Light Propagation Volumes in CryEngine 3

Anton Kaplanyan[1]

[1] antonk@crytek.de

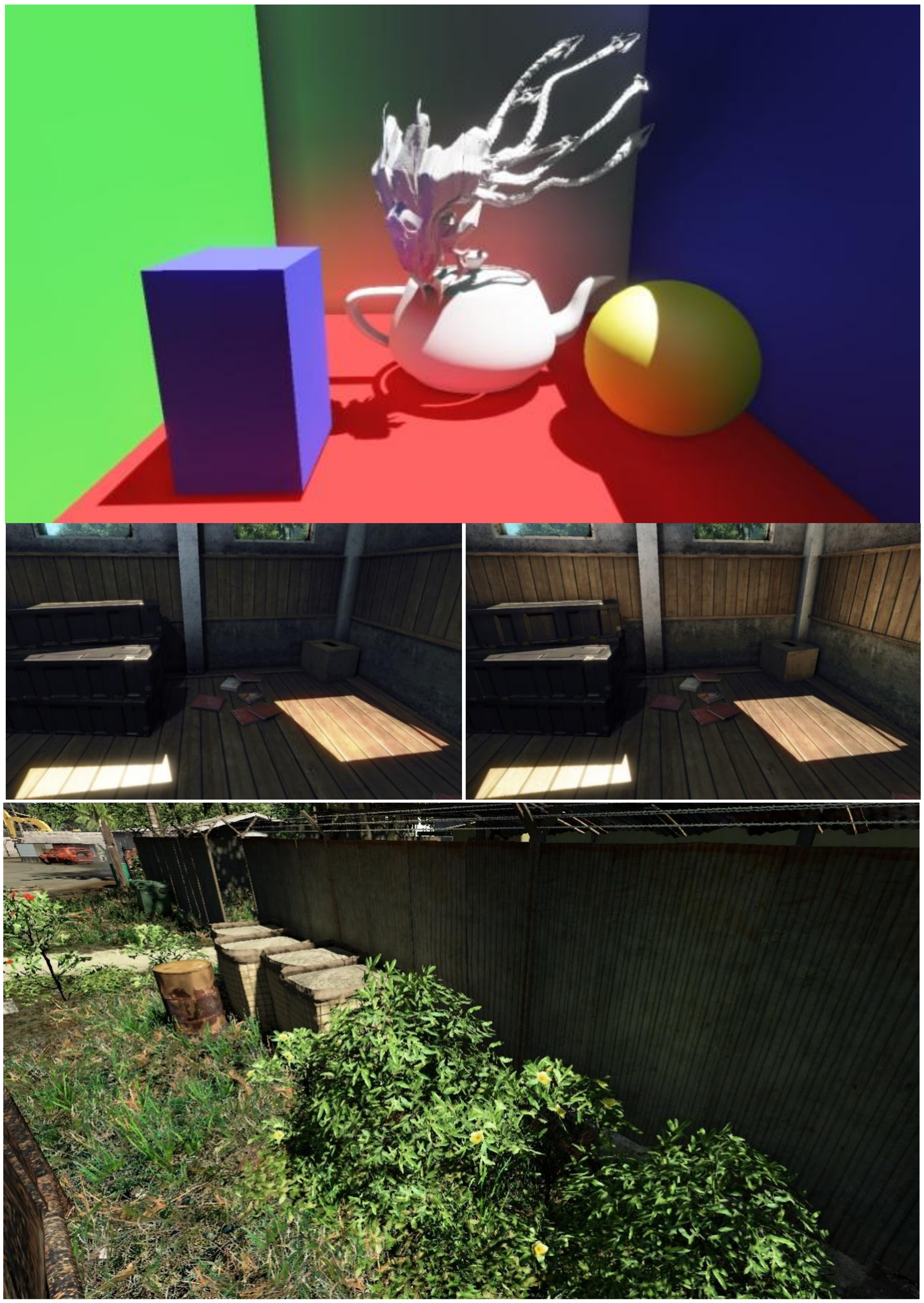


***Figure 1**. Examples of current technique in CryEngine® 3. Top: Cornell box-like environment, middle left: indoor environment without global illumination, middle right indoor environment with global illumination, bottom: outdoor environment with foliage. Note the indirect lighting in shadow areas.*

1 Abstract

This chapter introduces a new technique for approximating the first bounce of diffuse global illumination in real-time. As diffuse global illumination is very computationally intensive, it is usually implemented only as static precomputed solutions thus negatively affecting game production time. In this chapter we present a completely dynamic solution using spherical harmonics (SH) radiance volumes for light field finite-element approximation, point-based injective volumetric rendering and a new iterative radiance propagation approach. Our implementation proves that it is possible to use this solution efficiently even with current generation of console hardware (Microsoft Xbox® 360, Sony PlayStation® 3). Because this technique does not require any preprocessing stages and fully supports dynamic lighting, objects, materials and view points, it is possible to harmoniously integrate it into an engine as complex as the cross-platform engine CryEngine® 3 with a large set of graphics technologies without requiring additional production time. Additional applications and combinations with existing techniques are dicussed in details in this chapter.

2 Introduction

Some details on rendering pipeline of CryEngine 2 and CryEngine 3 could be found in [MITTRING07], [MITTRING09]. However this paper is dedicated to diffuse global illumination solution in the engine. As the Crytek team has already approached the limit of existing real-time direct lighting techniques, we realize the importance and atmospheric influence of indirect illumination to the game scene (see **Figure 1**. Examples of current technique in CryEngine® 3. Top: Cornell box-like environment, middle left: indoor environment without global illumination, middle right indoor environment with global illumination, bottom: outdoor environment with foliage. Note the indirect lighting in shadow areas.). Hard production time constraints and limited hardware were the challenging parts of the indirect lighting research. Since we position ourselves as cross-platform engine, it was decided to allocate 10% of frame as the budget for the global illumination solution, which is around 3.3 ms per frame for 30 frames per second on current generation of consoles (see **Figure 2.**). This budget is achievable even on NVIDIA 7-series and Microsoft Xbox 360 GPUs with current approach.

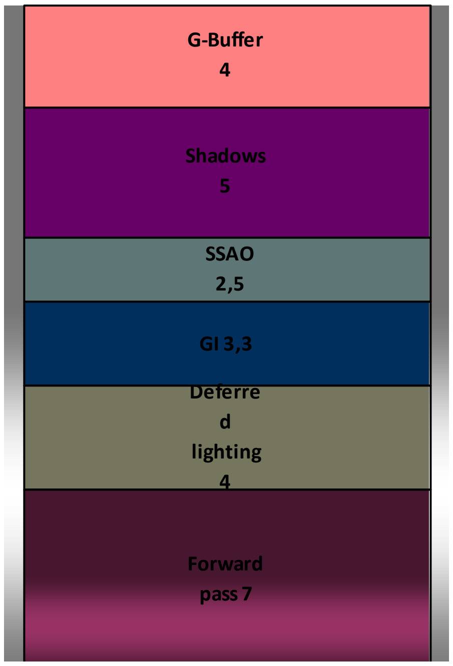


Figure 2. *Average budget (in ms) for rendering one frame of content on current-generation consoles such as Microsoft Xbox 360 or Sony Playstation 3*

As shown in [TABELLIONLAMORLITTE04], it is sufficient to have only one bounce of indirect lighting to introduce the visual veracity even for movie production quality. Thereby, our research also does not take into account multiple bounces because of their unjustified computational complexity.
In this chapter, we introduce a new solution to the single-bounce diffuse global illumination problem for real-time applications. The chapter is structured as follows: first, we partially classify the previous research on this topic. We will describe some of our techniques attempted during the search for the best solution for this problem, outlining the inadequacies of the existing approaches within our constraints, in order to provide a better perspective for our current solution. A brief description of the technology is followed by detailed introduction into the algorithm and existing implementation challenges. We provide solutions to majority of problems and discuss how to combine our method with other lighting techniques. Finally, performance and quality results are analyzed.

3 Previous Work

Nowadays a lot of interactive, real-time and semi-dynamic techniques for diffuse global illumination exist. It should be emphasized that most techniques differs in a way of construction and reconstruction of different kind of global illumination data. These techniques can be partially classified as follows:

Precomputed light transport

The main idea of this class of techniques also known as precomputed radiance transfer (as described in [SKS02], [TATARCHUK04], [CHENLIU08]) is to precompute the light transport at every point in the static scene with arbitrary granularity and store this information with one of approximated forms (e.g. spherical harmonics, spherical wavelets etc.). Obviously the main limitations of this class of methods are the heavy constraints on scene dynamics and increased complexity of game production. This class is one of the most popular and widely used in games because of its relatively good run-time performance, which is provided by the fact that all the computationally heavy parts are done offline during the preprocess stage and a simple relighting can be done in a few shader instructions.

Instant radiosity based methods

This class of approaches (such as in [DACHSBACHERSTAMMINGER05], [DACHSBACHERSTAMMINGER07], [LSKLA07], [RGKSDK08]) is based on the idea of representing indirect lighting as a cloud set of *virtual point light sources* (VPL) [KELLER97]. Consequently, this technique has a great potential to speed up with GPU. Its main advantages are good veracity and absence of any scene/lighting/camera constraints. Unfortunately, the main disadvantage of these methods is inadequate performance primarily because of the necessity to render at least 300-400 shadow-casting VPLs for an arbitrary scene to represent the precise solution without artifacts and flickering. This technique will be discussed in details in the next subchapter.

Photon-mapping based methods

These methods (such as in [STAMATE08] , [SHISHKOVTSOV05], [MCGUIRELUEBKE09])are less popular than the others in real-time graphics because of their performance issues. Usually this class of techniques is based on classical photon-mapping approach [JENSEN00]. These methods usually use GPU texture fetching and rendering units to accelerate the photon map evaluation. The usual optimizations for these techniques are irradiance caching, importance sampling and the incremental approach. One drawback of these methods is that the scene needs to be preprocessed to get the unique representation for the photon map. Another problem is photon map updates caused by scene and lighting changes, which leads to highly inconsistent performance and intermittent stalls.

Geometry approximation methods

This is a novel class of techniques based on the idea of fast superpositioning of light transports of atomic precomputed occluders, usually with discs (such as in [BUNNELL05]), spheres (such as in [SGNS07], [GJW08]) or surfels (such as in [DSDD07], [EVANS06]) being taken as atomic elements. As the processing power of commodity GPUs increases, these methods start becoming more popular. However, approximating an arbitrary scene with these building blocks can present quite a number of practical challenges. Furthermore, it is necessary to have this additional information in the scene representation (e.g. colors, lighting etc.) to take proper indirect lighting into account. The need to generate the additional information can place undue constraints on game production, as well as require complex implementation, and, thus, are rarely used in games at this time.

There are a number of other real-time approaches to diffuse global illumination, for instance, screen-space global illumination [RGS09] (the extension for screen-space ambient occlusion), which in turn takes only screen-space information into account and thus has single view 2D locality as a disadvantage. It will be demonstrated that this technology is a good supplement to our approach, because it allows achieving the first-bounce diffuse global illumination for both small and far geometry casters.

4 The Path to Our Solution

From CryEngine® 2 onwards, we have dynamic lighting, including real-time time of day change support (as described in [WENZEL06]) and a breakable and highly dynamic world. For these reasons we chose to avoid precomputed approaches to the global illumination solution.
A *splatting indirect illumination* approach [DACHSBACHERSTAMMINGER07] was considered as a promising solution for our purposes. The main reasons are: this is the most generalized approach (i.e. it does not have any precomputations or hard limitations), and it is based on a movie production method ("Instant radiosity" [KELLER97]). Moreover, this is the most efficient GPU technique for generation of secondary light sources. However, rendering of huge number of secondary light sources is a challnge for real-time applications.

Several attempts were made by us to solve this problem. Deferred techniques were used to decouple lighting and geometry complexity.
The first idea was to use grouped [WFABDG05] and tile-based [BALESTRA08] deferred rendering of light sources. Unfortunately, this did not provide the required speed-up since these solutions provide only some small bandwidth optimizations.
Afterwards it became clear that there were a huge number of redundant calculations for indirect lighting. Thus there should be good optimization opportunities for a lot of dull light sources with huge coverage.
This led to the idea of half-resolution rendering followed by bilateral upsampling, similar to the trick used in [SGNS07]. However this introduced significant quality degradation in the case of a scene with high-frequency depth changes (e.g. foliage, forest etc.).
The third idea is to use the interleaved deferred light accumulation buffer [SIMP06] to reduce the GPU fill-rate and bandwidth impact. This technique is more accurate than the bilateral upsampling. Despite this giving a good speed-up, there are still some unavoidable artifacts at the edges that comes from the initial interleaved samples.
None of these techniques reduces the complexity by order of magnitude. Thus the rendering performance in all these cases was still unacceptable for modern games with complex scenes and high-definition display resolution.

There are several different solutions proposed addressing this issue, such as [NICHOLSWYMAN09], [NSW09], which still yielded only a partial solution to the performance problem. Moreover, these techniques have almost no benefit in case of arbitrary scene with highly discontinuous geometry, like

foliage, and high depth complexity, which is usual scene for games. Although we did not implement these solutions because of time constraints.
Initially, the number of secondary light sources is $O(n \cdot p)$ to $O(n \cdot \log(p))$ in solutions described above, where n is a number of light sources and p is a number of rendered pixels, and the performance overhead for drawing this many lights is not acceptable for current generation of hardware. Therefore we reached the conclusion that the brute force rendering of a large number of secondary light sources is too heavy for with current generation of hardware. However, if the number of lights could be decreased to $O(n + p)$, then the approach becomes more palatable.

In addition, there is another problem: lack of occlusion information for secondary light sources. It has been recently addressed in [RGKSDK08] within *imperfect shadow maps.* This is a useful idea that allows avoidance of scene rasterization for each secondary VPL. But there is still one significant production complication: additional computation step is necessary to create and manage point-based geometry representation of dynamic scene. This requirement becomes especially nasty in massive, complex scenes with high depth complexity.

It is possible to use imperfect shadow maps for occlusion generation in our technique as well. But this technique was not implemented because of research time constraints. Also we wanted to avoid storing another point-based scene representation because of precomputations and additional memory overhead and assets production complication.

5 Light propagation volumes

Firstly, we define a list of terms used in this chapter. Note that all vector terms are lower-case bold, operators are upper-case bold.

