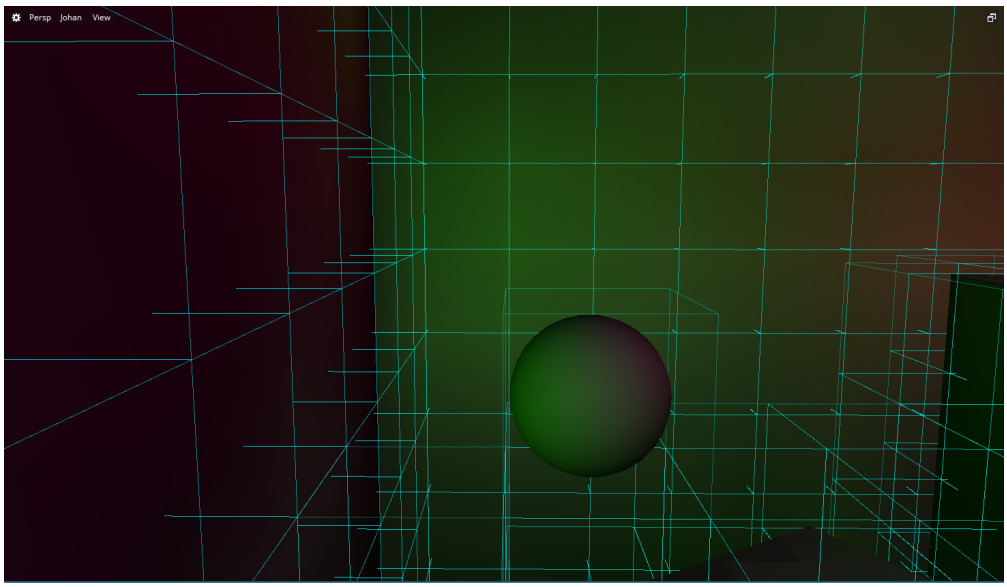


Global illumination for static and dynamic objects using light probes

Master's thesis in Computer Science: Algorithms, Languages, and Logic

JOHAN BOWALD

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis

Global illumination for static and dynamic objects using light probes

Master's thesis in Computer Science: Algorithms, Languages, and Logic

JOHAN BOWALD, johan@bowald.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

Global illumination for static and dynamic objects
using light probes
Master's thesis in Computer Science: Algorithms, Languages, and Logic
JOHAN BOWALD

Examiner: Ulf Assarsson, Department of Computer Science and Engineering

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

Cover: Cornell box scene with only indirect light, using the implemented method with a resolution of 16 voxls. The sphere is a dynamic object. The teal boxes indicates light volumes.

Typeset in LaTeX
Gothenburg, Sweden 2016

Global illumination for static and dynamic objects
using light probes
Master's thesis in Computer Science: Algorithms, Languages, and Logic
JOHAN BOWALD
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

Today, real-time 3D applications commonly use two separate techniques for global illumination of static and dynamic objects. Using two different systems such as lightmaps and point clouds (probe-based solutions) can lead to different appearances depending on the type of object. The purpose of this thesis project has been to research and develop a data structure used to sample indirect light for both static and dynamic objects. Integrating the indirect light data for static objects in a probe-based solution has additional advantages over lightmaps. For example, an artist will be able to apply small changes to a level and instantly have an adequate perception of the global illumination. A data structure has been implemented in Autodesk's game engine, Stingray, and tested on three different scenes. The data structure uses symterical probe placement, a method to filter out occluded probes, and 3^{rd} order spherical harmonics as transfer basis. The data structure can be run in real time while being memory efficient. However, visual artifacts arise for complex geometry, such as spheres. Even if this project did not achieve the visual quality strived for, its findings can be used when developing a better solution.

Keywords: Real-time global illumination, light probes, indirect light, static and dynamic objects.

Acknowledgements

I would first like to thank Anders Lindqvist, Sr Software engineer at Autodesk, who has been my supervisor at Autodesk during this project. He has been an inspiration, always offered help with understanding the source code of the game engine and enouraged me to test different methods to overcome problems during the research.

I also would like to acknowledge Magnus Pettersson, who made it possible to do this thesis project at Autodesk and helped me with the proposal. I also like to thank him for being my supervisor before Autodesk's re-organization.

Thanks to all the employees at Autodesk for making me feel welcome.

Special thanks to Amanda Nilsson for proofreading this thesis, she has provided very valuable comments on both report structure and grammar.

Thanks to Adam Sandberg Ericsson for his opposition on this thesis.

Lastly, I would like to thank Professor Ulf Assarsson at the department of Computer Science and Engineering, for being the examiner of this thesis and also for helping me to get in contact with Autodesk in the first place.

Johan Bowald, Gothenburg, June 2016

Contents

1
Introduction

1.1 Background

Indirect lighting is commonly used in video games. Game studios strive to add as many effects as possible in a game and therefore require that each effect can be rendered in line with a time budget. Reaching at least 30 *frames per second* (FPS) on limited hardware (consoles) is a requirement. Therefore, the focus of the algorithms generating visual effects tends to be on optimizing rendering time while achieving satisfying visual results. A drawback of these algorithms can be visual artifacts, which graphical artists can remediate during level or model design by changing the geometry. This is a time consuming procedure.

Real-time indirect lighting is not only used in games. An example of another graphical application is *Computer-aided design* (CAD) tools, which are used in industrial design and architecture to create prototypes. In these applications, the final image is rendered using a non-real-time renderer such as a ray tracer where the rendering time depends on the desired final quality. The purpose of real-time rendering in these applications is to give an approximation before performing the ray tracing. To have a good approximation, indirect lighting needs to be considered. Unlike in the gaming industry, tweaking either geometry or textures to avoid visual artifacts is not possible.

This thesis covers the development of a data structure for sampling pre-calculated indirect light in real time. The main priority has been to reduce the number of major visual artifacts, regardless of the geometry in the scene, and to minimize the amount of human interaction. The proposed solution is independent of meshes and uv-maps. Minimizing the rendering time has been the second priority with the intention to reach real time. The data structure contains light probes, automatically placed covering both static and dynamic objects in the scene.

The knowledge gained from this research can be applied in cases where there is a need for generating an adequate image without manual interaction. An example can be gaming applications, where instead of sampling indirect light from light maps, the algorithm presented in this report can be used.

1.2 Problem definition

Indirect lighting is a crucial component to reach realistic visuals. There exist numerous approaches to render indirect light in real time. Two main categories are UV-based methods and light-probe interpolation.

UV-based methods use pre-calculate irradiance values from 2D textures, and map them to the objects in the scene using UV-coordinates (Akenine-Möller et al., 2008, 9.9 Precomputed Lighting), often referred to as *Lightmapping*. Even though lightmaps are memory efficient, they are lacking in terms of artifact seams. These seams are introduced when unwrapping a 3D model's surfaces to fit in a 2D texture. The unwrapping procedure is often automatic. To minimize seams, the transformation can be manually tweaked but this procedure is time consuming.

Light-probe interpolation is performed by calculating the indirect light for certain positions in the scene and using this data to interpolate the result for either a model or a surface (Greger, Shirley, Hubbard, & Greenberg, 1998; Sukys, 2014; Bentley, 2014; Cupisz, 2012). These light-probe techniques are mainly used for dynamic objects and share a drawback from light-leakage artifacts due to interpolation errors. There is an interest of finding a method which is not dependent on UV-maps, is fast enough to be evaluated in real time, can be applied to both static and dynamic objects, and contains few to no major visual artifacts.

To find a solution to this problem, it has been divided into two subproblems: placing light probes in the scene and associating surface fragments with these probes. These subproblems are described in more detail in the following two sections.

1.2.1 Associating surface fragments with light probes

Each visible surface fragment in a scene needs to be shaded with indirect lighting. The indirect light is determined by interpolating data from the probes associated with a surface fragment. This problem consist of two parts, extracting the data fast enough for real time and filter out occluded probes such that no light leakage occurs.

The first part of the problem can be defined as: *Given a world coordinate, traverse the data structure to identify which volume the coordinate corresponds to and obtain illumination data from the light probes defining that volume.* The solution must provide a data structure that is fast enough to allow the application to run in real time.

The second part of the problem can be defined as: *Given a set of probes placed across the scene, associate surface fragments with a number of probes, such that erroneous probes do not contribute to the result, yet provide enough data for each surface fragment.* Assuming that the amount of probes needed to cover the scene exceeds the memory capacity of the GPU, there is also a need for the data structure to be streamable. The quality of the solution is evaluated based on the number of major visual artifacts, the memory complexity and render time.

1.2.2 Placement of probes

The amount of probes affects rendering time, memory requirements and light detail. By having few probes, the data structure requires less memory and can potentially be faster. However, having more probes increases both details in indirect light and the possibility to avoid artifacts.

This problem can be defined as: *Minimize the amount of probes needed, yet cover*

the entire scene and provide enough data to avoid visual artifacts. Probe placement and minimizations of probes can be done during an offline stage and not during runtime. The quality of a solution to this problem is evaluated based on memory efficiency.

1.3 Limitations

The implementation of the project has been done in Autodesk's game engine *Stingray*. Stingray supports both gaming consoles and personal computers (PC). This project is coded with PC in mind, however, the usage of functions from DirectX 11.0 API is limited to functions supported by Stingray.

Finding an optimal generation of irradiance data for the probes is not considered during this research. Instead the existing path tracer in Stingray is modified to generate the data.

This research project considers only low-frequent indirect light when generating the data for the probes. Even if it is possible to generate high-frequent light, such as caustic effects, the detail will depend on the resolutions of volumes defined by the light probes. For example large volumes will lose high-frequency details during the interpolation, due to small sample size. To allow varying volume sizes, high-frequent light will not be considered.

