Stochastic Ray Tracing

Bachelor's Thesis

submitted to the
Institute of Computer Science and Applied Mathematics
University of Bern

by
Michael Pfeuti

2007

Supervisors:

Dipl. Inf. Ph. Robert

Prof. Dr. H. Bieri

Abstract

The goal of ray tracing is the synthesis of realistic images. Images generated by the Whitted algorithm lack several important effects which occur in reality. Stochastic ray tracing is a method to increase the realism of ray traced images. With stochastic ray tracing it is possible to generate penumbra, fuzzy reflection and translucency, depth of field, and motion blur. The Whitted algorithm does not produce these effects which are essential to realistic images. This bachelor's thesis explores the ideas proposed by Cook, Porter, and Carpenter in 1984 and the resulting enhancement of image quality as well as the loss of speed performance caused by stochastic ray tracing.

Contents

Chapter 1

Introduction

1.1 Motivation

One goal of computer image synthesis is to generate realistic looking images. The term photo-realism is often used in this context. To produce photo-realistic images, it is necessary to simulate many light effects which occur in reality. One technique to generate realistic images is ray tracing. The aim of this bachelor's project is to implement stochastic ray tracing and analyse it with regard to image quality and speed performance.

With stochastic ray tracing, some important light effects can be correctly simulated, which would be missing with standard ray tracing. Depth of field or penumbra are just two examples. The resulting images should, therefore, be of distinctly higher quality.

On the downside, a performance loss is an issue that arises. Hence, it will become one aspect to minimize this performance loss. This should be possible within certain boundaries.

1.2 Outline

This thesis is structured in four parts. The first part consists of an introduction to the rendering equation of James T. Kajiya [7], a brief discussion of standard ray tracing, a theoretic discussion about stochastic ray tracing which includes an in-depth analysis of the idea proposed by Cook, Porter, and Carpenter in 1984 [3], and an outline of a range of distribution techniques with their advantages and disadvantages. Additionally, there will also be a comparison between standard and stochastic ray tracing for every new feature.

The second part gives information on how the features are implemented. The ideas behind the algorithms of the different distribution will also be explained. Furthermore, one possibility to counteract the performance loss will be proposed.

The third part is an analysis of the image quality, the performance loss caused by stochastic ray tracing, and the acceleration methods. Each feature will be analysed separately.

The forth part gives a conclusion of the work and sketches ideas for future work.

1.3 Acknowledgements

I would like to thank Prof. Dr. H. Bieri and Ph. Robert for offering me this topic as a bachelor's thesis. Furthermore, I would like to thank Philippe for his support and very helpful advises and consulting hours.

Chapter 2

Theory of Stochastic Ray Tracing

2.1 Rendering Equation

In 1986, James T. Kajiya [7] proposed an integral equation that describes how light is transported in space. This integral equation basically consists of four main components, namely a reflection, an emission, an incoming and an outgoing light function. Figure 2.1 shows a simple scene with the main components of the rendering equation.

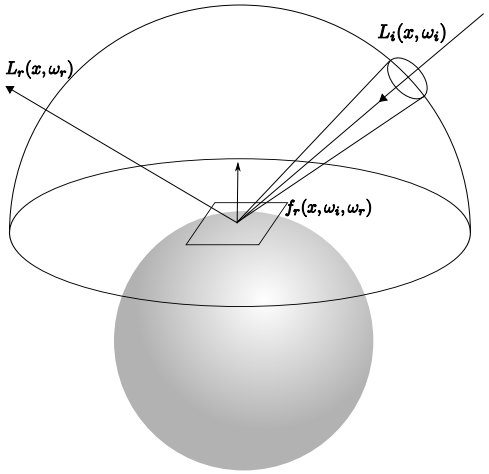

Figure 2.1: Functions of the rendering equation. The emission function is not shown. Light rays are portrayed as vectors. The tiny vector pointing upwards is the normal vector of the sphere at the location x. The scene shows one light ray coming from the right (ω_i) and being reflected on the sphere in the direction ω_r.

The reflection function $f_r(x, \omega_i, \omega_r)$ has three parameters which are the

location x on the intersected surface, the incoming ω_i and outgoing ω_r light direction. The function returns the percentage of how much of the incoming light from ω_i at the location x is reflected in the direction of ω_r.

The emission function $L_e(x, \omega_r)$ returns how much light the surface emits in a direction ω_r at the location x.

The incoming and outgoing light function return the amount of light which arrives at, respectively leaves from, the location x on a surface in direction ω_i, respectively ω_r. These functions are called $L_i(x, \omega_i), L_r(x, \omega_r)$, where L_i stands for incoming light, and L_r stands for reflected light.

With these four functions, James T. Kajiya formulated the integral equation which is known as the rendering equation. It is:

$$L_r(x, \omega_r) = L_e(x, \omega_r) + \underbrace{\int_A f_r(x, \omega_i, \omega_r) L_i(x, \omega_i) d\omega_i}_{(1)}$$

This equation states that the amount of light leaving a surface in a certain direction ω_r is the sum of the emitted and the reflected light in this direction. The integral (1) is exactly the amount of the reflected light, because it is an integral over the upper hemisphere of x (denoted by the letter A which stands for the area of the upper hemisphere). This means that the integral is the sum of all the reflected (in direction ω_r) incoming light of the upper hemisphere.

With the rendering equation, there is a formal construct which describes perfectly illuminated images; in other words, the lighting is most realistic. As one aspect of computer image synthesis is the creation of realistic images, it is a goal to solve the rendering equation. Normally, the rendering equation cannot be solved easily due to the complexity of the integral. Fortunately, there are methods of approximation.

A result from probability theory is that an approximation to the integral can be attained by calculating only a finite set of directions of the upper hemisphere and summing up the individual results. This method is called Monte-Carlo Integration. The "law of large numbers" justifies the Monte-Carlo Integration. In [10], further information about Monte-Carlo theory can be found.

Ray Tracing is one technique to approximate the rendering equation. However, it is rather imprecise in most cases.

2.2 Standard Whitted-Style Ray Tracing

In nature, each light source sends out light rays. These rays are refracted, reflected, and absorbed when they intersect with an object. In 1979, Turner

Whitted had the idea to simulate this behaviour by recursive ray tracing [11]. This will be explained now.

First of all, in a common 3D computer scene, there are often one to ten light sources. Therefore, the amount of rays in such an environment is enormous. Additionally, only a small fraction of all rays end in the observer's eye. In terms of computational cost, tracing the rays from the light source to the observer is too inefficient. Efficiency can be gained by reversing the tracing process; in other words, the tracing starts from the observer's eye.

In order to compute the image the observer sees, it is necessary to take the color of the reflected and refracted rays into account. In fact, each ray carries the color value of its intersection spot. The color values of the reflected and refracted rays are obtained by recursion (Figure 2.2).

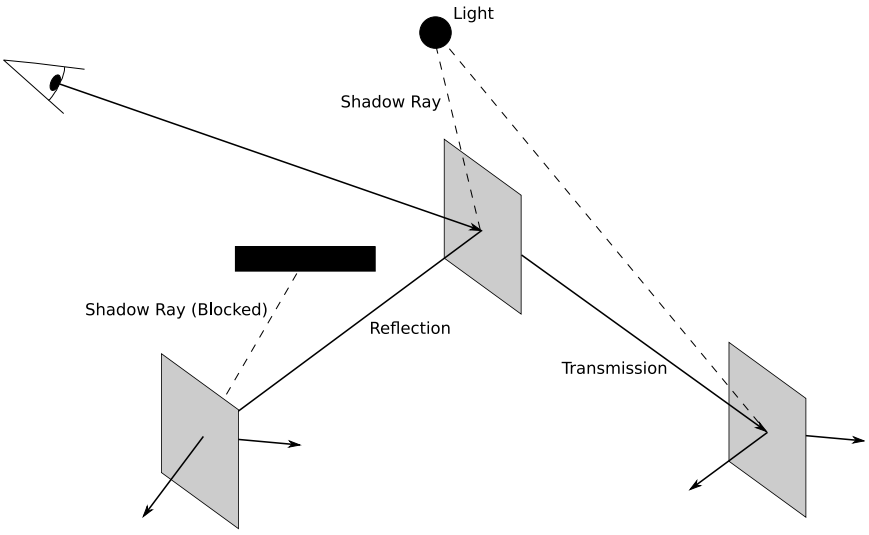

Figure 2.2: Standard Whitted-style ray tracing scene with reflection and transmission. One transmission, reflection and shadow ray is calculated for each intersection. Each of the transmission and reflection rays is traced recursively.

The ray tracing recursion starts with the sight rays, which originate in the observer's eye; they are also called primary rays. In Figure 2.3, it can be seen how a scene is projected on the image plane (usually the screen) with primary rays. The image plane is located between the observer (the camera in Figure 2.3) and the actual scene. Usually, the pinhole camera is used for the projection. This means, all primary rays have the same origin. Only their directions are different. Now, one primary ray is utilized for each pixel of the image plane. The location of the pixel center in the scene is used to calculate the direction of the corresponding primary ray. If a primary ray intersects with an object in the scene, the illumination is calculated

by sending out rays to each light source in the scene; these rays are called shadow rays. Shadow rays can either reach the light source or can be block by an object which is in front of the light source. As a result, it can be determined if the intersection point is in the shadow of an other object or if it is illuminated. In a next step, the reflection and refraction rays are traced recursively. Only one ray for the reflection and one for the refraction are traced with Whitted-style ray tracing. A maximal recursion depth must be given. Otherwise the recursion would never stop, which would lead to an infinite loop.

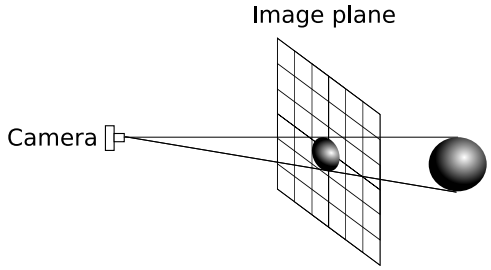

Figure 2.3: The primary rays are utilized to project the scene onto the image plane. The image plane resides between the observer and the scene. Standard ray tracing uses the pinhole camera model, thus all primary rays have the same origin but not the same direction.

For an image with a resolution of 800x600 there are 480000 primary rays given, one primary ray for each pixel. In a scene with one light source, for each ray that hits an object, three new rays are generated, namely the refraction, reflection, and the shadow ray. Each of the refraction and reflection rays will produce three new rays if they intersect with an object in the scene. Thus, the number of rays in a scene grows exponentially. By reversing the tracing process, so that the tracing starts from the observer's eye, it is possible to reduce the number of rays, but there are still millions of rays that must be traced. As a result, millions of intersection tests must be executed in order to decide if a ray intersects with any surface in the scene. One intersection test is executed quickly. However, there are millions of these tests, which lead to rather high computational costs.

In terms of realism, it is possible to produce very photo-realistic images with the principle of ray tracing, because the rays are treated as if they were real light rays.

Speaking of realism, as described in the previous section the rendering equation describes most realistic images. The Whitted-style ray tracing technique is only a poor approximation of the rendering equation. This is obvious, because only one ray and, therefore, only one direction is considered instead of the whole upper hemisphere. Hence, the images from this stan-

dard ray tracing algorithm will not be very realistic. Many effects which can be observed in nature are missing; for example depth of field and penumbra.

