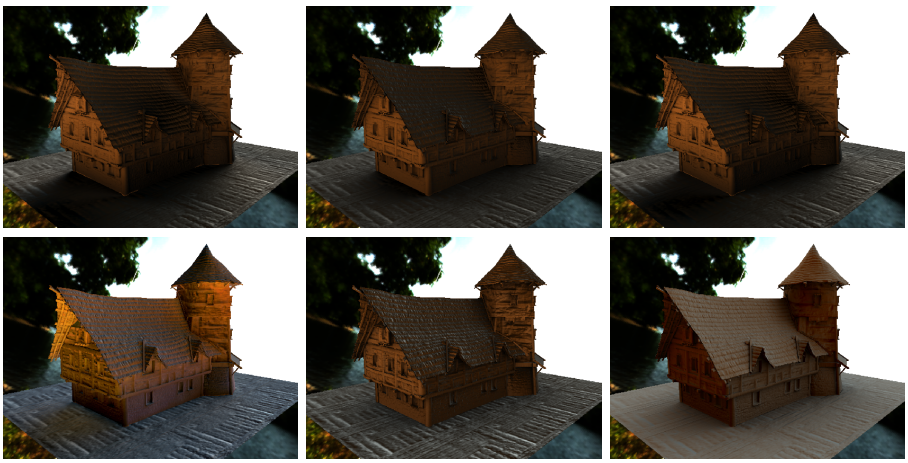


Figure 4: *Top: Unc6, PCA4, PCA16. Bottom: SHLmap, Preconv, SH2.*

The SH Lightmap offers the best quality because it models indirect lighting, unlike the other methods, leading to more subtle and realistic shadowing. However this method does not allow for any changes in the lighting. Using just the preconvolved environment map with scalar AO fails to model any shadowing of the bright light source on the right. The all-shader version using 2^{nd} order SH captures some shadowing but not with very well pronounced regions. Our method generates compelling shadows, and the compressed version (with 16 PCA basis vectors) yields results comparable to the uncompressed version at a much lower performance and memory cost. Our primary limitation is a lack of indirect illumination, which is especially noticeable when the lighting consists of a single strong directional source. This can be avoided with environmental lighting environments covering the whole sphere or in introducing additional terms to simulate the indirect component. We leave this last point as an area of future work.

Acknowledgements

Thanks to Paul Debevec and Brian James for the lightprobes used to light the models, Stephen Hill and Derek Nowrouzezahrai for comments. The house model is courtesy of Michal Buczkowski and Tomasz Polit, CD Projekt RED.

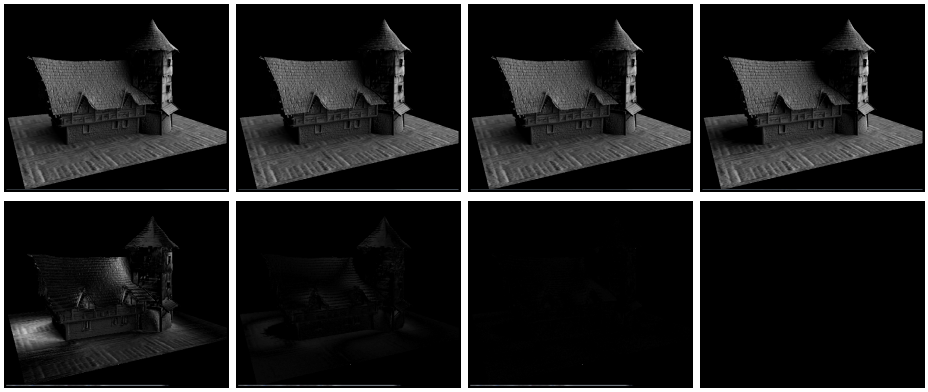


Figure 5: *Comparison between different compression settings. Top: 4 PCA, 12 PCA, 24 PCA, uncompressed; Bottom: difference between 4/12/24/uncompressed and uncompressed version (scaled by 400 %.)*

