Thesis no: BGD-2014-01

Automatic spotlight distribution for indirect illumination

Lukas Orsvärn

Faculty of Computing
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Bachelor of Science in Digital Game Development. The thesis is equivalent to 10 weeks of full time studies.

Contact Information:
Author(s):
Lukas Orsvärn
E-mail: lukas.orsvarn@gmail.com

University advisor:
Stefan Petersson
Dept. of Creative Technologies

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. Indirect illumination – the light contribution from bounce light in an environment – is an important effect when creating realistic images. Historically it has been approximated very poorly by applying a constant ambient term. This approximation is unacceptable if the goal is to create realistic results as bouncing light contributes a lot of light in the real world.
Objectives. This thesis proposes a technique to use a reflective shadow map to place and configure spotlights in an environment to approximate global illumination.
Methods. The proposed spotlight distribution technique is implemented in a delimited real time graphics engine, and the results are compared to a naive spotlight distribution method.
Results. The image resulting from the proposed technique has a lower quality than the comparison in our test scene.
Conclusions. The technique could be used in its current state for applications where the view can be controlled by the developer such as in 3D side scrolling games or as a tool to generate editable indirect illumination. Further research needs to be conducted to make it more broadly viable.

Keywords: Reflective shadow map, global illumination, spotlight.

Acknowledgments

I would like to extend sincere thanks to **Stefan Petersson** for his valuable support and guidance throughout this project as my supervisor.

I would also like to thank **Max Danielsson** and **Alexander Vestman** for discussing ideas, providing feedback and helping out when I got stuck with various programming issues.

List of Figures

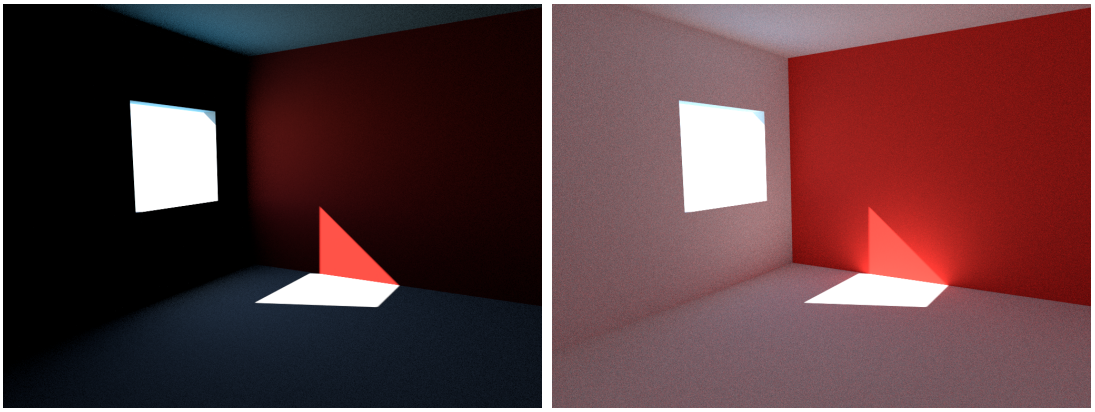
Contents

Chapter 1

Introduction

Indirect illumination is the light contribution to a surface that does not come from direct exposure to a light source. This thesis proposes a technique for automatic distribution of spotlights in a 3D scene to approximate that effect. It works by generating a reflective shadow map (RSM) and then distributing spotlights in the scene based on that data. The purpose of this chapter is to place the thesis in a bigger context as well as describe its scope and aim.

1.1 Background



(a) Indirect illumination off (b) Indirect illumination on

Figure 1.1: Indirect illumination.

Indirect illumination is a huge area of research for both real time and offline applications. The effect makes a big difference in some scenes and less in others. For instance, imagine a room with just one window with sunlight coming in as in Figure 1.1. Without indirect illumination the room would be almost entirely dark, save for the sunlight directly hitting the floor or walls and light coming through the window from the sky. When indirect illumination is added to a scene such as this, the room becomes much brighter. In some cases an entire room could be sufficiently lit by indirect illumination alone.

This effect is present everywhere, and is very evident in indoor areas containing surfaces that are hit by sunlight. In some scenes it is more subtle, like in an outdoor scene on a cloudy day. But even then some real world scenes are not reproducible without indirect illumination.

A very crude approximation of indirect illumination can be achieved by applying a static ambient term to the entire scene. In other words the indirect illumination is reduced to a uniform, dim color to make sure nothing is too dark, and thus the effect that light has when bouncing around in an environment is very broadly approximated. It is possible to make point and spot lights contribute indirect illumination in a similar way. The bouncing light is then produced by having light sources contribute some light to surfaces within its range that is not dependent upon the angle of the surface relative to the position of the light, but still has the same or similar falloff.

This way of approximating the effect by utilizing an ambient term has severe limitations as it does not take into consideration the color of the surface that reflects the light, nor is there any variability as to how much light reaches an area. One might even argue that these techniques should not be considered indirect illumination techniques at all since it is really more of a way to make sure nothing is too dark than an attempt at imitating indirect illumination.

Static lightmaps can contain high quality pre-calculated global illumination [1]. A static lightmap is a pre-calculated texture containing light data that is multiplied with the color of a surface during rendering of the environment. There are some significant drawbacks and advantages of this method. It is inherently static, it needs to be pre-calculated and stored as the calculation process often takes a long time. If a high quality shadow map is desired, the resolution of the lightmap needs to be quite high, further increasing already lengthy calculation times as well as load times and game size. Since it usually takes a long time to calculate, it is impractical to create during runtime.

However these drawbacks will not affect an application that does not need to update its light in real time, and the technique does provide high quality visuals. This is the method of choice to create static lighting – and as part of that – indirect illumination in a number of current game engines like the Source Engine [2], Unreal Engine 4 [3] and Unity [4]. It can be combined with real time shadows to achieve sharp shadows from direct sunlight and moving objects close to the player, but use the smoother, high performance static lightmap in distances and to get the indirect illumination effect. Our findings indicate that combining static lightmaps and dynamic real time lightmaps in this manner is the technique used in Counter-Strike: Global Offensive [5].

Another way of pre-computing indirect illumination is using spherical harmonics [6]. This is a more recent technique that is a more accurate solution as it does not only "paint" on the geometry in a scene, but instead saves light data for several points in space. This data is then used in lighting calculations to get the final image. While this technique takes more computing power than static

lightmaps, it also interacts with dynamic objects, something static lightmaps do not do.

