Dissertation Type: research

DEPARTMENT OF COMPUTER SCIENCE

Learning the incident radiance for a continuous state space rather than a discrete one is more beneficial for Importance Sampling in Monte Carlo Path Tracing

Callum Pearce

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Friday 10th May, 2019

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Callum Pearce, Friday 10th May, 2019

Contents

Executive Summary

In the field of Computer Graphics, Monte Carlo path tracing is an algorithm which is capable of rendering photo-realistic images. Traditionally Monte Carlo path tracing has been thought to trade quick render times for superior image quality. This is due to the lengthy process of accurately finding each pixel's colour using Monte Carlo integration. At a high level, many light rays are traced from a camera, through each pixel and into scene, where each ray intersects with a point on a surface. Then, a new direction is sampled for each ray to continue their path in, where they intersect with another surface. A direction to continue the path of a ray in is continually sampled until it intersects with a light source. Each one of these rays traced through a pixel, around the scene and to a light source are a sample used to solve an integral by Monte Carlo integration which determines the pixel's colour value.

As path tracing computes each pixel value in an image by Monte Carlo integration, importance sampling can be applied, whereby directions which are considered to be more 'important' for determining a pixel's colour value have a higher probability of being sampled. Therefore, if we have a good idea of which directions are more important to continue rays in, each one of our sampled rays per pixel will contribute more to the pixel's value, meaning fewer samples will be required to accurately approximate the pixels colour value via Monte Carlo integration. In other words, the noise present in each pixel's colour value will be lower by using importance sampling with the same number of samples per pixel than without. If the noise in each pixel's colour estimate is lower, then the noise across the rendered image by path tracing will be lower, which ultimately improves the quality of the image at the same computational cost.

In order to determine which directions are more important to continue a ray in at a given intersection point, a reinforcement learning method, specifically temporal difference learning method has been used in a new path tracing algorithm to learn what is known as the incident radiance function. The approximation of this function made by the algorithm is used to importance sample directions to continue light paths in, which successfully reduces image noise in Monte Carlo path tracing. However, the algorithm approximates the function at a discrete set of locations within the scene, where in actual fact the scene is a continuous 3-dimensional space. Therefore, we have tested the following research hypothesis:

> Learning the incident radiance function for the continuous set of locations in a scene is more advantageous for importance sampling in Monte Carlo path tracing, compared to learning the function for a discrete set of locations.

In doing so, our main areas of work include:

- Built a path tracing engine from scratch which supports all path tracing algorithms we will introduce and the code for assessing them. This sums to over 7000 lines of C++/CUDA C code.
- Spent 50 hours researching into the field of efficient light transport simulation for path tracing techniques.
- Spent 130 hours researching into Reinforcement Learning and Deep Reinforcement Learning.
- Implemented the state of the art path tracing algorithm proposed by NVIDIA in [20] in which the incident radiance function is approximated for a discrete set of locations in a scene.
- Designed and implemented our newly proposed Neural-Q path tracer which approximates the incident radiance function for the continuous set of locations in a scene.
- Assessed the performance of the newly designed Neural-Q path tracer against NVIDIA's path tracer for various metrics, including the accuracy of the approximated incident radiance function.

Supporting Technologies

1. The `SDL2` library was used for displaying and saving rendered images from my Path tracing engine.
2. The `OpenGL` mathematics library was used to support low level operations in my Path tracing engine. It includes GPU accelerated implementations for all of its functions.
3. `CUDA Toolkit 10.1` parallel computing platform was used for accelerating Path tracing algorithms. This means the `CUDA nvcc` compiler must be used to compile our Path tracing engine.
4. All experiments were run on a desktop machine with an NVIDIA `1070Ti` GPU, Intel `i5-8600K` CPU and 16GB of RAM.
5. We used the C++ API for the `Dynet` neural network framework to implement all of our neural network code, as it is able to be compiled by the `CUDA` compiler.
6. All graphs created by our own Python scripts using `Matplotlib` and `NumPy`.

