Indirect illumination using photon splatting

Miklas Hoet

Supervisors
dr. Michael Wand
Marries van de Hoef, MSc

Master's Thesis
ICA-3981134

Universiteit Utrecht

Abstract

Indirect illumination algorithms exist in many forms, but the state-of-the-art real-time algorithms often limit the rendering equation in terms of visibility determination or leave it out completely. The main focus of this thesis is to adapt the existing high quality photon mapping method in order to make the visibility determination closer to the real-time field. Similar to the splatting indirect illumination technique, our technique works by placing many indirect light sources in the scene. However, we consider these light sources as small orthographic area light sources instead of point light sources. This allows us to introduce a novel technique for visibility determination. This technique bundles light rays into a single ray for a more efficient, but still plausible, indirect illumination if many samples are averaged. The final illumination is then applied by splatting this single ray using an orthographic shadow map that serves as a local visibility determination for the entire bundle. Furthermore, we introduce a new technique for stratifying these light sources in 3D space.

A highly parameterized version of this algorithm was implemented, as well as a photon mapping implementation to serve as a ground truth. Both algorithms are assessed in terms of quality, resources and speed. Our result show an improvement in rendering time with minor loss of quality, although the time required for our algorithm to converge is still far out of the real-time range. However, our techniques have been concocted keeping hardware acceleration in mind. Such implementation might show promising results.

Preface

This work would not have been possible without the countless meetings and the endless stream of e-mails with my supervisors, Michael Wand and Marries van de Hoef, and the support by friends and family with Marjolein Geldof and Michiel Hoet in particular.

Furthermore a special thanks to Larian Studios for providing me with an internship, with Bert Van Semmertier in particular for serving as my daily supervisor and for the insights in commercial graphics applications.

Table of Contents

1 Introduction

Indirect illumination is a key aspect for rendering a realistic scene: almost every videogame or movie that aims to achieve realism includes a form of indirect illumination. It is responsible for shading an entire scene based on the light that reflects off already lit surfaces. A part of this reflected light might get reflected again. Calculating the indirect illumination of a scene requires solving a complex and continuous function. Every additional light bounce loses some energy, depending on the material properties. This property can be exploited to limit the indirect illumination techniques to a few bounces, as in many cases even a single bounce is enough to create a plausible result [1].

However, the function remains complex. In contrast to direct illumination, the number of light sources is a lot higher if we know that every surface that receives light becomes a secondary light source. Furthermore, popular direct illumination techniques (such as deferred rendering [2]) omit all geometry that is not directly visible from the camera. Indirect illumination might come from geometry outside of the viewing frustum, making it impossible to neglect any geometry. The continuous function combined with a high light source count and a high geometric complexity make up the main challenges.

Due to its importance, a lot of techniques exist to calculate indirect illumination, ranging from real-time approximations to high-quality implementations that might take hours to render one frame. Most of these real-time techniques pose very strict time limits and often contain noticeable inaccuracies. High-quality approaches do not contain these inaccuracies but go well over the real-time budget. For those reasons, real-time applications often revert to precomputed solutions: a high quality technique is used to compute an intermediate result that can be applied in real time. Unfortunately, all of these methods include downsides such as artifacts, limited dynamic geometry, long rendering times, etc.

The goal of this thesis is to lessen the gap between the set of real-time and high quality techniques. Many real-time techniques neglect the visibility determination. Can the visibility determination from a high quality technique such as photon mapping technique [3] be adapted in order to calculate visibility more efficiently?

Our adaptations involve using a stratification pass, orthographic shadow maps for indirect light source visibility determination and splatting the indirect illumination directly into an accumulation buffer. Our proposed pipeline is highly compatible with state-of-the-art techniques, such as the reflective shadow map [4] and imperfect shadow map [5] techniques, allowing a possible hardware accelerated implementation.

2 Related work

As mentioned before, a large set of indirect illumination algorithms exist. This section briefly lists the major techniques in the field of global illumination. It goes in a bit more detail regarding the *photon mapping* and *splatting indirect illumination* techniques, as they lay the basic foundations for our adaptations.

2.1 Real-Time solutions

The most basic real-time approximation of indirect illumination is *ambient lighting*. The albedo of every pixel is multiplied with a constant value so that shadowed pixels do not appear completely black. A popular adaptation is *screen space ambient occlusion* (SSAO [6]) or the improved *image-space horizon-based ambient occlusion* (HBAO [7]). These techniques include the depth buffer to approximate local visibility: surfaces that are partly enclosed by other geometry are assumed to receive less indirect illumination. Albeit being extremely fast and simple, these techniques come with the downside that the near-uniform shading is perceivably wrong. Furthermore, the assumption that enclosed geometry always receives less illumination is theoretically not correct and lacks color bleeding[1], but it does make for an impressive approximation. The main assumption in these techniques is used in our work as a quality assessment parameter: enclosed geometry such as grooves are generally shaded slightly darker, and there are generally no completely dark areas within a scene. As our implementation (see section 4) is limited to single bounce indirect illumination, a basic ambient color can be used to fill in for unshaded areas.

Another common set of techniques used in real-time environments are precomputed solutions. An intermediate result is generated using a high quality technique and is then applied in real time. A widely used set of precomputed techniques are based on radiosity [8]. In a first step, the geometry is divided into patches and the visibility between the patches is calculated. In a second pass, the patches receive and bounce light to other patches. As soon as a dynamic

[1] A surface that receives colored light from a photon that bounced from a colored surface.

object comes into play, the illumination for objects affected by this movement must be recalculated. This poses a strict limit on the number of possible dynamic objects. Pre-computed techniques have the major advantage that the indirect illumination applied at real-time is of high quality but without the high cost. The major disadvantage is that the speed-up is limited to the pre-calculated data. Depending on the used technique, the geometry and lighting settings used in the real-time scenario must be largely equal to the precomputed settings. However, there is a trend towards WYSIWYG[2]: developers wish to see how the final result will look like without any pre-computation steps [9]. Therefore, our algorithm is developed without any use of pre-computed elements.

A more recent trend seeks to simplify the scene and to calculate the indirect illumination using this simplified model. The imperfect shadow map technique [5] starts by placing many virtual point lights (VPL's) within the scene. Because calculating the visibility for all of these light sources would be too expensive, the scene geometry is simplified into single points and extremely low resolution shadow maps are used. It is capable of generating plausible results at interactive frame rates, but this low resolution geometry and visibility determination causes small light leaking[3] artifacts. However, even with such low indirect shadow quality, the results are still plausible, showing that the visibility term of the rendering equation [10] should not be neglected, but can be approximated.

The light propagation volume technique [6] starts by creating a voxelized version of the scene geometry. Direct light is then injected into this voxel grid and propagated towards its neighbors. This extreme bundling of light has the major advantage of processing large light transports at an affordable cost: the technique is capable of approximating indirect illumination in a dynamic scene with dynamic lights in real-time. But again, as a result from the geometry simplification, small light leaks appear. Furthermore, every propagation step is expensive and suffers from numerical dissipation, causing the light to diffuse per step. Every iteration leads to an

[2] What You See Is What You Get.

[3] Light that appears to shine through a solid object.

exponential loss of precision. The concept of bundling rays is the foundation for our visibility determination and splatting technique.

Voxel cone tracing [9] uses a less coarse approximation by creating a voxel octree from the scene. Direct illumination is injected and filtered into this data structure. In a final step, cones are traced through this octree, gathering the indirect illumination. Due to this directional sampling, directional information can be used to compute specular indirect illumination where other methods often limit themselves to diffuse only illumination. Due to the usage of the octree, the scene geometry is simplified less when comparing to the voxel grid in Light Propagation Volumes. A grid tightly enclosing geometry causes less light leaks to occur. Unfortunately, the build-up of this data structure and the cone tracing are expensive. Data from the previous frames can be re-used, but this limits the number of dynamic objects. Furthermore, larger scenes require a memory intensive data structure.

2.2 High quality solutions

All of the real-time techniques include some form of limitation, ranging from limits on dynamic objects to visible artifacts such as light leaking. High quality techniques do not have these downsides but often require a multiple of the rendering time. One common approach is path tracing [10]: for every screen pixel, rays are shot into the scene. A ray bounces in a random direction from every hit surface until it hits a light source or a termination criterion is reached. A photon is then traced back to the camera pixel, while the material properties of every hit surface alter the color and intensity of the photon. By averaging the results of a large amount of rays per pixel, convergence is achieved, resulting in photorealistic images. By using stochastic sampling, the integration domain is much smaller as compared to distributed ray tracing [11]. Path tracing has interactive implementations [12], albeit not without artifacts such as noise.

Instead of shooting rays and direct them towards a light source, the photon mapping [3] technique starts from shooting photons from the light source. Similar to path tracing, the photon is bounced on based on termination criteria. A common criterion is called *Russian roulette*: samples are rejected based on a one in *n* chance (where *n* can be influenced by the material properties, the current bounce count, etc.). The photon's properties such as the remaining intensity and color are then stored in the *photon map*, a data structure optimized for fast nearest neighbor queries required in a next step. After a suitable structure, such as a balanced KD-tree, is built, a regular ray tracing pass is conducted. In this secondary pass, the location of the first intersection is used as input for a *k-nearest-neighbors* search. The average distance and properties of the *k* nearest photons are used as a density estimation. Selecting only the nearest neighbors might cause corners to become overly bright compared to flat surfaces since the distance between photons is smaller. This can easily be overcome by limiting the nearest-neighbors search to a disc-based search. The normalized color of all the nearest photons is used to simulate the color bleeding effect. To achieve plausible results, a high number of photons and nearest neighbors have to be used. The injection of these photons into the *photon map* and especially the querying of this data structure are very time consuming events. Because the photons are shot from the camera, this technique allows for easy prioritizing of certain surfaces, and is therefore a beloved technique for rendering caustics. A high number of photons is aimed towards surfaces such as glass or shiny metal, allowing for very high detail in these areas.

2.3 Splatting indirect illumination

Photon mapping works well, but has the bottleneck of an expensive injection and querying of the *photon map*. Furthermore, it is primarily based on ray casting [13], which generally does not map to the GPU hardware as well as the Z-Buffer approach [14]. The *splatting indirect illumination* technique [15] uses the same concept as photon mapping but uses a reflective shadow map [4] to determine the surfaces that cause first-order indirect illumination, allowing for the utilization of a rasterization pipeline. Every pixel within the shadow map is a surface that reflects lights and is considered to be a pixel light, similar to the basic idea of instant radiosity [16]. The most important pixel lights are selected using importance sampling, and their contribution is then splatted directly into an accumulation texture. Due to the vast number of pixel lights, even when using importance sampling, the visibility term of the rendering equation is ignored.