It is also possible to achieve indirect illumination in real time with no precomputation, this can be done with techniques such as cascaded light propagation volumes [7] or voxel-based cone tracing [8]. Full real time indirect illumination is something that has been more common in recent years as computers get more powerful. However, these real time techniques are not as high quality as the static lightmaps and often require much more processing power during runtime. But since they update in real time as the scene changes, they work better with dynamic scenes. Because they are fully dynamic they have to be really fast, which makes it harder to make the light behave as expected. These problems can be mitigated by the fact that indirect illumination is a subtle effect in some situations, meaning imperfect results often suffice.

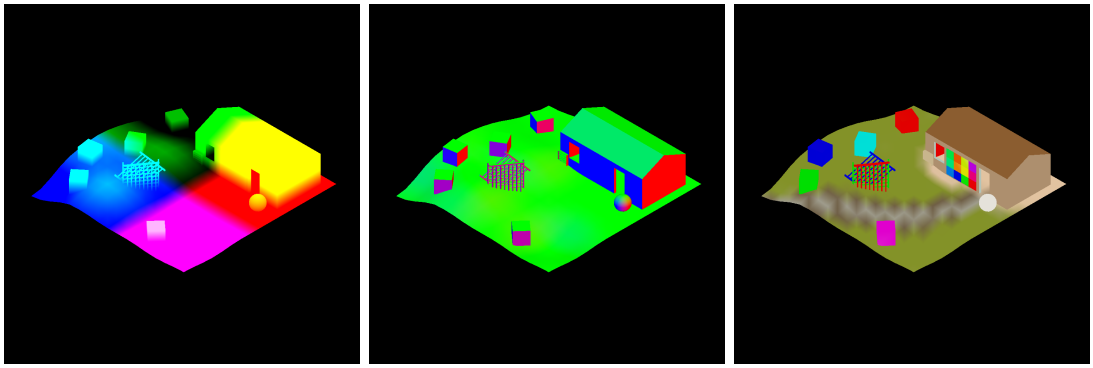
In short, indirect illumination is a huge area of research for both online and offline graphics rendering. No solution has been found yet for real time applications that give great dynamic results in real time, while also being fast enough to be usable on a broad range of hardware. The technique proposed in this paper explores the use of spotlights to create indirect illumination. Current real time rendering techniques like clustered deferred and forward shading allow for up to around a million lights to be rendered in real time [9]. We will explore utilizing this power by distributing spotlights in a scene offline that are then rendered as normal spotlights during runtime to achieve indirect illumination. The focus is on the placement and configuration of the spotlights in a scene to create the effect, not the performance of the technique or the visual accuracy of the results.

The proposed technique could be used to create tools for level designers to not only generate indirect illumination quickly, but also allow them to manually edit the result for greater control.

1.2 Related Work

In an article written by Dachsbacher et al. a technique is proposed to use a reflective shadow map (RSM) to create indirect illumination for the light emitted by a spotlight. This technique was to serve as an extension of the standard shadow map – that only contains the depth value of a texel – to enable calculating indirect illumination in real time [10]. This is accomplished by saving the surface normal, world position and flux value as seen in Figure 1.2 for each texel in addition to the depth value. The new data is used to calculate indirect illumination.

The proposed technique requires enough information about the surfaces that are hit by a directional light to be able to create a spotlight that has a hard-coded spotlight angle and falloff. So the spotlight angle and falloff attributes do not need to be saved for each individual spotlight. With that in mind, the data needed for each spotlight is position, direction and color. This means each texel



(a) World position (b) Surface normal (c) Flux

Figure 1.2: Three components of an RSM. It also contains a depth buffer, not shown here.

in an RSM contains the data needed to create a spotlight for our purposes.

Using an RSM to create spotlights was utilized in AMD's Leo Demo that was created to demonstrate their Forward+ [11] technique. Their implementation of Forward+ was first detailed in a paper by Harada et al. 2011 [12], and later explained further in GPU Pro4 [13]. Therein, a technique to create indirect illumination for spotlights using RSMs is briefly mentioned. They generate an RSM from the viewpoint of a spotlight and create new spotlights in real time using the data in that RSM to create spotlight in a pre-defined pattern. The proposed technique is based the same idea, but instead of generating spotlights in real time from a spotlight, we generate spotlights offline from a directional light with a focus on good spotlight placement.

1.3 Aim and Objectives

Aim: Develop, implement and evaluate a technique for automatically placing spotlights in a 3D scene to apply indirect illumination.

Objectives:

1. Propose and define a spotlight distribution technique.
2. Create a delimited 3D graphics engine.
3. Implement the proposed spotlight distribution technique.
4. Test and evaluate the proposed technique.

1.4 Research Questions

1. Can a reflective shadow map (RSM) [10] be used to automatically place spotlights in a 3D environment to create indirect illumination?

2. Will all surfaces visible in the RSM have one or more associated spotlights?

1.5 Method

An experimental application is created that implements the proposed spotlight distribution technique. Results are gathered by saving images of a test scene. To measure how well the technique works, two images are taken from the same place in the scene, one where there is one spotlight per texel in the reflective shadow map (RSM), and one where the proposed technique is utilized. The difference between the two images are measured and visualized to show how they differ.

Chapter 2

Terminology

To make this work easier to grasp we have defined some words that are used throughout the thesis. This chapter describes those words as well as other words that are important to fully understand the technique and how it works.

2.1 Types of Spotlights

In our implementation of the proposed technique there is technically only one type of spotlight. However the spotlights are given different names to signify how they are used and thus hopefully make this text easier to understand. reflective spotlights (RSes), aggregate spotlights (ASes) and sampling spotlights (SSes) are all actually the same thing: a spotlight. The different names are just used to indicate how they are used. Here follows a description the different types of spotlights.

An RS is one that is created from a single texel in a reflective shadow map (RSM). This name signifies that the spotlight is unaltered.

An AS is a spotlight created by aggregating several RSes.

An SS is a spotlight that is used to sample and compare RSes.

Again, keep in mind that these are only names used for spotlights to indicate how they are used, all spotlights are technically the same.

2.2 Flux

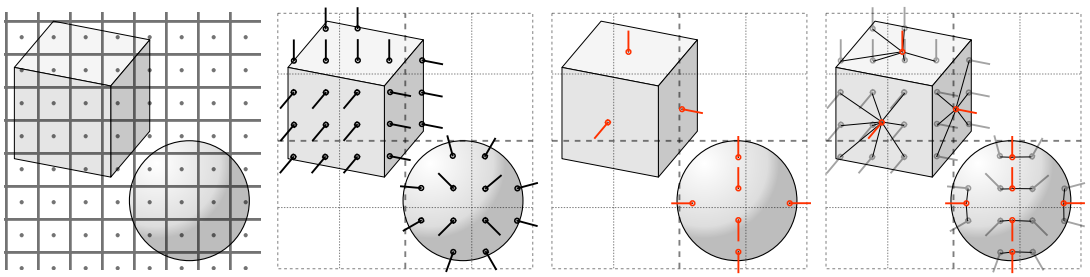
Flux is the color that is reflected by a surface under some light condition. So it does not only contain the color of the surface, or the color of the light that hits that surface, but a combination of those two; the color of the light that is reflected by that surface when hit by that light. Since the spotlights in the proposed technique represent reflected light, this data is used as the color of the spotlights.

