



Utrecht University

MASTER THESIS

Q-Learned Importance Sampling for
Physically Based Light Transport on the
GPU

Author:
Kevin van Mastrigt

Supervisors:
dr. ing. Jacco BIKKER
dr. Amir VAXMAN
Huub VAN SUMMEREN

ICA-3981061

*A thesis submitted in fulfillment of the requirements
for the degree of Master of Science*

in

Game and Media Technology
Department of Information and Computing Sciences

In collaboration with

3DIMERCE

February 23, 2018

Abstract

We present a GPU implementation of Dahm and Keller’s Q-learning based importance sampling technique [6]. The method requires a caching scheme to store the radiance distributions that are learned during path tracing. We tested the method on a photon map, a Poisson disk distribution and the Poisson disk distribution with a new addition called light occlusion. We found that the uniformly distributed points of the photon map produces the best results. The method itself was tested on four different scenes. We show that in an optimized GPU path tracer the method can have a positive influence on the performance, depending on the difficulty of the scene. In a comparison to a bidirectional path tracer we see that the method is able to outperform the bidirectional path tracer in a scene that is almost exclusively lit by indirect lighting. We conclude that it can be beneficial to implement the method in a GPU based path tracer.

Acknowledgements

I would like to thank dr. ing. Jacco Bikker for guiding me through the process of writing this thesis and for his wonderful feedback. Then I would like to thank Huub van Summeren and 3Dimerce for their excellent co-operation and for providing me with the equipment I needed to conduct the experiments. Furthermore I would like to thank Olaf Schalk and Mathijs Lardinoije for making these nine month of research a much more enjoyable experience. Finally I would like to thank my friends and family for listening to my complaints during the more difficult parts of the writing process. A special acknowledgement goes out to Miika Aittala, Samuli Laine, Jaakko Lehtinen, Crytek, Rui Teixeira, Eric Veach and Benedikt Bitterly for creating the 3D-scenes that were used for the experiments and to Ken Dahm and Alexander Keller for their research on which this thesis is based.

Contents

Abstract	i
Acknowledgements	ii
1 Introduction	1
1.1 Research Questions	2
1.2 Structure	2
2 Background	3
2.1 The Rendering Equation	3
2.2 Monte Carlo Integration	4
2.2.1 Importance Sampling	4
2.3 Path Tracing	5
2.3.1 Russian Roulette	6
2.3.2 Next Event Estimation	6
2.4 Expectation Maximization	7
2.5 Machine Learning	8
2.6 GPUs	8
3 Related Work	10
3.1 Caching	10
3.1.1 Irradiance Cache	10
3.1.2 Photon Map	11
3.1.3 Significance Cache	11
3.1.4 Poisson Disk Point Set	13
3.2 Advanced Importance Sampling Techniques	13
3.2.1 On-Line Learning of Parametric Mixture Models	13
3.2.2 Path guiding with SD-trees	14
3.2.3 Reinforcement Learning	15
Q-Learning	16
3.2.4 Q-learned Importance Sampling	17
4 Research Methodology	19
4.1 Testing Environment	19
4.1.1 The custom path tracer	19
4.1.2 Scenes	19
Ajar	20
Sponza	20
Bedroom	21
Bidir Room	21
4.2 Implementing Q-learning	22
4.3 Finding the optimal caching scheme	22

5	Implementation	23
5.1	Pointset	23
5.1.1	Photon map creation	23
5.1.2	Poisson Disk Set Creation	24
5.2	Q-learning	24
5.2.1	Structure of the Q-table	25
5.2.2	Nearest Neighbor Search	25
5.2.3	Updating	25
5.2.4	Sampling	26
5.2.5	Learning Rate	27
6	Results	28
6.1	Caching Point Distributions	28
6.1.1	Ajar	29
6.1.2	Sponza	31
6.1.3	Bedroom	32
6.1.4	Bidir Room	34
6.2	Q-learned Importance Sampling	36
6.2.1	Ajar	37
6.2.2	Sponza	40
6.2.3	Bedroom	43
6.2.4	Bidir Room	45
6.2.5	Influence of NEE and Russian roulette	48
6.3	Comparison to Bidirectional	50
6.4	Performance	52
6.4.1	Memory usage	53
6.4.2	Atomic operations	54
7	Discussion and Conclusions	55
7.1	Discussion	55
7.1.1	Q-learned Importance Sampling	55
7.1.2	Caching Scheme	56
7.2	Future Work	57
7.3	Conclusions	58
7.4	Contributions	59

1 Introduction

Physically based rendering is an increasingly popular model that can be used to render photo realistic images. It differs from rasterization techniques in that it attempts to simulate the way light transport works in the real world. The most common physically based rendering method is path tracing. Path tracing evaluates the lighting in a scene by taking many incomplete samples of the actual lighting and then averaging these samples together. This way phenomena such as caustics and color bleeding can be accurately rendered. A downside of path tracing, however, is that it is computationally intensive. It takes a lot of samples to produce a noise-free image and it can take quite some time to calculate a single sample for each pixel.

Many techniques exist to improve path tracing so that a higher quality image can be obtained in less time. Specifically, importance sampling allows us to focus on directions that are expected to yield more energy. A good approximation of the distribution of radiance at a certain point in the scene can significantly lower the variance of each sample. It is possible to create these distributions on the fly during path tracing. One way to do this is using reinforcement learning: during each sample we learn where most of the light in this scene is coming from. We can use this information to direct subsequent rays towards the light.

The radiance distribution information is stored sparsely in the scene. Several caching systems exist that can store such kind of information, which represent the scene using a set of points. Each of these points stores how the radiance is distributed over the hemisphere. When a ray intersects the scene we only need to find the point that corresponds with that part of the scene and sample the distribution stored in it to generate a new direction for the ray.

Path tracing is known to be an embarrassingly parallel algorithm, making it suitable for GPGPU computations. A GPU implementation can significantly lower the rendering time of the path tracer over a CPU implementation.

In 2017 Dahm and Keller showed that there are structural similarities between the rendering equation and the Q-learning algorithm [6]. Based on this information they developed an importance sampling technique that is based on Q-learning. They show that this technique is able to significantly increase the sample quality of a path tracer.

The original Q-learned importance sampling approach poses a number of challenges which are not addressed by the original work:

1. Q-learning requires a lot of synchronization which makes it less suited for a GPU. We investigate if the algorithm can be efficiently implemented in a GPU based path tracer.
2. We investigate which caching technique is best suited in combination with a GPU implementation of a Q-learning based importance sampling technique.

3. We know that Q-learning based importance sampling can lead to less variance, but it is poorly understood in what kind of situations this is the case.

The goal of this thesis is thus to gain more insight in the original algorithm and its characteristics. We implement the Q-learning based importance sampling technique in a highly optimized GPU based path tracer and test the method with several caching systems on a variety of scenes.

1.1 Research Questions

We have formulated the following research question:

Can an implementation of the Q-learning importance sampling technique improve the efficiency of GPU path tracing?

Which consists of three sub-questions:

What are the efficiency characteristics of the Q-learning method for different light transport scenarios?

,

Is it possible to implement the Q-learning importance sampling technique on the GPU in a way that improves the efficiency over a CPU implementation?

and

Can the Q-learning importance sampling technique be improved upon by using different caching systems?

1.2 Structure

This thesis is structured as follows: first, in chapter 2, we discuss background knowledge needed to understand the methods used. Then, in chapter 3 we discuss previous work relevant to the contributions of this thesis. In chapter 4 we show how we attempt to answer the aforementioned research questions. Chapter 5 goes into detail on the way the methods we used were implemented. The results of the conducted tests can be found in chapter 6. And finally, chapter 7 contains the discussion and conclusions.

2 Background

In this chapter we first provide background information needed for the methods that are detailed in the following chapters. We describe the rendering equation and how the path tracing algorithm evaluates this equation using Monte Carlo integration. We also explain the basic concept of importance sampling, as many methods of increasing the efficiency of path tracing are based on this technique, including the method implemented and tested in this thesis. A brief explanation of the basics of GPU programming will be included as the method will be implemented in a GPU based path tracer. Finally, because Q-learning is used to determine the probability distribution used for importance sampling, we provide a short overview of machine learning.

2.1 The Rendering Equation

The rendering equation [13] is a mathematical approximation of real-world light transport. It forms the basis for several physically based rendering techniques, including path tracing. The original equation models the incoming radiance at a point x from a point x' as the radiance directly arriving from light sources plus the radiance reflected by x' towards x . The irradiance at point x' is modeled as the directly incoming radiance at that point plus the radiance reflected from all possible points x'' in scene s that are visible from x' towards x . Because the incoming radiance at a point x depends on the incoming radiance at each visible point from x the equation is recursive. This is what is known as the *three point notation*.

Another way of representing the rendering equation is by modeling it as outgoing radiance in a given direction, ω . In this case the radiance emitted from x will be the direct emitted radiance from x plus the reflection of the incoming radiance from each direction ω_i over the hemisphere, Ω .

$$I_o(x, \omega) = I_\epsilon(x, \omega) + \int_{\Omega} \rho(\omega_i, x, \omega) I_o(x', -\omega_i) \cos\theta_i d\omega_i \quad (2.1)$$

Where:

- $I_o(x, \omega)$ is the radiance from point x into direction ω .
- x' is the first hitpoint after tracing a ray from x into direction ω_i .
- $I_\epsilon(x, \omega)$ is the direct lighting at point x into direction ω
- $\rho(\omega_i, x, \omega)$ is the reflectance distribution of radiance at point x coming from direction ω_i into direction ω .

Because this equation is both recursive and contains an integral the solution can not be evaluated analytically. Luckily complex integrals can be evaluated by replacing them with the *expected value* of a stochastic experiment.

2.2 Monte Carlo Integration

Monte Carlo Integration is a way of evaluating complex integrals. It works by sampling the integral N times at random positions, the estimated value of the integral is the mean of all acquired samples. The more samples we take, the closer we get to the actual value of the integral. The *law of large numbers* tells us that taking an infinite amount of samples will result in the actual value of the integral.

Say we have an integral over a function f :

$$I = \int_{\Omega} f(x) dx$$

We can approximate this integral by taking N random samples in the domain Ω . The Monte Carlo integrator of this integral would be:

$$I \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \quad \text{where } x_1 \dots x_N \in \Omega$$

Instead of taking random samples, we could also take samples from a distribution that more closely represents the shape of the integral. This is called *importance sampling (IS)*.

2.2.1 Importance Sampling

Importance sampling is a technique that is used extensively in statistics. IS can be used to reduce variance by drawing samples from a probability distribution that favors those parts of the integral that yield higher values. For example, if we want to estimate the distribution of light on a perfectly diffuse surface using Monte Carlo integration we would take samples according to a uniform distribution, even though more light is reflected close to the normal. In this case it would be better to sample from a *cosine weighted* distribution, as the chance of sampling directions that return more light becomes higher which results in a lower variance (see figure 2.1). Of course we do have to make up for using this different distribution by weighting the results. The distribution that we draw samples from is called the *probability density function (PDF)*. A PDF has to be equal to or larger than zero and has to integrate to one in order to keep the sampling process unbiased.

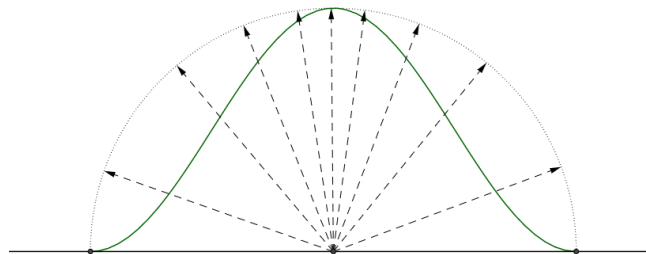


FIGURE 2.1: Rays distributed according to a cosine weighted PDF.

Suppose we want to evaluate an integral over a function $f(x)$ where p is a PDF on $\Omega \subseteq \mathbb{R}^d$:

$$I = \int_{\Omega} f(x)p(x)dx$$

We can introduce a new PDF, q , into the equation without modifying the outcome. As long as $p(x) > 0$ if $q(x) > 0$.

$$I = \int_{\Omega} \left(f(x) \frac{p(x)}{q(x)} \right) q(x)dx$$

Evaluating this integral using Monte Carlo integration would result in:

$$I \approx \frac{1}{N} \sum_{i=1}^N f(x_i) \frac{p(x_i)}{q(x_i)}$$

Where x is now distributed according to q . $p(x)/q(x)$ is called the *likelihood ratio*. The distribution q is the importance distribution and p is the nominal distribution. When using Monte Carlo integration to evaluate the rendering equation we can leave p out of the equation, as p is constant everywhere and integrates to 1:

$$I \approx \frac{1}{N} \sum_{i=1}^N \frac{f(x_i)}{q(x_i)}$$

Now that we have the ability to sample according to our own distribution we can try to find a distribution that gives us the lowest variance. In path tracing it is common to sample according to the BRDF of a surface, this results in relatively low variance most of the time, but will fail in scenes where the light is obstructed. Section 3.2 will provide more details on ways we can learn a probability distribution that is good in any situation.

2.3 Path Tracing

Path tracing is a physically based rendering technique developed by Kajiya [13] that solves the rendering equation using Monte Carlo integration.

Basic uni-directional path tracing works by shooting rays from the camera into the scene. Upon intersection with the scene two things can happen to a ray: if the ray hits a light source the path ends and the total contribution of that ray is taken into account, if the ray hits any other object it continues in a new direction depending on the material it hits. If the ray hits a perfect mirror, for example, it will bounce in a way such that the angle of incidence equals the angle of reflection. And if the ray hits a perfect diffuse surface it can bounce in any direction over the hemisphere projected onto the surface. This can be seen in Figure 2.2.

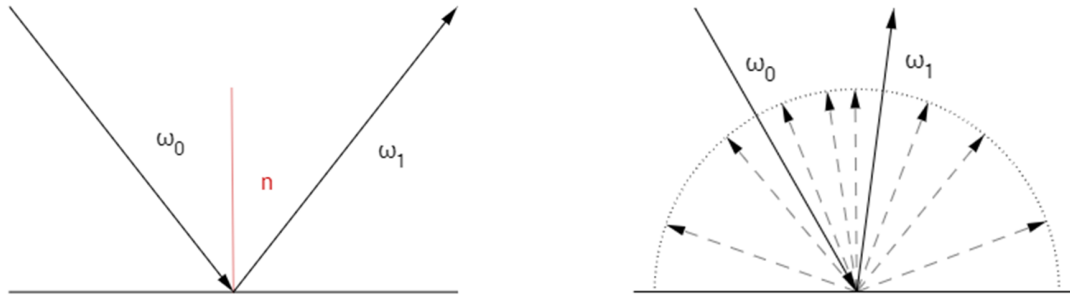


FIGURE 2.2: A ray, coming from direction ω_0 , bounces off a surface into direction ω_1 . Left: the ray hits a specular surface, it is reflected with the same angle to the normal as the incident direction. Right: the ray hits a diffuse surface, it could bounce in any direction on the hemisphere.

Each path keeps track of a throughput value which determines the maximum amount of energy that can flow through a path segment. When the path reaches a light source we can determine the light energy flowing through the path by multiplying the throughput with the intensity of the light source. Every bounce along the path the throughput changes based on the outgoing angle and the *Bidirectional Reflectance Distribution Function (BRDF)*. The BRDF defines how light is reflected at a certain surface, it can tell us the ratio of radiance reflected in a given direction.

In the classic path tracing algorithm every path will bounce around the scene until it hits a light source or leaves the scene. The final contribution of light of each path determines the color of the pixel the ray was initially shot through. To lower the variance and create a more accurate representation of the scene we trace many rays through each pixel and average their final color.

The downside of using Monte Carlo integration to solve the rendering equation is that it takes an infinite number of samples to reach the actual solution. Various research has been done both to speed up the core path tracing algorithm [14, 1] and to reduce the variance of each sample [11, 16]. One popular way to reduce variance in any Monte Carlo integration method is by using IS. Two well known IS techniques are *Russian roulette* and *next event estimation (NEE)*.

2.3.1 Russian Roulette

Russian roulette prunes long paths that will return little energy without biasing the result. This is done by having a probability, $P_{survive}$, at each intersection that decides if a ray is allowed to continue bouncing. A good probability for example is the maximum color value (scaled between 0 and 1) of the last vertex, but there are other possibilities as well. To keep the result unbiased we simply scale the energy of the surviving ray by $\frac{1}{P_{survive}}$.

2.3.2 Next Event Estimation

NEE is a method of separating the sampling of direct lighting from indirect lighting. Instead of just trying to find light by sampling the hemisphere we also sample the lights directly. Of course by sampling the hemisphere we will sometimes also sample a light source. To make sure that this does not bias the outcome we can either reject any

contribution of light by 'accidentally' sampling a light source or we can use multiple importance sampling (MIS) to combine the two methods of sampling.

2.4 Expectation Maximization

A probability distribution can be updated on the fly based on the samples we take during Monte Carlo integration. For example, if we would sample the incoming radiance at a certain point and store each sample along with the returned radiance and the direction from which the radiance came, we could estimate the distribution of radiance at that point which could then be used to generate new samples with less variance.

Expectation Maximization (EM) is a soft clustering method that can be used to estimate a probability distribution by fitting a *Gaussian Mixture Model (GMM)*. The method starts with an existing GMM, this can consist of Gaussians with a random mean μ and variance σ^2 or an estimate based on some other data such as the BRDF of the surface. EM consists of two steps: the expectation step and the maximization step.

Expectation: In the expectation step all data points are evaluated and for each point the probability, $P(g|x_i)$ of that point, x_i , belonging to each of the Gaussians, $g \in G$, is calculated:

$$P(x_i|g) = \frac{1}{\sqrt{2\pi\sigma_g^2}} \exp\left(-\frac{(x_i - \mu_g)^2}{2\sigma_g^2}\right)$$

$$P(g|x_i) = \frac{P(x_i|g)P(g)}{\sum_{g \in G} P(x_i|g)P(g)} \quad (2.2)$$

Where:

- x_i is the i th data point.
- μ_g is the mean of Gaussian g .
- σ_g^2 is the variance of Gaussian g
- $P(g)$ is the *prior* which is usually kept uniform, but could also be estimated.

Maximization: After calculating the probabilities the Gaussians are updated by recalculating μ and σ^2 :

$$\mu_g = \frac{\sum_{i=1}^N P(g|x_i)x_i}{\sum_{i=1}^N P(g|x_i)} \quad (2.3)$$

$$\sigma_g^2 = \frac{\sum_{i=1}^N P(g|x_i)(x_i - \mu_g)^2}{\sum_{i=1}^N P(g|x_i)} \quad (2.4)$$

These two steps are repeated until convergence, which means that the parameters will remain the same for any following iterations. EM does not necessarily reach a global optimum, it can easily get stuck in a local optimum. To counter this it can be beneficial to restart the method from several starting points and keep the best result.

2.5 Machine Learning

Machine Learning (ML) is a term used to describe a way computers can learn to do certain tasks that they were not explicitly programmed to do. To my best knowledge, the first paper on Machine Learning was written in 1957 by R.J. Solomonoff [22]. This work describes a machine that "*takes categories that have been useful in the past, and by means of small set of transformations, derives new categories that have reasonable likelihood of being useful in the future.*". In other words, the machine uses knowledge from past experiences to derive the solution for new (similar) tasks. Solomonoff also predicted that Machine Learning will ultimately result in computers being able to "*prove theorems, play good chess and answer questions in English*". And while we cannot use computers to prove theorems yet, Machine Learning has successfully been used to play chess [5] and produce complete sentences [2].

As Machine Learning is a very broad subject, this thesis will focus on using ML to solve a *Markov Decision Process (MDP)*. An MDP is a framework for *agents* to move around in. It can be used to model non-deterministic decision making. In a way, the path tracing method can be described as an MDP, which will be further discussed in section 3.2.4. We define a Markov Decision Process by:

- A set of states $s \in S$.
- A set of actions $a \in A$.
- A transition function $T(s, a, s')$.
- A reward function $R(s, a, s')$.
- A start state.
- Maybe one or more terminal states.

An agent starts in the *start state* and from there can take one or more *actions*. Taking an action will result in a new state, which is either a terminal state that ends the simulation or a state where the agent can take new actions. The transition function, $T(s, a, s')$ models the probability that taking action a in state s results in state s' . The reward function, $R(s, a, s')$, determines the reward the agent gets for taking action a in state s .

2.6 GPUs

A way of improving the speed of the core path tracing algorithm is by running the algorithm on the *Graphics Processing Unit (GPU)* instead of the *Central Processing Unit (CPU)*. Because GPUs have a highly parallel structure they are very efficient at computing algorithms where most of the work can be done simultaneously. Since rays can be traced almost completely independent of each other the path tracing algorithm benefits greatly from the parallel architecture of a GPU [1].

