6.1

Irradiance Volumes for Real-time Rendering

Chris Oat, **ATI Research, Inc.**

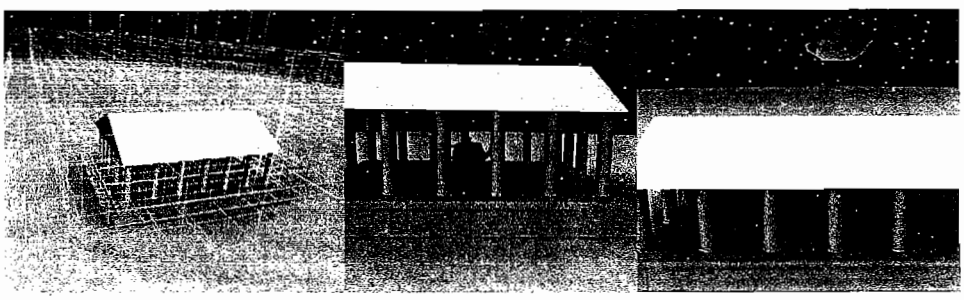

FIGURE 6.1.1 (Left) An adaptively subdivided octree is used to place irradiance sample points in the environment. The octree is finely subdivided in areas where irradiance changes quickly and is coarsely subdivided in areas where fewer samples are necessary because irradiance changes more slowly. (Center and Right) An irradiance volume is used to render a dynamic object (teapot) in a static scene. As the teapot moves through the scene, irradiance samples are chosen from the pre-computed volume and used to compute illumination.

Introduction

Real-time rendering relies on a host of tricks, optimizations and approximations to bring interactive graphics to consumer PCs. As each new generation of consumer graphics hardware exceeds the previous generation in terms of programmable flexibility and computational power, many traditionally offline rendering algorithms migrate out of back-room render farms and into the world of real-time. One of the many remaining discontinuities between real-time and non-real-time rendering is the use of global illumination for physically based, realistic lighting. While techniques such as light mapping can faithfully store global illumination on the surface of static scene geometry, light maps do not address the problem of lighting dynamic objects that move through the scene, and

thus we are left with beautifully rendered, globally illuminated scenes that contain unrealistic, locally lit dynamic objects. This problem is addressed by the irradiance volume [Greger98], which captures the effects of global illumination within a bounded 3D space by storing captured lighting samples for discrete locations within its bounds.

Irradiance Volumes

Irradiance volumes have been used by the film industry as an acceleration technique for high-quality, photo-realistic offline rendering. These volumes store irradiance distribution functions for points in space by utilizing some spatial partitioning structure that serves as a cache. Sampling the volume allows the global illumination at a point to be quickly calculated. Until recently, irradiance volumes were impractical for interactive rendering because of their high storage and bandwidth requirements, but it has been shown that spherical harmonics may be used for efficiently storing and evaluating irradiance samples, thus making precomputed irradiance volumes practical for real-time rendering. The remainder of this article deals with generating and using irradiance volumes for a static scene for which it is assumed that light maps have already been generated. An adaptive spatial subdivision scheme is presented, along with several subdivision heuristics, as a way of efficiently sampling an environment that contains varying degrees of geometric density. This subdivision scheme creates more sample points in areas where irradiance changes quickly in the spatial domain and creates fewer sample points in areas of slowly changing irradiance. Techniques for higher-ordered interpolation of irradiance samples within the volume using spherical harmonic irradiance gradients will also be discussed.

Before diving into the details of irradiance-volume generation, compression, and sampling, it is necessary to have a basic familiarity with the physical quantities we seek to represent: *radiance* and *irradiance*.

Radiance and Irradiance

Radiance is the emitted energy per unit time in a given direction from a unit area of an emitting surface [Cohen93]. Don't be confused by the term *emitting surface*. This doesn't mean that the energy originates at the surface in question; it could have arrived there as the result of multiple bounces. The important thing to understand is that there's some light energy coming from some surface in some direction. We could capture radiance at a point for *all* directions by placing a camera at the point and rendering the surrounding scene into a cube map. We would then scale every texel in that cube map by its projected solid angle (i.e., the projected solid angle of the rectangular texel onto a unit sphere located at the center of the cube map). This cube map would represent the radiance for all directions for the point at which it was captured. This is known as the radiance distribution function [Greger98] [Arvo95].

