# Cascaded Light Propagation Volumes for Real-Time Indirect Illumination

Anton Kaplanyan*
Crytek GmbH

Carsten Dachsbacher†
VISUS / University Stuttgart

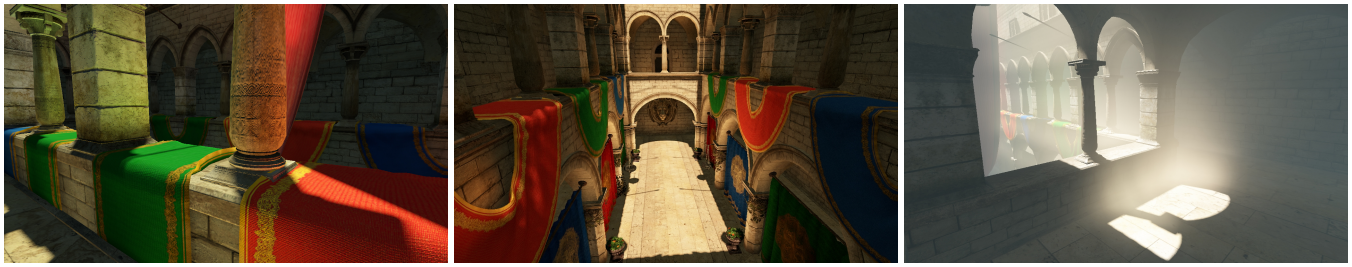

**Figure 1:** *Real-time rendering (58fps on a NVIDIA GTX285 at* $1280 \times 720$ *resolution) of the "Crytek Sponza" scene (262k triangles, available at http://www.crytek.com/downloads/technology/) using our method for indirect illumination with 3 cascades of light propagation volumes (LPVs, with 50, 25 and 12.5 meters grid spacing; the scene extend is about* $37 \times 15 \times 22m^3$*). Lights, camera, and geometry can be fully dynamic. The time required for the computation of the indirect illumination is about 10 milliseconds only. Right: We add participating media (single-scattering) effects rendering at 34 fps by ray marching through the LPV. The overhead for ray marching is about 18ms.*

## Abstract

This paper introduces a new scalable technique for approximating indirect illumination in fully dynamic scenes for real-time applications, such as video games. We use lattices and spherical harmonics to represent the spatial and angular distribution of light in the scene. Our technique does not require any precomputation and handles large scenes with nested lattices. It is primarily targeted at rendering single-bounce indirect illumination with occlusion, but can be extended to handle multiple bounces and participating media. We demonstrate that our method produces plausible results even when running on current game console hardware with a budget of only a few milliseconds for performing all computation steps for indirect lighting. We evaluate our technique and show it in combination with a variety of popular real-time rendering techniques.

**CR Categories:** I.3.3 [Computer Graphics]: Picture/Image Generation—Display Algorithms; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Shading

**Keywords:** global illumination, real-time rendering

## 1 Introduction

Many recent papers introducing methods for computing global illumination state that this is a hard problem for interactive applications – surely a valid statement due to the inherent complexity of light transport. Despite significant advances in recent years, rendering indirect illumination in real-time, without trade-offs or precomputation, is still an elusive goal. Since indirect illumination is known to be perceptually important, many attempts have been made to coarsely approximate it, with as little computation time as possible. There is consensus that an accurate computation thereof is often not necessary [Yu et al. 2009]. Recent work exploits this fact, for example, by using interpolated visibility [Křivanek and Colbert 2008], imperfect visibility [Ritschel et al. 2008], or by using screen-space techniques [Ritschel et al. 2009b].

In this paper, we present a novel method for rendering plausible indirect lighting for fully dynamic scenes where no precomputation is feasible. It has been designed having strict real-time requirements in mind and targets a computation time of only a few milliseconds per frame on contemporary graphics hardware for PCs and consoles. When adjusting the parameters of previous methods to meet these requirements, the approximate solutions usually exhibit disturbing artifacts such as a splotchy appearance with methods based on instant radiosity [Keller 1997], or apparent patches in radiosity methods, such as [Dachsbacher et al. 2007]. Even screen-space techniques, e.g. [Dachsbacher and Stamminger 2005; Ritschel et al. 2009b] that compute subsets of light transport paths, exhibit noise or banding when limiting the computation time, and typically introduce further trade-offs such as short-distance indirect light only.

Our method, which is a further development of the basic technique presented in the SIGGRAPH'09 course program [Tatarchuk et al. 2009], does not share these problems and produces plausible, visually pleasing results without flickering and high temporal coherence even with very limited computation time. We achieve this by using a lattice storing the light and the geometry in a scene. The directional distribution of light is represented using low-order spherical harmonics. We sample the surfaces using reflective shadow maps and use this information to initialize the lattice from scratch every frame. Based on this representation, we developed a data-parallel light propagation scheme that allows us to quickly, and plausibly, approximate low-frequency direct and indirect lighting including fuzzy occlusion for indirect light. Our propagation scheme is derived from the Discrete Ordinates Method [Chandrasekhar 1950] and we show that a simplified, more local, scheme is sufficient for plausible indirect lighting. Direct lighting from point lights, or small area lights, is computed using standard techniques such as shadow mapping. Our method has been integrated into the CryENGINE® 3, and handles large, fully dynamic scenes through nested lattices of different resolutions. We focus on plausible approximations for real-time applications rather than on physically-correct rendering of indirect illumination. Our method is best suited for low-frequency indirect lighting from diffuse surfaces; we discuss the limitations arising from spatial discretization, low-order spherical harmonics, and the propagation scheme.

*e-mail: antonk@crytek.de
†e-mail: dachsbacher@visus.uni-stuttgart.de

## 2 Previous Work

A tremendous amount of research has been conducted in the field of global illumination (GI). We focus on previous work that has been presented in the context of interactive rendering. For a more comprehensive overview of non-interactive methods we refer to [Dutré et al. 2006]. We roughly classify the methods for interactive applications into five categories.

**Classic methods** Huge progress has been made in the field of ray tracing. Although recent work achieves interactive performance with complex lighting effects [Wang et al. 2009], these methods are still not applicable to real-time applications with complex scenes. Recently, variants of radiosity methods tailored for graphics hardware have also been introduced. Dong et al. [2007] achieves interactive global illumination for small scenes, where visibility was evaluated directly on the hierarchical link mesh. Explicit visibility computations can be replaced by an iterative process using anti-radiance [Dachsbacher et al. 2007]. Although interactive GI in moderately complex scenes becomes possible, the use of dynamic objects is restricted. Bunnell [2005] coarsely approximates indirect illumination and ambient occlusion using a finite element technique allowing for deforming objects.