Chapter 3

Technique

This chapter gives an overview of how the proposed technique works and then delves into more detail on how specific parts are implemented. The experimental application was developed using C++, so minor C/C++ terminology is present in the text.

3.1 Overview



(a) A reflective shadow map (RSM) is created (b) RSes are created from the RSM (c) SSes are created from the RSes (d) ASes are generated from the RSes and SSes

Figure 3.1: Overview of the proposed technique.

The technique relies on a reflective shadow map (RSM) to create spotlights in a 3D environment that are then aggregated according to certain rules to arrive at the final spotlight distribution. Here follows a broad overview of the technique in chronological order.

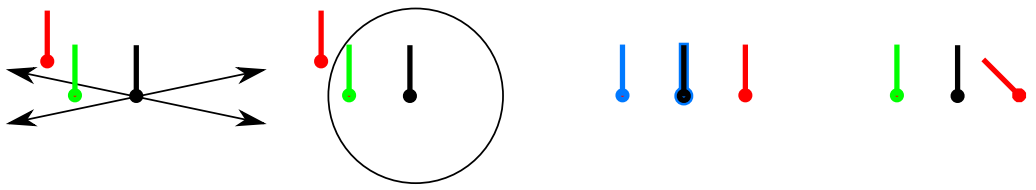
1. **Figure 3.1a:** An RSM is generated from the view of the directional light. Each texel in an RSM contains the data necessary to create a spotlight; a position, direction and color. With this data we are missing information regarding the spot angle and falloff of the light cone. That data is however not needed in this case since we will be approximating perfect diffuse reflection, so the angle will during rendering be set to cover 180 degrees, and the light cone falloff is predefined as well.

2. **Figure 3.1b:** A reflective spotlight (RS) is created for each texel in the RSM using the data in the respective texels. Those RSes are put into a list and pointers to them are put into each node they touch in an octree to speed up later comparisons. The size of the spotlights when pointers to them are put into the octree are the same as the distance value explained in Section 3.2.

3. **Figure 3.1c:** A number of sampling spotlights (SSes) are created and their configurations are optimized in the way described in Section 3.3.

4. **Figure 3.1d:** All RSes are aggregated into to their closest matching SS as described in Section 3.4. Then the original RSes are deleted as they are no longer needed. What remains is a set of aggregate spotlights that can be used in rendering.

3.2 Matching Spotlights



(a) Positional angle (b) Distance (c) Color (d) Direction

Figure 3.2: Spotlight matching criteria.

At several points in the proposed technique we will need to compare spotlights to one another to determine if their data match up closely enough. In this thesis that action is referred to as "matching". If two spotlights match one another, that means they pass this comparison test. To "match two spotlights" is to put them through this test.

Two spotlights are matching if the data in the spotlights match up to within certain empirical limits. These limits pertain to the position, direction and color values of a spotlight. All tests need to pass for two spotlights to be considered matching. The limits used for our test scene can be found in Section 4.1 and the code can be found in Appendix A. Here follows a description of the matching tests.

- **Positional angle**, the dot product between the direction of one of the spotlights and a vector pointing towards the other spotlight must be within the set limit. This prevents spotlights that are in front of and behind one another from being matched.

- **Distance**, may not be over the set limit to prevent spotlights that are too far away from each other from being matched.
- **Color**, after normalizing the colors of the spotlights, the difference between the same color channel in the two spotlights may not be over the set limit. This ensures the colors are kept reasonably pure when the spotlights are aggregated (see Section 3.4 for information on aggregation) and provides more predictable results.
- **Direction**, the dot product between the two spotlights have to be above the set limit. This is to prevent spotlights that are on opposite sides of corners or similar from being matched.

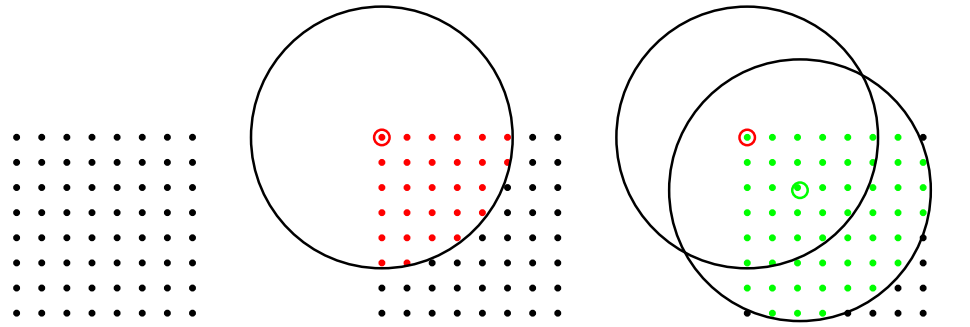
These tests determine how the spotlights are aggregated, the values used for them are empirical and thus should be adjusted for the type of graphics in the application. For instance in our test scene we had only vertex coloring, most color changes were smooth so a low color limit could be used. If textures are used in the application, the color limit could be higher to make sure an excessive amount of spotlights are not created on a heavily textured wall.

3.3 Spotlight Sampler Configuration Refinement

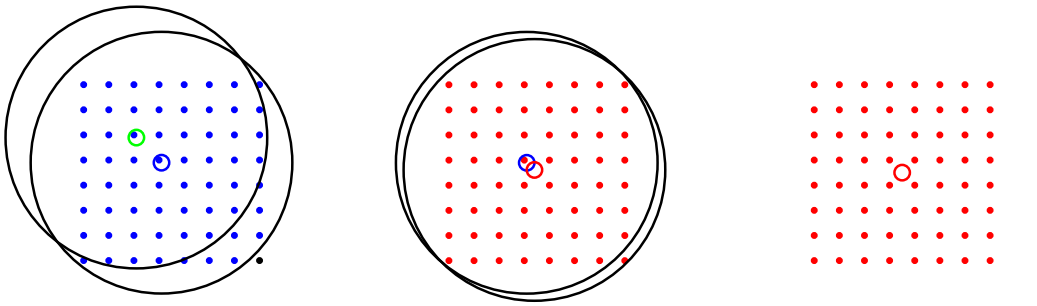
To make sure that spotlights are aggregated in the best way possible, we need to have a way of determining where sampling spotlights (SSes) should be placed. We could just take any spotlight and aggregate the ones around it and get reasonable results if we wanted to. But we can do better, this section describes a way to improve the placement of the SSes to lower the amount of and improve the placement of our aggregate spotlights (ASes). The code for our implementation of this technique can be found in Appendix B.