I	transport intensity
G	visibility operator, to be defined below
K	local reflection operator, to be defined below
e_0	emission distribution of primary light sources
$\boldsymbol{c}$	vector of spherical harmonics coefficients
$\boldsymbol{n_s}$	normal to surface
s	area of surface element
t	number of texels or cells
$\boldsymbol{l}$	unit vector towards a light source

L	radiance of indirect illumination
A_s	albedo of point on surface (spectral vector of R, G and B coefficients)

Table 1. *Symbols used in this chapter*

5.1 Overview

We start with considering the Neumann series of the rendering equation for particular point on the surface [KAJIYA86] in operational form with notation from [ATS94]:

$$I = (1 - KG)^{-1} G e_0 = \sum_{n=0}^{\infty} G(KG)^n \, e_0 \qquad (1)$$

where:

I - transport intensity;

G - linear visibility operator[2]:

$$(Gh)(x, w) \equiv h(x^{'}(w, x), w)$$

K – linear local reflection operator:

$$(Kh)(x, w) \equiv \int\limits_{S_2} k(x, w^{'} \to w) h(x, w^{'}) d\mu(w^{'})$$

e_0 - emission distribution of primary light sources.

The dependency on the surface point is omitted for clarity.

The series could be reduced to three summands in the case of finite bouncing limited by the second bounce:

$$I = I_0 + I_1 + I_2 \qquad (2)$$

where:

$I_0 \equiv G_0 e_0$ - light source visibility from viewer position, which usually represents some emissive object;

$I_1 \equiv G_0 K_1 G_1 e_0$ - scene direct lighting term with shadows;

$I_2 \equiv G_0 K_1 G_1 K_2 G_2 e_0$ - the second bounce of lighting (the first bounce of indirect lighting).

Now we consider each term of this equation in details.

The first term $I_0 \equiv G_0 e_0$represents direct visibility of the light source and is commonly solved in real-time graphics by rendering an emissive object representing the light source shape. The term G is usually resolved with regular depth buffering.

The term $I_1 \equiv G_0 K_1 G_1 e_0$ represents direct lighting from all light sources in the scene, where:

- the G_0 term is resolved using a classical depth buffer

[2] See [ATS94] for more details on the notation

- the K_1 operator which is given by the convolution over a hemisphere reduces to some particular function for each simple light source and a simple illumination model with analytical or precomputed or BRDF.
- the G_1 term is usually solved by one of many shadowing approaches. One of the most common techniques is shadow maps, which could be treated as another indirection of depth buffer in turn (it should be noted that shadow mapping technique proved itself as a very efficient solution).

Now we discuss in details the last term $I_2 \equiv G_0 K_1 G_1 K_2 G_2 e_0$, which represents the first bounce of indirect lighting:

- the G_0 term is resolved using classical depth buffering approach as well as in the previous terms
- the last two operators $K_2 G_2$ are efficiently resolved for pure diffuse surfaces with *reflective shadow maps* technique [DACHSBACHERSTAMMINGER05] as can be shown by this equality:

$$I_2 \equiv G_0 K_1 G_1 K_2 G_2 e_0 = G_0 K_1 G_1 I_1^{'}$$

where $I_1^{'} \equiv K_2 G_2 e_0$ - is a direct lighting term without visibility operator from viewer position.
Thus we have a scene representation lit by pure direct lighting as an output of $I_1^{'}$ term.
It should be noted that the term $K_2 G_2 e_0$ in $I_1^{'}$ can be successfully solved within current lighting and shadowing methods for each simple light source.
Thus, it becomes clear that the reflective shadow map technique is a natural and efficient solution for this term within modern GPUs.

The equation (2) can be regrouped in following manner:

$$I = G_0(e_0 + K_1 G_1 (e_0 + K_2 G_2 e_0)) \qquad (3)$$

So, the regrouping in the latter equation shows how the rendering equation could be modified to be more rasterization-friendly. The equation transforms to the more iterative and layered one. That is the way to solve it efficiently in parallel with current rasterization hardware.

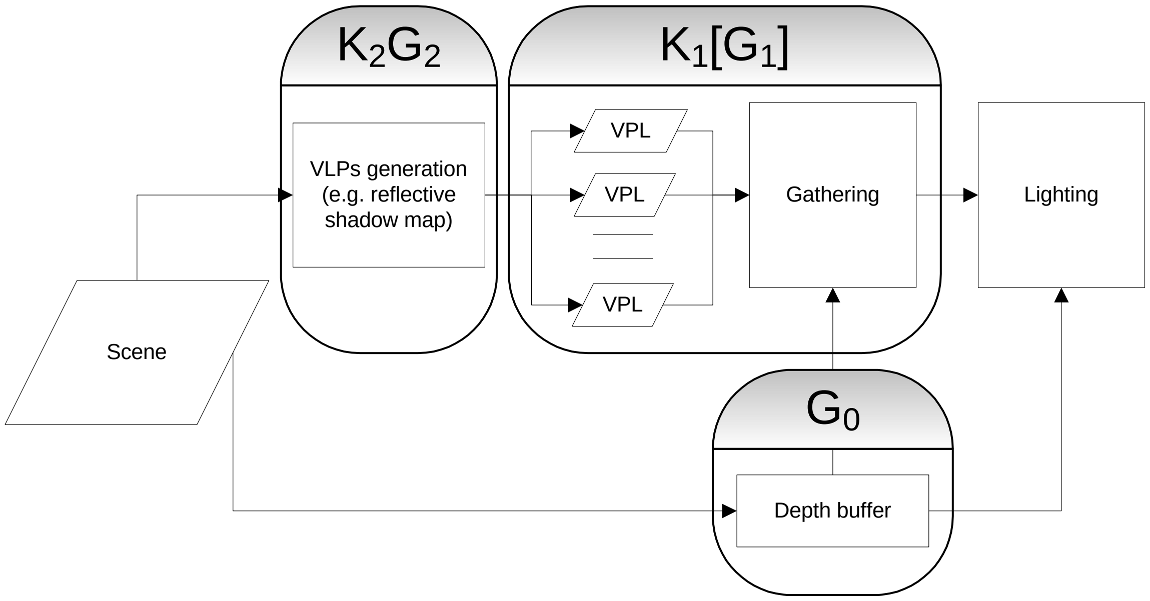
In the case of instant radiosity proposed by [KELLER97] we have a huge set of secondary light sources as an output of $I_1^{'} \equiv K_2 G_2 e_0$ term. The naive solution is to solve the term $K_1 G_1$ for each light source in this set:

$$K_1 G_1 K_2 G_2 e_0 \approx \sum_{i \in \Omega} K_1 [G_1] (K_2 G_2 e_0)_i \qquad (4)$$

where $(K_2 G_2 e_0)_i$ is the *i*-th secondary light source from the set.
As listed before, there are many techniques based on this technique. All of them suffer from a huge performance penalty from rendering such a large number of light sources, because the number of members in this sum is usually several hundred per pixel.
Another challenging part of this operator is occlusion detection, which is hidden in the outer G term. Most of mentioned techniques ignore the secondary occlusion because of the rapidly increasing number

of visibility tests that need to be done for each VPL. However it is still possible to take it into account with imperfect shadow maps [RGKSDK08] with good performance characteristics because of stochastic sampling utilization for visibility queries. But as mentioned before this chapter does not cover the secondary occlusion problem.

The usual pipeline of $I_2 \equiv G_0 K_1 G_1 K_2 G_2 e_0$ term is:



As it could be noticed from this scheme, the term $K_1 G_1$ is usually the most expensive gathering step. It consist of processing generated set of VPLs (such as importance sampling [DACHSBACHERSTAMMINGER07], Lightcuts [WFABDG05]), which is optional; and gathering step, which is usually a deferred shading or deferred lighting pass with some optimizations or hierarchical representations mentioned in sub-section 4.

We reformulate the term $K_1 G_1$ as a scattering operation rather than gathering one. Moreover, we provide a batched scattering for massive amount of light sources, where each particular light source brings a small contribution to the final picture. This is the exact case we have for indirect lighting rendering with VPLs.

Now we briefly describe our algorithm and propose the solution to the last term in the radiance propagation section.

The algorithm consists of four parts as shown at Figure 3:

Generation of radiance point set scene representation

This stage consist of generation of set of secondary light sources by rendering the scene into the reflective shadow map.

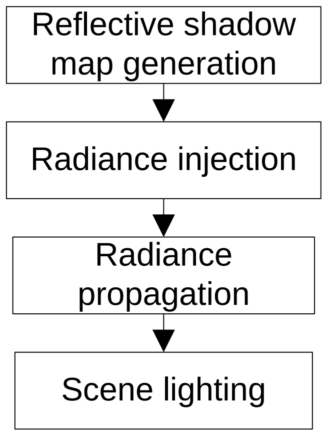


Figure 3. *Algorithm overview*

Injection of point cloud of virtual light sources into radiance volume

Given a point cloud set of virtual light sources from previous stage, inject it into the radiance field which is represented by a volume texture of spherical harmonics coefficients.

Volumetric radiance propagation

Within the initial radiance distribution, propagate radiance by iteratively solving differential scheme inside the volumetric grid. Store the results in the radiance volume.

Scene lighting with final light propagation volume

Apply resulting radiance volume to the scene lighting. Besides the classic way to apply light propagation volumes in the way of SH irradiance volume to scene lighting [TATARCHUK04] [OAT06], there are a lot of other applications to the resulting radiance volumes.
The integration over a hemisphere of the normals with a cosine lobe could be done in the SH basis on the fly in the shader to convert incident radiance to irradiance for diffuse surface (CHENLIU08]).

All these methods are described in detail in the corresponding sub-sections below.

Also it should be noted that this approach is similar to [EVANS06] in general. The main contributions of our approach is higher precision, along with no precomputation requirements and physically based light propagation.

There are still some challenges like spatial locality because of grid approximation and SH low-frequency approximation problems. We discuss all these problems, their importance and solutions in the "Improvements and optimization" sub-section.

5.2 Scene point cloud generation

We use *reflective shadow maps* (RSM) technique [DACHSBACHERSTAMMINGER05] to create a point cloud set of secondary light sources of the surrounding scene objects. The idea of RSM is fairly simple and applicable to our case: it allows sampling of only lit points of the scene with uniform sampling density in an acceptable time. It is one of the most efficient and highly parallel current methods for the GPU to sample the secondary light sources of a scene as mentioned before. Thus the input set of secondary light sources for the equation (4) can be easily generated.

We use the classical RSM layout [DACHSBACHERSTAMMINGER05]. This makes it possible to manipulate the final intensity of a clustered secondary VPLs during subsequent down-sampling pass. Moreover, this also facilitates injection of each VPL into the light propagation volume.

To reduce the number of resulting light sources, we use a smart filter to make an importance down-sampling of the resulting RSM, using an intensity-aware clustering. The metric of this filter is similar to [WFABDG05] and consist in clustering key VPLs by its intensities. This step lightens the follow injection stage. The implementation could be found in Appendix B of this chapter.

5.3 Injection

The idea of this stage is to transform a given point cloud set of VPLs into initial radiance distribution represented in SH coefficients similarly to [CHRISTENSEN08] and inject it into a *light propagation volume*. Light propagation volume (also used as a "*radiance volume"* term in this chapter) is a volumetric texture, which stores *radiance field* approximated by spherical harmonics at each texel. Thus, we have regular finite-element spatial approximation instead of dynamic data structures. We will operate with "*surfels*" from *point-based rendering* (PBR). In terms of PBR, this stage consists of rendering a lot of surfels into a volume texture. Therefore, it becomes not only a SH projection stage, but also creates an initial radiance field in the radiance volume. As the input of the injection stage, we have a set of secondary light sources stored as texels of the downsampled reflective shadow map. The goal of this stage is to inject the existing initial distribution of reflected radiance into the SH radiance volume using point-based rendering [BHZK05] into volume texture. The major problem of PBR is discontinuities between surfels. Since the size of individual texel in the RSM is guaranteed to be much less than the size of one grid cell of the destination volume texture in our case, there is no need to compute the exact size and orientation of each surfel. Instead, we need to take into account the weight of the final contribution. Thus it becomes just an additive accumulation step.

Note that the computations in this sub-chapter are performed only for a directional light like the sun, which is the most important and complex light source for diffuse global illumination problem. The computations for other types of light sources could be easily done in a similar way, but with care about a perspective transformation.

Additionally, an attention should be drawn to the fact that this is the last stage, which needs to be repeated for each additional primary light source in the scene. That means that only the RSM generation stage and the injection stage should be done for each primary light source, but the injection could be done into the same shared radiance volume that could be shared for multiple light sources. This is an important advantage because these first two stages are relatively cheap and could be easily afforded for many primary light sources with current generation of graphics hardware.

We use spherical harmonics coefficients up to the second band (which amounts to four coefficients) to represent the hemispherical secondary light sources in angular space. As it will be shown below, the approximation with the second band spherical harmonics is sufficient to represent indirect lighting in the radiance volume because of its low frequency nature. We convert the radiosity of each surfel into a hemispherical lobe projected to SH basis. Firstly, we evaluate the normal vector represented by its coordinates $\boldsymbol{n} = (x, y, z)$ of VPL into a direction represented by vector of SH coefficients on the fly in the shader. The polynomial form of the SH basis $\boldsymbol{c} = (c_{0,}c_{1,}c_{2,}c_3)$ is [SLOAN08]:

$$c_0 = \frac{1}{2\sqrt{\pi}}$$
$$c_1 = -\frac{\sqrt{3}y}{2\sqrt{\pi}}$$
$$c_2 = \frac{\sqrt{3}z}{2\sqrt{\pi}}$$
$$c_3 = -\frac{\sqrt{3}x}{2\sqrt{\pi}}$$

These coefficients should be renormalized afterwards to form a hemispherical lobe. The normalization coefficients for hemispherical light source are shown in Appendix A. This vector also needs to be scaled by the contribution of the VPL surfel, which are initial direct light source colored intensity $\boldsymbol{I_L}$, albedo color $\boldsymbol{A_S}$ multiplied by intensityI_S, and the weight of the surfel W_S to get the final spectral coefficients of the considered VPL:

$$\begin{pmatrix} \boldsymbol{c_r} \\ \boldsymbol{c_g} \\ \boldsymbol{c_b} \end{pmatrix} = \boldsymbol{c}^{\boldsymbol{T}}(\boldsymbol{I_L A_S} I_S W_s)$$

The intensity I_S of each surfel for diffuse surfaces can be calculated as usual:

$$I_S = (\boldsymbol{n_s}, \boldsymbol{l})_+$$

Term $(\boldsymbol{n_s}, \boldsymbol{l})_+$ means that the dot product $(\boldsymbol{n_s}, \boldsymbol{l})$ is clamped to the range of non-negative values.
Note that term $\boldsymbol{A_S} I_S$ represents the radiosity of the considered surfel.