One way to add more realism to these images is to extend standard ray tracing by the idea of sampling more than just one ray. Cook, Porter, and Carpenter had this idea already in 1984, which was two years before James T. Kajiya presented the rendering equation. The new technique was called distributed ray tracing [3]. Nowadays, it is more often called stochastic or diffuse ray tracing.

2.3 Stochastic Ray Tracing

Generally, Whitted-style ray tracing is not capable of producing highly photo-realistic images. The reason for this lack of quality is that the rendering equation is approximated with only one ray. Stochastic ray tracing uses more rays to generate the images (Figure 2.4). For instance, instead of shooting one shadow ray from the intersection point to a light source, multiple shadow rays are calculated. The same idea is used for reflections and transmissions. It is also possible to use more primary rays. The resulting effects will be discussed in the following sections. The difference between stochastic and standard ray tracing is the idea of calculating additional rays.

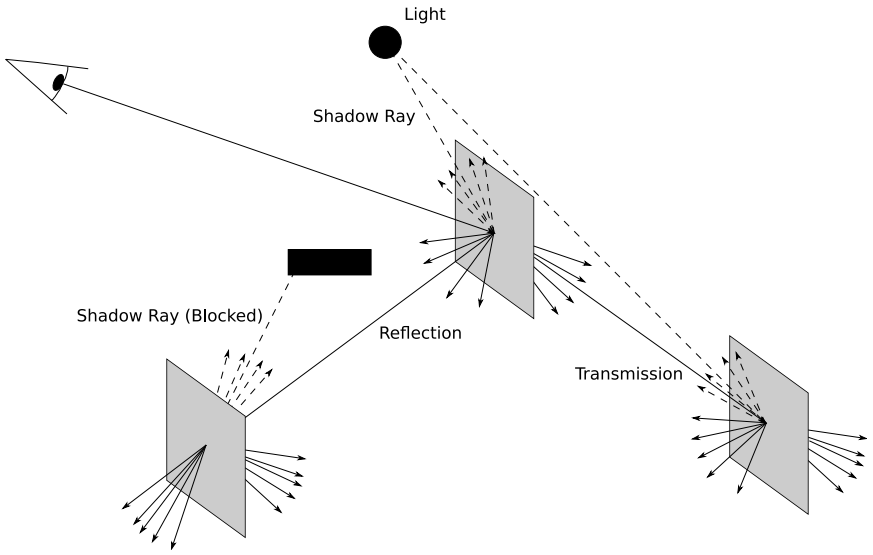

Figure 2.4: Stochastic ray tracing scene without multiple primary rays but with five transmission, reflection, and shadow rays. The diagram gives an idea of the exponential growth of the ray count.

By sending out more than one shadow, reflection and transmission ray per intersection, and more than one primary ray per pixel, several important

optical effects can be simulated. The following list gives an overview over all the effects which can be simulated with stochastic ray tracing.

1. Multiple light rays results in penumbras.
2. Multiple reflection rays results in blurry reflections.
3. Multiple transmission rays results in blurry refractions.
4. Multiple primary rays results in either an image with less aliasing or depth of field.
5. Multiple rays over a time span results in motion blur.

Each of the listed effects is essential for photo-realistic images. The following sections give an in-detail analysis of why stochastic ray tracing produces the effects listed above.

Another issue that needs to be mentioned is how the performance of a ray tracer decreases when the idea of stochastic ray tracing is implemented. A comparison of Figure 2.2 and Figure 2.4 shows how many more rays must be calculated as a result of sending out additional rays. The resulting performance loss is the price one must pay for obtaining more realistic images. In fact, this performance loss is more or less the only disadvantage that comes with stochastic ray tracing, but, unfortunately, it is a major disadvantage.

It was mentioned that additional rays are used, but the question remains what their origins and directions are. Obviously, it does not make sense that the additional rays have the same origin and direction as the ray from standard ray tracing. If they had, the final image would not be different at all, because no additional information could be retrieved from the scene. So, either the origin or the direction must differ. Figure 2.4 portrays how multiple shadow, refraction, and reflection rays are arranged, or rather, how the rays are distributed. It can be seen that multiple shadow, refraction or reflection rays have always the same origin but not the same direction. How the rays are distributed is discussed in Section 2.4. The situation is different with primary rays. When using multiple primary rays to produce depth of field, the origins and not the directions are changed. This will be explained in more detail in Section 2.3.5. With supersampling, it is the same as with reflection, refraction, and shadow rays. Here, the origins remain the same and the directions differ.

Now, it becomes clear why stochastic ray tracing is a better approximation of the rendering equation. With stochastic ray tracing more than one

ray is calculated. Thus, multiple directions of the upper hemisphere are considered, whereas in standard ray tracing only one direction is. The law of large numbers implies that stochastic ray tracing is a closer approximation, because more directions are taken into account.

2.3.1 Penumbra

With standard ray tracing, one shadow ray per light source is calculated; in other words, it is tested if the shadow ray is obstructed by an object in the scene or if the shadow ray reaches the light source. This procedure is done for every light source if there is more than one. This leads to a binary result, either the intersection spot is illuminated or it is in the shadow of an object. As a result, the shadows have sharp edges; hence, there is no gradient from a spot in the shadow to an illuminated spot. The color change happens abruptly. Thus, the image has no penumbras (Figure 2.5).

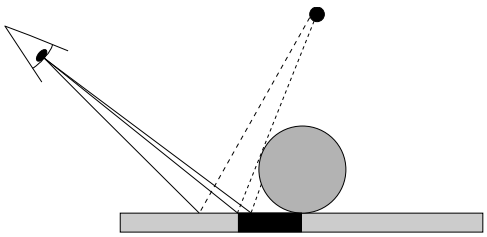

Figure 2.5: No penumbra with standard ray tracing, because a point on the plane is either illuminated by the light source, or not. This binary result produces sharp edged shadows.

However, penumbras occur in reality with almost every light source (Figure 2.6). The reason for this is that almost all light sources have an area and, therefore, produce penumbra. Hence, it is indispensable for photo-realistic images to have penumbras.

Figure 2.6: In reality, penumbra occurs often because most light source have an area and are, therefore, not point lights.

As already described, with stochastic ray tracing, multiple shadow rays are used. The rays are distributed over the surface of the light source. What is meant is that the shadow ray points at a spot on the surface of the light source (Figure 2.7). By the means of this technique, the result of the whole shadow calculation is not binary anymore. The result is a value between zero and one. It is zero, when all shadow rays are obstructed, and one, when all shadow rays are not obstructed. The value is the percentage of how much the spot is illuminated. Obviously, this enhances the image with soft shadows, in other words, with penumbras. It is clear that the more rays per light source are used the smoother the gradient will be. For example, when two shadow rays are used, then there are three percentage values. When twenty-five shadow rays are used then twenty-six percentage values can be calculated. If, however, more rays are used, the synthesis of the images takes longer, because more rays must be evaluated.

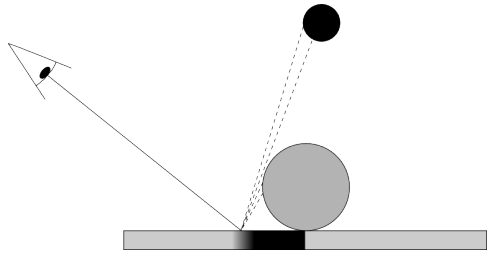

Figure 2.7: Stochastic ray tracing produces penumbra because an illumination percentage is obtained by sending out multiple shadow rays.

In conclusion, if multiple shadow rays are calculated for each light source, the image shows penumbras, which are very frequent in reality. Therefore, the image as a whole becomes more photo-realistic.

2.3.2 Reflection

If one pays attention to how reflections look like in reality, it can easily be observed that most reflections produce a blurry image. One of the few exceptions is the mirror. A mirror reflects its surroundings not blurred but as a perfect reflection. Despite the few exceptions, almost every reflection is blurry. A wooden or marmoreal table which is properly polished is such an example (Figure 2.8). The reflection is clearly visible, but it is blurry.

If only one reflection ray is calculated, as it is the case with standard ray tracing, reflections are always perfect. Just as it is the case with a mirror. That is because the reflection ray hits only one object, if it hits any at all. As a result, one point of the reflecting surface is the image of precisely one object (Figure 2.9). So, the reflection is perfectly clear and sharp.

With stochastic ray tracing, it is possible to produce blurry reflections. The idea is the same as with penumbras. Instead of just one, multiple

Figure 2.8: This marmoreal and wooden table shows a reflection. However, the reflection is not clear as it would be by a mirror; the reflection is blurred. It can also be seen that the blurring degree is related to the material of the surface. The reflection of the wooded side is more blurred that the reflection of the marmoreal side.

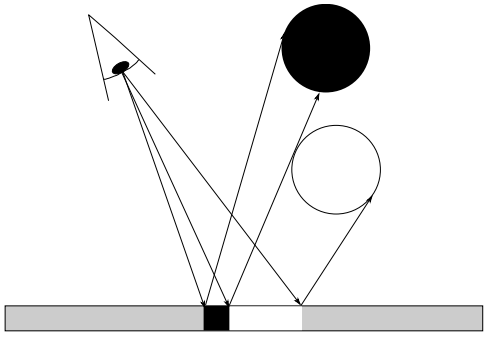

Figure 2.9: Standard ray tracing shows only clear reflections because one spot of the reflecting surface is the image of exactly one object.

reflection rays are sent out from an intersection point (Figure 2.10). Now, a spot on the reflecting surface is not only the reflection of one object but an average of several objects. This averaging process is the key to blurry reflections. Thus, it is possible to generate images with blurry reflections by using more than just one reflection ray.

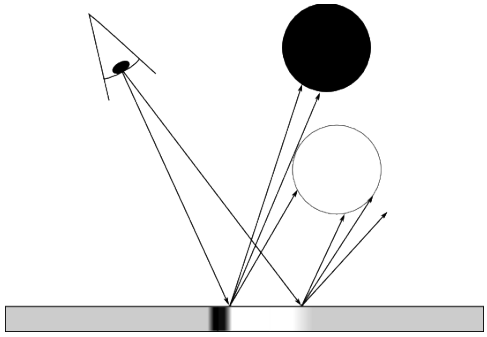

Figure 2.10: Stochastic ray tracing shows blurry reflection by sending out additional reflection ray. As a result, one spot on the surface is not the image of one object anymore; it is the average of several objects.

2.3.3 Translucency

In reality, there are materials which refract light in a way that the image is blurred. Compared to reflections, blurry refractions are less common, because if an object is transparent then one does rarely want to have a blurred image. For example, a window should not distort the object behind it. Thus, highly transparent objects have little blur in reality. Still, there are transparent objects which blur the objects behind them. A Plexiglas with a rough surface or objects that are not planar, a bottle (Figure 2.11) for instance, are examples of object that blur transmissions.