**Precomputed and low-frequency radiance transfer** Many techniques for real-time GI precompute the light transport including all visibility information which entails restrictions such as static [Sloan et al. 2002] or semi-static scenes [Iwasaki et al. 2007]. Hašan et al. [2007] approximate GI by many point lights and present a scalable GPU-technique for high-quality images targeting rendering times of few seconds. The radiance transfer is often represented in spherical harmonics (SH) basis. Recently, the limitations of these methods have been eased, e.g. Sloan et al. [2007] demonstrate real-time indirect illumination for low-frequency incident lighting and visibility when these are represented with a small number of SH.

**Screen-space methods** In recent years, GPU-friendly techniques operating in image space became popular, and are presently widely used. As any standard shadow map, the reflective shadow map (RSM) [Dachsbacher and Stamminger 2005] captures directly lit surfaces, but stores additional information that is required to compute the indirect illumination from these surfaces such that each pixel can be seen as a small light source. Rendering indirect illumination from the RSM can be implemented via sampling a subset of the pixel lights. Screen space ambient occlusion [Mittring 2007; Bavoil et al. 2008] is part of almost any real-time rendering engine nowadays. Recently, Ritschel et al. [2009b] extended these methods by accounting for directional lighting and show colored shadows and indirect illumination. Note that these techniques compute light transport over small distances (in image space) only, and typically require post-processing to reduce sampling artifacts. Image Space Photon Mapping recasts the initial and final photon bounces of traditional photon mapping as image-space operations on the GPU [McGuire and Luebke 2009], and achieves interactive to real-time frame rates for complex scenes. Ritschel et al. [2009a] accellerate final gathering with a parallel micro-rendering technique running on the GPU. They support BRDF importance sampling and report interactive frame rates for complex scenes.

**Instant radiosity methods** The entire lighting in a scene can be approximated by a set of virtual point lights (VPLs) [Keller 1997], and techniques based on this instant radiosity idea have gained much attention in recent years. RSMs can be used to create VPLs for one bounce of indirect light, and their contribution can be accumulated in screen space using splatting [Dachsbacher and Stamminger 2006], or multi-resolution splatting [Nichols and Wyman 2009]. The latter computes the indirect lighting at a lower resolution for smooth surfaces, and more accurately where geometric detail is present. This idea has been further improved with smart clustering of the RSM's pixels [Nichols et al. 2009]. Note that all aforementioned methods compute one-bounce indirect illumination without occlusion only. Ritschel et al. [2008] present an efficient method to quickly generate hundreds of imperfect shadow maps. These allow the approximation of the indirect illumination from VPLs with sufficient quality. However, in order to avoid flickering and to provide temporal coherence a large number of VPLs is required (typically hundreds to thousands), and obviously this takes a toll on performance and prevents high frame rates in dynamic scenes.

**Lattice-based methods** The Discrete Ordinates Method (DOM) [Chandrasekhar 1950] discretizes the quantities in the radiative transfer equation (RTE) in space and orientation (DOMs are typically used for computing radiative transfer in participating media). These discrete values are used to approximate the different terms in the RTE: the radiance distribution is stored in a 3D grid, and light is exchanged between neighboring volume elements, reducing the computation to local interactions only. A variant of these methods postulates a simple photon transport model that describes a diffusion process to compute light transport in participating media based on the lattice-Boltzmann method [Geist et al. 2004]. Recently, Fattal [2009] presented an improvement of DOMs to reduce light smearing (due to repeated interpolation) and ray effects (due to discretized directions). Note that although these methods are efficient, and (potentially) highly parallel, they do not run at interactive frame rates.

## 3 Light Propagation Volumes

Our work is inspired by the discrete ordinate methods and the Lattice-Boltzmann lighting technique (as such there is a conceptual similarity to volume grid radiosity and grid-based fluid advection methods). Similar to these approaches, we represent the lighting in a scene sampled on a lattice. This allows us to model the light transport using simple, local operations that in turn can be easily parallelized. We base our computation on intensity stored in each cell of this lattice. Our contributions include a fast initialization of the lattices for light propagation that can be computed from scratch for every frame, a novel propagation scheme that propagates along the main axial directions (opposed to 26 directions in other schemes) while still providing good results for low-frequency lighting, and a hierarchical approach for handling large scenes.

It is well known that grid based methods suffer from two major errors. The so-called ray effect stems from the discretization of the directions in which light can travel and causes distracting beams of light. Repeated averaging during the propagation itself further causes light smearing in space. It is evident that such methods – which are typically applied in the domain of participating media – are not well suited for high-frequency (direct) lighting. For these reasons we follow the idea of Ramankutty and Crosbie's modified DOM [1997], and use the grid-based lighting for low-frequency light only, where these errors are tolerable, especially in the real-time rendering context of our target applications.

In the following we describe the basic steps of our method. For this we assume that our scene is embedded into a 3D grid of a fixed resolution (we remove this limitation by introducing cascaded grids in Sect. 4). We have two grids, one storing the intensity that is initialized from the surfaces causing indirect lighting or low-frequency direct lighting; and a second grid that stores a volumetric approximation of the scene geometry and is used for fuzzy blocking as the light travels through the scene. Both grids store a spherical function represented as low-frequency SH approximation. We discuss difficulties arising from coarse light propagation volumes in Sect. 7.

Computing the indirect lighting with our technique consists of four subsequent steps:

- Initialization of the light propagation volume (LPV) with the surfaces causing indirect lighting and low-frequency direct light (area light sources).
- Sampling of the scene's surfaces using depth peeling from the camera, and multiple reflective shadow maps. This information is used to create a coarse volumetric representation of blocker geometry.
- Light propagation starting from the initial LPV and accumulating the intermediate results (yielding the final light distribution).
- Lighting the scene geometry using the propagated light. In addition to directly applying the LPV for diffuse lighting, we also present plausible approximations to glossy indirect lighting and participating media.

### 3.1 LPV Initialization

We use the light propagation volume (LPV) to compute the low-frequency lighting in a scene only, i.e. mainly the indirect light. The first step is to transform the surfaces causing that lighting into the directional intensity representation and initialize the LPV accordingly. Strong direct lighting and shadowing from point or directional light sources is computed using traditional techniques such as shadow mapping.

The initialization is based on the idea that one can convert the low-frequency lighting into a set of *virtual point lights* (VPLs), in the spirit of [Keller 1997]. However, we use a significantly larger number of VPLs than typical instant radiosity methods, since we do not compute their contribution individually, but only use them to initialize the LPV.

