CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Application of Bump-effects in a Post-processing step

Master's thesis in Computer science and engineering

Anthony Tao
Marcus Gideflod

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Application of Bump-effects in a Post-processing step

Anthony Tao
Marcus Gideflod

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Application of Bump-effects in a Post-processing step
Anthony Tao, Marcus Gideflod

Supervisor: Erik Sintorn, Department of Computer Science and Engineering
Advisor: Mikael Engbom, Johnny Johansson, Spark Vision
Examiner: Ulf Assarson, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Application of Bump-effects in a Post-processing step
Anthony Tao, Marcus Gideflod
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

With the rise of internet shopping, the importance of Computer-generated Imagery is rapidly increasing. Visualizations help companies sell their product, especially customizable ones such as bathroom interiors. Such applications must provide images with a degree of realism that accurately represents the real features of the product, but also be able to produce them within an acceptable time frame.

A common way of producing realistic visualizations of products is using path-tracing, but for interactive applications, the technique is overly time-consuming. Still, it can function well as an offline method by pre-rendering images to be used on demand. However, with customizable products it is not a feasible solution as the permutations would quickly lead to an unimaginable amount of images.

This thesis focuses on customizing bumps on the surface of walls and applying it as a post-process to avoid the permutation issue. The 3D scene will not be available during run-time, meaning information has to be stored beforehand. The proposed method simply accepts a bump map and computes the new shading as a result of new normals. The method only considers direct lighting where the camera position is static, but includes realistic soft shadows from static geometry. The resulting images' visual quality comes close to a ray-traced image, but without further optimizations the method does not meet the required render time for larger scenes. The reasons for this are discussed, and ideas for improvement are suggested.

Keywords: Computer graphics, Master thesis, Post-process, Shadow masks, Bump effects, Soft shadows.

Acknowledgements

We would like to thank our supervisor Erik Sintorn for the time he spent providing us guidance and feedback that have played a crucial role over the course of this thesis. We would also like to thank Spark Vision and our advisors Mikael Engbom and Johnny Johansson for the opportunity to work with the company and all help they provided by setting up a work environment and guiding us in the development.

Anthony Tao, Marcus Gideflod, Gothenburg, June 2022

Contents

List of Figures

List of Tables

1
Introduction

Computer-generated Imagery is rapidly becoming the standard for visualizing products due to its many advantages. Compared to photography, utilizing digital rendering tools requires less time and resources as nowadays photo-realistic images can be rendered relatively easy and quickly. When it comes to online shopping of customizable products, real-time applications have been developed that allow the customer to quickly plan, customize and visualize the product. One example of this is Scania's truck builder [20]. The visualized product can be anything customizable, from the interior of a bathroom to different car designs. For such an application, it is essential to provide visually satisfactory images of the product that accurately portrays its real features and appearance. Equally important is optimizing for interactive response times, which generally lie within the span of 10 to 1000 milliseconds.

To produce digital visualizations of products the most common method is utilizing 3D Computer Graphics, where a 3D model of the product is created and rendered to an image. In cases where photo-realistic images are required, an accurate method is rendering using path-tracing. However, most often quality comes at the cost of performance and this also applies to path-tracing. In general, path-tracing is too time-consuming to work well in an interactive application, therefore it is mostly used as an offline method, where the images are rendered offline and provided on demand. With customization to consider however, this method becomes unfeasible since there is often an extensive number of customizable features in a product, and simply 10 different features may result in millions, up to infinite amount of permutations. Thus, no matter the amount or quality of hardware, it is not practical to pre-render images when considering the total render time, memory usage and scalability. As such, ways to provide customization after the scene is rendered, during what we call post-processing, need to be explored.

The focus of this thesis lies on customizing bumps on the surface of walls, and researching ways to do this as a post-process. This thesis is done in collaboration with Spark Vision, who require a method for applying bump patterns as a customization on photo-realistic bathroom walls visualized using Computer Graphics. Figure 1.1 illustrates an example of bump effects as highlights on the sides of the wall tiles. Bumps are important as they provide a sense of depth on surfaces through shadows and reflections, adding to the realism of the image. They are typically applied when rendering the images, so if bumps are a customizable feature as in the case of Spark Vision's application, pre-rendering the images with all permutations of bumps applied would have several disadvantages, as previously explained.

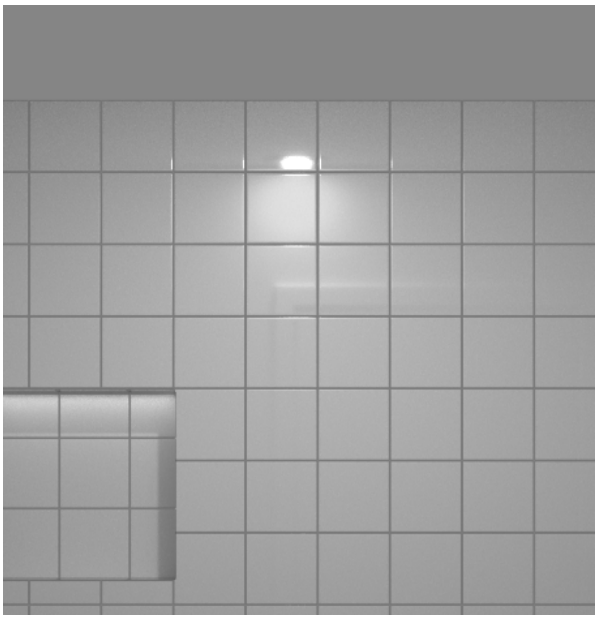


Figure 1.1: Example of ray-traced bathroom wall with bumps pre-applied

1.1 Problem Definition

The setting in Spark Vision's application differs from usual Computer Graphics problems as it is static in camera position, geometry and light sources but has configurable materials and bumps. Furthermore the application of bumps needs to be done as a post-process, which means that the 3D scene is not available. I.e., the information necessary for computing bumps has to be stored offline and then brought to use at run-time. In the following subsections, we identify the two main challenges and finish with summarizing the limitations and delimitations of the thesis.

1.1.1 Realistic Bump Shading

When applying bump effects to a scene, what is actually computed is a new shading caused by a change in the normals of a surface. For this thesis, the new shading should be comparable to the shading of the ray-traced base image. This means that shadows need to be soft where surfaces are partially visible to the area light, and the shape of the reflections should depend on the shape of the area light. We separate this problem into two parts where one is finding a way to evaluate the visibility correctly to produce soft shadows, and the other is to calculate the lighting accurately to achieve realistic reflections.

Compared to video games that require rendering frames in real-time, this application has more leeway as the image only needs to be rendered once when a bump pattern has been changed. Nevertheless, the application needs to deliver the image fast enough to preserve a satisfactory user experience. According to Spark Vision's goals, satisfactory is defined as under 200ms, when considering only the bump computation. To reach this goal, we simplify the performance requirements by excluding indirect lighting for this thesis as we deem it too costly in terms of performance, and difficult to solve given time limits.

1.1.2 Storing Essential Information

In order to produce any shading at all, firstly geometry and light information are required according to any shading model, for example Blinn-Phong reflection model. Such information include light color, incident angle, visibility, surface normals, and material attributes [1]. This information is necessary for every pixel of the surface in the scene that needs to be re-shaded according to new bumps, and we will refer to it as "per-pixel data" moving further. Since there is no 3D scene to work with during post-processing, the information needs to be stored into memory when they are available, i.e. offline.

This leads us into the next problem; storing and transferring the per-pixel data may lead to high memory consumption, so depending on the performance, ways to optimize the information stored will be necessary. One concern arises when multiple lights are present since the size of the information increases with the amount of light sources taken into account. Computing reflections for a single light source will only require information from that one light, but will not produce correct shading if multiple light sources are present. On the other hand including all light sources would be more accurate but the trade-off is additional memory usage. Furthermore, the memory consumption scales with the per-pixel data resolution. Since aliasing effects will occur if we use per-pixel data that has too low resolution, additional memory optimizations might be required to accommodate for a per-pixel data resolution that successfully removes all aliasing effects.

1.1.3 Limitations and Delimitations

For our specific problem, there are important limitations and delimitations that the reader should keep in mind. Firstly, the hardware and software used in this thesis were pre-determined due to the collaboration with Spark Vision, and these are:

- 3D Studio Max (3ds Max), used to create the scene used in the application
- Redshift 3D, the offline render engine for rendering the scene created in 3D Studio MAX and outputting per-pixel data
- Nvidia Geforce 1080 Ti with 11 GB GDDR5X memory and 11 Gbps memory bandwidth [16]

Furthermore, as delimitations in our method we assume that:

- the camera is static at all times, i.e. no camera movement or rotation
- only 2 shapes of area lights are considered; discs and rectangles
- indirect lighting is not included
- the furniture in the scene are static, i.e. reflections and shadows of furniture do not change
- the surface of the objects with bumps applied are only flat in our test environment, but presumably, the method should be able to handle non-flat objects
- generating the bump-maps is not a part of the method, they are given as input to the algorithm

1.2 Contributions

The main contribution of this thesis is a method for computing realistic bump effects as a fast post-process. In more specific terms it is able to, without a 3D scene, compute the realistic reflections and soft shadows that results from a change in the normals of a surface. This means the method can compute not only bumps, but also a whole static scene with dynamic material and bumps using only per-pixel data. At the time of writing we have found no similar method available, thus our main contribution is knowledge about the potential such a method has, especially for applications that require configurable materials and bumps given the absence of a 3D scene at run-time. In order to accommodate for the large memory consumption of per-pixel data, useful optimization techniques are presented and implemented in the Method chapter. Additionally for both practical and research purposes, we perform benchmarks on how different quality settings affect performance, which are presented in the Results section. To summarize, the contributions of this thesis are:

- a method to apply realistic bump effects to surfaces in a ray-traced image by inputting a bump map (given available per-pixel data)
- a method capable of reconstructing a static scene with dynamic material and bumps. Only direct lighting is considered, but it includes realistic soft shadows from static geometry by using stored per-pixel data
- a presentation of available optimization techniques for minimizing the compute time and memory usage of the method
- a benchmark on how the optimized method performs on a GTX 1080Ti

2
Related Work

This chapter presents reviewed literature which attempt to solve individual problems such as shading and visibility, or a problem similar to ours as a whole. Both have provided understanding and inspiration in solving the different problems in this thesis.

2.1 Illumination Techniques

Nowadays there are many viable global illumination algorithms that can produce realistic shading from area lights. Some of the most prominent ones in terms of quality are ray-tracing and path-tracing, which intersect rays with the scene geometry to compute photo-realistic reflections and shadows [7]. For more real-time friendly techniques, many-light methods are developed with the purpose of efficient rendering of scenes that contain a great amount of light sources, facilitating scalability within the rendering process. *Scalable Realistic Rendering with Many-Light Methods* by C. Dachsbacher surveys methods for efficiently computing global illumination [4]. The survey covers techniques that include the use of Virtual Point Lights (VPLs) and shadow mapping.

Instant Radiosity by A. Keller uses Virtual Point Lights to approximate global illumination. The algorithm is performed in two passes, first by tracing rays throughout the scene and placing VPLs on the surfaces hit. These are then used in a second pass to compute shading and shadows by e.g. shadow mapping. Instant Radiosity is applied directly onto textured scene surfaces and is combined with graphics hardware to produce photo-realistic images [14]. The Lightcuts method later developed by B. Walter collects a large number of VPLs that represents indirect illumination and constructs a hierarchy that helps speed up the rendering process [23].

Photon mapping is a method which sends out photons from the light sources to construct photon maps, which store the information provided from photons hitting surfaces in the scene [12]. Using photons, it is possible to estimate the surface radiance with the help of arbitrary BRDFs. Progressive photon mapping is a multi-pass algorithm based on the photon mapping algorithm [9]. This version is built upon multiple passes and the first consists of ray-tracing followed by photon-tracing passes.

Light Field Rendering by M. Levoy creates light fields from images that in later stages can be used to produce both inward and outward views, all done through

the sampling and combining of existing images [15]. *Light Field Mapping* is another technique which makes use of light fields that are parameterized on a surface. This is done in real-time due to a compact representation that is suitable for the graphics hardware pipeline [3].

Heitz et al. developed a method for computing realistic shading of area lights by transforming the physically based BRDFs to what they call Linearly Transformed Cosines (LTC). By approximating the BRDF as LTC, the lighting can be analytically integrated over arbitrary spherical polygons in real-time [10]. However, the method is limited to direct lighting and does not consider shadows, thus for it to be viable in the context of this thesis, an extension of the method to include shadows will be required.