A GPU works with groups of threads (usually 32), called *warps*. These warps operate in a *Single Instruction Multiple Thread (SIMT)* manner: all threads in a warp are executing the same instruction at the same time, in lock step. These warps are themselves placed in blocks. Threads can share data with each other within a block. Where a CPU reduces latency by storing frequently used data in high speed caches,

a GPU hides latency by rapidly switching between active warps. If a warp needs to access data from memory the GPU will start to fetch the data and will immediately switch to a different warp so that it can continue processing instructions. When the data requested by the first warp arrives the GPU will be able to switch back and continue where it left off.

Ray tracing methods are usually implemented on the GPU by having each thread of handle a single ray. Each warp will simultaneously process 32 rays. However, as all threads within the warp have to be executing the same instruction at the same time problems arise when different rays in the warp intersect with different objects. For example, a ray hitting a specular surface is processed in another way than a ray hitting a diffuse surface and some rays will terminate earlier than others. This is solved using *masking*. A mask is nothing more than a boolean array, with each boolean representing a single thread in the warp. They can be used to make a calculation done by a thread have no effect on the ray it is processing. So when one ray hits a specular surface and the other hits a diffuse surface, calculations for both surfaces are done by all threads, but the calculations for the specular surface have no effect on the ray that hit the diffuse surface and vice versa.

While masking solves the problem of diverging rays it does make the program less efficient, as all calculations for all rays have to be run sequentially on all threads.

CUDA is a parallel computing platform that allows us run general purpose programming tasks on the GPU. Through *CUDA compute kernels* can be created that provide instructions to be executed by each thread. The GPU implementation detailed in this thesis is developed using *CUDA*.

3 Related Work

This chapter details previous research relevant to the method implemented and tested by this thesis. To combine Q-learning with path tracing we need a caching system that can efficiently hold the radiance distributions scattered over the scene. In section 3.1 we will explain the basics of several caching methods used in graphics applications. The photon map (section 3.1.2) and the Poisson disk set (section 3.1.4) are the caching methods that are actually implemented in this thesis. The second half of the chapter will describe several methods that guide the path tracing algorithm using estimated radiance distributions, ending with the method of Dahm and Keller which utilizes reinforcement learning to generate these distributions. This method is the basis of the research presented in this thesis.

3.1 Caching

There is an infinite number of points where a ray can collide with the scene. Most machine learning algorithms require a finite number of states. So we need to find a way to subdivide the scene into a set of points. These points work as a cache: they will store the learning values and they can be used as the states of the ML method. This section will detail various existing caching methods to store information of 3D scenes efficiently.

3.1.1 Irradiance Cache

Ward et al. developed a caching method to store indirect lighting information, the *irradiance cache (IC)* [25]. The irradiance cache works as follows: rays are cast into the scene, every time a ray hits an object we check if there is already a valid cache entry near the intersection point. If such an entry is found, the illuminance value from that entry is used. If there is no valid cache entry nearby, the illuminance at that point is computed and the value is stored in a new cache entry. This way there will be no redundant points, as only points that are reachable by the rays are added to the cache.

Illumination in an area varies less over distance in flat open areas than in areas where there is a larger gradient. So areas with large gradients, such as spheres or rough terrain, require more entries in the cache than areas with a low gradient. Therefore whether a nearby cache entry is "*valid*" depends on the gradient of the surface, the distance to the point and the normal of the point. Using these values an estimation can be made of the change in illuminance over a surface. Also the number of bounces of a ray must be taken into account so that final illuminances don't replace earlier values. This is done by maintaining a cache at each recursion level. To find nearby points a data structure is needed. Ward et al. used an octree for this.

The irradiance cache is an efficient way to store the illuminance values of a 3D scene and the scheme has the potential to be used to store machine learning values. However, because the entries in the irradiance cache are created during ray tracing it will

require a lot of synchronization when implemented in a multi-threaded environment, such as a GPU. Gautron et al. developed a more GPU friendly version of the irradiance cache [9]. This method removes the need to find the nearest points by splatting the radiance onto the screen plane, but they still do the adding of new cache points mainly on the CPU. While this produces good results when using the irradiance cache to calculate illumination it will not be efficient to store learning information, as we do not stop the path tracing algorithm once we found a cache entry.

Frolov et al. also introduce a GPU friendly irradiance cache implementation [7]. Their method is broken into two separate phases: the "*irradiance cache creation*" phase and the "*final rendering*" phase. The entire set of points is created in the first phase and is then used by the second phase to render the final image. However, just as with [9] new points are still inserted on the CPU.

3.1.2 Photon Map

Another caching method that is closely related to the irradiance cache is the *Photon Map*, developed by Jensen and Christensen [12]. The photon map is created before the actual rendering phase. In the creation phase rays are shot from the light sources into the scene. Where and how many rays are shot from each light source is determined by the size of the light source compared to the dimensions of the scene. Each ray resembles a single photon. These photons bounce through the scene taking into account the BRDF of each object they hit, at each hit point the normal, the intersection point and a small amount of the energy the ray carries is deposited into the photon map. This is done using a kd-tree, so that finding the nearest neighbors of each photon is efficient.

Not all nearest neighbors to the photon are evaluated. If the dot product of the normal at position x and the normal of the stored photon is negative the photon is rejected. Including these photons would negatively impact the approximation of irradiance at x as the normal of the photon deviates too much from the normal at x .

This method has the same problems as the irradiance cache when ran on a GPU. Purcell et al. [20] solved this problem by making a few modifications to the original algorithm:

The light tracing process of the first phase is almost the same as in the original method, the main difference is that each time a photon bounces off a surface the old photon is terminated and a new photon is created. This way it takes several passes for the first phase to finish. The photons are stored in a uniform grid that is created entirely on the GPU.

To find the nearest points in the grid the Parcell et al. developed a *k-nearest neighbors* method that works in a grid. This method still uses the expanding radius as with the original method, but each gridcell within the radius is checked for valid photons.

3.1.3 Significance Cache

The significance cache, developed by Bashford et al. [3], is another caching method used to determine global illumination. The basic idea is the same as with the Irradiance Cache in that at each point x we search the cache for the nearest neighbors and when nearby points are found the information from the cache is used, otherwise a new point is added to the cache. However, instead of using the stored illumination

values to approximate global illumination, the cache is used to direct the path tracing algorithm, i.e. at every point x a new direction is sampled according to the stored values. When no nearby points are found the BRDF is used for sampling as usual and the point is added to the cache and updated when the path ends.

At each point the incoming radiance is stored along with the incoming direction and the position of the point. The radiance itself is stored using cosine lobes. Cosine lobes can be easily modified and stored using only minimal amounts of information. On top of that they can be analytically sampled. This cosine lobe model is closely related to the LaFortune BRDF [15]. Although the method can be used to model the radiance distributions when more generic BRDFs are used it might not be as effective.

The significance cache influences the distribution from which the importance is sampled: instead of only sampling according to the BRDF of the surface, importance is sampled proportional to the product of the BRDF, incoming luminance and the cosine falloff factor [3]. To do this first a distribution needs to be build that closely resembles the actual distribution of radiance. We find the n points closest to the shading point x , each of these points contains an oriented cosine lobe. A weight is added to each lobe to keep the method unbiased. This leads to the following distribution:

$$\tilde{D} = \left(\sum_{i=1}^{N_{cache}} w_i D_i(x) \right) + w_{N_{cache}+1} D_{BRDF}(x) \quad (3.1)$$

Where:

- w_i is the weight of the i th cosine lobe.
- N_{cache} is the set of the N points closest to x .
- $D_i(x)$ is the i th cosine lobe.
- $D_{BRDF}(x)$ is the BRDF at x .

These cache points are stored in an acceleration structure so that the points nearest to a position x are efficiently found. Only points with roughly the same orientation as x that are within a certain range are accepted.

All of the points that are used in the distribution need to have a weight. To keep the algorithm mathematically correct all the weights need to sum up to one. The weights are calculated as the product of the BRDF, the incoming radiance and the cosine term and are then normalized the sum of all weights. The weighted distribution can easily be sampled by first picking a lobe to be sampled an then drawing a sample from the lobe.

Once the incoming radiance is returned, cache points are updated based on information returned from shooting the ray. Three components of each cosine lobe have to be updated, that is the lobe direction the lobe exponent and the radiance contained within the lobe.

This method has two drawbacks. The first is the increased rendering time, which is, according to the authors, roughly 15%. However, the overall performance benefits from the improved efficiency [3]. The second is that it is, just as the Irradiance Cache and the Photon Map, hard to efficiently implement on the GPU because of all the required synchronization.

3.1.4 Poisson Disk Point Set

Bikker and Reijerse presented a caching scheme that uses a point set which is generated using a Poisson Disk process [4]. From now on this scheme will be referred to as the *Poisson Disk Point Set*. The Poisson Disk distribution prevents aliasing artifacts and provide a good representation of the scene geometry.

To acquire the Poisson Disk distribution rays are shot from the light sources in the same way as with the Photon Map. These rays bounce around the scene and each intersection point of a ray with the scene is saved and will be used as a '*dart source*'. Darts will be shot into the scene from each dart source, a dart is basically a ray that does not bounce. At the intersection point of each dart with the scene the *ambient occlusion* [28] is calculated. The ambient occlusion is used to determine a search radius. If a dart already exists within this search radius, the new dart is discarded. This results in a point set with a density that is greater in colluded areas, and very sparse on large planar surfaces [4].

The creation of this cache is too slow to be done in real time, it has to be created beforehand. However, as the cache is independent of the view direction it only has to be created once per scene, as long as the scene geometry does not change. Although the short paper does not compare the method to other methods, the results look promising. Because the machine learning methods explained in section 3.2 also benefit from more cache entries in colluded areas, it seems that this caching method will be very suitable to use for caching the learned distributions.

3.2 Advanced Importance Sampling Techniques

The caches from section 3.1 can all be used to store importance distributions. These importance distributions can then be used to guide the Monte Carlo path tracing algorithm, which will reduce the variance of the method.

This section will detail several methods for generating importance distributions. At the end of the section we will explain the basics of reinforcement learning and show how Dahm and Keller apply this to create an importance distribution.

3.2.1 On-Line Learning of Parametric Mixture Models

Vorba et al. introduced a method that models indirect lighting distributions using a *Gaussian Mixture Model (GMM)* that is created using a modified *Expectation-Maximization (EM)* algorithm [24]. Their method is split into two separated phases: a *training phase* and a *rendering phase*.

The training phase consists of several passes. In each pass rays are cast from the camera (*importons*) and from the light sources (*photons*). In the first training pass the importons are cast into the scene without guidance after which the initial distributions are created. These radiance distributions are then used to guide each following pass. At the end of each pass the distributions are updated with the newly found results.

Just as with the irradiance cache method distributions are stored in a cache and new entries are added to the cache while learning. If there is no cache entry close to shading point x a new entry is added to the cache. After each training pass all distributions in the cache are updated. This is done by finding the N particles closest to x and apply *stepwise EM* to cluster them. Each cluster represents a Gaussian and

by combining these Gaussians a GMM is created. If there is no existing distribution at x then the *off-line stepwise EM* algorithm is used, which runs until convergence on the first batch of N particles. If there is an existing distribution then the *on-line stepwise EM* algorithm is used which starts with the previously found distribution and processes each new particle only once.

In the rendering phase the cached distributions can be used to guide any path tracing method, including bidirectional path tracing. If no cached distribution is found at a point x then a new distribution is trained from the latest batch of particles.

A downside of this method however, is that they only sample proportional to the incoming radiance, while a better result could be obtained by sampling according to the entire integrand, which also includes the BRDF. Herholz et al. attempt to solve this by also modeling each BRDF as a GMM and then calculating the product of each BRDF and the learned radiance GMM [10]. Their method builds on the method of Vorba et al. [24] by adding an extra preprocessing step and slightly modifying the sampling method.

The preprocessing step is needed to create a Gaussian Mixture Model of each BRDF in the scene. These GMMs are then stored in a BRDF cache for quick access.

The training phase works in the same way as with [24], photons and importons are cast into the scene and radiance distributions are created using the online EM algorithm and then cached.

In the rendering phase, at each location x the BRDF is queried from the BRDF cache and the illumination is acquired from the illumination cache. The product p_{\otimes} of these two values is calculated. As both these values are GMMs, their product is also a GMM. The outgoing direction of the new ray is created by sampling according to p_{\otimes} . To improve performance the GMM reduction algorithm by Runnals [21] is applied to each BRDF GMM after the preprocessing phase and to each illumination GMM after the training phase.

The method of Herholz et al. [10] shows a slight improvement over the original method by Vorba et al. [24] and a significant improvement over standard path tracing and bidirectional path tracing. However it still requires a separate training phase, which makes the method less than ideal for a real time path tracer.

3.2.2 Path guiding with SD-trees

Instead of Gaussian Mixture Models Müller et al. use a five-dimensional adaptive *spatio-directional tree (SD-tree)* for guiding paths [19]. The radiance in the scene is represented by an SD-tree, which consists of a three-dimensional binary tree that partitions the scene space and a two-dimensional quadtree that partitions the directional space of each position.

Each iteration paths are traced until they hit a light source. The vertices each path hits along the way are saved. When the path ends they iterate over every vertex of the path. For every vertex the binary tree is traversed to find the leaf node corresponding to the position of the vertex. From the leaf node of the binary tree they enter a quadtree that models all possible directions. The quadtree is traversed by only expanding nodes that contain the direction of the path at that vertex. The acquired radiance at that vertex is stored in every expanded node of the tree. The radiance stored in the tree is then used to guide the path tracing in the next iteration.

After each iteration the structure of the SD-tree is adapted so that the next iteration can provide a better estimation of the radiance. The binary tree is updated by splitting nodes that store more than $c \cdot \sqrt{2^k}$ path vertices, where 2^k is proportional to the number of paths traced in iteration k , and c is derived from the resolution of the quadtrees [19]. This ensures that regions with a high contribution of radiance get refined more than regions with a lower contribution.

The directional quadtree is updated by descending through the tree and subdividing each node where the fraction of collected flux flowing through the node is larger than $\rho = 0.01$, where the fraction of collected flux is calculated by dividing the flux flowing through the node with the total flux flowing through the tree [19]. If a node is already divided and it does not meet the subdivision requirements than it's children are pruned. At every new subdivision a quarter of the flux is assigned to its children and the subdivision criterion is applied to each child node. Just as with the binary tree, this ensures that areas of the hemisphere that provide more radiance are subdivided more often than areas that provide a lower contribution.

This SD-tree that updates every iteration allows for an iterative learning scheme in the same way as with the method of Vorba et al. [24]. In the first iteration paths are guided according to the BRDF, just as with regular path tracing. At the end of the iteration the SD-tree is filled with the sampled radiance. Each next iteration then guides path by combining samples from the BRDF and the SD-tree of the previous iteration using multiple importance sampling. To sample a direction from the directional part of the SD-tree the hierarchical sample warping method of McCool and Harwood [18] is used.

The SD-tree method is compared to various other methods, including the method of Vorba et al., and shows promising results. The method can generate images with less variance with a lower number of samples in a shorter amount of time compared to the GMM method. Another advantage over Vorba's method is that there is no separate training phase, which makes the method more suitable for real-time applications. However, the method might be hard to implement on the GPU because adapting the SD-tree after each iteration is hard to parallelize and may need to be done on the CPU.

3.2.3 Reinforcement Learning

We could also use machine learning to determine the radiance at a certain point. However when path tracing we don't really know the transition function, T , or the reward function, R , beforehand; these are determined during the tracing of rays. As a result that in this case we can not solve the MDP in a straightforward way. A common way to solve an MDP without knowing T and R is using *Reinforcement Learning (RL)*. Reinforcement Learning is a Machine Learning technique where a computer, or *agent*, is trained using a reward system. The agent gets a positive reward if it does something 'good', and a negative reward if it does something 'bad'. All learning is based on the outcome of the actions the agent takes, as the agent does (initially) not know which states are good or what each action it can take does.

A common RL approach is to learn a model of the environment based on the observations the agent makes when exploring its surroundings. This is called a *model based* approach. The agent tries out a large number of actions and uses the results to create an estimate of T and R . These estimates can then be used by any planning

algorithm to find an optimal policy. The downside of this is that it takes a long time to learn the model and learning has to be done off-line, meaning that intermediate data can not be used by the agent. It also uses a lot of memory to store all data of the model.

Another approach is to skip learning the model all together and just learn the optimal policy. We can even do this in a way where intermediate results of the learning algorithm can already be used by an agent. The agent will try out various actions in each state and see what reward it gets for taking that action. The next time the agent encounters that same state it will take into account the rewards that it got the last time it was there, but it will still have a (smaller) chance to take either an unexplored action or to take an action that previously resulted in a low or negative reward. The reason for this is that it might be possible to get a higher reward in the future by first taking an action that will give a low reward. This is called *model free* RL.

Because model based approaches are slow and can only be used once the complete model is known. This thesis will focus only on a model free approach. With a model free approach we can benefit from the learned data after each iteration which makes it more suitable for real-time path tracing.

Q-Learning

A popular model free RL algorithm is *Q-Learning*. Q-learning is an algorithm introduced by C.J.C.H. Watkins in 1989 [27]. It can learn an optimal policy for an MDP and it is very efficient in learning models where a higher reward can be found after first taking steps that result in low or negative rewards (known as delayed rewards). Watkins and Dayan later proved that Q-Learning always converges to the optimal policy [26].

In Q-Learning an agent moves through the world as usual, taking an action in each state and learning from its results. The goal of the agent is to *maximize its expected reward*. Each state, action combination will have a *Q-value*, i.e. the expected reward after taking that action in the corresponding state. After taking each action the Q-values are updated as shown in equation (3.2).

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \left(R(s, a) + \gamma \cdot \max_{a' \in A} Q(s', a') \right) \quad (3.2)$$

Where:

- a is a possible action taken from a finite set of actions, A , at state s .
- $Q(s, a)$ is the value of Q for taking action a at state s .
- $R(s, a)$ is the direct reward for taking action a at state s
- s' is the resulting state after taking action a .
- α is the learning rate, with $\alpha \in [0, 1]$
- γ is the discount factor, with $\gamma \in [0, 1]$

There are two user defined variables in this formula: the learning rate, α , and the discount factor, γ . The learning rate determines how quickly new experiences are taken into account. When α is zero the agent will not learn anything, when α is one it will dismiss all past knowledge every iteration. The discount factor determines how much delayed rewards are taken into account. When γ is 0 the agent will only consider immediate rewards, while if γ approaches 1 the agent will favor more delayed rewards.

Choosing the right values for α and γ can be very important for the efficiency of the agent.

The Q-learning algorithm is shown in algorithm 1.

Algorithm 1 Q-Learning algorithm

```

s = start state
for all iterations  $n \in N$  do
   $a = \arg \max_{a' \in A} Q(s, a')$ 
  UpdateQValue( $s, a$ )
   $s = \text{TakeAction}(s, a)$ 
end for

```

3.2.4 Q-learned Importance Sampling

Dahm and Keller developed a method for machine learned importance sampling using Q-learning without the need for an off-line training phase [6]. To apply Q-learning to the path tracing algorithm some slight modifications to the formulation as specified in eq 3.2 are needed. Instead of a finite space of actions A we now have a continuous action space and instead of selecting the action with the highest Q-value we take the average over all actions in state s :

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot \left(R(s, a) + \gamma \cdot \int_A \pi(s', a') Q(s', a') \right) \quad (3.3)$$

Where:

- $Q(s, a)$ is the value of Q for taking action a at state s .
- $R(s, a)$ is the direct reward for taking action a at state s
- s' is the resulting state after taking action a .
- $\pi(s', a')$ is a function that weights the value $Q(s', a')$
- α is the learning rate, with $\alpha \in [0, 1]$
- γ is the discount factor, with $\gamma \in [0, 1]$

In their paper Dahm and Keller show that the mathematic formulation of Q-learning (eq 3.3) shows several structural similarities with the rendering equation (eq 2.1). The location x from eq 2.1 can be seen as the state s from eq 3.3, tracing a ray from location x into direction ω can be seen as taking action a , the direct lighting $I_\epsilon(x', -\omega_i)$ from hitpoint x' towards x can be seen as the reward function $R(s, a)$ and the BRDF $\rho(\omega_i, x, \omega)$ can be seen as the weighting function $\pi(s', a')$. Combining the two equations, we get:

$$Q(x, \omega) = (1 - \alpha) \cdot Q(x, \omega) + \alpha \cdot \left(I_\epsilon(x', -\omega) + \int_s \rho(\omega_i, x', -\omega) Q(x', \omega_i) \cos \theta_i d\omega_i \right) \quad (3.4)$$

Where:

$Q(x, \omega)$	is the value of Q for tracing a ray in direction ω from position x .
x'	is the first hitpoint of tracing a ray from x into direction ω .
$I_\epsilon(x', -\omega)$	is the direct lighting from point x' into direction $-\omega$.
$\rho(\omega_i, x', -\omega)$	is the reflectance distribution of radiance at point x' coming from direction ω_i into direction ω .
α	is the learning rate, with $\alpha \in [0, 1]$
γ	is the discount factor, with $\gamma \in [0, 1]$

This Q-function is then used and updated during the path tracing algorithm. At every intersection point x of a ray with the scene, we choose a direction ω by sampling according to the distribution of Q-values at x . Every time a ray is traced from a position x into direction ω we update the Q-value for that combination of x and ω . Since Q-learning requires us to use a finite set of actions we not only have to subdivide the scene into a set of states but we also have to stratify the hemisphere from which we sample the directions. At each intersection point x one $Q(x, i)$ value is stored per stratum i . The level of stratification of the hemisphere has some impact on the performance of the algorithm: choosing a high number of strata results in slower learning (because there is a lower chance of updating each stratum), but a higher precision when sampling a direction and choosing a low number of strata results in faster learning, but a lower precision when sampling a direction (because the stratum covers a larger part of the hemisphere).