Once the radiance distribution function (L) is known, it may be used to compute irradiance. For a point p on a diffuse surface where the surface normal is N, irradiance is defined as the integral over a hemisphere of incident directions, centered at p and

oriented around N, of the radiance distribution function multiplied by the cosine of the angle of incidence (Lambertian term).

$$I(p,N_p)=\int_{\Omega^+} L(p,\vec{\omega}_i)(N_p\cdot\vec{\omega}_i)d\omega_i$$

We could compute irradiance at a point for *all* normal directions by performing a convolution of the radiance distribution function with a cosine kernel. The result of this convolution (see Figure 6.1.2) would be an irradiance distribution function that would look sort of like a radiance distribution function, except much blurrier.

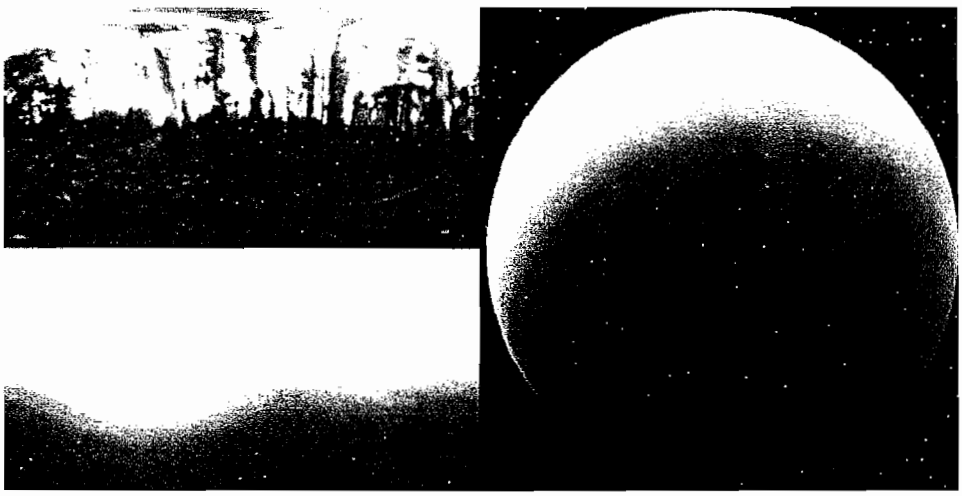

FIGURE 6.1.2 The Eucalyptus Grove Light Probe. (Top left) Captured radiance. (Bottom left) Irradiance map generated from captured radiance data. (Right) A sphere rendered using the resulting irradiance map. (Courtesy of Paul Debevec; *www.debevec.org/Probes/*.)

Just as the radiance distribution function can be stored in a cube map, so can the irradiance distribution function. A cube map could be generated at an object's centroid and reused for all points on the surface of that object. This cube map could be used for real-time rendering. It would be indexed by the surface normal of the point you're rendering, and it would return the sum of all diffuse illumination at that point [Brennan02] [Debevec98].

We could take this idea one step further and store irradiance cube maps for many points in a static scene and we will have created an irradiance volume. There are only two small problems. First, storing many cube maps for an entire scene would quickly fill up the video memory of your graphics card, and rendering would slow to a halt. The second problem is that we've made the assumption that irradiance changes slowly with

respect to p's position in space. This allows us to generate a single cube map at our object's centroid but use this same cubemap for all points on the surface of the object. In general this is not a bad assumption to make since in many cases irradiance really does change very slowly with respect to position, but it certainly is not true in *every* case.