**Indirect light** We first create VPLs accounting for indirect lighting by rendering a reflective shadow map (RSM) [Dachsbacher and Stamminger 2005] for every light source. A RSM is an extended shadow map that can be quickly created in a single render pass on the GPU. It captures the directly lit surfaces that cause the first-bounce indirect lighting. Each texel of a RSM can be interpreted as a small area light source with a spectral and directional intensity distribution, $I_p(\omega)$, determined by the orientation $\mathbf{n}_p$ of the texel and its reflected flux $\Phi_p$:

$$I_p(\omega) = \Phi_p \left\langle \mathbf{n}_p | \omega \right\rangle_+ ,$$

where $\langle . | . \rangle_+$ denotes the dot product with negative values clamped to zero; we omit spectral dependence here for simplicity. The next step is to transform all VPLs into the SH representation and store their contributions in the LPV cells. We can easily determine the cell in which the VPL resides. However, if the VPL points away from that cell's center, we do not want to add its contribution to this cell, but rather to the next cell in order to avoid self lighting and shadowing. For this, we virtually move each VPL by half the cell spacing in the direction of its normal, before determining the cell. In that cell, we only take the orientation of the VPL into consideration, i.e. we ignore its exact positioning inside that cell.

We use SH to represent the directional distribution of intensity. Using $n$ bands of SH yields $n^2$ coefficients, $c_{l,m}$, for the basis functions, $y_{l,m}(\omega)$, both indexed by the band $l$ and degree $m$ with $-l \leq m \leq l$. We can easily derive analytical expressions for the SH coefficients of a clamped cosine-lobe centered around a given direction vector, the VPL normal $\mathbf{n}_p$ is this case. A clamped cosine oriented along the z-axis can be expressed in zonal harmonics [Ramamoorthi and Hanrahan 2001], and rotated to the direction $\mathbf{n}_p$ [Sloan 2008]. We scale these coefficients by the flux to obtain the SH coefficients for a VPL.

Note that the flux of a VPL already accounts for the area of the pixel from which a VPL has been created. Thus there is no further scaling required, and we can accumulate the intensities, i.e. the coefficients of the SH approximation, in the grid cells. In fact, we use 3 coefficient vectors to represent RGB data; however, for the explanation of our method we shown only one component.

Every VPL is treated in this way and then "injected" into the LPV. After determining the grid cell we simply accumulate the SH coefficients. Note that this process introduces spatial discretization of the lighting and inherently assumes that the surfaces causing indirect lighting do not occlude each other inside a single cell.

**Low-frequency direct light** The second kind of VPL accounts for low-frequency direct lighting from area lights, environment maps and larger groups of point lights, such as those stemming from particle systems. Again, we create a dense sampling, i.e. several hundreds to thousands of VPLs for these light sources, and inject them into the LPV in exactly the same way as those created from RSMs. VPLs from environment maps are injected into the outer layer of the LPV cells; other VPLs outside the grid are omitted.

The result of the LPV initialization pass is a grid of initial intensity distributions that is used to compute the light propagation in the scene. In order to reduce the number of injected VPLs, we also examined clustering similar to the light cuts method [Walter et al. 2005]. However, the injection pass proved to be very cheap and clustering mostly did not amortize in our experiments.

### 3.2 Scene Geometry Injection

In addition to the initial intensity distribution we create a volumetric representation of the scene's surfaces. This coarse approximation of the geometry is used for blocking the light during propagation and thus for computing indirect shadows.

We aim at fully dynamic scenes without precomputation and consequently this information has to be created on the fly as well. To this end, we re-use the sampling of the scene's surfaces that is stored in the depth and normal buffers of the camera view (using a deferred renderer) and in the RSMs. Note that we typically created RSMs for numerous light sources and thus have a dense sampling of a large portion of the scene's surfaces. If required, we can gather more information by adding depth-peeling passes for the RSMs or the camera view. It is also possible to use a precomputed point sampling of the surfaces, similar to [Ritschel et al. 2008], but this implies additional storage and transformation costs.

**Fuzzy occlusion** Each sample represents a small surface element (*surfel*) with a given location, orientation, and size. We model the occlusion in spirit of [Sillion 1995], and assume that we can use the accumulated blocking potential of surfels in a grid cell as a probability for blocking light from a certain direction going through that cell. By this, we can render soft shadows, but surfaes smaller than the grid size, e.g. foliage, do not produce resolved shadows.

The amount of blocking by one of these surfels depends on its size, and on the cosine of the angle between its normal and the light direction in question. The blocking probability of a single surfel with area $A_s$, and normal $\mathbf{n}_s$ in a cell of grid size $s$ is thus $B(\omega) = A_s s^{-2} \langle \mathbf{n}_s | \omega \rangle_+$. Note that we assume that scene objects are closed surfaces and thus use a clamped cosine lobe for the

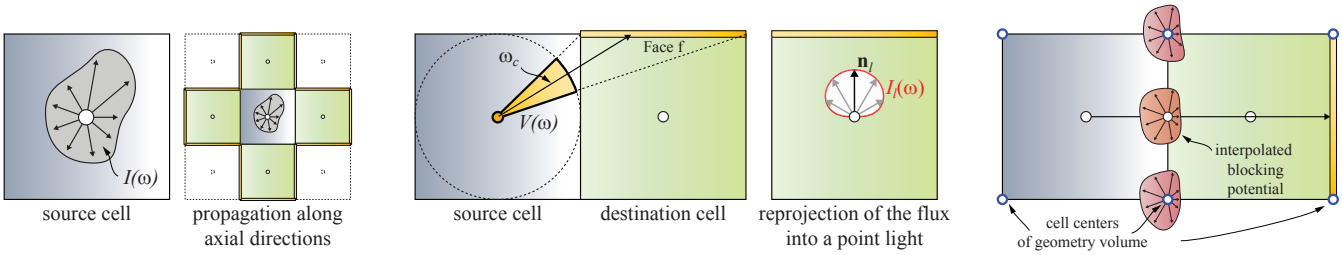


**Figure 2:** *Left: Each cell of the LPV stores the directional intensity used to compute the light that is propagated from a source cell to its 6 (4 in 2D) neighbors. Center: We compute the flux onto the faces of the destination cell to preserve directional information. Right: We account for fuzzy occlusion by storing a volumetric representation of the scene in a second grid.*

blocking probability. This is also because a low-order SH projection of an absolute cosine degrades to a near isotropic function.

**Injection** Similar to the VPL injection we accumulate the SH projections of the blocking potential into the geometry volume (GV). This accumulation process is correct if there is only one surface intersecting a grid cell as the mutual occlusion of surface samples within a cell cannot be resolved (this is ameliorated by the use of screen-space ambient occlusion; see Sect. 7). The GV has the same resolution as the LPV, but is shifted by half a cell, such that its cell centers are located at the corners of the LPV cells to achieve better interpolation of the blocking potential during light propagation.