1.4 Method

During the early stages of the project different light probe techniques where researched in order to find a solution to the problems defined in Section 1.2.2 and 1.2.1. Also, different data structures where studied to allow the data to be queried efficiently.

To get aquinted with the game engine Stingray, a part of the project was dedicated to learning the code base by implementing small examples releated to the reaserch. When enough knowledge had been gained about the game engine, a simple data structure was implemented to test the performance in real time. Radiance used for the probes was first random color values and later hardcoded directional radiance data. When the data structure could be queried in real time, a solution to find occluded probes was implemented. To test the accuracy of the implementation, the probes needed to contain radiance values based on their position in the scene. During the next iteration of the project, generation of radiance for the probes was implemented by modifying Stingrays path tracer. When the probes had been assigned correct radiance values, artifacts generated by the probe occlusion algorithm were found and refinments of the algorithm were sought. Due to time constraints other strategies to solve light leakage were not implemented. When the implementation was considered finished, tests with three different scenes were conducted and performance, memory efficiency and visual quality was evaluated.

1.5 Outline

The thesis is divided into five chapters. First, the introduction where the background, problem definition, limitation and method is provided. The second chapter, Previous work, offers information related to the project. Some techniques described are used within the project, others are techniques which solve similar problems. The third chapter, Theory and implementation, contains descriptions of the algorithms implemented in the data structure. The first section of this chapter describes the pre-calculation phase and the second section describes the run-time phase. The fourth chapter, Results, provides data on memory complexity, run-time performance and visual examples for the three test scenes. The fifth chapter, Discussion, offers an analysis of advantages and disadvantages of the implementation. The implementation is compared to other methods described in the previous work chapter. This chapter also contains a future work section, with suggestions on improvements of the implemented data structure.

2
Previous Work

This chapter presents previous work connected to this thesis. The two first sections describe similar methods and approaches to sample indirect lighting using light probes, while the last one presents a relevant technique used in the data structure.

2.1 Indirect lighting in Killzone 3

Valient (2014) presented during Game Developers Conference 14 a prototype method used to achieve indirect lighting in Killzone 3 (Guerrilla Games, 2011). The method is applied to both static and dynamic object and calculated per-pixel. The purpose of the approach was to replace light mapping for static objects and use the same system for both dynamic and static objects. By using the same lighting solution for both types of objects, differences in lighting between the types would be at minimum. An advantage with this approach was that even if some changes were applied to the geometry in the scene, the lighting would still be adequate. In comparison, a UV-based approach would not be this robust, due to the indirect lighting here being based on the meshes in the scene and therefore needing to be re-rendered. The method presented consisted of two parts. A pre-calculation phase to build the data structure and a runtime phase to query the irradiance from the data structure per-pixel.

2.1.1 Probe placement

To build the data structure, probes need to be placed in the scene. The first step in placing the probes is to voxelize the scene. The diameter of a voxel is 1m in gameplay areas and up to 10m in distant areas of the level. Light probes are placed in empty voxels next to voxels containing geometry. Volumes are defined by Delaunay tetrahedralization (Cupisz, 2012). Delaunay tetrahedralization divides the scene into tetrahedrons based on the probes' positions, such that the globally minimal angle between probes are maximized (the tetrahedralization can be compared to triangulation in $2D$). This associates four light probes with a tetrahedron, one at each vertex.

A drawback of this probe placement technique was that long and sliver-like tetrahedrons could be formed. To counter this problem, filler probes were added in empty spaces at a lower frequency than the original probes. This created more regularly shaped tetrahedrons, covering more space above the geometry, which dynamic objects could use to sample indirect lighting.

2.1.2 Querying the data structure

A level using the light probe system can consist of up to a few hundred thousand probes. The method is applied per-pixel, meaning that for each pixel in each frame all light probes need to be traversed to find the corresponding irradiance value. The tetrahedrons are partitioned into a sparse grid, where each cell is $16m^3$. When using a perfect hashmap for the sparse grid, the per-pixel search can be speeded up by only searching within a cell for the corresponding tetrahedron. Furthermore, the tetrahedrons are stored in a *Binary space partitioning* (BSP) tree per grid cell, which also speeds up the search time. When the corresponding tetrahedron is found, the surface fragment is shaded using interpolation between the probes that are not occluded.

2.1.3 Occlusion of erroneous probes

When performing the interpolation between different probes in a volume, it is important to filter out probes which are occluded, otherwise light leakage artifacts occur. To solve the problem, additional data is stored in each tetrahedron. The data consist of either a number of *occlusion shadow maps* or *occlusion splitting planes*. A splitting plane is used when geometry in the volume is simple, such that the volume is split along a certain axis by the geometry. Since there are three axes, up to three occlusion splitting planes can be stored per tetrahedron. If geometry inside the tetrahedron is more complex, one occlusion shadow map per probe can be used. The triangle plane on the opposite side of the probe is tessellated such that 16 sub triangles are generated. For each subsample the distance to the closest geometry is stored, the value being proportional to the length of the tetrahedron. Using these two data sets, light leakage can be decreased to an acceptable degree.

2.2 Precomputed Radiance Transfer probes

During the thesis work a presentation of a method for solving a similar problem was held at Game Developers Conference 16. Stefanov (2016) presented the method used to achieve Global Illumination (GI) effects in *Tom Clancy's The Division* (Ubisoft Massive, 2016). The method is called *Precomputed Radiance Transfer probes* (PRT). Since Tom Clancy's The Division takes place in a large open world, where the environment contains a large amount of objects, a light map solution was not feasible due to memory constraints. This approach also updated the irradiance dynamically to attain different types of lighting throughout the day.

2.2.1 Probe placement

The probe placement is automated using two different methods: using a ray casting grid and placement along building walls. The ray casting method uses a grid with 4m spacing to place the probes. Rays are fired in a top-down manner from the grid and a probe is spawned at each intersection. If a probe should intersect geometry, the probe is slightly moved such that it does not intersect an object. The game

is set in Manhattan and contains large buildings. The walls need to be covered with probes due to the varying sky visibility as seen from the streets. Probes are therefore generated along the walls of tall buildings so that they can capture radiance transitions.

2.2.2 Generation of radiance for probes

Each probe contains a list of surfels which are visible to that probe. A surfel is a surface element, associated with a number of parameters, where the position and the normal are important when calculating radiance values for the probes. The list of surfels can be compared to a G-buffer cube map, or to firing a lot of rays and storing the intersected surfaces. Each direction missing a surfel is treated as visible sky, resulting in a spherical shadow term for the sky. The idea is that radiance from each surfel for a probe is stored as an irradiance component for that probe. The irradiance from the sky is scaled by the sky visibility. To avoid light leakage for the sky values, the shadow map of the sun is used to tweak the sky radiance values such that objects in the shadows are not illuminated. The irradiance for a probe is stored using the *Half life 2 ambient cube* as a transfer basis (Mitchell, McTaggart, & Green, 2006). The HL2 ambient cube is basically 6 axis aligned vectors each associated with a light value.

To be able to generate probe radiance during runtime a data structure and a number of optimizations are presented. First, the probes are divided into a sector grid where each cell has the size of 64m^2. A cell can at most contain 1000 probes, however, in the game a cell typically contains 200-300 probes. The sectors are streamed in and out depending on the position of the player, where a maximum of sectors on the GPU is set to 25. Secondly, the surfels are clustered in a two-level hash grid. The first grid level offers the positions, normals and albedo for the surfels associated with it. A cell's size is 1m^3 and only treats surfels with roughly the same normal, i.e a number of cells can be positioned at the same place if the geometry inside is complex, however, at most six cells in the same position(two directions for each axis). Furthermore, cells containing surfels facing same direction are combined into an irradiance brick, the second grid level. The brick has a size of 4m^3. Bricks are associated with an array of weights, which determines how much irradiance a probe receives from a particular brick. Thirdly, the surfel data is shared between the probes, such that when radiance is generated for a specific surfel it does not need to be recomputed for each probe that sees it. The light for each surfel is computed using the albedo and normal in a similar way as a deferred lighting pass (Deering et al., 1988). To achieve multiple bounces, each surfel is associated with the closest probe and it can sample irradiance from the previous frame. The irradiance value is used as the ambient light term for that surfel.

2.2.3 Querying the data structure

To query irradiance for fragments in the scene, a volumetric grid is used. The grid consists of $32 \times 16 \times 32$ voxels per axis, spanning $100 \times 50 \times 100$m in the game world. The grid is aligned with the player's camera, such that it always follows the view

of the player. For each voxel, the closest probe is chosen as that voxel's radiance value and geometry in the scene is shaded by trilinear interpolation of the voxels. Geometry that is not covered by the volumetric grid, i.e is further away than 100m in the z-direction (into the scene), are shaded using a fallback solution. The fallback solution consists of a 2D texture, where the texels are treated as sector probes, i.e in the same way as a probe in a voxel. These texels only contain direct illumination from the sky. The geometry outside the volumetric grid samples those values.

2.2.4 Occlusion of erroneous probes

The approach occludes irradiance in two different ways. Filtering out probes placed on the outside from probes on the inside and distinguishing between probes placed in different rooms. The latter is used to avoid light leakage between walls. The other is partly used to avoid light leakage but also used to make sure skylight does not bleed through walls. By storing volumes, defined by probes, which are either placed on the inside or outside in two different textures, a stencil buffer containing only the building model can be used to decide whether the surfel is located inside or outside the building. Surfels on the inside only load irradiance from the inside texture and vice versa. To avoid light leakage between rooms inside a buidling, i.e located in the same texture, Axis aligned bounding boxes (AABB) matching the extent of a room are used. Surfel reads are clamped using the AABBs so only probes inside the same rooms are used.