Consider a spotlight that's aimed at an actor on a stage. The actor's "centroid" is fully contained by the spotlight's cone, but if the actor reaches out his or her hand, the actor's fingertips might end up outside of the spotlight's cone. In this case the irradiance distribution function sampled at the actor's centroid (inside the light cone) won't work well for computing irradiance at the tip of the actor's outstretched hand (outside the light cone). In this case irradiance changes quickly over a short distance, and a single sample taken at the actor's centroid is insufficient for computing lighting at the actor's fingertip. Both of these issues are addressed in the next section; spherical harmonics are used for compressing irradiance maps so that we can precompute and store many of them for use at render time, and spherical harmonic gradients are used to encode the rate at which irradiance changes around a given point.

Spherical Harmonics

Precomputation works well for eliminating the cost of dynamically generating diffuse environment maps, but we're still left with the cost of storing maps for many different points in our scene as well as the bandwidth overhead of indexing these maps at render time. Compressing irradiance maps by representing them as a vector of spherical harmonic coefficients allows us to reduce both the storage and bandwidth costs. Spherical harmonics are an infinite series of spherical functions that may be used as basis functions to store a frequency space approximation of an environment map [Ramamoorthi01]. Microsoft's DirectX 9.0 SDK includes functions for projecting a cube map into a representative set of spherical harmonic coefficients as well as functions for scaling and rotating spherical harmonics. For a more in-depth look at the theory, as well as code snippets for writing your own spherical harmonic helper functions, see Robin Green's *Spherical Harmonic Lighting: The Gritty Details* [Green03].

Using Microsoft's DirectX 9.0 SDK, a cube map may be projected into spherical harmonics using the `D3DXSHProjectCubeMap()` function. Once the captured cube map (representing incident radiance) is projected into spherical harmonics, the methods described in [Ramamoorthi01] may be used to approximate an irradiance distribution function.

Reconstructing from Spherical Harmonics

Once the irradiance distribution function has been found, it can be efficiently evaluated in a pixel or vertex shader, in the direction of the surface normal, for all surface locations on our model [Sloan04]. It's no longer necessary to store and sample irradiance from a cube map. We can instead store just nine spherical harmonic coefficients for each color channel of our irradiance distribution function. In fact, because spher-

ical harmonic evaluation doesn't require texture lookups, we can now compute irradiance per-vertex if we want and avoid per-pixel irradiance computation all together. Shader code for evaluating irradiance samples stored as spherical harmonic coefficients is provided in [Sloan04] and is also included in the sample on the companion CD-ROM.

ON THE CD

Irradiance Gradients

Irradiance samples only store irradiance for a single point in space. If an irradiance sample is used to shade the surface of an entire object, the potential error increases as we move farther away from the original sample point. Irradiance gradients allow us to store the rate at which irradiance changes with respect to translation about the irradiance sample (Figure 6.1.3) [Ward92].