2.2 Shadow Mapping

In *Casting Curved Shadows on Curved Surfaces*, L. Williams presents a method for rendering shadows using values stored in the depth buffer from the observer and light source, i.e. Shadow maps [24]. The observer's depth values are compared to the Shadow map, producing accurate shadows. The Imperfect Shadow Maps (ISM) method by T. Ritschel renders low resolution Shadow Maps (SMs) from VPLs in the scene. This technique can be utilized as a viable approximation of visibility in a global illumination technique using VPLs [19]. *Perspective Shadow Maps* introduced by M. Stamminger are SMs generated in a normalized device coordinate space with the possibility of reducing SM aliasing [22]. *Variance Shadow Maps* by W. Donnelly aims to solve the problem of aliasing effects that may occur within SMs due to the fact that they cannot be filtered in the same way as color textures [6]. This technique stores the mean and mean squared of a distribution of depths instead of a single depth value. *Moment Shadow Maps by C. Peters* is similar to Variance Shadow Maps, but in this technique four moments of the depth is stored in the kernel which are then used to compute a lower bound as approximation of the shadow intensity [17]. *Reflective Shadow Maps* by C. Dachsbacher is an extension of SMs that interprets every pixel as an indirect light source, which are sampled in order to calculate the light contribution [5].

2.3 Relighting Techniques

What this thesis tries to achieve is essentially a relighting of the scene according to dynamic normals, therefore different relighting papers were reviewed. The paper *A Fast Relighting Engine for Interactive Cinematic Lighting Design* presents a method that achieves dynamic lighting in a scene by storing per-pixel data into what they call deep-framebuffers. The deep-framebuffers consist of both geometric and optical information. The geometry information include pixel world position, visibility and normals, and the optical information describe the Bidirectional Reflectance Distribution Function (BRDF) of the surfaces. These are generated as a pre-computation pass where they also partially compute the shading. In a second pass, they provide a set of lights which can be interactively positioned. When this is done, the third pass renders the image by computing the new shading using the deep-framebuffers and

newly positioned lights [8]. Although the setting is similar to this thesis problem, the method does not take into account that the surface normals can be dynamic, nor does it produce sufficiently soft shadows. In our problem, the BRDF can not be pre-computed since the normals need to be dynamic. Still, inspiration can be drawn from how they store and utilize the geometry information.

Another paper, *Image based Relighting using Neural Networks* presents a neural network regression method that is able to relight real world scenes using only a small amount of images for training [18]. These images are taken on the scene from the same view point, but with different lighting conditions. After training using the images, a light transport matrix can be reconstructed and a scene with dynamically positioned lights can be rendered. The result preserves global illumination and soft shadows well, however since material information is missing, it does not properly model high variation in material properties. Despite the paper focusing on dynamic lighting and only training the network based on captured images, an interesting idea for our specific problem is to train using path-traced scenes with different surface material properties rather than lighting conditions.

3

Theory

This chapter presents the theoretical background to physically based rendering and shadows in Computer Graphics. Furthermore, the theory behind producing bump effects with bump-mapping, and dealing with aliasing effects using anti-aliasing techniques are described. Finally, an explanation on how large textures can be stored efficiently using Sparse Virtual Textures is provided.

3.1 Global Illumination

In general when an image is recorded from a scene, what results in visible colors is the total reflected light from the surfaces towards the camera or eye. In Computer Graphics, the Rendering Equation, developed by J. Kajiya, models the equilibrium radiance that is reflected from a surface in the direction of the camera [13]. The Rendering Equation is an integral over the directions on a hemisphere defined as:

$$L_o(p, \omega_o) = L_e(p) + \int_\Omega f(p, \omega_o, \omega_i) L_i(p, \omega_i) |cos\theta_i| d\omega_i \tag{3.1}$$

where p is a point on a surface, L_o is the light leaving the point towards the camera, i.e. the displayed color, and Ω is the unit hemisphere around the surface normal at the point. L_e is self-emitted radiance and f is the BRDF, describing how the light reflects off an opaque material. L_i is the incident light at the point and $|cos\theta_i|$ is an attenuation on the incoming light depending on the angle at which the light hits the point.

In Computer Graphics, global illumination refers to a realistic simulation of light in the scene, where all or most lighting effects are considered. The algorithms that model global illumination typically include effects such as bounce light, meaning the effect of light that bounces or reflects off multiple surfaces, and light transmission, the result of light that refracts through a surface. Bounce light is visualized in Figure 3.1, where the arrows illustrate a light ray reflecting around in the scene before it reaches the camera, producing the color for a single pixel. In the Rendering Equation (3.1), this effect is accumulated in the incident light term (L_i), which represents incoming light from all directions. Meanwhile, direct illumination refers to only the light originating from a light source which hits a surface and bounces directly into the camera. Thus when computing direct lighting, the L_i term only includes light from light sources. Direct illumination techniques are often significantly less complex and demanding in terms of performance.

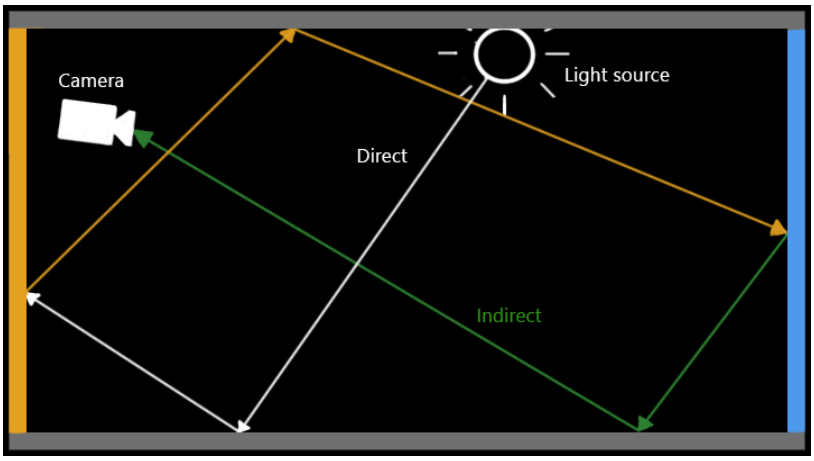


Figure 3.1: Illustration of bounce light

3.2 Light Sources

Light sources are a vital part of Computer Graphics, and as the name suggests they are the source of light in the scene. There is a wide range of different light sources such as point lights, spotlights, area lights and many more in different shapes, sizes and functionality. The characteristics of a point light source is that it is infinitesimally small and without a physical size. A directional light source is positioned at infinity making the incoming light rays parallel to each other, as for example the sun. Area light sources on the other hand, have a physical size that can be shaped in any geometrical form. One of the most common representations of an area light is a flat square where each part of the surface distributes light. Calculating direct lighting from a point light is straight-forward since there is only a single direction to consider, but with area lights there are multiple directions from which light might arrive. Because of this, a crucial difference between area lights and point lights is the resulting soft shadows from area lights instead of the sharp edges created between shadowed and lit areas by point lights. Shadows will be explained further in Section 3.5.

3.2.1 Computing Direct Lighting from Area Lights

To compute the direct lighting from area lights, a method which might be preferable is to calculate the lighting as an integration over area instead. For this to be possible, a modification to the Rendering Equation (3.1) is required. Since the Rendering Equation defined previously is an integral over directions, it needs to be changed by converting solid angle to area according to:

$$d\omega = dA\frac{|cos\theta'|}{d^2} \tag{3.2}$$

where $|cos\theta|/d^2$ is the Jacobian relating solid angle to area ($d\omega$ and dA). Figure 3.2 illustrates this relation between solid angle and area. In the figure, d is the distance between the shading point p' and a point on the light surface p'', n is the light surface normal and θ' is the angle between the normal and incoming light direction towards p'.

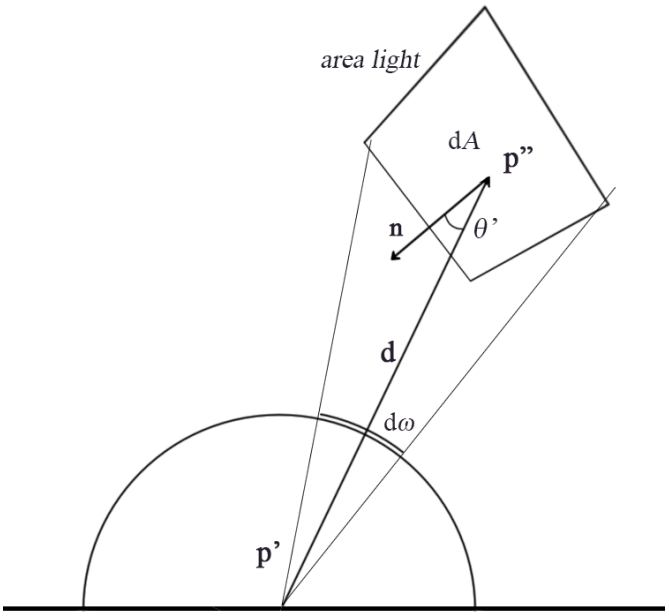


Figure 3.2: Illustration of relation between solid angle and area

Combining this conversion with the attenuation term $|cos\theta|$ and a binary visibility term V which determines mutual visibility between p' and p'', produces the geometric coupling term:

$$G(p' \leftrightarrow p'') = V(p' \leftrightarrow p'')\frac{|cos\theta||cos\theta'|}{d^2} \tag{3.3}$$

Ways to evaluate the visibility will be discussed later, in Section 3.5.1. For now, applying this modification to the Rendering Equation gives us the equation in surface form:

$$L_o(p' \to p) = L_e(p' \to p) + \int_A f(p'' \to p' \to p)L_i(p'' \to p')G(p' \leftrightarrow p'')dA(p'') \tag{3.4}$$

where p is the point where the outgoing light hits, e.g. the camera. Since performing an analytical integration of such a complex integrand is not possible, another way to evaluate it is with numerical integration, i.e. Monte Carlo Integration. The Monte Carlo estimator is defined in Equation 3.5. This thesis excludes self-emitting surfaces and because of that the emission term is taken out from the equation.

$$L_o(p' \to p) \approx \frac{1}{N}\sum_{p''}^{N}\frac{1}{pdf}f(p'' \to p' \to p)L_i(p'' \to p')G(p' \leftrightarrow p'') \tag{3.5}$$

To estimate the integral, firstly N amount of point samples are stochastically taken on the light surface. The estimate is then calculated as the mean of the accumulated point samples' outgoing light divided by the probability distribution function (PDF) for sampling an area light. When sampling area lights uniformly, the PDF is $1/A$, where A is the area of the light. Since sampling stochastically may result in clustered samples, to improve the distribution of the samples, stratified sampling can be used. Stratified sampling works by dividing the area light into parts called stratum, e.g. a rectangle into a grid, and stochastically drawing samples within each stratum. This way, each part of the area light is guaranteed to contain samples.

3.3 The Torrance-Sparrow BRDF

In physically based rendering, the f term of the Rendering Equation (3.1) is a particularly important factor. It is, as mentioned, the Bidirectional Reflectance Distribution Function, which models how light reflects at a surface of opaque material. Specifically, it models the ratio of outgoing radiance to incident radiance. For a Lambertian surface, i.e. a perfectly diffuse surface, the light reflects equally at all directions over the hemisphere, thus the BRDF is simply a constant: ρ/π. ρ is the color of the material, modelling the absorption of the incident light. To achieve photo-realism however, glossy materials that reflect light more towards a certain direction are also required, and these are often achieved using physically based BRDF models. One such model is the Torrance-Sparrow microfacet model, defined as:

$$BRDF = \frac{F(\omega_i)D(\omega_h)G(\omega_i, \omega_o)}{4(n \cdot \omega_o)(n \cdot \omega_i)} \tag{3.6}$$

where ω_i, ω_o and ω_h are the incident, outgoing and half angle vector, a vector pointing at the direction between ω_i and ω_o. Microfacet models can represent different materials by modelling the surface of the material as a collection of microfacets, where the combined properties of the microfacets determine the scattering of light at the surface. The properties are determined statistically with a distribution function D and masking function G, while F defines the ratio of light reflected by each microfacet. Figure 3.3 illustrates a surface with microfacets.