2.2.5 Performance

The PRT solution is able to render a frame in ca 0.95ms on Xbox one and ca 0.47ms on a PC using GTX 760 card. These are representative values, due to the exact timings depending on the number of probes in screen space. During this time, the radiance values of the probes in two sectors are also updated. The one which the player is in and a sector chosen at random which is loaded on the GPU.

2.3 Spherical harmonics, compact representation of directed light data

A light probe needs to have pre-calculated radiance values. These values can not be stored for only a particular normal, but need to be stored for all directions over a sphere. This can be seen as a spherical function. To express this in a compact way, the values can be stored as coefficients of a transfer basis and be reconstructed during runtime. For this project a 3^{rd} order Spherical Harmonics has been chosen as a transfer basis.

Spherical Harmonics is the solution to Laplace's equation using spherical coordinates (Sloan, 2008). It is defined as a basis function based on Legendre polynomials, meaning that Spherical Harmonics describe an infinitely large set of scaled functions which, when summed, can describe any spherical shape. In computer graphics an

approximation of the light is stored by truncating the sum of functions to only contain either the second order of coefficients or the third order. In this project 3^{rd} order Spherical Harmonics has been used to provide a sufficient approximation, meaning that 9 floating point values are enough to store directional radiance for a color.

Spherical Harmonics have a series of mathematical features. The Spherical Harmonics approximations are invariant for rotation, meaning that rotating the encoded values is the same as rotating the original values. Spherical Harmonics approximations are also additive, meaning that encoded radiance can be added together without decoding.

3
Theory and implementation

This chapter covers the theory and implementation to solve the different problems stated in the problem definition, Section 1.2. An acceptable solution provides the ability to render an image with minimum light leakage and a decent percepted light correctness, while being able to run in real time. The implemented solution is called a data structure and consists of various different algorithms described in each following section. The data structure is created and initialized during a pre-calculation phase and is used during a continuous phase which is active during the runtime of the application. The parts of these stages are described in more detail in each subsection. The implemented solution takes advantage of the symmetric resolution of equally sized cube shaped volumes, both for creating a perfect hash for accelerating the data structure and interpolating the result.

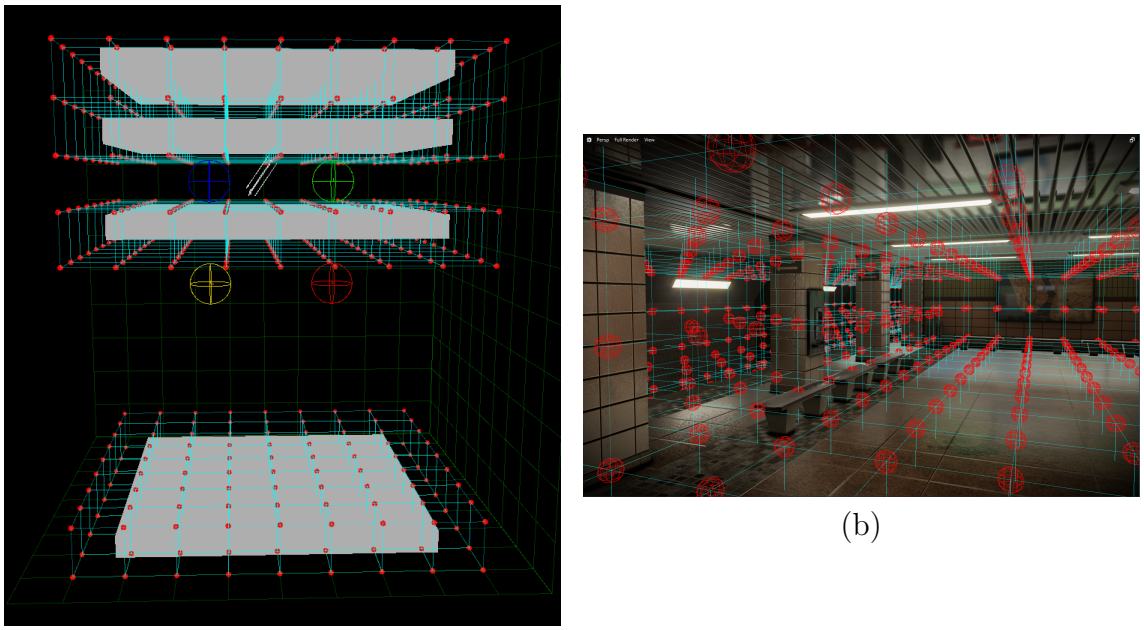
3.1 Pre-calculation phase

This phase refers to calculations that are performed before running the application. During this phase the generation of radiance and placement of probes are performed and stored in an acceleration structure. No optimization of runtime for this step is needed.

3.1.1 Dividing the scene into volumes for probe placement

The entire scene is divided into a number of volumes. A volume is defined as the space spanned by a set of vertices, where all geometry inside this space will be shaded by interpolated light values associated with each vertex. Given the volume resolution, k, the total number of volumes is defined in a cubical $k \times k \times k$ grid. Light probes are only placed in volumes containing geometry, which means that the set of volumes are represented by a sparse structure. Traversing volumetric data sets may be time consuming if the search algorithm is naive. Therefore an octree approach is used, where the root node is the entire scene's bounding box, and each leaf node is a volume. Each triangle of each mesh in the scene is tested against each volume using *fast triangle-box test* (Akenine-Möller, 2005). Given a triangle, defined by three vertices, and a box volume, this tests if the face of the triangle intersects the volume or if any of the vertices are inside of the volume. At each vertex of a volume containing geometry a probe is placed. The position of the volume where these probes are placed, is calculated from the amount of volumes, k, and the min and max position of the scenes bounding box. Two neighbouring volumes, both

containing geometry, share overlapping probes, i.e the same probe position is not stored twice. Thus, the maximum amount of probes is $(k+1) \times (k+1) \times (k+1)$.



(a) (b)

Figure 3.1: Teal cubes refer to the light volumes and red spheres indicate light probes. Figure(a) shows the Simple scene, Probes are only located near geometry. Figure (b) shows the subway example scene, no volumes are placed in the open hall in the right half of the image, however the pillars, roof and floor are covered in light volumes.

3.1.2 Generating radiance for each probe

Each probe needs to be associated with a radiance value. Radiance is described by a 5-dimensional function which, given a point and a direction, returns the outgoing light energy, Section A.4. To encode these values, 3^{rd} order Spherical Harmonics is used, Section 2.3. Radiance is generated using a modified version of Stingray's path tracer by sampling the indirect light for 256 paths starting from each probe position. This procedure is performed on the GPU.

3.1.2.1 Treatment of occlusion inside light volumes

Light leakage is a visual phenomenon where light is able to travel through non-transparent geometry, causing visual artifacts seen as illuminated areas where light should be occluded. During shading, irradiance needs to be calculated for each surface fragment by sampling the radiance of each probe not occluded by other geometry. Therefore, a method for filtering occluded probes is needed. This method generates a representation of which parts of the volume the radiance from each probe can be transferred to.

Each light volume is divided it into several subvolumes, for this project a $4 \times 4 \times 4$ resolution is used, yielding 64 subvolumes. Each probe in the volume has a corresponding bitmask where each bit marks if a subvolume is occluded or visible. To represent this bitmask a 64 bit unsigned integer is used, making the memory requirements for a volume, with eight probes, 64 bytes. Unlike the radiance data for each probe, these bitmasks cannot be shared with other volumes.

To illustrate the generation of the bitmask, a series of images are presented in Figures 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7. These images describe a 2D representation of a volume. A square with a 4×4 grid describes the volume and each grid cell corresponds to a subvolume. The circles at the corners of the square represent light probes. Since a bitmask is generated for each probe, the images focus on the generation of a bitmask for a particular probe. That probe is referred to as the *active probe* and is represented with a teal color in the images. The red lines correspond to geometry in the scene, where each line segment is a triangle in the 3D case. The 2D case is analogous to the 3D case, and is only used as a simplified example.

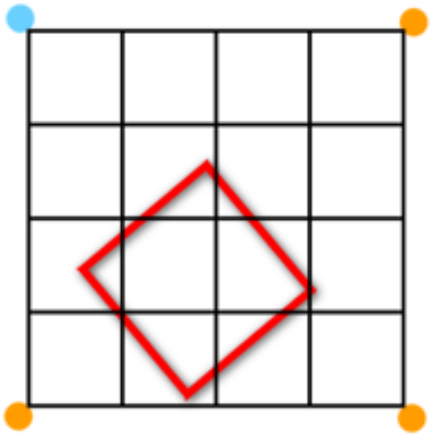


Figure 3.2: The circles in the corner represents the probes. The teal circle represents the active probe. The square containing the grid represents the volume and each cell represents a subvolume. The red lines represent geometry.

To determine if a probe is occluded, a path-finding algorithm is used. Starting at the active probe, we want to traverse the volume until we reach the opposite planes. This is done by creating several paths, each starting from the subvolume closest to the active probe and ending in a subvolume at the opposing plane. An image representing the opposing subvolumes for an active probe can be seen in Figure 3.3.

Whether a subvolume is occluded is based on geometry within the subvolume. In order to determine if a subvolume contains geometry, we used the same approach that was described in Section 3.1.1. The result from this operation is a bitmask which can be seen in Figure 3.4.