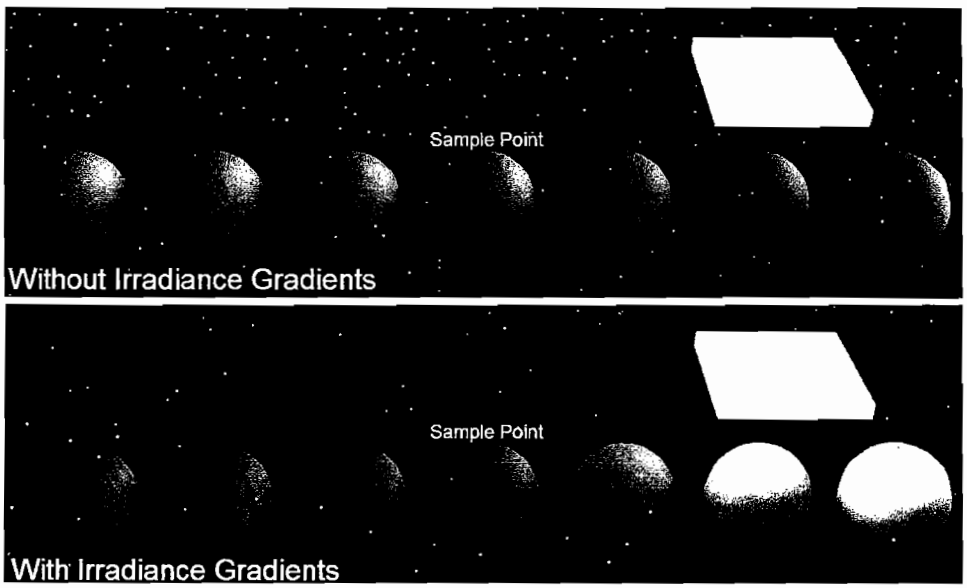

FIGURE 6.1.3 A single irradiance sample is taken in the center of the scene. The topmost row of spheres demonstrates how the irradiance error increases as the single irradiance sample is used to render a sphere that moves to either side of the sample location. The bottommost row of spheres uses the same irradiance sample as the top row but also stores a translational irradiance gradient and more accurately reflects the changes to irradiance as it moves to either side of the original sampling location.

Translational gradients for spherical harmonic irradiance samples may be computed in a number of ways [Annen04]. One simple way to find the gradient is to use central differencing to estimate the partial derivatives of the irradiance function as shown in Figure 6.1.4.

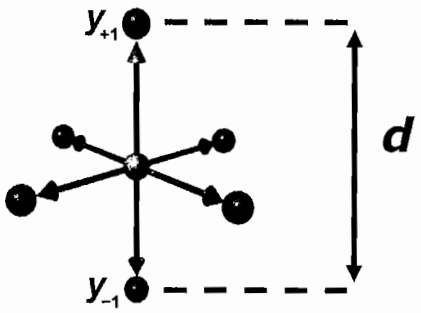

FIGURE 6.1.4 Central differencing may be used to compute a 3D gradient vector for each spherical harmonic coefficient.

Because our irradiance functions are stored as spherical harmonic coefficients, we need to project six additional irradiance functions into spherical harmonic coefficients (an additional sample for each of the $\pm x$, $\pm y$, and $\pm z$ world-space axes) and perform central differencing on each of the coefficients.

$$\nabla_y = \frac{y_{+1} - y_{-1}}{d}.$$

Once the gradient has been found, the six extra samples may be discarded. At render time, a new irradiance function may be extrapolated by computing a world-space vector from the location at which the sample was generated to the point being rendered. The dot product of this vector and the gradient vector is used to extrapolate a new irradiance function.

$$I_i' = I_i + (\nabla I_i \cdot d),$$

where I_i' is the ith spherical harmonic coefficient of the extrapolated irradiance function, I_i is the ith spherical harmonic coefficient of the stored irradiance sample, ∇I_i is the irradiance gradient for the ith irradiance coefficient, and d is a nonunit vector from the original sample location to the point being rendered.

Irradiance Volumes

By compressing our irradiance functions using spherical harmonics, it is possible to precompute and store an entire volume of samples for many points throughout a static scene. This precomputation step is key to real-time applications of irradiance volumes since, at run time, there may not be sufficient time to compute new irradiance values. An efficient data structure for storing the irradiance volume is needed so that samples in the volume may be quickly indexed using their world-space positions.

Uniform Subdivision

Subdividing a scene into evenly spaced cells is one way to generate and store irradiance samples. At preprocess time, the scene's bounding box is uniformly subdivided into a grid of cells. Irradiance samples should be computed for each of the eight corners of all the cells. At run time an object's world-space centroid may be used to traverse the grid's hierarchy to determine which cell contains the centroid. Now the eight samples at the cell's corners are used to perform trilinear interpolation (or higher-ordered interpolation if you've also stored irradiance gradients), and the approximate irradiance at the object's centroid is found.

The degree to which you should subdivide a given volume is dependent on the complexity of the scene you are sampling. Simple scenes may only need a few levels of subdivision, while large, complex scenes will require higher levels of subdivision to reduce lighting error. A uniform grid is easy to implement but quickly becomes unwieldy for large, complex scenes that require many levels of subdivision (see Figure 6.1.5). Choosing an adaptive subdivision scheme such as an octree will allow you to only subdivide the volume where subdivision is beneficial.