A surface might be sampled in the camera view and one or more RSMs, and we need to ensure that its blocking is not accumulated multiple times. For this, each buffer is accumulated into a separate GV and after injection the GVs are merged into one using a maximum operation on the SH-vectors. As an optimization we can also reuse the surface samples from previous frames (for static geometry), which reduces the cases where surfaces are not captured by the current view or RSMs.

### 3.3 Propagation Scheme

The inherent idea of grid-based methods, such as DOM, is that light propagation through the scene is computed using successive local iteration steps. In this regard, our method is no exception, but the propagation scheme itself differs and is detailed in this section.

**Intensity propagation** The input for the first iteration step is the initial LPV from the injection stage; subsequent iterations take the LPV from the previous iteration as input. Each cell stores the intensity as a SH-vector and the light is then propagated to its 6 neighbors along the axial directions (Fig. 2, left, shows the 2D case with 4 axial directions). In the following, we describe the propagation from one source to one destination cell only; the propagations along the other directions are computed analogously. Let us denote the SH-approximation of the intensity of the source cell as $I(\omega) \approx \sum_{l,m} c_{l,m} y_{l,m}(\omega)$. Next we compute the flux onto each of the faces of the adjacent destination cell. For this we define the visibility function, $V(\omega)$, of a face $f$ with respect to the source cell's center. $V(\omega) = 1$ if a ray starting at the source cell center in direction $\omega$ intersects the face, otherwise $V(\omega) = 0$. Fig. 2 (center) shows $V(\omega)$ for the top face of the destination cell. The total flux reaching the face can be computed by integrating over directions using $V(\omega)$ as $\Phi_f = \int_\Omega I(\omega) V(\omega) d\omega$.

The faces' visibility functions can be projected into SH yielding a coefficient vector $v_{l,m}$ with $V(\omega) \approx \sum_{l,m} v_{l,m} y_{l,m}(\omega)$. The integral over intensity times visibility can then be easily computed using the dot product of the SH-vectors $c_{l,m}$ and $v_{l,m}$. These "transfer vectors" $v_{l,m}$ can be precomputed once and stored for the propagation scheme. The problem, however, is that the integral value can be very inaccurate for low-order SH approximations, and thus we propose using a different strategy for this case. Instead of a transfer vector we compute the solid angle $\Delta\omega_f = \int_\Omega V(\omega) d\omega$ of each face in the destination cell, and determine the central direction $\omega_c$ of the visibility cone. The flux reaching the face is then computed as $\Delta\omega_f / (4\pi) \cdot I(\omega_c)$. Effectively this means that we take the intensity in direction $\omega_c$ as average intensity over the solid angle.

**Reprojection** Using this propagation we obtain the incident flux for each face of a destination cell and then transform it into outgoing intensity for the subsequent propagation again. For this we compute the intensity of a new point light source (with the same emission characteristics as our VPLs) at the destination cell's center pointing towards the face and *causing exactly as much flux as the face received* due to the propagation. That is, the flux on the face, $\Phi_f$, is equal to the total emitted flux of the point light: $\Phi_f = \int_\Omega \Phi_l \langle \mathbf{n}_l, \omega \rangle_+ d\omega$, and thus $\Phi_l = \Phi_f / \pi$. Similar to the light injection stage, we scale the clamped cosine lobe by $\Phi_l$ and accumulate the SH coefficients, for the new point light located at the cell's center, in the destination cell for the next iteration.

The propagation is computed for each source cell and each of its adjacent cell's faces (shown yellow in Fig. 2). Note that this process conserves energy that is expected from light propagation in vacuum. However, the propagation together with the reprojection introduces spatial and directional discretization. Note that this is common to all lattice-based methods, and we compare our propagation scheme in flatland to a reference solution and DOMs in Sect. 5.

**Blocking** We also need to integrate the blocking of light due to scene geometry into the propagation step. In the geometry injection stage we computed the GV from the surfaces in the scene that stores anisotropic occlusion probabilities for exactly this purpose. The GV is displaced by half the grid size with respect to the LPV. By this, a cell center of the GV resides on a corner of an LPV cell. Whenever we propagate from a source to a destination cell, we bi-linearly interpolate the GV's SH-coefficients at the center of the face through which we propagate, and evaluate the occlusion for the propagation direction to attenuate the intensity. Note that we do not consider this occlusion in the very first propagation step after injection in order to prevent self-shadowing.

**Iterations** The result after computing all propagations in the LPV is accumulated in a separate 3D grid after each iteration: the sum of all intermediate results is the final light distribution in the scene. The number of required iterations depend on the resolution of the grid. Similar to [Geist et al. 2004] we use two times the longest

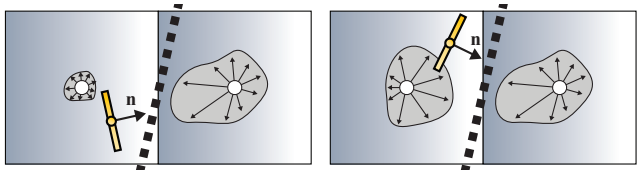


**Figure 3:** *Such intensity distributions can result from blockers between cells (indicated by dashed lines). We detect the discontinuity in the intensity by computing its gradient in normal direction* $\mathbf{n}$.

side of the LPV iterations as a heuristic for the scheme as described above. Obviously this is not feasible for our real-time requirements for reasonably sized LPVs. The multi-resolution scheme introduced in Sect. 4 yields similar results with less iterations.

Obviously there is inherently strong blurring caused by the propagation scheme. However, since we use this method for indirect light only, the results – especially for low-band SH approximations that we use for real-time rendering – are of sufficient quality and can be computed very efficiently (discussed in Sect. 5 and 7).

### 3.4 Using the Light Propagation Volume for Rendering

The accumulated results of all iterations represent the light distribution in the scene. In the simplest form we query the intensity by a tri-linearly interpolated lookup of the SH coefficients. We then evaluate the intensity function for the negative orientation of the surface, similar to irradiance volumes [Greger et al. 1998; Oat 2006]. However, since we store intensity we need to convert it into incident radiance and due to spatial discretization we assume that the distance between the cell's center (where the intensity is assumed to be), and the surface to be lit is half the grid size $s$.

One problem of coarse volumes paired with low-frequency approximations of the lighting is self-illumination and light-bleeding. We found that a dampening factor based on the directional derivative of the intensity distribution greatly reduces these artifacts. For a surface location $\mathbf{x}$ and normal $\mathbf{n}$, we determine the tri-linearly interpolated SH coefficients $c_{l,m}$ and the directional derivative in normal direction $\nabla_{\mathbf{n}} c_{l,m}$ (computed via differencing). Whenever the the derivative is large, and $c_{l,m}$ and $\nabla_{\mathbf{n}} c_{l,m}$ are deviating, we dampen $c_{l,m}$ before computing the lighting (Fig. 3 shows two such typical situations).