The path generation is an iterative process where one subvolume is considered at a time. If a subvolume contains no geometry it is visible and the next subvolume in the path is considered. If a subvolume contains geometry a test is carried out to determine whether it is occluded or not. If a subvolume is occluded, no new

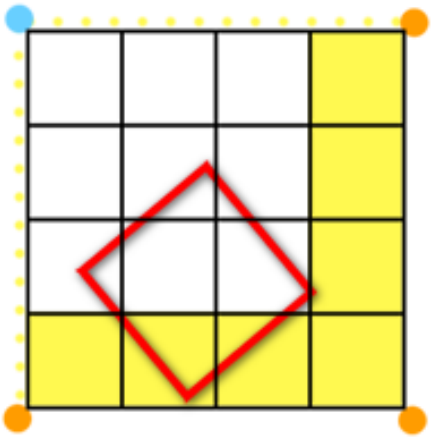


Figure 3.3: A volume containing a number of subvolumes and geometry. The active probe is located in the top-left corner. Each yellow cell shows an opposing subvolume for the active probe.

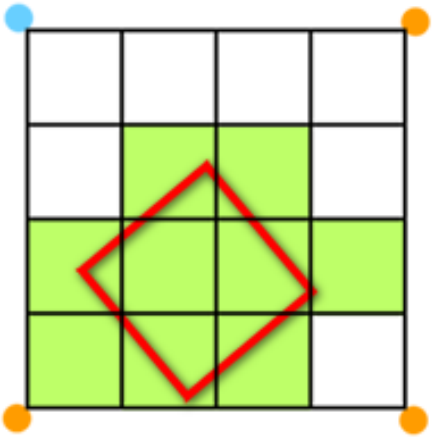


Figure 3.4: Geometry inside a light volume. The green squares represent which subvolumes that contain geometry.

subvolumes are considered for that path. A subvolume is occluded if it contains more backfacing geometry than geometry facing the active probe. The next subvolume to try is chosen based on the number of subvolumes between the current subvolume and the goal, in accordance with the path shown in Figure 3.5. The path is branched if two subvolumes are equally far from the goal.

A mesh surface has a backface and a frontface defined by the normal of the triangle spanned by the vertices. This poses the question of how to treat light leakage for infinitely thin planes. In this implementation, backfaceing areas can not receive light and frontfaceing should receive light. An example would be if part of a sphere is inside a light volume, such that at least one probe is inside the sphere, and the normals of the sphere points outwards. In this case the probe inside the sphere

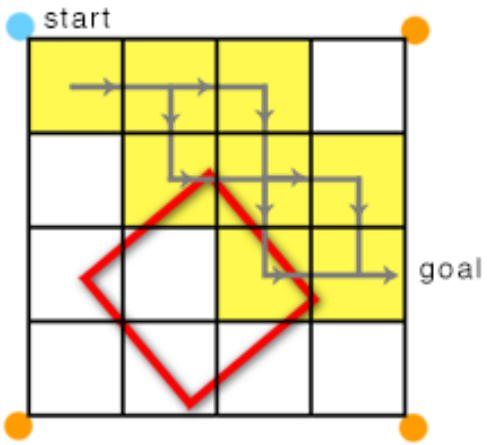


Figure 3.5: An example how a full path between a start subvolume and a goal subvolume.

should not contribute to the lighting of the surface, however, errors on the inside of the sphere are accepted.

To test if a subvolume containing geometry is occluded, the area of each triangle inside the subvolume is calculated. The goal with this part is to associate each probe in a volume with a score based on the accumulated triangle area facing that probe. A triangle is considered frontfacing if the face normal of the triangle is within 90 degrees of a vector from the volume's center towards the probe's position. In Figure 3.6a, the normals for the geometry inside the current subvolume are shown. The total triangle area inside the probe is also stored and a probe contributes to the result if the ratio between area associated with the probe and total area inside the volume precedes a certain threshold. The threshold yielding best result is 50%. Which can be seen in Figure 3.6b. If the subvolume contains a majority of backfacing geometry the path is considered occluded and no new subvolume candidates are generated from this subvolume.

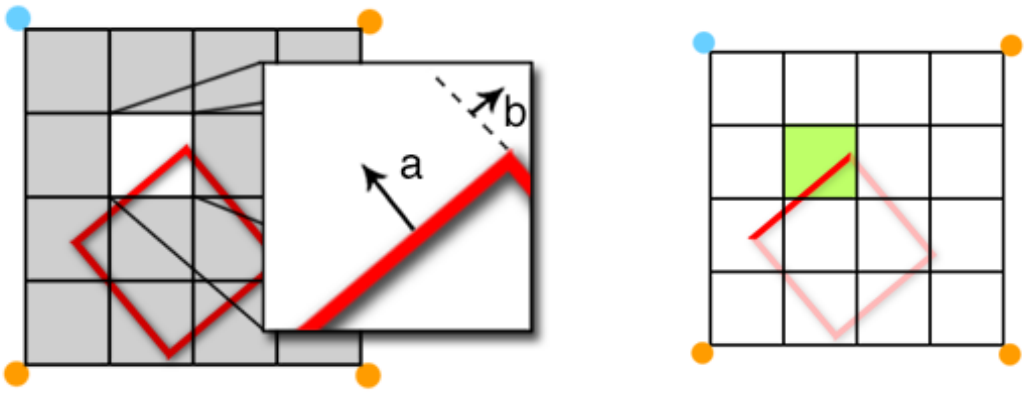
$$\text{Is visible} = \frac{Probe_{area}}{Total_{area}} > \text{Threshold} \tag{3.1}$$

3.2 Continuous phase

The continuous phase is active during the runtime of the application. For each frame and for each pixel, the correct radiance value needs to be sampled from the data structure and interpolated to achieve the indirect lighting.

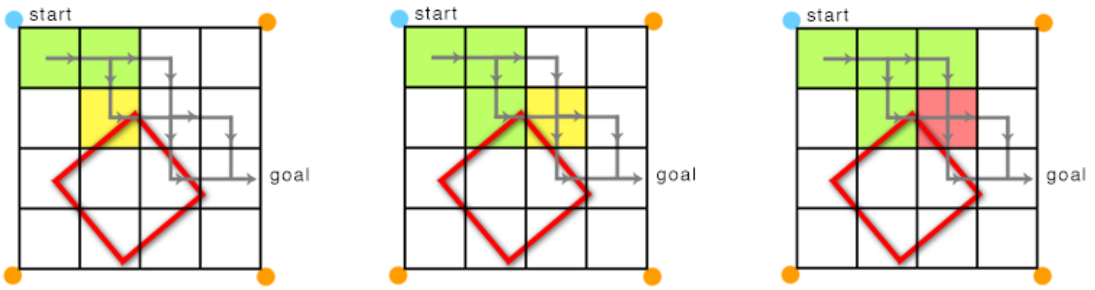
3.2.1 Sample the correct light value

To be able to sample the correct irradiance values in real time from the probes, we need to traverse all the probes in an efficient way. Similar to the method used in



(a) (b)

Figure 3.6: In Figure (a) the two normals for the different lines are shown. The normal label a is facing the active probe, The angle of the origo-to-probe vector and the normal b vector is 90 degrees and therefore its area not included. Figure (b) shows all area which is considered facing the active probe, where the transparent parts of the geometry is considered backfacing.



(a) (b) (c)

Figure 3.7: Green squares indicates subvolumes visible from the active probe, yellow squares show current subvolume, red square shows occluded subvolume. In (a) the subvolume with geometry facing the active probe is tested. Also the two probes not containing geometry is marked as visible. In (b) the previous subvolume is marked as visible, the current subvolume contains geometry which is backfacing. In (c) the subvolume treated in (b) is marked as occluded, the previous branch in the path has been taking, however also leading to the occluded subvolume and thus terminating the path.

Killzone 3, described in Section 2.1, this approach makes use of a perfect hashmap to find the probes corresponding to the current pixel.

On the GPU four probe-related buffers are stored, one containing the radiance values for each probe, one containing occlusion bitmasks for each probe in a volume, and two buffers spanning the entire volume with indices to the buffers containing probe data. Also sent to the GPU is the voxel resolution of the data structure, the size of a voxel in world space, and also the min and max position of the scenes

bounding box. By calculating the world space position of the current pixel, one can calculate which volume corresponds to that pixel, using the size of the voxel and where the first voxel is located. The probe index buffer array is packed such that the indices to the first volume's probes are located in the first eight elements, the second volume's are located in the next eight elements etc. Thus, by knowing which volume the pixel corresponds to, the probe data can be retrieved in $O(1)$ time. The occlusion bitmasks is obtained using the same technique. However, since the bitmasks is unique for each probe and volume, in regards to the radiance value for the probe's, a different buffer with indices needs to be used.

The memory needed for each of the four arrays can be seen in the equations following Equation (3.2). Equation (3.3) describes the memory needed for the SH coefficients, the array consists of 27 float values (9 coefficients $\times$ three colors). The array is sparse and the size is dependent on n, the voxel-resolution, and the number of volumes containing geometry. The data structure uses sparse arrays for the SH coefficiants and the occlusion bitmasks. The sparse arrays are queried using a two separate index arrays. One map to each volume and takes into account that volumes can share SH coefficiants between probes. The other is used to query the occlusion bitmask. Equation (3.2) describes the size in bytes needed for accessing the Spherical Harmonics coefficients, n denotes the voxel-resolution.

$$\text{SH-index}_{size} = (n+1)^3 \cdot 4 \quad [bytes] \tag{3.2}$$

$$\text{SH-coefficients}_{size} = 27 \cdot 4 \cdot \mathcal{O}(n^3) \quad [bytes] \tag{3.3}$$

$$\text{Occlusion-index}_{size} = 4 \cdot n^3 \quad [bytes] \tag{3.4}$$

$$\text{Occlusion-bitmask}_{size} = 2 \cdot 4 \cdot 8 \cdot \mathcal{O}(n^3) \quad [bytes] \tag{3.5}$$

3.2.2 Interpolating the result

When shading a specific surface fragment, using that fragment's world position, the voxel-resolution, min and max position for the data structure, the volume which the surface fragment belongs to can be calculated. To aquire the irradiance from each probe, the probes' SH coefficients are evaluated based on the surface normal and the bitmasks are retrieved. Using the world position and the number of subvolumes in a volume, the subvolume corresponding to the surface fragment can be calculated. Then, for each probe, the occlusion bitmask determines if the subvolume is occluded for that probe. If the probe is not occluded it should contribute to the interpolation, otherwise it should not. The resulting irradiance is obtained by using trilinear interpolation between visible probes where the weights are based on the distances from the current surface fragment to each probe. When interpolating between two probes where one is occluded, the result of the interpolation is the radiance value only for visible probe.