FIGURE 6.1.5 A uniform subdivision of this scene would be inefficient. Ideally, more irradiance samples would be stored in and around the central building structure, where lighting changes more rapidly than in areas such as the sky or outlying areas of the scene.

Adaptive Subdivision

Knowing which areas of your scene need further subdivision is a challenge. For example, a character standing just inside a house will appear shadowed on a sunny day, but if the character moves over the threshold of the door and into the sunlight, he should

appear much brighter; irradiance can change very quickly. We need a way to find areas of rapidly changing irradiance so that these areas can be more finely subdivided.

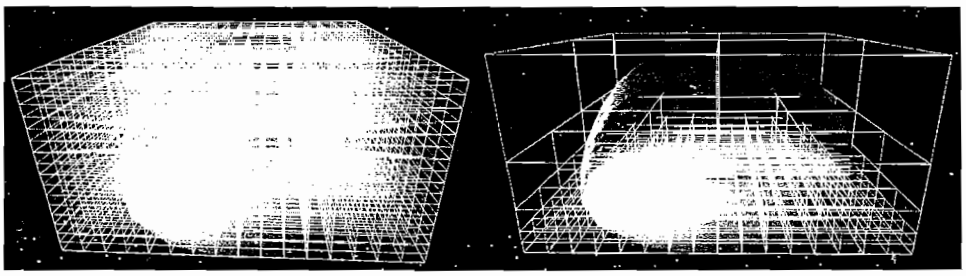

FIGURE 6.1.6 The scene on the left uses a uniformly subdivided irradiance volume. The scene on the right uses an adaptively subdivided irradiance volume.

Since irradiance sampling is done as a preprocess, one option is to use a brute force method that starts by super-sampling irradiance using a highly subdivided uniform grid. After this super-sampled volume is found, redundant cells may be discarded by comparing irradiance samples at child nodes using some error tolerance to determine if a cell was unnecessarily subdivided. This brute force method isn't perfect, though, because it assumes you know the maximum level of subdivision or super-sampling that is needed for a given scene. Instead, certain heuristics may be used to detect cells that might benefit from further subdivision.

Adaptive Subdivision Heuristics

Measuring irradiance gradients and flagging cells where the irradiance is known to change quickly with respect to translation (large gradient) is one way to test if further subdivision is necessary [Pharr04]. Testing gradients isn't perfect, though, because this will only subdivide areas where you know that irradiance changes rapidly. There may still be areas that have small gradients but contain subregions with quickly changing irradiance.

Greger [Greger98] suggests subdividing any cells that contain scene geometry, while Pharr and Humphreys suggest in [Pharr04] that it's better use the sample's distance from nearby geometry to determine when subdivision is needed. They use the harmonic mean of the depth values as their heuristic (explained below). The idea is that areas that contain a lot of geometry will have more rapidly changing irradiance. This assumption is based on the observation that the more geometry there is surrounding a sample point, then the more opportunities there are for shadows and bounced lighting to occur. We can test for this by rendering the scene's depth into a cube map at each irradiance sample point. This depth cube map is then read back into system memory, and the harmonic mean of the depth values is found:

$$HM = N \Big/ \sum_i^N 1/d_i,$$

where N is the total number of texels in the cube map and d_i is the depth value of the ith texel. The harmonic mean gives us a maximum distance for the sample's usefulness. If the neighboring irradiance samples are further away than this maximum distance, their associated cells should be subdivided. Subdivision may also be desirable if the harmonic mean changes dramatically between neighboring samples. The harmonic mean is chosen over the arithmetic mean (or linear average) because large depth values—owing to infinite depth if no geometry exists in a given direction—would quickly bias the arithmetic mean to a large value. For our purposes, samples with small harmonic means are very important because they indicate potential changes in irradiance that need to be accounted for.