Also, the RSM texel area and the projected radiance volume cell area should be taken into account. As mentioned before, RSM surfels are guaranteed to be much less than one texel of destination volume texture, thus we do not need to care about discontinuities. However we still need to weight the contribution of each surfel.

Thus we calculate the weight term w_s for each surfel. Firstly, the area of each texel in an isometric RSM can be calculated as follows:

$$s_{splat} = \frac{s_{RSM}}{t} = \frac{s_{area}}{t}$$

Where:

s_{RSM} - is the area covered by reflective shadow map
s_{area} - is the whole area covered by the considered approach (the area of the cut of the radiance volume which is perpendicular to the direction of light source being injected)
t - is the number of texels in the reflective shadow map

Here we choose the coverage of the RSM s_{RSM} completely to match the coverage of the radiance volume s_{area}. That assumption will simplify following calculations.

The area of one cell can be approximated as an area of the cut of the radiance volume. Note that the area of this cut is named as s_{area}. Thus we have:

$$s_{cell} = \frac{s_{area}}{t_{cells}}$$

So we can approximate the weight for each surfel as:

$$w_s = \frac{s_{splat}}{s_{cell}} = \frac{t_{cells}}{t}$$

The resulting weight of each surfel does not depend both on the size of the radiance volume and on the size of the RSM. That is due to the choice of the RSM area which always tightly fits the coverage of the radiance volume.

5.4 Propagation

There are several approaches to solution of the *rendering equation* (1) as described in [KAJIYA86]. Most of them are split into two main groups: *forward solutions* and *backward solutions*. The most common examples are forward and backward ray-tracing. The latter group is the most popular in computer graphics because it is possible to efficiently get partial solution to the rendering equation without redundant computations.
The propagation stage belongs to the group of forward solutions. This method has some analogies to SH solutions of *scattering equation* proposed in [KAJIYAVONHERREN86] and could be deduced from it by assuming the absence of participating media in the scene.
Because of the low-frequent nature of indirect lighting, we can represent an outgoing indirect radiance distribution by a few bands of SH. Also the initial distribution from the injection stage is guaranteed to have at least hemispherical distribution or smoother. In one of his papers Ravi Ramamoorthi says that you need 3 bands for the diffuse convoluted environment map [RAMAMOORTHIHANRAHAN01]. Although we use 2 bands of SH coefficients, which is enough to represent hemispherical cosine lobe outgoing indirect radiance distribution in our case.

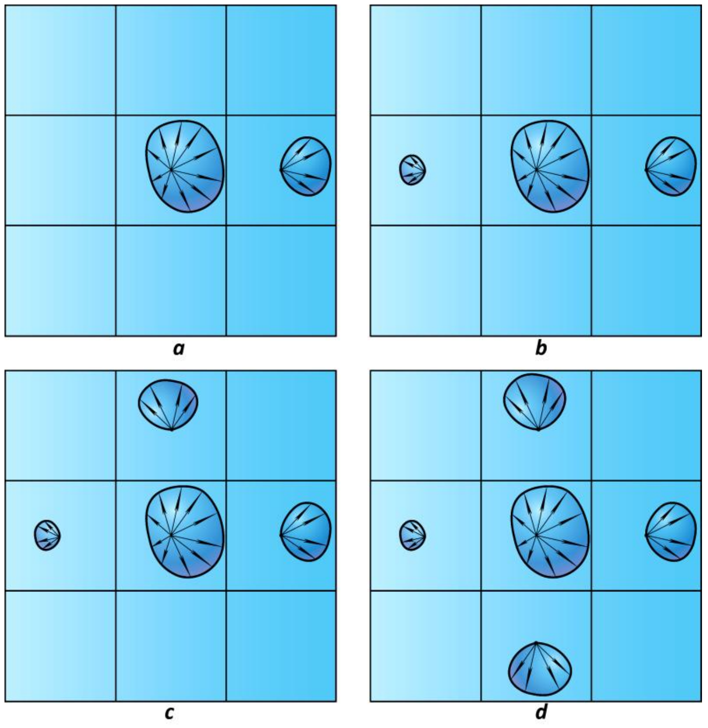


Figure 4. *Radiance propagation iteration*

The propagation stage consists of several sequential iterations. Each iteration represents one discrete step of light propagation in the radiance volume's grid space.

Suppose $L(x)$ is an initial indirect radiance distribution in cell x. During the injection process we add radiance contribution of each VPL into the closest cell regardless of its position inside of this cell. Since the exact distribution of VPLs is unknown inside of this cell after the injection process, we have to treat it as a cube with edge length equal to the distance between cells. The radiance of collected VPLs is uncertainly distributed inside of this cube. Note that we can have an average distribution of the radiance on faces of this cube.

We will propagate the directional energy between adjacent faces to neighboring cubes. In **Figure 4.** Radiance propagation you can see the single propagation iteration for the central cell from adjacent cells. Each propagation step for one cell consists of two stages:

- Compute the flux through each face by computing the integral of the outgoing radiance from the adjacent cube through this face.
- Add the contribution of collected incident energy to the radiance representation of the current cube. We assume that all the energy collected at this face moves only orthogonally to the face. That means that we particularly lose incident radiance direction for this individual face. But the overall directional distribution of energy remains the same after each propagation step for all cubes, since eventually we propagate radiance over the complete solid angle for each source cube and redistribute it onto a closed surface, which represents a wave front approximation.

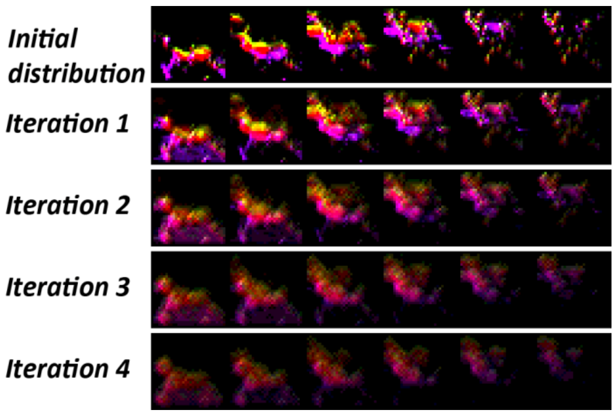


Figure 5. *Example of 6 slices of radiance volume and results of 4 sequential propagation iterations*

Thus, equation for single cell for *n*-th iteration could be described as follows:

$$L^{n+1}(x) = \sum_{i=1}^{6} P^n(L^n(x + d_i)) \qquad (5)$$

Operator P in this equation is a propagation operator. This operator extracts incident radiance from adjacent cell and collects it to the current cell.
Proposed scheme consists of an axial stencil on *Cartesian Cubic* grid, which is widely used for solving scattering equation on a uniform grid.
We use gathering scheme because it is GPU friendly.
Here is a short pseudo code of the gathering propagation algorithm:

```
for_each cell
  for i from directions
    incoming_radiance_dir = get_radiance_over_face(cell.adjacent_cell[i], directions[i])
    cell.radiance += incoming_radiance_dir
```

The example result of radiance propagation algorithm is shown in **Figure 5.** Example of 6 slices of radiance volume and results of 4 sequential propagation iterations. The detailed implementation is provided in Appendix C of this chapter.

5.5 Light propagation volume rendering

This stage consists of lighting the scene with the radiance from the resulting SH volume texture. This can be done by directly fetching radiance distribution from this volume texture using world space pixel position in the regular rendering pass like in any usual forward renderer. However that can be also done by any other scene lighting method, for instance by rendering this texture completely into the deferred diffuse light accumulation buffer as a deferred volume primitive. We use the latter way due to the deferred lighting approach and complex layered ambient lighting model used in CryEngine 3.

Assuming it is usually very expensive pass (see the timing table in the “Results” sub-section), it is very important to optimally utilize hardware during rendering.

It is important to note here that hardware capability to render into a volume texture tremendously improves performance of this technology in general. This dramatically simplifies shader workload since we do not need to emulate a trilinear filtering in the pixel shader. Instead the hardware unit is utilized and cache efficiency is highly improved due to the optimized memory layout of 3D texture. So it is possible to copy the final radiance information to volume texture on consoles or with the Microsoft DirectX® 10 API, but is not possible with the Microsoft DirectX® 9.0c API.
Additionally the main issue of simple trilinear interpolation of an SH representation is undesired light bleeding (e.g. at thin double-sided geometry; see **Figure 8.** Light bleeding through a thin double-sided object caused by sparse spatial approximation, **Figure 9.** Light bleeding through a thin roof caused by the sparse spatial approximation and trilinear filtering of the 3D texture(left). Shifting of radiance injection and anisotropic filtering as a solution (right)). There are several solutions to this problem that will be discussed in the next section.

5.5.1 Deferred lighting

Since we use deferred lighting approach in CryEngine 3 [MITTRING09] the radiance volume is rendered directly into the diffuse light accumulation buffer in addition to multiple ambient passes. This allows us to use all available deferred optimizations like stencil pre-pass and depth bound test for this pass as well. Moreover, it is also possible to compose complex layered lighting, as discussed in sub-section 6.4.

5.6 Other applications

Finally two results could be extracted from radiance propagation and injection passes:

- initial secondary radiance distribution as a result of injection stage
- propagated radiance distribution

Each of them could be used for different lighting and visualization effects.

5.6.1 Light sources

Another interesting application is to inject already propagated radiance. For simple light types we can do this analytically in the shader. Since the hemispherical lobe can be precisely approximated by the second band of SH [SLOAN08], it is enough to represent the analytical solution in this space. Thus, the radiance volume becomes an efficient radiance cache for large dynamic lights. For example, to represent the analytical solution for a point light source, it is sufficient to know the intensity or attenuation radius and represent the radiance in a cell by projecting the direction of the radiance onto SH basis (see Appendix A and **Figure 6.** *Massive rendering of point light sources by injecting analytically pre-propagated radiance into the radiance volume. From left to right top down: rendering with usual deferred lighting, rendering of radiance volume with injected radiance, placement of light sources, error introduced by radiance volume. The error does not exceed 25% for this particular frame and depends on the light radii/grid density*
ratio. There are 417 light sources in this example. The radiance injection and radiance volume rendering takes around 2 ms in total on Xbox 360 and PlayStation 3.). Notice that the error mostly comes from linear interpolation between cells instead of quadratic one (because all light sources here has a quadratic attenuation).

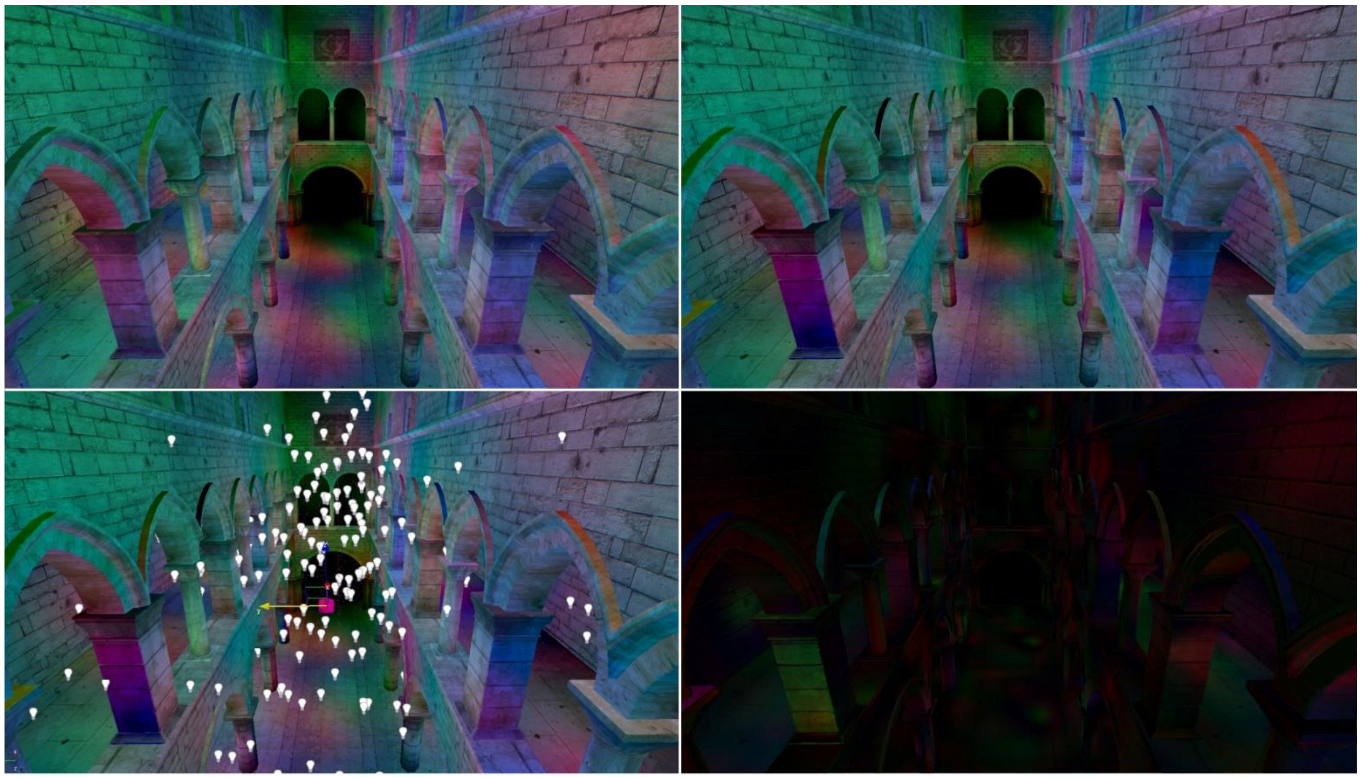


Figure 6. *Massive rendering of point light sources by injecting analytically pre-propagated radiance into the radiance volume. From left to right top down: rendering with usual deferred lighting, rendering of radiance volume with injected radiance, placement of light sources, error introduced by radiance volume. The error does not exceed 25% for this particular frame and depends on the light radii/grid density ratio. There are 417 light sources in this example. The radiance injection and radiance volume rendering takes around 2 ms in total on Xbox 360 and PlayStation 3.*

We use light source radius threshold to make a decision to render the light source into radiance volume or not. This threshold depends on the resolution of the radiance volume.
Specular reflection could be done as described in the next sub-chapter.