Notation and Acronyms

Miscellaneous

GPU	:	Graphical Processing Unit
API	:	Application Programming Interface
RAM	:	Random Access Memory
MAPE	:	Mean Absolute Percentage Error

Monte Carlo Integration

MC	:	Monte Carlo
PDF	:	Probability Density Function
F	:	True solution to an integral
$\langle F^N \rangle$	:	Approximate solution to the integral using N samples
$pdf(x_i)$	:	Probability density function evaluated at x_i
$\mathbb{E}[X]$	:	Expectation of a random variable X, $\mathbb{E}[X] = \sum_{x \in X} p(x)x$
σ_N^2	:	Variance of the N sampled solutions
μ_N	:	Mean of the N sampled solutions
$Var(\langle F^N \rangle)$	:	Standard error of the approximation $\langle F^N \rangle$
L_o^N	:	Outgoing radiance estimate using N sampled light paths
ρ_i	:	PDF over reflected ray directions for position x_i evaluated at the angle of incidence ω_i

Monte Carlo Path Tracing

SPP	:	Sampled light paths Per Pixel
BRDF/f_r	:	Bidirectional Reflectance Distribution Function
L_o	:	Outgoing radiance function
L_e	:	Emitted radiance function
L_i	:	Incident radiance function
$\mathbf{n}$	:	Surface normal
x	:	Position, $x \in \mathbb{R}^3$
ω	:	Direction, $\omega \in \mathbb{R}^2$
Ω	:	Set of all incident directions in a hemisphere surrounding a point
$\cos(\theta_i)$	:	Cosine of the angle between the surface normal and a direction ω_i
h	:	Hit-point function, returning the closest intersection
m	:	Total number of discrete directions represented by the adaptive quadrature
N	:	Number of SPP
$\mathbf{v^x}$	:	Set of vertices converted into a coordinate system centred at point x
v_i	:	The ith vertex in the scene
$v_i^{\{x\}}$	:	The ith vertex in the scene converted into a coordinate system centred at point x
n		: Total number of vertices in the scene

Reinforcement Learning and Deep Reinforcement Learning

MDP	:	Markov Decision Process
TD learning	:	Temporal Difference learning
ANN	:	Artificial Neural Network
$\mathcal{S}$	:	Set of all possible states
$\mathcal{A}$	:	Set of all possible actions
S_t	:	The state at time step t
A_t	:	An action taken in time step t
R_t	:	Reward signal received at time step t
γ	:	Discount factor
s	:	A state
a	:	An action
p	:	Probability of receiving a reward r in state s' for choosing action a in state s
G_t	:	Return (cumulative discounted reward) following time step t
π	:	A policy (Reinforcement Learning) / The optimal policy (Deep Reinforcement Learning)
$q_\pi(s, a)$	:	Value function under policy π evaluated at state-action pair (s, a)
q_*	:	The optimal value function
Q	:	Approximate value function
α	:	Learning rate
θ	:	Parameter vector for an ANN
$J(\theta)$	:	Loss function evaluated at parameters θ
∇_θ	:	Derivative w.r.t θ
$\hat{q_\theta}$	:	Approximate value function modelled by an ANN with parameters θ
ϵ	:	Current value of the ϵ-greedy policy
δ	:	Decay rate of the ϵ-greedy policy

Acknowledgements

We want to thank Dr. Neill Campbell and Dr. Carl Henrik Ek for their ongoing support during the execution of this project.

Chapter 1

Introduction

The aim of this chapter is to give a tour of the concepts we will be working with and set out a clear motivation for why we believe our work is necessary. We will avoid delving into any technical depth here, the pre-requisite knowledge to understand our work is instead given in chapter 2. Our journey begins with an overview of Monte Carlo path tracing and how importance sampling is beneficial for this. Next, we introduced temporal difference learning as a branch of Reinforcement Learning and how this relates to light transport for path tracing. Finally, we give our wider motivation for our work, as well as our objectives and the significant challenges standing in the way of achieving them.

