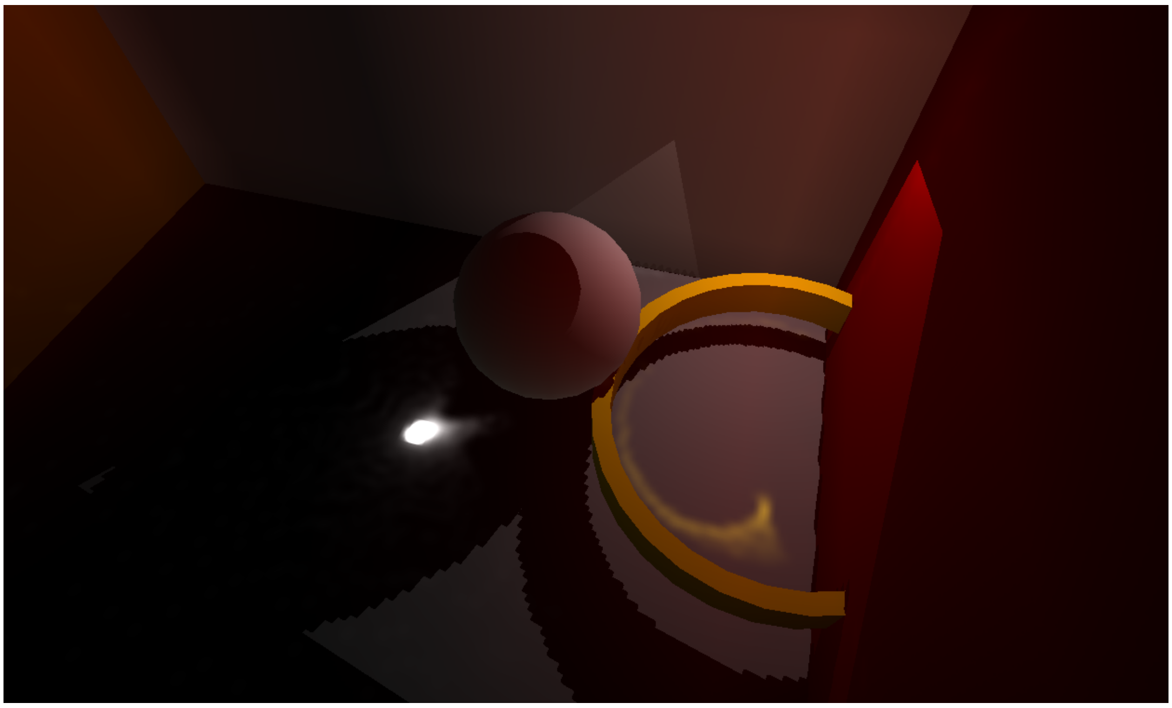
CHALMERS



Real-time Indirect Illumination

Image Space Light Lattice Photon Mapping with Spherical Harmonics

Master of Science Thesis in Computer Science

FREDRIK NORÉN
PATRIK SJÖLIN

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, 2010

Real-Time Indirect Illumination
Image Space Light Lattice Photon Mapping using Spherical Harmonics

Fredrik Norén
Patrik Sjölin

Examiner: Ulf Assarsson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Cover:
Image illustrating some of the effects achieved with the technique presented in this thesis.

Department of Computer Science and Engineering
Göteborg, Sweden 2010

Abstract

Rendering accurate indirect illumination is a hard problem that recently have seen algorithms closing in on real-time performance.

We present an algorithm that takes inspiration from two recently developed techniques on the subject. The algorithm provides both low-frequency indirect illumination and caustic effects with render times comparable to state of the art techniques in the area. The downsides are mainly discretization problems in the final rendering and the high dependancy on the CPU.

Sammanfattning

Rendering av högkvalitativ indirekt ljussättning är ett svårt problem inom datorgrafik där det nyligen presenterats algoritmer som närmar sig exakta resultat i realtid.

Vi presenterar en algoritm som är influerad av två nya algoritmer som behandlar detta problem. Algoritmen stödjer både lågfrekvent indirekt ljussättning och kaustik-effekter och är jämförbar i hastighet med det allra senaste inom området. De största nackdelarna med algoritmen är diskreteringsproblem i slutrenderingen och att den är CPU-beroende.

Contents

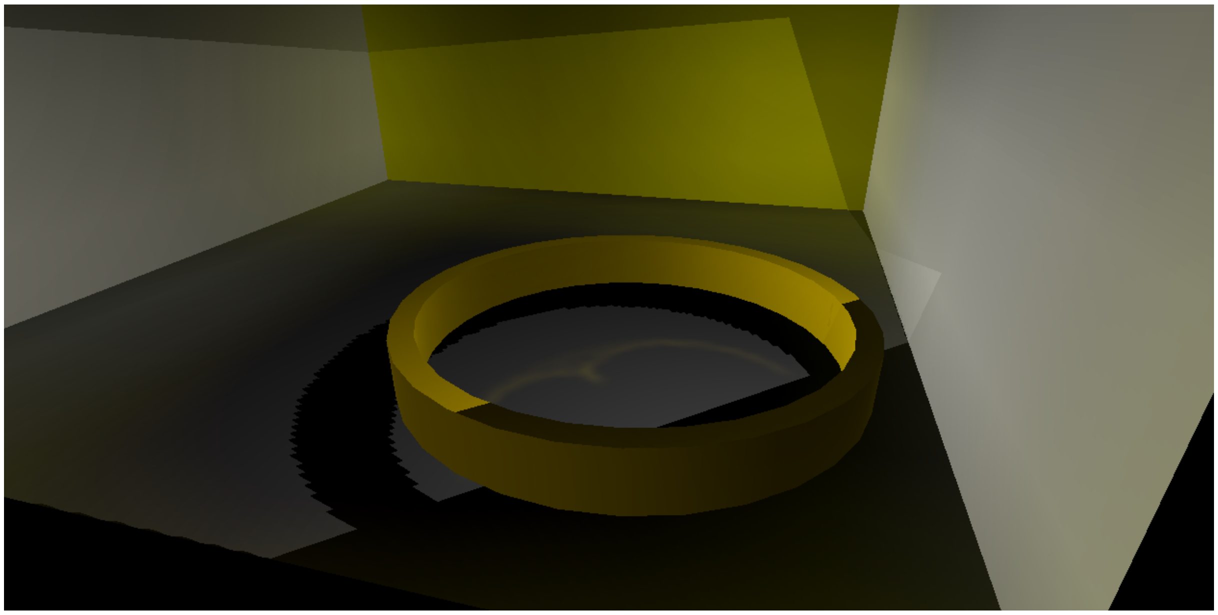


Figure 1: *Example image rendered using our method. Three effects are especially visible in this image; the indirect illumination and color bleeding on the walls, the caustics in the center of the ring, and the indirect shadows underneath the ring. The indirect illumination intensity is exaggerated for demonstrative purposes.*

1 Introduction

One of the most important groups of techniques in realistic offline rendering in recent years are the so called Global Illumination techniques. The term refers to techniques which simulate secondary bounces of light in a scene, which adds realism in the final rendering. Global illumination adds visual appeal and can sometimes help make images clearer to the observer, and achieving the same effect with other techniques is both difficult and time consuming. Traditionally, global illumination solutions have been limited to offline renderers, but recent advances in computer hardware, especially graphics cards, has open up the door to real-time, or at least interactive, global illumination solutions. A complete solution which runs in real-time and handles all of the most common lighting effects is still to be found, but partial solutions have been developed to take care of effects such as ambient occlusion, indirect illumination (color bleeding), contact shadows and caustics.

Our method, Image Space Light Lattice Photon Mapping (ISLLPM), takes the best of two recent papers, Image Space Photon Mapping (ISPM) [McGuire and Luebke, 2009] and Light Propagation Volumes (LPV) [Kaplanyan and Dachsbacher, 2010] (both described in more detail later), which both claim to handle subsets of global illumination in real-time. Our idea is to substitute the slow and often bottlenecking photon splatting in the ISPM paper, by a lattice of spherical harmonics, similar to the one presented in the LPV paper. By doing this we hope to increase the performance of the ISPM-technique without losing any capabilities of the technique.