We can further subdivide by testing for cells that contain scene geometry. This test can be performed by examining the depth cube map that we read back. Care must be taken here since the depth cube maps are sampled at each corner of the cell, so some of the texels in the cube map don't correspond to areas within the cell. Alternatively, occlusion queries (which are supported by many consumer-level GPUs these days) can be used to detect scene-cell intersections. We perform occlusion queries by placing a camera at the center of the cell we want to test, and for each face of the cell we render a quad that covers the cell's face and then render the scene. If any of the scene's draw calls pass the occlusion query, we know that a part of the scene must penetrate the cell and thus the cell should be subdivided.

When writing a preprocessing tool for automatically subdividing and sampling a scene, it is important to provide your artists with controls for specifying a minimum and maximum level of subdivision. A minimum subdivision control is extremely useful since frequently the subdivision heuristics will fail to detect that the coarsest cell (i.e., the scene's bounding box) needs further subdivision since the cell corners are typically very far from the majority of the scene's geometry. You will also want to provide thresholds for each of your subdivision heuristics so an artist can control how strongly a particular heuristic can influence whether subdivision should occur. Finally, after you've fully subdivided and sampled the volume, it's useful to go back through the octree and reject any redundant samples; if a cell has been subdivided and its children do not differ significantly from the parent, these child samples may be culled.

Render Time

Now that we have a volume of irradiance samples, we'll want to use this volume for lighting the dynamic objects in our scene. Because we've used an octree to store the irradiance samples, searching the tree for the cell that contains an object's centroid is straightforward. We start at the root of the tree and determine which one of the root's eight children contains our object's centroid. If this child node is subdivided, we then perform the same test on each of its eight children. This traversal continues down the octree's hierarchy until we reach a node that hasn't been subdivided. This will be the

smallest cell in the octree that contains our object's centroid, and we can use this cell's eight irradiance samples (one at each corner) to interpolate an irradiance distribution function at our object's centroid. Some care must be taken when interpolating the irradiance samples for a particular cell. It's important to realize that simple trilinear interpolation may only be used with a uniformly subdivided octree, or popping will occur in our lighting results as an object passes between cells that have different levels of subdivision, as shown in Figure 6.1.7.

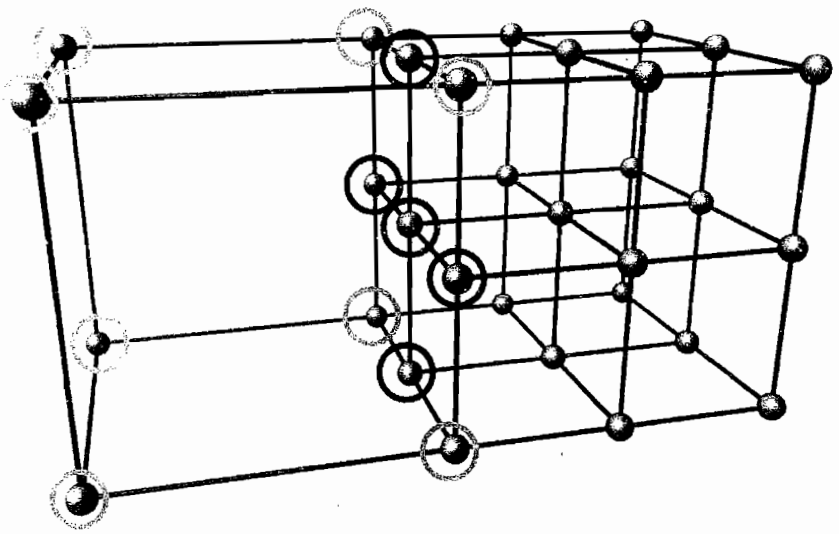

FIGURE 6.1.7 When transitioning between cells that have been adaptively subdivided, trilinear interpolation can produce popping artifacts. The large cell on the left uses the samples marked with gray circles, while the smaller cells use additional samples that have been marked with black circles. When moving from a small cell to the large cell, the additional samples will suddenly be ignored. Popping may be reduced by using a distance-weighted average of all nearby samples. To eliminate popping entirely, methods for scattered data interpolation may be used (a full survey and description of which is beyond the scope of this article).