1.1 Monte Carlo Path Tracing for Light Transport Simulation

Monte Carlo path tracing, more generally known as path tracing, is a Monte Carlo method for rendering photo-realistic images of 3D scenes by accurately approximating global illumination [17]. Figure 1.1 summarises on a high level how path tracing produces a 2-dimensional image of a 3-dimensional scene. For each pixel multiple rays are shot from the camera through the pixel and into the scene. Any ray which intersects with an area light terminates, otherwise a new direction is sampled for the ray and it is continued in this sampled direction. This process is repeated until all rays have intersected with a surface which emits light, known as an *area light* to obtain a colour estimate. A pixel's colour value can then be calculated by averaging all ray colour estimates which were fired through that pixel. Each ray's colour estimate is calculated based on the material surface properties it intersects with before intersecting with the light, as well as the area lights properties it intersected with. The full path a ray takes from the camera to intersecting with an area light is known as a *light path*, and we will use this terminology from here on. Therefore, each sampled light path represents a sample used to solve an integral to determine a pixel's colour value by Monte Carlo integration. The more Sampled light paths Per Pixel (SPP), the more accurate the Monte Carlo approximation is of the integral to determine the pixels colour value. Meaning, the higher the SPP, the lower the noise in each pixel and in turn, the rendered image [35].

Path tracing is said to simulate *light transport* [37], meaning it simulates the interactions of photons with surfaces. This is achieved by the process described above where each light path can be thought of as a photon. However, light paths in path tracing begin from the camera and reflect off surfaces until until they intersect with a light source, whereas in reality photons begin from the light source and reflect off surfaces until they intersect with the camera lens. It turns out that simulating light paths in this way is identical to simulating photons how they naturally occur, because the physics behind light transport does not change if the paths are reversed [30]. This means path tracing is able to faithfully simulate light transport whilst only simulating light paths that intersect with the camera lens to contribute to the image.

As path tracing simulates light transport, it also simulates global illumination as it accounts for both direct and indirect illumination. Direct illumination light paths contribute to the image by reflecting off exactly one surface before intersecting with a light source, whereas indirect illumination is where light paths reflect 2 or more times before reaching a light source. In figure 1.2, an identical scene is shown with only direct illumination (left) and the other with global illumination (right). The globally illuminated scene displays a range of light effects, which are not present in the directly illuminated scene For example, effects such as; colour bleeding which is clear on the white walls by the boxes, soft shadows of the boxes silhouette, and indirect diffuse lighting causes the shadow of the box to not appear pitch black. These light effects are achieved by a simple process of determining each pixels colour value individually.

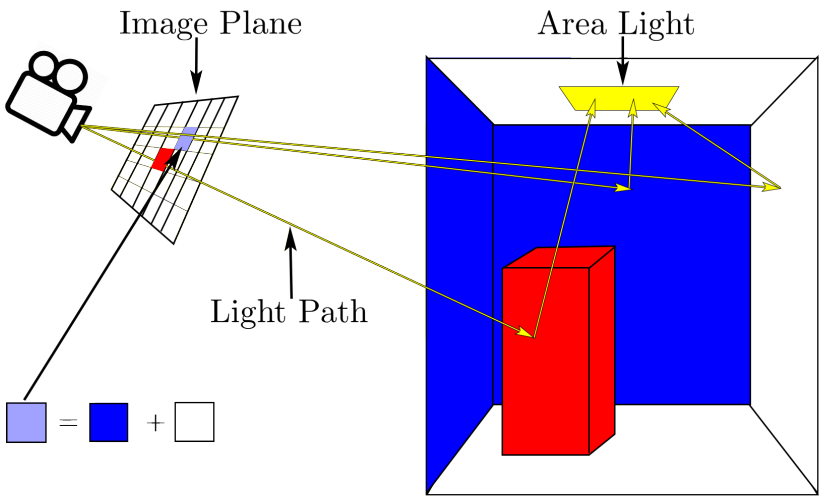
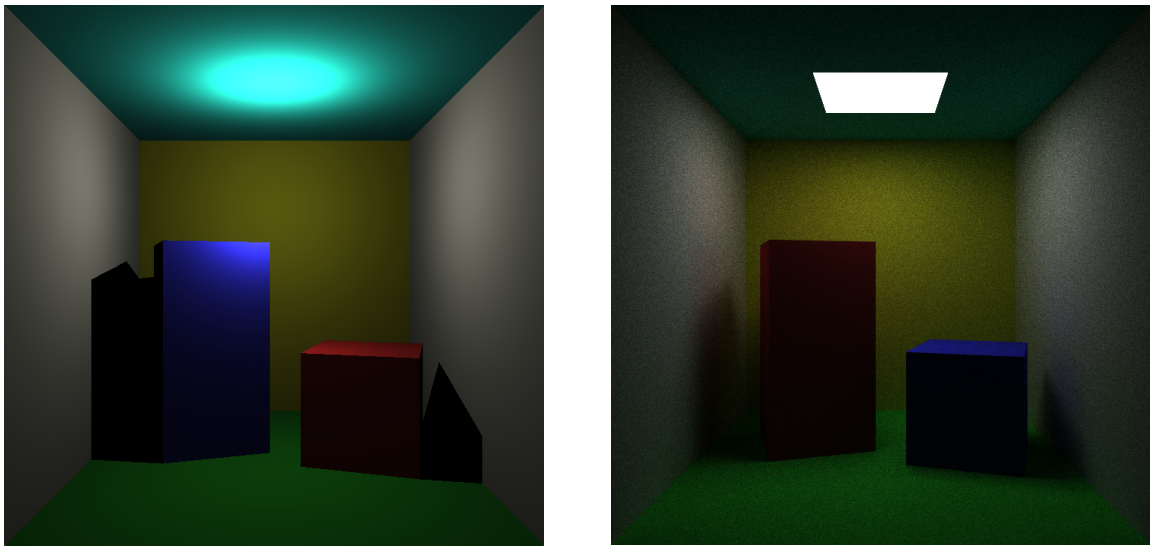