Compared to the LPV technique, we should be able to handle more high-frequency lighting effects.

Whereas many of the recent interactive/real-time global illumination techniques focuses on games, we believe that our technique is more suitible for previews of offline global illumination renderings, which in many cases can take hours before the artist gets a result. Our technique would enable the artist to assess how light will flow in the scene, where for example the caustics will be visible, what areas will be too dark, or too brigth and so forth. Moreover, the artist would be able to move around objects, change material properties and lights, and get an instant response in the preview.

1.1 Problem statement

Light exiting a surface can be described using the so called rendering equation:

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_\Omega f_r(\mathbf{x}, \omega', \omega) L_i(\mathbf{x}, \omega')(\omega' \cdot \mathbf{n}) d\omega'$$

where L_o is the light exiting from a surface point $\mathbf{x}$ in the direction ω, L_e is the emitted illumination, f_r is the material properties, L_i is incoming light and $\mathbf{n}$ is the surface normal.

To render a scene we have to be able to solve the rendering equation. Both a rasterizing and a ray-tracing renderer have to at some point calculate how much light exits a surface at a particular position, in a particular direction; that is, solving the rendering equation. A first thing to notice about the rendering equation, is that we can split the incoming light into two groups; direct and indirect light, where direct light indicates light that comes directly from a light source and has not bounced of any surfaces, and indirect is all other light, that has bounced on one or several surfaces.

This gives us:

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_\Omega f_r(\mathbf{x}, \omega', \omega) L_{i_{direct}}(\mathbf{x}, \omega')(\omega' \cdot \mathbf{n}) d\omega' + \int_\Omega f_r(\mathbf{x}, \omega', \omega) L_{i_{indirect}}(\mathbf{x}, \omega')(\omega' \cdot \mathbf{n}) d\omega'$$

Depending on the material properties, this can be split down into three more sub-categories; **lambertian**, **glossy** and **reflective/refractive** materials.

All of these materials can be approximately solved for the direct illumination fairly easy, and most 3D applications do this today. This is usually done approximately by representing the reflection of the light source as a specular highlight on the surface.

This report will focus on solving two things; first we will simulate light transport in the scene, in which we handle any kind of material, and secondly we will use this information to render the indirect illumination.

Light transport in our solution is just a coarse approximation of reality, which means that we only have limited information about the light transport for each pixel we render. Reflective and glossy materials require more precision in the light transport solution, since their material functions heavily depend on the angle, whereas the lambertian materials are indifferent to where the light comes from and thus can work with all the light that hits the surface. Because of this we focus mainly on lambertian surfaces when it comes to rendering.

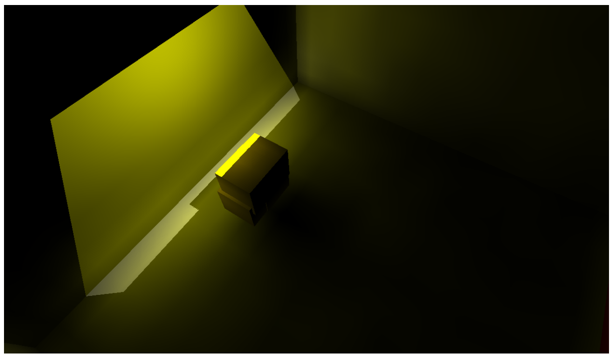


Figure 2: *Example of indirect shadows with ISLLPM*

2 Previous work

A large amount of work has been put into indirect lighting, both real-time and offline. Since this paper focuses on real-time indirect illumination, we will in this section primarily present work relevant to this field.

Common to almost all techniques presented here is the use of a deferred renderer, producing Geometry buffers (G-Buffers), which was introduced in [Saito and Takahashi, 1990]. Creating G-Buffers is usually the first step in a deffered renderer. The buffers store information such as normals, depths and diffuse colors. These buffers are then used to compute the final image.

Virtual Point Lights

Virtual Point Lights (VPLs) was first introduced in the classical Instant Radiosity paper [Keller, 1997]. The idea was to create a large number of point lights, each with a shadow map, to simulate indirect illumination. The biggest problem was however that you needed to render all these shadow maps for each VPL, and thus you had to choose between poor performance or temporal inconsistencies.

There exists a number of papers dealing with countering this tradeoff, such as [Laine et al., 2007] in which they present a clever scheme for deciding which VPLs needs to be removed and inserted to minimize temporal inconsistency.

[Ritschel et al., 2008] extended the instant radiosity idea further by rendering so called imperfect shadow maps for each VPL. Their idea is to create small and partial shadow maps for each VPL each frame, which they demonstrated gave little visual error. To do this, they created a point representation of the scene geometry, and for each imperfect shadow map they choose a subset of these points. This subset was then rendered to a texture, creating a coarse approximation of a shadow map, which was then improved in a push and pull step. By creating all these shadow maps in the same pass on the same texture resource, they were able to do this very quickly. These imperfect shadow maps were then used as shadow maps for the VPLs, in the same manner as the classical Instant Radiosity paper.

Reflective Shadow Maps

One way to quickly create VPLs is by rendering a so called Reflective Shadow Map (RSM) [Dachsbacher and Stamminger, 2005]. The RSM technique builds on the classical shadow mapping technique [Williams, 1978], but augments the shadow map with material and surface normals information. The original RSM paper does not create any explicit VPLs, but rather consider each texel of the RSM as a VPL. These VPLs are then randomly sampled during rendering to calculate the indirect illumination.

Other screen-space techniques

Screen-space techniques has become ubiquitous recently, and usually comes at a much lower cost than world-space alternatives. One fairly recent technique is the Screen-Space Directional Occlusion (SSDO) [Ritschel et al., 2009b] which builds on the popular Screen-Space Ambient Occlusion (SSAO) [Mittring, 2007] technique and enhances it with directional information, which can then used to calculate local indirect illumination, or together with an environment map; directional shadows. Although, among other drawbacks, it does not provide physically correct results, it is very fast and adds very little overhead to the traditional SSAO.

[Ritschel et al., 2009a] took this idea even further and did final gathering in screen space by quickly doing so called micro rendering of the scene from the world space position of each screen space pixel, through a technique similar to the Imperfect Shadow Maps technique. These were then used to calculate the indirect illumination of the surface.

Photon mapping

Traditional photon mapping was introduced in [Jensen, 1996], and the idea is to do path tracing from the light sources rather than from the screen, and then use this information to calculate indirect illumination. Path tracing itself was presented in its earliest form in [Kajiya, 1986].

[Dachsbacher and Stamminger, 2006] introduced the idea to use splatting for indirect illumination. Their idea was to create a limited number of VPLs from an RSM, and then render these VPLs using splatting, i.e. by drawing a geometry around the kernel to identify the pixels the VPL might affect, and then calculate the contribution and additively blend the results together.

The idea in ISPM [McGuire and Luebke, 2009] (detailed in section 3) is to accelerate parts of the photon mapping technique on the GPU, specifically the initial bounce and the final gathering, using splatting. The tracing however was still done on the CPU.

Lattice based

LPV [Kaplanyan and Dachsbacher, 2010] (detailed in section 4) presents a method which is based on light propagation in a lattice. Spherical Harmonics (SH) (excellent introduction in [Green, 2003] and [Sloan, 2008]) is used to represent both light and geometry in the grid cells. The propagation scheme is inspired by the Discreet Ordinance Method [Chandrasekhar, 1950].

The idea to use SH to represent geometry has also been presented in [Guerrero et al., 2008].

3 Image Space Photon Mapping

ISPM is a technique developed by Morgan McGuire and David Luebke in 2009, which utilizes the GPU to accelerate parts of the traditional photon mapping. The technique is reasonably fast and runs in real-time on concurrent hardware for some scenes, and converges towards perfect results (i.e. the same as offline rendered images) given enough time and number of photons.