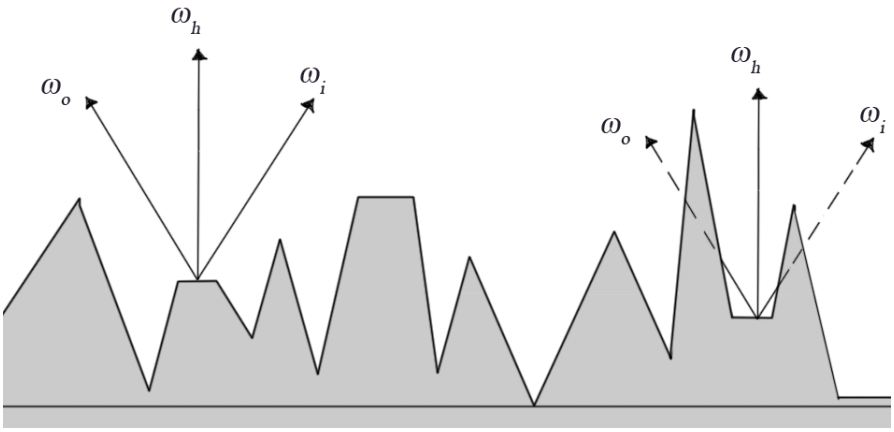


Figure 3.3: Illustration of microfacets
A surface composed of small microfacets. Incoming and outgoing light might be occluded by other microfacets, depicted to the right.

In the Torrance-Sparrow model, F is the Fresnel term and models the effect where light is reflected in a mirror-like way when incident at grazing angles. In this thesis an approximation of the full Fresnel equations developed by C. Schlick is used [21]. The term is defined in Equation 3.7, where R_0 is a reflection coefficient dependent on the indices of refraction of the two media that the light interacts with. In implementations, the term is a modifiable material parameter.

$$F(\omega_i) = R_0 + (1 - R_0)(1 - w_h \cdot \omega_i)^5 \tag{3.7}$$

The D term of the model is the Microfacet Distribution Function, which defines the distribution of only the microfacets with the normal ω_h. In the model, the microfacets are considered as perfect mirrors, thus only the microfacets with a normal ω_h will reflect light towards the camera. Therefore, the higher the variation of microfacet normals determined by D, the rougher the surface becomes and vice versa. In this thesis the normalized Blinn-Phong function is used, and is shown in Equation 3.8 where s is the material shininess.

$$D(\omega_h) = \frac{(s+2)}{2\pi}(n \cdot \omega_h)^s \tag{3.8}$$

The view or light directions may be close to grazing angles, which results in the incoming or outgoing light of some microfacets being occluded by adjacent microfacets. This is illustrated to the right in Figure 3.3. The masking function G models the fraction of microfacets that are visible from both directions according to the following equation:

$$G(\omega_i, \omega_o) = min(1, min(2\frac{(n \cdot \omega_h)(n \cdot \omega_o)}{\omega_o \cdot \omega_h}, 2\frac{(n \cdot \omega_h)(n \cdot \omega_i)}{w_o \cdot \omega_h})) \tag{3.9}$$

3.4 Bump Mapping

Bump-mapping denotes a collection of techniques which all aim to visualize three-dimensional details on objects without altering or adding to the geometry. The details are created by adding a perturbation to the surface's normal based on a texture map, causing the light to reflect in a different direction from the surface. This results in shadows and highlights in new areas. One of the first versions of this technique was developed in 1978 by James F. Blinn where a height map is used to derive perturbations of the normals, typically by the finite difference method [2]. The perturbations are applied to the surface normal with the goal of generating images with microscopic wrinkles among similar effects. Making use of texture maps instead of altering the geometry results in equivalent levels of detail, while also computing faster and consuming less resources. A disadvantage is that the method can not handle bumps that protrude out of the geometry bounds, however that is not required in the context of this thesis.

The bump-mapping technique that will be used in this thesis project is called normal mapping. To describe the bump pattern, normal mapping stores the normal perturbation directly in a normal map, which is an RGB texture. Each texel is an RGB value where the red, green and blue components determine how much the normal should be perturbed in the X, Y and Z coordinates respectively. Figure 3.4 shows an example of how a normal map looks.

3.4.1 The TBN Matrix

In a normal map, the normals are defined in a local coordinate space called the tangent space, and are assumed to be oriented toward the positive Z-direction. Since the normal map is in tangent space, to calculate the perturbation of a surface normal that is pointing in another direction, the perturbation first needs to be

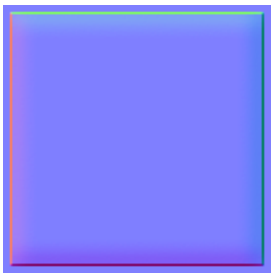


Figure 3.4: Illustration of a normal map representing a tile

transformed to align with that direction. This is where the Tangent, Bi-tangent and Normal vectors that combine into a TBN matrix are helpful, see Figure 3.5 for an illustration of the three vectors.

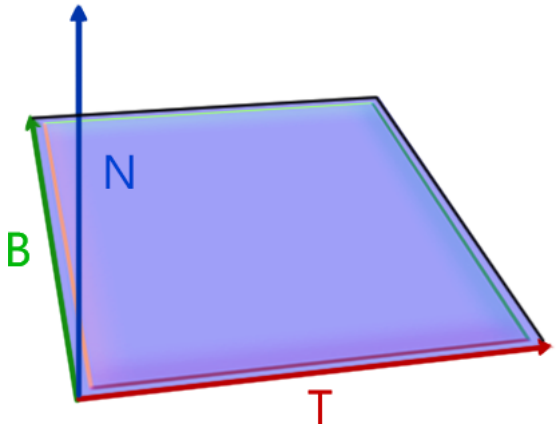


Figure 3.5: Illustration of TBN vectors
The vector N represents the normal of the surface, T the tangent and B the bi-tangent.

The TBN matrix, when multiplied with the normal in tangent space will transform it to the coordinate space that the matrix is constructed of. To construct the TBN matrix, firstly the bi-tangent and tangent vector need to be computed. The normal should already be known, however the tangent and bi-tangent need to be derived from the texture coordinates of the surface. Alternatively, only one of the bi-tangent or tangent vectors need to be computed, and the third vector can be computed as a cross product of the normal and the second vector. Figure 3.6 shows how the tangent and bi-tangent vectors are oriented relative the geometry and its UV texture coordinates.

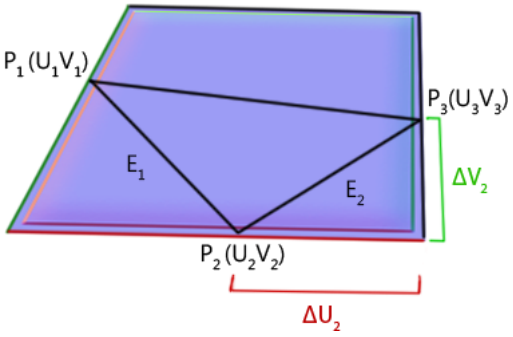


Figure 3.6: Illustration of a triangle and its tangent and bi-tangent orientation
The tangent (red) and bi-tangent (green) vectors are aligned with the U and V axes.

Calculating the bi-tangent and tangent vectors requires solving two equations. Assuming the vectors are to be computed from a triangle as in Figure 3.6, the tangent and bi-tangent are aligned with the U and V coordinates respectively, which means the edges E1 and E2 can be expressed as a linear combination of the two vectors:

$$\begin{aligned} E1 &= \Delta U_1 T + \Delta V_1 B \\ E2 &= \Delta U_2 T + \Delta V_2 B \end{aligned} \tag{3.10}$$

An easy way to solve the equations is to instead write them in matrix multiplication form:

$$\begin{bmatrix} E_1x & E_1y & E_1z \\ E_2x & E_2y & E_2z \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} \tag{3.11}$$

and then solve for the T and B vectors by multiplying both sides with the inverse of the delta UV matrix. After all vectors are computed, they can be composed into a TBN matrix as following:

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \\ N_x & N_y & N_z \end{bmatrix}$$

3.5 Shadows

Shadows are useful in distinguishing movement and depth of objects in a scene, making them an essential visual effect when rendering realistic images. When evaluating shadow quality, the focus is mostly on the edges of the shadow that can be either hard or soft. Figure 3.7 illustrates how soft shadows are created when using an area light. With area light sources, there may be parts of the scene that are not visible to every part of the light source, resulting in an area called the penumbra. The region from where the light source does not reach at all is called umbra. As the distance increases between the light source and the occluder, the light source gradually behaves more like a point light and the shadow edges become harder. Conversely, as the distance decreases, the shadows become softer. To achieve as

realistic shading as possible with area lights, soft shadows should have a seamless, gradient-like transition from shadow to light in the penumbra.

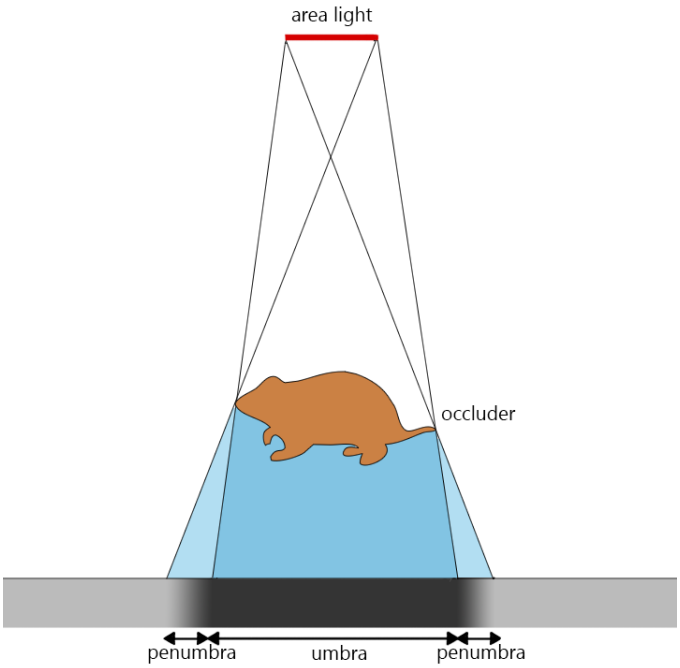


Figure 3.7: Illustration of soft shadows created by an area light

3.5.1 Evaluating Visibility

Most methods that compute shadows rely on evaluating the visibility from the shading point to the light source. Evaluating visibility can be done in several ways; one method is sending shadow rays from the point to a light source and checking whether it hits an occluder in the way or not. If it intersects with an occluder, the point is not lit by that specific light source. For computing shadows from point lights, a more efficient technique is Shadow Mapping, which is based on depth comparisons. Firstly, the depth of every pixel is rendered from the perspective of the light where the result is stored in a texture called a Shadow Map (SM). Next, the depth of a pixel as viewed from the camera, whose visibility is currently being evaluated, is transformed into the same coordinate space as the light source. When determining visibility, that pixel depth is compared to the depth stored in the SM. If the SM depth is less, then the point is occluded and considered to be in shadow. Otherwise it is visible from the light source and thus lit. This technique is fast because it eliminates the need of sending and intersecting shadow rays from each pixel in the scene. Instead, only the creation of the SM for every frame and a look-up in the SM including a single comparison for every pixel is needed. However, depending on the SM resolution and amount of lights in the scene, rendering and storing several SMs can be heavily demanding on performance.

3.5.2 Computation of Soft Shadows

Computing soft shadows is a non-trivial task, as the visibility from the area light needs to be evaluated for the whole surface of the light. When using shadow rays

to evaluate visibility, realistic soft shadows can be produced by shooting rays to sampled points on the light, where only samples that are unoccluded contribute with light. A similar result can be produced with Shadow Mapping, by rendering Shadow Maps from multiple sampled points on the area light. However, as just mentioned, rendering and evaluating many SMs can in total be very costly in terms of performance. One way to deal with this is explained in the paper *Imperfect Shadow Maps for Efficient Computation of Indirect Illumination*, which presents a method that uses low resolution SMs ($256x256$), capable of producing and evaluating 512 Shadow Maps at interactive frame rates [19]. By approximating an area light with 512 point lights, it is indeed possible to produce soft shadows. The trade-off when such low resolution SMs are used however, is that the overall quality of the shadows will not be as good when compared to tracing shadow rays. Especially obvious are the umbra regions, where the shapes are inaccurate and clearly in low resolution.

3.6 Anti-Aliasing

When rendering an image that contains rapid color changes, in for example shadow boundaries or edges between geometry, unwanted artifacts called aliasing effects may appear. Shown at the left of Figure 3.8 is the sampling of a signal, a real-world object described by continuous curves. When sampling the signal with insufficiently high frequency, referred to as under-sampling, it can lead to the situation on the right of the figure, where the edges appear as jagged. Looking closer at the left image, each pixel (square) has a dot in the middle representing the sampling point. The sampling point determines the color of the entire pixel, but as can be seen, the pixels at the color edges contain mixed black and orange in the signal. Since only one sample is used to determine the pixel color, the mix of colors is not taken into consideration, and jagged edges are the result. Another possible effect of aliasing shown in the figure, is that small details in the signal such as thin lines may be missing if the sampling is too low.