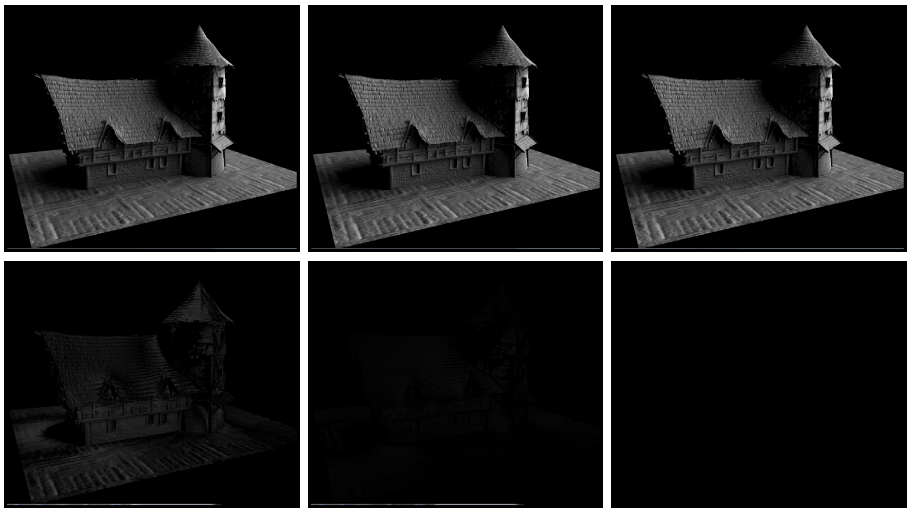
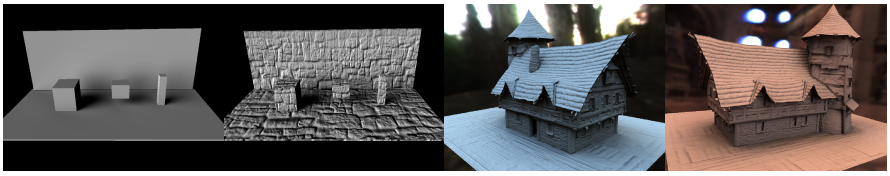


Figure 6: *Comparison of different SH orders used to encode visibility. Top: 3^{rd} order, 4^{th} order, 6^{th} order. Bottom: difference between 3^{rd} order and 6^{th} order, 4^{th} order and 6^{th} order (difference is scaled by 400 %).*



(a) *No normal Map* **(b)** *Normal Map* **(c)** *Eucalyptus Grove* **(d)** *St. Peter's basilica*

Figure 7: *Other scene and lighting environments.*

References

[Blinn 78] James F. Blinn. "Simulation of Wrinkled Surfaces." In *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12, 12, 1978.

[Blythe 06] David Blythe. "The Direct3D 10 system." *ACM Transactions on Graphics* 25:3.

[Chen 08] Hao Chen. "Lighting and Materials of Halo3." In *Game Developers Conference*, 2008.

[Good and Taylor 05] Otavio Good and Zachary Taylor. "Optimized Photon Tracing Using Spherical Harmonic Light Maps." In *Siggraph 2005 Sketches*, 2005.

[Green et al. 07] Paul Green, Jan Kautz, and Frédo Durand. "Efficient Reflectance and Visibility Approximations for Environment Map Rendering." *Computer Graphics Forum (Proc. EUROGRAPHICS)* 26:3.

[Heidrich and Seidel 98] Wolfgang Heidrich and Hans-Peter Seidel. "View-Independent Environment Maps." In *Proceedings of Graphics Hardware 1998*, 1998.

[Hill 10] Stephen Hill. "The Rendering Tools And Techniques Of Splinter Cell: Conviction." In *Game Developers Conference*, 2010.

[Ko et al. 08] Jerome Ko, Manchor Ko, and Matthias Zwicker. *ShaderX*^{6}, Chapter Practical Methods for a PRT-based Shader Using Spherical Harmonics, pp. 355–380, 2008.

[Lehtinen et al. 08] Jaakko Lehtinen, Matthias Zwicker, Emmanuel Turquin, Janne Kontkanen, Frédo Durand, François X. Sillion, and Timo Aila. "A meshless hierarchical representation for light transport." *ACM Transactions on Graphics* 27:3.

[McTaggart 04] Gary McTaggart. "Half-Life 2 Source Shading." In *Game Developers Conference*, 2004.

[Ng et al. 04] Ren Ng, Ravi Ramamoorthi, and Pat Hanrahan. "Triple product wavelet integrals for all-frequency relighting." *ACM Transactions on Graphics* 23:3.