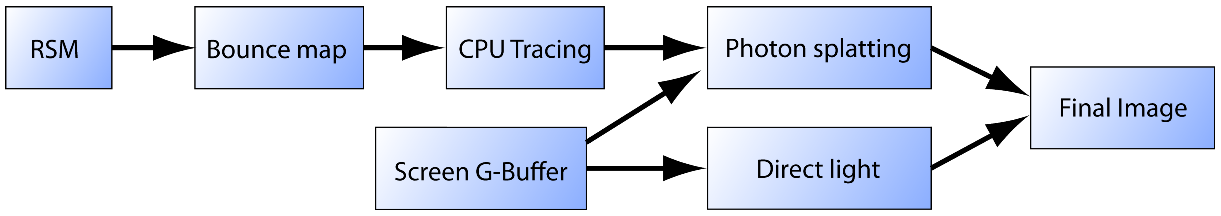


Figure 3: *Flow chart of the ISPM algorithm*

Some of the lighting effects the technique handles is; indirect illumination (including color bleeding), caustics (both reflective and refractive) and indirect shadows (also known as contact shadows). The technique comes with several drawbacks though; beyond the classical problems of photon mapping (area light sources and participating media) it also suffers from two bottlenecks (both of which will be detailed later); for some scenes the CPU tracing is a bottleneck, for other scenes the photon volumes splatting overdraw is an bottleneck.

We will here give a very brief overview of the technique. More details can be found in the original paper.

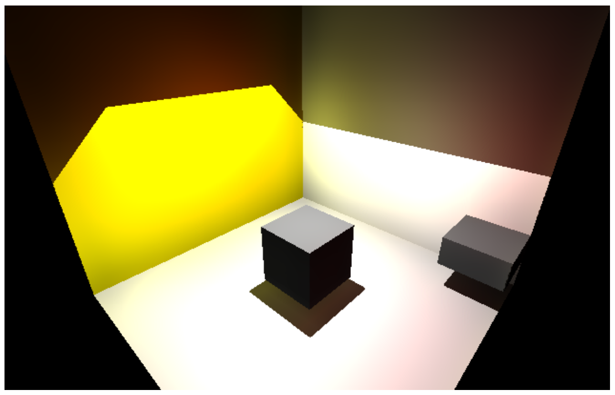


Figure 4: *Example of image rendered with ISPM*

3.1 Traditional Photon Mapping

Traditional photon mapping is based on the idea that you trace photons **from** the light source out into the scene, inserting a photon into a photon map each time the photon bounces off a surface. Then you use this photon map in the rendering step to calculate the indirect illumination a point receives.

The initial bounce is the computationally heaviest, since it contains the largest number of photons that are still alive. A typical survival rate of photons per bounce is 20%, meaning that all in all, the initial bounce often corresponds to almost 80% of the computations.

The final step is done by finding the N-closest photons to a screen space pixel's world space position, often accelerated by using for example a k-d tree [Fussell and Subramanian, 1988].

Both the first bounce and the final rendering steps can be accelerated using the GPU.

3.2 First bounce

The first bounce can be calculated by rasterizing the scene onto an RSM and then calculating the bounce maps, which contain the photons after hitting their first surface with all their values, and discards the photons that gets absorbed. These maps are used, together with a noise texture for random values, to calculate the initial bounce photons in their state just before leaving the surface. Most of these photons will be discarded.

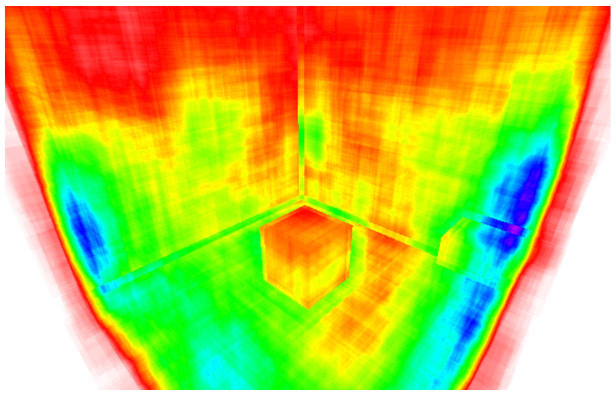


Figure 5: *Overdraw visualized. Darker colors indicate more overdraw, with a maximum of about 200 overdraws in the purple areas.*

3.3 Tracing

The bounce map is then read to the CPU, where the tracing of the photons continue. For most scenes, one CPU bounce is enough, but doing more than one is easily handled. At each bounce the photons are stored in the photon map, in the same manner as traditional photon mapping.

3.4 Photon splatting

Finally, we draw the the scene with direct lighting, and then render the photons on top of this. Each photon is rendered with a volume, covering its kernel size, using additive blending. For each pixel the incident light is calculated and permuted with the surface properties at that position, together with information from the photon, such as power, incident direction and path density.

3.5 Discussion

The major drawback with this technique is twofold. First, the CPU tracing puts a heavy load on the CPU, and becomes the bottleneck in complex scenes. Second, for the splatting to be smooth we need large kernel sizes, resulting in massive overdraw and thus a fillrate bottleneck.

The advantages are that the technique converges to the ground truth, and that it handles caustics and indirect shadows.

4 LPV

Cascaded Light Propagation Volumes for Real-Time Indirect Illumination (hereon referred to as LPV) [Kaplanyan and Dachsbacher, 2010] utilizes the GPU to a high degree to perform indirect illumination. It primarily handles low-frequency light, but can also handle glossy surfaces and participating media. However, it cannot handle caustics and other high-frequency lighting effects.

4.1 Reflective shadow maps and Virtual point lights

The first step of the algorithm is the initialization of the LPV. This is done by rendering a RSM, on the GPU, from every light source in the scene. Each texel in the RSM correspond to a VPL. The contribution from the VPLs are not computed individually. They are only used for the initialization of the LPV.

4.2 Injection

Two volumes are used to store injected data, a LPV and a geometry volume (GV). The LPV is a three dimensional grid where every grid cell contains a SH representing the directional distribution of intensity. The GV is also a three dimensional grid with grid cells containing SHs. It represents the geometry in the scene and is used during propagation in order to simulate ambient occlusion and avoid light to reach blocked surfaces. The SHs in the GV represents the directional distribution for probability of light propagation.

For every VPL stored in the RSM, SH coefficients are calculated using a conical projection on the normal to the surface of the pixel. The coefficients given by the projection is scaled with the flux stored in the VPL. Geometry information stored in the RSMs, such as normals and depths, are further used in order to initialize the GV in a similar manner.

4.3 Propagation

The propagation, as the name suggests, is a process of spreading light throughout the grid according to the directional distribution intensity stored in the SHs.

The first iteration operates on the LPV received from the injection stage. Every cell in the LPV propagates information to its six neighbours according to the three axial directions. Every neighbour of the source grid cell receives light on five of its six faces. The contributions from the five faces are added together and blended into the SH of that grid cell. Figure 7 illustrates this process in two dimensions by showing how an SH adds its contribution $I(\omega)$ to a face in direction ω. To determine the visibility of a face from the source cell's center, the solid angle is calculated from each face, which corresponds to the area of the face projected onto the sphere's surface. The final contribution is then calculated by multiplying the projected area with the intensity $I(\omega)$ found in the direction from the source cell's center to the center point of the face as seen in figure 7. The contribution $I(\omega)$ is then reprojected into the neighbouring cell's SH in the direction of the normal pointing towards that face.