5.6.2 Glossy reflections

As we have the initial distribution of secondary radiance after the injection stage, it becomes possible to compute the radiance $L(\mathrm{x}, \mathrm{w})$ at a surface point x towards an individual reflected direction w by solving the sum (5) for $L_{\mathrm{n}}(\mathrm{x}, \mathrm{w})$ for the first few iterations by means of tracing the source volume texture with regular step. Thus, we can fetch several cells towards considered direction and apply integration over a cone with angle, which is the angle of view to current cell from a point x. The glossiness of the resulting specular reflection directly depends on the initial cone angle of the outgoing reflected ray (see **Figure 7.** Glossy reflections on the metallic container from a red teapot and other surrounding.
Left: final picture, right row: corresponding specular lighting buffer.).
In turn, the minimum cone angle directly depends on the spatial resolution of the source radiance volume texture.

Glossy reflections are demonstrated more evidently in accompanying video.

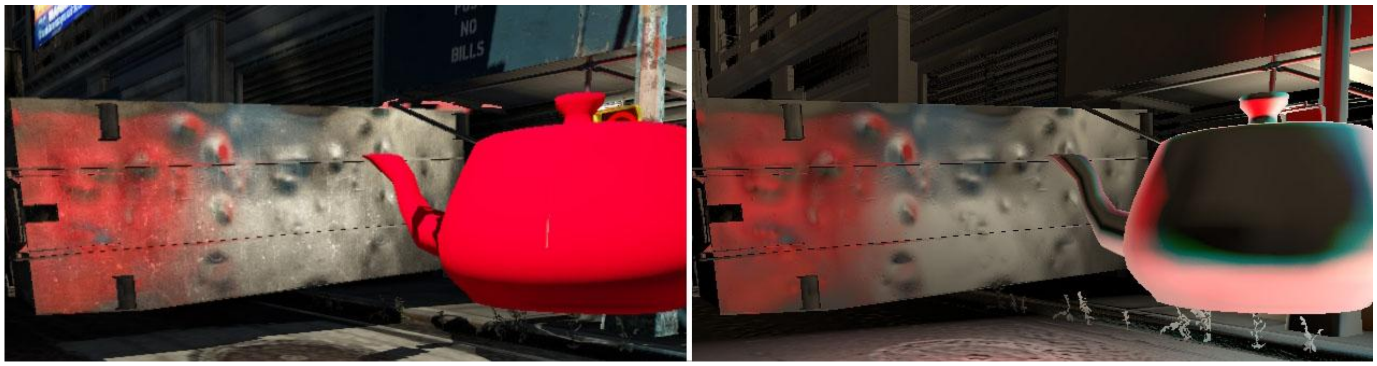


Figure 7. *Glossy reflections on the metallic container from a red teapot and other surrounding.
Left: final picture, right row: corresponding specular lighting buffer.*

6 Improvements and optimizations

Proposed solution still has some undesirable bleeding because of spatial approximation (see **Figure 8.** Light bleeding through a thin double-sided object caused by sparse spatial approximation below as an example).

Additionally temporal flickering occurs during camera movements, because we use camera-attached approach. The temporal flickering mostly occurs because of limited resolution of RSM.
These issues are discussed here in details and solutions are proposed.

Moreover, different combinations and compatibility with other technologies as well as improvements are discussed in more details in this subchapter.

6.1 Solution stabilization

A lot of attention has been paid making the solution stable temporally and on scene changes. This is a very important aspect, owing to the fact that there are a few secondary light sources that are major participants in the final radiance distribution.
Thus, we should provide a stable, consistent image even if we have noticeable camera/direct lighting/scene changes. Multiple methods are employed to obtain this consistency:

- ***2D texel snapping for RSM movement***
  This increases rasterization consistency under camera movement in the case of a directional light source. This approach is well-known for orthogonal shadow maps and provides consistent rasterization during camera movements. Thus it brings a consistent scene representation with surfels independent on RSM movements.
- ***High redundancy of secondary light sources***
  By using excessive number of secondary light source we avoid flickering and other sudden radiance changes during movements of light source, camera and scene objects. That means that number of surfels in RSM should be enough per cell of light propagation volume to provide a stable solution during any changes.
- ***Smart down-sampling of RSM***
  This method not only improves the initial radiance distribution stability, but accelerates the injection stage as well.
- ***One-cell 3D grid snapping for radiance volume movement***
  With discrete integer stepping we can maintain consistent point cloud set injection during camera movement.

Thus, the radiance distribution solution remains smooth and stable in time and space even during significant changes of scene conditions.

6.2 Geometry-aware light injection and shifting

Because of the sparse spatial approximation, it becomes necessary to shift the injection of the radiance contribution of each VPL p to avoid self-illumination (see **Figure** 9**.** Light bleeding through a thin roof caused by the sparse spatial approximation and trilinear filtering of the 3D texture(left). Shifting of radiance injection and anisotropic filtering as a solution (right)). We propose to shift initial radiance towards direction of normal and towards light direction by half cell in sum at maximum. Thus, a minimal error is introduced and the self-illumination is avoided in most cases. For more implementation details see Appendix C. Nevertheless shifting of injecting radiance is not sufficient to completely avoid self-illumination and undesired radiance bleeding.

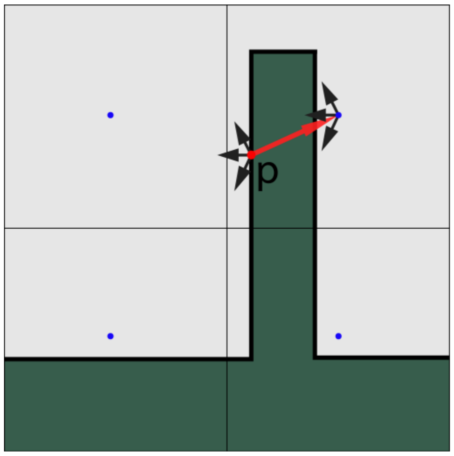


Figure 8. *Light bleeding through a thin double-sided object caused by sparse spatial approximation*

6.3 Anisotropic upsampling of radiance volume

As mentioned previously in sub-section 5.5, SH trilinear interpolation of spatially approximated radiance may cause serious artifacts during the final rendering of radiance volume, such as unwanted self-illumination and bleeding (see **Figure 9.** Light bleeding through a thin roof caused by the sparse spatial approximation and trilinear filtering of the 3D texture(left). Shifting of radiance injection and anisotropic filtering as a solution (right)). There are at least three possible solutions to this problem.

The first one is naive approach which consists of shifting each injected VPL within an offset in such a way that the self-illumination is excluded. Since this offset is usually more than one cell long, it would introduce significant shifting of the whole radiance distribution, thus giving bad results for closely located objects in the scene. Thereby we do not use this approach.

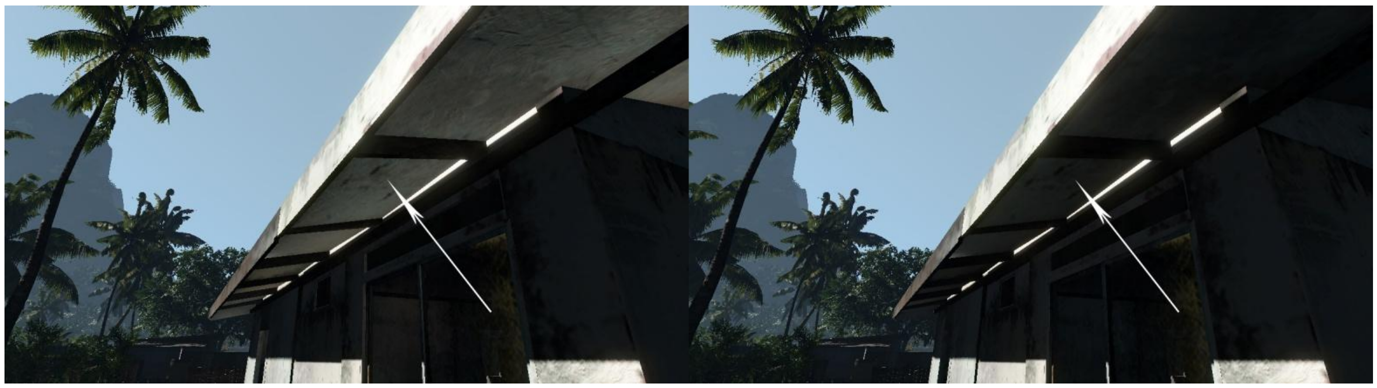


Figure 9. *Light bleeding through a thin roof caused by the sparse spatial approximation and trilinear filtering of the 3D texture(left). Shifting of radiance injection and anisotropic filtering as a solution (right)*

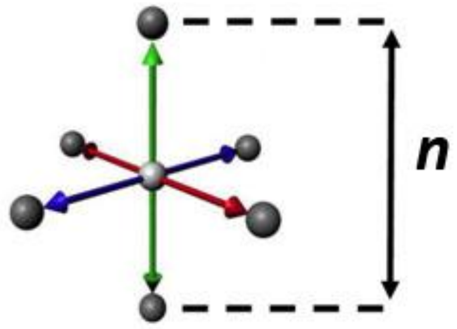


Figure 10. *Spherical harmonics central difference*

The second approach consists of adding so-called "antiradiance" [DSDD07] of the back sides at the injection stage. This makes the radiance distribution have higher "contrast" at places with close opposite-oriented geometry, thus solving the trilinear interpolation softness issue. The main disadvantage of this solution is that we need to render the reflective shadow map twice: firstly, a usual front-facing scene rendering (bounced lighting); and secondly, a back- facing scene rendering for negative lights. This increases the complexity of the technique. Furthermore, the injection stage itself should be repeated for the injection of this negative radiance set after the propagation stage. Also the incorrect darkening at dense locations of opposite-oriented geometry is another unavoidable issue here. Owing to these issues, we do not use this approach either.

Finally, we developed a solution for direction-dependent upsampling of the resulting radiance volume. The main idea of this solution is to detect bad interpolation of the radiance by traversing along the direction of the normal to the surface and calculating the radiance directional derivative in this direction.

We use the central difference scheme for SH coefficients (see **Figure 10.** Spherical harmonics central difference). The central difference towards direction ***n*** is [TATARCHUK04], [AKDS04] to calculate the radiance gradient:

$$\nabla_{\boldsymbol{n}} \boldsymbol{c}(\boldsymbol{x}) = \frac{\boldsymbol{c}(\boldsymbol{x}+\frac{\boldsymbol{n}}{\boldsymbol{2}}) - \boldsymbol{c}(\boldsymbol{x}-\frac{\boldsymbol{n}}{\boldsymbol{2}})}{\|\boldsymbol{n}\|}$$

Thus, by comparing the radiance directional derivative with the actual radiance direction, it can be calculated whether the radiance distribution starts further than its trilinear interpolation for this point.

The results of this method are shown in **Figure 9.** Light bleeding through a thin roof caused by the sparse spatial approximation and trilinear filtering of the 3D texture(left). Shifting of radiance injection and anisotropic filtering as a solution (right).

6.4 Cascaded light propagation volumes (multiresolution approach)

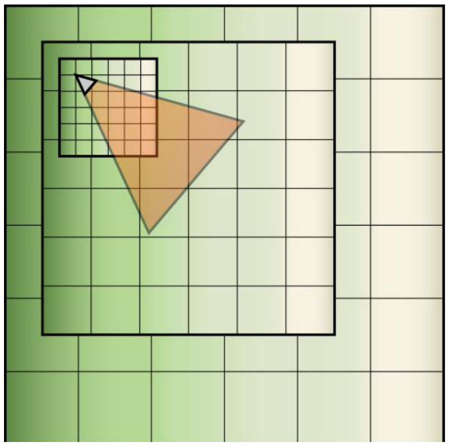


Figure 11. *Cascades of radiance volumes*

The regular grid is not efficient in the case of representing lighting details in different resolutions and for large area coverage because of sparse spatial locality and an $O(n^3)$ memory storage characteristics, where n is a dimensionality of the grid. The density of the spatial approximation mainly depends on the size of some particular object and the distance to the viewer in case of diffuse global illumination.

One solution for this problem could be an adaptive irregular grid, but this solution is somewhat inefficient as well due to the heavy density computation and the quickly growing complexity of the shader program.

Our approach is to use hierarchical cascaded light propagation volumes, which are similar to cascaded shadow maps [STAMMINGERDRETTAKIS02] , [DIMITROV07]. The difference here is that light propagation volumes are nested inside each other, because unlike direct lighting we need to preserve the solution around the camera (see **Figure 11.** Cascades of radiance volumes). Thus our solution allows emulating a discrete multiresolution representation of the indirect radiance of the scene and solving it with a low and predictable performance cost.

During injection in overlapping regions radiance is injected into the appropriate finer grid and vice versa. Light propagation is computed on all grids independently, and during rendering we simply add the contributions from all grids.

Since separate RSM needs to be rendered for each cascade, we have a full control over the objects which could be injected in what cascade. Thereby, each cascade contains radiance from bleeders with strictly defined range of size. Thus it could be treated as a fractional analysis for bleeders over these cascades. The cascaded approach is an efficient hierarchical solution, which is similar to some VPL preprocessing (like Lightcuts or RSM importance sampling).

Thus we have adaptive resolution for objects at different distances (see **Figure 12.** Cascaded approach takes into account small objects near the viewer position. Left: one Light Propagation Volume. Right: three nested cascades of Light Propagation Volumes used.).

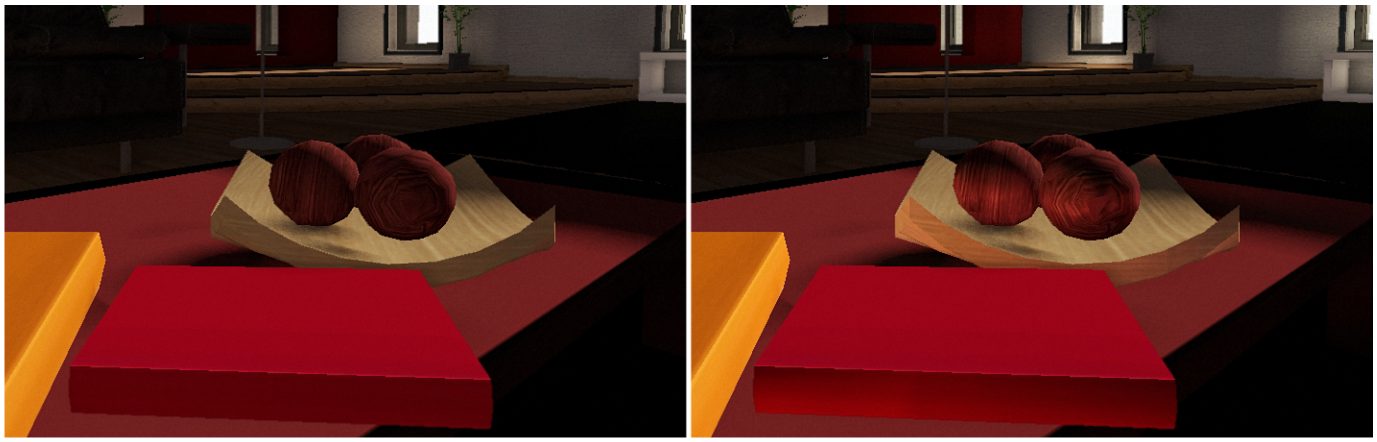


Figure 12. *Cascaded approach takes into account small objects near the viewer position. Left: one Light Propagation Volume. Right: three nested cascades of Light Propagation Volumes used.*

6.5 Combination with SSAO technology

We use the *screen-space ambient occlusion* (SSAO) technique in CryEngine 3 [MITTRING07], [KAJALIN09] for ambient occlusion calculations.