3 Photon splatting

The main contribution of this thesis is the introduction of the *photon splatting* pipeline that requires less resources than the original photon mapping implementation by splatting the indirect illumination directly into an accumulation buffer. In contrast to the *splatting indirect illumination* technique, our technique calculates indirect visibility efficiently using orthographic shadow maps and introduces a stratified photon sampling technique. The algorithm is built as a pipeline: the output of every step is the input for the next step. This means that any step can be implemented in any possible way, as long as the results are the same. Figure 1 gives an overview of the entire algorithm, which will be described in detail throughout this chapter.

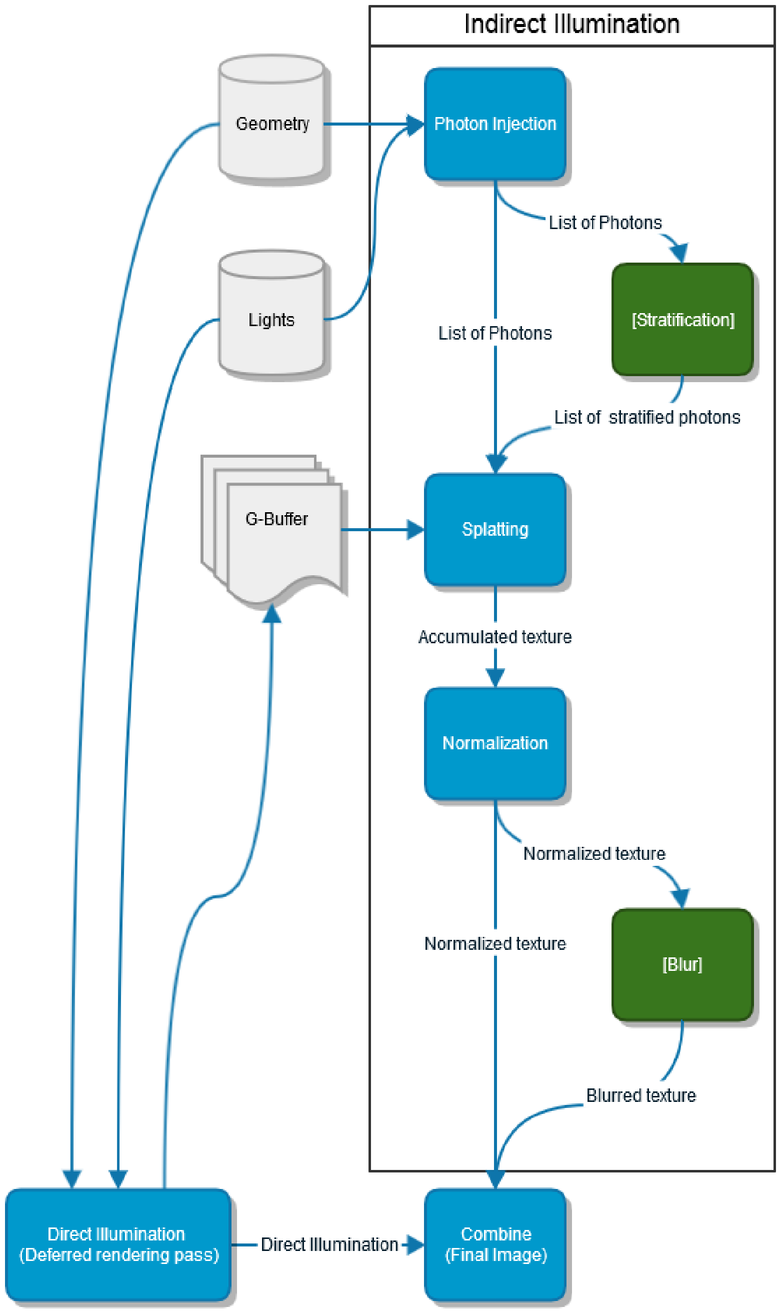


Figure 1: Algorithm overview

3.1 Direct illumination

A first step is to calculate the direct illumination, separated from the indirect illumination. Our approach is capable to converge to a correct direct illumination solution as well, but as direct illumination requires higher detail, a specialized algorithm is preferable. A deferred rendering based algorithm [2] is recommended since the *splatting* stage can benefit from reusing the geometry buffer (G-Buffer) instead of a full-geometry pass.

3.2 Photon injection

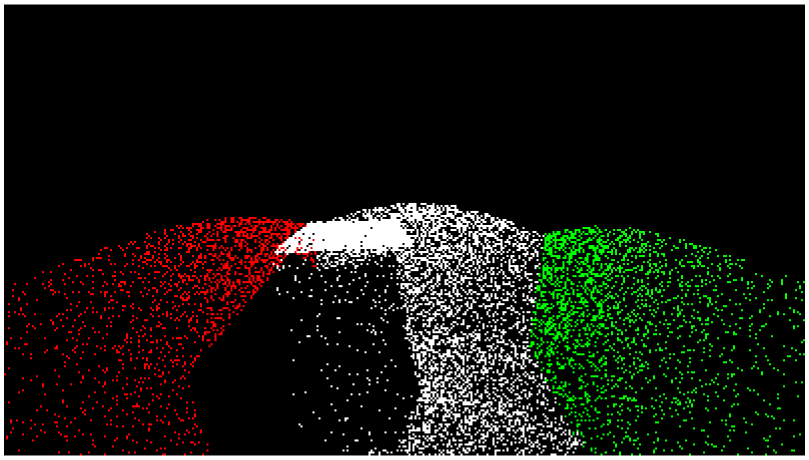


Figure 2: Direct hit photon locations

The first step regarding the indirect illumination is finding suitable locations for virtual point lights. Any surface that receives direct light is a valid indirect light source. There are many different approaches that can be utilized, as long as the output is a list of indirect light sources. A first possible method is using the first step of the *photon mapping* algorithm: photons are shot from the light source and the first point of intersection with the scene geometry is calculated. These points are visualized in Figure 2. For single bounce indirect illumination, the location combined with the photon's color (based on the light color, the surface albedo and material properties) and a pseudorandom direction are stored within a suited data structure. This direction is limited to point away from the surface (within 90 degrees of the surface normal vector). If more than a single bounce is required, the photons can be bounced on multiple times before being stored.

A second, faster but more limited method, is generating a rasterized shadow map for the light source: every pixel in this map can then be considered as a pixel light source. Due to the rasterization nature of this pass, it is generally faster, but generating second order indirect illumination becomes a non-trivial assignment.

3.2.1 Stratified photons

As both suggested strategies might introduce a very large number of light sources, an optional stratified sampling pass is introduced. Similar to photon mapping, our algorithm requires a high overlap to converge to the correct solution. Furthermore, the *splatting* step is computationally more expensive than the injection step. These factors make it beneficial to combine neighboring light sources.

In a first step, a voxel grid is placed encapsulating the scene geometry. Instead of storing the photons regularly, they are injected into the corresponding cell. The resolution of this grid can be chosen freely: a higher count will result in a better, but computationally more expensive image.

In a second step, all the information from the photons within a cell is then combined. The color can simply be atomically added, while the position and direction need a better heuristic. Atomic operations are considered to be expensive and tend to limit the multithreading capabilities. Nevertheless, research by Kaplanyan et al. [17] has shown that hardware accelerated atomic operations on voxel grids with a 32^3 resolution can easily be calculated in real-time. Positions cannot be averaged as the average position might end up within geometry and averaging directions would result in a biased direction. The simplest solution to both problems is selecting a random photon's position and direction. This random direction causes the expected value to be equal to the ground truth, yielding an unbiased heuristic. An issue that might arise is the selection of a less relevant photon: in case 99% of the photons within a voxel point in a certain direction, there is still a 1% chance that the chosen direction is a deviating one. As all color information is combined, this might appear as an artifact such as wrongful color bleeding.

A more complex heuristic that solves this issue can be achieved by selecting the photon the closest to the average position or finding the direction closest to the dominant direction.

However, every possible heuristic can still introduce artifacts, a common and well-known disadvantage of any technique that utilizes scene discretization.

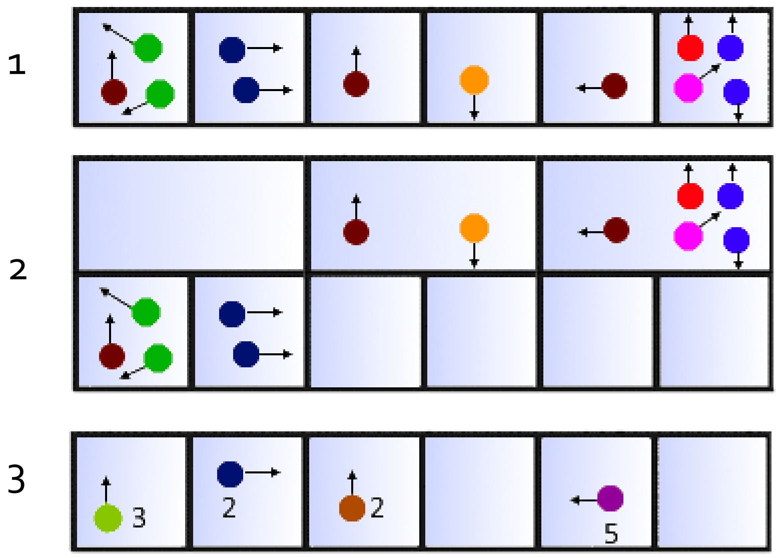


Figure 3: example of the photon stratification process

Some voxel cells might contain a very low number of photons, contesting the integrity of the stratified sampling. To prevent this, the nearest neighbors of this voxel grid can be used to combine cells that contain less than a certain threshold. Figure 3 explains this process in detail. First, all photons and their properties (color, location and direction) are injected into the voxel grid. Secondly, all cells that contain less than the threshold (a minimum of two photons per cell in this example) are joined with their neighboring cell. If the threshold is still not reached after the first combination, this second step can be repeated on a next level. Finally, the photons colors are averaged and the photons are combined. In this example, the random photon heuristic is chosen. The combined photons, with an additional weight, are then stored for use in the next pipeline phase. Due to randomness of this discretization, the expected result remains correct and unbiased, but diverges from the ground truth. Convergence can be achieved by combining many different iterations. An infinitely small voxel size would converge to the ground truth, but would combat the goal of the stratification.

Depending on the scene and voxel grid size, some photons might fall outside of this grid. This might be the case in large outdoor environments. A possible solution is to implement a cascading system to greatly increase the coverage, similar to [17]. Another solution would be to simply ignore light sources that are too far away from the camera.

3.3 Splatting

After the list of virtual light sources has been assembled and optionally stratified, the illumination has to be applied to the scene. In contrast to the *splatting indirect illumination* technique [15], we do not consider these light sources as point lights.