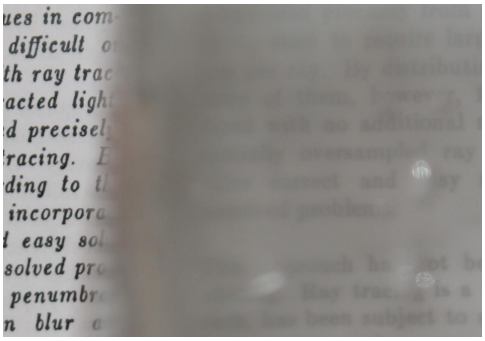

Figure 2.11: The blurry refraction through a PET bottle is cause by the non-planar shape of the bottle.

In general, the ray behaviour of transmissions is more or less the same as with reflections. If there is only one transmission ray, the objects behind the transparent material are perfectly sharp. As it was mentioned, there are situations where this sharp transparency is incorrect. Blurry transmissions can be achieved by sending out multiple refraction rays.

Standard ray tracing is only capable of producing transparency without blur effect. To get rid of this lack of realism, one can easily calculate more than one transmission ray. As reflections are very similar to transmissions, the same argumentation holds.

2.3.4 Supersampling

It is a known issue that in a raster image a line cannot be displayed without distortion. The line looks like a stair if it is neither horizontal nor vertical. This effect is called aliasing. Methods to reduce aliasing are called "antialiasing" techniques. Supersampling is one antialiasing technique [4].

In general, only one primary ray is assigned to a pixel of the image plane. With supersampling, not only one primary ray per pixel is calculated but several rays. Usually, the supplementary rays are aligned in a grid (Figure 2.12). So, it is possible to look at supersampling as a technique that virtually enlarges the image resolution. Normally, one pixel is split into a square number of virtual subpixels. The final pixel's color is the average of all color values from the virtual subpixels.

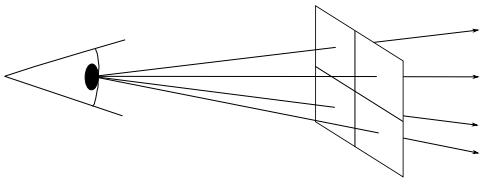

Figure 2.12: With supersampling, a physical pixel is split into virtual subpixels by using multiple primary rays. The color of the physical pixel is the average of its virtual subpixels.

2.3.5 Depth of Field

Depth of field is the area around the focal plane where objects are in focus when a lens camera model is used. Any object either farther away or closer to the camera is blurred and out of focus. Actually, every object either in front or behind the focal plane is out of focus (Figure 2.14). However, in the area of depth of field, the objects are still very sharp. Thus, they seem to be in focus. The size of this area is given by the aperture; the bigger the aperture the smaller the area of depth of field. In contrast to the lens camera model, when using a pinhole camera every object is in focus, no matter how

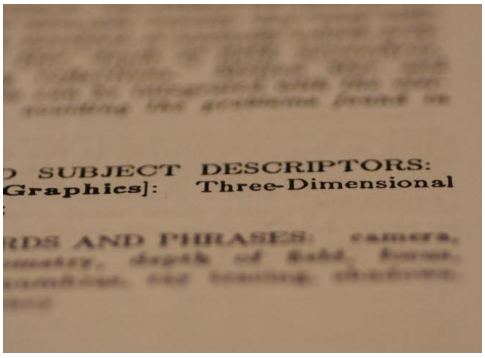

Figure 2.13: A picture taken with a lens camera has a limited depth of field. The foreground and the background are blurred; only the text in the center is in focus.

far away it is. The human eye is not a pinhole camera, it is a lens camera. Therefore, humans have a limited depth of field (Figure 2.13). As a result, humans consider images with a limited depth of field as realistic, whereas they consider an image with an infinite depth of field as unnatural. Hence, if a computer generated image has a limited depth of field it automatically looks more real than with an infinite depth of field. Standard ray tracing uses the pinhole camera as its camera model by assigning one primary ray to one pixel. Therefore, an image rendered with standard ray tracing looks artificial due to the infinite depth of field.

Depth of field can be added by using more than one primary ray. The idea is to simulate a thin-lens camera model by sending out multiple primary rays. To understand why this leads to a limited depth of field and puts objects out of focus, one has to look at how a lens camera is built (Figure 2.14).

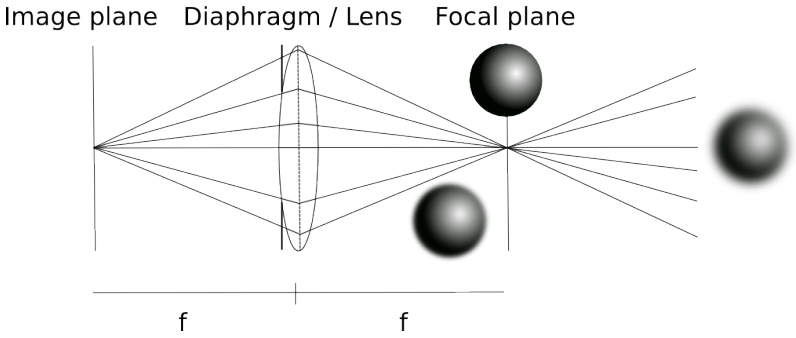

Figure 2.14: A thin lens camera has a focal length f. The distance between the image plane and the lens is equal to the distance between the lens and the focal plane. This distance is f. Only the objects near the focal plane are in focus; all objects closer or farther are blurred.

With a thin lens camera, the distance between the image plane and the lens is equal to the distance between the lens and the focal plane. This distance is the focal length. The only objects in focus are those on the focal plane. However, the objects which are close to the focal plane appear to be in focus too; just as described at the beginning of this section. The area where objects seem to be focused is the depth of field area and it is determined by the aperture; the smaller the aperture the bigger the area. In Figure 2.15, it can be seen that the projected image on the image plane is upside down. To fix this, the image plane is put between the focal plane

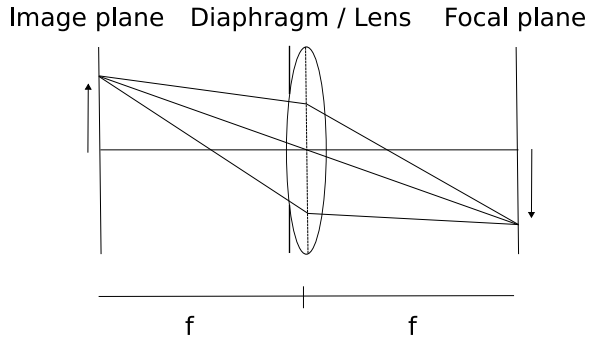

Figure 2.15: The projected image on the image plane is upside down.

and the lens, similar to the pinhole model. It needs to be noticed that all rays which contribute a color value to one pixel, intersect on the focal plane in a single point. Now, to see the difference between the pinhole camera and the lens camera Figure 2.16 shows the two models overlayed. It can be seen that with the lens model, all rays of one pixel go through the same spot on the focal plane, namely the intersection of the focal plane and the primary ray of the pinhole model. The directions of the additional rays are given by this intersection spot and their origins. The origins are different for each ray; hence, their directions differ as well. The origins are distributed over the area of the lens, which is the aperture.

In conclusion, the additional rays originate on the lens and go through the intersection point of the primary ray of the pinhole model and the focal plane. Furthermore, the aperture can be controlled by the size of the area over which the origins are distributed. The focal length can be controlled by moving the focal plane and, therefore, the intersection spot. Finally, the average of all rays is taken for the final pixel color. It is clear that, by using this technique, only the objects close to the focal plane are in focus and the rest is not. How close the object must be to be in focus depends on the aperture. The result is a limited depth of field.

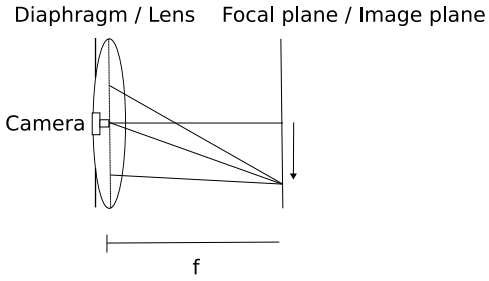

Figure 2.16: The lens and pinhole camera model are overlayed to show how the additional rays differ. All rays of one pixel intersect in a single point on the focal plane. Namely, the intersection point of the primary ray of the pinhole camera model and the focal plane. The origins of the additional rays are distributed over the aperture of the lens.

2.3.6 Motion Blur

Motion blur occurs when a still image is not taken with a infinitely fast shutter. This means the image represents a time span a not only a single moment. When objects move in this time span, they become blurry (Figure 2.17). This effect is called motion blur; it is different compared to the previous effects, because the motion blur effect only occurs with animations.

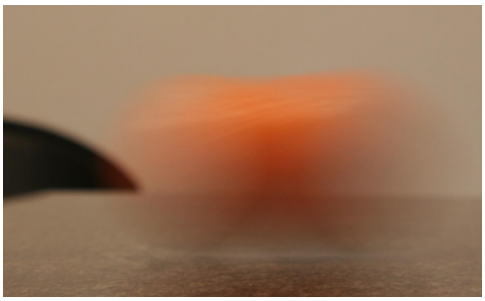

Figure 2.17: Motion Blur occurs when an image represents a time span and an object moves within this time span.

It is possible to simulate motion blur with stochastic ray tracing. Again, the idea is to use more than one ray per pixel. Although, this time every additional ray is assigned to one frame in an animation. The color of a pixel in the final image is the averaged color value of the rays distributed over the time span. This averaging process produces the motion blur effect.

2.4 Ray Distributions

The basic idea of stochastic ray tracing is to use multiple rays. The remaining questions are where the origins and what the end points of the new rays

are; to put it in other words, how the rays are distributed. There are many possibilities how one can distribute rays. The five most common distributions will be discussed now. These are the gird, jitter, semi-jitter, Poisson and random distribution. Figure 2.18 shows examples of how points might be distributed by using the corresponding distribution method.

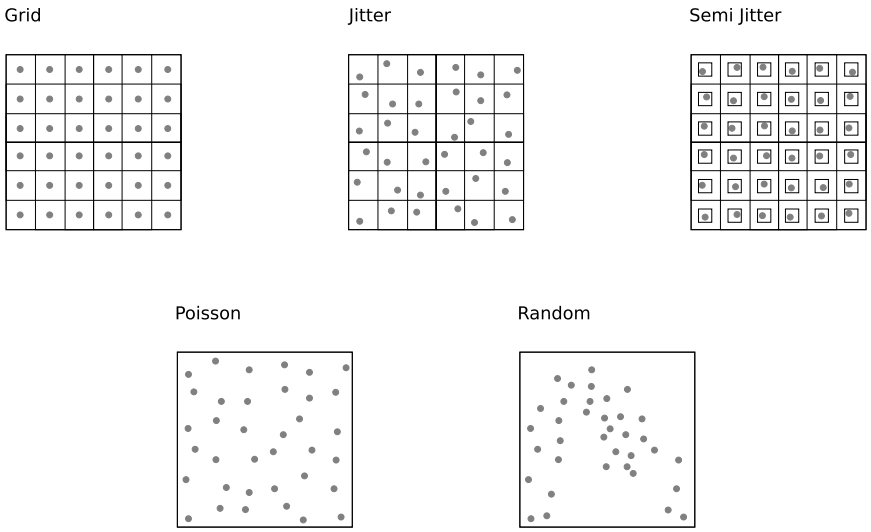

Figure 2.18: Five most common distributions

Random The random distribution is probably the simplest distribution. The probability for a point occurring at a specific location is the same for all locations. Unfortunately, it can happen that the points are unevenly distributed. For example, if n points are distributed and all are placed in the upper left corner and not a single one in the bottom right corner.