Dahm and Keller show that this method can lead to a 20% efficiency improvement over normal path tracing as the number of light paths without contribution to the end result decreases the more the algorithm learns. They claim that using Q-learning allows the algorithm to learn much faster than previous methods such as the *EM* method explained in section 3.2.1.

While the paper shows some statistics of the method it would be beneficial to acquire more knowledge of exactly how the system behaves in several key scenarios. Information such as how the method performs over time or how the quality of the image develops compared to standard path tracing are left out.

Dahm and Keller implemented their method in a basic CPU path tracer without basic optimizations such as NEE and Russian roulette. A GPU path tracer without this method can reach an efficiency improvement of more than 20% over a CPU path tracer. To measure the real-world performance of the method it needs to be implemented in an optimized GPU based path tracer.

4 Research Methodology

This chapter will explain how the research questions listed in chapter 1 will be answered. We detail the testing environment used, how the tests are conducted and how we implemented Dahm and Kellers method. The method will be implemented on the GPU. Due to the parallel nature of the GPU some changes will need to be made to the method. The main goal of this thesis will thus be to test whether a GPU implementation of the method can be just as effective as the original CPU implementation. To determine this the q-learning method will be tested with several caching schemes on various scenes.

4.1 Testing Environment

The Q-learning method will be implemented and tested on different scenes. Performance will be measured in several ways:

- The convergence speed of each method is measured by calculating the *mean squared error (MSE)* between each generated sample and a reference image (a fully converged path traced image of the same scene). Each test will run until the MSE reaches a certain threshold. The time until that value is reached is the convergence speed.
- To determine the efficiency of the learning we will measure the percentage of paths that return energy during each sample. If the percentage of paths that return energy increases the algorithm is learning where the light comes from.
- Another way of determining the efficiency of the learning is to measure the average number of bounces a path makes before it is terminated. Less bounces means that the algorithm is directing the paths towards the light.
- A third way to determine the efficiency of learning is by measuring the average energy per path segment. If this number is high it means that more paths are reaching the light and that they do this with less steps than without learning.

All tests will be conducted in a custom GPU based path tracer that is highly optimized.

4.1.1 The custom path tracer

The custom path tracer used incorporates most effective optimization techniques, including: a bonzai BVH [8], NEE, Russian roulette, wavefront [17] and persistent threads [1]. A GPU implementation of the Q-learning method will be tested with this path tracer to see if it can give an increase in performance in real-world scenarios.

4.1.2 Scenes

The method will be tested in four different scenes. Each of these scenes has its own characteristics that affect different aspects of the learning method.

Ajar

This is a scene by Miika Aittala, Samuli Laine and Jaakko Lehtinen which is also used in [6]. All light in the scene is coming from a single large light source that is behind the door. This light source is not directly visible from the camera's position, meaning that there is no direct lighting in the visible parts of the scene. This makes the scene ideal for Q-learning as next event estimation becomes less useful.



FIGURE 4.1: The reference image of the Ajar scene.

Sponza

Sponza is a well-known scene within the graphics community created by Crytek. The Sponza scene shows the courtyard of a palace lit only by the sun and the sky. The same viewpoint is chosen as in [6].



FIGURE 4.2: Reference image of the Sponza scene.

Bedroom

The Bedroom scene (by Rui Teixeira) is another hard to render scene. The scene consists of a simple bedroom where the light is coming from two light sources just outside the window. In front of the window are two thin curtains that allow light to pass through. The glass window and the curtains make the light hard to reach. The Q-learning method will have to learn to send the light towards the curtains even though it will only get a reward when the ray passes through the curtains and the window.



FIGURE 4.3: Reference image of the Bedroom scene.

Bidir Room

The Bidir Room (originally by Veach, recreation by Benedikt Bitterli) is a scene with a light shining onto a glass egg. This creates a caustic, which is quite hard to render for a unidirectional path tracer. Veach uses this scene to test a bidirectional path tracer, as bidirectional path tracers excel in scenes with caustics [23]. It will be interesting to see if the Q-learning method is able to guide paths through the glass egg to create the caustic.



FIGURE 4.4: Reference image of the Bidir Room scene.

4.2 Implementing Q-learning

Implementing the Q-learning importance sampling technique on the GPU is more involved than a CPU implementation. The need for synchronization during the actual Q-learning is a problem on the GPU: every time a ray hits a surface a cache entry has to be updated, since the updating of the rays happens in parallel the chance that two or more rays will try to update the same value is quite high. To make sure that this doesn't lead to corrupted values we will use CUDA's *atomic operations* wherever synchronization is required. While these operations are quite fast on modern GPUs they can still slow down the efficiency of the method. The performance tests in section 6.4.2 will show the significance of this slowdown.

4.3 Finding the optimal caching scheme

The Q-learning algorithm requires a cache to store its learning values. To find which caching scheme works best we will implement and test the Photon Map and the Poisson Disk Point Set and compare them. The Poisson disk set will also be extended with an extra rejection criterion based on the reachable light from a given point. These caches will be created before the actual rendering. The performance of each cache will be determined by applying the measurements as discussed in section 4.1.

5 Implementation

This chapter details how the Q-learning based importance sampling method is integrated in the GPU based path tracer. The first part of this chapter describes the implementation of the different caching schemes and the second part describes the actual Q-learning method itself. Pseudo code will be provided for the key parts of the method.

5.1 Pointset

5.1.1 Photon map creation

A photon map is created partially on the GPU and partially on the CPU. The implementation consists of several kernels that each have their own specific task.

The first kernel creates rays from the light sources and/or the skybox. Rays from the light sources are generated by randomly choosing a light source based on the radiance emitted by the light and then choosing a random point on that light source. This point is used as the origin of the ray and a random direction is sampled using a cosine weighted distribution centered around the normal of the light source. Rays from the skybox are generated by choosing a random point within the bounds of the scene and then creating a ray from a random point in the sky (out of the bounds of the scene) towards the point in the scene.

The second kernel traces the rays by traversing the BVH of the currently loaded scene. The distance a ray has traveled and a reference to the material at the intersection point of the ray are saved so that they can be used by the next kernel. This is the same kernel that is used to trace the rays during the path tracing algorithm.

The third kernel saves the intersection point of a ray as a 'dart source' based on the type of material at the intersection and the number of bounces of the ray. Dart sources are not created on light sources as a ray hitting the light source will always be terminated and not on passable surfaces as the direction of a ray hitting such a surface is predetermined. The number of bounces of the ray also influences the possibility of saving the intersection point; the more bounces the higher the chance of saving the intersection point. This is to prevent having most dart sources near the lights.

If the ray has less than a predetermined number of bounces a new direction for the ray is generated. These directions are randomly generated as if the ray hit a purely diffuse surface, unless the surface is passable at which point it will have a direction passing through the surface. This allows the point set to work in sections of the scene which are enclosed by passable surfaces (e.g. a room with closed windows).

These three kernels are looped until a given number of dart sources is reached. After the loop is done the next three kernels are invoked. They create the point set by shooting 'darts' (rays that don't bounce) from the dart sources.

The fourth kernel will generate darts from each generated dart source. Each dart is generated with a random direction away from the normal of the dart source.

The fifth kernel is the same as the second kernel, it is used to find the intersection point of each dart.

The sixth and final kernel saves the intersection point of each dart. The intersection points are then send to the CPU where they will be added to the final point set. Each point coming from the GPU will be checked against the nearby points that are already in the point set, if the new point is closer than a given minimum distance to any of the points in the set it is discarded.

These final three kernels are then looped until the number of new points that are accepted for the point set is less than one percent of the total points in the set. This way the loop keeps running until the scene is filled, and we don't have to determine the maximum amount of points for each scene.

5.1.2 Poisson Disk Set Creation

The Poisson Disk Set is implemented in the same way as the photon map. The only differences are the acceptance criteria for new points when they are added to the point set. The Poisson disk set has a maximum and a minimum rejection distance. If the new point is further away than the maximum rejection distance to each point in the set it is automatically accepted. If it is closer then the minimum rejection distance to each point in the set it is automatically rejected. However, if the point is between the maximum and minimum rejection distance to one or more of the points in the set we calculate the ambient occlusion at the location of the new point. The ambient occlusion is a number between zero and one, one meaning that the point is completely occluded by nearby geometry and zero meaning that it is not occluded. This value is then used to determine a new rejection distance for that point, resulting in a larger rejection distance when the ambient occlusion is lower.

The ambient occlusion is calculated by casting a number of rays from the point in random directions. If a ray hits any scene geometry closer than a given distance the ambient occlusion increases. The ambient occlusion is the percentage of rays that hit an object closer than the given distance.

An optional rejection criterion is a check based on the light visible from the point to be added to the set. This criterion will be referred to as the 'light occlusion'. It works in a way similar to the ambient occlusion. A number of rays are shot from the point towards random points on light sources in the scene. Each ray that hits the light source (so no objects between the ray and the light) decreases the light occlusion value. The light occlusion is the percentage of rays that can't reach the light. This criterion is only used in combination with the ambient occlusion criterion, the average of the two values are used for the final rejection distance.

5.2 Q-learning

The Q-learning is mostly implemented as described by the pseudo-code in [6] to keep comparisons fair. There are, however, some differences needed for the method to work on the GPU. The custom path tracer we use is a "wavefront path tracer" [17]. This means that the generation of rays, the traversal of rays, the traversal of shadow rays and the sampling are all done in different kernels. On top of that each kernel uses

persistent threads [1], which means that each kernel is initialized with a predefined number of threads. These threads will process tasks from a large pool until all tasks are done. Since all updates to the Q-table are done in parallel we need to have a separate kernel after the update kernel that calculates the maximum and the *cumulative distribution function (CDF)* of each distribution.

The complete path tracing loop with Q-learning becomes as follows:

1. Generate: Generate new rays from the camera.
2. Extend: Trace rays and calculate their intersection points.
3. Update: Update the Q-table
4. Calculate CDF: Calculate the maximum Q-value and CDF of each distribution.
5. Sample: Update the radiance of each pixel, generate new direction by sampling the Q-table for each ray that isn't terminated and generate shadow rays.
6. Connect: Trace the shadow rays.
7. Step 2 to 5 are repeated until all rays are terminated.

5.2.1 Structure of the Q-table

At each point in the point set we subdivide the hemisphere into n strata of equal area. For each stratum we store a q-value (in QTable). This value is updated by the *Update* kernel as explained in section 5.2.3. At each point we also store the CDF (in QCdf) and the maximum of all q-values (in QMax). On top of that we keep track of the number of times each stratum is visited (in QVisits), this is used to automatically calculate the learning rate. In the pseudo code further in this chapter the function *FindIndex()* is used to find the index of a stratum for a given point and direction.

5.2.2 Nearest Neighbor Search

When a ray intersects the scene we need to find the nearest cache point in the point set. This cache point is used for both updating and sampling, so we store the index of the point in the ray itself so that we don't have to recalculate it in both kernels. The points are stored in a grid, each grid cell can store more than one points. A grid cell consists of two integers, an index to the first point in the cell and an index to the last point in the cell. If a cell is empty the first value will be -1. To find the nearest neighbor to the intersection point we find the grid cell corresponding with the intersection point. We then check each point in that cell and each neighboring cell to find the nearest point. Only points with a similar normal as the intersection point are considered. If no point is found in any of those cells we don't update the Q-table for that intersection point and we sample a new direction using just the BRDF. In the pseudo code in the following subsections the function *FindCell()* finds the nearest neighbor to the given intersection point.

5.2.3 Updating

The Q-table is updated by sampling equation 3.4 during path tracing using Monte Carlo integration, which results in:

$$Q(x, \omega) = (1 - \alpha) * Q(x, \omega) + \alpha * (I_e(x', -\omega) + \frac{2\pi}{n} \sum_{i=0}^{n-1} Q_i(x') \rho(\omega_i, x', -\omega) \cos \theta_i) \quad (5.1)$$

Where:

$Q(x, \omega)$	is the value of Q for tracing a ray in direction ω from position x .
x'	is the first hit point of tracing a ray from x into direction ω .
$I_e(x', -\omega)$	is the direct lighting from point x' into direction $-\omega$.
$Q_i(x')$	is the i 'th value of Q at position x' .
$\rho(\omega_i, x', -\omega)$	is the reflectance distribution of radiance at point x' coming from direction ω_i into direction ω .
α	is the learning rate, with $\alpha \in [0, 1]$

In practice we update the Q-table after the rays are traced. We find the nearest cache point at the origin of the ray and update the Q-table at that position with the radiance gained by tracing the ray to its current position. If the current position is a light source the Q-table is updated with the radiance coming from that light source and if the current position is a surface the Q-table is updated with the change in attenuation for hitting that surface.

The updating of the Q-table is performed in a separate kernel to make sure that the table is completely done updating when the sampling starts. Each thread in the kernel processes a single ray. Since multiple rays might want to update the same position in the Q-table at the same time the updating is done using an *atomic add*. To minimize corruption of the table we first calculate the change that is going to be made to that position and then increment the value with the calculated change:

Algorithm 2 Updating the Q-table

```

index = GetThreadIndex()
ray = rays[index]
idxPrev = FindCell(ray.origin, ray.prevNormal)
idxCurr = FindCell(hitpoint, normal)
if ray hit skybox or light source then
    energy = max(GetRadiance(hitpoint))
else if ray hit surface then
    energy = max(GetLastAttenuation(ray)) * QMax[idxCurr]
end if
idxQ = FindIndex(idxPrev, ray.direction)
qValue = Q[idxPrev]
update = ((1 -  $\alpha$ ) * qValue +  $\alpha$  * energy) - qValue
AtomicAdd(Q[idxPrev], update)

```

5.2.4 Sampling

In the sample kernel we determine if rays are terminated or if they continue in a new direction. Rays are terminated when they hit a light source or leave the scene. If a ray hits a purely specular surface or a passable surface its new direction is predetermined, so we don't have to sample the Q-table. For every other surface it is possible to generate a new direction using the Q-table. However, the more specular a surface

is the harder it becomes to generate a valid direction and it might be more beneficial to sample the BSDF instead. We determine how to sample using a hard threshold, if the surface reaches a certain specularity we will sample a direction using the BSDF otherwise we will use the Q-distribution.

For the sampling itself we need a *cumulative distribution function*. This function represented as a table similar to the Q-table. At each position the i 'th value of the CDF is the sum of all Q-values at that position up to i . The CDF can be used to sample a stratum by binary searching the table for a random value between zero and the sum of all Q-values at that position. A direction can then be obtained by uniformly sampling that stratum. This process is shown in algorithm 3.

Algorithm 3 Sampling the Q-table

```

index = GetThreadIndex()
ray = rays[index]
idxCurr = FindCell(hitpoint, normal)
sum = QSum[idxCurr]
r = random value between 0 and sum
idxQ, qValue = BinarySearch(QCdf[idxCurr], r)
direction = UniformSampleStratum(idxQ)
pdf = (qValue / sum) * (n / (2 * π))

```

5.2.5 Learning Rate

The learning rate automatically determined by the number of times a stratum has been sampled. The exact formula for the learning rate that is used in every test is:

$$learningRate = \min \left(0.05, \max \left(0.0001, \frac{1}{1 + QVisits[idqQ]} \right) \right) \quad (5.2)$$

These values were handpicked in a single scene that was not used for any other tests.

6 Results

To test the efficiency of the method we conduct several tests. We test our method on four scenes. Each scene has different characteristics that will influence the behavior and performance of the Q-learning. Our experiments cover three aspects of the implemented method:

1. The distribution of the cache entries (see section 6.1): We test several schemes and parameters and evaluate performance.
2. Q-learning characteristics (see section 6.2): We explore what types of light transport benefit from the method.
3. Performance (see section 6.4): We analyze the overall efficiency of the algorithm.

The memory requirements of the method will be analyzed in section 6.4.

6.1 Caching Point Distributions

We test three caching point distributions:

1. Photon map
2. Poisson Disk Set
3. Poisson Disk Set with light occlusion check

The photon map has one parameter: the minimum rejection distance, which determines the minimum distance between two points. The Poisson disk set has an additional parameter: the maximum rejection distance. The ambient occlusion and light occlusion determine a rejection distance between the minimum and maximum distance. Therefore the maximum distance directly influences the difference in density between parts of the scene where occlusion is high and parts where occlusion is low.

We test each type of caching point distribution on each scene using various parameters. Each setting is tested by running the Q-learning method using the generated point set. We gather statistics for a fixed workload of 50 samples and a fixed time budget of 5 seconds. The quality of the point set is then quantified as the MSE we get when we compare the resulting image with the reference image. The 50 samples test ignores the overhead of the algorithm and provides an upper bound on the gains of the method, given an ideal implementation. The 5 second test measures the overall efficiency of the algorithm and takes into account the quality of the implementation. Therefore we use the set that comes out best in the 5 second test on that scene as the base for the tests on Q-learning characteristics.

6.1.1 Ajar

Figure 6.1 shows the resulting MSE values after running the learning method for 50 samples on the generated point set. Please notice that in these graphs the values on the MSE axis do not start at zero for clearer visibility of the differences.

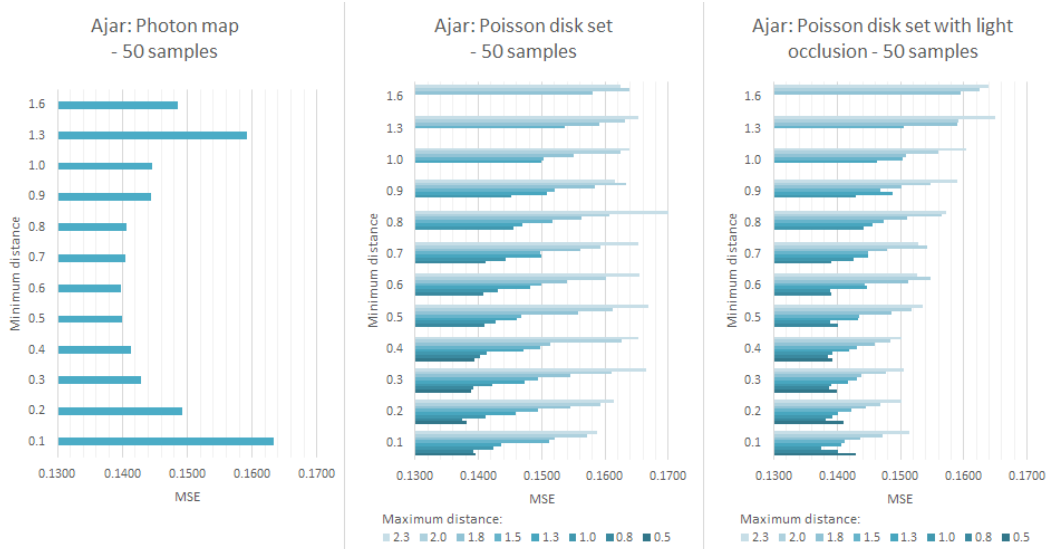


FIGURE 6.1: MSE values after 50 samples of testing each point distribution on the Ajar scene.

The photon map performs best with a minimum rejection distance of 0.6. This results in a quite evenly distributed set of points. However a minimum distance of 0.6 is too large to cover the more complex geometry of the teapots very well.

The Poisson disk set performs best with a much smaller minimum rejection distance of 0.2 and a maximum rejection distance of 0.8. Since the scene consists of a small room and the area we are looking at contains various geometry the ambient occlusion will return medium to high values.

The Poisson disk set with light occlusion performs best when picking a even lower minimum distance of 0.1 and a maximum distance of 1.0. The lights in this scene aren't visible from our viewpoint, so the light occlusion values will be high in most parts of the scene. Because of the very low minimum distance, the teapots will now be adequately covered by the points.

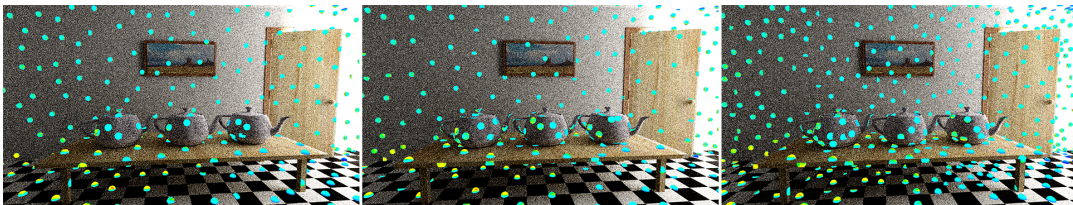


FIGURE 6.2: The three best performing point sets for each type after 50 samples on the Ajar scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

Figure 6.2 shows that the photon map and the Poisson disk set without light occlusion yield very similar point sets. However, the Poisson disk set performs better than the photon map. The Poisson disk set with light occlusion results in a more dense points set and performs about the same as the set without light occlusion.