4
Results

The results of using the implementation during run time are presented in this chapter. The chapter has been divided into two sections, the first covers the memory and performance tests while the second presents the visual results of the implementation. The tests have been carried out on three different scenes. A *simple* scene to test performance for basic shapes, the *cornell box* scene to test correctness of indirect light and the *subway example scene* to test performance during a more game-like setting. Each scene is further described in Appendix A.3.

4.1 Memory and Performance

During the performance test, two different screen resolutions were tested, 1435×650 and 1435×935. The perfromance result was measured in render time for a frame and is shown in Table 4.1. The data structure used one pixel shader during runtime for querying the irradiance. The measurment was started when the fullscreen pass was invoked, for the pixel shader querying the irradiance, and was stopped when the fullscreen pass was completed. To test the worst case scenario for the implemented solution, the camera was placed such that the entire view port was filled with geometry. This is so that each pixel is rendered by the implemented shader. The time presented is an average over 5 seconds, with a deviation of at most 0.4 milliseconds.

The memory requirement for different buffers used by the data structure is shown in Table 4.2. Different scenes have been tested using differnt voxel resolution for the data structure. The resolution has been chosen to generate visually pleasing results.

Table 4.1: Average render time measured in milliseconds for the different scenes with a deviation of 0.4 miliseconds. Measurment with lower voxel resolution for the Cornell and Subway is not included.

Probe Resolution	Screen Resolution	Simple	Cornell	Subway
$8 \times 8 \times 8$	1435×655	6.4ms	-	-
$16 \times 16 \times 16$	1435×655	6.4ms	6.4ms	6.4ms
$16 \times 16 \times 16$	1435×967	9.8ms	9.8ms	9.8ms
$32 \times 32 \times 32$	1435×655	6.4ms	6.4ms	6.4ms

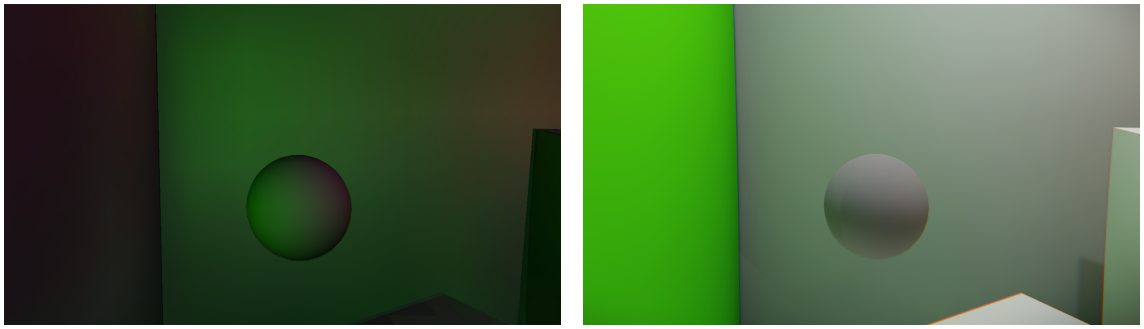
Table 4.2: Memory requirement for different scenes. n denotes the resolution of the data structure. SH-index and Occlusion-index denotes the buffers used for storing the indices in the sparse representation.

Scene	n	Active volumes	SH-index	SH-Coefficients	Occlusion-index	Occlusion-mask	Total
Simple	8	256	3 kb	61 kb	2 kb	16 kb	82 kb
Cornell	16	563	20 kb	121 kb	20 kb	36 kb	197 kb
Subway	32	1210	144 kb	225 kb	131 kb	77 kb	577 kb

4.2 Visual results

This section contains the visual results generated by the data structure. Each figure contains two subfigures, one shows the indirect lighting generated by the data structure while the other shows a reference image which uses the standard shader in Stingray.

Figure 4.1a shows the visual result using a volume resolution of $16 \times 16 \times 16$ voxels. The scene used is the cornell box scene and in this iteration a dynamic object, the sphere, has been added. The sphere is shaded using the same probes as the static wall in the background. The green color bleeding is visible at the left hemisphere. The red light in the image comes from the red wall opposing the green wall.

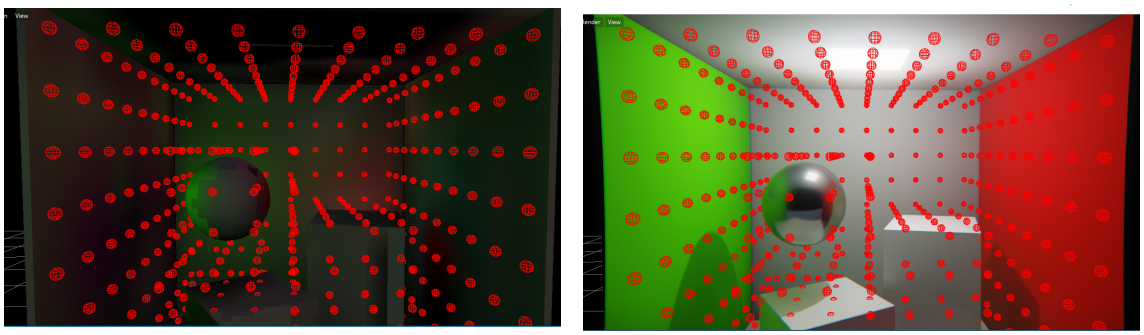


(a) (b)

Figure 4.1: Figure (a) shows the cornell box scene with only indirect light, using the implemented method with a resolution of 16 voxels. Figure (b) is used for reference, it shows the cornell box scene with all effects enabled and uses the standard light map for indirect lighting. The sphere in the center of the image is a dynamic object. Figure (a) shows how the indirect light from the green wall on the left side of the image affects both the static wall in the middle of the room and the sphere.

Figure 4.2a shows the cornell box scene. In this render all objects are static. The walls of the room are affected by color bleeding from the colored walls. Also the sphere has been shaded with the bouncing light from the wall. Both the walls and the sphere have been illuminated by indirect light from the implemented solution. The discontinuities shown as uniform green square patterns across the sphere constitute a region where a large amount of probes have been occluded. This leads to interpolation using too few samples and results in the seen artifact.

The difference in brightness between 4.2a and 4.2b is because only indirect light is shown in (a) and this is an additive effect, thus the darkness will not affect the end result. The red spheres indicate probe placements and are not part of the end result.



(a) (b)

Figure 4.2: Figure (a) shows the cornell box scene, using the research method for indirect light, with a resolution of 16 voxels. Figure (b) is used for reference, it shows the cornell box scene with all effects enabled and uses the standard light map for indirect lighting. The walls of the room are affected by color bleeding from the colored walls. Discontinuities can be seen on the left side of the sphere as uniform green square patterns.

Figure 4.3a presents the result of the implemented solution rendering indirect light for a part of the subway example scene. The camera focuses on the entrance where a light source is placed, simulating sunlight from the outside. The resolution of the data structure is 32 voxels. Some discontinuities can be seen at the wall at the top of the staircase on the left side. The discontinuities consist of uniform colored squares and appear due to using too few probe samples when interpolating.

(a) (b)

Figure 4.3: Figure (a) shows the subway scene, using the research method for indirect light, with a resolution of 32 voxels. Figure (b) is used for reference, it shows the subway scene with all effects enabled and uses the standard lightmap for indirect lighting. Some discontinuities can be seen at the wall at the top of the staircase on the left side. The discontinuities consists of uniform colored squares and appears due to using too few probe samples when interpolating.