Grid With the grid distribution all points have the same distance from one another. On the one hand, the disadvantage of the random distribution is not present. On the other hand, Moiré patters might occur, which apparently decrease photo-realism.

Jitter The jitter distribution lays out the points in a grid and jitters them randomly within a subsquare. This reduces the Moiré effect at a small efficiency loss.

Semi Jitter The semi-jitter distribution is the same as the jitter distribution, with one small difference. The square within which the points are randomly moved is only half the size of the jitter distribution.

Poisson The Poisson distribution is the most sophisticated of the five distributions. Theoretically, there should be hardly any Moiré effects,

and the disadvantage of the random distribution is not present. The disadvantage compare to the other distributions is that the computational cost is higher. The following paragraph describes how the location for a point is determined.

The location for each point is found by the following procedure. For each new point the random distribution is used. However, if the new point is too close to a point which is already placed, then the new point will not be set at this position. A new position is chosen randomly and again, if it is too close to another point, the new one is dismissed once more, but if the distance is big enough the position is selected. This process is repeated until a given number of points is set.

A fast algorithm for the Poisson distribution was presented by Dunbar and Humphreys [5].

To sum up this chapter, the theory of stochastic ray tracing was explained by first presenting the rendering equation, which describes perfectly illuminated images. Secondly, standard Whitted-style ray tracing was introduced as an approximation of the rendering equation; however, a rather imprecise one. Thus, several light effects which occur in reality are missing from images rendered with Whitted-style ray tracing. Then, stochastic ray tracing was presented as a technique to enhance images with some of the missing light effects, namely, penumbra, fuzzy reflection and refraction, depth of field, and motion blur. The main idea is to calculate additional direction, which results in a closer approximation of the rendering equation. Eventually, five ray distributions were presented. The next chapter is about the actual implementation of stochastic ray tracing.

Chapter 3

An Implementation

Lumen is a ray tracing application, developed by Ph. Robert. The main goal of this bachelor's project was the extension of Lumen by stochastic ray tracing. The major goal Philippe is pursuing in his PhD thesis "Streaming Ray Tracing on a Non-Uniform Graphics Architecture" is real-time ray tracing [8]. Considering his goal of real-time ray tracing, the addition of stochastic ray tracing is a major drawback. This is because of the additional rays, which lead to additional intersections tests and ray traversals. This slows down the generation of images. Obviously, this counteracts the purpose of Lumen being a real-time ray tracer. However, this bachelor's project does not focus on producing images in real-time. Its focus is on the improvement of image quality; i.e. to make the images more photo-realistic.

In this chapter, the actual implementation will be explained, especially the implementation of the different distributions. This is the core of stochastic ray tracing, because all features are just an application of a certain distribution in a specific situation. Finally, two acceleration methods are proposed at the end of this chapter.

3.1 Basic Algorithm

As stated before, the implementation of the features itself is just an application of the distributions. The following steps are executed for each recursion:

1. Determine the original ray from standard ray tracing (perfect reflection, refraction, center of the light source, center of the pixel).
2. Apply a distribution of choice to either its origin or its end point.
3. Trace each of the new rays (the ray from 1 is not traced).
4. Return the color of the location by calculating the average of all new rays.

3.2 Ray Distributions

A very important matter of stochastic ray tracing is how the rays are distributed. In Section 2.4, five distributions have been introduced. This section now explains how the distributions are implemented in lumen. For the calculation of the new directions, the origin and a further point are required. The latter is called end point. The direction results through vector subtraction. In the following part it is assumed that the end point of a ray is distributed and not its origin. The idea applies also for the distribution of the origin.

The main idea for all distributions is to find a normal plane. It is supposed to be normal to the ray around which the distribution happens (Figure 3.1). This ray is called the original ray, because it would be the ray from Whitted-style ray tracing. A plane can be represented by a vector basis. Thus, two vectors that span the normal plane need to be found. Now, additional rays are scattered around the original ray. The maximal distance between the end points of a distributed ray and the original ray is $\sqrt{2}\cdot$`radius`. This means that the end point of the original vector is in the center of a square (Figure 2.18). As soon as the two basis vectors are given, only the two dimensional coordinates of the point in the square of the desired distribution are required, because the three dimensional coordinates of this point can found through the vector basis. The two dimensional coordinates define a vector in the normal plane. This vector is called translation vector. The distributed ray results through vector addition of the original ray and the translation vector.

The portrayed square in Figure 3.1 is the situation for all distributions except the two random distributions. The correct random distribution uses polar coordinates. Therefore, the square is replaced by a circle. The case for the fast random distribution is different and will be explained later.

The following sections describe how all distributions are implemented. In the code the variables `origRay`, `distributedRay_i` are three dimensional vectors that represent the end points of the rays. `distributedRay_i` are the new rays for stochastic ray tracing. The value of `radius` is a given number. RND() is a function that returns a random value between zero and one. For the grid, jitter, and semi-jitter distribution, the number of rays must be square. Otherwise, the coordinates in the normal plane are ambiguous. For the random and Poisson distribution the number of rays can be any integer bigger than zero.

Grid Distribution

First, it is necessary to find the vector basis $\{u, v\}$ for the normal plane (Listing 3.1). This is the done for all distributions except the fast random distribution.

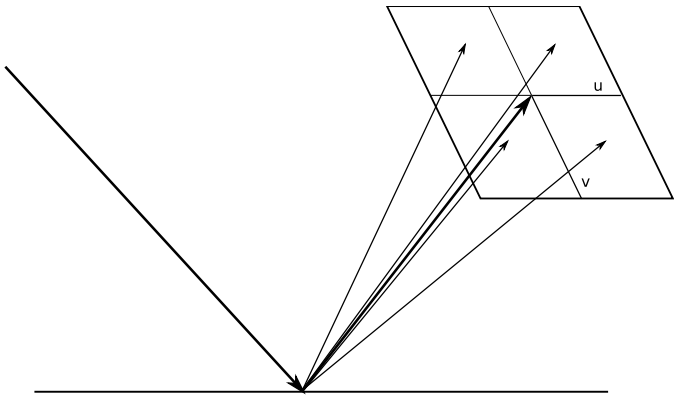

Figure 3.1: This diagram shows stochastic ray tracing with the grid distribution. The bold vector in the middle of the square is the one of the perfect reflection. The normal plane which is spanned by $\{u, v\}$ is a sector of the upper hemisphere. $\{u, v\}$ is determined first for all ray distributions except the fast random distribution. Then, additional reflection rays point to a location on this normal plane. The portrayed square is the case for all distributions except the random distribution. There, the square is replaced by a circle because of the polar coordinates.

Listing 3.1: Calculation of basis vectors $\{u, v\}$

```
1  IF (origRay.x==0 && origRay.y==0 && origRay.z!=0) THEN
2      u = (0, origRay.z, 0)
3  ELSE
4      u = (origRay.y, -origRay, 0)
5
6  NORMALIZE(u)
7
8  // cross product on outgoing vector and u
9  v = VECTOR_CROSSPRODUCT(origRay, u)
10
11 NORMALIZE(v)
```

The two dimensional coordinates of the distribution, which are regarded as the translation vector, needs to be calculated in a next step. Finally, the translation vector is added to the original ray (Listing 3.2).

Listing 3.2: Calculation of the grid coordinates and distributed rays

```
12 sqrtOfNumRay = SQUARE_ROOT(desiredNumOfRays)
13
14 FOR i = 0  TO i = desiredNumOfRays DO
15    uCoord =  radius*(1 - 2*((float)(2*((int)i/
          sqrtOfNumRays + 1)- 1)/(2*sqrtOfNumRays)))
16    vCoord =  radius*(1 - 2*((float)(2*((i MODULO
          sqrtOfNumRays) + 1) - 1)/(2*sqrtOfNumRays)))
17
18    translation_vector = VECTOR_DOTPRODUCT(u,uCoord) +
          VECTOR_DOTPRODUCT(v,vCoord)
19    distributedRay_i = origRay + translation_vector
20 ENDFOR
```

Semi-Jitter Distribution

Again, the basis vectors are calculated first (Listing 3.1). The two dimensional coordinates for the translation vector is the only difference to the grid distribution. Hence, lines 15 and 16 are replaced by the code of Listing 3.3. The idea is to calculate the grid coordinates `uCoord,vCoord` first (line 15 and 16). Then `uCoord,vCoord` are randomly moved within a square with a length of `radius / 2`. In other words, random values $(x, y) \in [-radius/2, radius/2]^2$ are calculated, and added to `uCoord,vCoord` (line 18 and 19). `uCoord,vCoord` are now the two dimensional coordinates for the semi-jitter distribution.

Listing 3.3: Calculation semi-jitter coordinates

```
15    uCoord =  radius*(1 - 2*((float)(2*((int)i/
          sqrtOfNumRays + 1)- 1)/(2*sqrtOfNumRays)))
16    vCoord =  radius*(1 - 2*((float)(2*((i MODULO
          sqrtOfNumRays) + 1) - 1)/(2*sqrtOfNumRays)))
17
18    uCoord = uCoord + (RND()*2-1)*radius/(sqrtOfNumRays*4)
19    vCoord = vCoord + (RND()*2-1)*radius/(sqrtOfNumRays*4)
```

Jitter Distribution

The jitter distribution differs only in two places from the semi-jitter distribution. Namely, the random values are now $x, y \in [-radius, radius]^2$.

Thus, the pseudo code is the same as in Listing 3.3 except that the lines 18 and 19 from Listing 3.3 are replaced with the lines 18 and 19 of Listing 3.4. Here, the division by two of the latter summand is missing, because `uCoord,vCoord` are randomly moved within a square with a length of `radius` and not `radius/2` as before.

Listing 3.4: Calculation jitter coordinates and distributed rays

```
18    uCoord = uCoord + (RND()*2-1)*radius/(sqrtOfNumRays*2)
19    vCoord = vCoord + (RND()*2-1)*radius/(sqrtOfNumRays*2)
```

Poisson Distribution

To avoid ambiguity in this section, two terms are introduced. First, the term Poisson distribution refers to distribution of point in an n-dimensional space. Second, the term Poisson ray distribution refers to the distribution of rays where either the origins or the end point are Poisson distributed.

The basic concept of the Poisson distribution was already explained in Section 2.4. There are many different algorithms for the Poisson distribution in n-dimensional spaces. Dunbar and Humphreys [5] analysed some algorithms for speed complexity in great detail. However, the paper by Bridson [2] is far simpler. It gives a step-by-step instruction on how a Poisson distribution can be generated. The latter was the basis for the implementation of the Poisson ray distribution in Lumen. `uCoord,vCoord` are generated with the proposed algorithm of Bridson. This results in the translation vector, which is added to the original vector for the new ray.