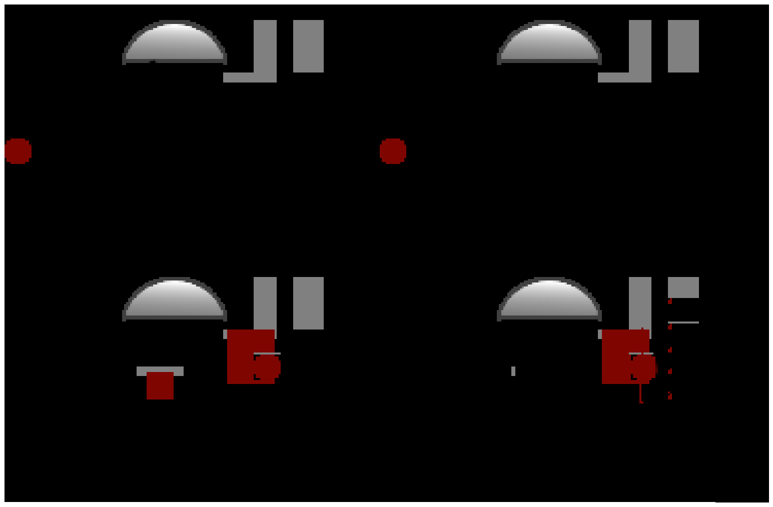


Figure 4: splatting location determination

Point lights have the disadvantage that they have a large radius of influence, causing a high number of pixels involved per light source and prohibitively expensive visibility queries. As this technique is based on the Monte Carlo[4] principle, it requires as much overdraw as possible in order to converge. Even with a light volume optimization[5], the number of affected pixels remains fairly large. Instead we propose a splat close to the affected geometry, allowing for far less pixels to be affected per splat while keeping the visibility calculations relatively cheap.

Finding the splat location starts based on the data from the injection step. Recap that in the injection step, a photon is shot from the light source and its properties (position, color and direction) are generated based on the surface of the first intersection (see image 1 in Figure 4). From this photon's position, a ray is shot in the previously defined direction and the distance *d* to the first intersection is found (see image 2). From this distance, a small step *b* is taken back along the ray (see image 3) and this position is used to create an orthographic shadow mapping frustum (see image 4). This orthographic shadow mapping frustum represents the bundled rays.

[4] Monte Carlo methods rely on many random samples to solve a numerical problem.
[5] Rasterizing geometry that resembles the light type and range, such that only those pixels require calculations.

Figure 5 displays the difference between our approach and a typical splat. Splatting such point light that is located against a wall requires a hemisphere as a light volume. If it is a powerful light source with a large radius and a low geometry count in the vicinity, this might result in a large number of wasted calculations. By doing a small extra step and finding the geometry location, much more efficient splats can be conducted. Furthermore, due to the small size of the individual splats, many small shadow maps can be used which require less precision as the light volumes are closer to the actual geometry. The major disadvantage is that many samples must be taken in order to emulate one single light source and that the closest geometry for every sample must be found. In addition, it is hard to find how many rays have to be shot to emulate a hemispherical light source. Therefore, only a single direction is chosen per light source, and the average of many light sources is taken. This process cannot be easily rasterized, but modern ray casting implementations [12] are capable of tracing millions of rays per frame, well over the thousands required for our pipeline.

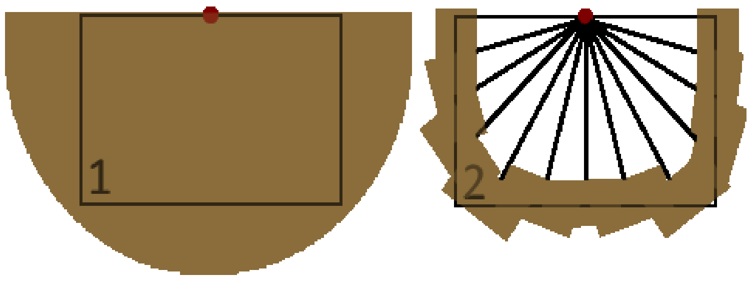


Figure 5: Point light hemisphere splatting (1) vs close range splatting (2)

After a suitable location is found, the depth of the scene is rendered into an orthographic shadow map to determine indirect illumination visibility. This frustum represents a bundle of parallel rays originating in the vicinity of the light source, mimicking an area light source. If the ray originates close to other geometry, the bundle represents non-existing information and might introduce artifacts. However, due to the overlap of the many samples, these errors are smoothed out. The actual width and height of this shadow frustum depends on the parameters set by the artist: a larger shadow map will cause more overlap, resulting in a faster convergence and requires less photons. However, less photons and larger regions might smooth out high-frequency details. The depth of the frustum is dependent on the light source range and the number of samples. A larger depth will cause more overlap, but requires more computational power.

An issue with an orthographic shadow map is that it might intersect with the scene geometry. This means that back-facing triangles have to be considered to prevent light leaks. Furthermore, wrong depth values might be caused by the shadow map frustum sticking trough geometry, such as in corners. In some scenarios this might cause the depth test to fail and therefore create falsely shadowed areas. This latter issue is easily solvable by adapting the shadow depth test to include these borderline cases.

After or interleaved with the generation of the shadow maps, the pixels affected on screen by the respective shadow map have to be updated. The G-Buffers from the direct illumination step can be reused as we are only interested in the indirect illumination within the camera frustum. The illuminated pixels are then found similar to regular shadow mapping approaches: the positions within the shadow map are compared with respective positions from the depth buffer in order to determine visibility.

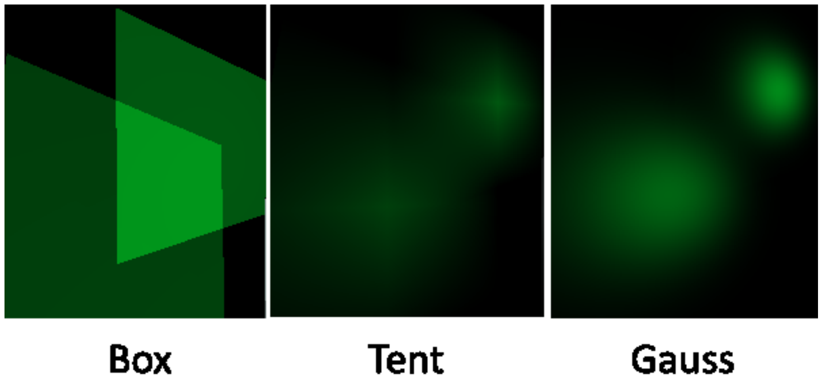


Figure 6: Example filters of two neighboring splats, with equal kernel sizes.

The shading of every splat causes an area to become darker or brighter. The shading itself is done using the Lambertian reflection model, where the splat color (*S*) is defined by the angle between the incoming light (*L*) and the surface normal (N), as well as the photon's color (*C*) and surface albedo (*A*).

$$S = (L \cdot N) * C * A$$

The final color that gets written into the buffer is the splat color multiplied by a weight. This weight comes from a filter function. Possible functions are a box, a tent or a Gaussian filter (see Figure 6). In the illustration, the kernel sizes have been kept equal. But in order to work efficiently, a Gaussian or a tent filter requires a larger size in order to produce the same level of overlap.

The center of the filter is applied around the middle of the shadow map, and the resulting color of the filtering operation combined with the weight are atomically added. These weights are required to account for a smooth overlap between different splats. Combining splats with a box filter causes sharp edges between different splats, whereas a tent and Gaussian filter produce increasingly smooth overlaps respectively. A main advantage of using shadow mapping as a visibility test is that they have less issues with light leaking, as is the case with voxel-based solutions or range queries such as in regular *photon mapping*. On the other hand, depending on the used shadow map technique, common artifacts of these techniques (such as shadow acne[6], aliasing[7] or Peter Panning[8]) might appear. Due to low frequency details and the high overlap, these artifacts should remain within reason.

[6] Moiré patterns within a shadowed area due to precision issues causing depth tests to fail.
[7] A low shadow map resolution might cause jagged shadow edges.
[8] Shadow detached from the caster, creating a wrongful floating impression.

3.4 Normalization

After all photons have been splatted into the accumulation buffer, they have to be normalized. Due to the Monte Carlo nature of our algorithm, certain areas might receive more splats than others. However, the number of splats onto a pixel does not make for a correct solution, yet the shading of every splat does. A larger number of splats approximates the ground truth better, but also requires more processing power. Because the colors and weights are all properly stored within the accumulation buffer, the normalization step is nothing more than iterating over the color value of every pixel within the buffer and dividing it by the total accumulated weight.

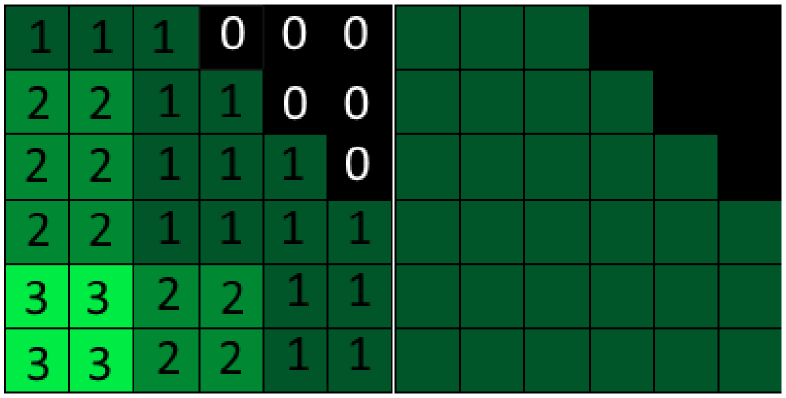


Figure 7: Accumulation buffer before and after normalization

An example of color values and their respective weights in the accumulation buffer can be seen in Figure 7. This weight can also be adapted in order to approximate the result better with a lower sample count.

Our Monte Carlo algorithm requires a lot of overlap to converge. Increasing the number of splats works, but has a large impact on the final rendering time. A possible solution to this problem would be to use the weights as an estimator of intensity. However, due to the Monte Carlo nature of our algorithm, this might introduce noise. Furthermore, the ratio of the weight with respect to the maximum weight has to be used as an estimator, requiring an expensive atomic operation to determine.

3.5 Blur

Our splatting strategy might introduce noise if the solution was undersampled. A common solution would be to take more samples. However, more samples also requires more computational power. Therefore, an optional blur pass is introduced. Indirect illumination has very few sharp details, so blurring might be a cheaper solution than taking more samples in certain scenes.

3.6 Combine

In the final pipeline step, the direct and indirect illumination are combined by simply adding them together. An exposure parameter allows artists to define the contribution of the indirect illumination to the scene. Additional post processing effects such as bloom can easily be added to improve visual quality. The final image can then be presented and the pipeline can start all over again.

3.7 Overview

Our proposed novelties fit into a common indirect illumination pipeline perfectly, as long as the proper output for every step is respected. The major advantages of our strategy with respect to photon mapping is the computational cost. Photon mapping requires storage of a large amount of photons within a computationally expensive balanced KD-tree. Our approach bins the photons within the respective voxel.