Figure 6.3 shows the resulting MSE values after running the learning method for 5 seconds on the generated point set.

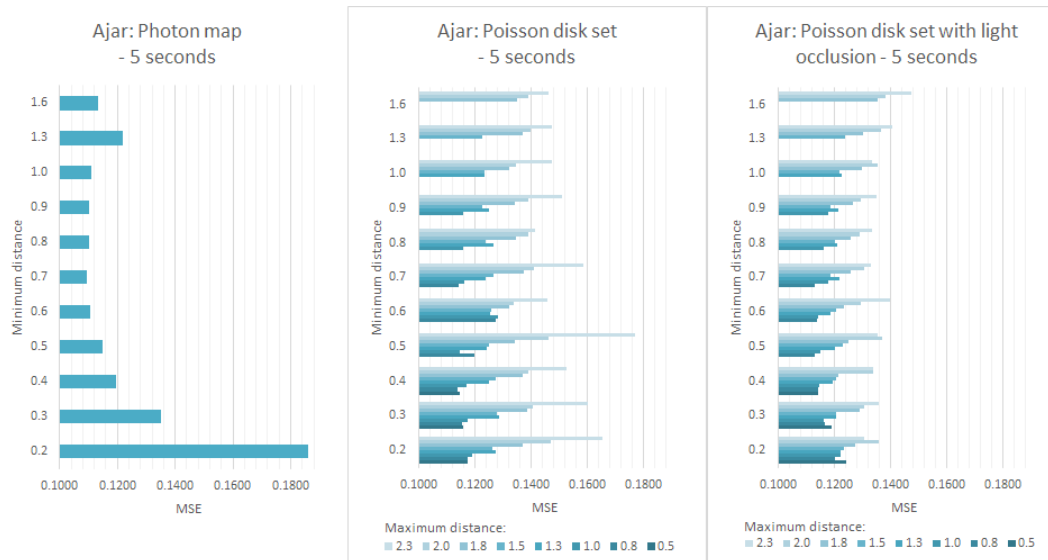


FIGURE 6.3: MSE values after 5 seconds of testing each point distribution on the Ajar scene.

In this time-based test the photon map performs best with a minimum distance of 0.7. Meaning that the resulting set is a little bit less dense than in the sample-based test.

The Poisson disk set now performs best with a minimum distance of 0.4 instead of the 0.2 of the sample-based test. The maximum rejection distance is now 0.8, so the resulting set is a little less dense than before and is more uniform as the difference of the minimum and the maximum rejection distance has decreased.

The Poisson disk set with light occlusion has a minimum distance of 0.7 and a maximum distance of 0.8, which differs quite a lot from the original combination of 0.1 and 1. This small difference between the minimum and maximum distance yields a very uniform point set, not unlike the photon map.

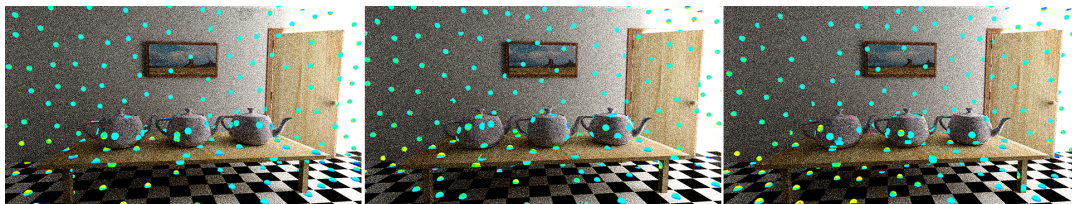


FIGURE 6.4: The three best performing point sets for each type after 5 seconds on the Ajar scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

Figure 6.4 shows the resulting sets of points. Now all three types look practically identical.

6.1.2 Sponza

The Sponza scene is much larger than the Ajar scene and is directly lit from a large area light far above the scene, making this quite an easy scene to render. As with the Ajar we tested the three different point sets with several different parameters. The resulting MSE values after running the method on the point set for 50 samples can be seen in figure 6.5.

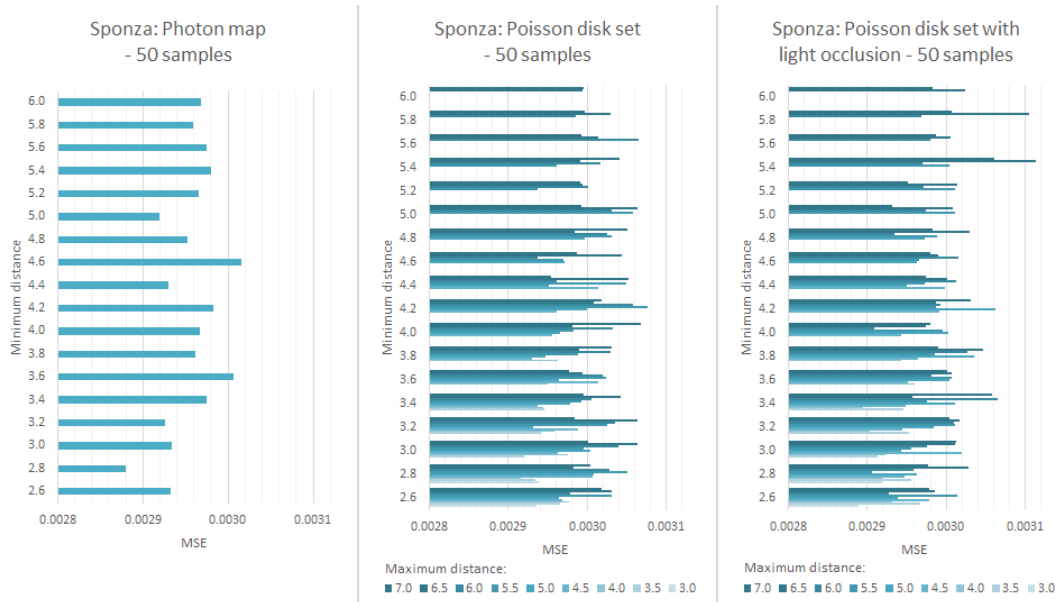


FIGURE 6.5: MSE values after 50 samples of testing each point distribution on the Sponza scene.

The resulting values are all very close to each other. All MSE values are between 0.0029 and 0.0031. The test returns the photon map as the overall best. However, as the values are so close together this can be because of random noise.

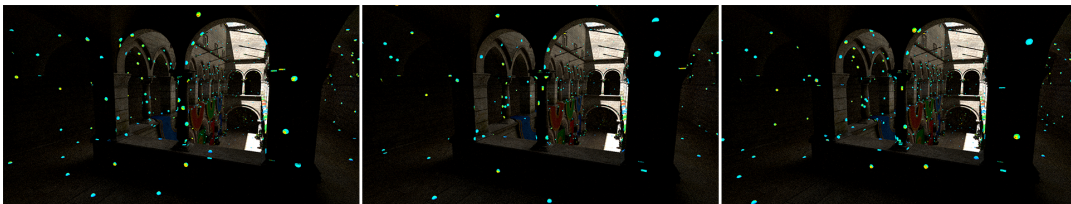


FIGURE 6.6: The three best performing point sets for each type after 50 samples on the Sponza scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

There seems to be almost no difference between the density and placing of the point in the three resulting sets.

In figure 6.7 we see the resulting MSE values after 5 seconds of Q-learning on each given setting. The best result of this test is what is used for further tests of the method on this scene.

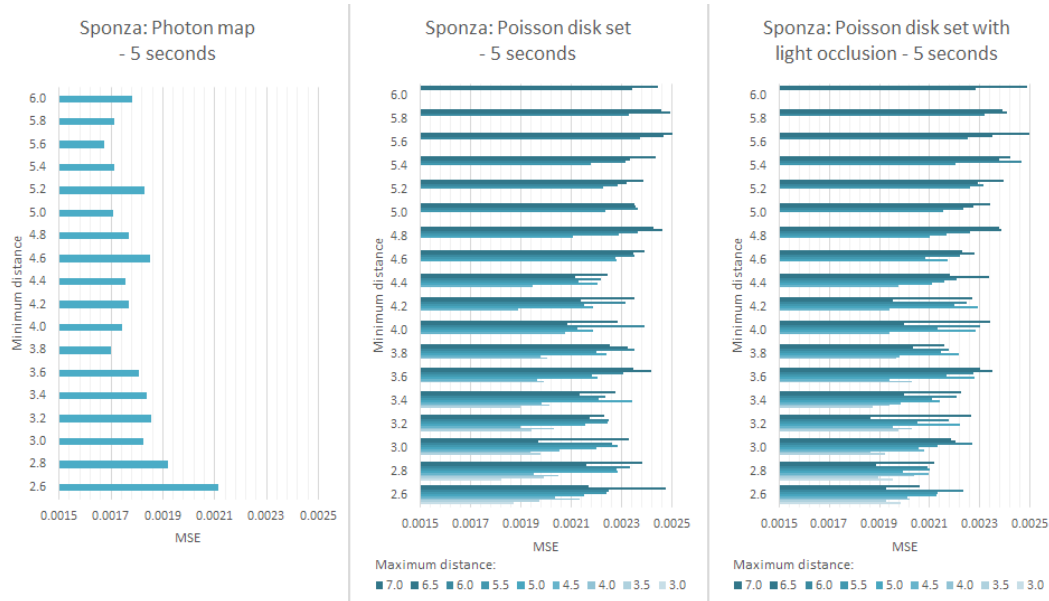


FIGURE 6.7: MSE values after 5 seconds of testing each point distribution on the Sponza scene.

The photon map performs best with quite a large minimum rejection distance. Although, as with the 50 sample test, the resulting values are very close together. A similar MSE value of 0.0017 can also be seen at a minimum distance of 3.80, which produces quite a different set from the chosen 5.60.

The best result of the Poisson disk set comes close to the photon map when using a small difference between the minimum and maximum rejection distance.

The Poisson disk set with Light occlusion comes in last with a medium minimum distance and a large maximum distance. The differences between the results of the three sets are minimal. A difference of 0.0001 in MSE is far from substantial.

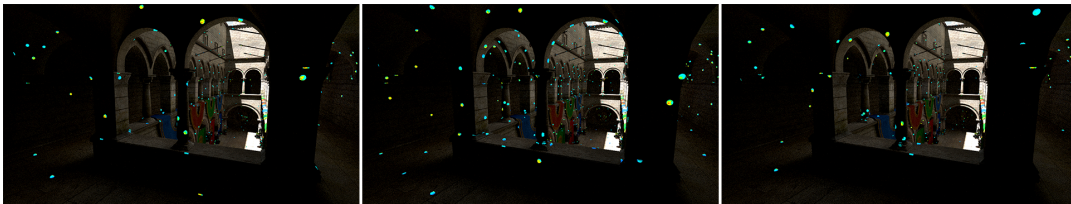


FIGURE 6.8: The three best performing point sets for each type after 5 seconds on the Sponza scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

When looking at the resulting sets we can see that in this test there are more differences than in the 50 samples test. Especially the Poisson disk set produces a denser set of points due to its lower minimum distance.

6.1.3 Bedroom

The bedroom scene is about the same size as the Ajar scene, but there is a lot more geometry in the scene. This would mean that a denser point set is required to apply Q-learning in every part of the scene. As with the previous two scenes we first tested each set with several parameters by running the method for 50 samples, and then for five seconds. The results of the 50 sample test can be seen in figure 6.9.

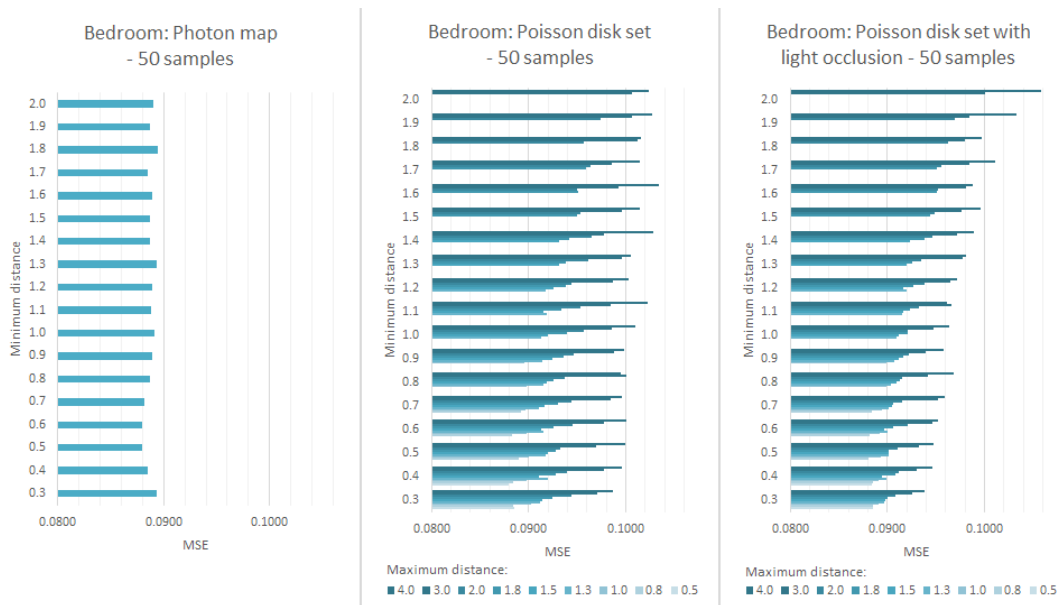


FIGURE 6.9: MSE values after 50 samples of testing each point distribution on the Bedroom scene.

The photon map performs best with a minimum distance of 0.6, same as in the Ajar scene. Although the results of 0.5 and 0.7 are so close that the difference in quality is unnoticeable.

The Poisson disk set performs about the same as the photon map, but with a lower minimum distance and a maximum distance of 0.5. Resulting is a somewhat denser point set, as can be seen in figure 6.10.

The Poisson disk set with light occlusion performs just a tiny bit worse than the other two sets. But the difference is too small to notice. The set performs best with a minimum distance of 0.5 and a maximum distance of 0.8.

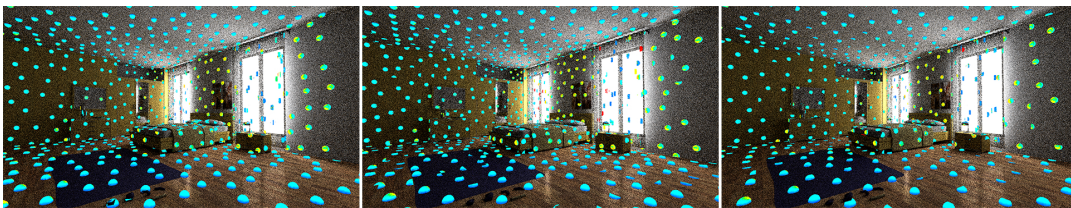


FIGURE 6.10: The three best performing point sets for each type after 50 samples on the Bedroom scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

The five second test produces somewhat different results. The best result from this test is what is used to test the performance of the learning in this scene. Results are shown in figure 6.11.

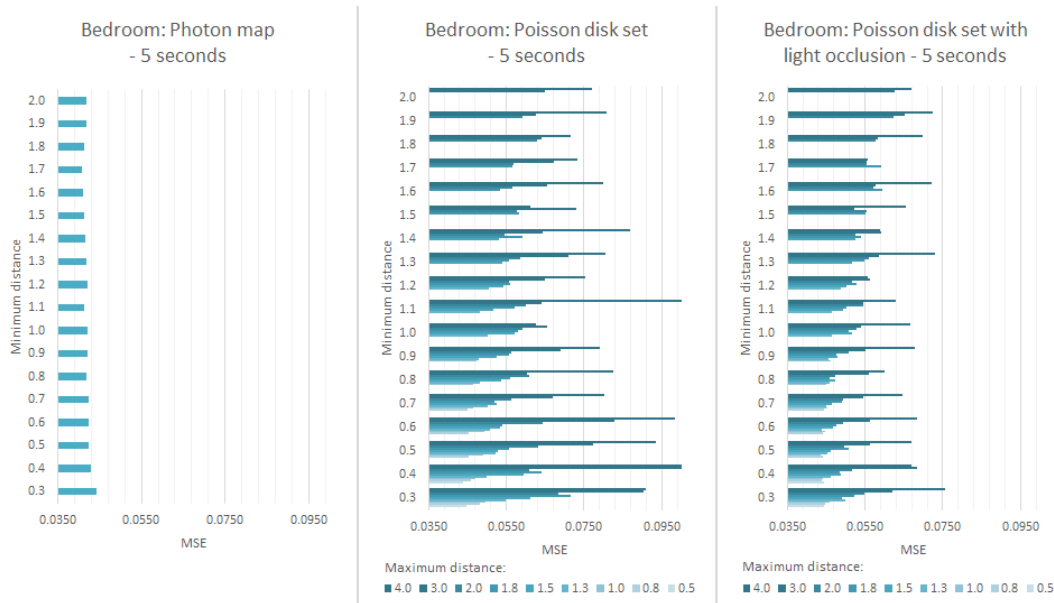


FIGURE 6.11: MSE values after 5 seconds of testing each point distribution on the Bedroom scene.

The photon map now needs a much higher minimum distance than in the 50 sample test. And also higher than the photon map in the 5 second test of the Ajar scene. This would lead to a less dense set as can be seen in figure 6.12.

The Poisson disk set interestingly performs best with the same parameters as in the sample test. The set is now much denser than the photon map, as can be seen in figure 6.12.

The Poisson disk set with light occlusion now works best with a slightly larger maximum distance of 1.0. Leading to a set that looks very similar to the photon map.

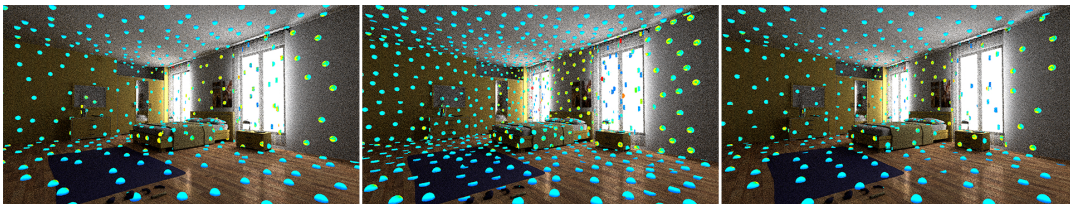


FIGURE 6.12: The three best performing point sets for each type after 5 seconds on the Bedroom scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

6.1.4 Bidir Room

The Bidir Room is quite a small scene without complex geometry. The difficulty in this scene is in the small light sources that are somewhat hard to reach and the glass egg that causes a caustic.

Figure 6.13 displays the resulting MSE values after 50 samples of learning enabled rendering on the point set created with the specified parameters.

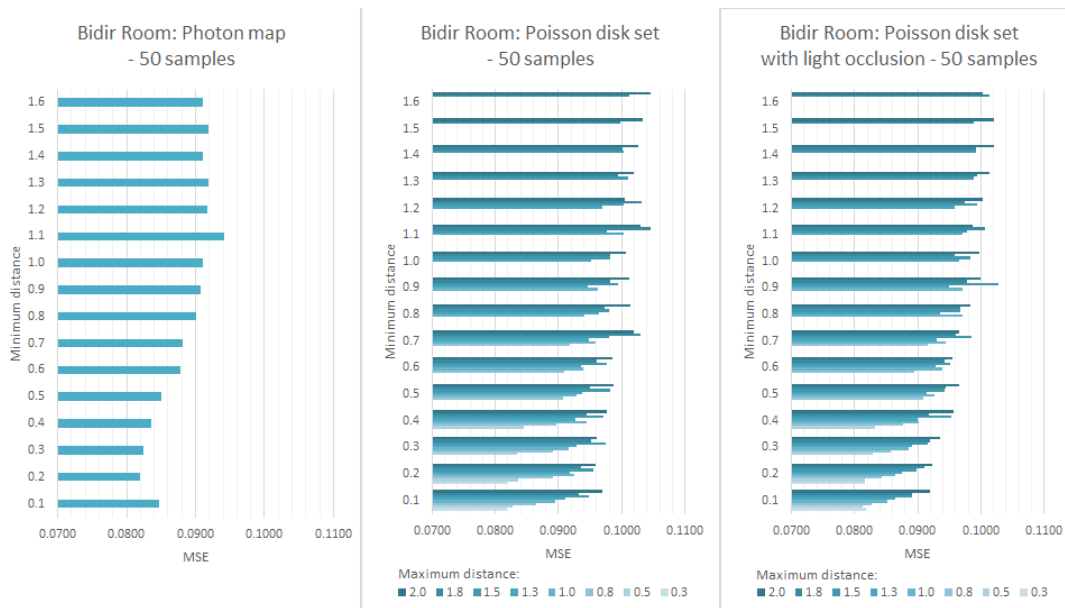


FIGURE 6.13: MSE values after 50 samples of testing each point distribution on the Bidir Room scene.