Ambient occlusion is related to accessibility shading, which determines occlusion based on how easy it is for a surface to be touched by arbitrary lighting. Since the indirect lighting mainly consist of some constant or precomputed ambient term and the first bounce contribution from light propagation volumes, it is possible to accurately treat the ambient occlusion term as a secondary occlusion term for this model. It is also possible to improve the SSAO algorithm in this case by taking into account the major radiance direction which could be extracted from the light propagation volume to compute the directional component for secondary occlusion. Unfortunately, this improvement was not implemented because of hard time constrains of the research.

Thus, since the secondary occlusion is not taken into account during the propagation phase, SSAO is used as a multiplicative factor during light propagation volume rendering (see **Figure 13.** Combination of SSAO and our technique. Left to right: only SSAO, only global illumination, combined approach. The bottom line shows indirect lighting buffer.).

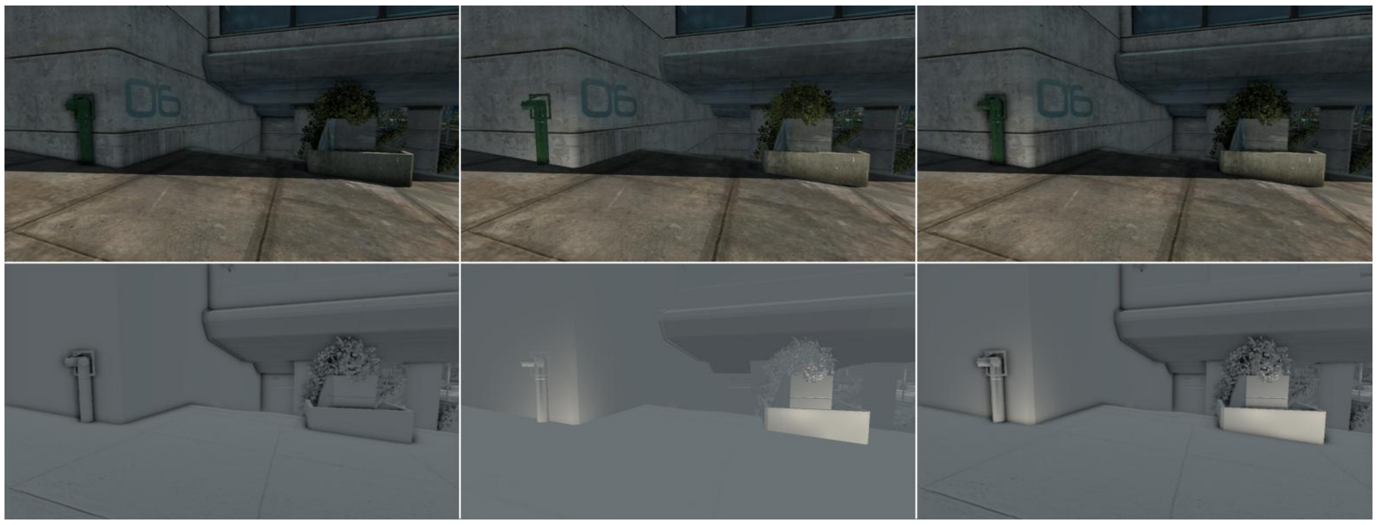


Figure 13. *Combination of SSAO and our technique. Left to right: only SSAO, only global illumination, combined approach. The bottom line shows indirect lighting buffer.*

6.6 Combination with SSGI technology

In contrast with SSAO, the *screen-space global illumination* (SSGI) technique supplements the current solution.

For distant areas that are not covered by light propagation volumes, the SSGI approach needs to be completely enabled to add color bleeding to the rest of the scene.

Also we blend in SSGI for places with micro-bleeding, which are not properly covered by light propagation volumes due to the sparse spatial approximation. Thus it allows us to keep SSGI kernel radius relatively small compared to the pure SSGI approach for such kinds of effects.
Thus the SSGI technique becomes an orthogonal solution for light propagation volumes having good performance characteristics. Moreover the kernel radius could be kept relatively small since we need either local or distant indirect lighting information which is local in screen space in turn.

Since the kernel radius R_{SSGI} for SSGI might generally vary in world space and also might depend on the distance to the viewer, we need to take into account only the color bleeding information that is not counted by the radiance volume. Also we need to take into account that the radiance volume resolution R_{Grid} might change depending on different cascades. Thus we can define the weight for the SSGI factor as:

$$k_{SSGI} = \begin{cases} 1 - \dfrac{R_{SSGI}}{R_{Grid}}, & R_{SSGI} < R_{Grid} \\ 0, & otherwise \end{cases}$$

This term means that SSGI supplements micro bleeding only in places where the granularity of the radiance volume's cells is not sufficient.

Thus, SSGI is an orthogonal supplement to the current technique (see **Figure 14.** Combination with SSGI. From left to right: pure direct lighting with constant ambient term, light propagation volumes without SSGI, light propagation volumes with SSGI. Diffuse lighting accumulation buffer is shown at the bottom respectively.).

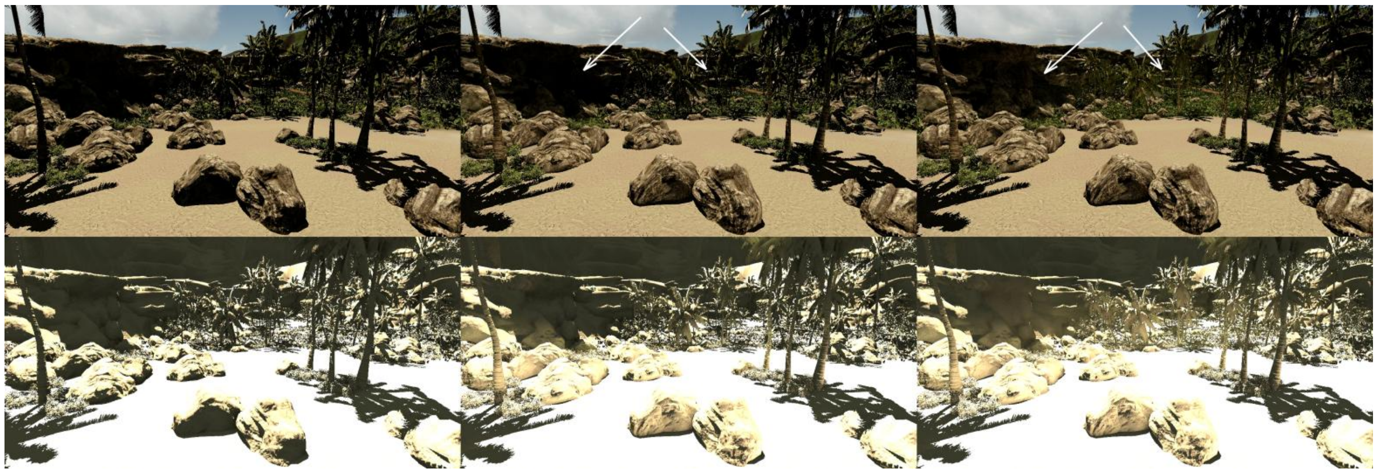


Figure 14. *Combination with SSGI. From left to right: pure direct lighting with constant ambient term, light propagation volumes without SSGI, light propagation volumes with SSGI. Diffuse lighting accumulation buffer is shown at the bottom respectively.*

6.7 Combination with deferred light probes

We use a custom adjustable ambient term in CryEngine 3. More correct and complex lighting could be achieved with deferred light probes without introducing a significant performance impact in areas without direct lighting [MITTRING09]. Complex ambient term is a very important and powerful tool of the real-time indirect lighting.

The idea behind deferred light probes is to locally improve the indirect ambient term by using local cubemaps with preconvoluted irradiance for ambient term computations [ISIDORO05]. Thus, it is possible to have precomputed distant diffuse indirect lighting with a diffuse-preconvoluted cubemap. This

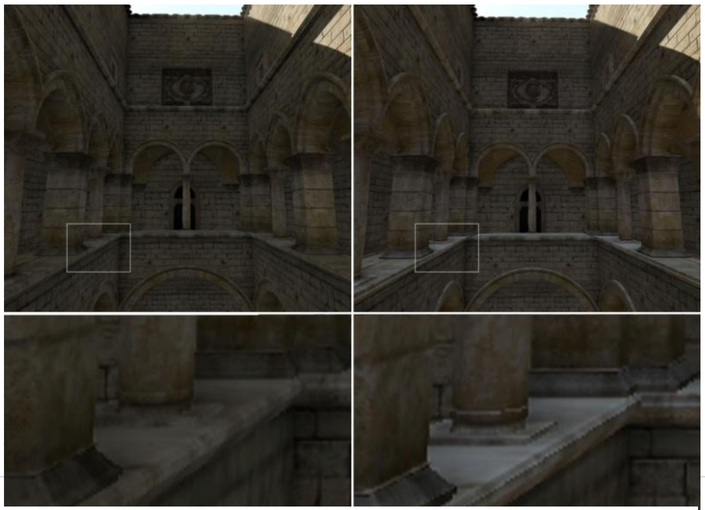


Figure 15. *Left: GI on, deferred light probes off. Right: GI on, deferred light probes on. The contribution of deferred light probe was intentionally increased.*

solution is not designed to bring a fine approximation of diffuse global illumination, because of sparse locations of light probes and quickly increasing memory consumption. However this solution is widely used as an acceptable trade-off for real-time graphics.
It is also possible to naturally combine deferred light probes as a precomputed indirect lighting solution with current technology. By having precomputed distant indirect diffuse irradiance from the cubemap we can add local color bleeding effects from the light propagation volume thus defining the whole solution more precisely.
Thus the current technology could be applied additionally to the convolved diffuse term of the deferred light probe by using GPU blending (see **Figure 15.** Left: GI on, deferred light probes off. Right: GI on, deferred light probes on). Note how deferred light probes could be combined with current technique to improve indirect lighting and approximate multiple bounces. Additionally note a more complex ambient term which introduces the contribution of the bright sky above.

6.8 Consoles optimizations

On both consoles (Xbox 360 and PlayStation 3) we utilize MRT pipeline to render into three render targets, 4 channels each. 3D texture look-ups with trilinear interpolation are utilized for final light propagation volume rendering on both consoles as well. Also the framebuffer blending precision on 8 bit rendertargets turned out to be not sufficient and needs to be improved. So, we implemented that as a simple range scaling in shaders. We use temporary 64-bits per pixel target for RSM injection because of blending precision.

6.8.1 Xbox 360

Despites Xbox 360 API allows to resolve into some particular slice of 3D texture directly from the EDRAM, it has an issue with that related to the offsets calculations for the proper destination memory location. The work-around for this issue could be found in Appendix D.
Additionally we use vertex texture fetching for RSM injection into light propagation volume.
Using 3D texture look-ups during rendering stage is crucial for the GPU performance on this console since it offloads cycles from the GPU math unit.
We use signed ARGB16 format as a target format in EDRAM, however we resolve into signed ARGB8 format after each render pass.

6.8.2 PlayStation 3

We use 32 bits per pixel unsigned ARGB8 render targets for light propagation volumes. All three targets are cleaned with middle grey color, which represents 0. The reason is we need to have signed render targets. The sampled result can be easily shifted and expanded in the shader. During rendering into such render target we use signed additive blending and custom blending with read-back of destination in pixel shader. Both of these modes are supported and have good performance characteristics.

For RSM injection on this platform we use render to vertex buffer. This works well and is implemented as a memory remapping from render target to vertex buffer.
Memory remapping is also used to treat unwrapped 2D texture as a 3D texture with slices. Note that in both cases the texture cannot be tiled or swizzled.

7 Results

We provide timings for this technique for console hardware and show the scalability on PC hardware.
All light propagation volumes in this sub-chapter have the same dimensions of 32^3 texels.
The first table shows timings in milliseconds for Xbox 360 and for PlayStation 3 for one light propagation volume and one 128^2 reflective shadow map.
Sponza scene has 300 draw calls and 1,41 million triangles.
Island scene has 2100 draw calls and 2,32 million triangles.
Apartment has 620 draw calls and 1,14 million triangles.
The scene rendered at resolution 1280x720 without MSAA. Images of the measured scene are provided at **Figure 16.** Results of gloabl illumination on consoles (XBox 360 and PlayStation 3)..