Our splatting approach iterates over every virtual light source and the pixels it affects. Photon mapping iterates only once over every pixel, but has to do an expensive nearest-neighbor search per pixel. Due to the bundling of the rays, the visibility determination should remain reasonably cheap and the number of wasted fragment calculations low.

4 Implementation

The basic pipeline can be implemented in many ways. We have created an implementation that is built to validate the integrity and flexibility of the algorithm. The main focus was to test the versatility of the algorithm and was therefore highly parameterized. As speed was not the main concern, most of the implementation allows for a faster approach. During this chapter, our implementation is described combined with hints towards faster methods where applicable.

4.1 Direct illumination

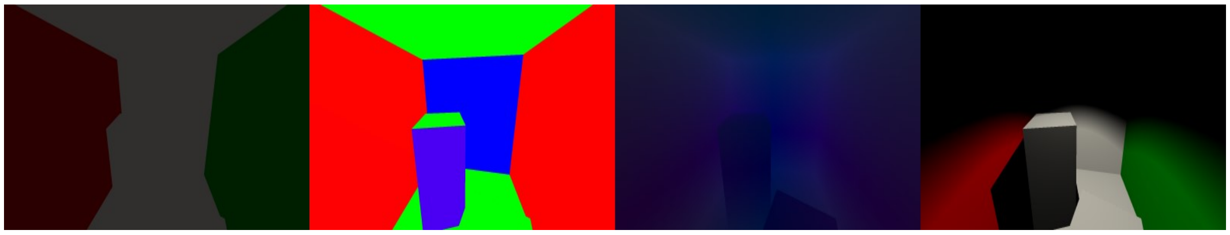


Figure 8: From left to right: albedo, absolute normals, depth and direct illumination

The first step of the pipeline has been implemented using the game industry's standard approach: using deferred shading [2]. In a first pass, all the geometry closest to the camera is found using a naïve CPU ray tracer. This means neither hardware acceleration nor acceleration structures were used. A hardware accelerated rasterized implementation would greatly outperform a brute force CPU ray tracer, but falls outside the scope of this thesis. After the first intersections have been found, their data (surface albedo, position & normals, see Figure 8) is stored in the respective buffers. An additional pass is conducted for every light source to additionally blend the direct lighting onto the scene. Direct shadows are generated using shadow feelers[9], allowing for pixel perfect shadows. Other shadowing techniques, such as shadow mapping or shadow volumes, could be used as well when using a rasterized pipeline.

[9] Shadow feelers are rays cast from a surface towards a light source. If the light source is not the first intersection, the surface is shadowed.

4.2 Photon injection & stratification

Similar to *photon mapping* [3], rays are cast in random directions (limited to the light sources angles). The first surface hit's position, color and a pseudorandom direction (limited to the hemisphere defined by the surface normal) are stored into an array. In case the optional stratification step is selected, the photons are stored into this array as well, but instead of indexing them sequentially, they are indexed using their grid position. The grid is created around the center of the camera. The size of the grid is determined using the scene geometry by finding the boundaries and dividing the remaining space. Figure 9 displays the Cornell box scene with the voxel grid's index position as a color. Photons that hit the surface are injected into the respective voxel cell. After all photons have been injected into the grid, the grid is iterated in blocks. Every block contains eight neighboring cells. If one of those cells contains less photons than the threshold parameter, the colors of all 8 cells are combined and a random photon is selected to represent this cells direction and position. If every cell does contain enough photons, the list of photons within a cell's color is averaged as well and again a random photon is selected. This makes the maximum number of photons with a voxel grid equal to the number of cells. This latest fact is interesting for possible hardware accelerated implementations where certain sizes for resources have to be known beforehand.

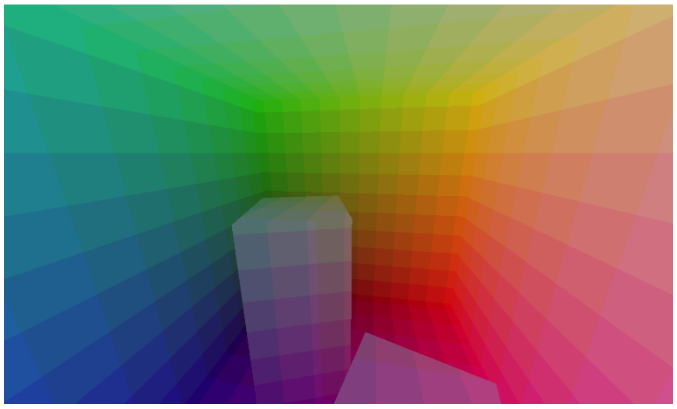


Figure 9: Voxel grid indexes (16^3 voxel grid applied on the Cornell box scene)

Another possible, faster, approach is used in the *splatting indirect illumination* technique [15]. The scene is rasterized from the light source and the texture's pixels are used as indirect light sources. This technique is a lot faster than ray casting, but also limits the indirect illumination to a single bounce. If a high resolution texture is used, a stratification pass is beneficial. For such purpose, a 3d texture can be used, possibly with hardware-accelerated stratification in the form of mip-mapping[10].

A third possible method is to use a scene simplification model, similar to *light propagation volumes* (LPV) [17]. The scene geometry is simplified into voxels first. In contrast to the original LPV technique, a valid position and direction within the voxel has to be defined as well. The scene is then rasterized from the light source to generate a shadow map. Finally, all voxels are traversed and their visibility is determined using the stored position and the shadow map. The content of every voxel is then similar to our results after stratification.

4.3 Splatting & normalization

Finding the splat location is as simple as doing an intersection test between the scene geometry and the ray defined using the photon's location and direction as calculated in the previous step. A small parameterized value is then subtracted from this position to serve as the center of an orthographic viewing volume. The size of this volume is dependent on both the artist's parameter and the optional stratification pass: if the photons were not stratified, it would have resulted in multiple splats whereas now there is only a single splat. This is counterbalanced with a larger shadow map size. The number of pixels within this shadow map is kept low as the shadows of indirect illumination generally do not have high frequency details. Even with such small shadow maps, performance of rendering thousands of shadow maps is not possible in real-time, but a real-time approximation such as imperfect shadow maps [5] could be a proper substitute for indirect illumination visibility determination.

After the visibility is determined, a full screen pass is conducted where the pixel values from the depth buffer (as used in the direct illumination rendering passes) are compared with the depths in the shadow maps. Indirect light sources might illuminate off-screen objects as well, but as we are only interested in single bounce indirect illumination we only consider visible geometry. Higher order indirect illumination is easily achievable by allowing photons to bounce

[10] Decreasing lower resolution versions of the original image.

multiple times in the injection phase, before storing them. As an optimization, a light volume can be rasterized to limit the number of fragment shaders.

The final color that gets added into the accumulation buffer is the color from the photon, multiplied with the surface's albedo, multiplied with a filter weight. The distance with respect to the center of the shadow map is used as input for the respective filter (box, tent or Gaussian). This weight is stored within the alpha channel of the same texture. As multiple threads are splatting at the same time, locks are required to prevent race conditions[11]. The normalization pass is then just a matter of iterating over every pixel and dividing the color by its weight.

Due to our implementation being limited to single bounce indirect illumination, certain areas might appear completely black. These areas have a zero-weight value. As we need to check for zero-values in order to prevent artifacts caused by a division by zero, we can fall back on the ambient illumination technique easily. In these cases, we just add a small amount of light to these pixels.

4.4 Blurring & combination

To deal with any possible noise, which is highly possible when the solution is undersampled, a simple Gaussian blur has been implemented. As the noise depends on the number of samples and the scene, multiple kernel sizes can be chosen (3x3, 5x5, 7x7 or 9x9 pixels). As a Gaussian filter is separable, two passes are conducted: a horizontal and a vertical blur. After the optional blurring, the indirect illumination is ready to be combined with the direct illumination. This is done by simply iterating over all the pixels and adding the values, where the indirect illumination is first multiplied by an exposure value to allow brighter or darker scenes.

[11] If multiple threads access the same resource, they might overwrite each other's results.

4.5 Photon mapping

Next to our own algorithm pipeline, the standard *photon mapping* algorithm [3] has been implemented to serve as a ground truth. *Photon mapping* is a high quality approach and has many similarities with regard to our approach. This allows for the implementation to be built onto the same framework. The exact same deferred rendering algorithm is utilized to calculate the direct illumination separately, as the positions from the depth buffer can be reused to find the position for the nearest-neighbor query. The first step towards calculating indirect illumination is shooting photons from the light source, bouncing them until the Russian roulette fails and storing them in a Knn tree. In a second step, every position in the depth buffer is traversed and the *n* nearest-neighbors are found. The build-up, balancing and querying of the KD-tree is done using the nanoflann [18] library. Faster, GPU based implementations can be found in [19], but were not required (see the timings in section 5.1). The resulting colors and positions from the query serve as parameters for the final color calculations. To deal with the light bleeding problem of regular Knn sampling, the nearest-neighbor query weights sample further away from the tangent vector[12] less heavily or ignore them completely if they point away more than 90 degrees from the normal vector.

4.6 Overview

This implementation can be seen as a proof-of-concept. The main focus is on the versatility and possibilities of the proposed algorithms from section 3. The same goes for the photon mapping implementation: faster techniques exist, but a highly parameterized implementation is preferred in order to compare both.

[12] A vector perpendicular to the normal vector

5 Results

In the two previous sections, the algorithm and a specific implementation were described. This chapter focusses on the result achievable using these descriptions. Our tests are conducted on two scenes that have been used in many global illumination solutions: the Cornell box scene and the Conference scene[13]. These scenes have been selected as the Cornell box is ideal for testing color bleeding while the Conference scene poses a challenge for visibility determination due to its detailed geometry. For the Cornell scene, a spotlight emitting a dim white light is placed on the center of the roof facing down. For the conference room scene the light source is placed outside of the room, mimicking an evening sun.

All of the results as presented in this section were gathered on a single machine, powered by an AMD Phenom II x4 955 processor, 12 GB's of RAM and the Windows 8.1 operating system. As described in section 4, the implementation is highly parameterized. Images were rendered using a 1600 x 900 resolution without AA.

5.1 Performance analysis & quality assessment

Although performance was not the main design criterion, the photon mapped ground truth is built upon the same architecture and therefore shares the same performance limitations. Although faster implementations of photon mapping exist (such as hardware accelerated versions [20] [21]), our method is built upon the framework as presented in chapter 4. Therefore, a direct comparison in terms of calculation time allows to roughly situate our algorithm.