Figure 1.1: An illustration of path tracing, where three light paths are traced from the camera through a pixel, to the light source in a simple 3-dimensional scene. The light paths are used to determine the pixel colours of the rendered image.

This allows artists to increase productivity and perform less manual image tweaking in the production of photo-realistic images. Due to this, the Computer Graphics industry has seen a large resurgence in research and usage of path tracing and other rendering methods which simulate light trasnport in the past decade [41].



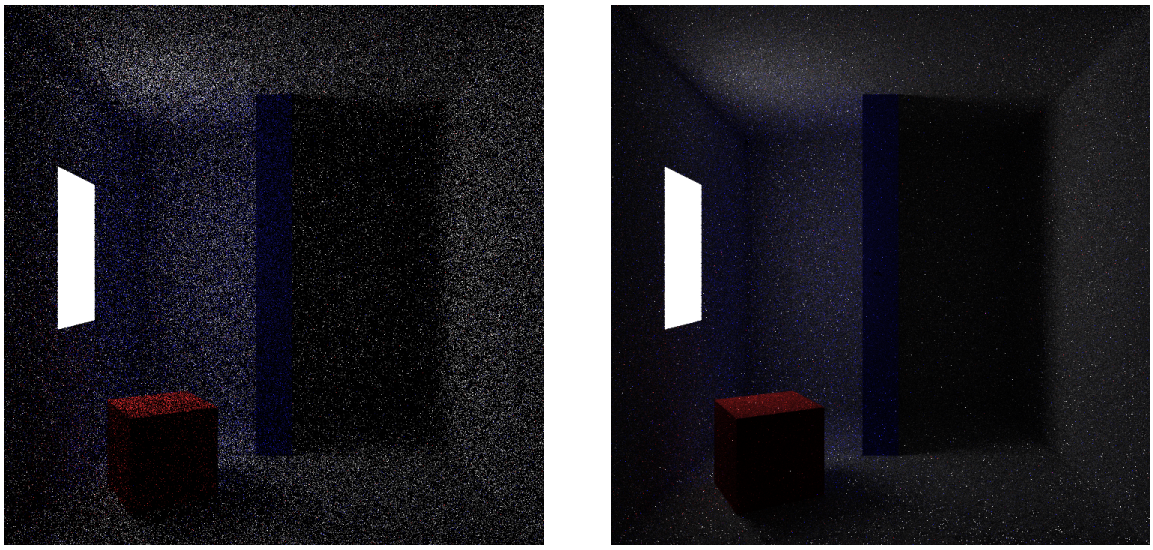
(a) Direct Illumination

(b) Global Illumination

Figure 1.2: Two renders of the Cornell Box, where the left is directly illuminated and the right is globally illuminated.