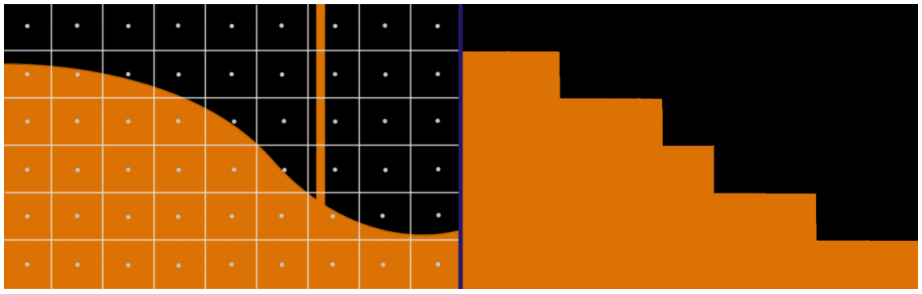


Figure 3.8: Illustration of the aliasing problem

One way to solve aliasing effects is sampling multiple times within a pixel. Super-sampling solves the problem by rendering the image at a higher resolution, and then down-sampling back to the original resolution by averaging the sampled colors. Averaging the colors results in smooth looking transitions between edges, which eliminates the jagged effect.

3.7 Sparse Virtual Textures

With higher resolution and an increased level of detail, the size of textures continues to increase to the point that they may be too large to fit in the GPU memory. Sparse Virtual Textures (SVT) is a technique that can be used when this problem occurs. This technique works in the same way as virtual memory by using both software and hardware to operate. Data are stored in a physical address using RAM and then a memory management unit can map the address to RAM. If at some point the space in RAM is needed for something that is more urgent it is possible to swap data from RAM into virtual memory. When a swap is happening, the operating system divides memory with a fixed number of addresses into page files. All pages are stored on disk and when a specific page is needed the OS copies the page into main memory and translates the virtual address into a real address. By dividing a texture into parts called tiles, only loading the relevant tiles into memory, it essentially works in the same way as a virtual memory paging. In practice, this functions by loading e.g. only the functional parts of a texture into pages in the GPU cache and storing the rest that are not currently used in system memory. The pages that have been loaded are made into a Sparse Virtual Texture in which texture tiles can be swapped out with another that is stored in system memory according to a memory management policy chosen by the user.

4
Method

The implementation of our solution is written in C++ and utilizes the GPU for shading. The goal with the final implementation is being able to produce highlights and shadows from direct light that are comparable to path-traced references, while keeping the render time under 200 ms. In this chapter we describe our solution and implementation in detail, beginning with a general overview of the solution to the problems mentioned in Section 1.1.

4.1 Solution Overview

There are numerous solutions of different complexity to the shading problem. For our base-line we decided on a brute-force but efficient method; approximating the area lights with a set amount of light samples. N positions are sampled on the area lights, a point light is placed on these positions to pre-compute visibility, and direct lighting is estimated using Monte Carlo Integration. To pre-compute visibility, we offline render N binary images that define which pixels on the scene are visible to a point light or not, which we refer to as "shadow masks". The idea is that with sufficient amount of light samples, this will result in accurate soft shadows.

When rendering images with the same per-pixel data resolution as the displayed image, we immediately realized there was aliasing present in both normals and shadows. The most efficient solution in terms of quality is to super-sample the whole scene, i.e. increase the resolution of all per-pixel data, as this way all the aliasing effects will be solved. However the trade-off is a proportional increase of memory. As an example, if each pixel is super-sampled by 4x4 the memory usage increases 16 times. In addition, there is increased computation time as we perform pixel shading for 16 times the amount of pixels. What magnitude of super-sampling to use is determined by increasing the sampling and evaluating primarily the image quality. The super-sampling level that removes all aliasing effects and has the least low performance impact is chosen.

With the additional increase of memory due to super-sampling, memory optimization becomes even more of a necessity to avoid bandwidth limiting the application. For the pixel world positions, normals and tangents information, we halve the data by reducing the floating point precision from 32-bit to 16-bit, as with lower precision the difference in the light incident angles are minimal and will not affect the shading much. To further minimize memory usage, we have utilized Sparse Virtual Textures for the shadow masks, as described in Section 3.7.

4.2 Implementation of Solution

The implementation of this method is separated into two major stages, preparation stage and post-processing stage. The first stage refers to what is performed offline; storing and processing of output data from 3ds Max and Redshift. The second stage is the post-processing stage where the image is rendered at run-time. This stage applies the updated shading to the image as material and bumps change. Both of these stages will be thoroughly explained in this section, including the steps performed for optimization, and measures taken to achieve a satisfactory visual quality. Before that, we describe the scene used in the implementation.

4.2.1 The Scene

The implementation tests are based on two scenes built in 3ds Max. In the simplest case the scene is 808×1080 in resolution and only contains one disc shaped area light source, but for the benchmark we also render a bigger scene (1874×1080) with more lights, including rectangular lights. However, for simplicity this section will only illustrate implementation of the small scene. The small scene contains mainly 4 surfaces; a left, back and right wall and one floor that during run-time can change materials and bumps. The scene also contains furniture that can be found inside a bathroom, i.e a toilet, washing stand and mirror. All of the per-pixel data exist in the 3ds Max scene, and what these data consist of and how they are prepared will be explained next.

4.2.2 Preparation Stage

The preparation stage as mentioned is offline, where time is not limited. The first step is to sample the light source. Using 3ds MAXScript, the built-in scripting language for 3ds Max, we stochastically sample the surface of the light source N times and acquire positions where a light sample can be. The amount of samples taken is a variable that has significant impact on the end result, as higher number of samples produces more accurate shading but consumes more memory and increases computation time in the post-processing stage. Stratified sampling, as described in Section 3.2.1, is used to distribute the samples more evenly over the light source surface. See Figure 4.1 for an example of 64 sampled positions.

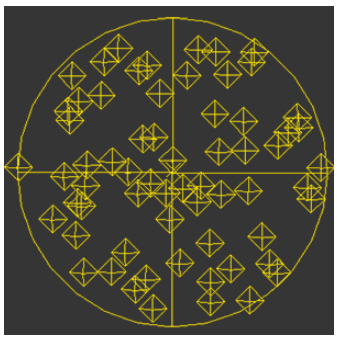


Figure 4.1: Illustration of 64 sampled positions on a disc light source
The big circle is the light source and the diamond shaped points are sampled positions.

After sampling the light source, we gather the per-pixel data and vectors necessary in the post-processing stage. By using MAXScript we can write the camera position and each light sample position, including the light's area and normal to a txt file. In Redshift, a functionality called Attributed Output Variable (AOV) provides the possibility of specific render outputs. World Position AOV is used to render an image that for each pixel contains the world space position in RGB float values, shown in figure 4.2. The same procedure is used to extract per-pixel normals and tangents. The method to compute tangents is as described in section 3.4.1. These per-pixel data contain all values needed for calculating the incoming light (w_i), outgoing light (w_o) and normal vectors (n) for each pixel. A UV map, i.e. texture that ranges from 0 to 1 in U and V coordinates for every surface, is illustrated in Figure 4.3. It is used to project normal map texels to pixel coordinates in the scene. In this thesis they were already rendered and provided by Spark Vision. A UV map for each dynamic surface in the scene was provided, and the only requirement was to manually compose them into a single UV map.

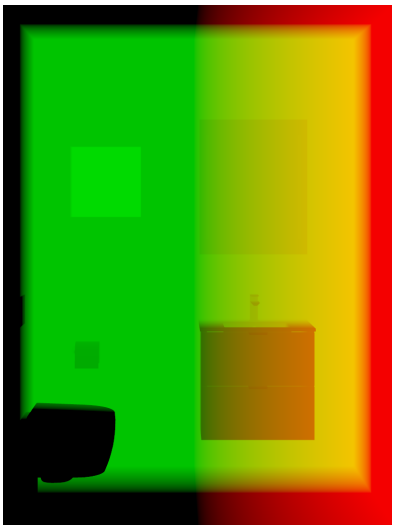


Figure 4.2: Illustration of the per-pixel world positions

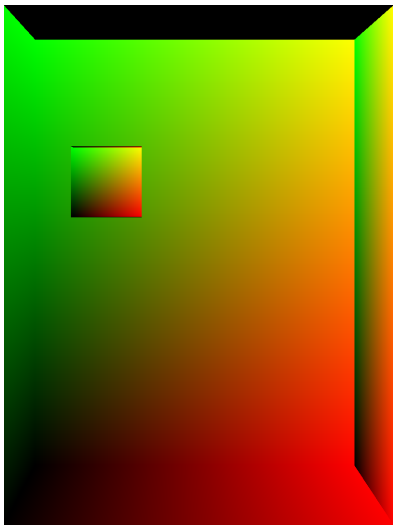


Figure 4.3: Illustration of the UV map

With each position that was previously sampled on the light, a single point light was animated to take the position of the sample every frame. In this thesis, a shadow catcher material was applied to the surfaces which made it possible to render out only shadows in the scene per frame, resulting in one shadow mask per sample. For each sampled position, a shadow mask was rendered with only the contribution of a single point light sample, this in order to store the visibility from each sampled light position. All of the shadow masks were then processed to only contain binary information, in this case if a pixel is lit (1) or not (0). This is illustrated in Figure 4.4 where black marks regions that are not visible from that particular light sample.

A texture with the purpose of accelerating the visibility computation in the post-processing step is also generated. In addition to accelerating computation, it will help determine which pages to commit in a Sparse Virtual Texture. Referred to as

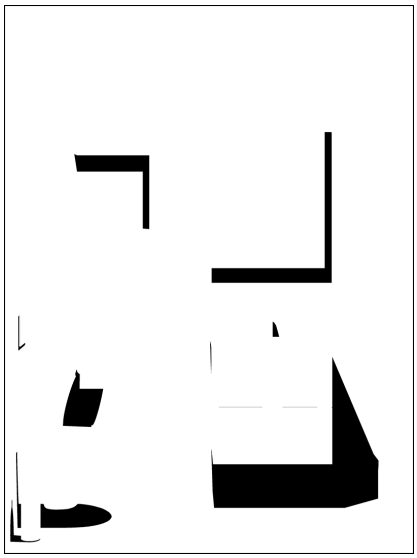


Figure 4.4: Illustration of a shadow mask, black pixels represent shadow

the "Shadow Info" image, its texels represent the areas in umbra, penumbra and full light of the camera view. Basically, the camera view is divided into a grid where each element is a texel in Shadow Info and also the size of a virtual page in the shadow mask. If the virtual page size is $x \times y$ and the shadow mask is $X \times Y$ resolution, then the Shadow Info texture is $X/x \times Y/y$ resolution. An example of a Shadow Info image is shown in figure 4.5. Using this texture in the shading stage will save some computation time, while the SVTs will save a significant amount of memory, further explained in the following subsection.

Figure 4.5: Illustration of Shadow Info
Black marks pixels in umbra, grey penumbra, and white fully lit.

In summary, the essential information to store in this stage is:

- every light sample's world position to txt file
- the camera world position to txt file
- each pixel's normal and tangent as EXR files
- each pixel's world position as EXR file
- UV maps for bump-map look-up as EXR file
- shadow masks that determine visibility from every pixel to all light samples, stored as grayscale PNG
- Shadow Info image that shows whether each page area is in umbra, penumbra or fully lit, as PNG

To minimize the memory consumption, all of the per-pixel images are stored as RGB16F (16-bit float) instead of RGB32F (32-bit float) format, which does not affect the end result noticeably.

4.2.3 Post-processing Stage - Initialization

The post-processing stage is executed during run-time, and it is during this stage that shading is computed based on normals from a given bump-map. Before explaining how shading is performed however, we will cover the initialization where important textures are generated. The information gathered from the previous stage are loaded in the initialization together with the scene variables. Larger information such as pixel world positions, normals and tangents, normal maps and UV maps are stored in 2D textures while a few vectors such as light sample positions and directions and camera position are stored in variables.

During the initialization, a Sparse Virtual Texture is created for storing the visibility information required from the shadow masks. It is created by looping through the Shadow Info image, to check whether each page region should be committed. In Shadow Info, a pixel value of (0) or (1) specifies if the page is in umbra or fully lit respectively, and a value of (0.5) indicates penumbra. Only if the page is in penumbra will it be committed to the virtual texture. If a page is committed, all shadow masks will be looped through in the committed page region and the visibility bit of each texel in a shadow mask will be submitted to an SVT. To pack the data as compactly as possible, they are submitted to a RGBA32UI texture format (32 bits per channel, 128 bits per texel), where every bit in a SVT texel represents a visibility value from one shadow mask. This means one SVT fits visibility data from 128 shadow masks, and to fit more shadow masks, we simply generate more textures. With this and all other per-pixel data ready, they can be transferred over to a fragment shader.