Pass	**Xbox 360, ms**	**PlayStation 3, ms**
RSM generation	0,5	0,8
Injection	0,2	0,4
Propagation	0,8	0,7
LPV rendering	2,0	1,5
Total	**3,5**	**3,4**
Complete frame time	**30,3**	**32,1**

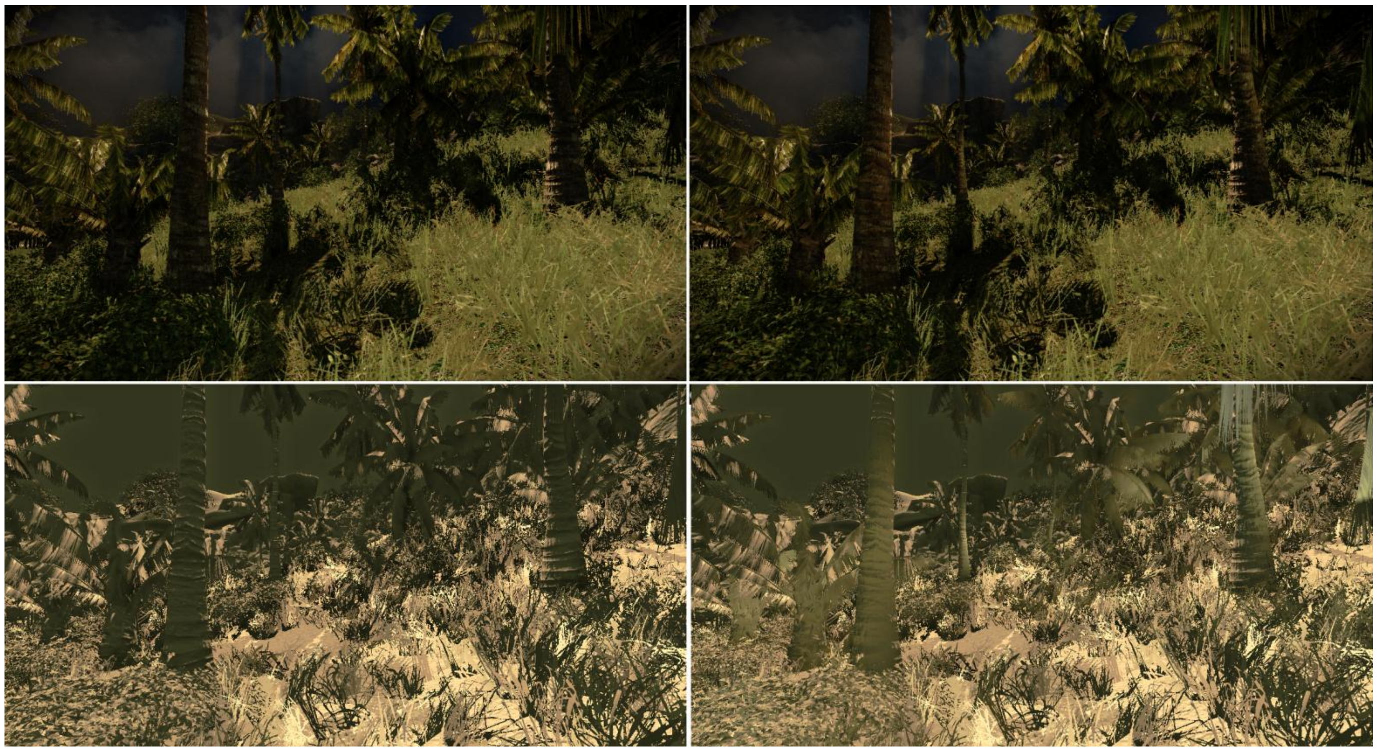


Figure 16. *Results of gloabl illumination on consoles (XBox 360 and PlayStation 3).Left: constant ambient term, right: global illumination enabled. Diffuse light accumulation buffer is on the bottom.*

The injection and propagation stages are pretty similar in performance on both platforms. And the final LPV rendering stage is much faster on PlayStation 3 because we use half-resolution rendering mode with MSAA remapping trick.

The second table provides timings in milliseconds with for different numbers of cascades for hierarchical nested light propagation volumes solution. Note that the time is provided for the Sponza scene.

Number of cascades	**Sponza atrium, ms**
1 cascade	0,72
2 cascades	1,3
4 cascades	2,6
6 cascades	4,0

The hardware used is an NVIDIA GeForce^{TM} GTX 280 GPU and Intel Core 2 Quad CPU @ 2.66 GHz. Testing settings: DirectX 9.0c API, HDR rendering, resolution: 1280x720, MSAA off.
Images are provided at **Figure 17.** Global illumination results for Sponza scene with two cascades and **Figure 18.** Global illumination results for Apartment scene with two cascades @ 120 frames per second..

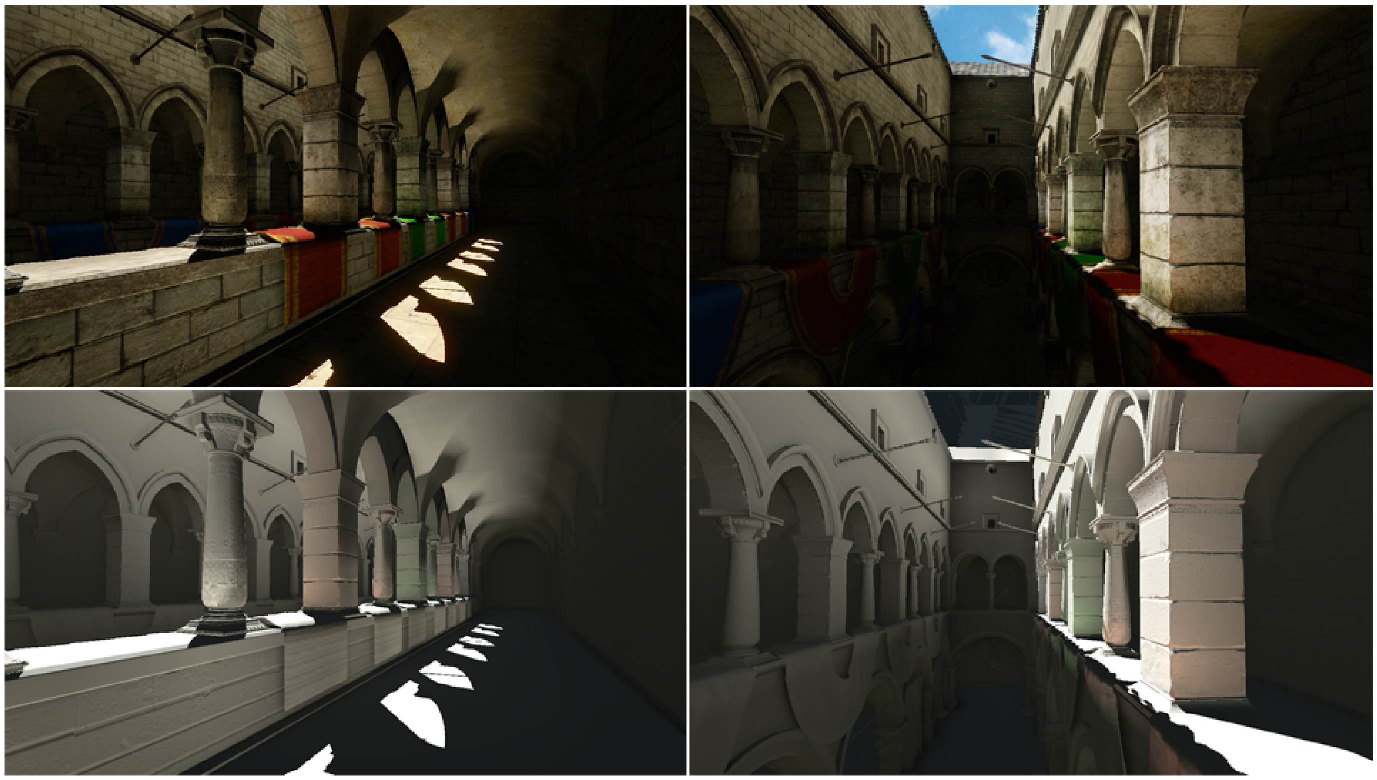


Figure 17. *Global illumination results for Sponza scene with two cascades @ 140 fps.*

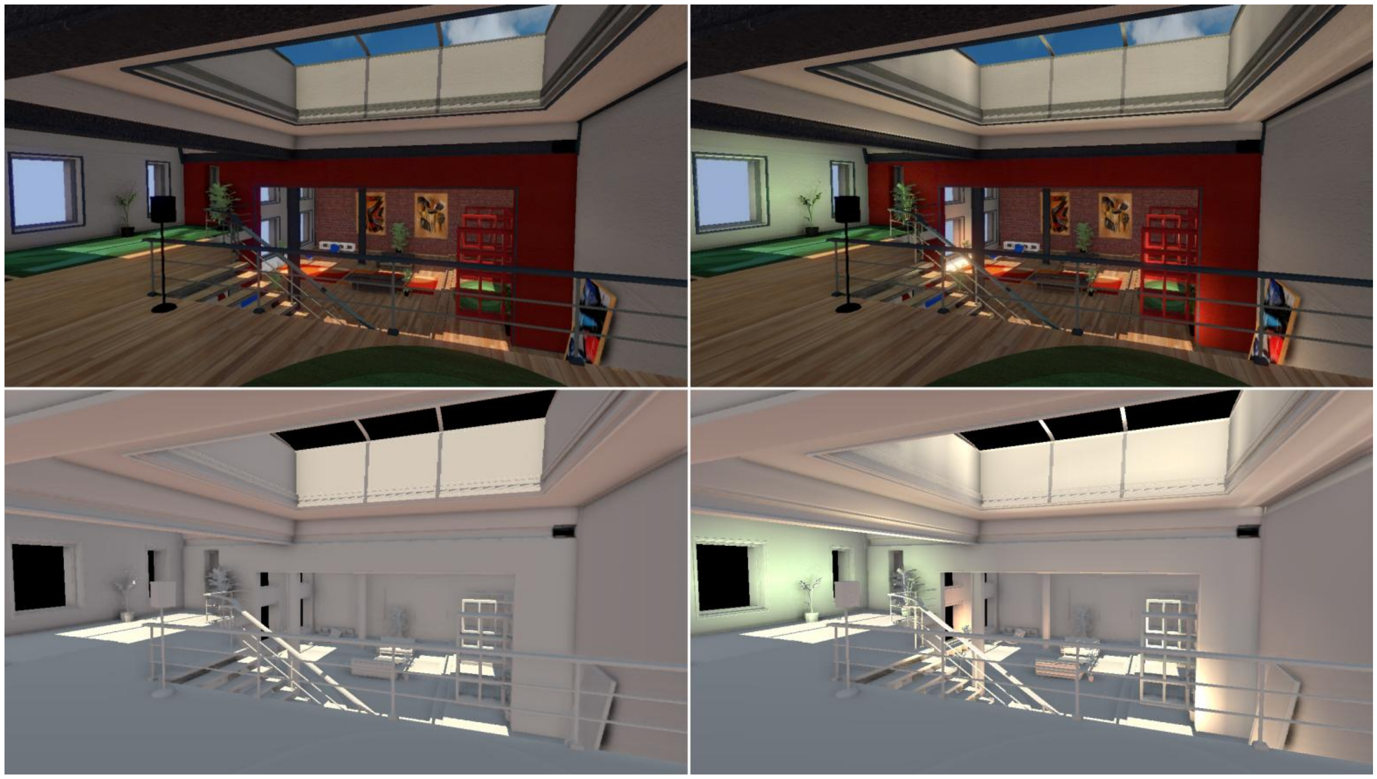


Figure 18. *Global illumination results for Apartment scene with two cascades @ 120 frames per second.*

Additionally, the performance of deferred lighting and our approach is compared for the scene with arbitrary overlapped point light sources:

Number of light sources	Deferred lighting, ms	Light propagation volumes, ms
100	0,62	0,29
200	1,2	0,52
300	1,9	0,65
400	2,5	0,81
500	3,1	0,85
3000	16,5	2,9

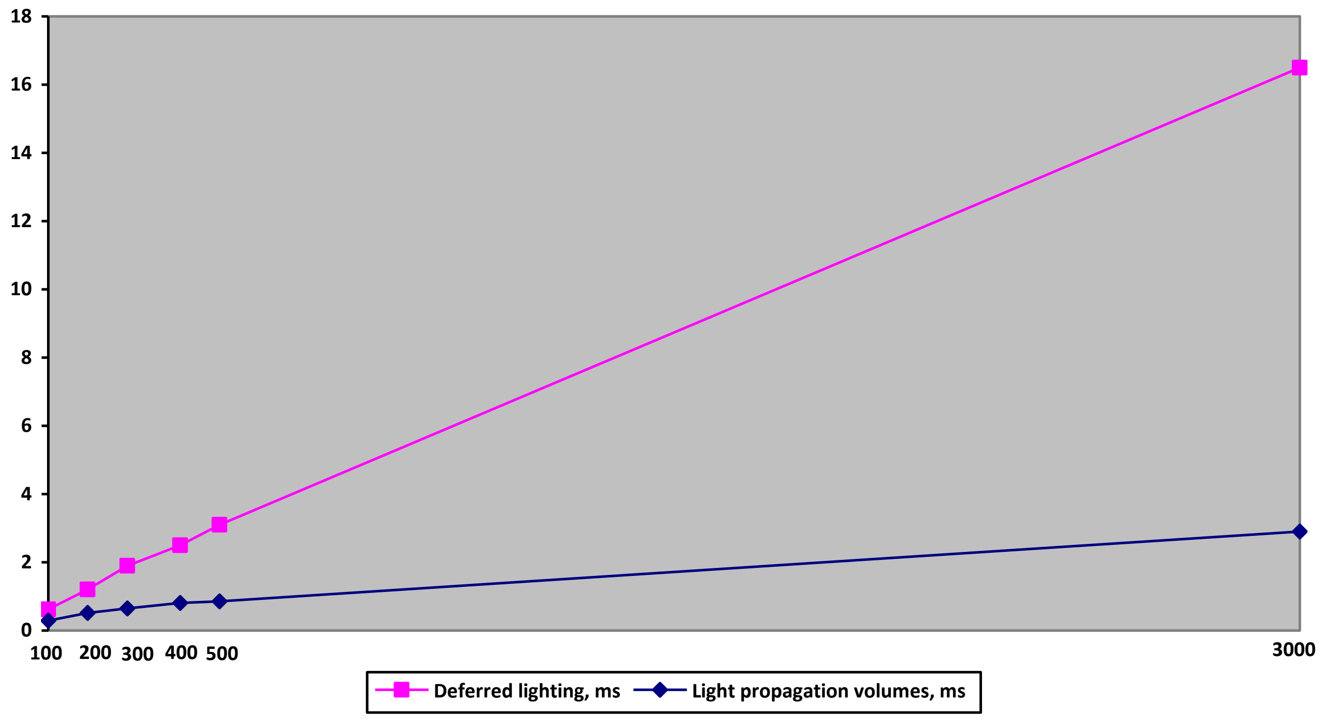


Chart 1. *Performance trends for deferred light sources and for light sources rendered with radiance volume (for 100, 200, 400, 500 and 3000 light sources respectively).*

Light sources are injected into radiance volume without instancing technique, which is a great potential optimization. All light sources have equal radius of 4 meters. Images of results and light overlapping are provided in **Figure 19.** Massive diffuse light sources. Left to right, top down: overdraw for 200 light sources, 200 point light sources, 500 point light sources, 3000 point light sources.. As it could be noticed from chart, the performance cost is drastically lower for huge amount of light sources rendered with radiance volume, which could be explained by decreased bandwidth and lowered by order of magnitude computations necessary for each single light source. Injecting light sources using instancing could also impressively decrease the slope and injection overhead.