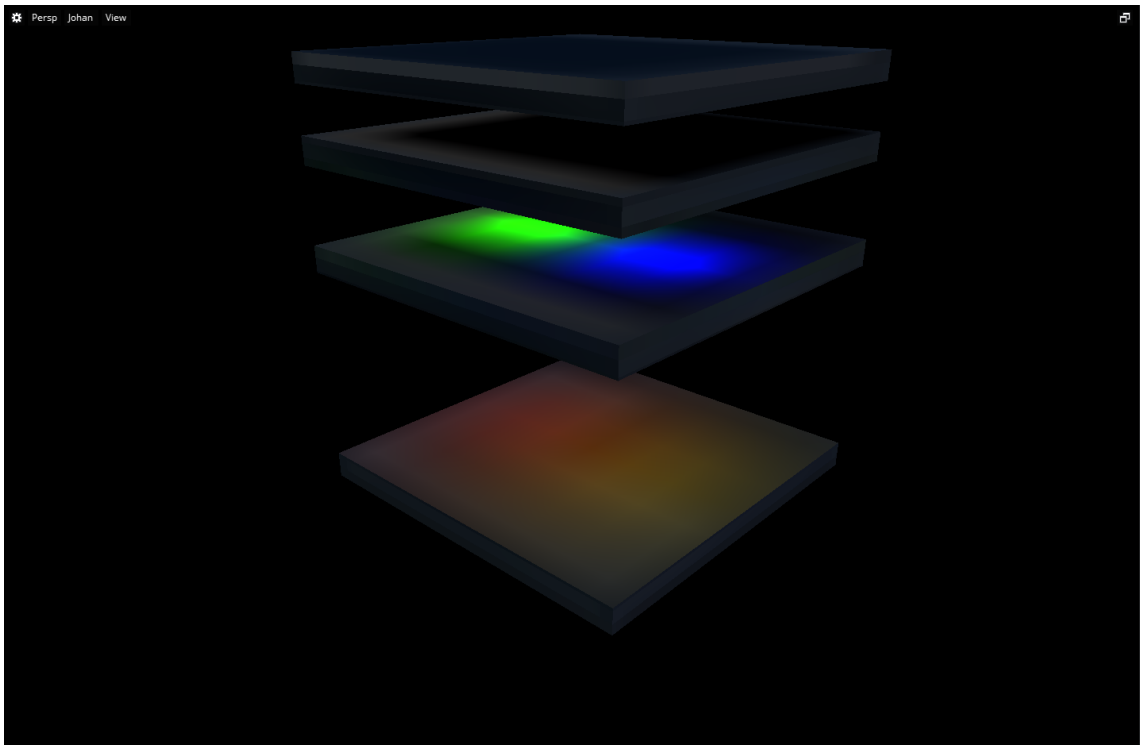


Figure 4.4: Shows the simple primitive scene, rendered with a resolution of 8 voxels.

A visual result of the quality using the implementation can be seen in Figure 4.4. It shows a render of the simple box scene. No light leakage can be seen between the planes. Some discontinuities can be seen at the side of the boxes near the top of each plane.

5
Discussion

The indirect lighting for static and dynamic objects is commonly shaded using different techniques. During GDC14, Valient (2014) presented a method which shaded both types of objects using the same technique, henceforth referred to as the *Killzone solution*. The technique was a probe based solution, usually used only for dynamic objects. Applied per pixel, this method could replace lightmaps for static objects. This thesis further explored the concept of a probe based solution as a replacement technique for lightmaps and the results are here compared to the Killzone solution and viewed in the light of a recently presented technique, Precomputed Radiance Transfer probes (PRT) (Stefanov, 2016).

The resulting implementation can be run in real time but uses a large render-time budget per frame. Memory complexity for the implemented solution is low compared to lightmaps, see appendix A.5, and theoretically as low as in the Killzone solution. The implemented solution for reducing light leakage generates artifacts in the final images, due to interpolating too few samples for a surface fragment. These three aspects will be discussed in more detail in the following sections.

5.1 The relation between light volumes and transfer basis from a runtime perspective

Definition of volumes and the choice of transfer basis are different for the implemented solution, the Killzone solution and the PRT solution. The implemented solution uses eight symmetrically placed probes to form a cube shaped volume, described in section 3.1.1, and 3^{rd} order Spherical Harmonics, section 2.3, as transfer basis. The Killzone approach defines volumes as tetrahedrons using four probes, section 2.1.1, the transfer basis used is not presented. The PRT solution uses a cubical grid, section 2.2.3, and HL2 ambient cube, section 2.2.2, as a transfer basis.

Real-time applications strives to at least be run in 30 frames per second leaving the render time for a single frame to a maximum of 33ms. The resulting render time for different scenes and resolutions, can be seen in table 4.1. The results are only dependent on the screen resolution and the number of pixels which uses the data structure. Values as high as 9.8ms leaves little time budget to other effects and there is a need to further optimize the run time. In the calculation of a frame the most expensive component is evaluation of the SH coefficients, which is done up to eight times per volume. This can be concluded by testing without SH, where instead a color vector is used for each probe. When using a single color vector per probe the frame is rendered in 1.5 ms and when using 3^{rd} order SH it takes 9.8 ms

to render, further described in Section A.6. Therefore the choice of transfer basis is dependent on the volume shape. In the killzone approach each volume will at most sample four different probes, potentially decreasing the number of samples by half. The PRT method uses a similar volume definition as in the project implementation, however, uses the HL2 ambient cube as transfer basis, which is cheaper to evaluate. Allowing a volume to consist of more probes.

5.2 The visual quality using the implemented occlusion bitmask

A 3D Occlusion mask, Section 3.1.2.1, has been implemented to filter out occluded probes for a particular surface fragment in a volume. Each probe in a volume has a bitmask describing which subvolumes are visible in that volume. The advantage of this method is the ability to store multiple depth values in a particular direction. A disadvantage is the lack of precision when storing where an occlusion occurs. An example of when the lack of precision produces an artifact is shown in Figure 5.1. The entire top face of the box should be visible for the probe located above the lower right part of the box top corner. The probe, however, should not be able to illuminate the backfacing areas of the box sides. The method implemented determines that the backfacing area is greater than the frontfacing area and occludes the top left subvolume for the probe. This produces a square shaped artifact on the plane due to interpolating between too few irradiance samples compared to the neighbouring subvolumes. An example of such an artifact is shown in Figure 5.2. Even by subdividing the volume further the problem will still not be solved and the memory complexity will increase. The Killzone solution uses a shadow map approach, Section 2.1.3, which to a greater extent eliminates this kind of problems by storing where the occlusion occurs with higher precision.

In retroperspective, another interesting approach would be to disregard elemination of light leakage completely, but rather use a permissive approach which focuses on getting the overall ambient light of a room. Ambient Occlusion could then be used as a post process to cover local light leakage, often already included in these types of real-time applications.

5.3 Future Work

The main drawback of the current implementation is the square like artifacts which are generated by the bitmask filtering technique. To be able to use this implementation, how to filter out occluded probes needs further investigation. The usage of a shadow map approach, similar to the one used in the Killzone solution, where a more accurate depth value for occluded geometry can be stored is interesting to further research. Another aspect worth researching, which has been outside the limitations of this research, is to take occlusion into account in the baking process. An example would be to take occlusion into account when storing the irradiance values, such that the radiance is blocked for certain directions. If a successful method is found,

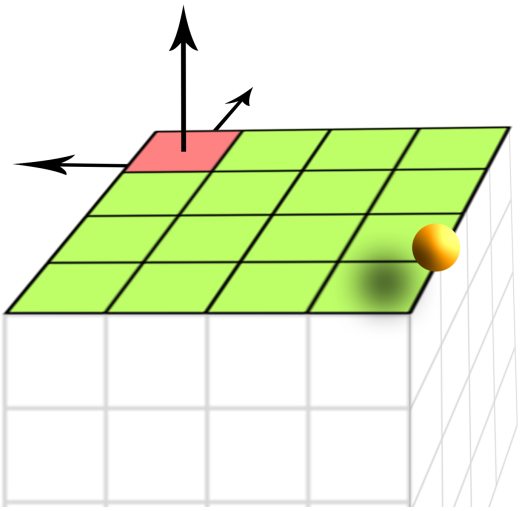


Figure 5.1: An example of when a subvolume is treated as occluded where it should be visible. Each grid cell corresponds to a subvolume in the bitmask. The bit mask corresponds to the probe, shown as an orange sphere in the figure. The green squares correspond to subvolumes which are visible to the probe and the red square indicates an occluded subvolume. The normals of the cube's three planes are shown for the top left subvolume. Two points away from the probe, while on points towards, thus the majority is back facing and the subvolume is treated as occluded.

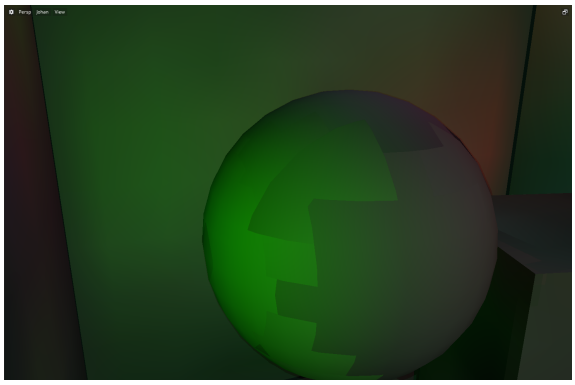


Figure 5.2: An example of square shaped artifact due to too few interpolation samples. The artifacts are seen at the left part of the sphere.

all probes in the light volume could be used during the interpolation. This would result in a simple trilinear interpolation which can offer a performance improvement by using GPU Volume Textures hardware accelerated trilinear interpolation.

Another way to optimize the visual quality and performance is to consider light volumes of different sizes. A light volume containing complex geometry can be subdivided to an appropriate size, while simple geometry uses larger volumes. The current implementation uses the same volume size regardless of what kind of geometry it holds. Using asymmetrical volumes, the lookup structure needs to be partitioned in such a way that it provides faster search times than a linear search. Another reason to partionate light volume is the possibility to stream an amount of light volumes to the GPU, making the solution able to handle levels several orders

of magnitude larger than the tested subway scene.

The current solution does not distinguish between distant and close surfaces. Distant surfaces do not fluctuate in irradiance as close surfaces do and therefore do not need the same resolution. Performance could be improved by using some sort of clamped values for distant surfaces, similar to what is used in the PRT solution described in Section 2.2.3.

As stated in section 5.1, the definition of volumes and the choice of transfer basis are related. This implementation would benefit from either storing the values in another type of transfer basis cheaper to evaluate or from choosing a volume definition that uses fewer probes.