## 4 Cascaded Light Propagation Volumes

Using a single LPV to compute the light propagation in an entire scene (with acceptable resolution) would require a very large grid. Instead we use a set of nested grids moving with the viewer (Fig. 4), similar to geometry clipmaps [Losasso and Hoppe 2004] but in 3D. The grids are not exactly centered around the viewer, but displaced into the view direction. This provides high spatial resolution in parts of the scene close to the camera, and also covers distant parts with lower resolution. The nested grid approach allows us to use grids of smaller size (typically $32^3$ cells) and thus reduce the number of required propagation iterations.

### 4.1 Nested Grid Propagation

Our method as described in Sect. 3 remains largely unchanged. VPLs and geometry are injected into all nested grids at the same time. However, if the size of an object (determined using its bounding box) is smaller than the cell size of a grid, we do not create VPLs from this object by not rendering it to the RSM. This is to prevent aliasing artifacts (see next section).

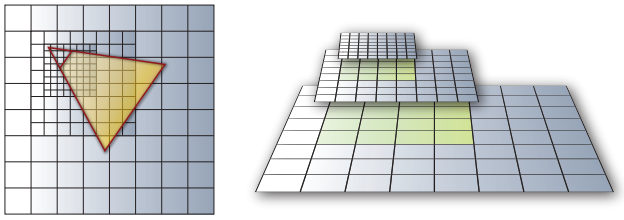

**Figure 4:** *We use a set of nested, or cascaded, LPV and GV grids to represent large scenes with fine resolution close to the camera, and coarse resolution further away.*

Light propagation is computed for all grids independently. Note that VPLs in a fine grid have also been injected into the coarser grids (green cells in Fig. 4). This ensures that indirect light from nearby objects bleeds into distant parts of the scene. When using the LPVs for lighting, we look up the finest grid at the respective location. At the boundary of a fine grid we create a smooth transition to the next-coarser grid by interpolation between both levels (similar to [Losasso and Hoppe 2004]). Note that the indirect light due to VPLs that have been injected into a coarser grid (but not into finer grids) is cut-off due to that blending. We did not, however, experience distracting or even visible artifacts from this cut-off.

### 4.2 Coherent Solutions in Dynamic Scenes

Providing a stable solution under object, camera, and light movement is very important for interactive applications. The primary cause for potentially inconsistent light propagation stems from the spatial discretization in the LPV. To prevent this, we snap the cascaded LPVs – that move with the camera – to multiples of the grid size. By this "one-cell grid snapping" we maintain a consistent VPL and geometry injection under camera movement.

The second source of inconsistency – in other algorithms often causing flickering – is the sampling of light emitting and reflecting surfaces. As we can afford to excessively sample these surfaces and inject a huge number of VPLs (several hundred thousand), there is a high redundancy of light emitters in the LPV. In contrast to other methods, such as instant radiosity based methods, this redundancy allows the rendering of indirect illumination from moving and complex, detailed objects, even such as foliage, without flickering.

However, we noticed that small moving objects – if their size is below the grid cell size – can cause artifacts in the form of unsteady moving indirect lighting. In this case the spatial discretization of the LPVs becomes noticeable. As an ad-hoc solution, we fade out the VPLs from such objects. This solution seems tolerable as the lower grid resolutions are further away from the camera.

## 5 Qualitative Evaluation of the Propagation

We show flatland (2D) scenes to discuss the deviations and approximation errors due to our propagation scheme. Fig. 5 plots light propagation in two simple settings with 1 and 2 area light sources computed analytically (gray), using the standard DOM propagation to all 8 neighbor cells (DOM8, green), and our method (LPV, red). For comparison, we also include a modified DOM scheme (DOM4, blue) that propagates to the main axial directions only, similar to ours, to demonstrate the effectiveness of computing the flux per face compared to a center-to-center transfer.

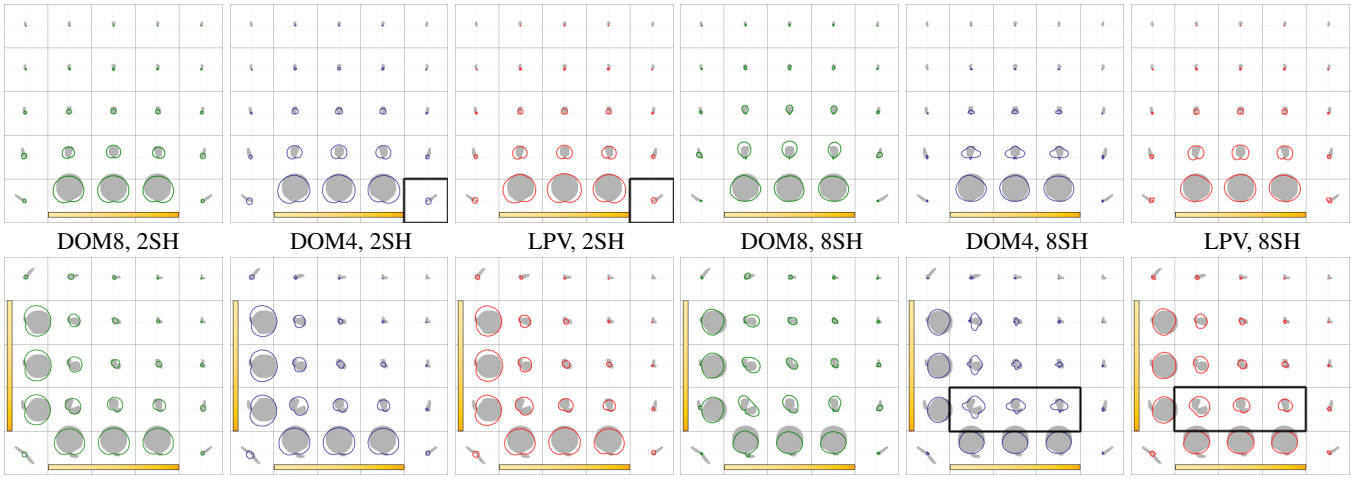


**Figure 5:** *Two simple settings in flatland (top row: 1 area light source, bottom row: 2 area lights in L-shape): for comparison we plot the intensity for each grid cell computed using DOMs with 8 and 4 propagation directions (green, blue), our propagation scheme (red), and analytically (grey).*

We observe that for few SH bands, all three yield comparable results with expectedly blurred intensity distributions. Our LPV scheme preserves the main intensity directions better than DOM4, e.g. light leaving at grazing angles (highlighted cells in top row), but obviously is blurry as well. For more SH bands, DOM4 produces very strong ray effects, whereas LPV produces no such artifacts (highlighted in bottom row). However, DOM8 preserves distribution much better. This leads us to the conclusion that our scheme is well-suited for few SH bands (2 to 4) and large light sources (larger than 1 cell). In these cases it yields good approximations, preserves the main direction of light transport well, and only requires 4 (6 in 3D) neighbor cells for propagation compared to 8 (26 in 3D) for DOM8.