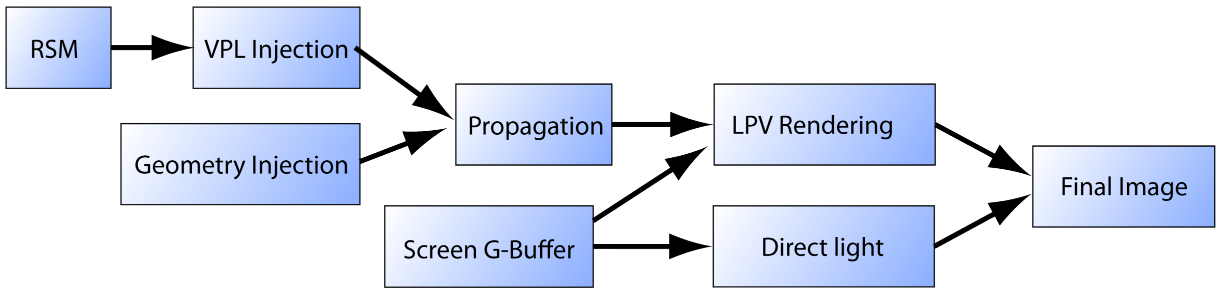


Figure 6: *Flow chart of the LPV algorithm*

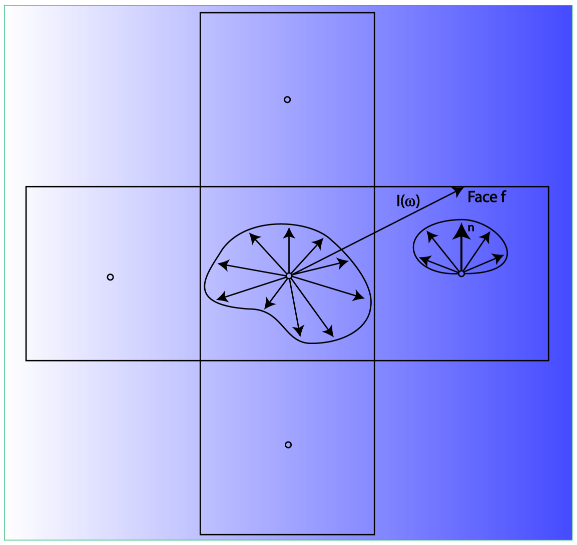


Figure 7: *This illustrates the process of propagating light into neighbouring grid cells where $I(\omega)$ is the intensity found by decoding the SH with direction ω. The neighbouring SH receives the contribution by projecting it with the normal n.*

To get good looking low-frequency lighting, several iterations needs to be performed. As mentioned earlier, the first iteration is run on the LPV received from the injection stage. Further iterations are run in a "ping-pong" manner where the second iteration is run on the results produced by the first iteration and the third run on the second and so on. During this process, a LPV containing the accumulated results from all iterations is used.

4.4 Cascaded Grids

The LPV-algorithm uses cascaded LPVs where grids closer to the viewer are of higher resolution than those far away. The benefit in doing so is that fewer iterations are needed during propagation thanks to the finer details produced from having a higher density of SHs closer to the viewer. To determine the grid resolution they simply scale the local grid resolution with the distance from the camera in the viewing-direction.

4.5 Rendering

The final rendering takes the LPV containing the accumulated results as input which is recieved from the iterations run in the propagation step. In figure 8, a result showing low-frequency indirect illumination can be seen.

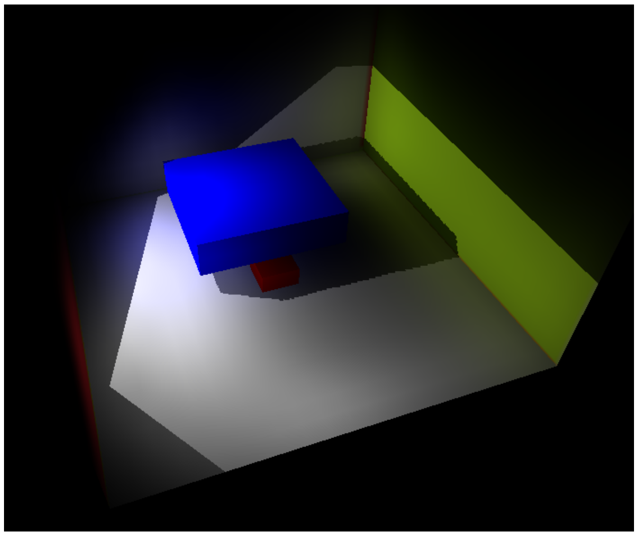


Figure 8: *This illustrates the low-frequency indirect illumination produced by the LPV-algorithm.*

The actual process of rendering the final image is very straightforward thanks to the simplicity of dealing with SHs. By extracting the information from the LPV based on the normals of the pixels seen by the camera, the LPV-algorithm additively blends the results together.

5 Our method

Our method is inspired by both the ISPM and the LPV papers. From ISPM we have the ideas of using an RSM to calculate the first batch of photon bounces, a so called bounce map, and to then trace these photons in the scene. From LPV we take the notion of representing lighting in the scene with spherical harmonics in a lattice, as well as parts of the injection scheme.

The result of this is that we can simulate all the effects the traditional photon mapping technique handles, i.e. our technique converges to photon mapping given enough time and processing power. At the same time it is also very fast, since we eliminate the overdraw bottleneck the ISPM technique suffers from.

The tradeoff consists of having to do the tracing on the CPU, which for us is the bottleneck on most systems.

The algorithm can be broken down into the following steps:

1. Do the first photon bounce on the GPU by calculating a **bounce map**
2. Read the photons to the CPU and do the remaining bounces as in traditional photon mapping
3. Inject the photons by point rendering into a **Spherical Harmonics Lattice** (SHL)
4. Create a G-Buffer for the screen
5. Calculate direct lighting
6. Draw the indirect illumination by using the SHL and the screen G-Buffer

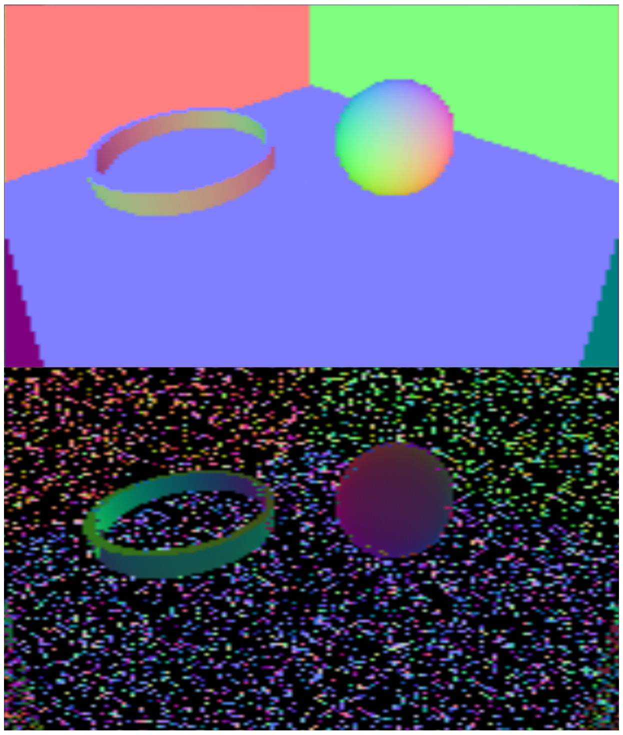


Figure 10: *Surface normals coupled with the outgoing direction vectors of the corresponding bounce map. The box is built using diffuse surfaces while the ring is fully reflective and the sphere fully transmissive.*

Name	Size	Description
P_i	float3	Photon power
n_{out}	float	Index of refraction for hit surface
ρ_p	float	Path density (scaled by likelihood of taken photon path)
ω_o	float3	Outgoing direction of the photon
$\vec{x}$	float	Photon position depth

Table 2: Bounce map layout

5.1 Bounce map

The first bounce when performing photon mapping on a scene is by far the heaviest bounce, since no photons have been absorbed yet. The number of live photons decreases significantly at each bounce, making computations a lot faster for greater depths. Thankfully, we can move the first bounce calculations to the GPU by rendering an RSM, which in turn is used for bounce map generation.