5.4 Ethical aspects of improved computer graphics

Computer graphics is merely a tool to present content in an application. The application itself could be used for entertainment, advertising, scientific visualization or industrial design. Different applications treat different ethical aspects. An example is that video games aimed toward children need to have other focuses than computer-aided design tools. Since this thesis treats global illumination, with the intent of providing life-like computer graphics, an ethical issues could be behavioral changes due to exposure of virtual reality (VR).

Weisel (2015) presented an article discussing the effects of VR in psychoanalytic terms. Virtual reality differs from that of reading a book or playing a board game. A user can lose the sense of time and space in a VR application. An example is in video games, where one can reload the game after a mistake, and go back to a previous point and thus experience a new beginning. The conclusion of the article is that transitions between the fictional and the real world can be blurred.

Which elements of the virtual world could be transferred to the real world, is an open research question (Fritz, 2003). Fritz presents a theory that the first signs of changes in behavior of an individual, are *associative transfers* from the virtual to the real world. Associative transfers are when one associates video game events with real life events. An empirical study showed that two-thirds of computer game users have experienced these transfers (Fritz, 2003).

Both these article points to virtual reality as an influential medium. Brey (1999) makes a comparison with more traditional mediums. He states that even though immoral actions and behaviours exists in other medias such as television and literature, they are hard to compare to VR due to how differently user engages in such medias. Television and literature are experienced in a passiveness, where a VR application requires the user to actively engage to experience it. Even though there exist other active medias such as board games, VR still distinguishes from those, by being immersive.

One of the most researched areas of VR environments is that of violence in video games. Over the years a number of empirical studies and scientific researches have been performed, however, if there exists a link between violence in VR and in real life is still uncertain (Lishner, Groves, & Chrobak, 2015). Lishner et al. performs a

meta-analysis where they question if researchers are biased towards a certain result, since no consensus has been found regarding violence and VR. Thus, how influential VR environments are is a difficult question to answer.

In a VR application it is up to the developer to design the intention of the program and to implement which actions a user can perform. As VR is an influential medium, where there is no limitation on what kind of actions that can be implemented, developers need to take precaution regarding ethical aspects or immoral actions during development. An example is military VR simulators for combat training. When developing a simulator with realistic intents there could be variations in the degree of accuracy, such as wounds in the simulator are not expressed in a graphically realistic way or that enemies portrayed in the simulator may be based on stereotypes or propaganda. When training in such an environment it could be hard to retain a critical mind towards the application. To conclude, reaching life-like computer graphics in itself is not an ethical issue. However, developers should take precautions when it comes to ethical aspects of immoral behavior in an application with realistic graphics, a recomendation is to make use of test groups with a diverse background to test the application.

6
Conclusion

This thesis has researched the possibility of using a single technique to query indirect lighting for both dynamic and static objects. A data structure has been developed which uses a probe-based solution to define irradiance values for both types of objects. The resulting implementation can be run in real time but uses an extensive amount of render time per frame. This is due to the use of an expensive transfer basis which is sampled multiple times for a single surface fragment. Memory complexity for the implemented solution is low compared to light maps. The visual quality of the rendered images is lacking because too few probes are used during the interpolation. This is due to the method being prone to occluding probes. To further improve the data structure, a number of methods have been proposed.

To optimize runtime, the solution will either need to use light volumes consisting of fewer probes or use a cheaper transfer basis. Another way to increase performance is to clamp distant surface fragments to a fixed set of probes, which can be reused without large impact of the visual result.

The method used to filter out occluded probes generates artifacts and is the main drawback of the implemented solution. One way to fix this problem is to use a filtering technique with higher precision, similar to a shadow map. Another interesting approach is to disregard elemination of light leakage completely, but rather use a permissive approach which focuses on getting the overall ambient light of a room. Ambient Occlusion can be used as a post process to cover local light leakage.

To summarize, the use of a probe-based method for illuminating static and dynamic objects is possible. The winnings in memory savings and visual quality by sharing the same light system is worth researching more. Even if this project did not achieve the quality strived for, its findings can be used in developing a better solution.

References

Akenine-Möller, T. (2005). Fast 3d triangle-box overlap testing. In *Acm siggraph 2005 courses* (p. 8).

Akenine-Möller, T., Haines, E., & Hoffman, N. (2008). *Real-time rendering.* CRC Press.

Bentley, A. (2014). *Infamous second son engine postmortem.* `http://suckerpunch.playstation.com/images/stories/GDC14_infamous_second_son_engine_postmortem.pdf`. (Game Developers Conference)

Brey, P. (1999). The ethics of representation and action in virtual reality. *Ethics and Information technology*, *1*(1), 5–14.

Cupisz, R. (2012). *Light probe interpolation using tetrahedral tessellations.* `http://twvideo01.ubm-us.net/o1/vault/gdc2012/slides/Programming%20Track/Cupisz_Robert_Light_Probe_Interpolation.pdf`. (Game Developers Conference)

Deering, M., Winner, S., Schediwy, B., Duffy, C., & Hunt, N. (1988). The triangle processor and normal vector shader: a vlsi system for high performance graphics. In *Acm siggraph computer graphics* (Vol. 22, pp. 21–30).

Fatshark. (2014). *Warhammer: End times - vermintide.* [Video Game] Deep Silver.

Fatshark. (2016). *Escape dead island.* [Video Game] Fatshark.

Fritz, J. (2003). Wie virtuelle welten wirken. *Über die Struktur von Transfers aus der medialen in die reale Welt. U: Fritz, Jürgen/Fehr, Wolfgang (ur.): Computerspiele. Virtuelle Spiel-und Lernwelten. Bonn.*

Greger, G., Shirley, P., Hubbard, P. M., & Greenberg, D. P. (1998). The irradiance volume. *Computer Graphics and Applications, IEEE*, *18*(2), 32–43.

Guerrilla Games. (2011). *Killzone 3.* [Video Game] Sony Computer Entertainment.

Kajiya, J. T. (1986). The rendering equation. In *Acm siggraph computer graphics* (Vol. 20, pp. 143–150).

Lishner, D. A., Groves, C. L., & Chrobak, Q. M. (2015). Are violent video game-aggression researchers biased? *Aggression and Violent Behavior*, *25*, 75–78.

Mitchell, J., McTaggart, G., & Green, C. (2006). Shading in valve's source engine. In *Acm siggraph 2006 courses* (pp. 129–142).

Sloan, P.-P. (2008). Stupid spherical harmonics (sh) tricks. In *Game developers conference* (Vol. 9).

Stefanov, N. (2016). Global illumination in 'tom clancy's the division'. `http://mrakobes.com/Nikolay.Stefanov.GDC.2016.pdf`. (Game Developers Conference)

Sukys, P. (2014). Light probe cloud generation for games.

Ubisoft Massive. (2016). *Tom clancy's the division.* [Video Game] Ubisoft.

Valient, M. (2014). *Taking killzone shadow fall image quality into the next generation* (Vol. 14). https://www.guerrilla-games.com/read/taking-killzone-shadow-fall-image-quality-into-the-next-generation-1.

Weisel, A. (2015). Virtual reality and the psyche. some psychoanalytic approaches to media addiction. *Journal of Analytical Psychology*, *60*(2), 198–219.

Appendices

A

Appendix

A.1 Autodesk Stingray

Stingray is a 3D game engine with support for all major platforms. The game engine was developed by Fatshark, a game studio based in Stockholm, under the name *Bitsquid*. Games built with the engine include titles as *Warhammer: End Times-Vermintide* (Fatshark, 2014) and *Escape dead island* (Fatshark, 2016). The game engine is implemented in C++, Lua, and supports both OpenGL and DirectX. The data structure presented in the thesis is implemented in C++ and using the internal API for DirectX. Shaders have been written in HLSL.

A.2 Test System specification

All tests performed on the different scenes have been carried out on a single high performance desktop computer. The specification of the computer can be seen in Table A.1

A.3 Scenes to test different aspects of the implemented solution

To evaluate the different parts of the data structure, different scenes were used to test for specific properties. There are three types of scenes used for the project, one using simple primitive geometry, one for testing color bleeding effects and one similar to a real game level.

A.3.1 Simple primitive Scene

The *Simple primitive scene* consists of a number of primitive meshes. A robust system is required to handle basic shapes, such as the ones in this scene.

The purpose of this scene is to test if the shaded surfaces are seamlessly interpolated when some probes are occluded, such that no edges between different light volumes are visible.

The scene has four light sources, a red, green, blue and yellow one and consists of four rectangular box shapes of equal size. The shapes are overlapping in the z-direction. The green light and the blue light are placed between the second and the

Table A.1: Specifications on the test system.

2x Intel Xeon CPU E5-2697 v2	
Core Count	12
Clock Speed	2.7 GHz
Cache	30MB
Max Turbo Frequency	3.5GHz
Nvidia Geforce GTX 580	
CUDA Cores	512
Graphics Clock (MHz)	772 MHz
Processor Clock (MHz)	1544 MHz
Memory Clock	2004 MHz (4008 data rate)
Standard Memory Config	1536
Memory Interface	GDDR5
Memory Interface Width	384-bit
Memory Bandwidth (GB/sec)	192.4
Other	
Ram	32 GB
Operating system	Windows 8.1 Enterprise

third box. The yellow light and red light are placed between the third and fourth box.

A.3.2 Global illumination test scene

Global illumination is tested using Stingray's version of the Cornell Box scene with a slight modification. The original example consists of four different cornell boxes, where the version tested in this project only contains one. The box itself has one face missing. The floor, the ceiling and one of the walls are white. The left wall is green and the right wall is red. A light source is placed in the middle of the ceiling. Two rectangle primitives are distributed on the floor, while a sphere is floating in the middle of the room. The purpose of this scene is to test the quality of the indirect lighting. The expected result is to have color bleeding from the walls to the different objects on the floor. The orignal version of this scene for Autodesk Stingray can be retrieved from the *creative market*, `https://creativemarket.com/apps/stingray/examples`.