In this scene we see that the photon map now prefers a very short minimum rejection distance as opposed to the larger minimum distances in the other scenes on the 50 sample test. This results in a very dense set of points, covering the entire area of the scene quite well.

Just like the photon map the Poisson disk set also picks a short minimum distance. This, combined with the short maximum distance, again results in a very dense point set that is very similar to the result of the photon map. As can be seen in figure 6.14.

The Poisson disk set with light occlusion performs best with a very short minimum distance and a medium maximum distance of 0.5. This still results in a very dense set, although the points are now less evenly distributed. The set is less dense along the side walls, as the Ambient Occlusion is low there.

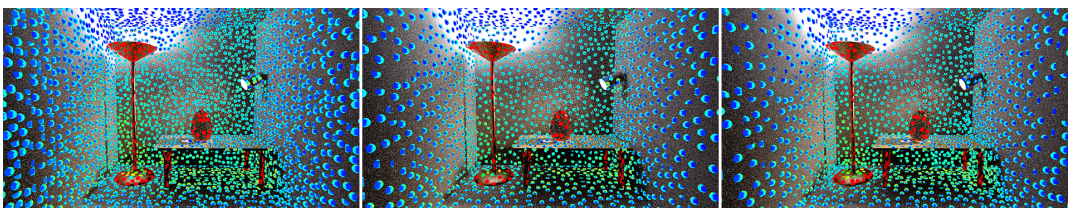


FIGURE 6.14: The three best performing point sets for each type after 50 samples on the Bidir Room scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

In figure 6.15 we see that the 5 second test results in quite similar point sets in this scene.

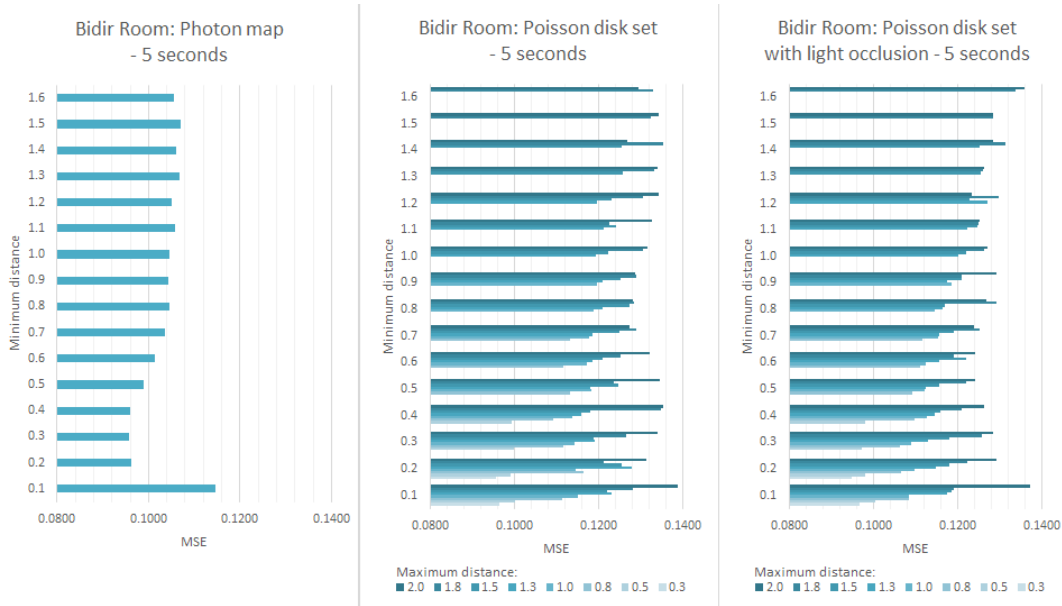


FIGURE 6.15: MSE values after 5 seconds of testing each point distribution on the Bidir Room scene.

The photon map now prefers a slightly larger minimum distance, resulting in a slightly less dense set.

The Poisson disk set performs best with the same parameters as in the 50 sample test, which obviously results in roughly the same set (randomness in the creation of the point set results in some very minor differences).

Interestingly the Poisson disk set with light occlusion now prefers the same settings as the Poisson disk set without light occlusion. The enabling of the light occlusion makes the set a little bit denser than the regular Poisson disk set. This can be seen in 6.16.

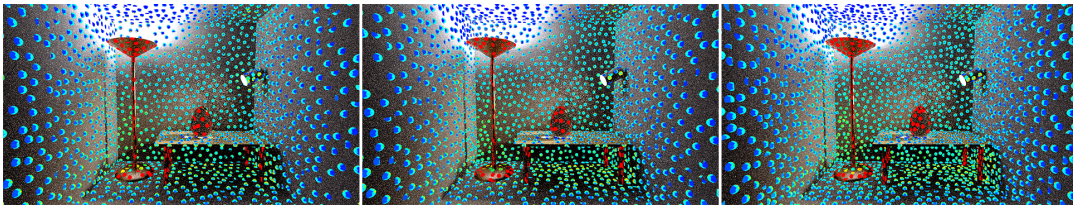


FIGURE 6.16: The three best performing point sets for each type after 5 seconds on the Bidir Room scene. Left: photon map. Middle: Poisson disk set. Right: Poisson disk set with Light Occlusion

6.2 Q-learned Importance Sampling

To test the learning algorithm itself we run several tests. Each test evaluates a different aspect of the method. These are the tests that are ran for every scene:

1. Standard: We run the standard path tracer for a number of samples and measure the MSE with the reference image. This result is used for comparison with the other tests.
2. Same samples: The Q-learning method is run for the same number of samples, again measuring the resulting MSE value. This test provides an upper bound on the gains of the algorithm, given an ideal implementation.

3. Same time: We now run the learning method for the same time as it took to produce the result of the standard test. This provides insight in the overall efficiency of the method and takes into account the quality of the implementation.
4. Same quality: The learning method is ran until the same MSE value is reached as in the standard test. This test again displays the overall efficiency of the method by showing what it takes to produce the same image.

Two more tests are conducted that closely measure the characteristics of the method as it progresses:

1. Sample efficiency test: We run 1000 samples of path tracing and track the progress along the way. We measure the MSE, percentage of rays returning energy, average number of bounces per ray and average energy per bounce after every sample for the first 50 samples, then after every fifty samples.
2. Time efficiency test: We path trace for 60 seconds and track the MSE after every second.

These tests are conducted three times: once using the regular path tracer, once using the learning method and once using the learning method after training for 500 samples.

We first run these tests on every scene with NEE and Russian roulette enabled, as any modern path tracer will have these features. To compare our results to Dahm and Keller's we run the tests again on the Ajar scene without NEE and Russian roulette.

6.2.1 Ajar

The Ajar scene almost entirely consists of indirect lighting, meaning that most paths will struggle to find a light source. This makes it a scene that can really benefit from efficient path guiding. Figure 6.17 shows a comparison of our method versus conventional path tracing. These tests are run with NEE and Russian roulette enabled.

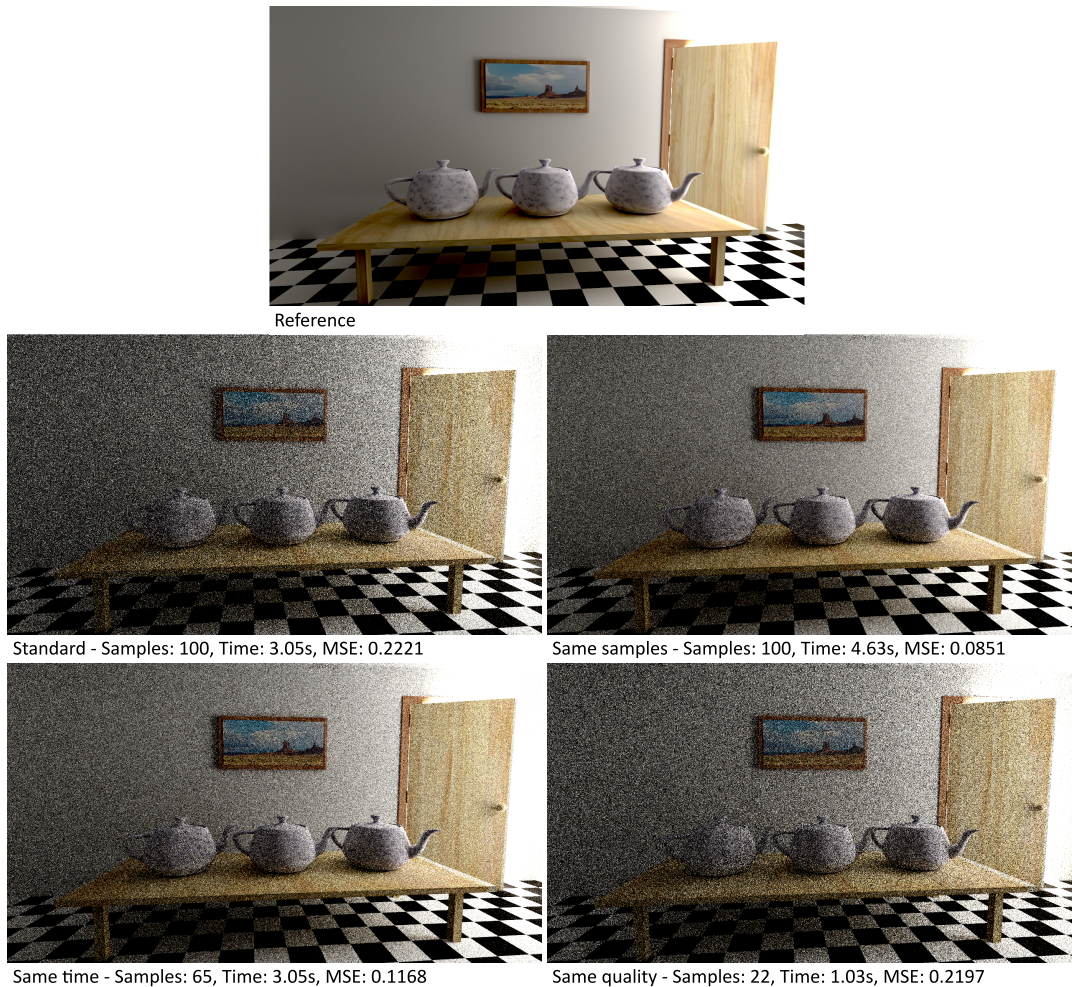


FIGURE 6.17: Comparison of results on the Ajar scene. Compares 100 samples of the standard path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

We see that in this scene the Q-learning method outperforms conventional path tracing in every possible way. After running both the standard path tracer and the learning method for 100 samples we see that the learning method produces a much clearer image with less than half the MSE. The walls and flooring of the scene are smoother with Q-learning enabled, especially in the darker parts of the scene. We can also see that the method does a decent job rendering the teapots, even though these surfaces are not completely covered by the point set.

If we run the method for the same time we see that even when taking into account the overhead costs the learning method is still clearly producing a higher quality image than the standard path tracer. This is backed up by the result of the quality test: an image of the same quality can be produced in less than half the time.

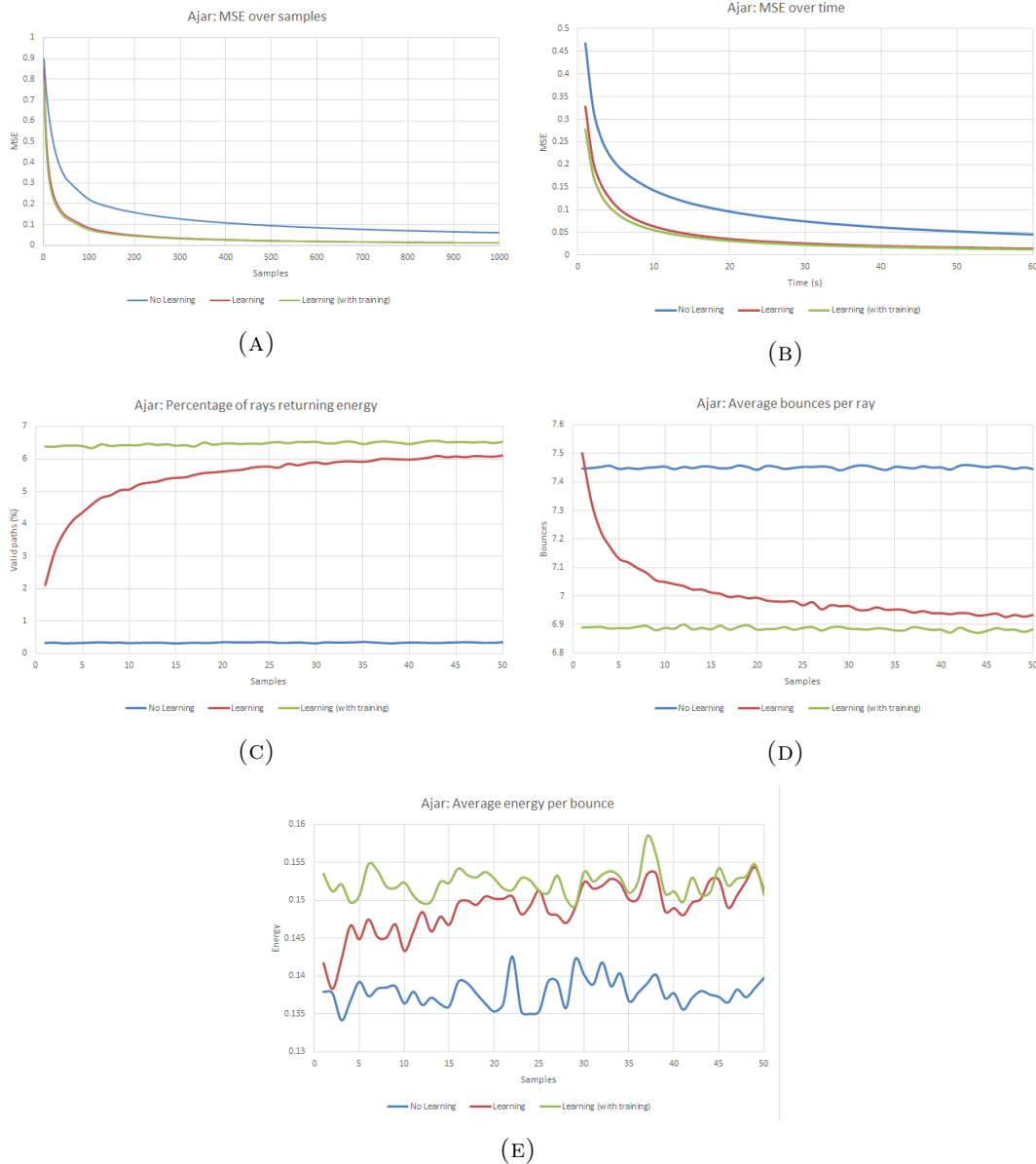


FIGURE 6.18: Results of the Q-learning tests on the Ajar scene

Figure 6.18a shows the development of the MSE over samples. The MSE converges significantly faster while Q-learning. Only after 1000 samples does the standard path tracing method start to catch up with the learning method. The training beforehand does produce a better result, but the difference is quite small.

If we look at the development of the MSE over time in figure 6.18b we see that the learning method still outperforms the standard method. Even though the time to process a single sample increases because of the updating and sampling of the Q-table.

In figure 6.18c we see the percentage of rays that return energy. Keep in mind that all rays that exceed 16 bounces will be terminated and that the Russian roulette can terminate a ray before it finds a light source. This influences the number of rays returning energy, but it has the same impact on the method with or without learning. This is quite a complex scene so the numbers are low. In the standard path tracer less than one percent (0.33%) of the rays actually return energy. Keep in mind that contributions from shadow rays are not taken into account when calculating this value.

The learning method starts out low, but slowly increases until it reaches about 6% after 50 samples. The learning with training starts a bit higher, at about 6.4%, and does not change much over the course of 50 samples.

Figure 6.18d shows the average number of bounces a ray makes before it is terminated. As explained above, keep in mind that the maximum bounce limit and the Russian roulette influence these numbers. With learning the number of bounces starts out higher than the standard path tracing, but quickly reaches a lower value of 6.9.

The average energy per bounce in figure 6.18e is very noisy. This is because the samples are still random; there is always a chance that a ray that had a really small chance of returning radiance actually does find a light source. If that happens a couple of time the sample will return more energy. However, even though the result is noisy it is still clearly visible that the Q-learning method carries more energy per bounce than standard path tracing.

6.2.2 Sponza

Even though the camera is placed in a dark position in the Sponza scene the amount of light coming from the sky makes this quite an easy scene to render. Because of this the scene is not as ideal as the Ajar scene for the Q-learning method. We performed the same tests as with the Ajar scene so that we can compare the effectiveness of the method in the two scenes. The comparison of our method and the standard path tracer can be seen in figure 6.19.

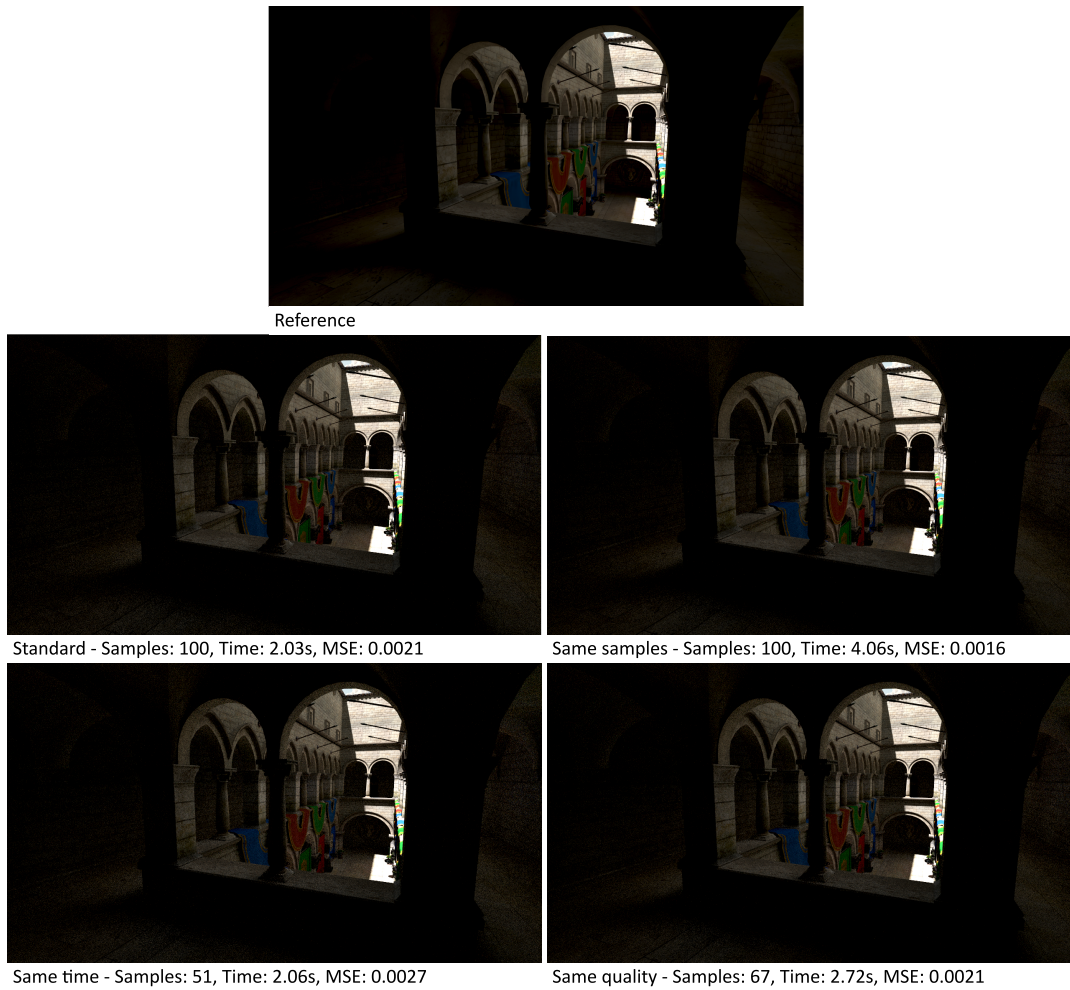


FIGURE 6.19: Comparison of results on the Sponza scene. Compares 100 samples of the standard path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

The difference in noise levels are not as substantial in this scene when comparing to the Ajar scene. Minor differences can be seen when zooming in to the darker sections of the image. When run for the same number of samples the method produces a somewhat clearer image. However, it takes longer to reach the same quality image, meaning that it is more efficient to use the standard path tracer in this scene. This is again visible in the time test: after the same time the learning method produces an image with more noise than the standard path tracer, although the differences are minor.