Correct Random Distribution

Two random distributions were implemented; a slow uniform and a fast non-uniform. The idea behind the correct random algorithm is to use polar coordinate. First, a random angle `phi`$\in [0, 360]$ is chosen, then, a random radius `rad`$\in [0, radius]$. These two values are interpreted as polar coordinates that specify a location on a disk. These polar coordinates are then transformed in $\{u, v\}$ coordinates `uCoord,vCoord`, which can be seen as the translation vector from the previous distributions (Listing 3.5). Finally, the distributed ray is given by vector addition of the translation and the original vector.

Fast Random Distribution

Unfortunately, the calculation of the two basis vectors takes some time. Considering the random distribution, this time is more or less wasted, because by using the random distribution the rays might be unevenly distributed. For

Listing 3.5: Correct Random Distribution

```
12 FOR i = 0  TO i = desiredNumOfRays DO
13    phi = RND() * 360
14    rad = RND() * radius
15
16    uCoord =  rad * COSINUS(phi MODULO 360)
17    vCoord =  rad * SINUS(phi MODULO 360)
18
19    translation_vector = VECTOR_DOTPRODUCT(u,uCoord) +
          VECTOR_DOTPRODUCT(v,vCoord)
20    distributedRay_i = origRay + translation_vector
21 ENDFOR
```

example, it might happen that all random points are in one particular corner. To get a random distribution in faster time, it is possible to use the following algorithm. First, pick a random vector $(x, y, z) \in [-radius, radius]^3$, (x, y, z) can be seen as the translation vector. The resulting random ray is given by vector addition of the original ray and (x, y, z). Listing 3.6 shows the pseudo code of this algorithm.

Listing 3.6: Fast Random Algorithm

```
1 FOR i = 0  TO i = desiredNumOfRays DO
2    x = ( RND() * 2 * radius ) - radius
3    y = ( RND() * 2 * radius ) - radius
4    z = ( RND() * 2 * radius ) - radius
5
6    translation_vector = (x, y, z)
7    distributedRay_i = origRay + translation_vector
8 ENDFOR
```

This algorithm does not distribute the rays uniformly. This means that the probability for a new ray having a certain direction is not the same for all directions. Assuming the original ray enters the box through a corner. The probability that the direction of the random ray is the same as the original rays is bigger than when the original ray enters in a right angle to a surface of the box. When the end point of a random ray lays on the straight line spanned by the original vector, then the original ray and the random ray have the same direction. Now, because the diagonal is the longest line inside a cube, the probability that the random and the original ray have the same direction is higher in the first scenario. Both scenarios might occur because the cube is always in alignment with the coordinate system (Figure 3.2).

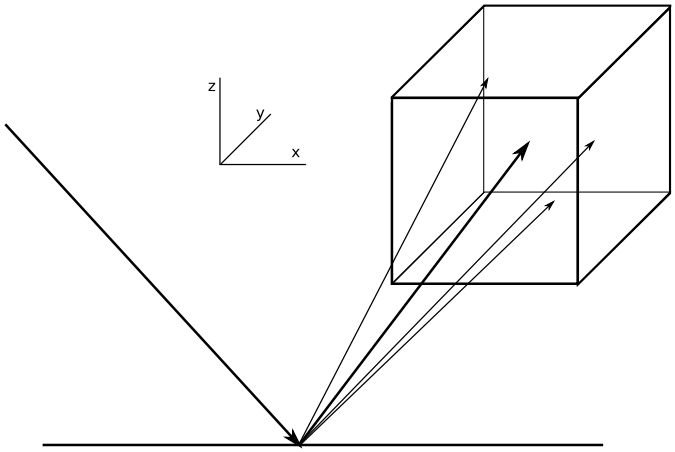

Figure 3.2: A random translation vector within a cube with a side length of 2·`radius` is chosen for the fast random algorithm. The distributed rays result from vector addition of the translation vector and the original ray.

3.3 Features

Penumbra, blurry reflection and transmission, depth of field, and supersampling are the features which were added within the scope of this bachelor's project. For penumbra, reflection, and transmission it is possible to choose from all the introduced ray distributions. For depth of field and supersampling, only the grid distribution is available. Motion blur is not implemented, because the performance of Lumen on current hardware is not high enough for rendering animations.

To apply the algorithm in Section 3.1, the radius, and whether the origins or the end points of the additional rays are distributed need to be specified. These information are required for all distributions. The following list describes how the ray distributions are applied for each feature.

Penumbra As described in Section 2.3.1, the radius is given by the size of the light source. For penumbra, the end points, which determine the directions of the rays, are distributed.

Reflection / Refraction The end points are distributed as well. For reflections and refractions, it holds that the bigger the radius the blurrier the image. Thus, the radius is a material parameter. However, this parameter is not given by most formats. Hence, the radius is derived from the reflexivity value of the material. In Lumen, `(1-reflexivity )*0.2` was used. This expression was found by experiment and has no claim to be physically correct. Still, it seems to be a sensible radius.

Supersampling The radius is half the size of a pixel. If the radius is bigger, some primary rays of a pixel go through its neighbor pixel. As seen in Section 2.3.4, the end points are distributed with the grid distribution.

Depth of Field The radius is the aperture, which is given by the user. This is the only feature that requires the origins to be distributed. Just as before, for the origins the grid distribution is used. The direction of the rays are specified by the end points, which are located at the intersection of the focal plane and the primary ray of the pinhole model. The distance between the lens and the focal plane is given by the focal length.

3.4 Acceleration Metrics

If a scene contains relatively big objects, then many reflection rays can be omitted because the blur effect is only visible near the edge of an object. In the center of a reflected object, one ray would be sufficient, because any additional ray would only hit the same object and, therefore, would not change the color value. Thus, it is possible to reduce the number of rays.

One idea is to use metrics. A possible metric is to measure the change between the final color with $n-1$ rays and n rays. If the difference between the two color values is lower than a given threshold, then no additional ray will be calculated. Apparently, this method recognizes the previously described situation, because if a reflection is of the center of a big object, then the color value does not change significantly with additional rays. Hence, redundant rays can be saved.

It could happen that the color difference between $n-1$ and n is below the given threshold but with the next ray the color would change distinctively. This effects occurs often with the first couple of rays. To prevent the metric from a wrong decision, one can specify a minimal number of rays (α) which must be processed. Hence, the metric is applied after the calculation of more than α rays. Still, it might come to a wrong decision, but less frequently.

A similar metric can be used to reduce the number of shadow rays. The idea here is to check after a given number of light rays if all reached the light source, or if all were blocked by an object. If either is the case, then it can be assumed that any additional ray would reach, respectively would not reach the light source as well. Thus, any additional ray would not change the shadow percentage. Obviously, the higher the given number of rays the more accurate the final image will be. Once again, it is clear that some wrong decisions might be made, similar to the previous metric, but the difference is for both metrics hardly noticeable. The effects of both metrics will be discussed in the next chapter.

When the jitter, semi-jitter or grid distribution is used, it is important not to calculate "sequential" rays. That is, when the ray in the upper left corner is calculated first and then the ray next to it and so on. Otherwise, the bottom right corner is practically never considered, which is not optimal. Hence, one has to pay attention to the order in which the rays are traced.

This chapter showed a possible implementation of stochastic ray tracing. Furthermore, an idea to accelerate the image generation by the means of metrics was proposed. In the next chapter, the image quality and the performance will be analysed for each of the implemented features.

Chapter 4

Evaluation

The Stanford bunny is used in various scenes for the evaluation. It consists of 35,947 vertices, which results in 69,451 triangles. In addition, there are around 100 triangles from other objects like the surface on which the bunny sits. The recursion depth is three, and the resolution is 800x600 for all images. However, most of the screenshots are cropped.

Each feature will be discussed separately in terms of image quality, performance, and metric, if available. For the discussion of the performance, the ray count and not the computation time will be given. That is because the computation time may vary distinctly on different hardware. The ray count, however, is independent of the hardware settings.

4.1 Supersampling

The goal of supersampling is to eliminate aliasing effects. Four screenshots of the Stanford bunny were rendered to illustrate the gained image quality. The top left image in Figure 4.1 was rendered without supersampling. The image on the top right hand side (in Figure 4.1) was rendered with 4x supersampling, which means four primary rays were used per pixel instead of just one. The images at the bottom were rendered with 8x (left) and 16x (right) supersampling.

Image Quality

The aliasing effect is clearly apparent in Figure 4.1 (left). All edges of the bunny show the distinctive staircase effect which is typical for raster images. The edges of the top right image have only very little aliasing. What is remarkable, is that with only 4x supersampling the aliasing effect is barely noticeable. 4x supersampling is the lowest possible version of supersampling with subdividing a pixel in a square number of virtual pixel. Higher than 4x supersampling does not increase the image quality any further (Figure 4.1

bottom left and right). Thus, 4x supersampling should be used at most.

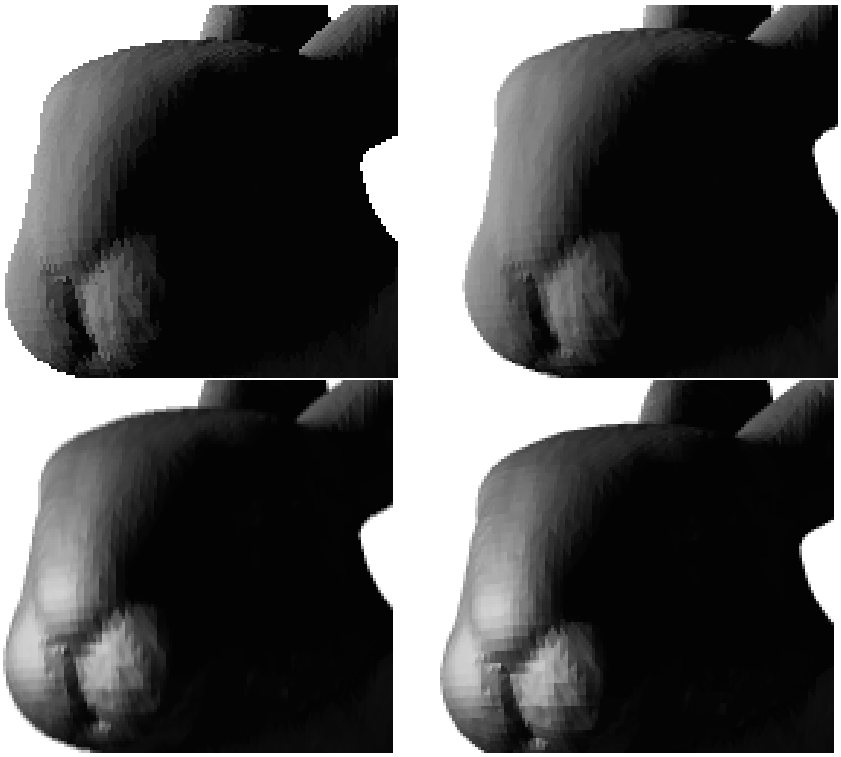

Figure 4.1: A crop of the bunny's head. The aliasing effect is clearly visible without supersampling (top left). With 4x supersampling almost all aliasing effects disappeared (top right). Higher supersampling does not increase image quality any further. 8x supersampling was applied for the bottom left image and 16x supersampling for bottom right image.