[Peercy et al. 97] Mark Peercy, John Airey, and Brian Cabral. "Efficient Bump Mapping Hardware." In *SIGGRAPH 97 Conference Proceedings*, *Annual Conference Series*, edited by Turner Whitted. ACM SIGGRAPH, 1997.

[Ren et al. 06] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. "Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation." *ACM Transactions on Graphics* 25:3.

[Shirley and Chiu 97] Peter Shirley and Kenneth Chiu. "A Low Distortion Map between Disk and Square." *Journal of Graphics Tools* 2:3.

[Sloan et al. 02] Peter-Pike Sloan, Jan Kautz, and John Snyder. "Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments." *ACM Transactions on Graphics* 21:3.

[Sloan et al. 03a] Peter-Pike Sloan, Jesse Hall, John Hart, and John Snyder. "Clustered principal components for precomputed radiance transfer." *ACM Transactions on Graphics* 22:3.

[Sloan et al. 03b] Peter-Pike Sloan, Xinguo Liu, Heung-Yeung Shum, and John Snyder. "Bi-scale radiance transfer." *ACM Transactions on Graphics* 22:3.

[Sloan et al. 07] Peter-Pike Sloan, Naga K. Govindaraju, Derek Nowrouzezahrai, and John Snyder. "Image-Based Proxy Accumulation for Real-Time Soft Global Illumination." In *Pacific Graphics*, 2007.

[Sloan 06] Peter-Pike Sloan. "Normal Mapping for Precomputed Radiance Transfer." In *ACM Symposium on Interactive 3D Graphics and Games*, pp. 23–26. ACM Digital Library, 2006.

[Sloan 08] Peter-Pike Sloan. "Stupid Spherical Harmonics (SH) Tricks." In *Game Developers Conference*, 2008.

[Tabellion and Lamorlette 04] Eric Tabellion and Arnauld Lamorlette. "An approximate global illumination system for computer generated films." *ACM Transactions on Graphics* 23:3.

A. Choice of Spherical Parameterization

We store many low resolution environment maps and would like to leverage texture hardware as best as possible. The signals we are storing are represented using spherical harmonics, so the ability of a given parameterization to reconstruct the spherical harmonics accurately is of utmost importance. To this end, we compare four parameterizations of the sphere:

- Dual Orthographic Projection
- Dual Parabaloid Projection
- Cube maps that do not interpolate across a face (DX9)
- Cube maps that do interpolate across a face (DX10 and later)

These are all parameterizations that can be evaluated efficiently in hardware, and have at least $C0$ continuity of the signal[3] The parameterizations based on dual hemispheres use a common mapping from the disk to the sphere [Shirley and Chiu 97] and shrink the mapping so that the equator maps exactly to the outer ring of texel centers on the square. This generates redundant texels between hemispheres, but guarantees continuity.

To determine the efficiency of a given parameterization, we evaluate the SH basis functions at each texel center and reconstruct one million points over the sphere, computing the mean squared error. The figure below is a log-log plot of MSE as a function of the total number of samples. The DX10 parameterization is best if available.

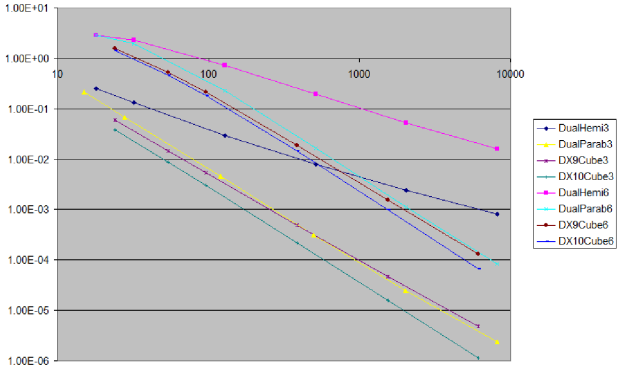


Figure 8: *MSE for 3^{rd} and 6^{th} order SH functions.*

[3]The DX9 cube maps do not have this property, do not work well for low resolution textures, and are only used for comparison purposes.

Web Information:

Michal Iwanicki, CD Projekt RED
(michal_iwanicki@naughtydog.com)

Peter-Pike Sloan, Disney Interactive Studios
(ppsloan@nvidia.com)

Received [DATE]; accepted [DATE].