A very important property of any indirect illumination algorithm is that it converges to a correct result when given a vast amount computation time. To verify our implementation, we use the photon mapping implementation with high quality settings to serve as a ground truth reference. The images are compared using a root-mean-square metric as defined in the

[13] Slightly adapted version (decimated models to lower the triangle count) from the conference scene as found on http://graphics.williams.edu/data/meshes.xml

following formula, where *i1* equals the ground truth image, *i2* equals the compared image, *w* and *h* equal the images dimensions:

$$error = \sqrt{\frac{1}{(w * h)} \sum_{n=0}^{w*h} (i1_n - i2_n)^2}$$

The error is calculated for every color channel (red, green and blue, ranging from 0 to 255) and is added together to find the final error value.

As both algorithms calculate the direct illumination separately, the time required to generate the G-buffers is discarded as the timings would be equal. The parameters for each algorithm (photon mapping, photon splatting with a small shadow map, photon splatting with a large shadow map and stratified photon splatting) are configured for optimal results. This means that it should render as fast as possible, with as little artefacts as possible. The algorithms require, besides processing power, certain amounts of memory in order to work. Resources such as the texture to store the indirect illumination are required for all algorithms and are therefore left out of the comparison as well. Furthermore, the few variables required to store parameters to control the algorithms are considered insignificant.

From a theoretical point of view, both algorithms require a runtime of $O(n)$ in order to find all photon locations. In the photon mapping algorithm, these photons represent the final illumination. Our algorithm sees these single photon locations as bundles of light rays, requiring less photons. Balancing the KD-tree used in photon mapping requires a $O(n \log_2 n)$ pass, but so does our stratification pass. Yet, the stratification pass was limited in our implementation to a single level, causing every voxel to be indexed exactly one time, therefore decreasing the effective runtime to $O(n)$. In a final step, the data structures have to be sampled. For photon mapping, locating m photons in a tree with a total of *n* photons is $O(m \log_2 n)$. For photon splatting, only the smaller list of photons has to be linearly traversed and splatted, but as a splat affects multiple pixels it is expensive nonetheless.

In terms of algorithm specific memory (see Table 1), the voxel grid technique is the most expensive one. Every voxel cell contains an averaged position, color, direction and count. Even if the count is zero, the memory has to be reserved. More memory-optimal implementations are possible, but the memory gain does not justify the large penalty in execution time. In order to converge to a plausible result, a lot of overlap is required, resulting in a high voxel count. Depending on the scene geometry, a high voxel count combined with the sparsely located geometry does not map well to our data structure, suggesting the use of a spatially more interesting structure such as a dynamic octree.

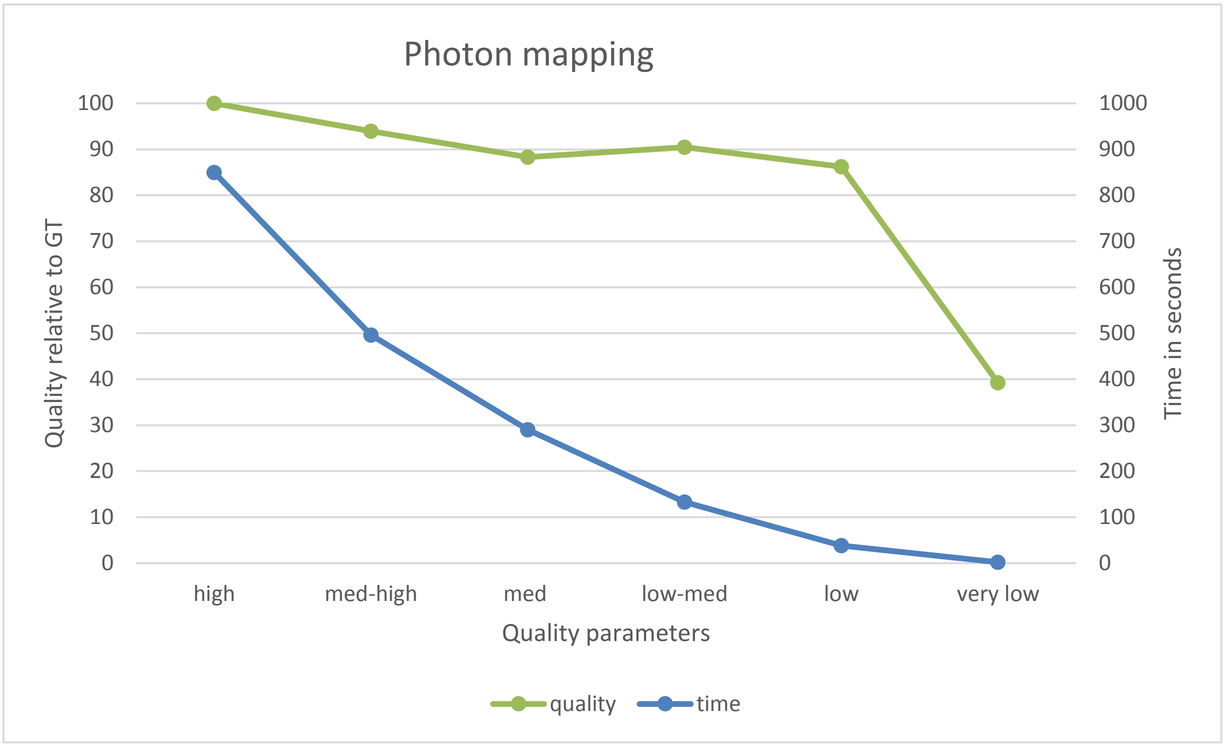
Table 1: Memory requirement for the algorithm specific data structures in the Cornell scene, using the highest quality parameters.

	Injection	Unit	Total memory
Photon mapping	~50000 photons	Color, position, direction (3x3 floats = 36 bytes)	~1.71 MB
Photon splatting Small shadow map (w/o stratification)	~25000 photons	Color, position, direction (3x3 floats = 36 bytes)	~0.85 MB
Photon splatting Large shadow map (w/o stratification)	~2500 photons	Color, position, direction (3x3 floats = 36 bytes)	~0.08MB
Photon splatting (with stratification)	256^3 voxels	Color, position, direction, count (3x3 floats + 1 integer = 40 bytes)	576 MB

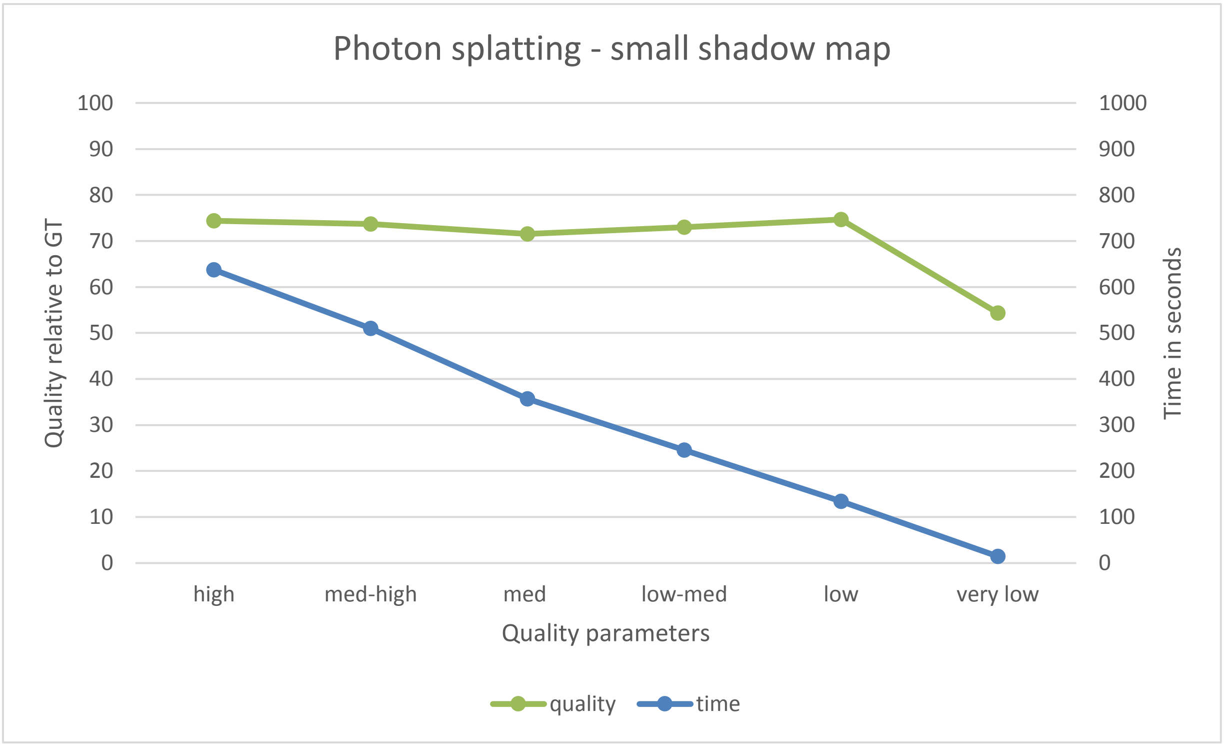
The detailed timings for the indirect illumination and error metrics can be found in Graph 1 to Graph 4. The parameters are adapted for high to low quality. In case of photon mapping, the number of photons and nearest neighbors is increased for higher quality results. For the non-stratified implementation the number of photons is increased. For the stratified implementation, the number of photons as well as the number of voxels is increased.

There is a logical trend visible between the quality and timings when comparing the splatting approaches to photon mapping. In general the quality is lower as well as the time required to compute the image. This is a logic consequence of our novel strategies. Bundling the light rays causes cheaper, but possibly incorrect visibility determination, balancing performance versus quality.