The high level idea is that we see how many reflective spotlights (RSes) match an SS with its current placement. Then we aggregate the data of all those matching RSes into a new SS and see how many RSes match with this new data. If we have more matches with the new SS than with the source SS we can interpret this new placement as being better because it covers more RSes and is thus of higher importance. Here follows a more detailed explanation of the process used to refine the SS placement.

1. All RSes are marked as not used.
2. A “final” SS is created by copying an RS that is not marked as used.
3. The number of RSes that match the final SS is stored.
4. A “test” SS is created by copying the data of the final SS.



(a) Grid of matching RSes (b) An SS is created (c) SS Configuration is refined.



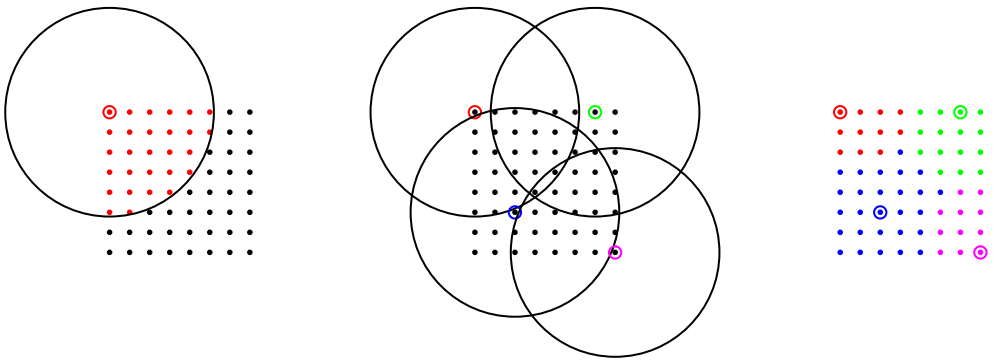
(d) Configuration is refined again (e) And again (f) Until attempting to refine further does not match more RSes

Figure 3.3: Spotlight sampler configuration refinement overview in chronological order.

5. The data of all RSes that match this test SS are aggregated into the test SS, used spotlights are not aggregated.
6. The number of RSes that match the test SS is stored.
7. The number of spotlights that match the final SS and the test SS are compared. If the test SS has a higher number, it is now considered the final SS.
8. Steps 4 to 7 are repeated until step 7 fails. At that point all RSes that match the final SS are marked as used and aggregated into the final SS that is then put into a list of SSes. Then the process starts over at step 2. When all RSes have been marked as used, the process ends.

When the process described just above has been executed, every RS has at least one matching SS.

Figure 3.4 shows what happens without the configuration refinement, more samplers end up at the edges of the surfaces, and the placement of those samplers



(a) SS is created (b) All SSes are created (c) Resulting aggregation

Figure 3.4: Spotlight sampler configuration without refinement.

are dependent on the orientation of the shape. However *with* the sampler refinement the placement is more predictable and less dependent on the orientation of the surfaces.

3.4 Aggregating Spotlights

The data for reflective spotlights (RSes) are aggregated into aggregate spotlights (ASes) the following manner.

The direction of several RSes are added to an AS by adding the directions of all the affecting spotlights and normalizing the result.

The position is aggregated by taking the position of the AS and then adding a vector to it that goes from the AS to the RS that is divided by the total amount of RSes that have been already aggregated.

The color of the AS is aggregated by summing the color of all RSes.

3.5 Spotlight Generation

The first step in generating the spotlights is to render out a reflective shadow map (RSM), RSMs are described in Section 1.2. The RSM textures are retrieved from the graphics card and each texel in the RSM is used to create a spotlight. The position for the spotlight is taken from the world position texture, the direction from the surface normal and the color of the spotlight is the flux.

It is worth noting that it is important to set the world position and flux textures on the graphics card to floating point, and perhaps a higher-than-default bit depth if needed. Otherwise data might be clipped, resulting in useless world position values and potentially less accurate flux values. You generally do not

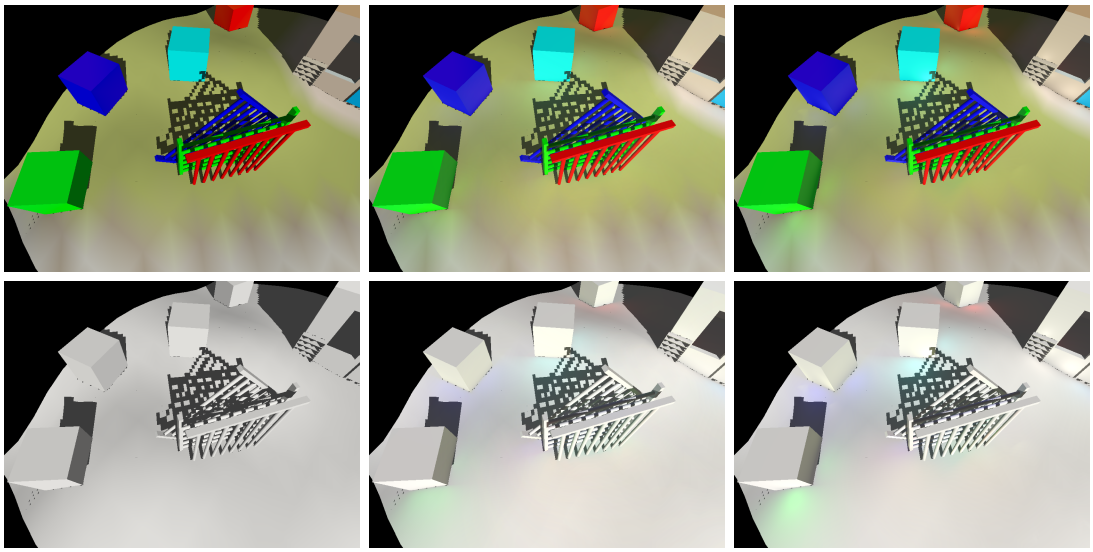
need to have high precision for the normals in this technique, so for the normal texture, 1 byte per component will suffice.

Chapter 4

Conclusion

An experimental application was developed to test the theory behind the proposed technique. This chapter presents the results gathered from that application, the analysis of those results, as well as the conclusion to this thesis.

4.1 Results



(a) No indirect illumination (b) One spot per texel (c) Optimized amount of spotlights

Figure 4.1: Results of the proposed technique. Top row: with vertex coloring. Bottom row: without vertex coloring.