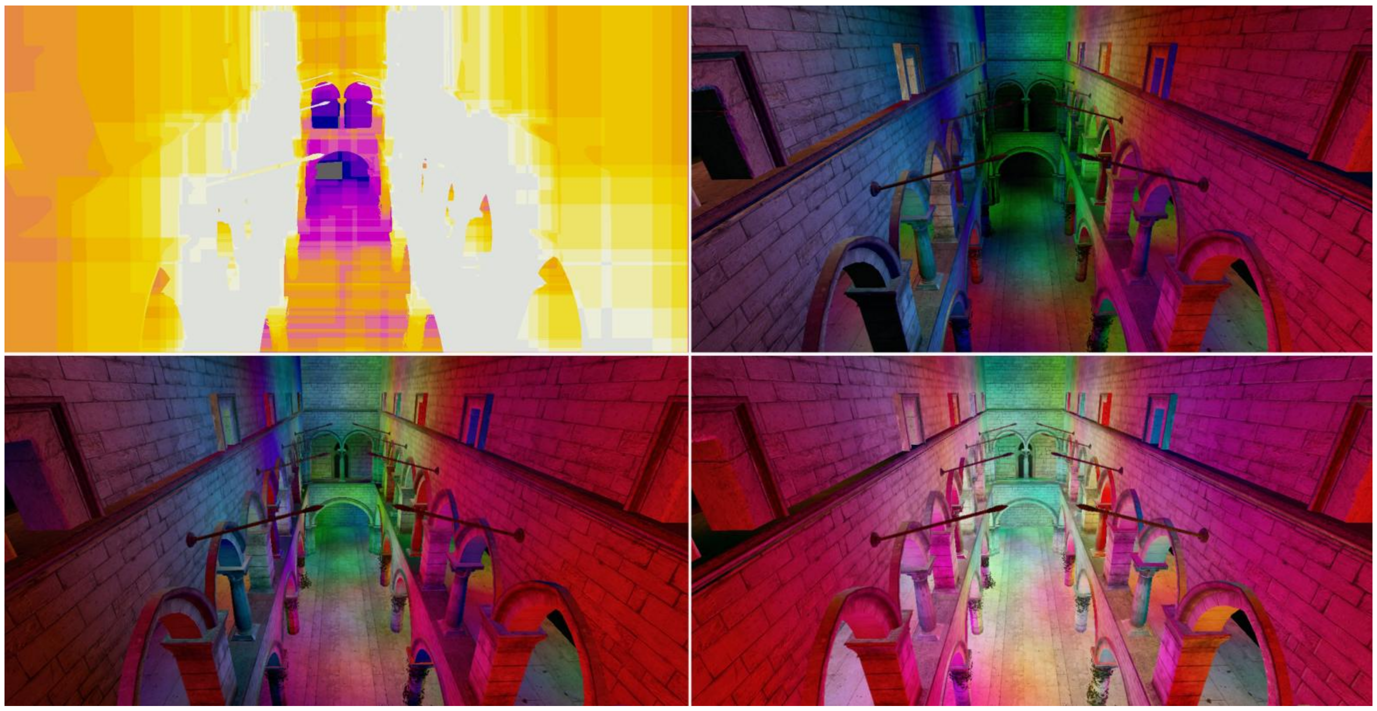


Figure 19. *Massive diffuse light sources. Left to right, top down: overdraw for 200 light sources, 200 point light sources, 500 point light sources, 3000 point light sources.*

8 Discussion and further work

This technique has many good characteristics for games production and since it is partially based on existing technologies, it was possible to implement it in relatively short time.

It should be noted the theoretical proof of this approximation is not provided as well as the approximation error is not measured in this paper as the emphasis is on practical applications rather that theory.

8.1 Participating media lighting

It is possible to approximate a scattered lighting of arbitrary participating media by traversing a radiance volume and collecting a radiance along the ray. The participating media could be defined analytically like "fog volumes" in [WENZEL06] as well as by volumetric representation (e.g. volume texture). This technique is not considered in this chapter because of time constraints, however we have a debug mode in the engine that shows all the radiance in radiance volume as a homogeneous participating media (see

Figure 20. Example of homogenous participating media lighting by sun with propagated radiance by traversing radiance volume texture.).

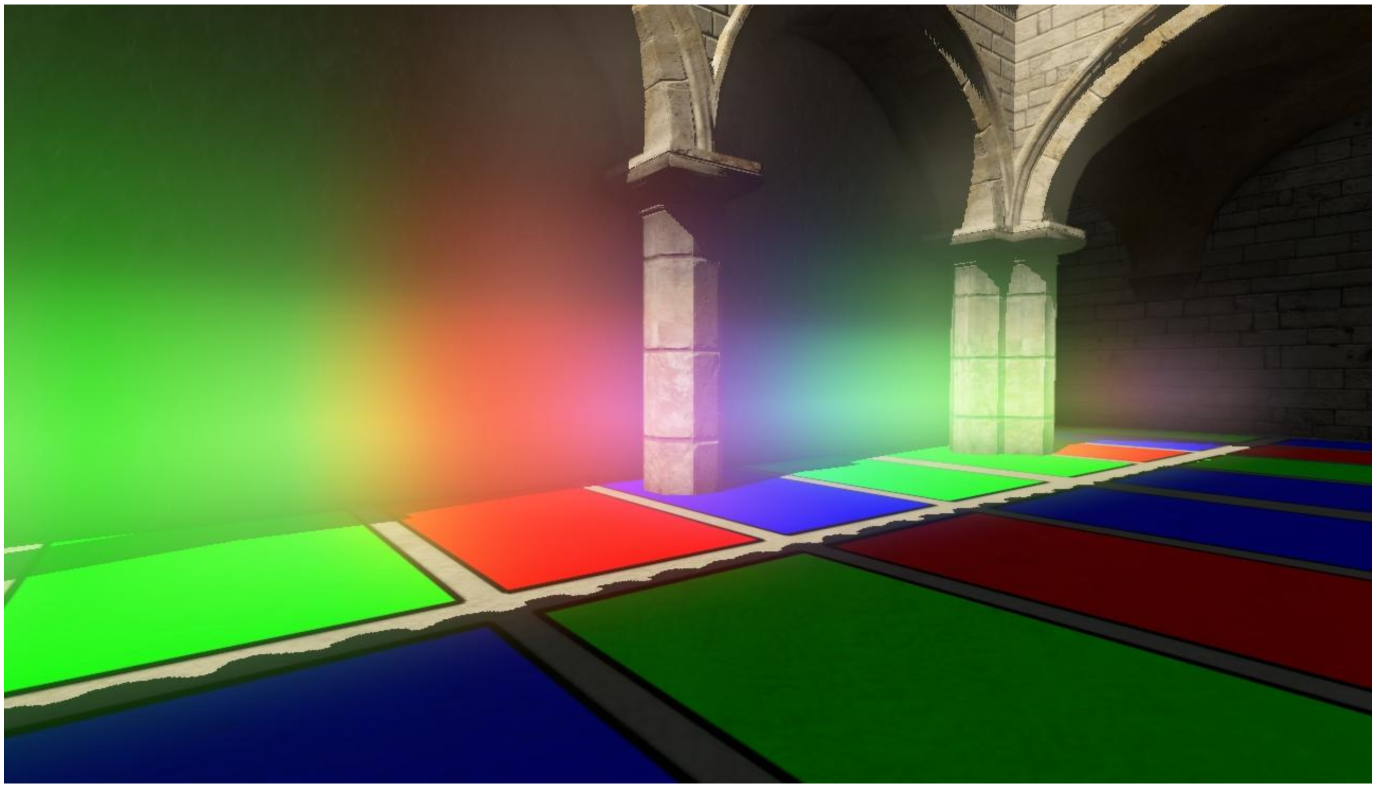


Figure 20. *Example of homogenous participating media lighting by sun with propagated radiance by traversing radiance volume texture. Lighting takes 0,88 ms on nVidia GeForce GTX 280 with 32 samples per pixel from unwrapped volume texture (a 2D texture which emulates 3D texture).*

9 Conclusion

In this paper we present the first highly parallel iterative diffuse global illumination method for real-time applications without precomputations already in use in a commercial game engine. Additionally, it is the first technique that involves a light propagation for diffuse global illumination. Consequently it has a very consistent and scalable performance which is crucial for real-time applications like games. Furthermore we describe how to achieve glossy reflections and use radiance volumes to cache the radiance from analytical light sources to speed up rendering and drastically increase the number of light sources. Also many improvements and combinations with other techniques was proposed.

9.1 Acknowledgments

Thanks to Carsten Dachsbacher for his inestimable and deep discussions and feedback, to Martin Mittring for careful content reviews, to Tiago Sousa for deferred pipeline implementation and good suggestions, to Natalya Tatarchuk for useful hints and course organization, to Mark Atkinson and Evgeny Adamenkov for stylistic review of this chapter, to Ury Zhilinsky for the SSGI implementation. And of course the whole Crytek R&D team as well as all others who helped and discussed the real-time graphics with me!

10 References

1. [AKDS04] ANNEN T., KAUTZ J., DURAND F., SEIDEL H.-P. 2004. Spherical Harmonic Gradients for Mid-Range Illumination. *Eurographics'04, http://www.mpi-inf.mpg.de/~tannen/papers/egsr_04.pdf*
2. [ATS94] ARVO J., TORRANCE K., SMITS B. 1994. A Framework for the Analysis of Error in Global Illumination Algorithms. *Cornell University course, http://eprints.kfupm.edu.sa/18022/1/18022.pdf*
3. [BALESTRA08] BALESTRA, C. 2008. The Technology of Uncharted Drake's Fortune. *GDC'08, http://www.naughtydog.com/corporate/press/GDC%202008/UnchartedTechGDC2008.pdf*
4. [BHZK05] BOTSCH, M., HORNUNG, A., ZWICKER, M., KOBBELT, L. 2005. High-Quality Surface Splatting on Today's GPUs. *EuroGraphics Symposium on Point-based graphics, 2005, http://graphics.ucsd.edu/~matthias/Papers/HighQualitySplattingOnGPUs.pdf*
5. [BUNNELL05] BUNNELL, M. 2005. Dynamic Ambient Occlusion and Indirect Lighting. *GPU Gems 2, http://http.download.nvidia.com/developer/GPU_Gems_2/GPU_Gems2_ch14.pdf*
6. [CHENLIU08] CHEN, H., LIU, X. 2008. Lighting and Material in Halo 3. *ACM SIGGRAPH 2008: Proceedings of the conference course notes, Advances in Real-Time Rendering in 3D Graphics and Games, Chapter 1, pp. 1-22, Los Angeles, CA, August 2008. http://ati.amd.com/developer/SIGGRAPH08/Chapter01-Chen-Lighting_and_Material_of_Halo3.pdf*

7. [CHRISTENSEN08] CHRISTENSEN, P. 2008. Point-Based Approximate Color Bleeding. *Pixar Technical Memo #08-01,* *http://graphics.pixar.com/library/PointBasedColorBleeding/paper.pdf*

8. [DACHSBACHERSTAMMINGER05] DACHSBACHER, C., STAMMINGER, M. 2005. Reflective Shadow Maps. *University of Erlangen-Nuremberg, http://www.vis.uni-stuttgart.de/~dachsbcn/download/rsm.pdf*

9. [DACHSBACHERSTAMMINGER07] DACHSBACHER, C., STAMMINGER, M. 2007. Splatting Indirect Illumination. *University of Erlangen-Nuremberg, http://www.vis.uni-stuttgart.de/~dachsbcn/download/sii.pdf*

10. [DIMITROV07] DIMITROV R. 2007. Cascaded Shadow Maps. *NVIDIA Corporation'07, http://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf*

11. [DSDD07] DACHSBACHER, C., STAMMINGER M., DRETTAKIS G., DURAND F. 2007. Implicit Visibility and Antiradiance for Interactive Global Illumination. *SIGGraph '07, http://www-sop.inria.fr/reves/Basilic/2007/DSDD07/ImplicitVisibilityAndAntiradiance.pdf*

12. [EVANS06] EVANS, A. 2006. Fast Approximations for Global Illumination on Dynamic Scenes. *Course 26: Advanced Real-Time Rendering in 3D Graphics and Games. SIGGraph*, Boston, MA. August 2006, *http://ati.amd.com/developer/techreports/2006/SIGGRAPH2006/Course_26_SIGGRAPH_2006.pdf*

13. [GJW08] GUERRERO, P., JESCHKE, S., WIMMER, M. 2008. Real-time Indirect Illumination and Soft Shadows in Dynamic Scenes Using Spherical Lights. *Computer Graphics Forum'08, http://www.cg.tuwien.ac.at/research/publications/2008/guerrero-2008-sli/guerrero-2008-sli-paper.pdf*

14. [GSHG97] GREGER, G., SHIRLEY, P., HUBBARD, P., GREENBERG, D. 1997. The Irradiance Volume. *Cornell University, http://www.gene.greger-weltin.org/professional/publications/thesis.pdf*

15. [ISIDORO05] ISIDORO J. 2005. Filtering Cubemaps: Angular Extent Filtering and Edge Seam Fixup Methods. *SIGGraph'05, http://ati.amd.com/developer/SIGGRAPH05/Isidoro-CubeMapFiltering.pdf*

16. [JENSEN00] JENSEN, H. 2000. A Practical Guide to Global Illumination using Photon Maps. *SIGGraph '00, http://graphics.stanford.edu/courses/cs348b-01/course8.pdf*

17. [KAJALIN09] KAJALIN, V. 2009. Screen-space ambient occlusion. *Shader X7, Wolfgang Engel, Ed., Charles River Media*

18. [KAJIYA86] KAJIYA J.T. 1986. The rendering equation. *SIGGraph'86, http://portal.acm.org/citation.cfm?id=15902*

19. [KELLER97] KELLER, A. 1997. Instant radiosity. *In Proceedings of SIGGraph 97, Computer Graphics Proceedings, Annual Conference Series, 49–56, http://portal.acm.org/citation.cfm?id=258769*

20. [KAJIYAVONHERREN86] KAJIYA J., VON HERREN B. 1986. Ray tracing volume densities. *Computer Graphics Volume 18, Number 3 July 1984, http://portal.acm.org/citation.cfm?id=808594*

21. [LSKLA07] LAINE, S., SARANSAARI, H., KONTKANEN, J., LEHTINEN, J., AILA, T. 2007. Incremental Instant Radiosity for Real-Time Indirect Illumination. *Eurographics'07, http://www.tml.tkk.fi/~samuli/publications/laine2007egsr_paper.pdf*

22. [MCGUIRELUEBKE09] MCGUIRE, M., LUEBKE, D. 2009. Hardware-Accelerated Global Illumination by Image Space Photon Mapping. *Eurographics'09, http://graphics.cs.williams.edu/papers/PhotonHPG09/ISPM-HPG09.pdf*