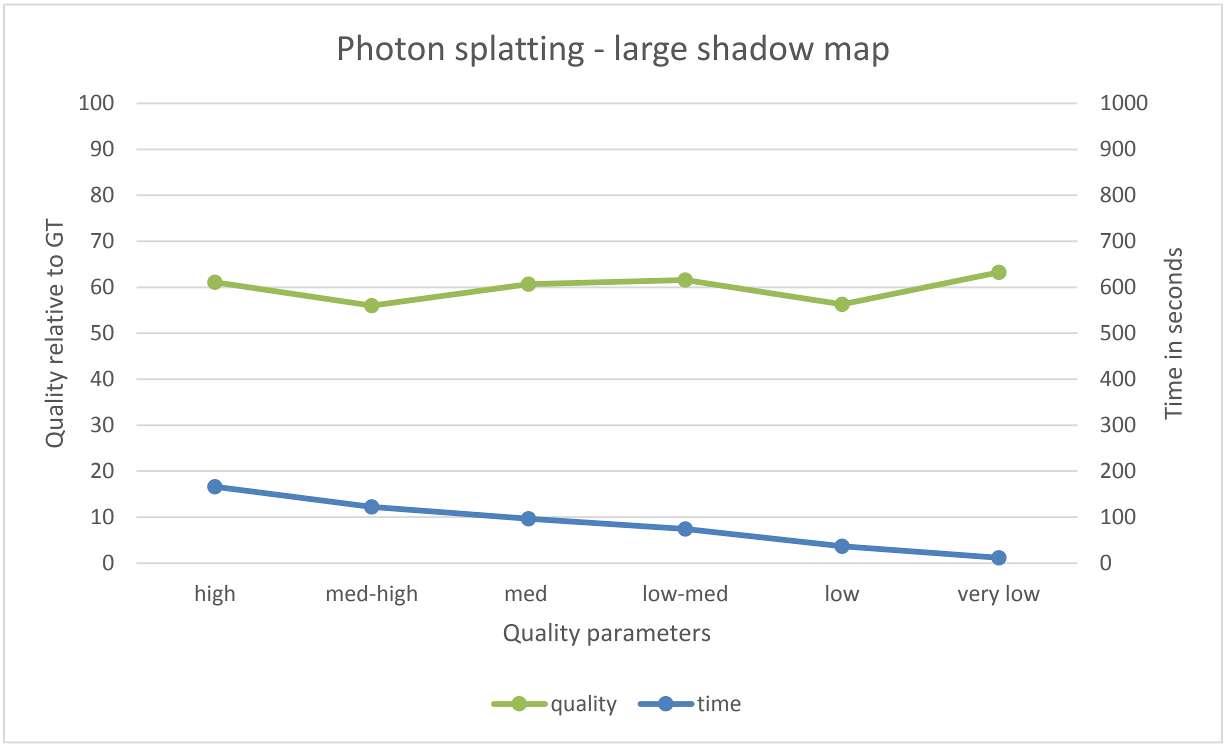
Keep in mind that two similar images might end up with a larger error metric when there is a slight difference in exposure in both images. Studies [22] show that a human perception test is more reliable, but such a test falls outside the scope of this thesis. Some images used to generate these metrics are provided in Figure 10, Figure 11 and Figure 12 to allow the reader to judge.



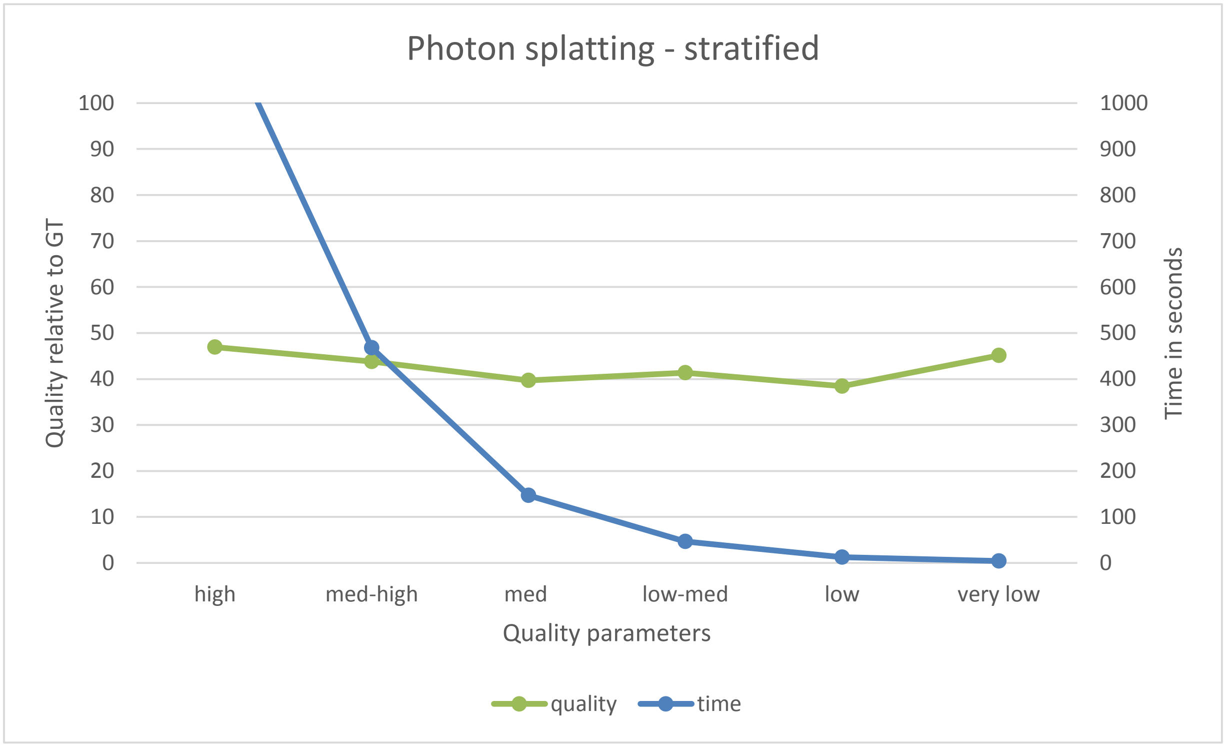
Graph 1: Influence of parameters on quality and timing to render a Cornell Box using Photon Mapping.



Graph 2: Influence of parameters on quality and timing to render a Cornell Box using Photon Splatting with a small shadow map size.



Graph 3: Influence of parameters on quality and timing to render a Cornell Box using Photon Splatting with a large shadow map size.



Graph 4: Influence of parameters on quality and timing to render a Cornell Box using Stratified Photon Splatting.

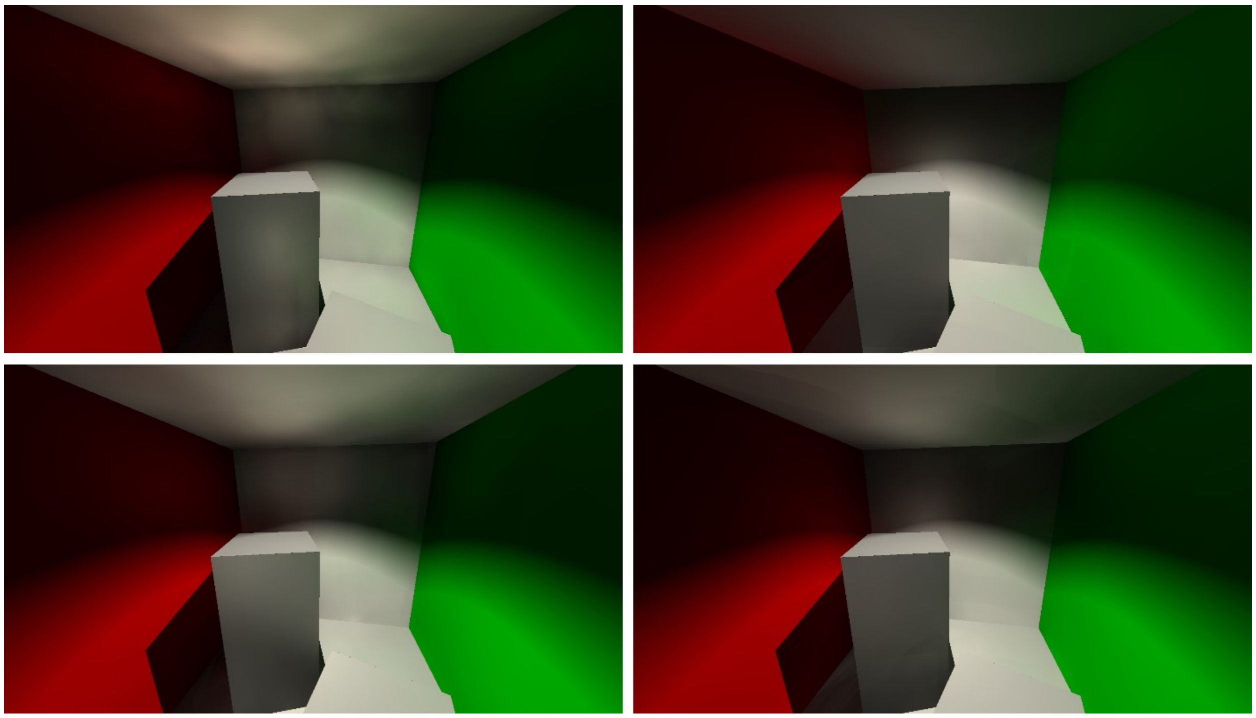


Figure 10: Low quality renders from the Cornell Box. Left top: Photon Mapping. Right top: Stratified Photon Splatting. Left Bottom: Photon Splatting using a small shadow map. Right bottom: Photon splatting using a large shadow map.

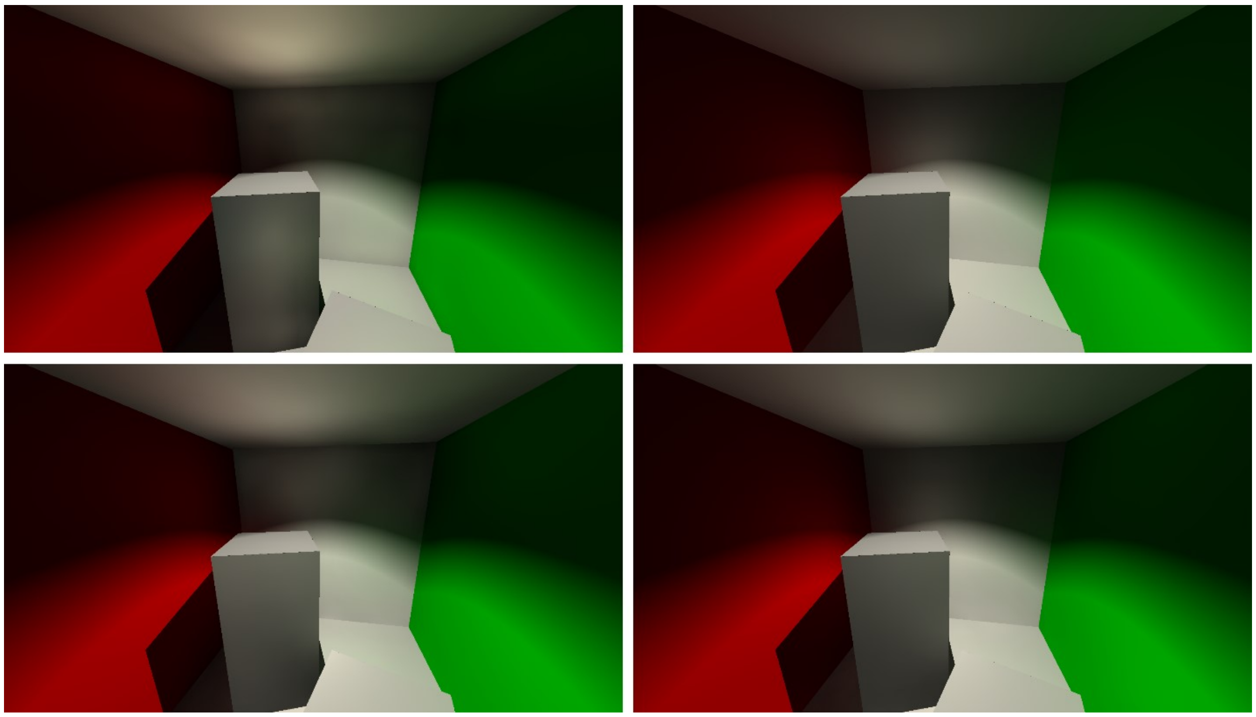


Figure 11: High quality renders from the Cornell Box. Left top: Photon Mapping. Right top: Stratified Photon Splatting. Left Bottom: Photon Splatting using a small shadow map. Right bottom: Photon splatting using a large shadow map.