The settings used for matching in this test scene are the following. Positional angle: 0.1, distance: 1.0m, color: 0.1, direction: 0.1. More information regarding what these values mean can be found in Section 3.2, and the code where they are defined and used can be found in Appendix A.

To determine the effectiveness of the proposed technique the difference between two images is created. Figure 4.1b shows what the scene looks like with

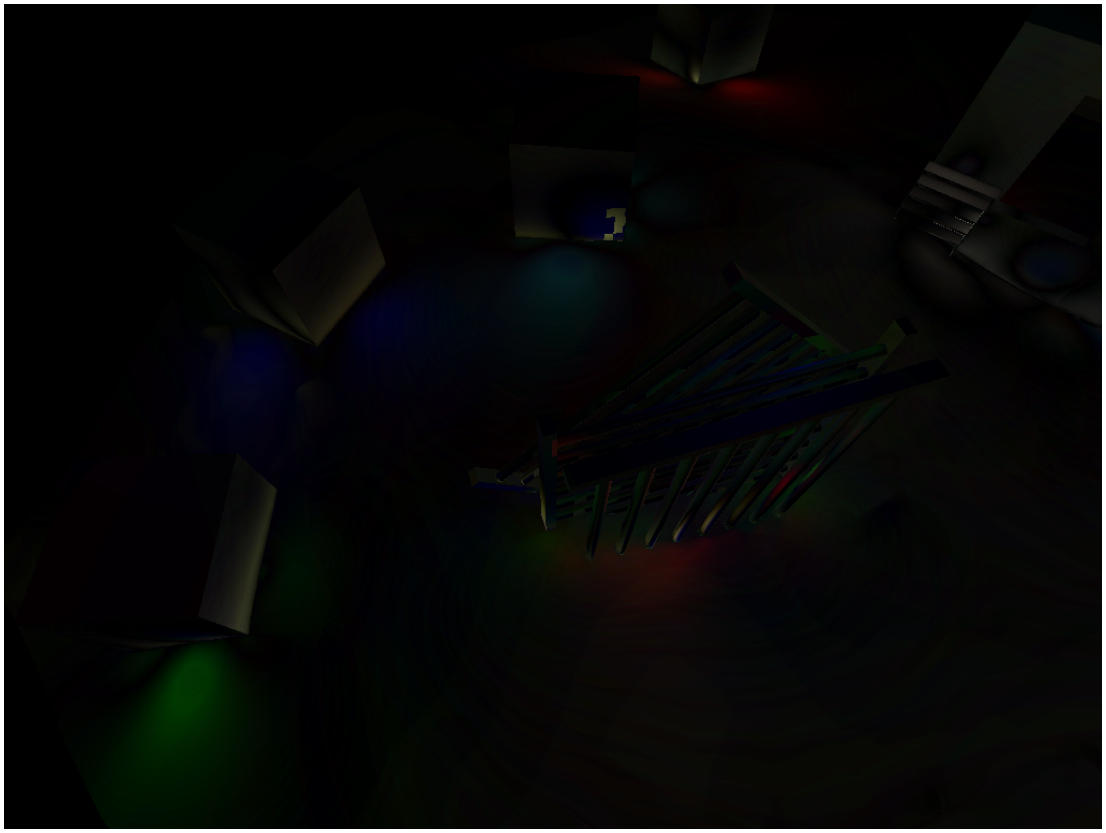


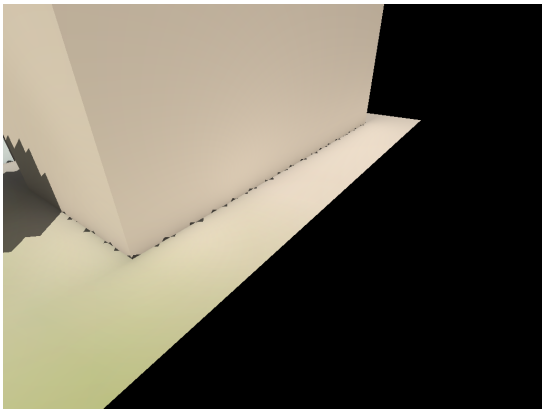
Figure 4.2: Difference between when using one spotlight per texel and the proposed technique. In this image the 0.0 to 0.5 color range has been remapped to 0.0 to 1.0 to more clearly show the difference.

one spotlight per texel in the reflective shadow map (RSM), Figure 4.1c shows what the scene looks like when using the spotlights resulting from the simplification process. The difference between those two images can be seen in Figure 4.2.

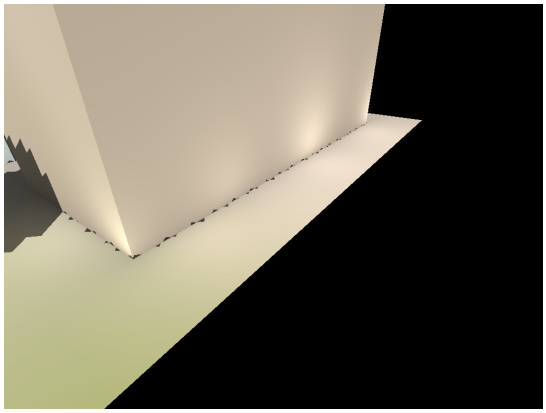
There are observations that can be made from Figure 4.2 regarding the indirect illumination effect. For instance, the simplified version is darker in some areas that are further away from reflecting surfaces. This is thought to be the result of an error in the calculation of the brightness of a pixel that causes for instance 10 lights with 1.0 in brightness spread over an area to illuminate the scene more than 1 light with 10.0 in brightness.

Since the simplified version has fewer spotlights with a higher power, it is in some places possible to see the contribution of individual spotlights in the final image. This is especially evident in Figure 4.3.