4.2.4 Post-processing Stage - Shading

In the fragment shader, we firstly calculate all the vectors required for evaluating direct light. Such vectors are incoming light w_i, outgoing light w_o, and the perturbed normal n. The incident and outgoing light vectors are calculated by vector subtractions based on pixel position and camera or light position. The perturbed normal is achieved by first performing a texture look-up in the UV map, and then

using the value from the UV map for a look-up of the normal perturbation from the normal map. Since the x and y perturbation is defined in the range 0 to 255 as RGB integer values in our normal maps, it is hereafter normalized and mapped to -1 to 1. Before using the perturbation from the normal map however, it first needs to be transformed to align with the surface normal's coordinates. As explained in Section 3.4.1, this is done by computing the bi-tangent as a cross product of the normal and tangent, and then constructing the TBN matrix used to transform the perturbation to the correct coordinate space. The outgoing light direction is a single vector from the pixel position to the camera, however the incoming direction has to be computed from the N light sample positions that approximate the area light, when these are evaluated. All of the vectors are normalized.

Before calculating the light contribution, we make further use of Shadow Info, the acceleration texture. Each texel of the Shadow Info image are sampled to receive the visibility information per page. If the value is (0), all pixels in the page are in umbra which means we can avoid any further calculations or texture look-ups. In case the stored value is (1) it means that all pixels are fully lit, and we can skip the visibility look-up and directly evaluate the lighting. Lastly if the value stored is (0.5) indicating penumbra, a look-up in the SVT will later be performed for every light sample in order to determine which light samples contribute. Moving on, the direct lighting will be calculated, where the Rendering Equation in surface form (3.4) and Torrance-Sparrow BRDF (3.6) are used.

When computing lighting, the contribution from all light sources in the scene need to be summed up. Therefore, to produce the total lighting, we loop over and calculate the lighting from all light sources before accumulating them into a single variable after the loop. The method used to compute the lighting from a single light source is the Rendering Equation approximated with Monte Carlo Integration, as defined in Equation 3.5. Firstly, to achieve soft shadows, the visibility has to be taken into account. See Figure 4.6 for an illustration of how the shadows depend on visibility from each individual light sample. The soft shadows are produced by evaluating each light sample's visibility to the shading point, so at this point, we loop through all previously sampled points of the light sources and perform a look-up in the shadow masks. However, this is only done if the area is in penumbra as explained. If the light sample is visible from the point and the normal of the surface at that point is not back-facing the light, then we continue with computing the light contribution.

The first step in computing the light contribution from a sample is calculating L_i, the incoming radiance from the light source. This is simply a product of the light color and an intensity multiplier divided by the area of the light:

$$L_i = (light\_intensity\_multiplier \times light\_color)/A \tag{4.1}$$

Following that, the Torrance-Sparrow BRDF is computed by using the four vectors n, w_i, w_o and w_h, and the material parameters R_0 and s as described in Section 3.3. In this thesis, only dielectrics are modelled since the walls are made of glazed ceramic tiles. For these materials, the scattered light is a combination of diffuse and specular

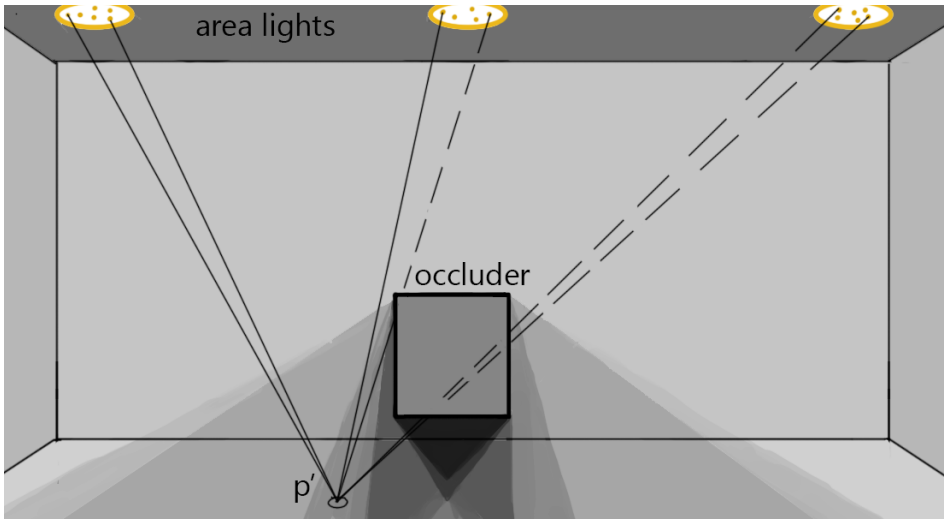


Figure 4.6: Visualization of how shadows are created
Dots in the area lights represent samples. Dashed lines from the shading point p' (small circle at bottom) represent not visible from the light sample, full lines are visible. Visibility is specified by shadow masks pre-rendered from the view of each light sample.

reflections, depending on its reflectivity. The reflected light for a perfectly diffuse surface is calculated according to Equation 4.2. Here we add L_a which represents ambient light, in order to roughly approximate the effect of indirect light, causing the scene to be brighter overall.

$$diffuse\_term = material\_color\frac{1.0}{\pi}|n \cdot w_i|(L_i + L_a) \tag{4.2}$$

In the case of a material being dielectric, the non-reflected light is refracted into the material where it reflects back and forth until it exits in a random direction. This is modelled by letting the refracted light reflect diffusely using the diffuse term in Equation 4.3. Here, F is the Fresnel term in the microfacet model.

$$dielectric\_term = BRDF \times (n \cdot w_i)L_i + (1 - F(w_i)) \times diffuse\_term \tag{4.3}$$

To blend between the diffuse and dielectric BRDF, Equation 4.4 is used where r is the material's reflectivity parameter. The blended term is also multiplied with the geometric coupling term G (3.3) and divided by the PDF $(1/A)$ to produce the light contribution from one sample.

$$L_{sample} = (r \times dielectric\_term + (1 - r) \times diffuse\_term) \times \frac{G}{pdf} \tag{4.4}$$

As a final step, the light sample contributions are summed and divided by N, the amount of samples, to produce the outgoing light L_o given by one light source. The outgoing light produced by all light sources in the scene are then summed together for the total outgoing light from the point being evaluated. With the shader implementation finished, it is possible to render the scene.

4.2.5 Super-sampling and Testing

After implementing the fragment shader and rendering the first image shown in Figure 5.1a, it was easy to notice all of the aliasing problems. For the tile normal maps used in this scene, a grout map was accompanied in order to visualize the grout. By visualising the grout, the aliasing effects become more clear. The most obvious ones are the missing horizontal lines on the floor and vertical lines on the side walls, in addition to missing highlights. Looking closer it is also possible to see jagged lines in the tile grout, and some grout lines are darker than others or appear to be wider.

In order to eliminate the unwanted aliasing problems, a super-sampling scheme was implemented. It was implemented by increasing the resolution of all per-pixel data, rendering the image at the higher resolution, and then down-sampling to the original resolution by averaging the super-sampled colors. For example, in this thesis 4x4 super-sampling was tested, which means computing shading for an image of 3232×4320 resolution, and then down-sampling the colors to the original 808×1080 resolution. As 4x4 super-sampling was not sufficient enough to erase all aliasing effects, 8x8 and 10x10 were also evaluated, and all results are shown in Figure 5.1.

Finally, to be completely certain that the implementation works as intended, further testing was performed. The testing consisted mostly of changing settings regarding the level of super-sampling, amount of light sources and light samples in order to produce a satisfactory quality. Since Spark Vision's application visualizes glossy walls, the goal is to replicate their material parameters. These material parameters were not available, and therefore the shininess s, reflection coefficient R_0 and reflectivity r had to be manually adjusted in order to properly compare the images with ray-traced references. Since the materials were missing, we could not render references with bumps ourselves, and instead they were provided by Spark Vision from their collection of bump images. The results of the tests are presented in the following chapter.

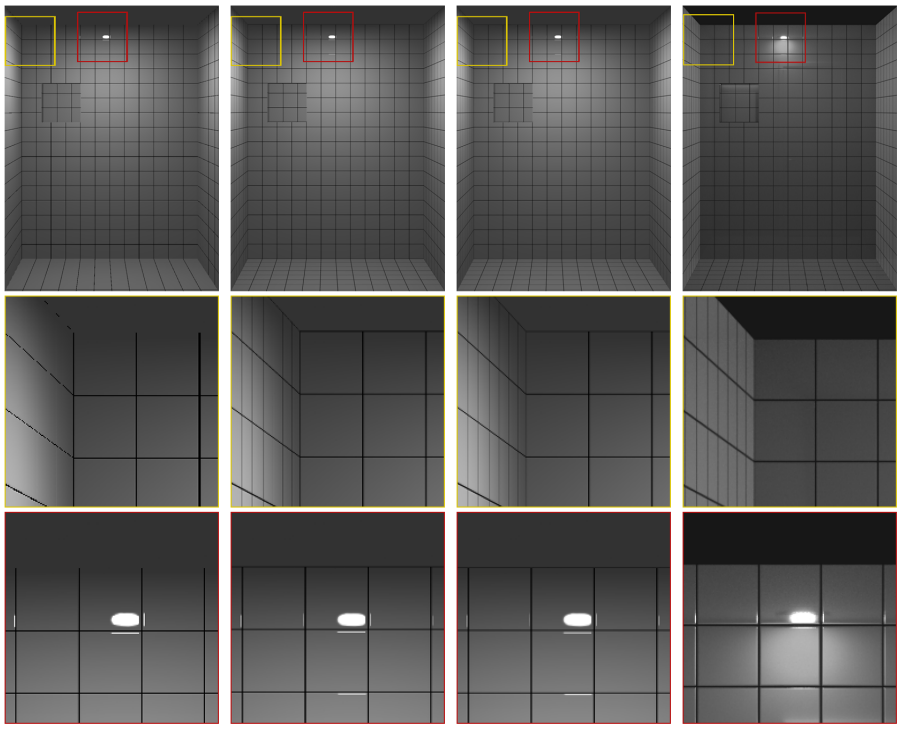
5
Results

In this chapter, the image quality and performance of the method are presented. The implementation was tested in two different scenes; a smaller scene with one light and a larger scene with more lights including a big window light. The smaller scene mostly tested the visual quality of effects such as different bumps, specular highlights in the curvature of tiles, and soft shadows around furniture. The bigger scene tested the quality and performance of a chosen optimal setting, with different amount of lights present.

5.1 Small Scene - Visual Quality

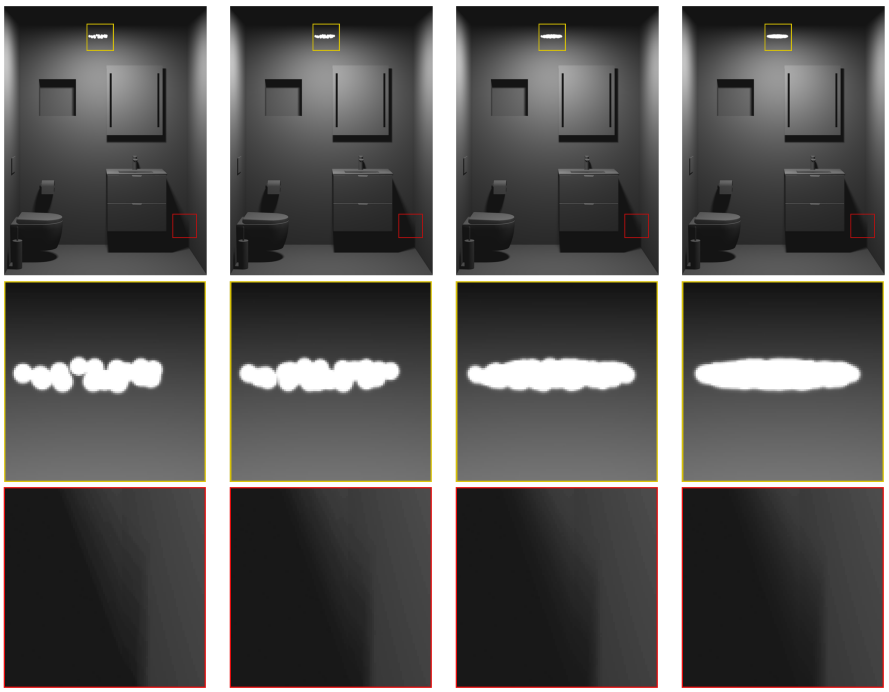
The small scene was mainly used to test image quality with different settings. Figure 5.1 shows the small scene rendered with different levels of super-sampling, and a ray-traced reference. Note that the ray-traced reference is missing the ceiling, and includes indirect lighting, which explains the diffuse bright spots. In the comparison, only the most bright highlights produced by the light are considered. Looking at Figure 5.1a that is rendered without super-sampling, the aliasing effects present are numerous. In particular, all vertical grout lines are missing on the side walls and some highlights are missing on the tile curvature. Figures b and c show the scene rendered with super-sampling, and improvements can be noticed. With 4x4 super-sampling all highlights appear, but one vertical line is still missing on the left wall. By increasing the super-sampling to 8x8, all aliasing effects were removed. No significant improvement was noticed when further increasing the super-sampling to 10x10.