Performance

4x supersampling produces at most four times more rays than without supersampling. This, in return, takes approximately four times longer to produce the image. The reason for this is because there is the fourfold of pixels present. These are actually only virtual pixels. Therefore, four times more primary rays need to be traced. This is acceptable, because the number of rays grows linearly. Table 4.2 and Table 4.3 show this behaviour.

4.2 Depth of Field

Six images were rendered to show how depth of field affects the image quality and the performance. It is possible to modify two parameter for depth of

field. One is the aperture, which regulates the blur degree of objects that are out of focus, and the other is the focal length, which regulates the distance between the lens and the focal plane. The first image has a small aperture and a short focal length (Figure 4.2 top left). The second image has bigger aperture, but the same focal length (Figure 4.2 bottom left). Thus, the bunnies in front are in focus and the rear bunnies are blurred in different degrees. The two images on the right in Figure 4.2 have a longer focal length as the images on the left. Therefore, the rear bunnies are in focus. The apertures of the images on the right correspond to the apertures on the left. Four primary rays were used for all four images. The remaining two screenshots show the dependency between aperture and the primary ray count. Figure 4.3 shows a big aperture with 8 (left) and 16 (right) primary rays.

Image Quality

The four images in Figure 4.2 seem very realistic because of depth of field, even with only four primary rays per pixel. The focus effect is very similar to reality (Figure 2.13). In Figure 4.2, the apertures have a reasonable size.

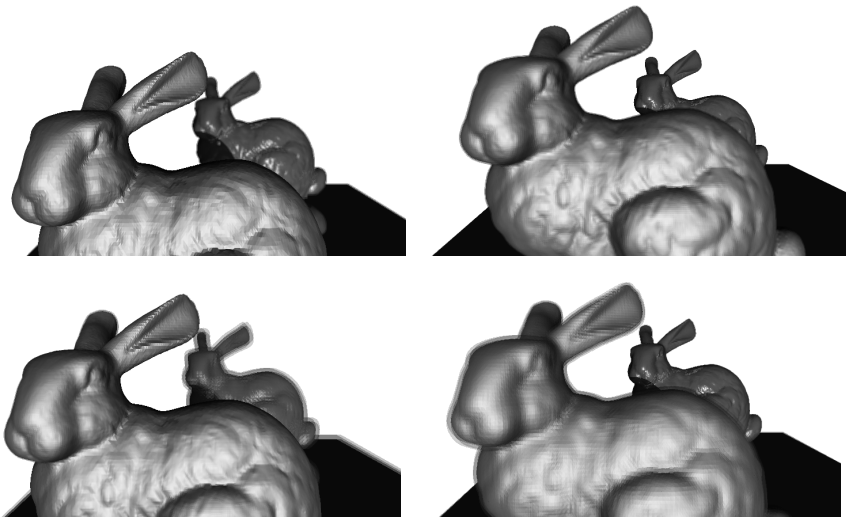

Figure 4.2: A demonstration of different settings for the aperture and focal length. Small aperture and short focal length (top left), small aperture and long focal length (top right), wide aperture and long focal length (bottom right) as well as wide aperture and short focal length (bottom left) were used for the four screenshots.

Thus, four primary rays are enough to produce an image with a decent blur. However, when its size is chosen to be wider, then four primary rays are not sufficient. The left image in Figure 4.3 does not have a smooth blur with

even eight primary rays. The screenshot on the right shows that a minimum of sixteen primary rays are required for a smooth image.

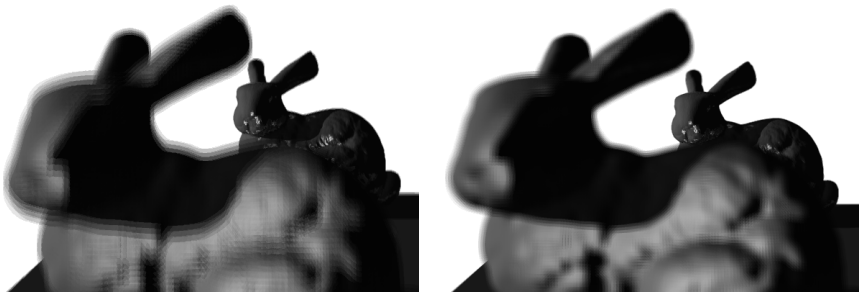

Figure 4.3: If the aperture is rather wide, a minimum of 16 primary rays is required for a smooth image. The left screenshot was rendered with 8 primary rays; hard edges can be seen in the blurred bunny. 16 primary rays were used on the right; for this wide aperture, this seems to be the minimum for a smooth image.

Performance

As with supersampling, depth of field with n primary rays produces at most n-times more rays. The argumentation remains the same as before. This means that the computational cost grows linearly, as well (Table 4.1). This is outstanding because depth of field is rather important for photo-realism and the additional costs are reasonable.

# Primary Rays	1 Ray	4 Rays	8 Rays	16 Rays
	$1 \cdot 10^6$	$3.5 \cdot 10^6$	$7.3 \cdot 10^6$	$14.8 \cdot 10^6$

Table 4.1: Depth of field shows a linear growth in the number of primary rays; just as supersampling.

4.3 Transmission and Reflection

For supersampling and depth of field one can only use the grid distribution. Now, with transmission and reflection all five distributions can be used. A scene with the Stanford bunny standing on a reflecting surface was rendered to demonstrate how the images quality differs with each distribution. Only two screenshots are shown for the transmission because the effects are the same as with the reflection.

Reflection Image Quality

The images in Figure 4.4 were rendered with 4x supersampling, and four instead of one reflection ray. The only exception is the top left image. Only one ray was used there. All distributions produce an acceptable image quality when many rays are considered; i.e., thirty two or even more. To show the disadvantages and differences between the distributions, the images were rendered with four rays only.

The grid distribution does not produce an acceptable result. Each of the four rays produces its own reflection. In the final image, these four reflections are combined and each is clearly visible (Figure 4.4 top right). Thus, the reflection with grid looks very artificial. The semi-jitter distribution is a bit better, but the individual reflections are still visible; just the edges are not as sharp as with the grid distribution (Figure 4.4 center left). The jitter distribution is already much better. The images of the different rays are hardly noticeable (Figure 4.4 center right); it is very similar to the random distribution. However, the latter shows a pixelation in the whole image (Figure 4.4 bottom right), whereas the jitter distribution shows the pixelation only near the edges. This is not the case with the other distributions. As for the Poisson distribution, the image quality is equal to the random distribution (Figure 4.4 bottom left). The pixelation is different, but in both cases equally apparent.

Figure 4.5 was rendered with random distribution, thirty-two reflection rays and 4x supersampling. Here, the pixelation disappeared almost completely. The minimal number of rays to produce a smooth reflection changes with the material properties. Fewer rays are needed if the reflection is less distorted and vice versa. The disappearance of disadvantages can be observer with all other distributions, as well. For example, if thirty-two ray are used with the grid distribution, the reflections of the separate rays can neither be seen. Hence, to produce a smooth reflection a fairly high number of rays must be traced. Otherwise, the pixelation or the multiple reflections are clearly visible, which obviously reduced the photo-realism of an image. Still, with stochastic ray tracing it is possible to simulate all kinds of reflecting surfaces in a very photo-realistic way.

Transmission Image Quality

The argumentation about the distributions and photo-realism is the same for transmissions. Therefore, it will not be discussed any further. The grid distribution with four rays was used for the blurry transmission (Figure 4.6 right). The screenshot on the left was rendered with Whitted-style ray tracing. Two things should be noticed here. First, Figure 4.6 shows nicely the effect of refraction. The bunny is magnified through the glass, which is exactly what happens in reality. Secondly, the screenshot on the right

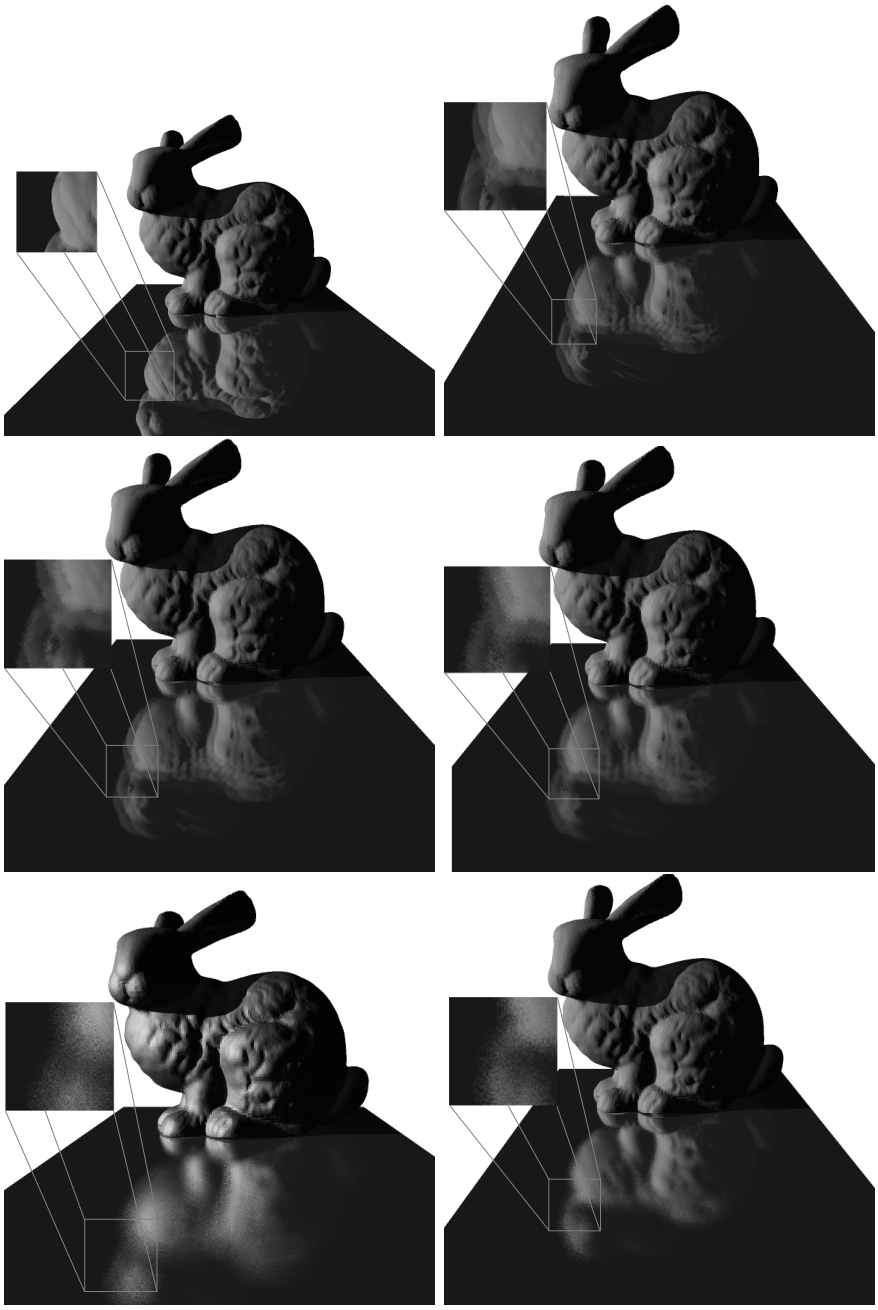

Figure 4.4: All images were rendered with 4x supersampling and four reflection rays, except the top left image, which is rendered with standard ray tracing. The grid distribution was used for the top right image. The images in the middle were rendered with the semi-jitter (left) and the jitter distribution (right). The Poisson distribution was used for the bottom left image. Finally, the image at the bottom on the right is an example for the random distribution.