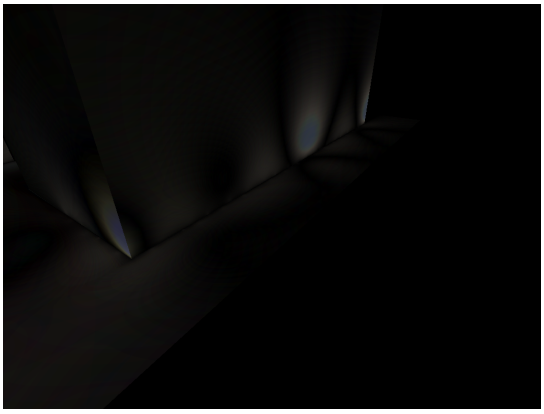
Figure 4.4 shows the spotlight placement before and after simplification. There was a total of 14,767 spotlights in the 12.0x12.0 meter large scene before simplification. after the simplification, that amount had been reduced to 270 when using the matching settings specified in Section 3.2, a 98.2% reduction and an average



(a) One spotlight per texel



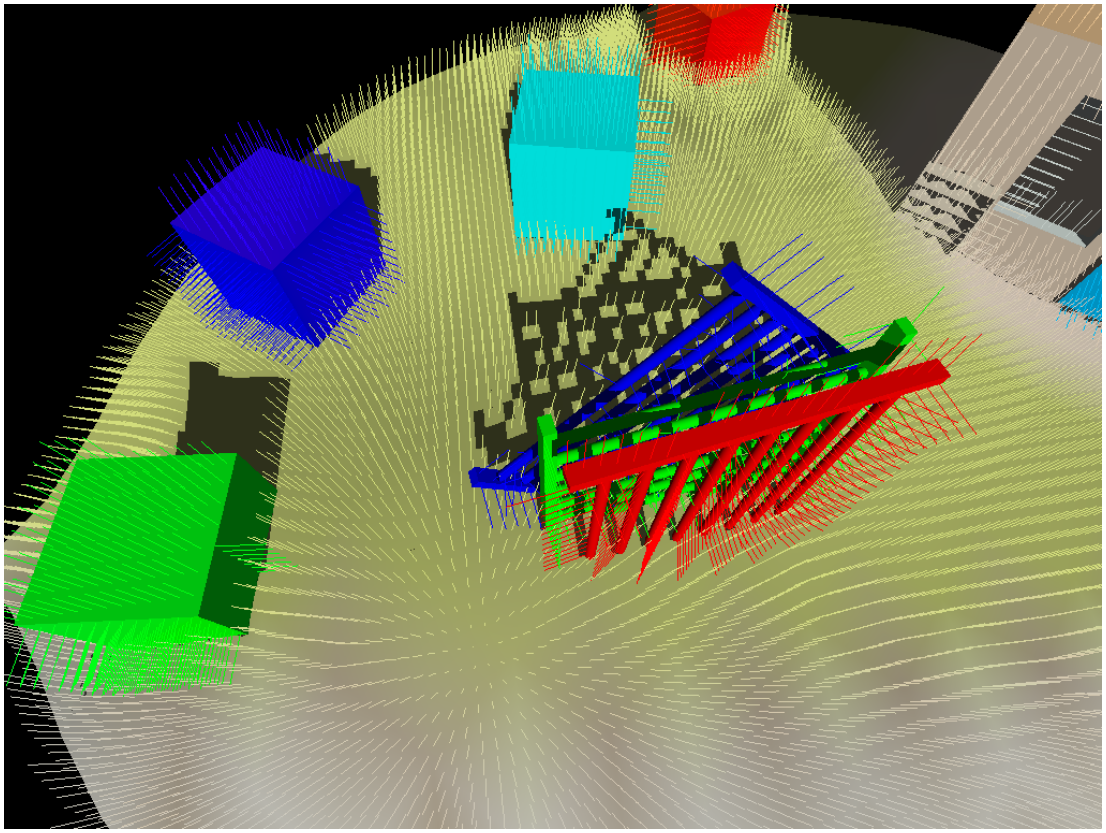
(b) Optimized amount of spotlights



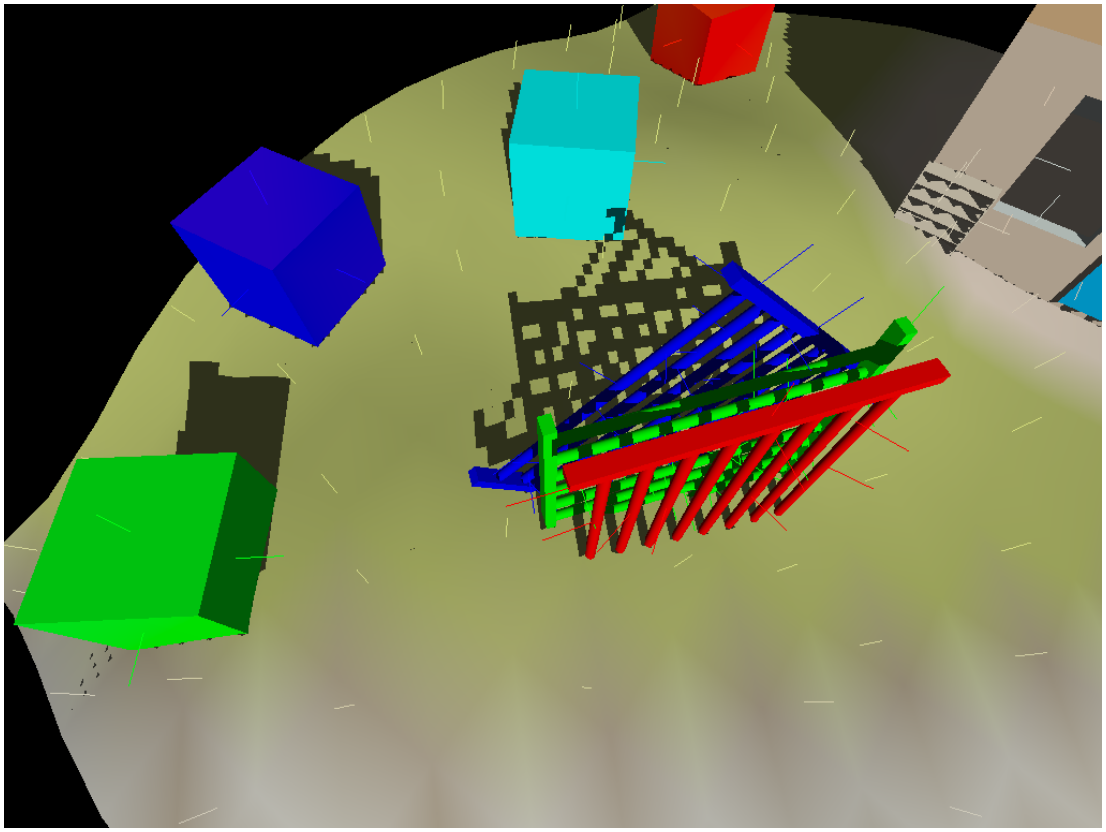
(c) Difference, the 0.0 to 0.5 color range has been remapped to 0.0 to 1.0 in this image.

Figure 4.3: Difference between when using one spotlight per texel and the proposed technique.

of 54.7 reflective spotlights (RSes) per aggregate spotlight (AS). To clarify, the reduction in spotlights will increase as more spotlights RSes are provided. With the settings specified in the beginning of this chapter, the entire process, from rendering the RSM to reducing the amount of spotlights takes 303ms on average in the test scene on a computer using an Intel Xeon E5-1650 CPU at 3.20GHz and an NVIDIA Quadro 4000 GPU.