In Figure 5.2, the small scene is rendered with a larger disc light and varying amounts of light samples used to approximate the light source. When increasing the size of the light source, there is also a need to increase the number of light samples used in the approximation. This is due to two reasons, the first being the quality of the light reflections shown in the yellow boxes. With insufficient light samples, the light reflection will appear as in Figure 5.2a, where each individual sample is distinguishable. With 512 light samples, it is no longer possible to discern the individual samples, but the reflection is still not completely uniform. The second reason is the soft shadows created by the furniture, shown in the red boxes in figure 5.2. With low amount of samples, it is possible to discern the overlapping of each individual shadow in the penumbra. By increasing the light samples, the overlapping becomes less apparent, e.g. at 128 samples, edges between each individual shadow are only noticeable at a closer look. In Figure 5.2d illustrating when 512 light samples are used, the transitions are smooth, resulting in a gradient-like penumbra.



(a) No SS **(b)** 4x4 SS **(c)** 8x8 SS **(d)** Reference

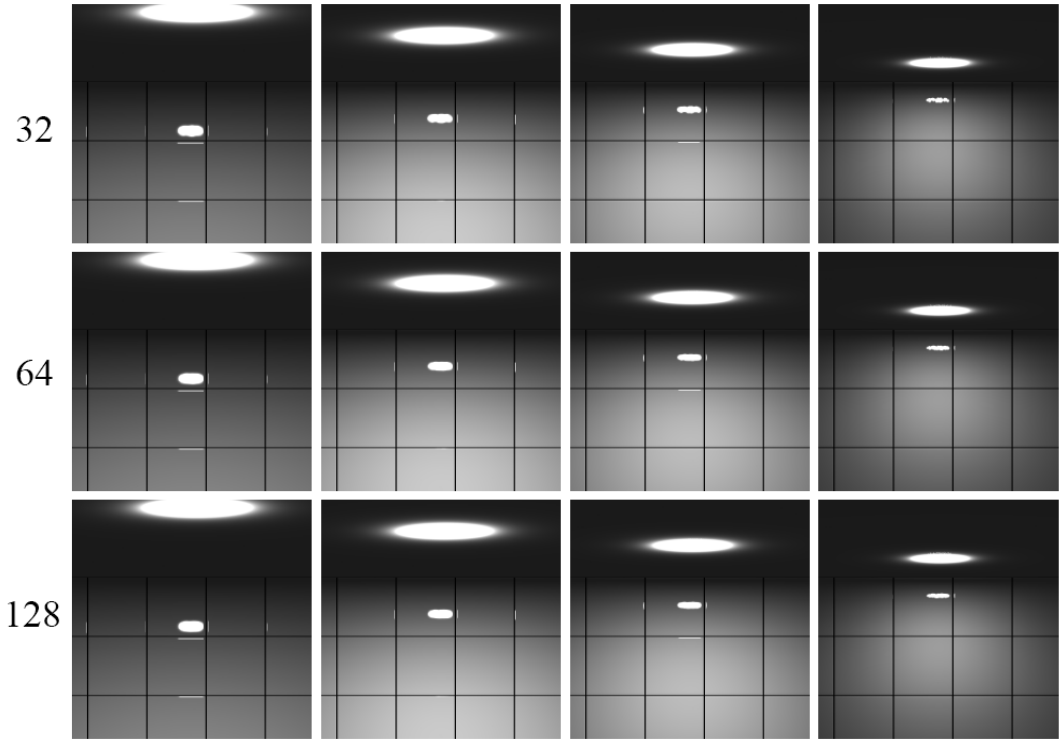
Figure 5.1: Figures of different super-sampling levels and a ray-traced reference



(a) 16 samples **(b)** 32 samples **(c)** 128 samples **(d)** 512 samples

Figure 5.2: Large disc light source with different amounts of light samples

Another factor that affects the light reflection quality is the distance from the light to the surface where the reflection appears. As seen in Figure 5.3, the light reflection becomes smaller as the distance to the back wall decreases. Simultaneously, the reflection becomes less uniform at lower light samples. When the light is very close to the surface, as depicted in Figure 5.3d, at least 128 samples are required for a relatively uniform reflection.



(a) Original position **(b)** 0.3m offset **(c)** 0.5m offset **(d)** 0.7m offset

Figure 5.3: Small disc light with varying distances to wall (column) and samples (row) The light position is shown on the ceiling. With decreasing distance to the back wall, the reflection becomes smaller and less uniform at lower light samples.

A similar effect to the above can be observed when the shininess value of the material increases. On the contrary, as the shininess decreases, the reflection blurs out and less samples are required. An example of the large disc light with lower shininess values requiring less samples (128 samples) for a uniform shape can be seen in Figure 5.4. Comparing with the reflection in Figure 5.2c, it is more blurry but also a lot more even in shape.

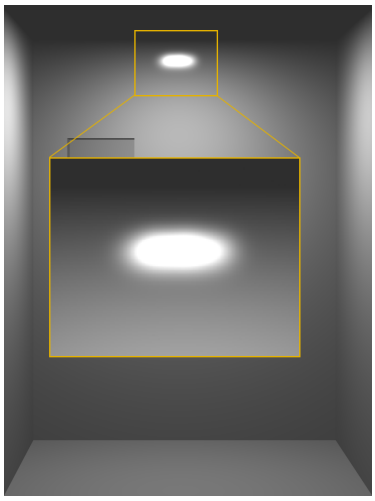
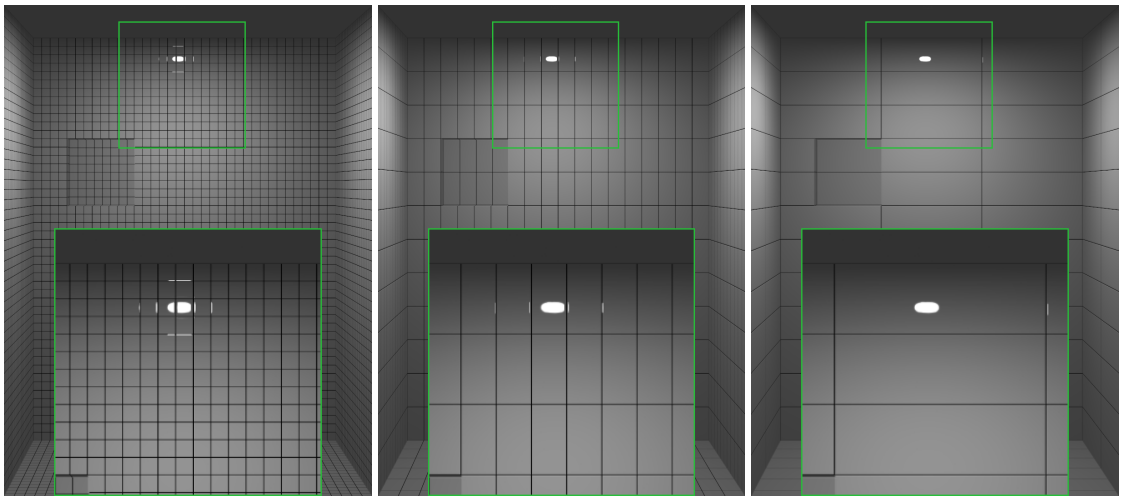


Figure 5.4: Large disc light with 128 samples
The wall material is less specular, resulting in a blurry but uniform reflection.

To show that the method can handle various normal maps, including smaller and more detailed ones, rendering tests with different normal maps were performed. The results can be seen in Figure 5.5. All figures show that the tiles are rendered without any aliasing effects in the grout, especially Figure 5.5a, where the tiles are small and highly frequent. Highlights on the tile curvature to the left and under the light reflection of Figure 5.5c seem to be missing. Further testing was performed with more distinctly patterned normal maps, including a wooden textured pattern showing that the method handles highly detailed normal maps well. These can be seen in Figure A.1.



(a) 50x50 tile size **(b)** 100x200 tile size **(c)** 600x200 tile size

Figure 5.5: Figures with different normal maps, rendered with 8x8 super-sampling

5.2 Small Scene - Performance

Super-sampling and increased light samples improve the quality, but also affect the performance of the algorithm. Figure 5.6 illustrates how the render time is affected by the increase in the two parameters, and Table 5.1 shows the exact times visualized in the graph. Since the application used for testing is capped at 60 frames per second, the lower limit of render time is 16.66 ms. Therefore, with no super-sampling there is no significant increase in render time when the amount of light samples increases. At higher super-sampling levels however, the graph shows that the render time mostly scales linearly with the amount of samples. With 4x4 super-sampling there is only a small increase in render time at 64 samples, and when raised up to 512 samples, the render time is still below 200 ms. Looking at 8x8 and 10x10 super-sampling, the render times are only acceptable at around 128 samples. For the optimal quality vs. performance setting for the small scene, 8x8 super-sampling with 128 samples takes the prize. At higher light sample count, the render times start becoming prohibitive, as it reaches up to 618 ms with 512 samples. Since 8x8 super-sampling provides good enough super-sampling, it will be used further when testing the big scene.

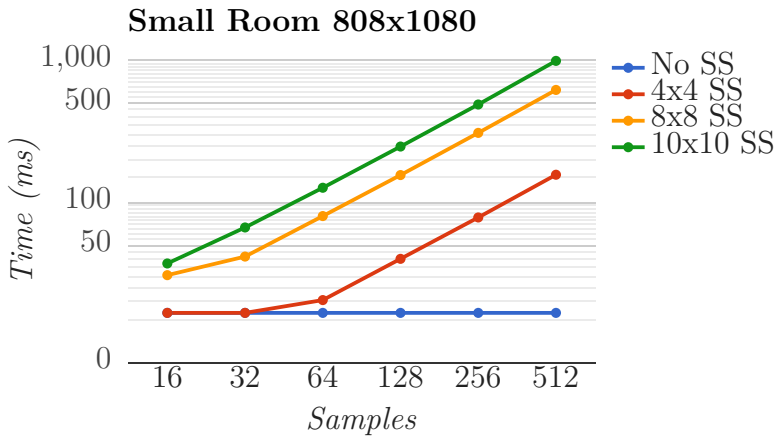


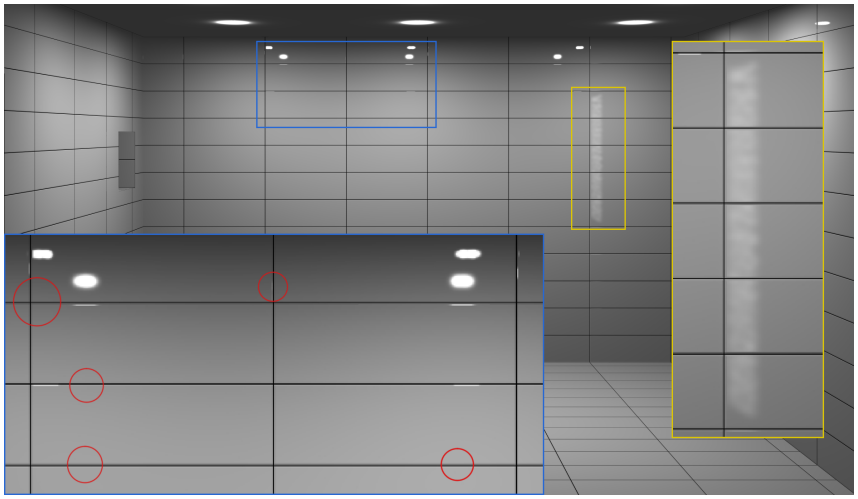
Figure 5.6: Graph of render times on small scene (808x1080)
Relates render time to amount of light samples, at 4 different super-sampling levels. The time (y) axis has a mirrored logarithmic scale.