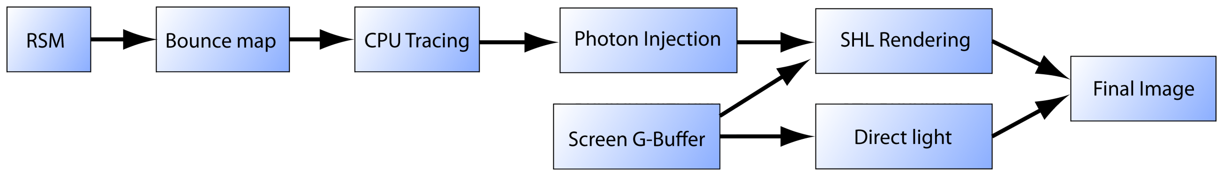


Figure 9: *Flow chart of our algorithm*

The RSM contain the world space position, the surface normal and the material properties of each texel as seen from the light source where each texel represents a live photon. These values are then used to calculate the properties of the outgoing photons (see table 5) surviving the first bounce.

There are four different types of bounces, not counting photon absorption, that can determine the outgoing direction of the photon after the bounce. These are:

- Lambertian - Uniform scatter in a hemisphere
- Glossy - Scatter in a lobe around the reflection vector
- Mirror - Perfect reflection
- Transmissive - Refraction

The resulting bounce map for an example scene is shown in figure 10.

The probabilities for the different types of bounces are contained in the previously mentioned material properties. They are divided into lambertian, specular (glossy and mirror bounces) and transmissive components with a sum $p = p_L + p_S + p_T \leq 1$ for each color. The probability of a photon being absorbed at the first bounce is thus $1 - \bar{p}$. The material properties also contain the index of refraction constant for the material as well as a specular exponent value that decides the characteristics of specular bounces.

In the case of a refraction bounce, the power is negated in order to be able to distinguish the refractive bounce from a normal bounce when extracting photons on the CPU. This is also the only case where the index of refraction variable might differ from its incoming value.

5.2 Tracing

Once the first bounce has been computed, the results are read back to the CPU for further tracing. This is done in the same manner as the ISPM technique, which in turn follows the traditional photon mapping tecnique quite closely.

While the bounce map stores the photons directly *after* their first bounce, the photon map stores the photons just *before* each bounce. The data stored for each photon contains all the information needed for the upcoming injection stage, where the color contribution to the target grid cell is computed. The final properties of the photons sent further to the injection stage can be found in table 3. The photons obtained from the bounce map are not stored in the photon map, since these surfaces are better handled by a direct illumination render pass.

Name	Size	Description
P_i	float3	Power
$\vec{x}$	float3	Position
$\vec{n}_p$	float3	Hit surface normal
$\vec{\omega}_i$	float3	Incoming vector

Table 3: Photon map layout

The tracing depth can be limited as a mean to reduce computation times. The first indirect bounce

Technique	Initial bounce	Remaining bounces	Rendering
ISPM	RSM + Bounce map	CPU Tracing	Photon splatting
LPV	RSM + Injection	Propagation	LPV rendering
Our method	RSM + Bounce map	CPU Tracing	Photon injection + LPV rendering

Table 1: Comparisons of steps in the techniques

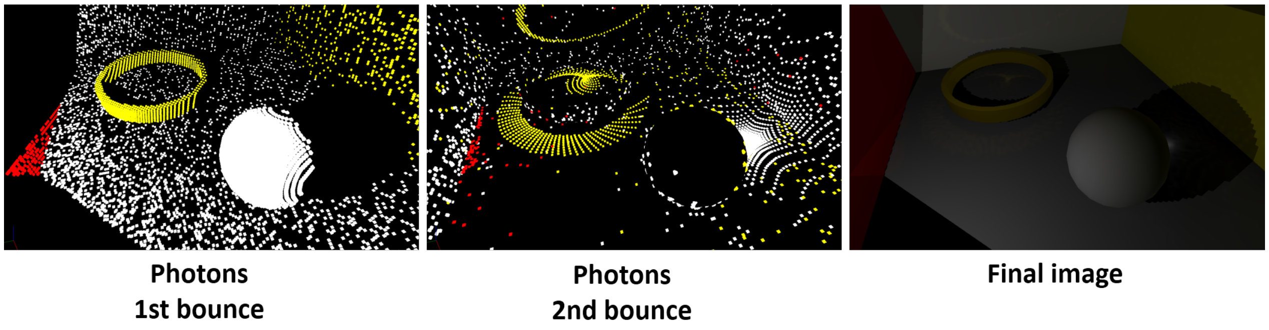


Figure 11: *Visual representation of a photon map at the first and second bounce respectively and the final rendered image. The photons in the first bounce are not stored in the accumulated photon map as this light is represented by direct lighting. The ring is reflective and the sphere is refractive/translucent, which is especially visible in the patterns produced by the photons in the second image.*

provides the most important contribution to the final image for diffuse and mildly glossy surfaces. It is therefore a reasonable optimization to skip further tracing for low-frequency light. Nevertheless, transmissive and highly specular surfaces still need multiple bounces since this light is high-frequency and gives small, but strong, contributions to the scene.

To accelerate the photon tracing, a kd-tree is constructed for each object. We also utilize multi-core capabilities of modern hardware by launching one thread per core to handle a subset of the photons. Since they operate on separate data we do not need any more synchronization mechanisms than a single barrier to tell when the tracing is completed on all threads.

5.3 Injection

Until here, our method has followed a similar algorithm as the ISPM to generate the photon maps. In the next step of our algorithm, we leave the ISPM completely to avoid the fillrate issues inherited with splatting used in ISPM. Instead of splatting photons to the screen using polyhedrons, our method uses lattices where the photons are represented by SHs as in the LPV-algorithm. In each grid cell there is a spherical harmonic approximating a color in a certain direction. Since each SH can only hold one color, three grids are needed in order to cover the red, green and blue spectrum.

In difference to the LPV-algorithm, our method does not need a second grid for storing blocking information from the geometry in the scene, since we use photon tracing.

When inserting photons into the grid, three values are needed per photon in order to perform projection, which is the construction of the coefficients corresponding to a SH. The formula used for projection is shown below, and is described in more depth in [Sloan, 2008].

$$f_l^m = \int f(s) y_l^m(s) ds$$

Each coefficient f_l^m is calculated by integrating the function $f(s)$, that we want to approximate, multiplied by the basis function y_l^m. Those coefficients will be used during the final rendering pass in order to decode the information encoded into the spherical harmonic. A representation of spherical harmonics contained in a grid is shown in figure 12.

Each photon is sent to the vertex shader together with its color, normal and incident direction. The incident direction s is sent as input to $y_l^m(s)$ and the color is used to scale the four coefficients returned by $y_l^m(s)$. Instead of integrating over the domain provided in the formula, we additively blend the contributions together for each photon corresponding to the given grid cell.

Our method wants to be able to represent light with different frequencies, spanning from low-frequency indirect illumination to high-frequency caustics. In order to achieve this, we need various densities of SHs corresponding to different frequency spectrums. We solved this by creating lattices of different resolutions, where higher resolution means a higher density of spherical harmonics and visa versa.

For each grid we perform a unique render pass covering all photons that corresponds to the frequency spectrum provided by grid being rendered. The actual algorithm for injection is very simple, as seen below.

1. Find the gridcell gc_i corresponding to the photon p_i
2. Compute the coefficients:
   $c_l^m = y_l^m(p_i.incidentDirection)$
3. Scale with red, green and blue:
   $red = p_i.power.red * c_l^m$
   $green = p_i.power.green * c_l^m$
   $blue = p_i.power.blue * c_l^m$
4. Blend the three results to their corresponding render target

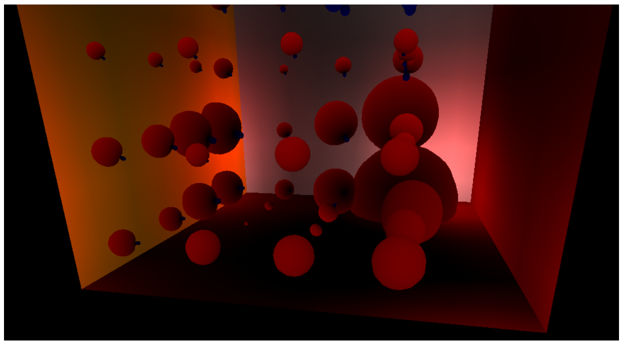


Figure 12: *This is a visual representation of spherical harmonics storing the directional distribution of red light. The blue sphere connected to each harmonics represents the negative direction of the SH. The size of the harmonics corresponds to the intensity of red light stored in that particular harmonic. Notice the large SH in the bottom right corner where the amount of color bleeding is large.*

5.4 Rendering

This section deals with the indirect illumination, which is the final step of the rendering. This final rendering is done after the direct lighting has been computed.