## 6 Implementation Details

Our method has been integrated into the CryENGINE® 3: a multi-platform (DirectX 9 and 10 on PC, Sony PS3® and Microsoft Xbox 360®) real-time rendering engine. Due to the performance requirements the implementation uses only 2 SH-bands (4 coefficients) per color channel for the LPV. The cascade consists of 3 grids, each stored as three RGBA 16-bit floating point textures of size $32^3$ (QUVW8 format, i.e. 8-bit signed integer per component, on consoles). The geometry volume is of the same format and resolution as the LPV (but with only one texture and no spectral data), and the surfels are created from the depth and normal buffers of the camera view and RSMs. High-resolution buffers, e.g. the camera view, are down-sampled prior to the surfel injection.

The RSMs store a depth buffer (32-bit float), and normals and flux (RGBA 8-bit each) of size $256^2$ ($128^2$ for consoles). This amounts to $2^{16}$, or $2^{14}$, VPLs per primary light source. The VPLs are injected into the LPV cascade using point rendering. This requires either vertex texture fetches (used for DirectX 9/10, and Xbox 360), or rendering to vertex buffer (faster on the PS3).

For the light propagation we perform a GPU-friendly gathering procedure with ping-pong rendering: for every cell (in every cascade) we look up the intensity distributions of the 6 neighbor cells. We then compute the flux onto the faces, reproject the flux and accumulate the output in a render target. In most scenes 8 iterations provided visually pleasing results. Note that this limits the distance of light propagation and might produce artifacts in large, sparse scenes (see Fig. 12). The GPU memory required for our method (including ping-pong render targets) is $32^3 \times (2 \times 4) \times 3 \times (\#cascades + 2) = 3.75$MB for the LPV and $32^3 \times (2 \times 4) \times \#cascades = 0.75$MB for the GV (for the consoles the memory is halved due to the QUVW8 format). Note that storing spectral reflectivity would triple the GV memory and bandwidth; in our examples with multiple bounces we used monochromatic reflection for further bounces only. Table 1 shows detailed timings.

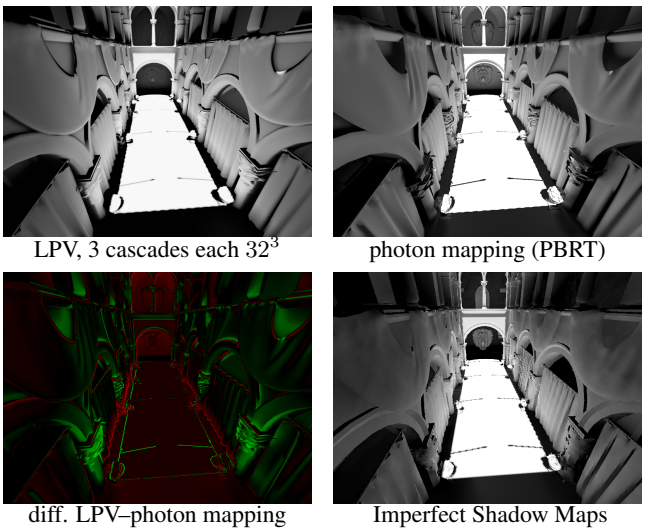


**Figure 6:** *Top left: rendering the Crytek Sponza (untextured for comparison) with LPVs, 3 cascades each $32^3$, at 60fps. Top right: ground-truth solution rendered with photon mapping and PBRT (single-bounce indirect light only, 200000 photons, 1024 final gather samples, approx. 45 minutes). Bottom left: difference image of the LPV rendering and the ground-truth solution; green regions are too bright with LPVs, red regions too dark. Bottom right: rendering with Imperfect Shadow Maps [Ritschel et al. 2008] at 15.6fps with 256 VPLs.*

## 7 Results and Discussion

Obviously our method computes a low-frequency approximation to the indirect lighting in a scene only. All high-frequency variations, i.e. surface details, are achieved using bump mapping and screen-space ambient occlusion. The results indicate that our method produces plausible renderings (the indirect light is exaggerated for illustration purposes) in many scenarios. Fig. 6 shows a comparison to ground-truth for the diffuse Crytek Sponza scene. Note that the images overall compare well, however, occlusion from nearby geometry in the LPV rendering is approximated by ambient occlusion and shows differences to the photon mapping image. All LPV renderings have been captured at 1280x720 resolution on a NVIDIA GTX285. In this section we describe further rendering techniques using the LPV, followed by a discussion of benefits and limitations.

**Multiple indirect bounces** We can easily modify the propagation step to account for (coarsely approximated) multiple diffuse bounces of indirect light (Fig. 7). When propagating from a source to a destination cell, we perform an additional lookup into the GV with an offset of one cell into propagation direction (Fig. 8, left). This yields a blocker approximation $B(\omega)$ that we use to immediately reflect the light that is propagated to the destination cell. We can estimate the diffuse illumination of the blocker geometry as $B(-\omega_c)$ (effectively computing the dot product of the incident light direction $-\omega_c$ and the surface normal), scaled by the propagated intensity and the reflectivity of the blocker geometry. The reflected light has the same directional distribution as the diffuse blocker geometry, and thus we scale $B(\omega)$ by the reflected intensity and inject it into the destination cell. Note that we only need to modify the propagation step causing an additional cost of approximately 0.3ms on a NVIDIA GeForce GTX 285 for a $32^3$ LPV and 8 iterations; all other steps remain unchanged.