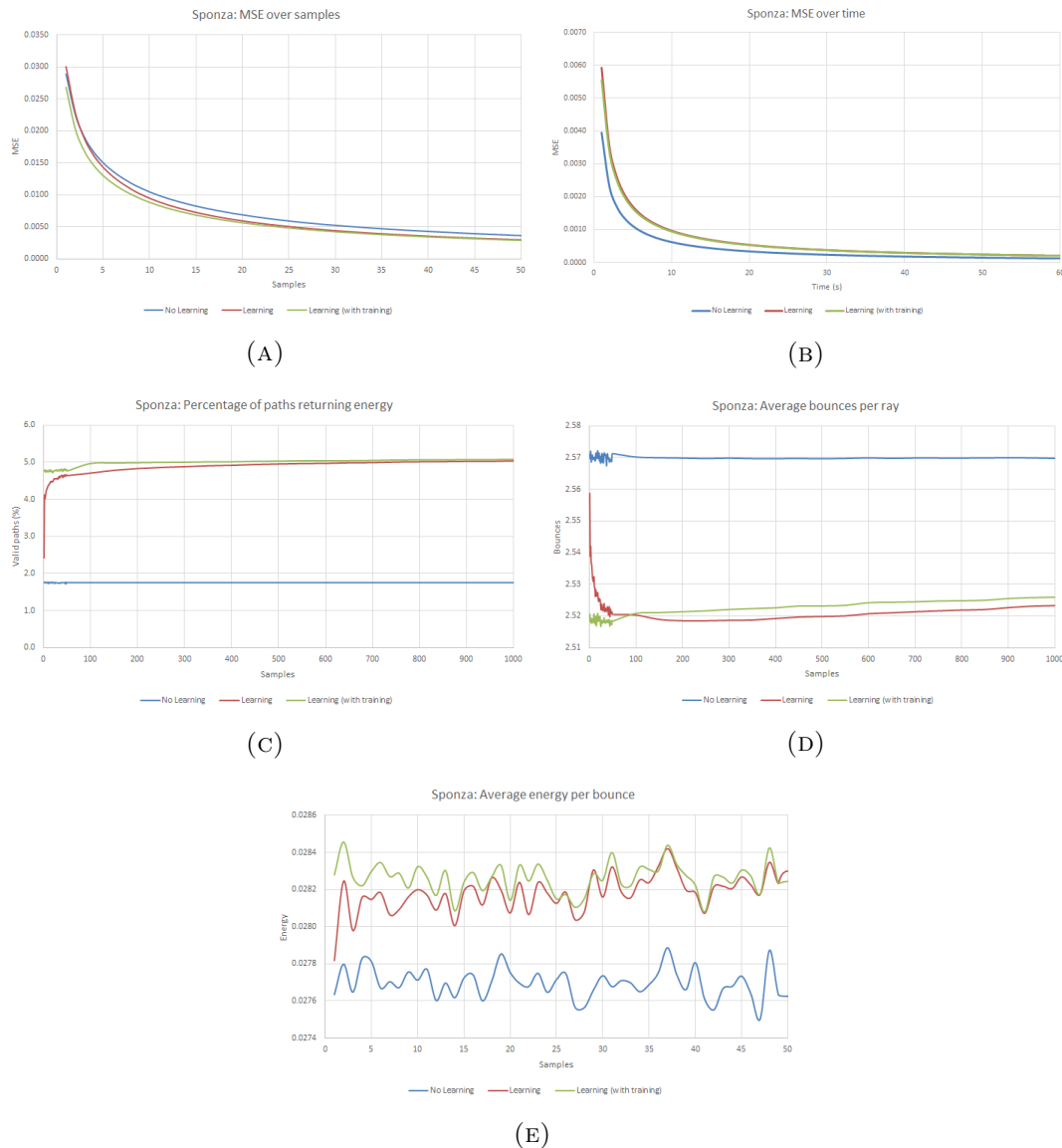


FIGURE 6.20: Results of the Q-learning tests on the Sponza scene

In figure 6.20a we see that the learning method still performs best over the number of samples, especially in the first 10 samples. After that the methods perform about the same.

When looking at the MSE over time in figure 6.20b we see that the standard path tracing performs significantly better in the first 20 seconds. The learning method only catches up after 60 seconds.

In figure 6.20c we can see that while the percentage of valid paths is still higher with Q-learning enabled we don't reach the same heights as in the Ajar scene (5% instead of 6.5%). And while the percentage of valid paths has decreased for the learning method it has increased for the standard path tracing (1.8% instead of 0.3%), decreasing the difference between the two methods.

We can see a similar trend in the average number of bounces per ray (figure 6.20d); even though the learning method still performs better than the standard path tracing the overall number of bounces is much lower in this scene and the difference between the methods is much smaller.

The average energy per bounce is slightly higher when learning, but the difference is minimal. Comparing energy levels between different scenes is difficult because the total energy level differs each scene. However we can see that the relative difference between the energy levels in the Ajar scene (about 11.7%) opposed to the difference in the Sponza scene (about 2.2%) has substantially decreased.

6.2.3 Bedroom

The bedroom scene is similar to the Ajar scene in that the light sources are hard to reach. The difference is that instead of the rays having to find a small opening the rays now have to bounce through some of the geometry (the curtains and the window) to find the light sources.

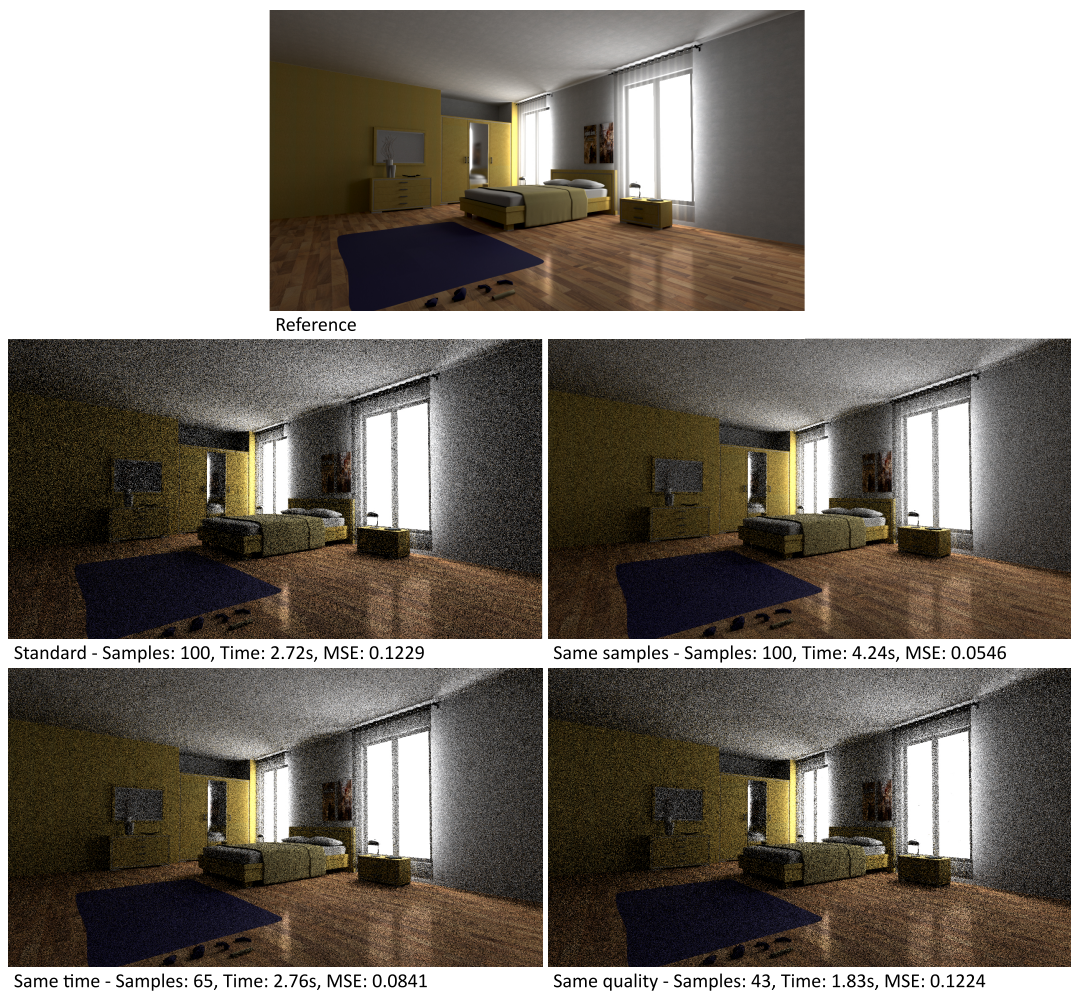


FIGURE 6.21: Comparison of results on the Bedroom scene. Compares 100 samples of the standard path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

In figure 6.21 we see that in this scene the method outperforms the standard path tracer, although the differences are not as substantial as in the ajar scene. Comparing the results after 100 samples we see that the learning method produces a much

smoother image; the MSE more than halves. The differences in noise are clearly visible in the whole scene, especially in the reflection of the window on the ground and on the gray walls / ceiling. This shows that the Q-learning is capable of finding the light coming through the window and directing the paths towards them.

When taking into account the overhead costs of the method the differences are less significant, but still clearly visible. We produce an image of the same quality in roughly 67% of the time.

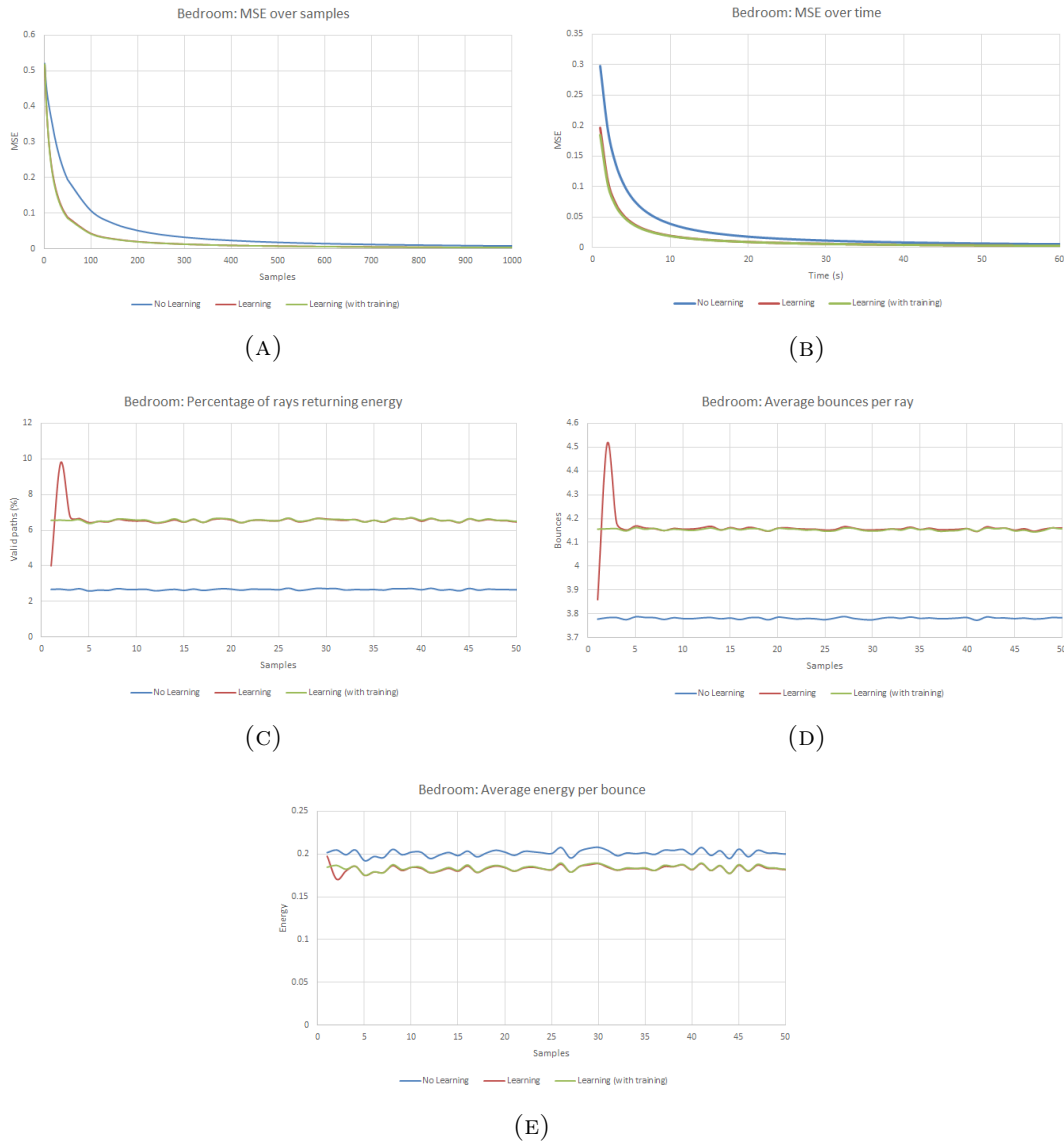


FIGURE 6.22: Results of the Q-learning tests on the Bedroom scene

A more detailed view of the progress of the method can be found in figures 6.22a and 6.22b. Here we see the development of the MSE over samples and over time. In both cases we see that the learning method quickly reduces noise in the first 200 samples or 10 seconds after which the render is getting so close to the reference that it becomes harder to improve.

In this scene we see again that the percentage of valid paths has increased marginally, but not as much as in the Ajar scene. We can see that even though the learning method still reaches about the same height the basic path tracing performs better in this scene than in the Ajar scene.

After the second frame there is a spike in the percentage of valid paths that goes up to almost 10 percent. In the third frame this increased efficiency is lost again and the method stabilizes at about 6.8 percent.

When measuring the average number of bounces and the average energy per bounce we get some interesting results. In figure 6.22d we see that the number of bounces actually increases in this scene when learning. On top of that we can see the same spike as in figure 6.22c.

In figure 6.22e we see that the energy per bounce when learning is now lower than the standard path tracer. Of course, the higher the average number of bounces the lower the average energy per bounce. Again we see the same spike as in figures 6.22c and 6.22d.

6.2.4 Bidir Room

The Bidir Room scene is again a quite difficult scene to render. This scene consists of two small light sources enclosed in lampshades. This makes the light harder to reach than in Sponza, but not as hard as in the Ajar scene, as there is still quite a large portion of the scene in direct visibility of the lights. The main problem in this scene is the caustic created by the glass egg. Caustics can be rendered quite fast by a Bidirectional path tracer [23]. The Q-learning based importance sampling should be able to learn that there is a lot of light coming through the egg and guide the rays towards it.

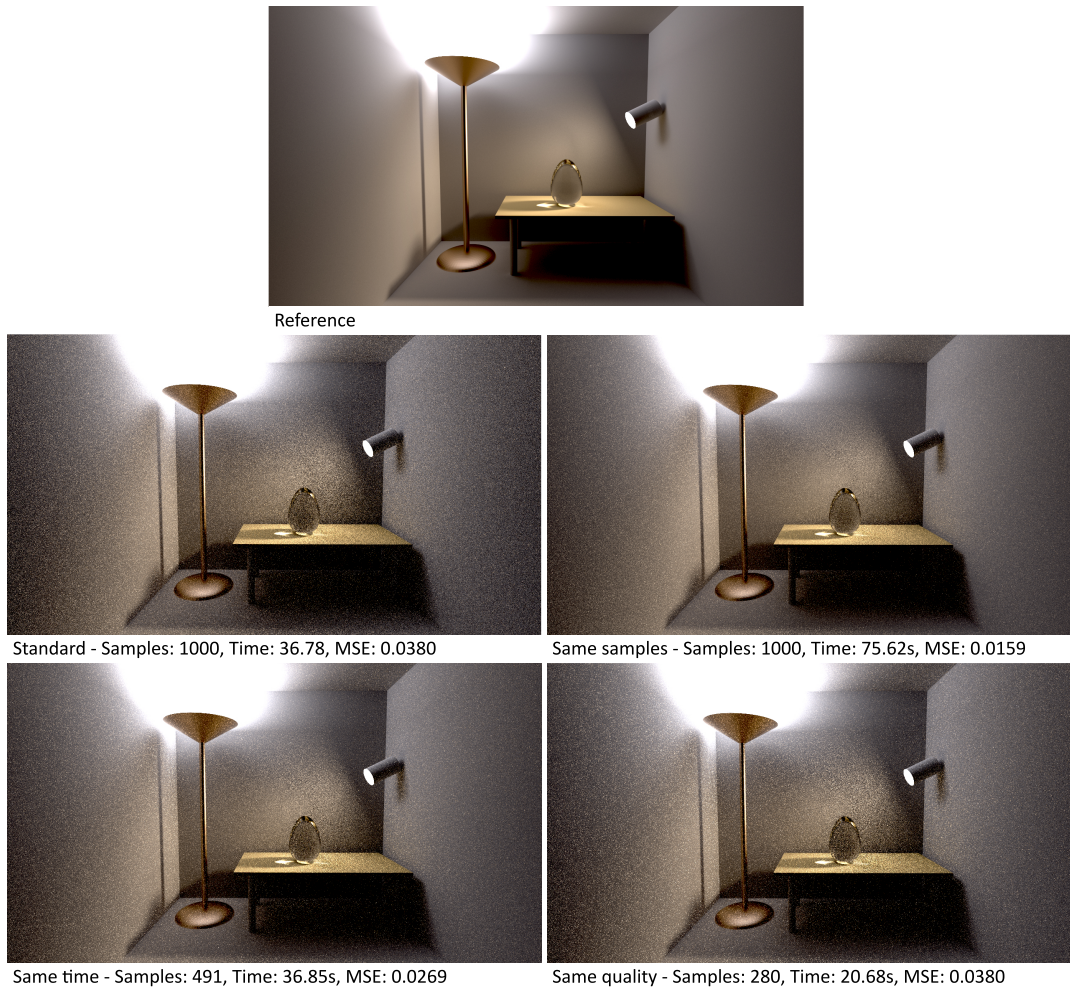


FIGURE 6.23: Comparison of results on the Bidir Room scene. Compares 1000 samples of the standard path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

In this scene the learning method again outperforms the standard path tracer. In the comparison of 1000 samples we see that both the standard path tracer and the learning method have decently rendered the caustic. The main difference between the image is in the noise on the wall, of which most of it is caused by the glass egg. The learning method handles the situation better and the walls look much smoother. The resulting MSE value of the learning method in the 1000 sample test is less than half the MSE value of the standard path tracer.

In the same time the method still produces a higher quality image. Differences are clearly visible in the lighting on the back wall and on the sides. The caustic does not differ much from the standard path tracer. Again, using Q-learning we can produce an image of the same quality in 56% of the time.

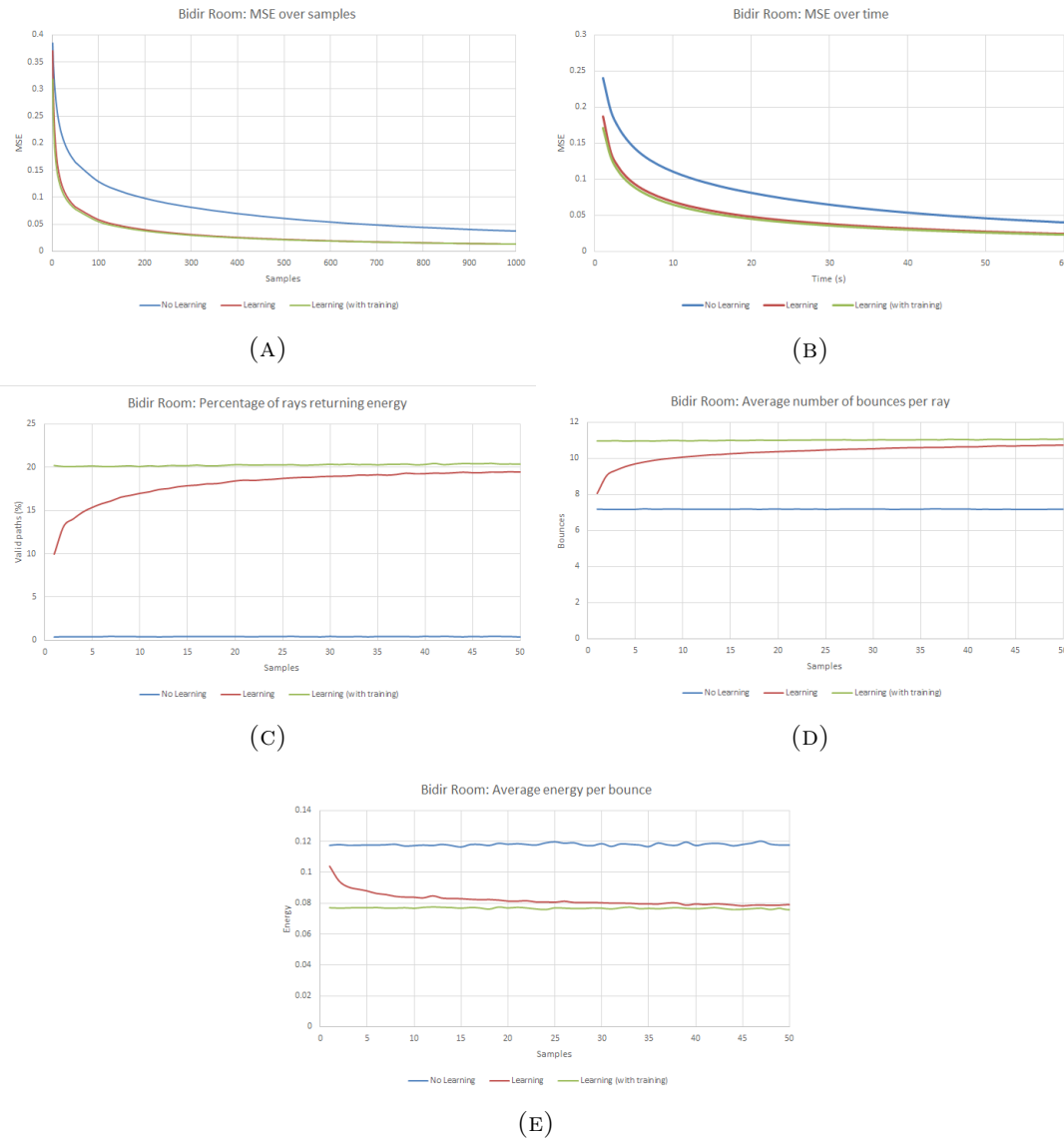


FIGURE 6.24: Results of the Q-learning tests on the Bidir Room scene

When looking at the progression of the MSE over 1000 samples in figure 6.24a we see that the learning method indeed converges a lot faster. Even after a thousand samples the MSE value of the learning method is less than half of the MSE of the standard path tracer, which means that the image is about four times less noisy. However, the long render times in this scene do take a toll on the effect of the method. Figure 6.24b shows that the difference is much less substantial when taking into account the render time of each method.