For being able to use the information stored in the grids to produce indirect lighting, a G-Buffer is needed where normals and depths are stored. With the direct lighting and G-Buffer rendered, the final render pass becomes an easy task thanks to the simplicity and efficiency inherited from SH. The formula used for extracting the coefficients representing colors from given directions can be seen below.

$$f(s) = \sum_{i=0}^{n} \sum_{m=-l}^{l} f_l^m y_l^m(s)$$

The power $f(s)$ in direction s is calculated by summing dot products of the basis function $y_l^m(s)$ and the four coefficients f_l^m stored in the grid. The direction s for which we want to approximate a color, is determined by the normal of that pixel. The world space normals are stored in the G-Buffer. The depth value in the G-Buffer is also needed in order to determine the world position of the pixel being rendered. The world position is then used to determine where to sample in the grid.

The three contributions $f(s)$, for red, green and blue, are additively blended into the final image that was previously drawn to the screen. This process is done in screen space for all pixels. Since we have a collection of grids spanning up the whole spectrum of frequencies, one render pass has to be done for each of those spectrums. No further work has to be done in order to perfect the final image. By just additively blending together the passes, a good looking result is found. The addition of indirect lighting is illustrated in figure 13.

The algorithm executed in the pixel shader can be described in a couple of small steps as seen below (notice that the color c is a vector of three components and s is a matrix of three sample vectors, each representing one of the red, green and blue grid).

1. Extract sample s from the grid using the world space position of pixel p.
2. Calculate final color c from $s \cdot y_l^m(n)$ where n is the world position normal to pixel p.
3. Blend color c to screen space position of pixel p.

A problem that arose from using sparse grids for low-frequency light is that borders appear between grid cells. This is because there is simply no guarantee that the information stored in two adjacent SHs results in a smooth transition on the surface reading from the two SHs. This problem is illustrated in figure 14:

We addressed the problem by blurring the information stored between adjacent SHs in order to produce redundancy amongst neighbouring harmonics to increase the likelihood of smooth transitions between grid cells. We tried using several kernels of different sizes and patterns.

With the grid being in three dimensions, a large amount of samples are needed in order to achieve

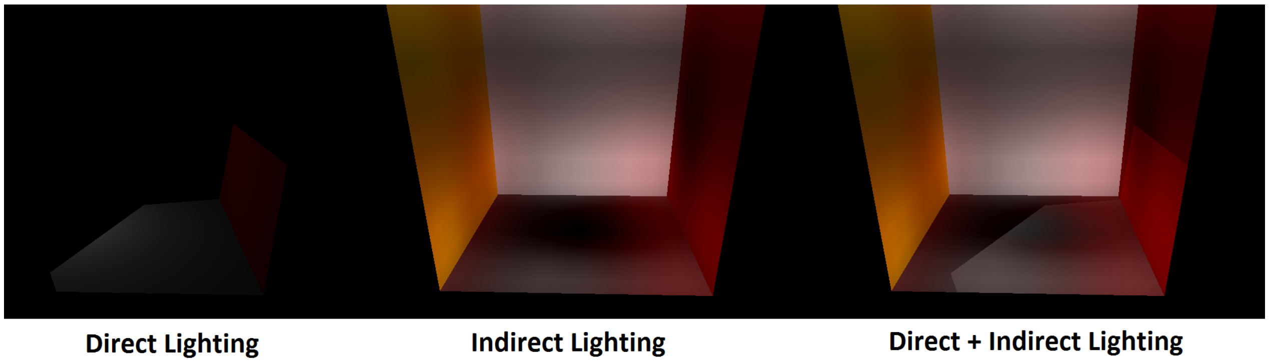


Figure 13: *An illustration of how the final rendering step adds indirect lighting to the scene. The second picture represents the indirect illumniation being added into the first picture representing the direct lighting. The final result from adding the two pictures together is displayed in the third picture.*

a sufficient amount of blurring. To get decent looking results we had to take at least one sample from every neighbour to a grid cell which gives rise to 26 samples due to the inherent complexity of three dimensions.

In figure 15 a 3x3x3 kernel is used to reduce the hard transitions between grid cells.

Since computing the blurring in screen space is not always favourable for low-freuqnecy grids, we perform the blurring in SHL space for sparse grids in order to save performance. This is similar to the propagation used in the LPV-algorithm where information was spread to neighbouring SHs.

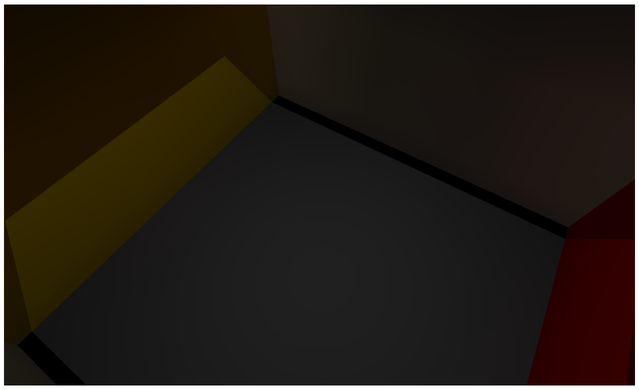


Figure 15: *A kernel using 27 samples (3x3x3) is used to invoke redundancy between neighbouring SHs.*

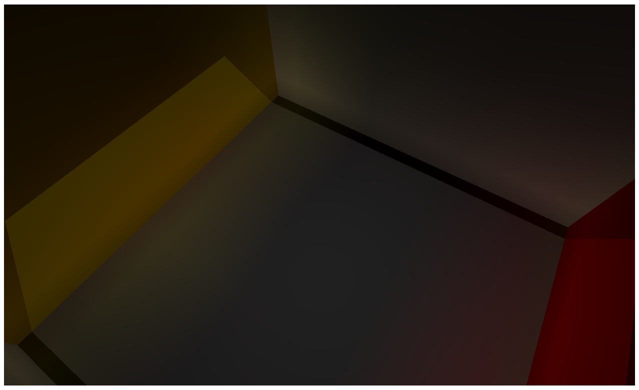


Figure 14: *Notice the borders that appear on surfaces where the information stored in the SHs differ in an unpleasant way.*

6 Implementation Details

This section goes into detail about the implementations that were used for running the tests described in section (7).

6.1 Platform

All tests have been run on an NVIDIA Geforce GTS 250 GPU together with an Intel Core 2 Quad Q8300 2.5GHz chip overclocked to 3.0GHz and 4GB of physical memory. The programs were written in HLSL and C# 3.5 using DirectX 10.0 via the SlimDX DirectX API and ran on a Windows 7 64-bit system. Our algorithm would have benefited greatly from a faster processor with more cores; more so than both of the compared techniques since they are more dependant on the GPU. Nevertheless, the chosen machine represents a typical gaming computer of today.

6.2 Photons

The initial photon count is set to 20000 photons with the G-buffer sizes set accordingly. The survival rate for diffuse surfaces is typically only 20%, but is much higher for more specular or transmissive surfaces.

The photons are currently set to only bounce once for diffuse surfaces in order to improve performance. As described in section 5.1, this does not affect the quality of the final image much.