A.3.3 Subway scene

The subway example level is another standard Stingray example scene. It consists of a hall and a subway platform, two train tunnels and a decent amount of complex objects, such as benches, newspaper piles, trash cans and phone booths. The scene has a number of light sources placed in the roof along the train tracks and the scene is also illuminated by indirect light from the daylight outside of the station, reaching the platform from the entrance to the station. The scene used in this project is slightly modified. All animated dynamic objects have been removed. The purpose

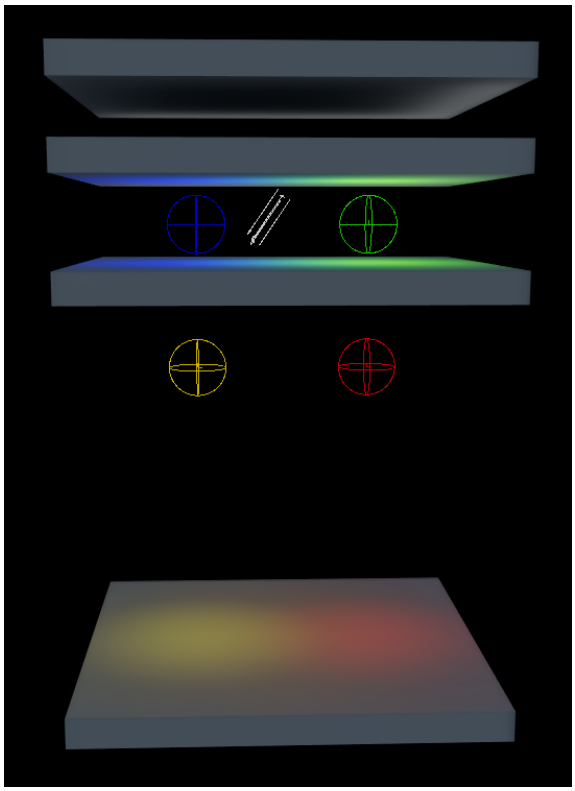


Figure A.1: A screenshot of the simple primitive test scene. All four rectangular boxes are visible, also the four light sources defined by the spherical gizmo in corresponig source color.

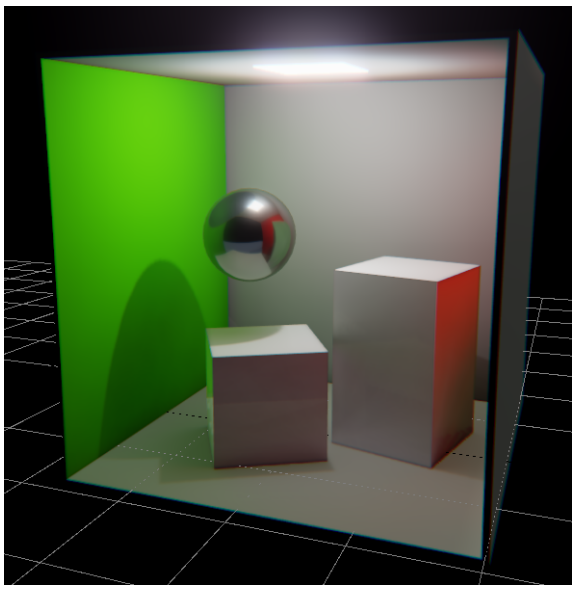


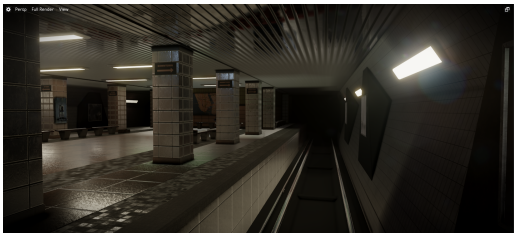
Figure A.2: Screenshot showing the beast example scene

of the scene is to test the scalability of the solution, and serves as a realistic test for the data structure.

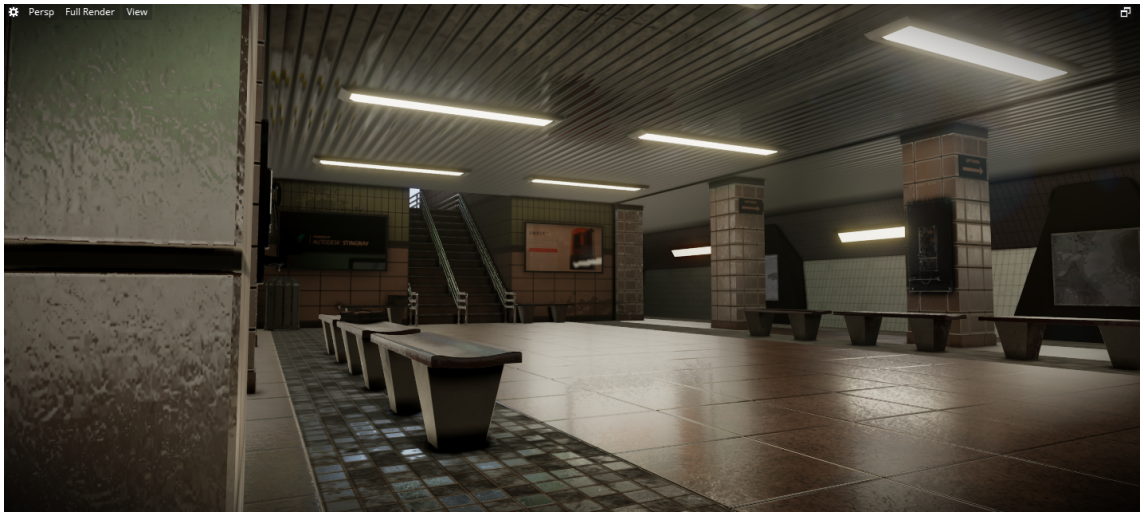
The scene is an Autodesk Stingray standard scene and can be retrieved from

creative market. `https://creativemarket.com/apps/stingray/examples`

(a)



(b)



(c)

Figure A.3: The subway scene used to test scalability of the solution.

A.4 Radiance and irradiance

Radiance is defined in the Rendering equation (A.1) (Kajiya, 1986), and describes the flow of light energy through a point in a given direction. Irradiance is the incoming light energy onto a certain point. Thus the irradiance corresponds to the integral in the Rendering equation.

$$L_o(\mathbf{x}, \omega) = L_e(\mathbf{x}, \omega) + \int_\Omega f(\mathbf{x}, \omega, \omega') L_i(\mathbf{x}, \omega') \cos(\mathbf{n}, \omega') d\omega'. \tag{A.1}$$

$L_i(\omega)$ is defined as the *field-radiance function*, which in this equation means the incoming radiance from direction ω. The cosine term scales the incoming radiance depending on the incoming direction. This is to represent that light with a steep angle smears out over a surface. Thus the steeper the angle the less it contributes to a surface.

A.5 Memory requirements for Lightmaps

Lightmaps have been generated using Stingrays new featured path tracer (currently in beta). The default settings have been used when generating the lightmaps, i.e the color format is 16 bit per channel and the texture scale is set to 10. The path tracer generates a lightmap for each material of each mesh. The size of all lightmaps generated for a level, and the number of meshes is displayed in Table A.2. The reason that the size of the lightmap for the Cornell Box scene is large may be due to being a level for testing global illumination and therefore uses lightmaps with high resolution.

Table A.2: Memory requirement for different scenes using lightmaps for each mesh

Scene	Meshes	Lightmap size
Simple	4	2298 kb
Cornell	4	56167 kb
Subway	135	22497 kb

A.6 Performance with and without 3^{rd} order Spherical Harmonics

To test the cost of 3^{rd} order Spherical Harmonics as a transfer basis for the implemented solution. A test was carried out that instead of evaluating the coefficient for each probe, a single RGB vector was used. The values are based on the probe index, such the extreme points are white and black, the probes in between are shaded based on where they are in the volume. The render of this image is shown in Figure A.4 and the results are shown in Table A.3.

Table A.3: Render time with and without Spherical Harmonics as transfer basis

Probe Resolution	Screen Resolution	Scene	Render time, SH	Render time, without SH
$16 \times 16 \times 16$	1435×655	Cornell	6.4ms	1.5ms

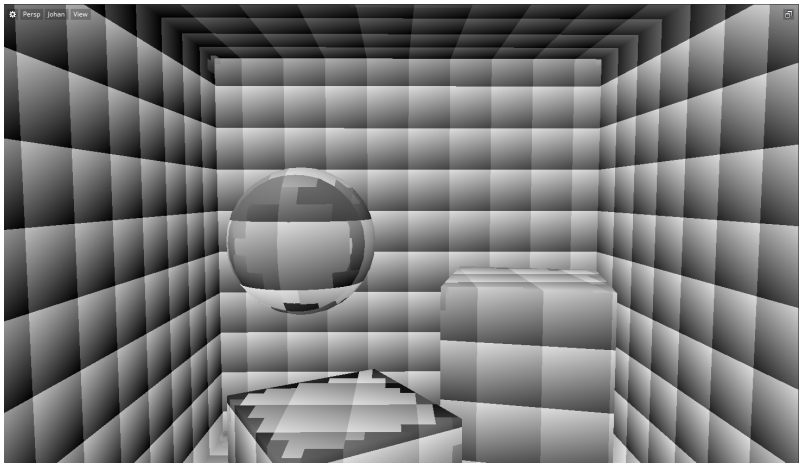


Figure A.4: Image rendering using a single vector as transfer basis instead of 3^{rd} order SH.