23. [MITTRING07] MITTRING, M. 2007. Finding Next Gen – CryEngine 2. ACM SIGGRAPH 2007: Proceedings of the conference on SIGGRAPH 2007 course notes, course 28, Advanced Real-Time Rendering in 3D Graphics and Games, pp. 97-121. 2007, http://portal.acm.org/citation.cfm?id=1281671

24. [MITTRING09] MITTRING, M. 2009. A bit more Deferred – CryEngine3. *Triangle Game Conference'09,*

http://www.crytek.com/fileadmin/user_upload/inside/presentations/2009/A_bit_more_deferred_-_CryEngine3.ppt

25. [NICHOLSWYMAN09] NICHOLS, G., WYMAN, C. 2009. Multiresolution Splatting for Indirect Illumination. *I3D'09,* *http://www.cs.uiowa.edu/~cwyman/publications/files/techreports/UICS-TR-08-04.pdf*
26. [NSW09] NICHOLS, G., SHOPF, J., WYMAN, C. 2009. Hierarchical Image-Space Radiosity for Interactive Global Illumination. *Eurographics Symposium on Rendering'09,* *http://www.cs.uiowa.edu/~cwyman/publications/files/imgSpRadiosity/egsr09_imgSpRadiosity.small.pdf*
27. [OAT06] OAT, C., 2006 Irradiance Volumes for Real-Time Rendering, *ShaderX 5: Advanced Rendering Techniques, Wolfgang Engel, Ed., Charles River Media.*
28. [RAMAMOORTHIHANRAHAN01] RAMAMOORTHI R., HANRAHAN P. 2001. An Efficient Representation for Irradiance Environment Maps. *SIGGraph'01,* *http://www.cs.berkeley.edu/~ravir/papers/envmap/envmap.pdf*
29. [RGKSDK08] RITSCHEL T., GROSCH T., KIM M., SEIDEL H.-P., DACHSBACHER C., KAUTZ J. 2008. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *SIGGraph Asia'08,* *http://www.uni-koblenz.de/~ritschel/dokumente/ISM.pdf*
30. [RGS09] RITSCHEL, T., GROSCH, T., SEIDEL, H.-P. 2009. Approximating Dynamic Global Illumination in Image Space. *Proceedings of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D) 2009,* *http://www.mpi-inf.mpg.de/~ritschel/Papers/SSDO.pdf*
31. [SGNS07] SLOAN, P.-P., GOVINDARAJU, N., NOWROUZEZAHRAI, D., SNYDER, J. 2007. Image-Based Proxy Accumulation for Real-Time Soft Global Illumination. *Pacific Graphics 2007,October,* *http://www.ppsloan.org/publications/ProxyPG.pdf*
32. [SHISHKOVTSOV05] SHISHKOVTSOV, O. 2005. Deferred shading in S.T.A.L.K.E.R. *GPU Gems 3, Chapter 9.* *http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter09.html*

33. [SIMP06] SEGOVIA, B., IEHL, J. C., MITANCHEY, R., PEROCHE, B. 2006. Non-interleaved Deferred Shading of Interleaved Sample Patterns. *Graphics Hardware'06,* *http://liris.cnrs.fr/Documents/Liris-2476.pdf*

34. [SKS02] SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments. *ACM Trans. Graph. 21*, 3, 527–536, http://research.microsoft.com/en-us/um/people/johnsny/papers/prt.pdf

35. [SLOAN08] SLOAN, P.-P. 2008. Stupid Spherical Harmonics (SH) Tricks. *GDC'08,* *http://www.ppsloan.org/publications/StupidSH36.pdf*

36. [STAMATE08] STAMATE, V. 2008. Real-time photon mapping approximation on the GPU. *Shader X6, Wolfgang Engel, Ed., Charles River Media*

37. [STAMMINGERDRETTAKIS02] STAMMINGER, M., DRETTAKIS, G. 2008. Perspective shadow maps. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques (Siggraph 2002), July 23-26, 2002, San Antonio, Texas,* *http://portal.acm.org/citation.cfm?id=566616*

38. [TABELLIONLAMORLITTE04] TABELLION, E., LAMORLITTE, A. 2004. An Approximate Global Illumination System for Computer Generated Films. *SIGGraph'04,* *http://www.tabellion.org/et/paper/siggraph_2004_gi_for_films.pdf*

39. [TATARCHUK04] TATARCHUK, N. 2004. Irradiance Volumes for Games. *ATI Research, Inc,* *http://ati.amd.com/developer/gdc/Tatarchuk_Irradiance_Volumes.pdf*

40. [WENZEL06] WENZEL, C. 2006. Real-time atmospheric effects in Games. *Course 26: Advanced Real-Time Rendering in 3D Graphics and Games. SIGGraph*, Boston, MA. August 2006, *http://ati.amd.com/developer/siggraph06/Wenzel-Real-time_Atmospheric_Effects_in_Games.pdf*

41. [WFABDG05] WALTER, B., FERNANDEZ, S., ARBREE, A., BALDA, K., DONKIKIAN, M., GREENBERG, D. 2005. Lightcuts: A Scalable Approach to Illumination. *SIGGraph'05,* *http://www.cs.cornell.edu/~kb/projects/lightcuts/lightcuts.pdf*

11 Appendix A. Spherical harmonics projections

11.1 Arbitrary Rotation of function with circularly symmetry around Z

Here is a listing of method that takes a direction and zonal harmonics coefficients as an input and returns SH coefficients of this function rotated towards given direction:

```
half4 SHRotate(const in half3 vcDir, const in half2 vZHCoeffs)
{
  // compute sine and cosine of thetta angle
  // beware of singularity when both x and y are 0 (no need to rotate at all)
  half2 theta12_cs = normalize(vcDir.xy);

  // compute sine and cosine of phi angle
  half2 phi12_cs;
  phi12_cs.x = sqrt(1.h - vcDir.z * vcDir.z);
  phi12_cs.y = vcDir.z;

  half4 vResult;
  // The first band is rotation-independent
  vResult.x =  vZHCoeffs.x;
  // rotating the second band of SH
  vResult.y =  vZHCoeffs.y * phi12_cs.x * theta12_cs.y;
  vResult.z = -vZHCoeffs.y * phi12_cs.y;
  vResult.w =  vZHCoeffs.y * phi12_cs.x * theta12_cs.x;
  return vResult;
}
```

11.2 Analytical generation of cone and cosine lobes

Here is a listing of method that takes a direction and a cone angle as an input and returns SH coefficients of this cone of given angle rotated towards given direction:

```
half4 SHProjectCone(const in half3 vcDir, uniform half angle)
{
  static const half2 vZHCoeffs = half2(
                        .5h * (1.h - cos(angle)),           // 1/2 (1 - Cos[\[Alpha]])
                        0.75h * sin(angle) * sin(angle));   // 3/4 Sin[\[Alpha]]^2
  return SHRotate(vcDir, vZHCoeffs);
}
```

Here is a listing of method that takes a direction as an input and returns SH coefficients of hemispherical cosine lobe rotated towards given direction:

```
half4 SHProjectCone(const in half3 vcDir)
{
  static const half2 vZHCoeffs = half2(.25h,   // 1/4
                                       .5h);   // 1/2
  return SHRotate(vcDir, vZHCoeffs);
}
```

12 Appendix B. Reflective shadow map down-sampling filter

Here is a listing of shader that down-samples a reflective shadow map to one quarter (four times in each dimension) and makes clustering of 16 VLPs with respect to fluxes and directions of virtual point light sources:

```
half3 GetGridCell(const in half2 texCoord, const in float fDepth)
{
  // calc grid cell pos
  float4 texelPos = float4(texCoord * half2(2.h, -2.h) - half2(1.h, -1.h), fDepth, 1.f);
  float4 homogWorldPos = mul(g_invRSMMatrix, texelPos);
  return GetGridPos(homogWorldPos);
}

half GetTexelLum(const in RSMTexel texel)
{
  return Luminance(texel.vColor) * max(0.h, dot(texel.vNormal, g_lightDir.xyz));
}

RSMTexelOut IVDownsampleRSMPS(in IVDownsampleRSMPsIn In)
{
  // choose the brightest texel
  half3 vChosenGridCell = 0;
  {
    half fMaxLum = 0;
    for(int i=0;i<2;i++)
    {
      for(int j=0;j<2;j++)
      {
        half2 vTexCoords = In.texCoord + half2(i, j) * g_vSrcRSMSize.zw;
        RSMTexel texel = FetchRSM(vTexCoords);
        half fCurTexLum = GetTexelLum(texel);
        if(fCurTexLum > fMaxLum)
        {
          vChosenGridCell = GetGridCell(vTexCoords, texel.fDepth);
          fMaxLum = fCurTexLum;
        }
      }
    }
  }

  // fliter
  RSMTexel cRes = (RSMTexel)0;
  half nSamples = 0;
  for(int i=0;i<2;i++)
  {
    for(int j=0;j<2;j++)
    {
      half2 vTexCoords = In.texCoord + half2(i, j) * g_vSrcRSMSize.zw;
      RSMTexel texel = FetchRSM(vTexCoords);
      half3 vTexelGridCell = GetGridCell(vTexCoords, texel.fDepth);
      half3 dGrid = vTexelGridCell - vChosenGridCell;
      if(dot(dGrid, dGrid) < 3)
      {
        cRes.fDepth += texel.fDepth;
        cRes.vColor += texel.vColor;
        cRes.vNormal += texel.vNormal;
        nSamples++;
      }
    }
  }

  // normalize
  if(nSamples > 0)
  {
    cRes.fDepth  /= nSamples;
    cRes.vColor  /= 4;
    cRes.vNormal /= nSamples;
  }

  // output
  return cRes;
}
```

13 Appendix C. Propagation scheme

```
void IVPropagateDir(inout SHSpectralCoeffs pixelCoeffs,
                    const in IVSimulationPsIn In,
                    const in half3 nOffset)
{
  // get adjacent cell's SH coeffs
  SHSpectralCoeffs sampleCoeffs = SHSampleGridWithoutFiltering(In.gridPos, nOffset);

  // generate function for incoming direction from adjacent cell
  SHCoeffs shIncomingDirFunction = Cone90Degree(-nOffset);

  // integrate incoming radiance with this function
  half3 incidentLuminance = max(0, SHDot(sampleCoeffs, shIncomingDirFunction));

  // add it to the result
  pixelCoeffs = SHAdd(pixelCoeffs, SHScale(shIncomingDirFunction, incidentLuminance));
}

SHSpectralCoeffs IVSimulatePS(const in IVSimulationPsIn In)
{
   SHSpectralCoeffs pixelCoeffs = (SHSpectralCoeffs)0;

  // 6-point axial gathering stencil "cross"
  IVPropagateDir(pixelCoeffs, In, half3( 1,  0,  0));
  IVPropagateDir(pixelCoeffs, In, half3(-1,  0,  0));
  IVPropagateDir(pixelCoeffs, In, half3( 0,  1,  0));
  IVPropagateDir(pixelCoeffs, In, half3( 0, -1,  0));
  IVPropagateDir(pixelCoeffs, In, half3( 0,  0,  1));
  IVPropagateDir(pixelCoeffs, In, half3( 0,  0, -1));

  return pixelCoeffs;
}
```

14 Appendix C. Radiance injection shader

```
#define NORMAL_DEPTH_BIAS 0.25
#define LIGHT_DEPTH_BIAS  0.25

IVColorMapPsIn IVColorMapInjectionVS(const in IVColorMapVsIn In)
{
  IVColorMapPsIn Out;

  // get texture coords by vertex ID
  float2 texelPos = In.texelPos;
  Out.texCoord = texelPos;
  half2 screenPos = texelPos * float2(2, -2) - float2(1, -1);

  // sample depth and normal data with vertex shader texture look-up
  float depth = tex2Dlod(DepthVertexSampler, float4(Out.texCoord, 0, 0)).r;
  Out.normal = tex2Dlod(NormalVertexSampler, float4(Out.texCoord, 0, 0)).rgb * 2 - 1;

  // get world space position of the texel in the colored shadow map
  float4 homogGridPos = mul(g_injectionMatrix, float4(screenPos, depth, 1));
  float3 gridPos = homogGridPos.xyz/homogGridPos.w;

  // calc dir from original placement of pixel to this cell
  half3 gridSpaceNormal = normalize(TransformToGridSpace(Out.normal)) / g_GridSize.xyz;
  half3 alignedGridPos = gridPos;

  // shift injecting radiance towards the normal direction of the surfel
  alignedGridPos += gridSpaceNormal * NORMAL_DEPTH_BIAS;
  // shift injecting radiance toward the light direction
  alignedGridPos += g_dirToLightGridSpace.xyz * LIGHT_DEPTH_BIAS;
  // align depth of the texel to integer slice value
  alignedGridPos.z = floor(alignedGridPos.z * g_gridSize.z) / g_gridSize.z;

  Out.position = IVScreenPos(alignedGridPos);

  if(!IsPointInGrid(alignedGridPos))
    Out.position.xy = -2;

  return Out;
}
```

14 Appendix D. Work-around to resolve into a slice of volume texture on Xbox 360

Here is two code examples of how to create and resolve into the volume texture with dimensions 32x32x32. The first part denotes the creation of volume texture and some additional headers:

```
hr = m_pd3dDevice->CreateVolumeTexture( 32, 32, 32, 1, 0, D3DFMT_A8B8G8R8,
                                        D3DPOOL_DEFAULT, &m_pVolumeTex[0], NULL );

DWORD dwBaseAddress = m_pVolumeTex[0]->Format.BaseAddress<<GPU_TEXTURE_ADDRESS_SHIFT;

for ( UINT i = 1; i < 32; ++i )
{
  m_pVolumeTex[i] = new D3DVolumeTexture;
  XGSetVolumeTextureHeader(32, 32, 32, 1, 0, D3DFMT_A8B8G8R8, D3DPOOL_DEFAULT, 0, 0,
                           m_pVolumeTex[i], NULL, NULL );

  XGOffsetBaseTextureAddress(m_pVolumeTex[i], (VOID*)(dwBaseAddress - i * 4096), NULL );
}
```

The second part should be placed where the resolve happens:

```
for (UINT iSlice = 0; iSlice < 32; ++iSlice )
{
  D3DRECT SourceRect = { 32 * iSlice, 0, 32 * iSlice + 32, 32 };
  m_pd3dDevice->Resolve(D3DRESOLVE_RENDERTARGET0, &SourceRect, m_pVolumeTex[iSlice],
                        NULL, 0, iSlice, NULL, 1.0f, 0, NULL);
}
m_pd3dDevice->SetTexture(0, m_pVolumeTex[0]);
```

The source render target is a horizontally unwrapped volume texture with dimensions 1024x32 in this case.