Importance sampling can be used in path tracing to improve the rendered image quality when using the same number of SPP. The reason being is that directions to continue light paths in for the default path tracing algorithm are sampled uniformly at random. However, some directions will lead to a light path contributing more to the approximated colour value of a pixel than others. These directions are said to be more 'important'. We wish to sample important directions more often as they will reduce the variance in our Monte Carlo approximation of a pixel's colour value, which is otherwise known as a reduction in noise. Therefore, by using importance sampling we can construct light paths in such a way that they are more likely to reduce the noise in each pixel's colour estimate, producing higher quality rendered

images using the same number of SPP. An example of this reduction in noise can be see in 1.3, where the default path tracers output is compared to our implementation of NVIDIA's path tracer [20], which uses Importance sampling. Note, any other rendering algorithm which simulates light transport can benefit from importance sampling, as they are all derived from the *rendering equation* [34, 37] which we will detail in chapter 2.



(a) Default Forward Path Tracer

(b) Expected Sarsa Path Tracer

Figure 1.3: Two renders of a simple room using 16 SPP. Where one does not use importance sampling in the construction of light paths (left), and the other does so based on a reinforcement learning rule [20] (right). A clear reduction in image noise can be seen by the use of importance sampling.

1.2 Temporal Difference Learning for Importance Sampling Ray Directions

We will be using temporal difference learning techniques to find out which directions are important to continue light paths in during path tracing. This section answers three important questions to detail our motivation behind doing this; a) what is temporal difference learning? b) How can temporal difference learning methods be used to importance sample new ray directions for a given intersection point in the scene? c) Why use temporal difference learning methods over other Importance sampling methods to do so?

1.2.1 What is Temporal Difference learning?

Temporal difference learning, which I will refer to from here on as TD-learning, is a set of model free Reinforcement learning methods. Firstly, Reinforcement learning is the process of an AI agent learning what is the best action to take in any given state of the system it exists within, in order to maximise a numerical reward signal [66]. The AI agent is not told which actions are best to take in a given state, but instead it must learn which ones are by trialling them and observing the reward signal. Actions taken may not only affect the immediate reward, but all subsequent rewards received for taking future actions. For example, picture a robot rover whose duty it is to explore the surrounding area as much possible. A state in this case is any sensory data the robot has, such as camera for observing its surroundings. Its possible actions are the directions to move in for a set distance. If it discovers a new area, it receives a positive reward signal, otherwise the reward signal is zero. Now, if the robot chooses to explore a given area it may not be able to get back from, say a canyon, the robot is limited to searching areas reachable from the canyon. Hence, all subsequent reward signals are limited to what can be received from exploration of the canyon, compared to not entering the canyon and exploring areas which can be returned from first.

1.2.2 Temporal Difference learning methods for Efficient Light Transport Simulation

As a small introduction to how reinforcement learning can be applied to light transport simulation, a state, action, and reward signal in the context of light transport simulation within path tracing are given below.

- **State**: A 3D intersection position of a light path in the scene.
- **Action**: Continuing a light path in a given direction from the current state.
- **Reward Signal**: The amount of light received by the next intersection location of the light path when continued in the direction sampled.

In this setting, we can use TD-learning methods to create an AI agent which learns by taking different actions in different states, which then observes their reward signals to find out for each state which actions have the highest value. By then converting the action space into a probability distribution weighted by each actions learned valuation, the AI agent can sample from this distribution to increase the average light path's contribution to a pixel, reducing the noise in rendered images.

1.2.3 Why use Temporal Difference Learning for Importance Sampling?

Traditional Importance sampling techniques for path tracing do not take into account the visibility of the object from light source [20]. A light blocker is shown in figure 1.4, where the blocking object stops rays from directly reaching the light. Due to the unknown presence of blockers, traditional importance sampling methods can fail to avoid sampling light paths which contribute nothing to the image being rendered. Therefore, scenes which are significantly affected by blockers will not receive the benefits from traditional Importance sampling and can even benefit more from uniformly sampling directions [58].