Instead of using trilinear interpolation, a distance-weighted average of all nearby samples (including samples from neighboring cells) may be employed. Annen et al. [Annen04] describe a method that may be used in a vertex shader for computing an interpolated irradiance sample at each vertex. This scheme requires that a set of nearby irradiance samples, along with their gradients, is sent to the GPU. A vertex shader then uses a distance weighting scheme to perform per-vertex interpolation of the spherical harmonic coefficients.

$$I_i' = \sum_j w_j \left(I_i^j + (\nabla I_i^j \cdot d_j) \right).$$

At each vertex we find the weighted sum of each extrapolated irradiance sample. I_i' is the ith spherical harmonic coefficient of the extrapolated irradiance function, I_i^j is the ith spherical harmonic coefficient of the jth nearby irradiance sample, ∇I_i^j is the irradiance gradient for the ith irradiance coefficient of the jth sample, and $\boldsymbol{d}_j$ is a nonunit vector from the original sample location to the point being rendered. Each sample's weight w_j is computed based on its distance from the sample point s_j to the point p being rendered. Annen et al. [Annen04] use the following weighting:

$$w_j = \frac{(1/\|p - s_j\|)^b}{\sum_j \left(1/\|p - s_j\|\right)^b}.$$

The exponent b controls the distance falloff of the weighting function. Larger values of b will decrease the importance of samples as they increase in distance from p and will increase the importance of samples that are closer to p.

Conclusion

A realistic lighting technique using irradiance volumes for rendering dynamic characters in static scenes has been presented. This irradiance volume uses spherical harmonics to compactly store and efficiently evaluate irradiance samples for many points in a scene. Along with irradiance samples, irradiance gradients may also be stored to increase the usefulness of each irradiance sample by allowing for new irradiance samples to be extrapolated from stored irradiance samples. Gradients also allow for more accurate interpolation of irradiance samples, which will help reduce popping when interpolating samples stored in an adaptively subdivided octree. A sample application, with source code, is provided (on the CD-ROM that accompanies this book) that implements many of the techniques described in this article.

References

[Annen04] Annen, Tomas, Jan Kautz, Fredo Durand, and Hans-Peter Seidel. "Spherical Harmonic Gradients for Mid-Range Illumination" *Proceedings of Eurographics Symposium on Rendering*, June 2004.

[Arvo95] Arvo, J. "Analytic Methods for Simulated Light Transport" PhD thesis, Yale University, December 1995.

[Brennan02] Brennan, C. "Diffuse Cube Mapping." *Direct3D ShaderX: Vertex and Pixel Shader Tips and Tricks*, edited by Wolfgang Engel. Wordware Publishing, 2002: pp. 287–289.

[Cohen93] Cohen, Wallace. "Radiosity and Realistic Image Synthesis" Academic Press Professional, Cambridge, 1993.

[Debevec98] Debevec, Paul E. "Rendering Synthetic Objects into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography." *SIGGRAPH* 1998.

[Green03] Green, R. "Spherical Harmonic Lighting: The Gritty Details." 2003. Available online at *http://www.research.scea.com/gdc2003/spherical-harmonic-lighting.html*

[Greger98] Greger, G., P. Shirley, P. Hubbard, and D. Greenberg. "The Irradiance Volume." *IEEE Computer Graphics & Applications*, *18*(2), (1998): pp. 32-43.

[Pharr04] Pharr, M. and G. Humphreys. "Physically Based Rendering" Morgan Kaufmann, San Francisco, 2004.

[Ramamoorthi01] Ramamoorthi, R., and P. Hanrahan. "An Efficient Representation for Irradiance Environment Maps." *SIGGRAPH*, 2001: pp. 497–500.

[Sloan04] Sloan, P.-P. "Efficient Evaluation of Irradiance Environment Maps." *ShaderX²: Shader Programming Tips and Tricks with DirectX 9*, edited by Wolfgang Engel. Wordware Publishing, 2004: pp. 226–231.

[Ward92] Ward, G. and P. Heckbert. "Irradiance Gradients." *Third Eurographics Workshop on Rendering* (1992): pp. 85–98.