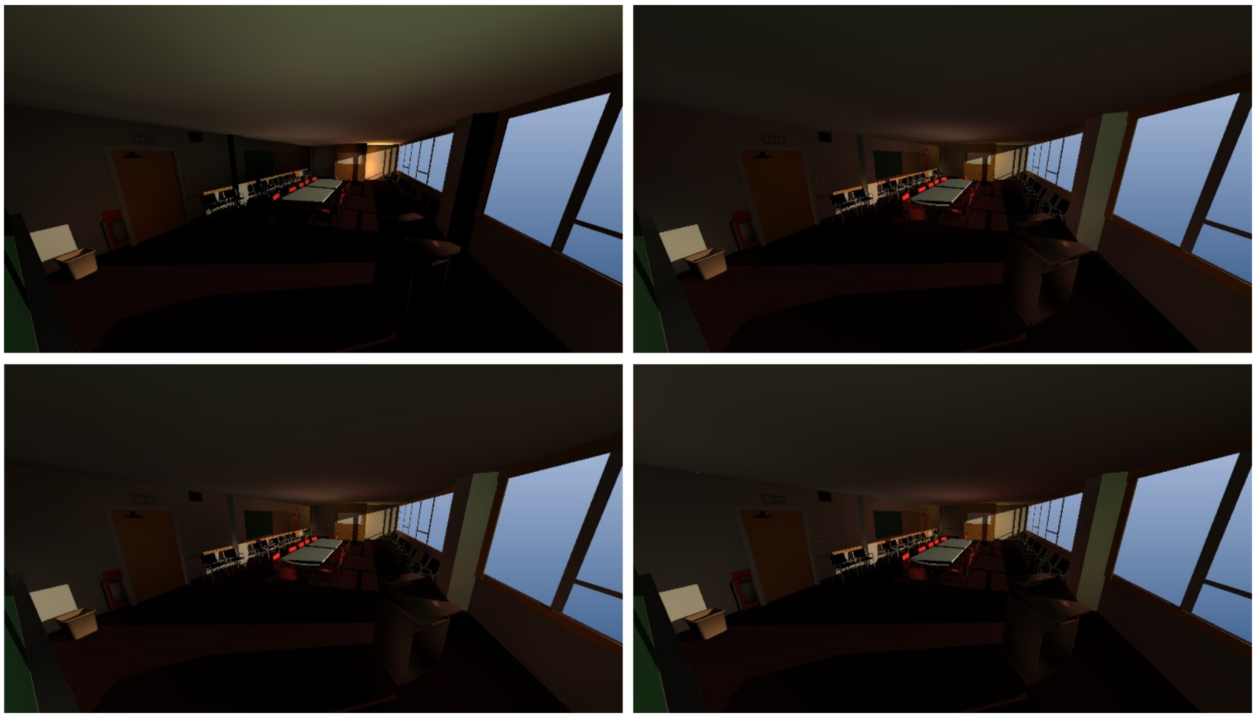


Figure 12: High quality renders from the Conference scene. Left top: Photon Mapping. Right top: Stratified Photon Splatting. Left Bottom: Photon Splatting using a small shadow map. Right bottom: Photon splatting using a large shadow map.

5.2 Common quality assessment criteria

Global illumination has some typical criteria that have to be met in order to render a plausible image. In this subsection, our approach as well as the photon mapped ground truth are assessed using these criteria.

Depth discontinuities

A property of global illumination is that depth discontinuities (such as grooves) often receive less illumination. Depending on the scene geometry and lighting, photons have a smaller chance of hitting these surfaces. This property is the main drive behind ambient occlusion techniques, and should be incorporated in any indirect illumination technique. Figure 13 shows a magnification of the corners of the Cornell scene where the corners are shaded slightly darker. It is noticeable that the stratified technique has this property a little less. This is caused by the requirement for a larger shadow map size, as is explained further in section 5.3.

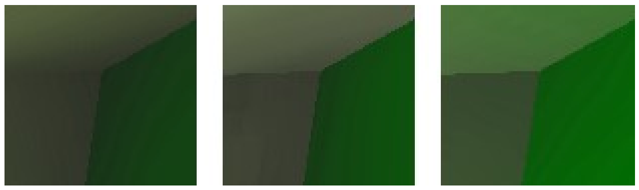


Figure 13: Magnified corners of the Cornell scene. From left to right: photon mapping, splatting and splatting with stratification. Brightness and saturation are slightly adapted in order visualize the differences more clearly.

Color bleeding

Surfaces that receive photons influence the color of the photon based on their material properties. If the photon is bounced on, following surfaces are influenced by that colored photon. This property is mainly visible where two differently colored materials meet, such as in corners. The roof and back wall displayed in Figure 13 both have a gray albedo color, but are shaded green due to the photons bounced from the nearby green wall.

Indirect shadows

A directly lit surface becomes an indirect area light source, casting soft shadows. As many of these generally overlap in a typical scene, the resulting shadow has a very soft appearance. The exit sign in Figure 14 is placed around 10 centimeters away from the wall. It is not illuminated by direct illumination, but it is illuminated indirectly. Every indirect light source in itself causes a sharp shadow, but the overlap and normalization of many splats causes a smooth final result.

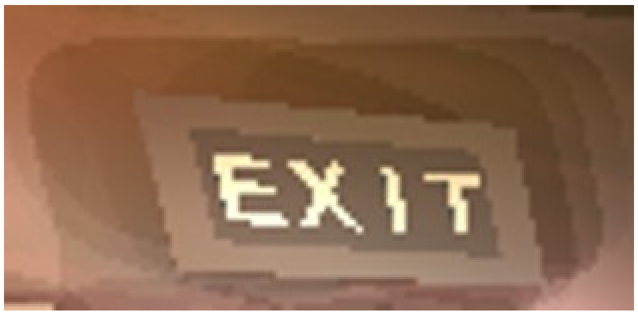


Figure 14: Multiple sharp shadows caused by indirect illumination. Screenshot taken before normalization to stress individual shadow maps.

Indirect intensity

Surfaces that receive illumination from multiple indirect light sources should be shaded brighter. As our algorithm has a normalization pass, all this information has to come from the evaluation of multiple shading passes rather than the density intensity as is the case with photon mapping. As stated in the algorithm description, the weights can be adapted in order to converge faster, but this causes the issue of introducing more noise between multiple splats. Figure 15 displays the intensity as seen on the roof of the Cornell box in Figure 11. The splatting technique approaches the ground truth while the stratified pass does not give its light discretization.

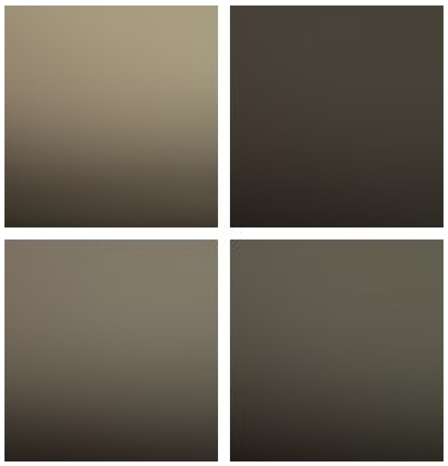


Figure 15: Brightness on the roof of the Cornell box. Left top: Photon Mapping. Right top: Stratified Photon Splatting. Left Bottom: Photon Splatting using a small shadow map. Right bottom: Photon splatting using a large shadow map.

5.3 Parameters

As explained in the theoretical and implementation sections, the algorithm depends on a number of control parameters. It is important to understand the influence of these parameters on the performance and quality on the final results. The main parameters for the visibility determination are analyzed in this sub-section.

The main difficulty to determine proper values for these parameters is that they are scene dependent. For a simple, low geometry scene such as the Cornell box, a low shadow map resolution will not cause any issues while the same resolution might introduce noticeable artifacts on the detailed geometry in the Conference scene. Therefore, these parameters are assessed using both scenes to cover various scenarios.

5.3.1 Visibility determination

The three main parameters in the visibility determination are the shadow mapping size, the shadow mapping resolution and the number of virtual light sources.

Recall from the visibility explanation (see section 3.3) that a greater shadow map size will cause more overlap and therefore converge faster. This works great with large open surfaces. However, artifacts may occur when a ray skims close to geometry, but does not intersect with it. Figure 16 displays such a ray. From the first point of intersection, a small step back is taken and the shadow frustum (red box) is calculated. However, some points in this frustum will be considered visible, albeit they might not have been visible when a regular shadow map would have been used (as is the case with the green ray), causing light leaks.

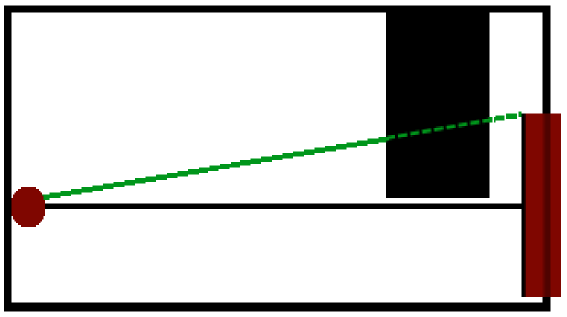


Figure 16: Cause of artifacts when a too large shadow map is used.

The Cornell box scene has two boxes close to the walls, making it a model example. By decreasing the number of photons and multiplying the shadow map size by factor three, the result as visualized in Figure 17 becomes visible. The light leaking causes the area behind the box to become overly bright.

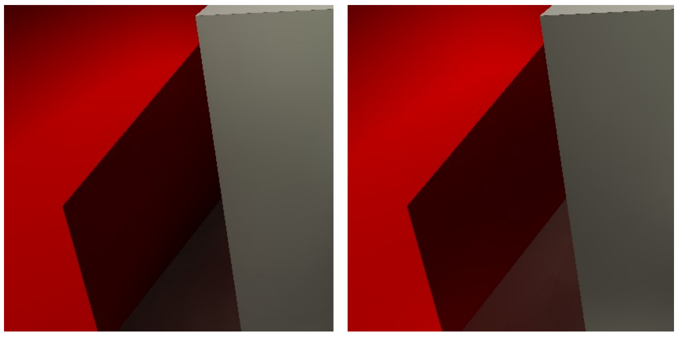


Figure 17: SM Artifacts. Right image has a 3x larger shadow map than the left image. Both use the same number of splatted photons.