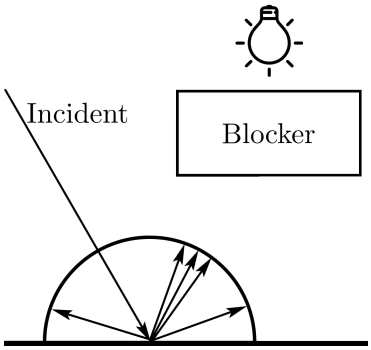


Figure 1.4: An illustration of a light blocker for an importance sampling scheme which does not consider visibility. Each arrow represents a possible direction the light path will be continued in. Clearly the reflected light path is likely to hit the blocker, reducing its contribution to the approximation of a pixel value.

Temporal difference learning methods are better equipped to tackle this problem [20]. The AI agent outlined in section 1.2.2 learns which directions contribute the highest power of light to a point, to concentrate its sampling in these directions. This means directions leading to blockers will have a low value, hence it is unlikely the AI agent will continue light paths in these directions. In fact, what we have described the agents goal to be is equivalent to learning what is known as the *incident radiance function*. The incident radiance function describes the power of light incident on a surface at a given point from a given direction.

1.2.4 From Discrete to a Continuous State Space

With it now conceptually clear how temporal difference learning can be used for importance sampling in Monte Carlo path tracing, we will introduce what our main area of work is for this thesis. As described in section 1.2.3, it is possible to construct an AI agent which learns the incident radiance function. This was first proposed by NVIDIA in [20], which when used in importance sampling for path tracing, it

significantly reduced the noise in rendered images. The incident radiance function was approximated at a discrete set of locations in the scene by using a temporal difference learning rule, the details of which will be given in chapter 2. However, the set of possible locations in a 3-dimensional scene are technically infinite. Think of the room which you are in now, there are infinitely many positions, this is the same for scenes in computer graphics. Therefore, we attempt to learn the incident radiance function for the continuous set of possible locations in the scene, by using a Deep Reinforcement Learning technique. The potential advantages this gives for importance sampling in path tracing compared to NVIDIA's algorithm include reduced image noise for complex scenes, easier hyperparameter tuning, and a significant reduction in memory usage.

1.3 Motivation

We have identified the overarching goal of our work in section 1.2.4. Which is to extend the approximation of the incident radiance function from a discrete set of locations to a continuous set, to benefit importance sampling for Monte Carlo path tracing in multiple ways. However, the timeliness and importance of our work in path tracing is yet to be explained, which is exactly what we will do here.

1.3.1 Real time Rendering using Accurate Light Transport Simulation

Our wider motivation for our work is contribute to the achievement of the long sought after goal of real-time ray-tracing. When we say ray-tracing we are specifically referring to methods which simulate light transport to render images, which includes path tracing. While our work here may not prove to directly reduce the render time for producing high quality images from path tracing, it does attempt to reduce the number SPP required to render these high quality images, in hope that future work will optimise our methodology for speed.

The excellent image quality from path tracing and other methods which simulate light transport for rendering has been well known since their conception [35, 28, 77, 34], as opposed to scanline renderers. Scanline renderers are the current 'go to' rendering algorithms for real-time rendering of computer graphically generated images due to their speed. However, they cannot simulate the light effects discussed in section 1.1 as easily or to such a high quality [75].

Furthermore, scanline rendering methods do not scale well with the number of polygons used to build a scenes surfaces, compared to that of methods like path tracing. Therefore, scanline rendering for scenes with extremely complex geometry in real-time is currently not an option. Rendering methods which simulate light transport therefore have great potential to be used in ultra realistic simulations for applications such as scenario planning and virtual reality learning environments [56]. Also, many games sell realism as one of their most important features. So, developing photo-realistic graphics in real-time has clear economic incentive for the video games industry which was valued at over $136bn by the end of 2018 [13]. An economic incentive can also be seen for the film industry, where reductions in render times will lead to a direct saving on compute time.