Small scene render times				
Light Samples	**No SS**	**4x4 SS**	**8x8 SS**	**10x10 SS**
16	<16.66 ms	<16.66 ms	30.7 ms	37.1 ms
32	<16.66 ms	<16.66 ms	41.5 ms	66.5 ms
64	<16.66 ms	20.5 ms	80 ms	126.6 ms
128	<16.66 ms	40 ms	155.5 ms	246.9 ms
256	<16.66 ms	78.2 ms	308.1 ms	488 ms
512	<16.66 ms	156.3 ms	618 ms	989.5 ms

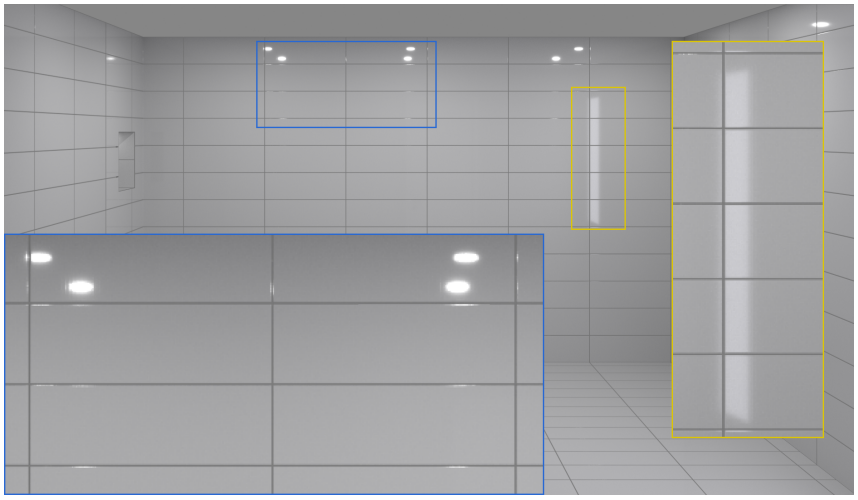
Table 5.1: Render times in ms for the small scene at different settings

5.3 Big scene - Visual Quality

The big scene consists of a window (rectangular light) on the right wall and 6 small disc lights on the ceiling. Figure 5.7 shows the big scene rendered in 8x8 super-sampling with our method and a ray-traced reference image which includes indirect lighting. The light sources in our render are approximated with 1216 light samples in total, 32 per disc light and 1024 for the window. Differences in the disc light reflections are visible, where the shape is slightly off with our method. The reflections seem to be slightly offset upwards and to the right compared to the reference, which also occurs in the small scene. Furthermore, some reflections in the curvature of the tiles are missing, indicated by the red circles. The window reflection shown in the yellow boxes show that our render produces less uniform shape compared to the reference, suggesting that 1024 samples are insufficient. A reflection on the left wall is missing in our render, however offline rendering the scene using Redshift without global illumination produces no such reflection either.



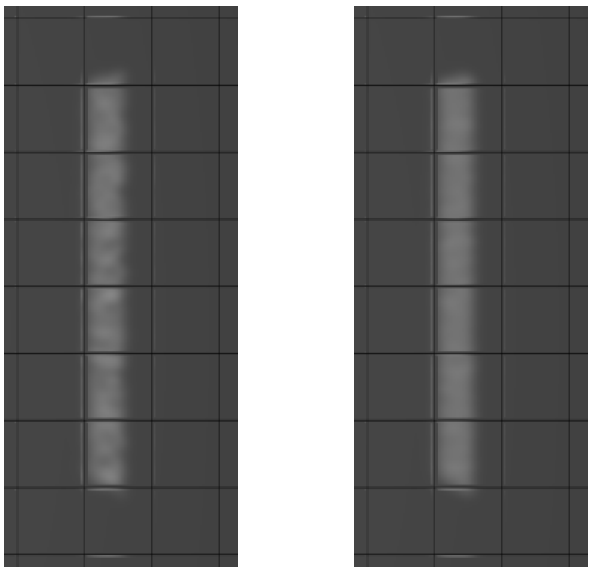
(a) Big scene with all light sources, total of 1216 light samples



(b) Reference image with direct and indirect lighting

Figure 5.7: Big scene (1874 × 1080) rendered with our method, and a reference image The red circles indicate missing highlights in our render.

Figure 5.8 shows only the window reflection of the big scene, approximated with 512 and 1024 light samples. In Figure 5.8a the individual light samples are discernible in the reflection, meaning 512 light samples are not enough to approximate the big window reflection. On the other hand the reflections in the curvature of the tiles are all present and does not suffer any negative effects from the amount of samples. When using 1024 light samples, the same reflections can be found in the curvature of the tiles, but the overall shape is a much smoother reflection that blends all the samples together. However, as seen in the Figure 5.7, it does not fully compare to the ray-traced reference where the edges are more defined.



(a) 512 light samples **(b)** 1024 light samples

Figure 5.8: Rectangular window light approximated by 512 and 1024 samples

Lastly, the big scene is rendered with furniture and shadows from only the 6 disc lights using 192 samples, excluding the window light. Although the window light with shadows could be rendered, in this testing we decided to exclude it since pre-computing the shadows would be too time-consuming given the remaining time. From what can be seen in Figure 5.9, the shadows from each light are correctly positioned and the shapes correspond to those in the ray-traced reference. At a closer look, the penumbra of our render are slightly darker and not as smooth when transitioning from umbra to penumbra, which suggests that a higher sample count may be required. When viewed at a distance however, the differences are not as noticeable, which may be acceptable in some cases. Also, some differences may arise because the ray-traced reference includes reflections from the environment, which brightens the shadows.

(a) Big scene with 6 disc lights (192 samples) and shadows

(b) Reference image with direct lighting and environmental reflections

Figure 5.9: Big scene (1874 × 1080) rendered with furniture and shadows, including a ray-traced version

5.4 Big scene - Performance

Now the performance of the big scene is presented and briefly discussed, including both render time and memory consumption when using shadows. Table 5.2 shows the render times for the large scene with different amounts of light samples and 8x8 super-sampling. With all lights present, the render time takes 4237 ms, which is way beyond the limits of our goal. However, this is mostly due to the window light which requires at least 1024 samples for a relatively uniform reflection. When considering only the 6 disc lights, the render time is 548 ms which is still over double the 200 ms goal, but it should be more manageable with further optimization of the algorithm.

Big room render times	
Light Samples	**8x8 Super-sampling**
192 (disc lights)	548 ms
512 (window)	2412 ms
1024 (window)	3820 ms
1216 (all lights)	4237 ms

Table 5.2: Render times in ms for the big scene (1874 × 1080)
8x8 super-sampling and different light sources and amount of light samples are used

When test rendering, we discovered that some memory could be saved since 8x8 super-sampling of the per-pixel data was only required for the UV map. The jagged edges that appear at shadow boundaries and edges of geometry were solved at 4x4 super-sampling for the shadow masks, pixel world positions, normals and tangents. Thus, in theory, the memory consumption when rendering the big scene with shadows, without optimization, are:

- per-pixel world positions, tangents and normals (4x4 super-sampled):
  $(1874 \times 1080 \times 4 \times 4) \times (3 \times 32/8) \times 3 =$ **1.165 GB**
- 1 UV map of the scene (8x8 super-sampled):
  $(1874 \times 1080 \times 8 \times 8) \times (3 \times 32/8) \times 1 =$ **1.55 GB**
- 192 shadow masks (4x4 super-sampled):
  $(1874 \times 1080 \times 4 \times 4) \times (1) \times (192/8) =$ **777 MB**
- for a total of:
  $1.17 + 1.55 + 0.777 =$ **3.497 GB**

By reducing the 32bit data to 16bit, the resulting memory usage is:
$0.585 + 0.775 + 0.777 =$ **2.137 GB**

The conclusion drawn from the calculations is that the shadow masks are the main bottleneck, since the light sample count could go up to 1000 or more. With 1000 shadow masks, the memory usage increases to $(1874 \times 1080 \times 4 \times 4) \times (1000bits/8) =$ **4.05 GB**. So halving the per-pixel data is not enough. By using Sparse Virtual Textures, the memory consumption of the shadow masks is reduced significantly. The reduced amount can be calculated by counting the amount of pages in penumbra, since only those are committed in the texture, and that resulted in 1833 virtual pages as illustrated in Figure 5.10. In the implementation, the virtual page size for RGB32UI format is 64×64, meaning the theoretical memory consumption with 192 samples is: $(1833 \times 64 \times 64) \times (192bits/8) =$ **180.2 MB**, approximately 23 percent of the original figure (777 MB). With all per-pixel data included, the memory consumption becomes a total of: $0.585 + 0.775 + 0.18 =$ **1.54 GB**. If 1000 samples were used, the memory consumption would be $4.05 \times 0.23 + 0.585 + 0.775 =$ **2.29 GB**, which is still a manageable figure for modern hardware. In practice however, this number can be slightly higher due to implementation details and depending on e.g. the amount of normal maps loaded.

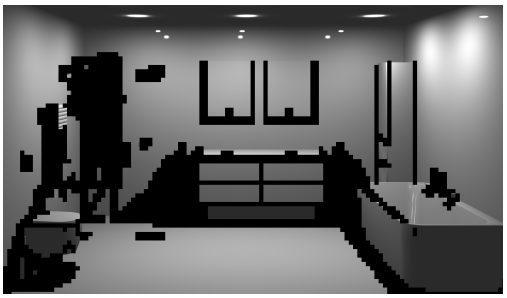


Figure 5.10: Illustration of Sparse Virtual Texture committed areas
The dark squares indicate areas in penumbra, i.e. pages where visibility data are submitted to the SVT. Other areas are uncommitted and do not consume memory.

5.5 Final Tests

Now, the result of some final tests on performance and visuals is presented. Firstly, although render times with and without using SVT were not tested on the large scene, for the smaller scene with 512 samples and 8x8 super-sampling, the outcome was:

- shadows without SVT (small scene): 700 ms
- shadows with SVT (small scene): 589 ms

The improvement is not immense, however when more samples are used, i.e. 1000 or more, the memory consumption will start becoming a serious bottleneck. In that case, using SVT will save significantly more render time.

Finally, Figure 5.11 shows a pre-rendered image of the small scene with bumps applied. The difference is not striking, however it does provide some sense of depth and gloss of the back wall that was lacking without the highlights on the tiles near the light reflection.

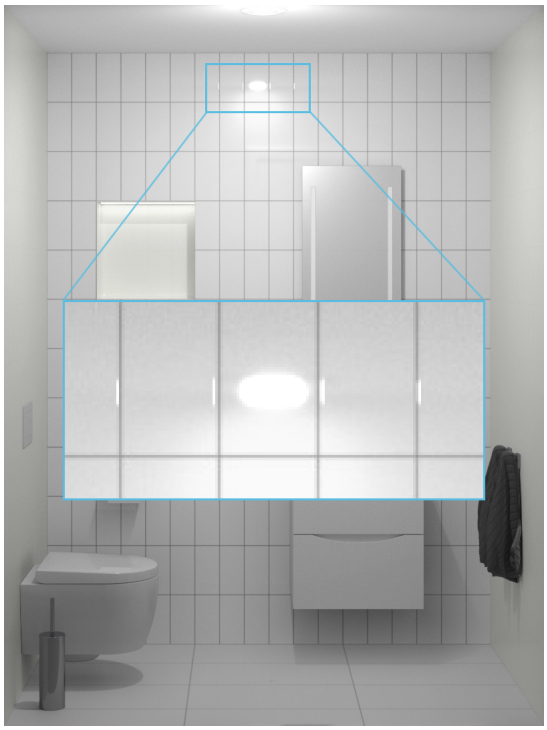


Figure 5.11: Illustration of pre-rendered small scene with global illumination and bumps applied using our method

6
Conclusion

The results have provided some closure but also raised questions about the method, and these matters will be discussed first in this chapter. Thereafter, different ideas on ways to improve the method in terms of quality and performance are brought up. Finally, a conclusion is drawn to the thesis with a recap and review on how well the method fulfilled the goals.

6.1 Discussion

The overall quality of the images achieved can be considered satisfactory with 8x8 super-sampling and sufficient amount of light samples. With sufficient amount of samples, the soft shadow quality is comparable to ray-traced references as far as the result shows. For example in Figure 5.2d where 512 samples are used, the overlapping of each individual shadow is indistinguishable, resulting in a gradient-like transition in the penumbra. Furthermore, with multiple lights, the shadows are correctly placed throughout the scene with appropriate soft shadows. As for the light reflections, when comparing the images in Figure 5.7, it is possible to notice a disparity. Firstly, the shape of the reflections are not identical, however this can be explained by differences in material parameters, in addition to indirect lighting in the reference image. Secondly, the reference image contains more reflections in the tile curvature and the reflections are somewhat offset. The reasons to this will be discussed next.