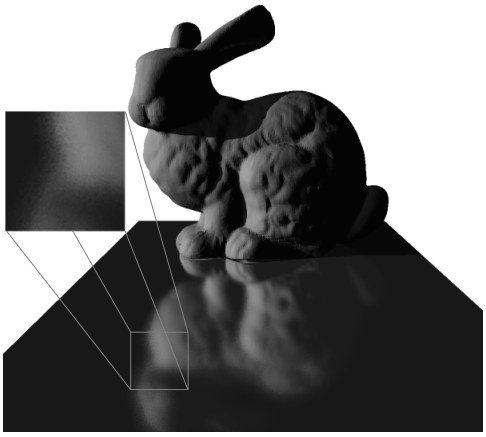

Figure 4.5: Thirty-two reflection rays should be traced to produce a smooth reflection. This image was rendered with thirty-two reflection rays, 4x supersampling, and the grid distribution.

demonstrates the blur effect, which is created by using four rays instead of one. Both images in Figure 4.6 look very photo-realistic.

Figure 4.6: The Stanford bunny is not blurred when using one transmission ray (left). Applying stochastic ray tracing blurs the bunny behind the glass (right).

Performance

Previously, it was shown that the performance of supersampling and depth of field decreases linearly with number of primary rays. This is rather good compared to the performance loss with stochastic reflections and transmissions. It is assumed that for each reflection, there are n rays instead of one. For each of these n rays, there will be n new rays again. This results in n^2 rays. For each of these n^2 rays there will be n new rays as well. Now, there are n^3 rays. And so on, until the desired recursion depth is reached. This small example, with recursion depth three, shows that instead of just three reflection rays with standard ray tracing, there are n^3 reflections rays with

stochastic ray tracing. Obviously, this is only the case if a scene has many reflecting objects. The same holds for transmissions. Figure 4.4 has only one reflective surface, therefore the ray count will not grow exponentially (Table 4.2). As a demonstration of the exponential growth, the scene from Figure 4.4 was used for Table 4.3, except that this time the bunny also has a reflective surface. The values in the tables are the number of rays that were traced. This example supports the argumentation above.

Mode \# Reflection Rays	1 Ray	4 Rays	32 Rays
Standard	$0.5 \cdot 10^6$	$4 \cdot 10^6$	$8 \cdot 10^6$
4x Supersampling	$1 \cdot 10^6$	$6 \cdot 10^6$	$32 \cdot 10^6$

Table 4.2: The scene from Figure 4.4, where the bunny is not reflective, shows linear growth for the ray count because only one reflecting surface is present.

Mode \# Reflection Rays	1 Ray	4 Rays	32 Rays
Standard	$1 \cdot 10^6$	$4 \cdot 10^6$	$118 \cdot 10^6$
4x Supersampling	$4 \cdot 10^6$	$15 \cdot 10^6$	$468 \cdot 10^6$

Table 4.3: The scene from Figure 4.4 with a reflective bunny shows exponential growth for the ray count.

Acceleration Metric

As described in Section 3.4, there are possibilities to accelerate the image generation process by minimizing the ray count. The metric, which was proposed in the previous chapter, can be used for reflections and transmissions. However, the analysis of the metric will be focused on reflection only, because argumentation is the same for transmissions.

It was mentioned in Section 3.4 that the metric might lead to wrong decisions. However, the resulting differences are only apparent if two images, one with and one without metric, are compared next to each other (Figure 4.7).

The difference between the images in Figure 4.7 is that the edges are a bit sharper. However, this is almost impossible to notice without a one on one comparison. So, the image quality remains roughly the same. Yet, the ray reduction rate is remarkable. When the left scene in Figure 4.8 was rendered with the metric, it required 720'854 reflection and 715'909 shadow rays. The same scene without metric required 810'880 reflection and 715'909 shadow rays. Additionally, 1'920'000 primary rays were used because of 4x supersampling. This means 3'356'763 rays were used for the scene with and

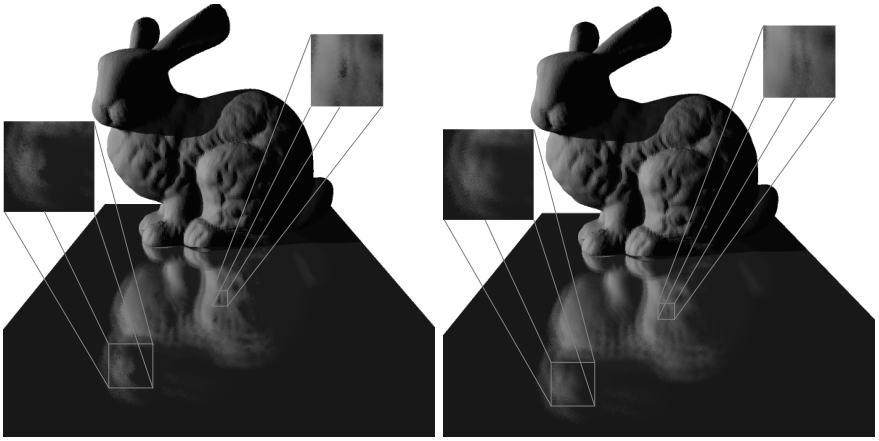

Figure 4.7: The left image was rendered with reflection metric and 4x supersampling. The right image was rendered without reflection metric but also 4x supersampling.

3'446'789 rays for the scene without metric. The achievement is a reduction of 2.5% (Table 4.4).

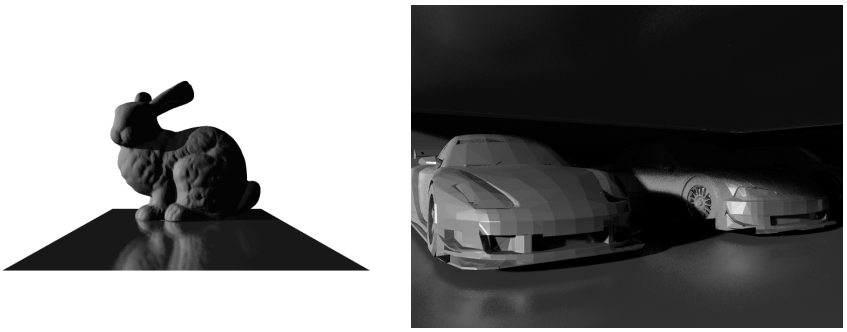

Figure 4.8: These two scenes were the basis for performance tests. Both images were rendered with 4x supersampling and metric. Four reflection and four shadow rays were calculated for the left image. For the image on the right, thirty-two shadow and reflection rays were used. These are the only two screenshots which are not cropped.

When rendering the scene, there are many primary rays which do not intersect with any object. In other words, the metric cannot act upon most of the 1'920'000 rays (approximately 80%). Omitting the primary rays which do not intersect, leaves approximately 400'000 rays. Now, the reduction rate is 5%, which is great for such a simple acceleration method. The ray reduction rate increases further in a more complex scene, because the metric can be applied on much more rays; hence, much more rays will be not be traced. As a demonstration of this effect, Figure 4.8 right was rendered

with two settings (Table 4.4). The first was one shadow ray, four reflection rays and no supersampling. The reduction percentage increases to 45%. The second configuration was thirty-two shadow and reflection rays and 4x supersampling. With this setting, the scene could not be rendered without metric. The scene has many reflections; therefore, it is assumed that there are 32^3 reflection rays. Additionally, there are 32 shadow rays for each of the reflection rays. This leads to a sum of $32^3 \cdot 32 \cdot 800 \cdot 600 \cdot 4 = 2 \cdot 10^{12}$ rays due to the 4x supersampling and the 800x600 resolution. It would take several weeks to render the scene with this setting, even on today's multicore CPUs. However, it was possible to render the scene with metric. $4.6 \cdot 10^8$ rays were calculated, which results in a theoretical reduction on 99%.

	+Metric	-Metric	Reduction
Figure 4.8 left 4r 4s 4x	3'356'763	3'446'789	2.5%
Figure 4.8 right 4r 1s 1x	$10 \cdot 10^6$	$18 \cdot 10^6$	45%
Figure 4.8 right 32r 32s 4x	$4.6 \cdot 10^8$	$2 \cdot 10^{12}$*	99%*

Table 4.4: This table shows how the reduction percentage increases with the ray count. Abbreviations: r stands for reflection rays, s for shadow rays, and x for supersampling. The values with an asterisk are theoretical values.

4.4 Penumbra

The last feature that was implemented is penumbra. Once again, the Stanford bunny was used to show the optical quality of penumbras. The four screenshots in Figure 4.9 show a scene with one light source. Once a point light source with no area (top left) and three times a light source with an expansion. The three images which show penumbra have the following settings. The top right image was rendered with nine shadow rays, no supersampling and the grid distribution. The two scenes at the bottom were rendered with nine shadow rays as well, but with the jitter distribution. The left one was rendered with 4x supersampling, and the right one without. Only one shadow ray but 4x supersampling was applied to the top left picture.

Image Quality

In Figure 4.9, it can be seen that even nine shadow rays do not produce a satisfying penumbra. Apparently, the penumbra is either pixelated or the gradient is not smooth enough. It is possible to enable supersampling to counteract the pixelation, but the result is disillusioning (bottom right). Even with supersampling and nine shadow rays, the image is still not smooth enough to give the impression of photo-realism. The ugly effect of the grid

distribution is here just as apparent as it is with reflections. In Figure 4.10, one can see that with thirty-two rays and 4x supersampling the quality is quite good. Here, the gradient is smooth and the pixelation is barely apparent. As with reflections, penumbras become smooth enough with thirty-two