6.3 Bidirectional scattering distribution function (BSDF)

We implemented a BSDF based on that of [McGuire and Luebke, 2009] that deals with lambertian, glossy, mirror reflections and transmissive terms. Instead of using three channels per term (one for each color component) a single value was used. The BSDF is used when building the bounce and photon maps.

6.4 Light Lattice

The [Kaplanyan and Dachsbacher, 2010] paper on Light Propagation Volumes lets the grid follow the camera as it moves around the scene. For simplicity, our implementations of both our own algorithm and LPV uses static non-cascaded grids fitted to the scene instead of the dynamic grid-solution presented in the LPV-paper. Implementing this for our algorithm should be no harder than doing it for the LPV-algorithm.

Three different grid sizes seem to be sufficient for representing low-, medium- and high-frequency light with grid side lengths of 4, 8 and 128 cells respectively for the cornell box shown throughout the report.

Various different sampling patterns used to solve the grid cell border problem described in section 5.4 were tested and a 3x3x3 grid with uniform weighting turned out to give the best results. Other tested sampling schemes were linear sampling, non-uniformly weighted 3x3x3 sampling, axis-adjacent sampling and a few different sampling patterns in a 5x5x5 grid.

6.5 Non-interactive mode

Considering one of our target applications of this technique is previews of renderings, we implemented a non-interactive mode. In this mode, the user first positions the objects in the scene. Then, a one time calculation of the lighting solution is computed, and stored in the SHLs. The SHLs are then used in the normal way, permitting the user to move around the camera freely, with an accurate lighting solution from any angel, but with a much higher framerate, since only parts of our algorithm is used at this stage. This also enables the user to use a much greater number of photons, getting a more accurate lighting solution.

7 Results & Discussion

In this section we present the results and limitations of our algorithm. We also present a comparison between our algorithm and the ISPM- and the LPV-algorithm.

7.1 Comparison between ISPM, LPV & ISLLPM

We ran the three algorithms on three different scenes as seen in figure 16. The first scene is a standard cornell box, the second is a cornell box with some more complexity added to it, and the final scene is sponza. In the cornell box scene, our method works fine running on 20 FPS or more, but in sponza the photon-tracing cannot match the performance of the LPV-algorithm.

7.2 Profiling

We profiled our algorithm on two different scenes. The algorithm is split into seven parts where two parts are run on the CPU. Those two are the initialization of the photon map and the bouncing. In table 5 the results are shown from profiling our algorithm in the cornell box scene which can be seen in figure 17.

By adding more complexity to the scene it is clear that the final step gets faster in comparison to the two steps run on the CPU, which can be seen in table 6. The scene used can be seen in figure 18.

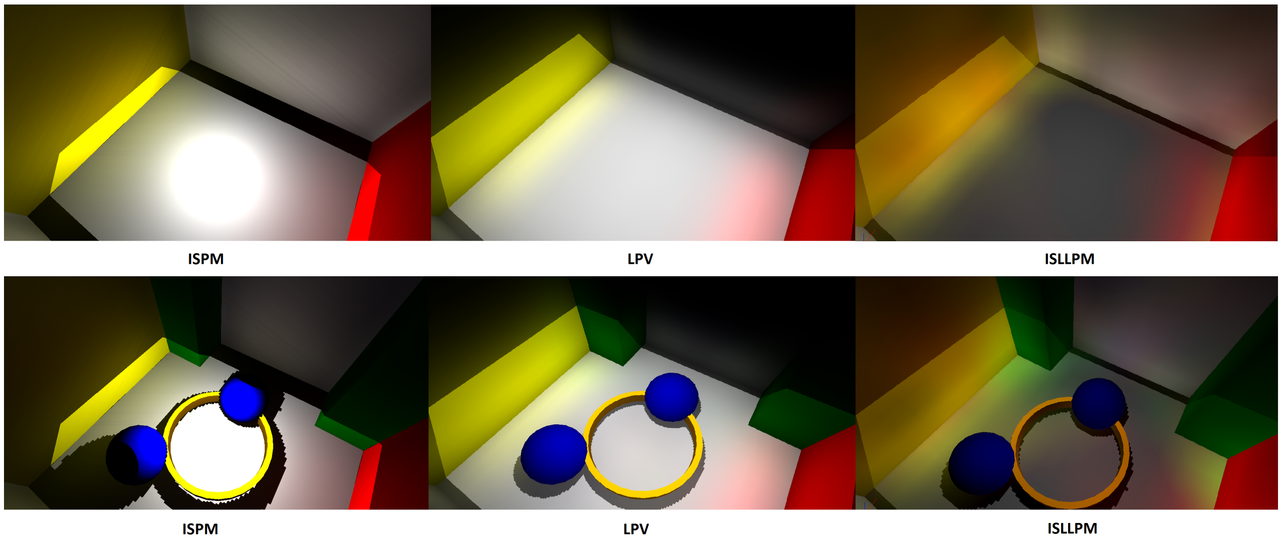


Figure 16: *The scene used for comparing the performance of the three different techniques presented in this paper.*

Scene	FPS ISPM	FPS LPV	FPS ISLLPM
Cornell box	3	66	25
Complex Cornell box	3	67	23
Sponza	1	12	1

Table 4: Comparison between the ISPM, the LPV, and our method for three different scenes.

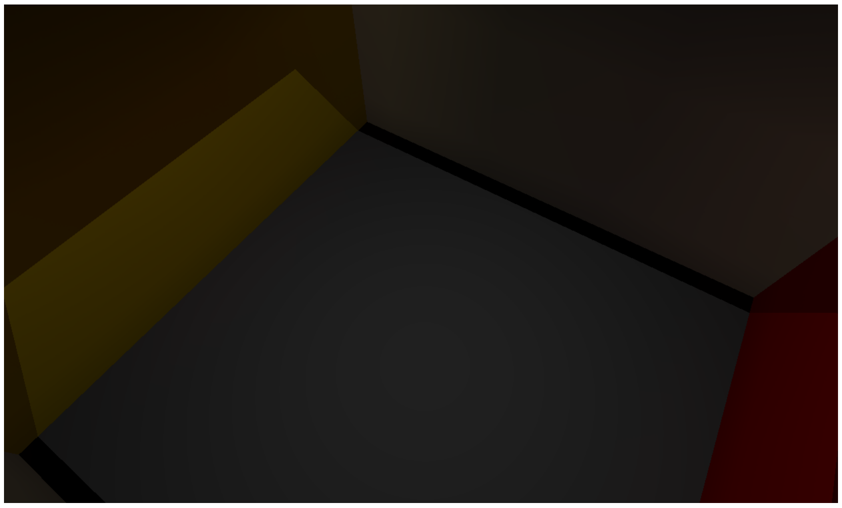


Figure 17: *Scene used for profiling.*

Operation	Time Elapsed [ms]	Amount of time [%]
Bounce map	< 1	< 1
Init Photon map	11	12
Tracing	45	49
Injection	2	2
Scrren G Buffer	< 2	< 2
Direct light	< 2	< 2
Draw lpv	31	34

Table 5: Profiling the different stages of the ISLLPM algorithm when only using low frequency lighting.

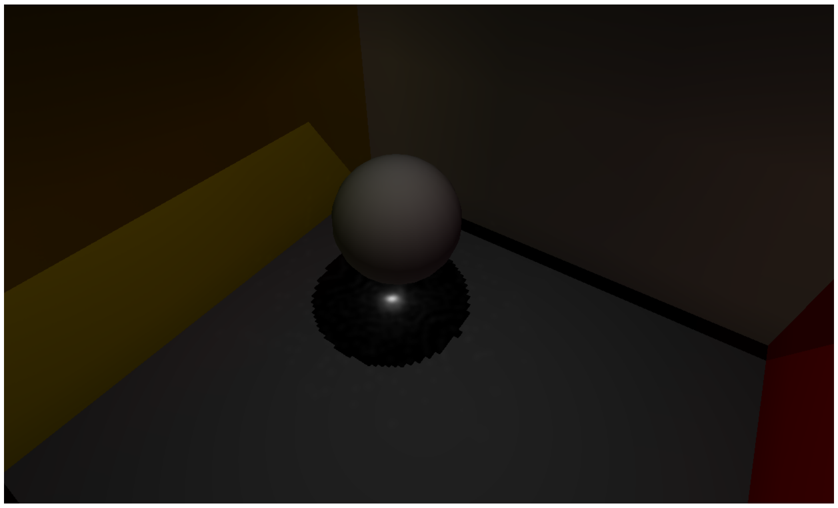


Figure 18: *Scene used for profiling.*

Operation	Time Elapsed [ms]	Amount of time [%]
Bounce map	< 1	< 1
Init Photon map	12	8
Tracing	94	66
Injection	2	2
Scrren G Buffer	< 2	< 2
Direct light	< 2	< 2
Draw lpv	31	21

Table 6: Profiling differenet stages of the algorithm when using both low and high frequency lighting.