There are several possible reasons to why highlights do not appear in some areas with our method. One is that the reference image includes indirect lighting which results in additional reflections from the surroundings. Another reason can be that the normal maps used were not accurate enough. When rendering images, we realized that the method was sensitive to noise in normal maps when it comes to glossy surfaces. Small discrepancies in the normals led to some highlights not appearing at all, but perhaps that is reasonable given how smooth the surface material is. Because of this sensitivity, inaccurate highlights are caused by saving the normal maps as RGB integer textures, since some precision is lost in the values due to rounding to integers. To give an example, a normal with no perturbation in x and y axis (0, 0, 1) results in the RGB integer value (127, 127, 255) in our implementation. However, the actual R and G value should be $255/2 = 127.5$, and thus the normal will be perturbed slightly towards the negative x and y direction. This explains the minor offset in the reflections as seen in Figure 5.7. A final reason could be that the UV maps are not fully accurate, which is observed by some grout lines being thinner or thicker.

With the light sources in set positions, it was possible to find the minimum amount of light samples required to achieve an accurately shaped reflection. Increasing the size of the light source was no issue, as the increase in light samples could be determined by the increase in area. However, by changing the light's position or increasing the material's shininess parameter, it was discovered that the amount of light samples needed to be adjusted further. Increased shininess or decreased distance to the surface where the reflection appears leads to more samples required to keep the reflection accurate. This means the minimum amount of light samples depends on not only the size of the light, but also the distance to where the reflections appear (which may be anywhere), and the shininess value. Because of this, it is difficult to derive a formula for calculating the minimum light samples required in a scene, and so, determining the appropriate light sample count will be up to the light artists.

As can been observed in Figure 5.1 and 5.2, to reach adequate quality in the images, both the super-sampling and light sample count need to be increased to a certain degree. In renders of the smaller scene where fewer and smaller light sources are used, the render times and memory consumption are acceptable. As the scene grows larger and multiple light sources are used, requiring more light samples, it results in higher memory consumption and longer render times. For all test renders of the large scene, the render time exceeded our limit of 200 ms. Especially for the large window light, excessive amounts of light samples were required to produce an acceptable reflection, which resulted in high render times. From the calculations of memory usage in section 5.4, the total memory consumption with 192 samples for the large scene is 1.54 GB. With the GTX 1080 Ti's memory bandwidth of 11 Gbps, the data is transferred in 1.54 GB / 11 Gbps = 0.14s. The render time is 0.548s, which suggests that the algorithm is compute bound and not memory bound. This is further confirmed by the fact that the render time increases linearly with the light sample count (without increasing shadow masks), which increases the arithmetic operations performed while memory stays constant. Therefore, to improve the render times, optimizations will be required on the shader stage. Since we spent most of the time optimizing for memory, there is likely room for arithmetic optimizations in the code. To address the high light sample count problem, what we realized late in the project is that sampling glossy material with light reflections is inefficient and could be improved by instead sampling directions based on the BRDF. More details about this will be explained later, in the future work section.

One question that may arise is why sampling the light is done as a pre-computation instead of dynamically for every pixel, as in distributed ray-tracing. This is because in our method, the visibility is determined by pre-computing the shadow masks. The shadow masks need to correspond to the sampled light positions, as evaluating the light contribution from one sample in one position but using shadow masks that are rendered from another will result in incorrect shading. To deal with this, a way to separate the visibility and light calculation is presented by McGuire et al. In the proposed method, stochastic shadows are computed and denoised, and combined with unshadowed lighting to produce soft shadows in real-time [11]. For future ref-

erence where dynamic geometry is investigated, this may be a preferable method for computing shadows as it avoids pre-computation. However, the quality of the soft shadows might not be up to par with the current method.

Another significant matter to take note of is that our current method does not differentiate one surface's material and bumps from another — all surfaces in the rendered scene will be of the same material and bumps. This can be corrected by adding an additional per-pixel texture which identifies every surface by a value, and then in the shader associate the value with a pre-defined material. This should not be difficult nor costly in terms of memory, but since it was not a requirement of the thesis, we decided to not spend time on implementing it.

6.2 Future Work

As mentioned in the discussion, there are several improvements that can be made to the method. For one, global illumination and dynamic geometry are not possible with this method, so further research to achieve more realistic lighting could include these. In this thesis grid super-sampling was implemented, but what has not been explored is the use of different sampling patterns. Using some other sampling pattern such as stratified sampling might decrease the amount of super-sampling (currently 8×8) needed and in turn improve the performance, making it worth exploring.

Due to the reflections being sensitive to the precision of the normals in the normal map, an improvement could be to instead use height maps and calculate the normals by using the finite difference method. Alternatively, the normals can be saved with higher precision, e.g. as an EXR file. Whichever results in optimal quality and performance can be explored.

A way to improve the quality of the specular reflections without increasing the light samples to excessive amounts, is to also importance sample directions from the BRDF. For smooth surfaces, this means sampling directions close to the angle of reflection, which is where the BRDF contributes most. This works well for specular reflections because the directions in which light contributes are in a narrow range, which means only a small amount of sampled directions are required. For rougher surfaces, the directions are in a wider range, in which case sampling the light is more efficient. The method would require a fixed grid of light samples with the position of the samples and the center of the light stored. Rays are sent in the BRDF sampled directions to check if they intersect the light source. If they do, the values in the visibility textures of the closest light samples are interpolated to calculate an approximation of the visibility. If there is no occlusion, then the light contribution is calculated according to the Rendering Equation in solid angle form.

Finally, alternative ways to evaluate lighting could be explored, such as the method which uses Linearly Transformed Cosines (LTC) developed by Heitz et al. [10], mentioned in the related works section. In the method, an analytical integration

over an arbitrary polygon-shaped light is possible by approximating the BRDF as LTC. In addition, the computation is done in real-time. An idea to include shadows in the method is to have visibility textures for sampled positions on the light, similar to the method described above. These are used to determine which parts of the polygon is occluded, and then only the visible parts of the light are integrated over. Whether this works well or not would be interesting to investigate.

6.3 Summary

The method developed in this thesis is capable of computing bumps dynamically without the use of a 3D scene. This is achieved by retrieving only the necessary geometric information beforehand when the scene is available. Furthermore, the method accurately evaluates direct lighting from area lights using Monte Carlo Integration and shadow masks, pre-computed textures which determine visibility. Altogether, it results in relatively realistic bump effects that come close to ray-traced references, but there is room for improvements. For example, currently the method only evaluates lighting by sampling points on the area light, but for specular reflections, sampling directions based on the BRDF is more efficient. As with global illumination effects and dynamic geometry, this will have to be a task for the future.

To deal with aliasing effects that were present when using per-pixel data with original resolution, 8x8 super-sampling was implemented, and all aliasing problems were removed. With super-sampling, a performance trade-off is inevitable, but at least the high memory consumption problem were mitigated by utilizing mainly Sparse Virtual Textures. As future work, there is potential for additional optimizations regarding the algorithm.

In conclusion, the current method will be able to generate the bump effects according to the performance goal of 200 ms or less, to avoid impeding the user experience. However, this is only for smaller scenes where fewer and smaller light sources are used. For bigger scenes that contain both multiple and larger lights requiring many light samples, the render times greatly exceed the limit of 200 ms. With further development and optimization however, handling larger scenes is not an impossibility.

Bibliography

[1] Akkanine Möller, T. Real-Time Rendering Fourth Edition. Tech. rep., 2018.

[2] Blinn, J. F., and Jpl, C. . SIMULATION OF WRINKLED SURFACES. Tech. rep., 1978.

[3] Chen, W.-C., Bouguet, J.-Y., Chu, M. H., and Grzeszczuk, R. Light Field Mapping: Efficient Representation and Hardware Rendering of Surface Light Fields. Tech. rep., 2002.

[4] Dachsbacher, C., Křivánek, J., Hašan, M., Arbree, A., Walter, B., and Novák, J. Scalable realistic rendering with many-light methods. *Computer Graphics Forum 33*, 1 (2014), 88–104.

[5] Dachsbacher, C., and Stamminger, M. Reflective Shadow Maps. Tech. rep., 2005.

[6] Donnelly, W., and Lauritzen, A. Variance shadow maps. In *Proceedings of the Symposium on Interactive 3D Graphics* (2006), vol. 2006, Association for Computing Machinery, pp. 161–165.

[7] Dutre, P., Bekaert, P., and Bala, K. *Advanced Global Illumination 2nd Edition*, 2nd edition ed., vol. 1. CRC Press, 8 2006.

[8] Gershbein, R., and Hanrahan, P. A Fast Relighting Engine for Interactive Cinematic Lighting Design. Tech. rep., 2000.

[9] Hachisuka, T., Ogaki, S., and Jensen, H. W. Progressive photon mapping. *ACM Transactions on Graphics 27*, 5 (12 2008).

[10] Heitz, E., Dupuy, J., Hill, S., and Neubelt, D. Real-time polygonal-light shading with linearly transformed cosines. In *ACM Transactions on Graphics* (7 2016), vol. 35, Association for Computing Machinery.

[11] Heitz, E., Hill, S., and McGuire, M. Combining analytic direct illumination and stochastic shadows. In *Proceedings - I3D 2018: ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (5 2018), Association for Computing Machinery, Inc.

[12] Jensen, H. W. Global Illumination using Photon Maps. Tech. rep., 1996.

[13] Kajiya, J. T. THE RENDERING EQUATION. Tech. Rep. 4, 8 1986.

[14] Keller, A. Instant Radiosity. Tech. rep., 1997.

[15] Levo, M., and Hanrahan, P. Light Field Rendering. Tech. rep., 1996.

[16] Nvidia. GeForce GTX 1080 Ti Specifications, 3 2017.

[17] Peters, C., and Kleiny, R. Moment shadow mapping. In *Proceedings of the 19th Symposium on Interactive 3D Graphics and Games, i3D 2015* (2 2015), Association for Computing Machinery, Inc, pp. 7–14.

[18] Ren, P., Dong, Y., Lin, S., Tong, X., and Guo, B. Image based relighting using neural networks. In *ACM Transactions on Graphics* (7 2015), vol. 34, Association for Computing Machinery.

[19] Ritschel, T., Grosch, T., Kim, M. H., Seidel, H. P., Dachsbacher, C., and Kautz, J. Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics 27*, 5 (12 2008).

[20] Scania. Scania Configurator.

[21] Schlick, C. An Inexpensive BRDF Model for Physically-based Rendering. Tech. rep., 8 1994.

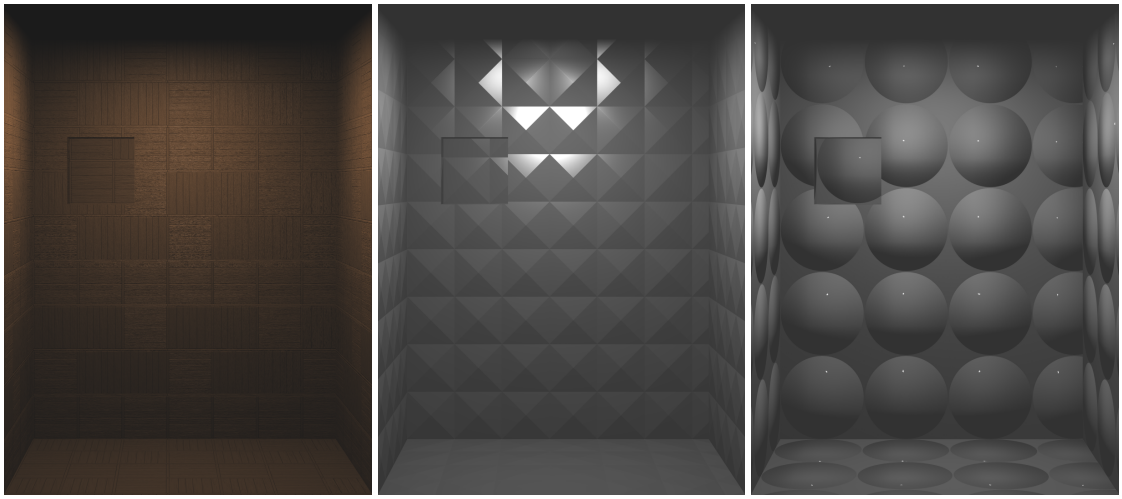
[22] Stamminger, M., and Drettakis, G. Perspective shadow maps. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '02* (2002), pp. 557–562.

[23] Walter, B., Fernandez, S., Arbree, A., Bala, K., Donikian, M., and Greenberg, D. P. Lightcuts: A Scalable Approach to Illumination. Tech. rep., 2005.

[24] Williams, L. CASTING CURVED SHADOWS ON CURVED SURFACES. Tech. rep., 1978.

A

Appendix 1



(a) Wooden bumps **(b)** Pyramid-shaped bumps **(c)** Sphere-shaped bumps

Figure A.1: Figure of different patterned bumps.

Figure A.2: Rectangular light source 512 samples