(a) Before simplification



(b) After simplification

Figure 4.4: Spotlight placement in test scene.

4.2 Analysis

That the scene gets overall more illuminated by many low-intensity lights than by a few high-intensity lights is not a big problem for our purposes. In this thesis we are first and foremost concerned with the placement of the spotlights and not the accurate rendering of those spotlights. However this is something that will need to be taken into consideration when implementing and refining this technique if a goal is to get close to physically accurate lighting. Though if that is a goal, regular point spotlights are not a good light type simply due to them not being physically accurate to begin with, it is not possible for any infinitesimal point in space to emit any amount of light. An area light model would be a better alternative in that case.

Individual spotlights' contributions being visible in some areas after simplification as shown in Figure 4.3 is unacceptable when trying to create a realistic effect and needs to be alleviated. It could be helped somewhat by lowering the minimum distance between spotlights, but this would instead result in many more spotlights being created, making the proposed technique less effective. For some purposes, this drawback could be acceptable, but to make the technique more broadly applicable, further work will have to be carried out in this area. A few suggestions to this end are provided in Section 4.3.

The simplification technique is working as intended, taking the normals, colors and positioning of the reflective spotlights (RSes) into consideration to create a satisfactory spotlight distribution. Current implementations of clustered forward and deferred rendering can handle immense amount of light sources, upwards to around one million [9]. This makes the proposed technique well suited for some applications like for instance 3D side scrolling video games, or a tool for level designers to create editable indirect illumination. However to make it more broadly viable in for instance first person shooters where the player has the freedom to inspect the world more closely and might more easily notice irregularities in the indirect illumination, more work has to be done.

4.3 Conclusions and Future Work

A technique has been proposed for offline distribution of spotlights throughout a scene to create the effect of indirect illumination. A delimited graphics engine was programmed and the technique was implemented. Images were taken using the proposed technique and compared with an image using a naive spotlight distribution method to determine the quality of the result.

The technique works as intended, spotlights are aggregated based on certain criteria, and all areas that are hit by the directional light in the scene are properly taken into account in the technique. The number of spotlights resulting from the process are of an amount that is easily renderable with current real-time

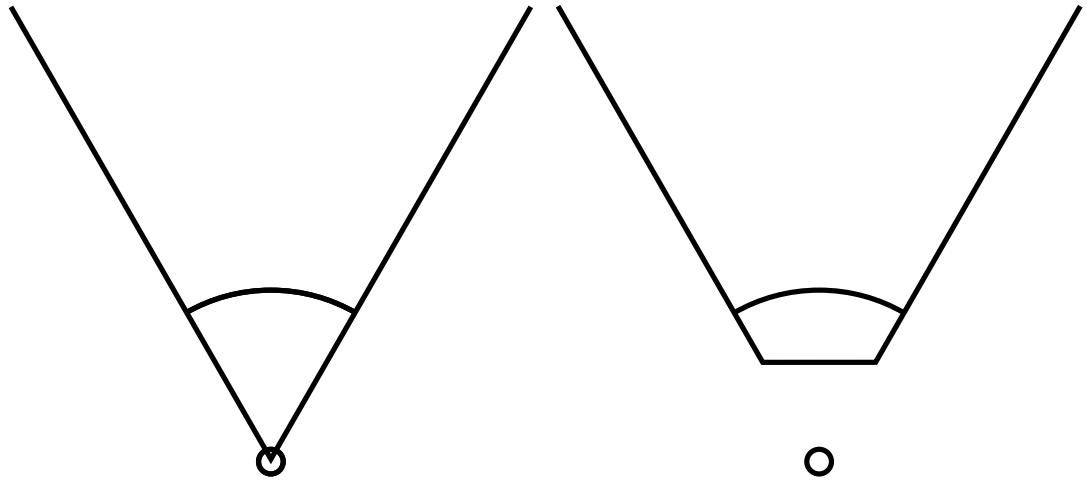
techniques.

However the difference between images showing the scene when using the proposed technique and images where a naive spotlight distribution method is used do display some differences. While some difference is acceptable, unfortunately the difference is quite clearly visible in that the contributions of individual spotlights can be seen in some areas, hurting the visual quality of the image.

Nonetheless the proposed technique is usable in its current state for some applications, for instance in 3D side scrolling games or as a tool level designers can use to create editable indirect illumination.

Research question 1, *Can a reflective shadow map (RSM) be used to automatically place spotlights in a 3D environment to create indirect illumination?* was answered. Spotlights were successfully placed in the test scene to produce the desired effect.

Research question 2, *Will all surfaces visible in the RSM have one or more associated spotlights?* was also answered. The technique implemented for matching spotlights works in such a way that all reflective spotlights (RSes) in the scene are taken into account, and no parts of the scene that are seen by the RSM are missing reflected light.



(a) Standard spotlight (b) Near clip spotlight

Figure 4.5: Two spotlight types.

To improve the result and thus extend the usefulness of the proposed technique, a different type of light could be used to propagate the light. Point spot lights do not simulate the effects of bouncing light very well because they are points, and bouncing light is reflected by a surface, not a point.

A spotlight could be edited to include a near clip plane as shown in Figure 4.5b and the spotlight could be moved far enough into the emitting surface to make the light start at the surface. The light contribution algorithm would be edited to have its light contribution be based on the amount of light that is emitted from

the surface covered by the light with its current near clip plane and spotlight angle.

Another way of solving the issue of what type of light to use would be to utilize area lights [14]. At that point it would probably be a better idea to not sample the surfaces by rendering them to pixels, but instead retrieve the vertices that are exposed to the sun light from the graphics card. They could then be made into area lights and used for indirect illumination.

The proposed technique is static and not optimized for being updated in real time. Exploring how this technique could be extended to real time is an area for future work. Evaluating other types of data structures for the spotlights, moving the technique to the graphics processing unit (GPU) or doing partial updates would all be good steps in that direction.

The technique could also be extended by adding more than one bounce, where each created aggregate spotlight (AS) would create new spotlights if it is bright enough in a similar way to how it works with the directional light.

References

[1] Y. Tokuyoshi, T. Sekiney, and S. Ogakiz, "Fast global illumination baking via ray-bundles," in *SIGGRAPH Asia 2011 Sketches, SA'11, December 12, 2011 - December 15, 2011*, ser. SIGGRAPH Asia 2011 Sketches, SA'11. Association for Computing Machinery, 2011, pp. ACM Spec. Interest Group Comput. Graph. Interact.; Tech. (SIGGRAPH); ACM Special Interest Group on Computer–Human; Interaction (SIGCHI).