The increase in the percentage of valid paths is quite substantial in this scene. Figure 6.24c show that in 50 samples the learning method has almost 20 percent of the rays returning energy, while the learning method produces a steady 0.5 percent of valid paths.

Just as in the bedroom scene we see that the number of bounces goes up instead of down with the Q-learned importance sampling enabled. Same goes for the average

energy per bounce, which decreases instead of increases. Another interesting observation is that the energy per bounce seems much more stable than in the other scenes. It doesn't fluctuate as much.

6.2.5 Influence of NEE and Russian roulette

Russian roulette and Next Event Estimation (explained in section 2.3) are two very common techniques to improve path tracing. They will be present in any real-time path tracer and can be easily used in combination with the Q-learning method. Therefore we choose to run most tests with NEE and Russian roulette enabled.

However, in this section we will test the Q-learning method on the Ajar scene without NEE and Russian roulette so that we can compare the results with Dahm and Kellers method [6]. As they compare their method to a basic path tracer without these optimizations.

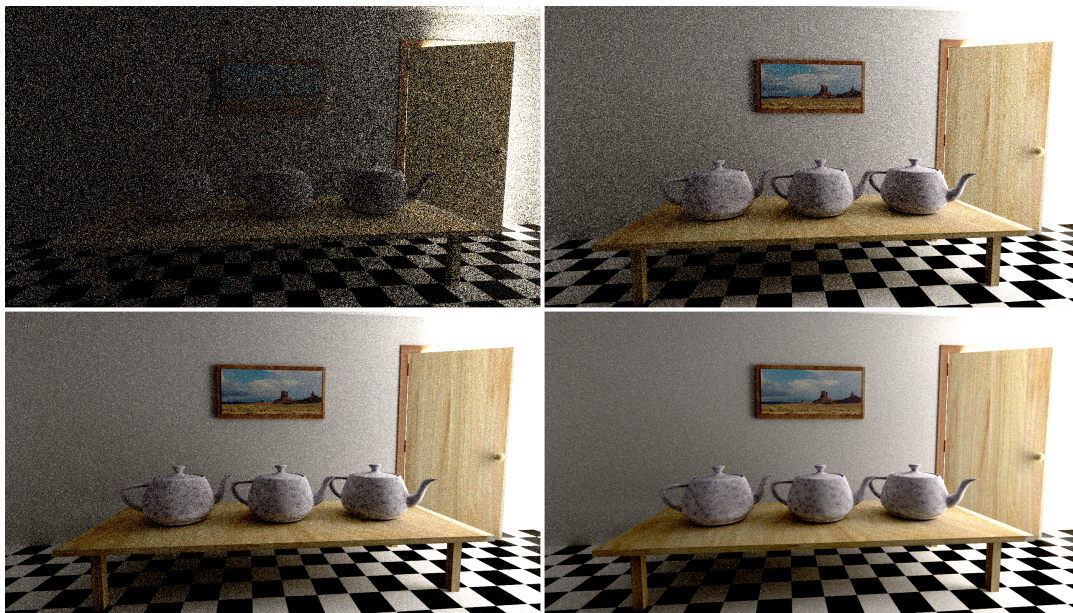


FIGURE 6.25: Comparison of results after 1024 samples. The left column shows the difference between basic path tracing (top) and the learning method (bottom) without NEE and Russian roulette. The right column shows the difference between basic path tracing (top) and the learning method (bottom) with NEE and Russian roulette.

The impact that NEE and Russian roulette have on path tracing can be seen in figure 6.25. On the top row you can see that the influence on the basic path tracer is quite substantial. Without NEE and Russian roulette the path tracer produces a very dark and noisy image. The bottom row show that the influence of NEE and Russian roulette is much smaller when Q-learning. When comparing the bottom-left and top-right image you see that Q-learned importance sampling is more efficient than NEE and Russian roulette. Of course the methods can be easily combined and this still produces the best result. This is backed up by the MSE values in table 6.1.

TABLE 6.1: Comparison of results after 1024 samples.

	Without NEE and RR		With NEE and RR	
	Time (s)	MSE	Time (s)	MSE
No Learning	489.98	0.5458	41.82	0.0593
Learning	205.28	0.0319	74.82	0.0121
Learning (with training)	201.31	0.0306	74.22	0.0115

However, when looking at the running times in table 6.1 you can see that path tracing without NEE and Russian roulette costs much more time. What's interesting is that without NEE and Russian roulette the learning method is about twice as fast as the basic path tracing, while with it it takes about double the time.

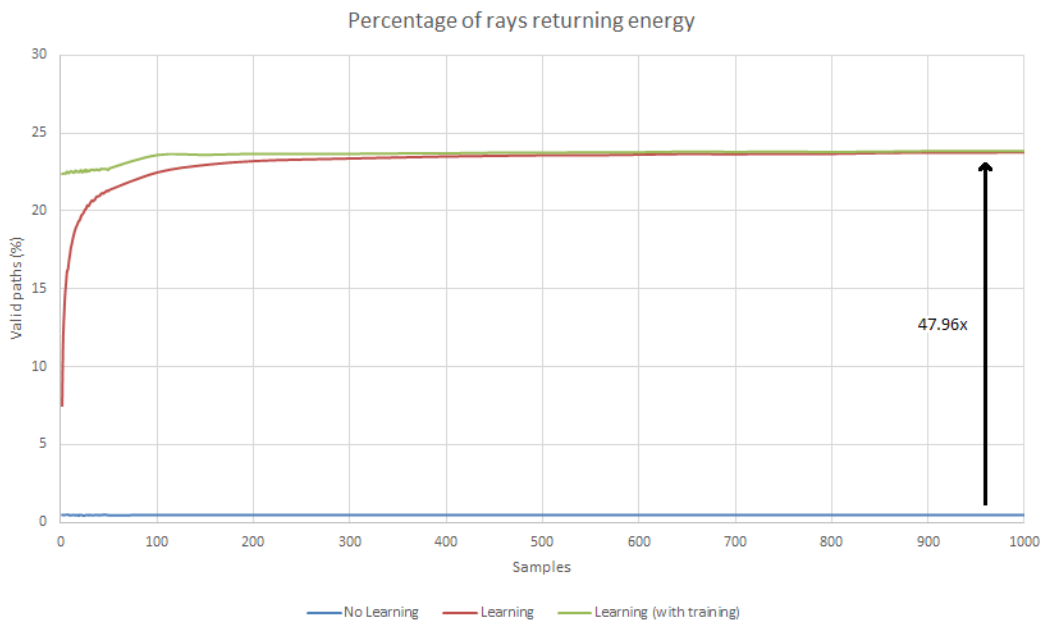


FIGURE 6.26: The percentage of paths that return energy (valid paths) over 1024 samples without NEE and Russian roulette.

Figure 6.26 shows that without NEE and Russian roulette the learning method has a much higher percentage of paths returning energy than with NEE and Russian roulette. While the basic path tracing produces about the same result as before. After 1024 samples the percentage of valid paths is 47.96 times higher with learning than without. This is close to the 43.49 times increase measured by Dahm and Keller [6]. However, our GPU based implementation rises much faster. Our implementation reaches its maximum at roughly 300 samples, while Dahm and Keller's CPU implementation reaches its maximum after 1024 samples.

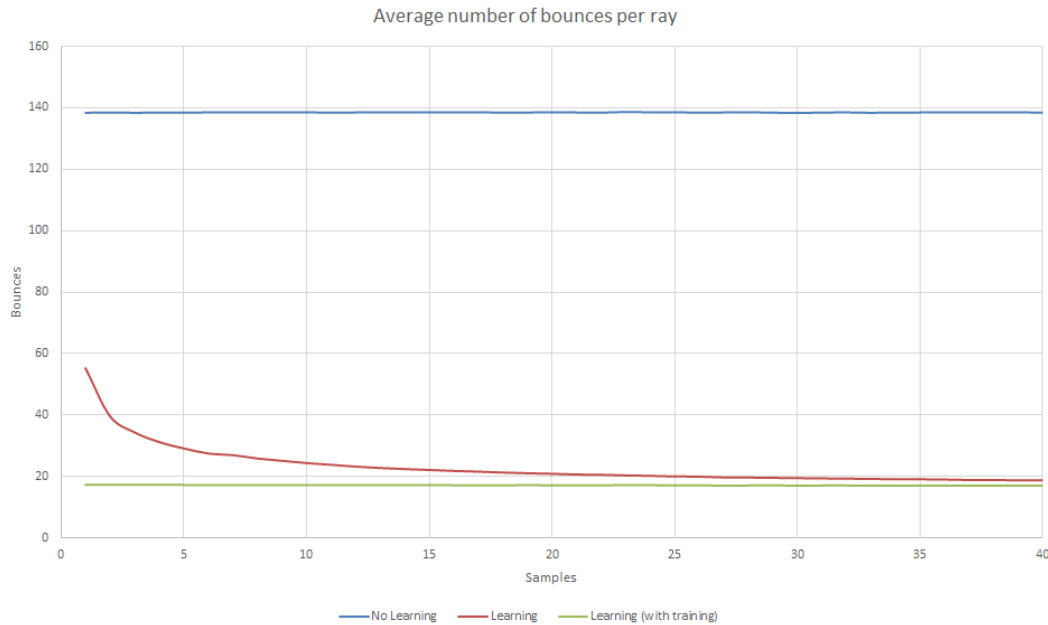


FIGURE 6.27: The average number of bounces per ray over 40 samples without NEE and Russian roulette.

The difference between the number of bounces increases substantially without NEE and Russian roulette. The basic path tracing averages at about 139 bounces per ray, while the Q-learning method converges to about 19 bounces per ray. This is a decrease in bounces of 86.33%, which is much more than the 40% measured by Dahm and Keller in the same scene[6]. Furthermore, what's interesting is that the average number of bounces when Q-learning starts at 55 after the first sample.

6.3 Comparison to Bidirectional

The previous results show that the Q-learning based importance sampling technique performs best on scenes where the light is hard to reach. These scenes are also ideally suited for a bidirectional path tracer. As with this Q-learning method, a bidirectional path tracer has an increased render time, but the possibility to produce a higher quality sample to make up for it.

In this section we compare the learning method to a bidirectional path tracer. The results from the bidirectional path tracer come from a different path tracer than the one used for the learning method. Therefore the results of the bidirectional path tracer are compared to its own reference image.

Comparisons are done in two scenes: the Bidir Room scene and the Ajar scene. The Bidir Room is a classic example of a scene where bidirectional path tracing works really well. The lights are more difficult to reach than in, for example, the Sponza scene, but they can still be reached from almost anywhere in the scene within two bounces. Figure 6.28 shows the results of the comparison in the Bidir Room scene.

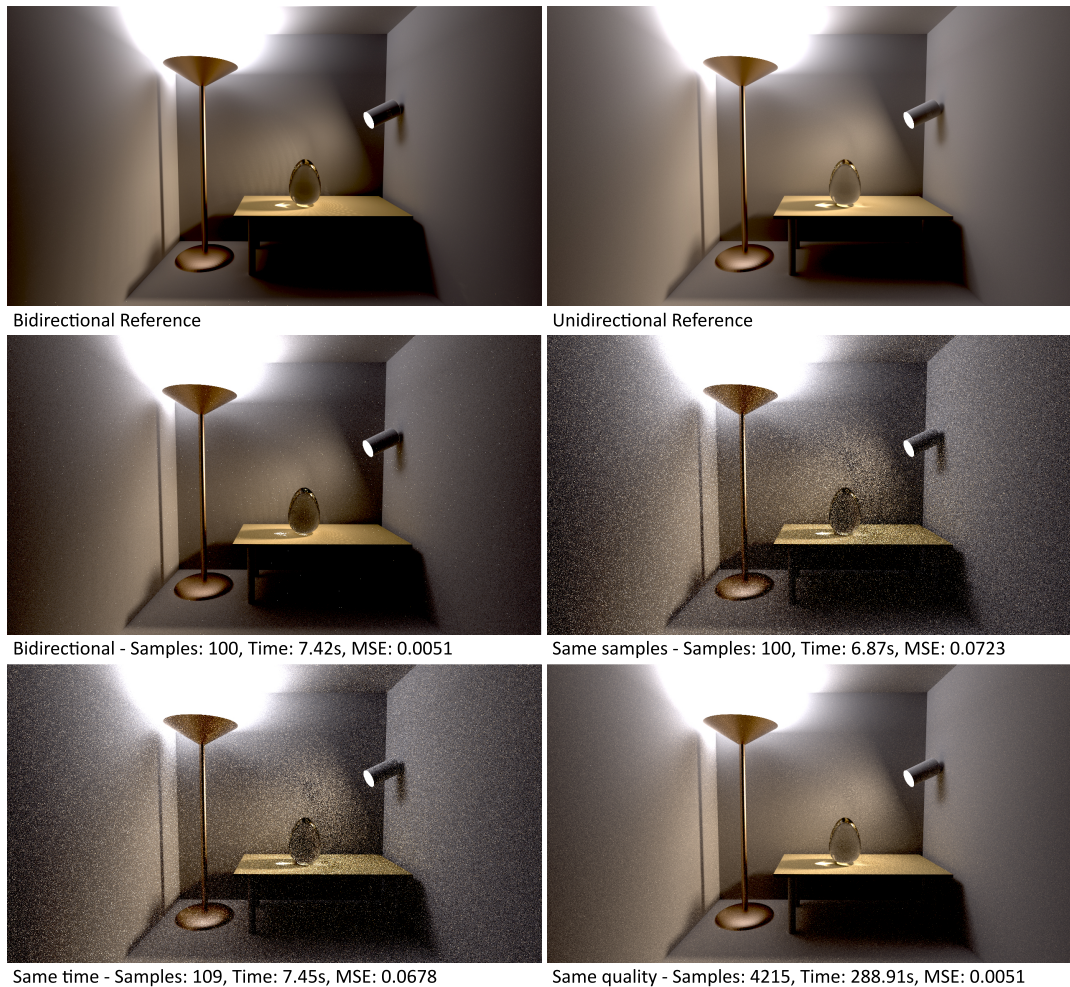


FIGURE 6.28: Comparison of results on the Bidir Room scene. Compares 100 samples of the bidirectional path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

In this scene the bidirectional path tracer outperforms the learning method on all fronts. The bidirectional path tracer creates an almost noiseless image in 100 samples, while the result of the learning method is far from converged. We do see that the bidirectional path tracer has more overhead costs than the learning method. In the same time the learning method can produce 9 more samples. However, the samples of the bidirectional path tracer are of much higher quality.

The Ajar scene is the scene where the learning method performs best. The light source is very hard to reach, but the learned radiance distributions is good enough to make the paths find the light source. Figure 6.29 shows the results of the comparison in the Ajar scene.

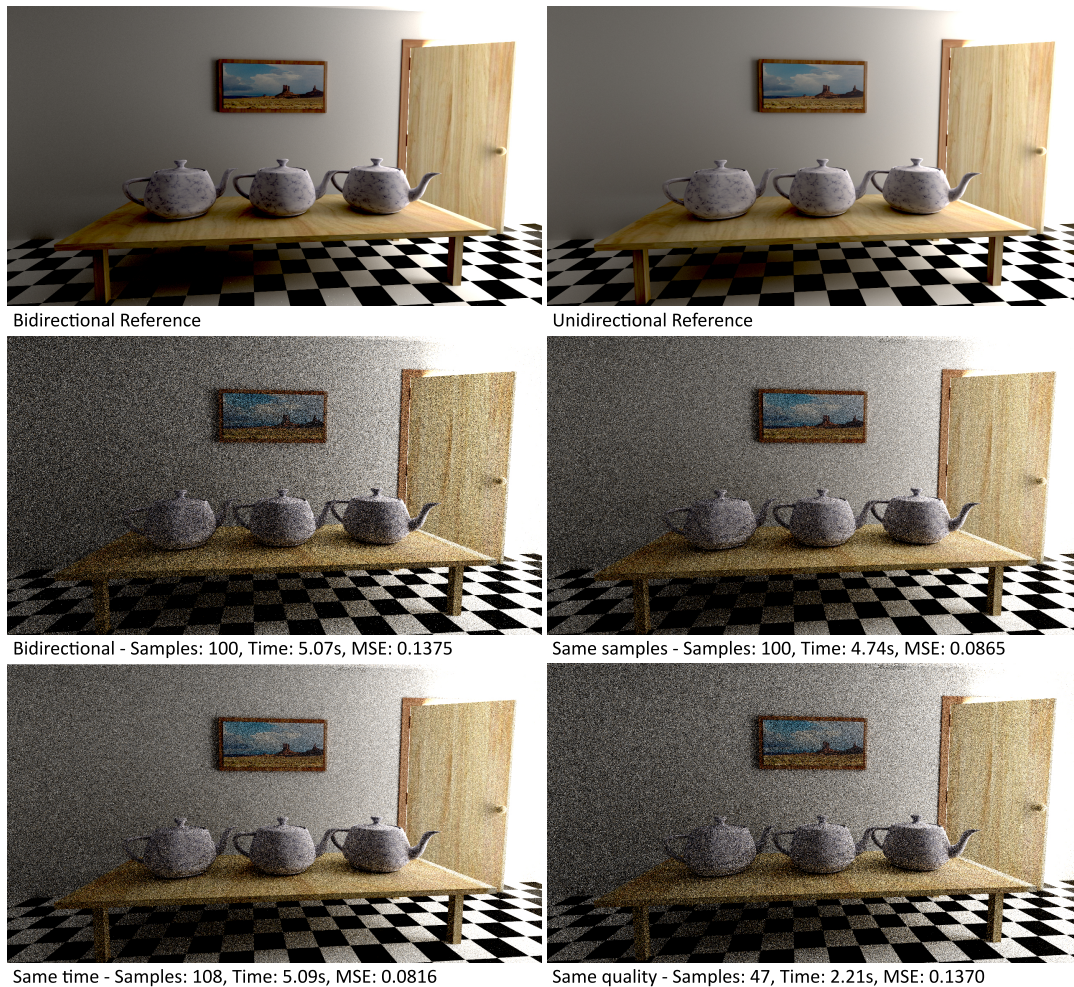


FIGURE 6.29: Comparison of results on the Ajar scene. Compares 100 samples of the bidirectional path tracer to three results of the learning method. First the result after the same number of samples, then the result after the same time and finally the result of the same quality (MSE) as the standard image.

Here we see that the learning method outperforms the bidirectional path tracer. In the same amount of samples the Q-learning method is capable of producing a much clearer image than the bidirectional path tracer. The increased overhead costs of the bidirectional path tracer allow the learning method to produce an even clearer image in the same time, as the quality of a single sample is higher and takes less time to calculate.

6.4 Performance

In this section we measure the performance of the method. The method consists of three parts: nearest neighbor search (NNS), updating the Q-table and sampling the Q-table. There are two parameters influencing the render time of the method: The number of points in the scene and the number of strata used to stratify the hemisphere. Figure 6.30 shows the influence of the number of points on each of the parts of the method.