In order to converge, overlap is required due to the Monte Carlo nature of our algorithm. The more photons that are shot and the larger the shadow map size, the faster the algorithm converges. In other words: for a higher quality image, more overlap is required. On the other hand, increasing the shadow map size might introduce artifacts while increasing the number of photons linearly increases the rendering time. The number of overlap for the high quality Cornell box with small shadow map (as seen in Figure 11) is displayed in Figure 18.

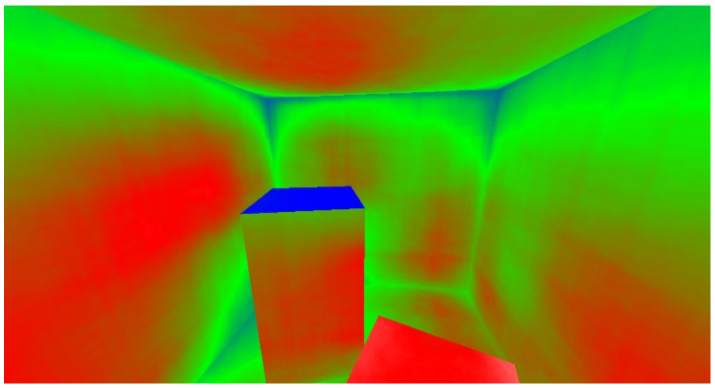


Figure 18: Overlap in Cornell scene. Heatmap color range from blue (no overlap) to green (200x overlap) to red (400x overlap) to white (600x overlap).

The final visibility setting is the shadow mapping resolution. The imperfect shadow mapping technique [5] shows that very low resolution shadow maps give plausible results, even for hemisphere point light sources. Since our algorithm calculates visibility in small orthographic volume, an extremely low shadow map resolution can be used. However, when thin geometry comes into play, indirect shadows might be off when using such a low resolution. Furthermore, when the shadow map area increases, the shadow map resolution should increase linearly in order to keep the same level of detail.

Figure 19 displays the difference in shadow map resolution for detailed geometry. In the left image, a low resolution shadow map is used and the shadows behind the chairs are barely visible. When increasing the resolution, the shadows behind the chairs gain much more detail. Since we are dealing with thousands of light sources, even a slight increase in resolution causes a large impact on the rendering times.

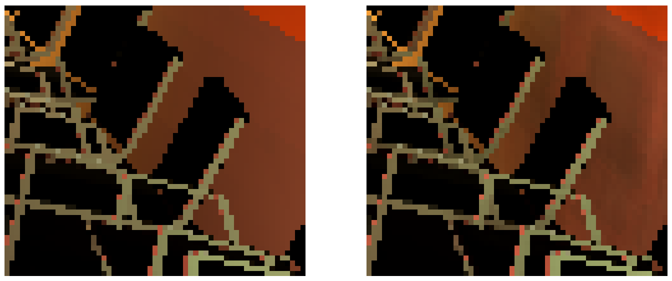


Figure 19: shadow map resolution influence on indirect shadows. Left: 4x4 shadow maps, right 32x32 shadow maps.

The optimal settings depend on the scene and the required quality. In general, the overlap (so shadow mapping size) should be maximized with as little photons as possible. The shadow mapping resolution should be set to the absolute minimum, where the minimum depends on the scene geometry and accepted quality.

5.3.2 Stratification

The three main parameters in the stratification pass are the number of photons that are shot into the scene, the number of voxels and the number of photons per bin.

As seen in Table 1, the memory required for the number of voxels increases cubically. This means that the memory limit is quickly reached and the number of voxels should be kept as low as possible. The required number of voxels depends on the scene size and detail. Larger scenes require more detail closer to the camera, suggesting a grid-like implementation of the stratification pass. Based on the distance away from the viewer, larger cells can be constructed, similar to the setup of cascaded light propagation volumes [17].

The number of photons per voxel depends on the bucket size and the number of photons injected into the scene. The color of every photon is combined and splatted in a single pass per voxel. Using a too high photon or bucket count will cause large disparities between voxels, causing artifacts in the overlap. Therefore, using a very high photon count will result in artefacts as visualized in Figure 20. As a rule of thumb, a similar number of photons as would be used in a non-stratified approach can be applied.

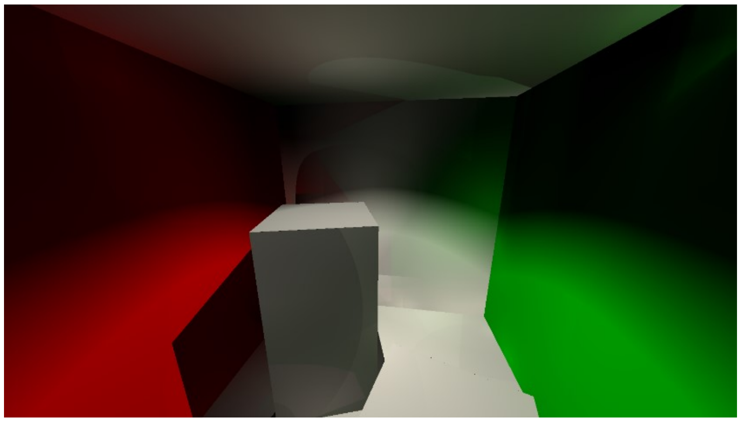


Figure 20: Artefacts when using high photon counts per voxel.

6 Conclusion & future work

In this work, we have presented a novel visibility determination technique and virtual light source stratification strategy. Our techniques are capable of rendering plausible indirect illumination with a reasonable amount of resources, but unfortunately stay far away from real-time frame rates. However, during the development of our algorithm, hardware compatibility was one of our main concerns. This means that large parts of the pipeline can be implemented using hardware acceleration and state of the art techniques. These techniques, such as imperfect shadow maps and tile based deferred rendering, are capable of dealing with visibility determination and shading of huge amounts of light sources. Furthermore, the existing stratification pass could be extended with common real-time improvements, such as a cascaded data structure, significantly lowering memory requirements. Additionally, multiple light sources could be stored per voxel, mimicking point light sources.

Besides from the speed of the algorithm, our results show that the final renders, even when given a large amount of time, still slightly diverge from the ground truth. The largest problem situates around intensities when using stratification. This suggests that the used formulas to calculate the properties of the stratified lights sources are not yet on-par. Further experimentation with these parameters can be done in order to approach ground truth even better. Furthermore, the entire algorithm is highly parameterized. More studies, perhaps using a mixed audience as reviewers, can define optimal settings between visual perception and performance.

Finally our research shows that there are still potential trajectories within the vast field of global illumination. Visibility determination is one of the most important factors of the rendering equation and is often neglected in current state-of-the-art real-time algorithms due to its expensive nature. Our method for bundling light in a single ray creates a cheaper visibility test with reasonable results. This ray casting method of finding the closest illumination could be applied to hemisphere point lights instead of photons as well, in order to compensate for the high cost to determine visibility for such light sources.

7 References

[1] Eric Tabellion and Arnauld Lamorlette, "An approximate global illumination system for computer generated films," *Proceeding SIGGRAPH '04 ACM SIGGRAPH 2004 Papers* , pp. 469-476, 2004.

[2] Michael Deering, Stephanie Winner, Bic Schediwy, Chris Duffy, and Neil Hunt, "The triangle processor and normal vector shader: a VLSI system for high performance graphics," *SIGGRAPH '88 Proceedings of the 15th annual conference on Computer graphics and interactive techniques* , pp. 21-30, August 1988.

[3] Henrik Wann Jensen, "Advanced global illumination using photon mapping," *Proceedings of the Seventh Eurographics Workshop on Rendering*, pp. 21-30, 1996.

[4] Carsten Dachsbacher and Marc Stamminger, "Reflective shadow maps," *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pp. 203-231, 2005.

[5] Tobias Ritschel et al., "Imperfect shadow maps for efficient computation of indirect illumination," *SIGGRAPH Asia '08*, vol. 27, no. 5, p. no 129, 2008.

[6] Martin Mittring, "Finding next gen: CryEngine 2," *ACM SIGGRAPH 2007 courses*, pp. 97-121, 2007.

[7] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov, "Image-space horizon-based ambient occlusion," *ACM SIGGRAPH 2008 talks*, p. Article No. 22 , 2008.

[8] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile, "Modeling the interaction of light between diffuse surfaces," *SIGGRAPH '84 Proceedings*

of the 11th annual conference on Computer graphics and interactive techniques, vol. 18, no. 3, pp. 213-222, 1984.

[9] Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann, "Interactive Indirect Illumination Using Voxel Cone Tracing," *Computer Graphics Forum (Proceedings of Pacific Graphics 2011)*, vol. 30, no. 7, September 2011. [Online]. http://maverick.inria.fr/Publications/2011/CNSGE11b

[10] James T. Kajiya, "The rendering equation," *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pp. 143-150, 1986.

[11] Robert L. Cook, Thomas Porter, and Loren Carpenter, "Distributed Ray Tracing," *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, no. SIGGRAPH '84, pp. 137-145, 1984.

[12] Jaco Bikker, "Ray Tracing for Real-Time Games," Delft, 2012.

[13] Arthur Appel, "Some techniques for shading machine renderings of solids," *Proceedings of the April 30 - May 2, 1968, spring joint computer conference*, pp. 37-45, 1968.

[14] Edwin Catmull, "A subdivision algorithm for computer display of curved surfaces," Utah, 1974.

[15] Carsten Dashsbacher and Marc Stamminger, "Splatting indirect illumination," *I3D '06* , pp. 93-100, 2006.

[16] Alexander Keller, "Instant radiosity," *SIGGRAPH '97*, pp. 49-56, 1997.

[17] Anton Kaplanyan and Carsten Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pp. 99-107, 2010.

[18] Blanco-Claraco Jose-Luis. C++ header-only fork of the FLANN library for KD-trees. [Online]. https://github.com/jlblancoc/nanoflann

[19] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo, "Real-time KD-tree construction on graphics hardware," *ACM SIGGRAPH Asia 2008 papers*, vol. 27, no. 5, p. Article No. 126 , 2008.

[20] Timothey J Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan, "Photon mapping on programmable graphics hardware," *SIGGRAPH '05 ACM SIGGRAPH 2005 Courses*, p. Article No. 258, 258.

[21] Michael Mara, David Luebke, and Morgan McGuire, "Toward practical real-time photon mapping: efficient GPU density estimation," *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 71-78, 2013.

[22] Martin Čadík, Robert Herzog, Rafał Mantiuk, Karol Myszkowski, and Hans-Peter Seidel, "New Measurements Reveal Weaknesses of Image Quality Metrics in Evaluating Graphics Artifacts," *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, vol. 31, no. 6, pp. 1-10, 2012.