**Glossy reflections** Obviously our propagation scheme causes strong blurring of the propagated light. However, we can use the LPVs to gather incident light for rendering glossy surfaces (Fig. 8 right, and 9). The idea is to compute the incident light from the reflection direction $\mathbf{r}$ by marching through the LPV and averaging the intensity $I(-\mathbf{r})$ of all cells, divided by the squared distance to the cell that we pass through; in practice 4 cells proved to be sufficient. This procedure can be interpreted as going back in propagation time (because we sample light that has not been propagated thus far), and look up the intensity distributions before they get smeared out further. The lookup into multiple cells smooths the estimate and prevents artifacts, such as discontinuities, in the glossy reflections.

| Stage | GTX285 | Xbox 360 | PS3 |
|---|---|---|---|
| RSM rendering | 0.16 | 0.5 | 0.8 |
| VPL Injection | 0.05 | 0.2 | 0.4 |
| GV Injection | 0.02 | 0.15 | 0.15 |
| Propagation | 0.8 / 1.1 / 1.4 | 0.8 / 1.1 / 1.5 | 0.7 / 1.1 / 1.5 |
| LPV Lookup | 2.4 | 2.0 | 1.5 |
| Total | 3.4 / 3.7 / 4.0 | 3.5 / 3.8 / 4.2 | 3.4 / 3.8 / 4.2 |

**Table 1:** *Detailed timings for the Crytek Sponza scene (see teaser) in milliseconds for the individual stages (for one $32^3$ LPV grid and 8 iterations). The three timings for the propagation step refer to: no occlusion, fuzzy occlusion, fuzzy occlusion and multiple bounces. Note that only the cost of the RSM rendering depends on the scene complexity. All measurements at 1280x720 resolution (no MSAA), and RSM size of $256^2$ (=number of injected VPLs) for NVIDIA GTX285 and $128^2$ for consoles.*

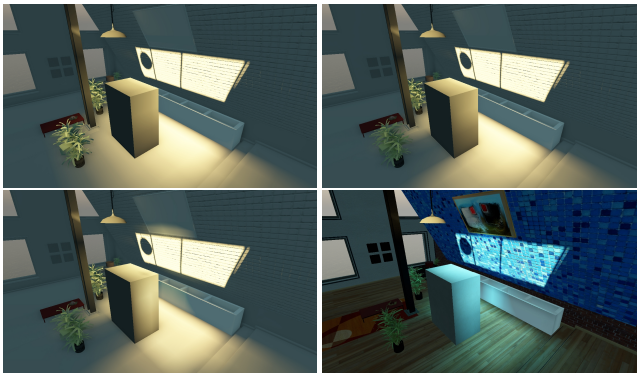

**Figure 7:** *Indirect lighting without and with fuzzy occlusion (top left 130fps, and right 90fps), and multiple bounces (bottom, 85fps) rendered using one LPVs at $32^3$. $256^2$ VPLs from the RSM have been injected into the LPV; $2 \times 256^2$ surfels have been created from the RSM and the camera view (downsampled) for the GV.*

**Participating media** We can also use the LPVs to render plausible effects of single-scattering participating media (see teaser). For this we perform a final render pass and ray march through the cascade of LPVs and accumulate the inscattered light assuming an isotropic phase function. Strictly speaking, adding inscattered light generates energy, and we need to attenuate light during propagation. The step size is proportional to the grids' cell size, i.e. we use larger steps in coarse volumes, and smaller steps close to the camera. In the teaser we show an example rendering with homogeneous media; the rendering cost increased by 18ms for the ray marching with a ray marching step size equal to the LPV grid cell size. Note that we can also plausibly render non-homogeneous media when using a volume texture to store scattering coefficients.

**Discussion** The main advantages of our method are the low cost and that it produces stable and flicker-free renderings. The latter is achieved with the high sampling of the surfaces using RSMs and the VPL injection. An insufficient RSM resolution can cause flickering (see accompanying video), fortunately, creating RSMs and injection make up a small fraction of the rendering cost only. Thus our method can be used to handle fully dynamic scenes including those with "intricate" geometry, such as foliage and fences (Fig. 10). The limitations of our method are obvious as well: first, the spatial discretization might be visible as light bleeding (Fig. 11, left); this is somewhat reduced by the cascaded approach. Second, due to the SH representation of the intensity and the propagation, the light strongly diffuses and is not propagated strictly straight ahead, and there is no reasonable chance of handling glossy surfaces during propagation. The impact of the number of propagation iterations is shown in Fig. 12; the cascaded approach reduces the computation

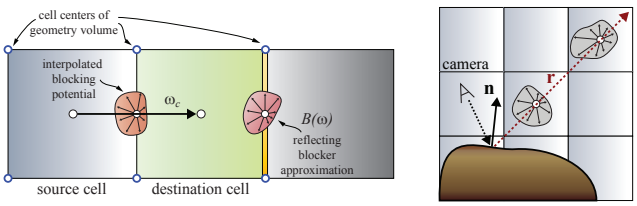


**Figure 8:** *Left: we use the GV to approximate more bounces of indirect illumination. Right: we can light glossy surfaces using ray marching in the LPV.*

**Figure 9:** *We ray march the LPVs along the reflection direction of specular surfaces to render plausible glossy materials (rendered at* $1280 \times 720$ *with 110fps on GTX285 using one LPV at* $32^3$*, no secondary occlusion; the scene consists of 265k triangles).*

cost in large scenes which would require high resolution grids (the cost is linear in the number of iterations and grid cells). Lastly, we have to exercise caution to make sure that geometry causing indirect light or blocking is sampled by RSMs or the camera view to inject the respective data into the grids (Fig. 11, right). Please see the accompanying video for further demonstration of benefits and limitations.

## 8 Conclusion and Future Work

We presented an efficient method for the rendering of plausible indirect lighting in fully dynamic, complex scenes in real-time that uses volumetric representations of the light and geometry in a scene. We demonstrated our method in various scenes in combination with wide-spread real-time rendering techniques. In the future we would like to reduce the limitations of our method, e.g. by following Fattal's [2009] ideas, and by investigating non-rectangular grid structures which might also be a promising research direction.


**Figure 10:** *LPVs render flicker-free indirect illumination (shown exaggerated) from complex geometry such as foliage; rendering at 50fps on a GTX285 (no fuzzy occlusion, one-bounce indirect).*

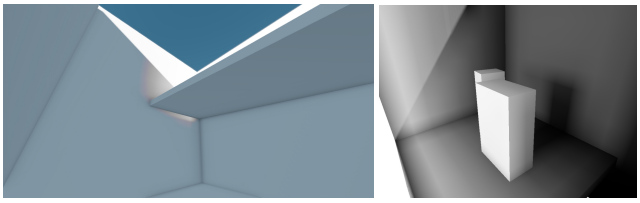

**Figure 11:** *Left: light bleeding from the right wall through the shelf due to low spatial discretization in the LPV and GV. Right: the shadow of the backmost box is missing – it is not captured in the RSM, and only a fraction is visible in the camera image; thus too few surfels are injected into the GV when not using depth peeling.*

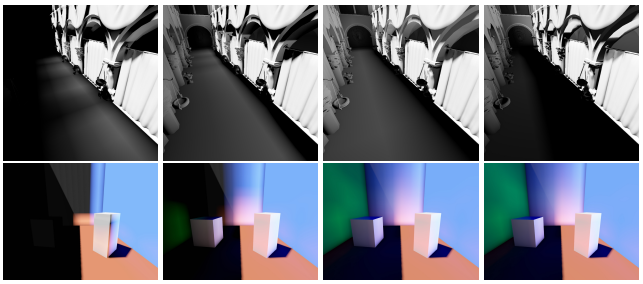

**Figure 12:** *Top row: the approximation of indirect light using a single LPV (30, 60, 80, and 100 meters grid spacing). Note that the cascaded approach is required to capture medium and long distance light transport with few iterations; short distance light transport is approximated with screen-space techniques. Bottom row: using a single LPV (no GV) with 1, 4, 8, and 16 propagation iterations; too few iterations limit the distance that the light travels. Typically, 8 iterations produce visually pleasing results, in particular with cascaded LPVs.*

## References


BAVOIL, L., SAINZ, M., AND DIMITROV, R., 2008. Image-space horizon-based ambient occlusion. ACM SIGGRAPH 2008 talks.