7.3 Limitations

We found some limitations causing problems for both the performance and the graphical quality. A hardware dependant limitation is that the method requires shader model 4.0 to get access to the geometry shader used frequently in our method.

7.3.1 Performance

The performance of our algorithm is two-folded. It can handle everything from low-frequency lighting to high-frequency lighting, but at a cost, which is performance. The initialization of the photon map and the bouncing becomes a troublesome bottleneck, even when running the tests on a Intel Core 2 Quad Q8300 2.5GHz chip overclocked to 3.0GHz. In the case of more complex scenes, which are suitable for games, the tasks computed by the CPU takes more than 99% of the total running-time, and thus releases almost all load on the GPU. Tendencies of this kind can be seen in table 6 where the amount of time doing tracing increases just by adding a couple of spheres to the cornell box.

With upcoming modern hardware, where more cores are introduced on the CPU, such techniques as ours can come in handy. Anohter possibility is to implement the photon tracing on the GPU which might become more popular, especially with the compute-shaders introduced with Directx 11.0.

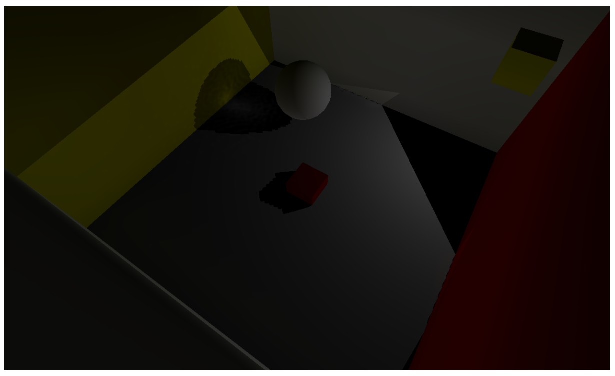


Figure 19: *The high-frequency caustics suffer from discretizing problems due to the grid.*

7.3.2 Visual defects

The only defect that we found using our algorithm, was the grainy high-frequency lighting. In most cases the grainy lighting was not visible. It can be partially solved by increasing the resolution of the grid or increasing the amount of photons, but we can never get around the inherited discretization problem that comes with using grids. Increasing the resolution of the grid also affects performance greatly since the number of grid cells increase exponentially with increasing the grid size.

Our method is not able to handle participating media since our photons are attached to surfaces and has no further communication with surrounding medias. The LPV-algorithm handles this very well thanks to their propagation step where information is further spread throughout the LPV.

Another graphical effect that we believe will be hard to achieve is glossy surfaces.

8 Conclusions

The algorithm presented managed to run faster than our implementation of the ISPM-technique while still providing caustics which is not handled by the LPV-algorithm. There were some discretization problems for high-frequency light as expected, but most of this could be removed by sacrificing performance.

The algorithm presented has many possible improvements and features that can be added and some of these are presented in the Future work section.

8.1 Future work

Photon tracing

In most scenes, our technique is bound by the CPU photon tracing. Although work has been done in our implementation to optimize this step, there is most likely much more that can be done.

An interesting step would also be to move the tracing to the GPU. Modern hardware and software allows generic programs to be run on the GPU, which opens up the door to practical ray tracing on the GPU.

High-frequency lighting

We handle high-frequency light by having a high resolution SHL. As described in the results section, the problem with this is that we have to trade between high memory usage and discretization problems.

An alternative would be to use photon splatting for high-frequency light [McGuire and Luebke, 2009], but still use our SHLs for the low frequency light. The problem in the ISPM paper with the photon splatting was overdraw, but since the high-frequency lighting effects usually only cover a very small portion of the screen, the overdraw should be limited.

Cascading SHLs

Our solution uses a hierarchical set of SHLs, to handle both high- and low-frequency light. However, these does not scale with the distance from the camera. This means that we in many cases get a too low SHL resolution close to the camera, and a too high resolution far away from it. To solve this, we suggest implementing something similar to the cascading LPVs in the LPV paper [Kaplanyan and Dachsbacher, 2010].

References

[Chandrasekhar, 1950] Chandrasekhar, S. (1950). *Radiative Transfer*. Dover Pubn Inc.

[Dachsbacher and Stamminger, 2005] Dachsbacher, C. and Stamminger, M. (2005). Reflective shadow maps. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 203–231, New York, NY, USA. ACM.

[Dachsbacher and Stamminger, 2006] Dachsbacher, C. and Stamminger, M. (2006). Splatting indirect illumination. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 93–100, New York, NY, USA. ACM.

[Fussell and Subramanian, 1988] Fussell, D. S. and Subramanian, K. R. (1988). Fast ray tracing using k-d trees. Technical report, Austin, TX, USA.

[Green, 2003] Green, R. (2003). Spherical harmonic lighting: The gritty details. *Archives of the Game Developers Conference*.

[Guerrero et al., 2008] Guerrero, P., Jeschke, S., and Wimmer, M. (2008). Real-time indirect illumination and soft shadows in dynamic scenes using spherical lights.

[Jensen, 1996] Jensen, H. W. (1996). Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pages 21–30, London, UK. Springer-Verlag.

[Kajiya, 1986] Kajiya, J. T. (1986). The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150.

[Kaplanyan and Dachsbacher, 2010] Kaplanyan, A. and Dachsbacher, C. (2010). Cascaded light propagation volumes for real-time indirect illumination. In *I3D '10: Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 99–107, New York, NY, USA. ACM.

[Keller, 1997] Keller, A. (1997). Instant radiosity. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 49–56, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Laine et al., 2007] Laine, S., Saransaari, H., Kontkanen, J., Lehtinen, J., and Aila, T. (2007). Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007*, pages xx–yy. Eurographics Association.

[McGuire and Luebke, 2009] McGuire, M. and Luebke, D. (2009). Hardware-accelerated global illumination by image space photon mapping. In *Proceedings of the 2009 ACM SIGGRAPH/EuroGraphics conference on High Performance Graphics*, New York, NY, USA. ACM.

[Mittring, 2007] Mittring, M. (2007). Finding next gen: Cryengine 2. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, pages 97–121, New York, NY, USA. ACM.

[Ritschel et al., 2009a] Ritschel, T., Engelhardt, T., Grosch, T., Seidel, H.-P., Kautz, J., and Dachsbacher, C. (2009a). Micro-rendering for scalable, parallel final gathering. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers*, pages 1–8, New York, NY, USA. ACM.

[Ritschel et al., 2008] 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. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pages 1–8, New York, NY, USA. ACM.

[Ritschel et al., 2009b] Ritschel, T., Grosch, T., and Seidel, H.-P. (2009b). Approximating dynamic global illumination in image space. In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 75–82, New York, NY, USA. ACM.

[Saito and Takahashi, 1990] Saito, T. and Takahashi, T. (1990). Comprehensible rendering of 3-d shapes. *SIGGRAPH Comput. Graph.*, 24(4):197–206.

[Sloan, 2008] Sloan, P.-P. (2008). Stupid spherical harmonics (sh) tricks.

[Williams, 1978] Williams, L. (1978). Casting curved shadows on curved surfaces. *SIGGRAPH Comput. Graph.*, 12(3):270–274.