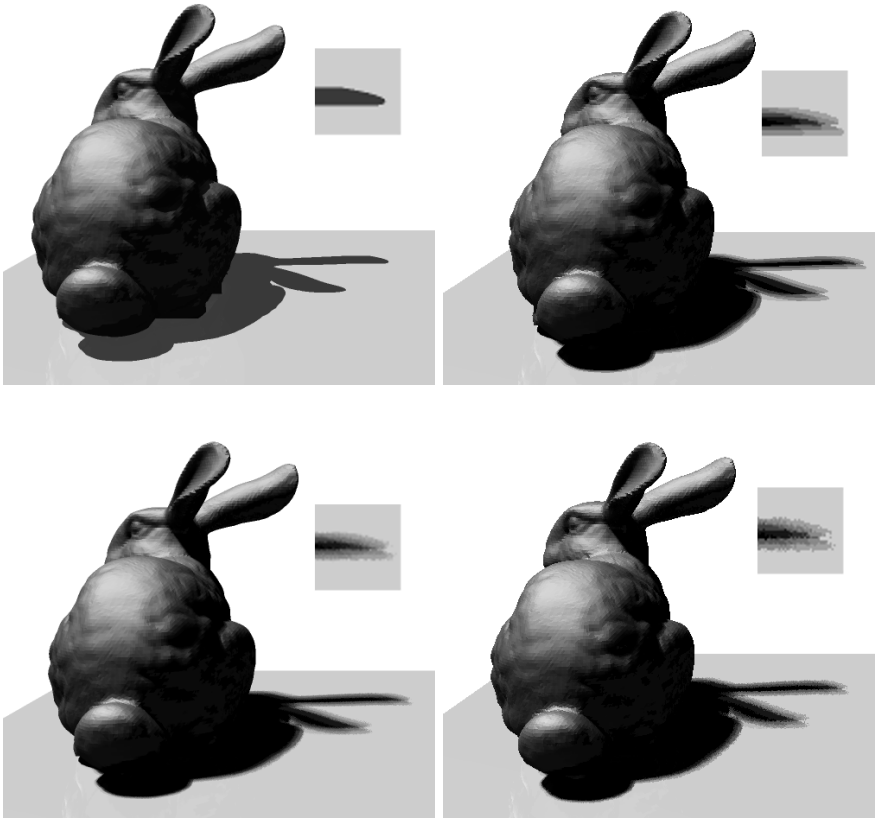

Figure 4.9: The top left image was rendered with one shadow ray and 4x supersampling. Nine shadow rays were used for the remaining three images. The top right image is an example of the grid distribution without supersampling. The two scenes at the bottom demonstrate the effect of 4x supersampling. Both use nine shadow rays and the jitter distribution. 4x supersampling was applied on the left but not on the right. The left sample is noticeably smoother.

and more shadow rays. Of course, this number depends on the size of the penumbra area. Figure 4.10 is more photo-realistic than the images in 4.9, only because the shadow looks very real. Again, if enough rays are traced the image quality increases strongly.

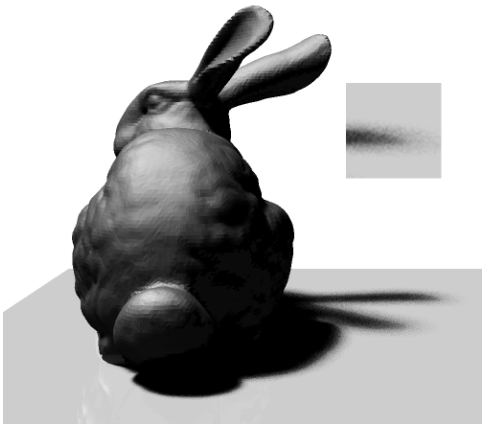

Figure 4.10: A minimum of thirty-two shadow rays should be used for a smooth and not pixelated penumbra. This screenshot was rendered with the random distribution and 4x supersampling.

Performance

As for the performance, the situation with penumbras is similar to supersampling and depth of field. If, for example, nine light rays are traced instead of one, then, for each intersection point, there are eight times more rays. The upper limit of rays is eight times the rays of standard ray tracing. Hence, the number of rays grows linearly, as well. Table 4.5, which is based on rendering the scene from Figure 4.9, shows this behaviour.

Mode \# Shadow Rays	1 Ray	9 Rays	32 Rays
Standard	$1 \cdot 10^6$	$3.5 \cdot 10^6$	$8.5 \cdot 10^6$
4x Supersampling	$4 \cdot 10^6$	$10 \cdot 10^6$	$35 \cdot 10^6$

Table 4.5: Ray count for penumbras grows linearly.

Acceleration Metric

The image characteristics with the metric for shadows are exactly the same as for reflections. Thus, the discussion about the image quality will not be repeated. The scenes from Figure 4.8 were used for determining the efficiency. The left scene rendered with metric required 810'880 reflection and 629'427 shadow rays. The same scene rendered without metric required 810'880 reflection and 715'909 shadow rays. Additionally, 1'920'000 primary rays were used because of 4x supersampling. This means, 3'360'307 rays were used for the scene with and 3'446'789 rays for the scene without metric.

Once again, a reduction of 2.5% was achieved. If both metrics are combined then the reduction increases to 5%. Furthermore, if the primary rays, which do not intersect with any object, are removed from the calculation (80%), the reduction is roughly 10%. The value increases even further if more complex scenes are rendered. Table 4.6 shows how the reduction percentage raises when more rays are traced.

Setting	+Metric	-Metric	Reduction
Figure 4.8 left 4r 4s 4x	3'360'307	3'446'789	2.5%
Figure 4.8 right 1r 8s 4x	$7 \cdot 10^6$	$12 \cdot 10^6$	42%
Figure 4.8 right 1r 16s 4x	$30 \cdot 10^6$	$90 \cdot 10^6$	66%

Table 4.6: This table shows how the reduction percentage increases with the ray count. Abbreviations: r stands for reflection rays, s for shadow rays, and x for supersampling.

4.5 Discussion

The expectations for all features were rather high, especially with regard to image quality. Our tests showed that, if sufficient rays are used, all features produce a smooth and realistic image; hence, as for image quality the expectations were met. However, the minimal numbers of rays for reflection, transmission, and penumbra were expected to be lower than they turned out to be. On the other hand, depth of field and supersampling are more efficient than estimated.

For supersampling, the expectations were that the aliasing effects would be reduced almost completely with 8x-16x supersampling. Therefore, the generation of an image without aliasing would take around eight to sixteen times longer. Astonishingly, it turned out that 4x supersampling is sufficient for most scenes. In fact, no scene was encountered, which required more than 4x supersampling for the aliasing effects to vanish. If a higher value than four is chosen, the image shows hardly any differences (Figure 4.1). Thus, supersampling surpassed our expectations.

Depth of field was the feature we were looking forward to the most. It was assumed that depth of field improves the photo-realism noticeably, which it actually does. However, the performance was expected to lower around eight to sixteen times. The reason is that the object out of focus is not smooth enough with only four primary rays. Fortunately, four rays are enough for regular apertures (Figure 4.2). Only if the aperture is chosen to be wide, sixteen and more primary rays are required to provide a smooth image (Figure 4.3). Still, depth of field is very effective with regard to the ratio of computational cost and image quality enhancement, because an image with depth of field looks very photo-realistic.

The expectation for both, transmission and reflection, were that the image quality is low if an insufficiently low number of rays are rendered. The image should show pixelation clearly. However, this effect should vanish if sufficient rays are calculated. Hence, the computation cost for a reasonable image quality is expected to be rather high. In fact, blurry transmissions and reflections can be produced with a decent quality. The disadvantage, however, is the enormous performance loss due to the high ray count, which is required to guarantee a high quality. For smooth reflections and transmissions a minimum of thirty-two rays are required (Figure 4.5). Without metric, this setting has such a high complexity, because of the exponential growth, that it cannot be computed in reasonable time. The metric is able to reduce the ray count to a feasible number with hardly any drawbacks in image quality. Still, the computation time is too high to meet the expectations; although, the image quality does.

At first, it was assumed that nine shadow rays should be sufficient to produce images with acceptably smooth penumbras. Just as before, if less than nine rays were traced, the image would become pixelated and, therefore, less photo-realistic. In general, the expectations were similar to stochastic reflections and transmissions. In contrast to transmissions and reflections, penumbras increase the photo-realism with a reasonable performance loss. The reason for this is because the number of shadow ray grows linearly. On the one hand, the minimal ray count for a smooth penumbra gradient is higher than expected, but this is still acceptable due to the linear growth. Furthermore, the metric can minimize the shadow ray count with hardly any quality loss. On the other hand, the penumbras look very realistic if many rays are used. In conclusion, the performance does not meet the expectations.

To sum up this chapter, each feature was analysed for image quality and performance. For penumbra, fuzzy reflection and refraction, the acceleration metrics were analysed as well. The final chapter will give a conclusion of the whole bachelor's project and an outlook for future work.

Chapter 5

Conclusion and Future Work

In this bachelor's thesis, it is verified that standard Whitted-style ray tracing is only a poor approximation of the rendering equation. Therefore, images generated by this technique lack of several important effects which occur in reality; for example penumbras, blurry reflections and transmissions, depth of field and motion blur. Stochastic ray tracing is able to simulate these effects. However, stochastic ray tracing does not solve the rendering equation either, but it is a better approximation than Whitted-style ray tracing.

The main idea of stochastic ray tracing is to trace additional rays. For instance, one reflection, transmission, and shadow ray is calculated for each intersection with Whitted-style ray tracing. In contrast, multiple rays are traced and averaged when applying stochastic ray tracing. This adds penumbra, blurry reflection and transmission to an image. Depth of field or supersampling can be achieved by shooting additional primary ray. An important aspect is how the new rays are distributed; in other words, where the origins and what the directions are. The grid, semi-jitter, jitter, Poisson, and random distribution were presented for this task and it was explained how to apply the ray distributions for each feature. Each ray distribution has its advantages and disadvantages, but when plenty of rays are calculated, all distributions show a realistic result.

Stochastic ray tracing brings great improvements in terms of photorealism. However, the images only look more real if enough rays are used. This leads to the downside of stochastic ray tracing, the enormous computational time. For each ray, an intersection test must be executed to determine if the ray in question intersects with any object in the scene. This intersection test takes some time. Hence, the more rays being used in a scene the longer it takes to generate the images. Stochastic ray tracing proposes to use multiple rays, where Whitted-style only uses one. Due to the additional rays, the computation time grows rapidly; especially when blurred reflections and transmissions are rendered.

Outlook

Further analysis should be conducted on how to reduce to computational time. As already said, there are situations where additional rays do not improve the image quality anymore. The proposed metrics are very simple. It is likely that there are far better and more sophisticated metrics. By using such a metric, the image quality might improve and the computation time might be reduced.

Additionally, alternative ray distributions could be implemented in Lumen and analysed. In this area, it might also be worth implementing the Bidirectional Reflectance Distribution Function (BRDF) in Lumen.

Bibliography

[1] J. Bikker. Raytracing: Theory and implementation. `http://www.devmaster.net/articles/raytracing_series/part1.php`, 10 2005.

[2] R. Bridson. Fast poisson disk sampling in arbitrary dimensions. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 22, New York, NY, USA, 2007. ACM.

[3] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, New York, NY, USA, 1984. ACM.

[4] M. A. Z. Dippé and E. H. Wold. Antialiasing through stochastic sampling. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 69–78, New York, NY, USA, 1985. ACM.

[5] D. Dunbar and G. Humphreys. A spatial data structure for fast poisson-disk sample generation. *ACM Trans. Graph.*, 25(3):503–508, 2006.

[6] P. Hanrahan. Computer graphics: Image synthesis techniques. `http://graphics.stanford.edu/courses/cs348b-01`, 1 2001.

[7] J. T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM.

[8] Ph. Robert. *Streaming Ray Tracing on a Non-Uniform Graphics Architecture*. PhD thesis, University of Berne, 2008.

[9] P. Shirley. *Realistic Ray Tracing*. A K Peters, Ltd., Natick, MA, USA, 2000.

[10] Ch. W. Ueberhuber. *Numerical Computation 2: Methods, Software, and Analysis*. Springer-Verlag, Berlin, 1997.

[11] T. Whitted. An improved illumination model for shaded display. *SIGGRAPH Comput. Graph.*, 13(2):14, 1979.