BUNNELL, M. 2005. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, ch. Dynamic Ambient Occlusion and Indirect Lighting, 636–648.

CHANDRASEKHAR, S. 1950. *Radiative Transfer*. Dover Pubn Inc.

DACHSBACHER, C., AND STAMMINGER, M. 2005. Reflective shadow maps. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, 203–213.

DACHSBACHER, C., AND STAMMINGER, M. 2006. Splatting Indirect Illumination. In *Proc. of the Symposium on Interactive 3D Graphics and Games*, 93–100.

DACHSBACHER, C., STAMMINGER, M., DRETTAKIS, G., AND DURAND, F. 2007. Implicit visibility and antiradiance for interactive global illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2007) 26*, 3.

DONG, Z., KAUTZ, J., THEOBALT, C., AND SEIDEL, H.-P. 2007. Interactive Global Illumination Using Implicit Visibility. In *Pacific Graphics*, 77–86.

DUTRÉ, P., BALA, K., AND BEKAERT, P. 2006. *Advanced Global Illumination*. AK Peters.

FATTAL, R. 2009. Participating media illumination using light propagation maps. *ACM Transaction on Graphics 28*, 1, 1–11.

GEIST, R., RASCHE, K., WESTALL, J., AND SCHALKOFF, R. J. 2004. Lattice-boltzmann lighting. In *Rendering Techniques 2004 (Proc. of the Eurographics Symposium on Rendering*, 355–362.

GREGER, G., SHIRLEY, P., HUBBARD, P. M., AND GREENBERG, D. P. 1998. The irradiance volume. *IEEE Computer Graphics Applications 18*, 2, 32–43.

HAŠAN, M., PELLACINI, F., AND BALA, K. 2007. Matrix row-column sampling for the many-light problem. *ACM Trans. Graph. 26*, 3, 26.

IWASAKI, K., DOBASHI, Y., YOSHIMOTO, F., AND NISHITA, T. 2007. Precomputed radiance transfer for dynamic scenes taking into account light interreflection. In *Rendering Techniques 2007 (Proc. of the Eurographics Symposium on Rendering)*, 35–44.

KELLER, A. 1997. Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, 49–56.

KŘIVANEK, J., AND COLBERT, M. 2008. Real-time shading with filtered importance sampling. *Computer Graphics Forum 27*, 4, 1147–1154.

LOSASSO, F., AND HOPPE, H. 2004. Geometry clipmaps: terrain rendering using nested regular grids. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, 769–776.

MCGUIRE, M., AND LUEBKE, D. 2009. Hardware-accelerated global illumination by image space photon mapping. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, 77–89.

MITTRING, M. 2007. Finding Next-Gen: CryEngine 2. In *SIGGRAPH'07: ACM SIGGRAPH 2007 courses*.

NICHOLS, G., AND WYMAN, C. 2009. Multiresolution splatting for indirect illumination. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 83–90.

NICHOLS, G., SHOPF, J., AND WYMAN, C. 2009. Hierarchical image-space radiosity for interactive global illumination. *Computer Graphics Forum 28*, 4, 1141–1149.

OAT, C. 2006. Irradiance volumes for real-time rendering. *ShaderX 5: Advanced Rendering Techniques*.

RAMAMOORTHI, R., AND HANRAHAN, P. 2001. On the relationship between radiance and irradiance: determining the illumination from images of a convex lambertian object. *J. Opt. Soc. Am. A 18*, 10, 2448–2459.

RAMANKUTTY, M. A., AND CROSBIE, A. L. 1997. Modified discrete ordinates solution of radiative transfer in two-dimensional rectangular enclosures. *J. Quantitative Spectroscopy Radiative Transfer 57*, 107–140.

RITSCHEL, T., GROSCH, T., KIM, M. H., SEIDEL, H.-P., DACHSBACHER, C., AND KAUTZ, J. 2008. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 27*, 5.

RITSCHEL, T., ENGELHARDT, T., GROSCH, T., SEIDEL, H.-P., KAUTZ, J., AND DACHSBACHER, C. 2009. Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009) 28*, 5.

RITSCHEL, T., GROSCH, T., AND SEIDEL, H.-P. 2009. Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, 75–82.

SILLION, F. 1995. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. *IEEE Trans. on Visualization and Computer Graphics 1*, 3, 240–254.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2002) 21*, 3, 527–536.

SLOAN, P.-P., GOVINDARAJU, N., NOWROUZEZAHRAI, D., AND SNYDER, J. 2007. Image-based proxy accumulation for real-time soft global illumination. In *Pacific Graphics*, 97–105.

SLOAN, P.-P., 2008. Stupid spherical harmonics tricks. Presentation, Game Developer Conference (GDC2008), San Francisco, CA, http://www.ppsloan.org/publications/StupidSH35.pdf.

TATARCHUK, N., CHEN, H., EVANS, A., KAPLANYAN, A., MOORE, J., JEFFRIES, D., YANG, J., AND ENGEL, W. 2009. Advances in Real-Time Rendering in 3D Graphics and Games. In *SIGGRAPH'09: ACM SIGGRAPH 2009 courses*.

WALTER, B., FERNANDEZ, S., ARBREE, A., BALA, K., DONIKIAN, M., AND GREENBERG, D. P. 2005. Lightcuts: A scalable approach to illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2005) 24*, 3, 1098–1107.

WANG, R., WANG, R., ZHOU, K., PAN, M., AND BAO, H. 2009. An efficient gpu-based approach for interactive global illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2009) 28*, 3, 1–8.

YU, I., COX, A., KIM, M. H., RITSCHEL, T., GROSCH, T., DACHSBACHER, C., AND KAUTZ, J. 2009. Perceptual influence of approximate visibility in indirect illumination. *ACM Transactions on Applied Perception 6*, 4, 1–14.