[2] "Lightmap," Aug. 2011, [Accessed 2014-05-26]. [Online]. Available: https://developer.valvesoftware.com/wiki/Lightmap

[3] "Lightmass global illumination," May 2014, [Accessed 2014-05-26]. [Online]. Available: https://docs.unrealengine.com/latest/INT/Engine/Rendering/LightingAndShadows/Lightmass/index.html

[4] "Lightmapping quickstart," Mar. 2013, [Accessed 2014-05-26]. [Online]. Available: http://docs.unity3d.com/Documentation/Manual/Lightmapping.html

[5] "Counter-strike: Global offensive," Aug. 2013, [Accessed 2014-05-26]. [Online]. Available: https://developer.valvesoftware.com/wiki/Counter-Strike:_Global_Offensive

[6] T. Annen, J. Kautz, F. Durand, H.-P. Seidel, A. Keller, and H. W. Jensen, "Spherical harmonic gradients for mid-range illumination," in *Rendering Techniques 2004 : Eurographics Symposium on Rendering*. Eurographics, 2004, pp. 331–336.

[7] A. Kaplanyan and C. Dachsbacher, "Cascaded light propagation volumes for real-time indirect illumination," in *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 2010, p. 99–107. [Online]. Available: http://dl.acm.org/citation.cfm?id=1730821

[8] C. Crassin, F. Neyret, M. Sainz, S. Green, and E. Eisemann, "Interactive indirect illumination using voxel-based cone tracing: An insight," in *ACM Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH'11, August 7, 2011 - August 11, 2011*, ser. ACM

SIGGRAPH 2011 Talks, SIGGRAPH'11. Association for Computing Machinery, 2011, p. ACM Spec. Interest Group Comput.; Graph. Interact. Tech. (SIGGRAPH).

[9] O. Olsson, M. Billeter, and U. Assarsson, "Clustered deferred and forward shading," in *4th ACM SIGGRAPH / Eurographics Symposium on High-Performance Graphics, HPG 2012, June 25, 2012 - June 27, 2012*, ser. High-Performance Graphics 2012, HPG 2012 - ACM SIGGRAPH / Eurographics Symposium Proceedings. Eurographics Association, 2012, pp. 87–96.

[10] C. Dachsbacher and M. Stamminger, "Reflective shadow maps," in *I3D 2005: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games, April 3, 2005 - April 6, 2005*, ser. Proceedings of the Symposium on Interactive 3D Graphics. Association for Computing Machinery, 2005, pp. 203–208.

[11] "Radeon™ HD 7900 series graphics real-time demos." [Online]. Available: http://developer.amd.com/resources/documentation-articles/samples-demos/gpu-demos/amd-radeon-hd-7900-series-graphics-real-time-demos/

[12] T. Harada, J. McKee, and J. C. Yang, "Forward+: Bringing deferred lighting to the next level," *Eurographics*, May 2012.

[13] T. Harada, J. C. Yang, and J. McKee, "Forward+: A step toward film-style shading in real time," in *GPU Pro4 advanced rendering techniques*, W. Engel, Ed. Boca Raton: CRC Press, 2013, pp. 115–135.

[14] J. Arvo, "The irradiance jacobian for partially occluded polyhedral sources," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, p. 343–350. [Online]. Available: http://doi.acm.org/10.1145/192161.192250

Appendix A

Matching technique

The matching technique is an important part of the proposed techinque. It determines whether the data of two spotlights are within a predefined range of each other. If they are, that means they are potential matches when talking about sampling spotlights (SSes). When aggregating spotlights, only matching reflective spotlights (RSes) can be aggregated into an aggregate spotlight (AS).

```
1  bool GlobalIllumination::Matches(const LightSpot &as, const LightSpot &rs) const
       {
2    // Maximum allowed differences.
3    const float max_direction_difference = 0.1f;
4    const float max_color_difference = 0.1f;
5    const float max_depth_difference = 0.1f;
6
7    // Calculate actual differences.
8    float direction_difference = glm::dot(glm::normalize(as.direction()), glm::
         normalize(rs.direction()));
9    float color_difference = glm::length(glm::normalize(glm::vec3(as.color().rgb))
          - glm::normalize(glm::vec3(rs.color().rgb)));
10   glm::vec4 as_to_rs = rs.position() - as.position();
11   float depth_difference = glm::abs( glm::dot(as_to_rs, as.direction()) );
12
13   // Return true if actual differences are lower than maximum differences.
14   return
15     1.0f - direction_difference < max_direction_difference &&
16     color_difference < max_color_difference &&
17     depth_difference < max_depth_difference;
18 }
```

Appendix B

Configuration refinement technique

The proposed technique was implemented in an experimental application written in C++. This appendix contains the code used to optimize the configuration of the sampling spotlights.

```
1  void GlobalIllumination::Simplify(){
2    std::vector<LightSpot> spotlight_samplers;
3    const unsigned int num_spotlight = my_octree_spotlight_ptr->num_light_spots();
4    unsigned int used_spotlight = 0;
5
6    while (used_spotlight < num_spotlight){
7      for (unsigned int i = 0; i < num_spotlight; i++){
8        if (!my_octree_spotlight_ptr->light_spots()[i].used()){
9          // Refine spotlight samplers positioning.
10         LightSpot current_sampler;
11         std::vector<LightSpot*> current_matches;
12         LightSpot test_sampler = my_octree_spotlight_ptr->light_spots()[i];
13         std::vector<LightSpot*> test_matches = GetMatches(test_sampler, *
               my_octree_spotlight_ptr);
14         do{
15           // Set current sampler to the test sampler.
16           current_sampler = test_sampler;
17           // Get the matching spotlights from the current test samplers data.
18           current_matches = test_matches;
19           // Make a new sampler from the aggregate data of all current matches.
20           test_sampler = AggregateSeveralSpotlightData(current_matches);
21           // Get the matches at this new sampling location.
22           test_matches = GetMatches(test_sampler, *my_octree_spotlight_ptr);
23           // Do it agian if we have more matches.
24         } while (test_matches.size() > current_matches.size());
25         // Set final spotlight sampler configuration.
26         current_sampler = AggregateSeveralSpotlightData(current_matches);
27         // Add the current spotlight sampler to the list.
28         spotlight_samplers.push_back(current_sampler);
29         // Mark all spotlights that match this spotlight sampler as used.
30         for (LightSpot* spot_ptr : current_matches){
31           spot_ptr->set_used(true);
32           used_spotlight++;
33         }
34         break;
35       }
36     }
37   }
38
39   // Aggregate spotlights using the resulting spotlight samplers.
40   AggregateSpotlights(spotlight_samplers);
41 }
```