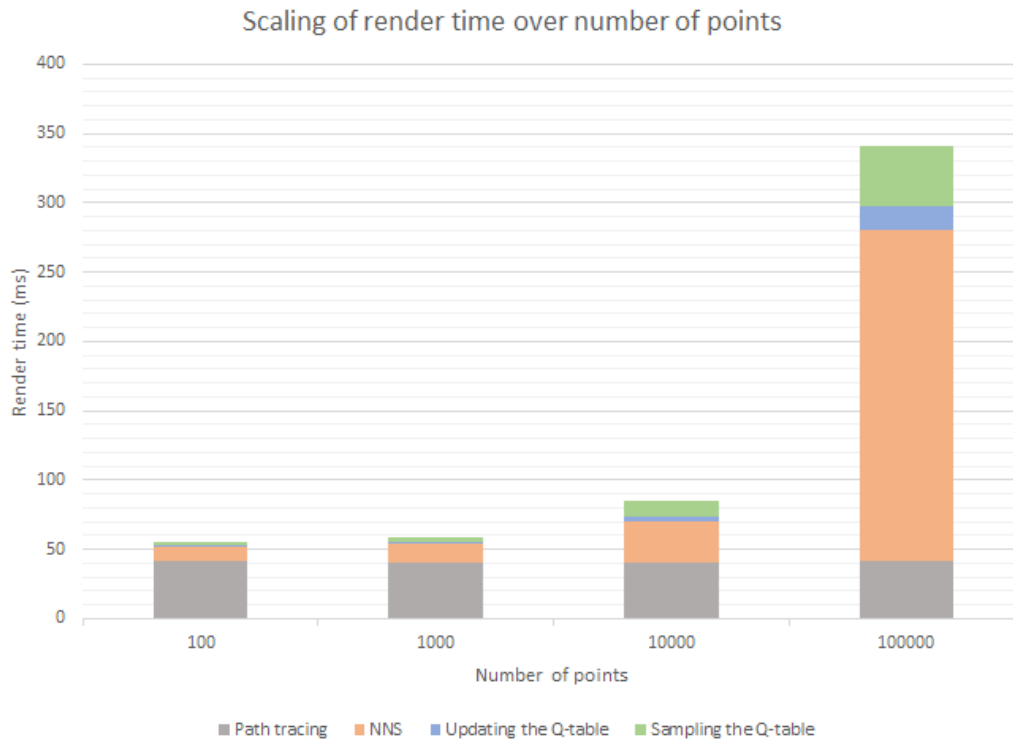


FIGURE 6.30: The division of render time over the number of points

We see that most of the time using the method is spend on the nearest neighbor search. The total render time seems to scale linearly with the number of points. Making the method not well suited for very large scenes.

6.4.1 Memory usage

Table 6.2 and figure 6.31 show the memory usage of the method. The memory depends mostly on the number of points in the scene and the number of strata we subdivide each hemisphere in. Since the number of strata used is the same for each test we only measured the difference in memory usage over the number of points. The resolution of the grid also influences the memory, although the influence is tiny compared to the size of the Q-tables.

TABLE 6.2: Memory usage of all GPU buffers used by the Q-learning method

Points	Memory usage (kB)						
	Grid	QMax	Point set	QTable	QCdf	QVisits	Total
100	48.0	0.4	3.2	56.0	56.0	56.0	171.2
200	48.0	0.8	6.4	112.0	112.0	112.0	342.4
400	48.0	1.6	12.8	224.0	224.0	224.0	684.8
800	48.0	3.2	25.6	448.0	448.0	448.0	1,369.6
1600	48.0	6.4	51.2	896.0	896.0	896.0	2,739.2
3200	48.0	12.8	102.4	1,792.0	1,792.0	1,792.0	5,478.4
6400	48.0	25.6	204.8	3,584.0	3,584.0	3,584.0	10,956.8
12800	48.0	51.2	409.6	7,168.0	7,168.0	7,168.0	21,913.6

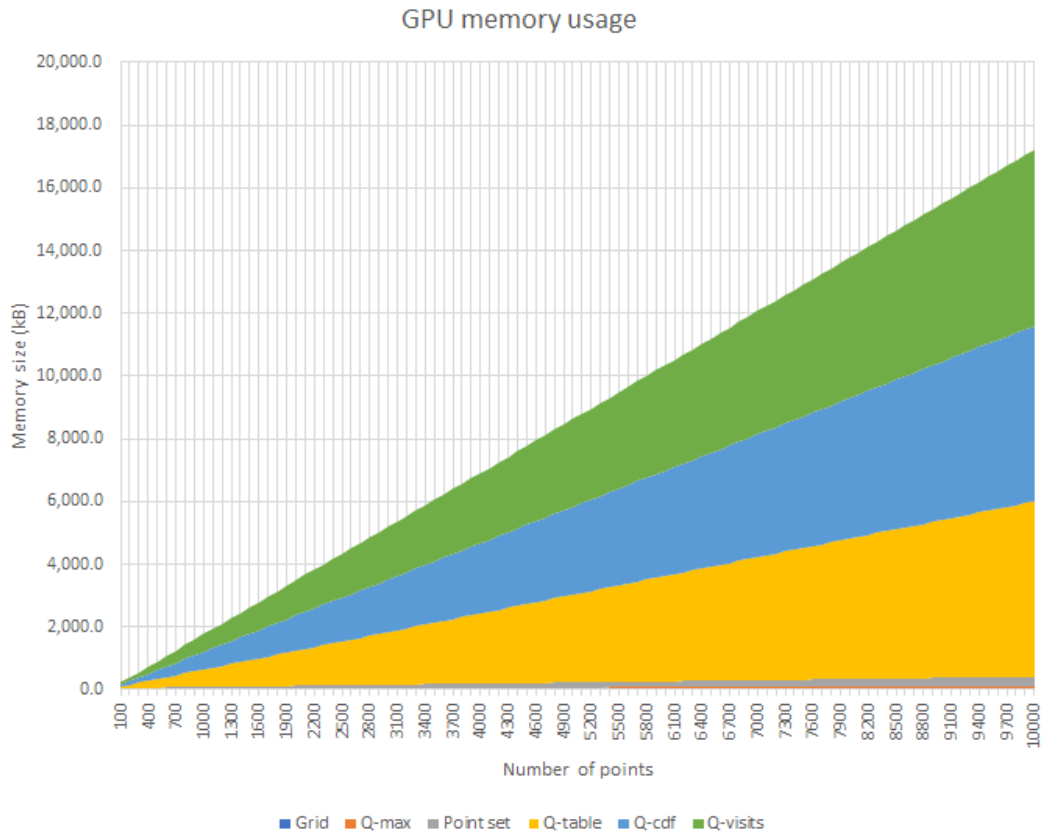


FIGURE 6.31: Development of the memory usage of all GPU buffers used by the Q-learning over the number of points

The memory also scales linearly with the number of points. While the total memory usage of the method easily fits in the RAM of any modern GPU it could still be beneficial to optimize the memory usage of the method.

6.4.2 Atomic operations

To update the Q-table on the GPU we use CUDA's atomic operations. This allows us to let multiple threads update the table at the same time without corrupting it. To measure the impact of the atomic operations on the performance we ran 1000 samples of the method on the Ajar scene and measured the running time with and without the atomic operations.

TABLE 6.3: Measurement of running time on the Ajar scene with and without atomic operations.

	With atomics	Without atomics
Running time (1000 samples)	73682 ms	73200 ms
Average running time per sample	73.7 ms	73.2 ms

As we can see in table 6.3 The difference in running time per frame is less than a millisecond, making the impact of the atomic operations negligible.

7 Discussion and Conclusions

In this chapter we first discuss the results of the experiments. Secondly we present several future research opportunities to extend our work. Finally we provide answers to the research questions and present a conclusion to this research.

7.1 Discussion

This section will first discuss the results of the experiments conducted on the Q-learned importance sampling method. Then we discuss the results of caching system experiments.

7.1.1 Q-learned Importance Sampling

In all scenes we observe that the percentage of rays returning energy is marginally increased by the Q-learning method. The goal of the Q-learning based importance sampling is to direct rays towards the light, so having more rays returning energy is a good indication that the method is effective.

We also see that in the Ajar scene and the Sponza scene the average number of bounces per ray decreases when learning is enabled, especially with Russian roulette and NEE disabled. The decrease in the number of bounces indicates that the method is able to find shorter paths to the light source(s) compared to regular path tracing. However, in the Bedroom scene and the Bidir Room scene the average number of bounces increases. The reason for this could be that since both scenes contain transmissive surfaces (e.g. the curtains and the glass egg) the learning method is directing more rays to these surfaces, which results in a larger number of bounces. In the Bidir Room the increase of the number of bounces is quite significant, even though the glass egg only covers a tiny portion of the scene. An explanation for this could be that in this scene there is no wall behind the camera, so rays traveling towards the camera will leave the scene without further bounces. Rays can leave the scene within a single bounce from almost any part of the scene, which leads to short rays even though they don't return energy. The learning method will prevent rays from traveling out of the scene as there is no energy to be gained that way. This leads to longer paths, which return more energy.

In each scene we see that the differences between learning with and without training are minimal. The reason for this is that the Q-learning rapidly finishes learning. In the graphs showing the percentage of valid paths we see that the method converges to the same level of the learning with training in about 50 samples. Because the method finishes learning so fast the learning without training only gets a minor head start at the beginning, after which both methods proceed in the same way.

The results show that a GPU implementation can be very effective. The synchronization requirements of Q-learning can be easily satisfied by separating the updating

of the Q-table from the sampling using different kernels and by using atomic operations to prevent two threads from updating the same entry in the table. The increase in the percentage of valid paths shows that the implementation is working as intended.

When Next Event Estimation and Russian roulette are disabled the effectiveness of the method increases marginally. Q-learning increases the percentage of valid paths by a factor of almost 48. The reason for this is that Russian roulette and importance sampling are trying to accomplish the same thing: a decrease of the number of low energy paths. Russian roulette does this by pruning low energy paths and importance sampling does this by sending paths in directions with that potentially return more energy.

With Russian roulette and NEE are enabled, the learning is less efficient, even though the overall performance improves. A possible reason for this is that through learning we can find paths that require several low energy bounces before finding a high energy light source (delayed reward). Russian roulette will try to prune these paths before they find the light source. When this happens the Q-learning method will learn that sending rays in that direction will lead to a low reward, when in reality a high reward could be found.

In the comparison of the method to the bidirectional path tracer we see that the learning method is not able to outperform the bidirectional path tracer in the Bidir Room scene. The learning method is not able to handle the light coming through the glass egg as well as the bidirectional path tracer, which is visible by the increased presence of 'fireflies' in the scene. In the Ajar scene the light sources are harder to reach than in the Bidir Room scene. The bidirectional path tracer is not able to handle the difficulty of this scene well and does not perform much better than the standard path tracer. The learning method clearly outperforms the bidirectional path tracer in this scene.

The learning method is not able to replace the bidirectional path tracer, but in some scenes it might be preferable. Another reason for choosing Q-learned importance sampling over bidirectional is that a bidirectional path tracer imposes strict requirements on the BRDF, which must be energy conserving. This is often hard to guarantee, e.g. when the BRDFs are user-definable. The learning method, when used with the unidirectional path tracer, has more relaxed requirements, which makes it an interesting alternative to bidirectional methods.

The results show that using Q-learning to create importance distributions can be beneficial in a GPU based path tracer. While the updating and the sampling of the Q-table increases the time to render a single frame, the overall efficiency of the system improves for most scenes. However, the Sponza results show that in scenes where the light is easily reachable the learning becomes much less useful and may even decrease overall efficiency. This indicates that using Q-learning is only beneficial in scenes that are hard to render for a standard path tracer.

7.1.2 Caching Scheme

When testing the caching schemes without considering running times we see that although the differences between each type of caching system are quite insignificant, the photon map almost always performs best. We do see that the Poisson disk set usually results in a set that is similar to the photon map. Either by picking a maximum distance close to the minimum distance or because of the properties of the scene (e.g.

the light being hard to reach in most of the scene leading to a high light occlusion value everywhere).

In the 5 second tests the results vary more per set. However, the photon map still outperforms the other two sets by a small margin. The photon map is also preferable because it is easier to calculate, there is no need to calculate the ambient occlusion or light occlusion.

The parameter combinations of the Poisson disk set now have a greater influence on the resulting MSE values. A larger distance between the minimum and the maximum distance has a negative influence on the MSE.

A reason for the performance of the photon map could be that the distribution of points in the photon map is more uniform. Since the points are more evenly distributed over the scene each thread will have roughly the same workload, which leads to lower *flow divergence*.

The ambient occlusion value leads to more points in parts of the scene with more complex geometry. Results show that this increase in points in these regions does not necessarily lead to less noise. A reason for this could be that more points in a small region will cause each point to be updated less often, making each distribution less accurate. A single accurate distribution has a greater influence on the MSE than several less accurate distributions.

The light occlusion value increases the number of points in parts where the light is hard to reach. However, the method works best in parts of the scene where the light is almost unreachable. In these parts the light occlusion value will always be at its maximum. This means that the rejection distance will be closer to the minimum distance than when light occlusion is disabled.

7.2 Future Work

The results of this thesis show that the Q-learning based importance sampling technique can be very effective. The next logical step for this implementation would be to optimize it further. The performance results show that a large part of the overhead costs is spend on searching the grid. Finding some way to combine the acceleration structure of the path tracing with the acceleration structure of the point set should be able to marginally speed up the process.

In this thesis we compared three caching systems. We hand picked the set of parameters that we tested the systems on. Since there are only two parameters, it should be possible to write a system that automatically determines the best pair. A hill-climbing algorithm, such as *simulated annealing*, would be able to quickly find a good working set of parameters in every scene. This only has to be done once for every scene, storing the two resulting parameters for re-use.

More experimentation should be done with the different caching techniques to really find out which technique best fits this method. Our tests are not enough to claim that either one of these systems is the most ideally suited for the learning method, but the results show that experimenting with point sets can influence the results of the method. The other caching systems detailed in section 3.1 could be tested or an

entirely new system specifically designed for this method could be created.

We've seen that the Q-learning method is able to outperform a bidirectional path tracer in certain scenes. However, further tests should be conducted to show exactly which characteristics of a scene favor a bidirectional path tracer and which parts favor the learning method. It would be useful if there was a way of automatically determining which method to use on each scene.

On top of that it should be possible to integrate the learning method in a bidirectional path tracer. Two Q-tables could be used to guide both the rays from the light sources and the rays from the camera.

7.3 Conclusions

In this thesis we have extensively tested a GPU implementation of Dahm and Kellers Q-learning based importance sampling method [6]. The method was tested on four different scenes with three different kinds of caching schemes. We have tested the efficiency of the method over the number of samples and over time by comparing the resulting frame with a perfect reference image. On top of that we measured the percentage of paths returning energy, the average number of bounces per ray and the average energy per bounce on each scene. The results of these tests are sufficient to adequately answer the research question of this thesis and its sub-questions.

The characteristic of the Q-learning method vary per scenario. We see that the difficulty of the scene has a large impact on the effectiveness of the method; in scenes where the light sources are hard to reach the method will make a larger impact than on scenes where the light is easy to find. There is no point in directing paths towards the light if they would have found it anyway.

The Q-learning method usually leads to a higher number of rays returning energy. In most cases these rays bounce less, unless there are transmissive surfaces involved or rays can leave the scene without finding energy.

Next event estimation and Russian roulette have a negative influence on the performance increase of the method. However, the combination of NEE and Russian roulette with the learning method still leads to better results than just the learning method without NEE and Russian roulette.

The results show that a GPU implementation of the method can be just as effective as a CPU implementation. The synchronization requirements of the Q-learning method can be satisfied without sacrificing any performance. Since path tracing on the GPU is more effective in general, reaching the same efficiency levels would make the GPU implementation preferable over the CPU implementation.

We also showed that the method is capable of outperforming a bidirectional path tracer, although only in certain scenes.

We tested three different caching schemes and found that the photon map performs best. However, differences between the three sets were too small to conclude that the photon map is always the best of the three. On top of more different caching schemes need to be tested to actually find out what scheme is best suited for the Q-learning method.

With these findings we can answer the main research question of this thesis: an implementation of the Q-learning importance sampling technique can improve the efficiency of a GPU path tracer. The method will almost always produce higher quality samples, although the difference in quality is not always large enough to make up for the increased render time. In scenes where light sources are hard to reach the method performs outstandingly and can produce an image of the same quality of a regular path tracer in less than half the time.

7.4 Contributions

The contributions of this thesis are as follows:

- We present a GPU implementation of Dahm and Kellers Q-learning based importance sampling technique [6]. The method is implemented in a highly optimized GPU based path tracer so that results can be used outside of research applications.
- Three different caching schemes are used and tested to cache the Q-distributions in the scene. We provide insight in the influence of a caching scheme on the effectiveness of the method.
- The method itself is extensively tested on four scenes that were all designed to test a different aspect of the method. These tests show the consistency of the method and its characteristics.

Bibliography

- [1] AILA, T., AND LAINE, S. Understanding the efficiency of ray traversal on gpus. In *Proceedings of the conference on high performance graphics 2009* (2009), ACM, pp. 145–149.
- [2] BARZILAY, R., AND LAPATA, M. Collective content selection for concept-to-text generation. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing* (2005), Association for Computational Linguistics, pp. 331–338.
- [3] BASHFORD-ROGERS, T., DEBATTISTA, K., AND CHALMERS, A. A significance cache for accelerating global illumination. In *Computer Graphics Forum* (2012), vol. 31, Wiley Online Library, pp. 1837–1851.
- [4] BIKKER, J., AND REIJERSE, R. A precalculated point set for caching shading information.
- [5] CAMPBELL, M., HOANE, A. J., AND HSU, F.-H. Deep blue. *Artificial intelligence* 134, 1-2 (2002), 57–83.
- [6] DAHM, K., AND KELLER, A. Learning light transport the reinforced way. *arXiv preprint arXiv:1701.07403* (2017).
- [7] FROLOV, V., VOSTRYAKOV, K., KHARLAMOV, A., AND GALAKTIONOV, V. Irradiance cache for a gpu ray tracer. In *Proceedings of 22-th International Conference on Computer Graphics and Vision Graphicon-2012* (2012).
- [8] GANESTAM, P., BARRINGER, R., DOGGETT, M., AND AKENINE-MÖLLER, T. Bonsai: rapid bounding volume hierarchy generation using mini trees.
- [9] GAUTRON, P., KŘIVÁNEK, J., BOUATOUCH, K., AND PATTANAİK, S. Radiance cache splatting: A gpu-friendly global illumination algorithm. In *ACM SIGGRAPH 2005 Sketches* (2005), ACM, p. 36.
- [10] HERHOLZ, S., ELEK, O., VORBA, J., LENSCH, H., AND KŘIVÁNEK, J. Product importance sampling for light transport path guiding. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 67–77.
- [11] JENSEN, H. W. Importance driven path tracing using the photon map. *Rendering Techniques 95* (1995), 326–335.
- [12] JENSEN, H. W., AND CHRISTENSEN, N. J. Photon maps in bidirectional monte carlo ray tracing of complex objects. *Computers & Graphics* 19, 2 (1995), 215–224.
- [13] KAJIYA, J. T. The rendering equation. In *ACM Siggraph Computer Graphics* (1986), vol. 20, ACM, pp. 143–150.

- [14] KAY, T. L., AND KAJIYA, J. T. Ray tracing complex scenes. In *ACM SIGGRAPH computer graphics* (1986), vol. 20, ACM, pp. 269–278.
- [15] LAFORTUNE, E. P., FOO, S.-C., TORRANCE, K. E., AND GREENBERG, D. P. Non-linear approximation of reflectance functions. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 117–126.
- [16] LAFORTUNE, E. P., AND WILLEMS, Y. D. Bi-directional path tracing.
- [17] LAINE, S., KARRAS, T., AND AILA, T. Megakernels considered harmful: Wavefront path tracing on gpus. In *Proceedings of the 5th High-Performance Graphics Conference* (2013), ACM, pp. 137–143.
- [18] MCCOOL, M. D., AND HARWOOD, P. K. Probability trees. In *Graphics Interface* (1997), vol. 97, pp. 37–46.
- [19] MÜLLER, T., GROSS, M., AND NOVÁK, J. Practical path guiding for efficient light-transport simulation. In *Proceedings of the Eurographics Symposium on Rendering* (June 2017).
- [20] PURCELL, T. J., DONNER, C., CAMMARANO, M., JENSEN, H. W., AND HANRAHAN, P. Photon mapping on programmable graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2003), Eurographics Association, pp. 41–50.
- [21] RUNNALLS, A. R. Kullback-leibler approach to gaussian mixture reduction. *IEEE Transactions on Aerospace and Electronic Systems* 43, 3 (2007).
- [22] SOLOMONOFF, R. J. An inductive inference machine. In *IRE Convention Record, Section on Information Theory* (1957), vol. 2, pp. 56–62.
- [23] VEACH, E. *Robust monte carlo methods for light transport simulation*. Stanford University Stanford, 1998.
- [24] VORBA, J., KARLÍK, O., ŠIK, M., RITSCHER, T., AND KŘIVÁNEK, J. On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 101.
- [25] WARD, G. J., RUBINSTEIN, F. M., AND CLEAR, R. D. A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics* 22, 4 (1988), 85–92.
- [26] WATKINS, C. J., AND DAYAN, P. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
- [27] WATKINS, C. J. C. H. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.
- [28] ZHUKOV, S., IONES, A., AND KRONIN, G. An ambient light illumination model. In *Rendering Techniques' 98*. Springer, 1998, pp. 45–55.