1.3.2 Recent Developments

Due to the incentives of real-time ray tracing, a number of different areas of research have emerged, besides importance sampling in light transport simulation alone. Purpose built hardware and deep learning post processing methods for reducing render times of these methods have received a large amount of research and investment recently. NVIDIA's Turing Ray Tracing Technology [55] represents a significant leap in hardware to support light transport simulation. It allows for real-time graphics engines to be a hybrid of both scanline rendering, and ray-tracing. The 20 series Turing GPU architecture has significantly improved the speed of ray-casting for light transport simulation. It has the capacity for simulating 10 Giga Rays per second. However, using this hardware alone with current rendering methods is not enough to perform accurate light transport simulation for complex scenes in real-time.

Post-processing methods are designed to take a noisy input image and reconstruct it, such that the presnce of noise is signficantly reduced. Generally, these methods rely on pre-trained deep neural networks to reconstruct the image far quicker than it would take for the renderer alone to produce an image of the same visual quality [8]. Once again NVIDIA has made significant advancements in this area with NVIDIA OptiX AI Accelerated Denoiser, which is based on their newly designed recurrent denoising autoencoder [15]. OptiX has been successfully integrated in to many of the top rendering engines which accurately simulate light transport, such as RenderMan [16] and Arnold [26]. Whilst post-processing has

significantly reduced the number of samples required to render photo-realistic images, there is still more work to be done to produce these images in real-time.

We believe the eventual progression of production path tracing engines will be to include both importance sampling and post processing for reducing the number of SPP required to render photo-realistic images. By using a path tracer equipped with these practices running on purpose built hardware for light transport simulation, real-time ray tracing may soon become a reality.

1.4 Objectives and Challenges

Once again, our primary goal is to approximate the incident radiance function for the set of continuous positions in a scene, which will be used for importance sampling light path directions in path tracing. To do so we have set the following ordered objectives for our work:

1. Implement NVIDIA's state of the art path tracer which learns the incident radiance function for a discrete set of locations in the scene for importance sampling in path tracing [20].

2. Devise a way to accurately learn the incident radiance function for the continuous set of locations in a scene by researching and using Deep Reinforcement Learning.

3. Research into the field of light transport simulation to design and implement an algorithm which uses the approximated incident radiance function to importance sample directions for continuing light paths in during path tracing.

4. Compare the newly designed algorithm against NVIDIA's to evaluate if it is beneficial in any way for importance sampling directions to continue light paths in during path tracing.

By achieving the objectives between (1-3) we will be able to test our research hypothesis during objective (4).

The objectives above require us to overcome some significant technical challenges during our work. Details of the path tracing algorithm proposed by NVIDIA are sparse, the single paper [20] is careful not to give away all implementation details, making it difficult to implement correctly. This is part of a more general problem with the field of light transport simulation, where a few large companies including NVIDIA and Disney dominate research in this area, and implementation details of their methods are very sparse. Objective (2) will require us to somehow use deep reinforcement learning to accurately approximate the incident radiance function, which has never been done before to the best of our knowledge. So we must find a way to make this possible before attempting to design and implement a new path tracing algorithm which uses this approximation. The algorithm itself will also require the use of existing data-structures to represent the incident radiance at a point, adding more complexity to the implementation. Finally, comparing NVIDIAs path tracer to our newly designed path tracer is no simple task. The incident radiance function is a 5-dimension function (the details of this will be discussed in chapter 2), making it difficult to assess the quality of its approximation across the scene. We will have to come up with new methods of comparing the quality of the approximation between NVIDIA's path tracer and ours.

Chapter 2

Monte Carlo Integration, Path Tracing, and Temporal Difference Learning

The goal of this section is to a give you as the reader a good understanding of the technical concepts which our work relies on. We begin with Monte Carlo integration, as the theory behind how a Monte Carlo approximation can be used to find the true value of an integral forms the mathematical basis of the path tracing algorithm. We then give a full description of the path tracing algorithm and how it relates to the *rendering equation*, as well as pseudo code which we extend upon in chapter 3. Finally, we provide a short introduction to Reinforcement Learning and with it, temporal difference learning. This is the foundation of how both we and NVIDIA learn the incident radiance function, but the extension of how this applies to path tracing is saved until chapter 3.

2.1 Monte Carlo Integration and Importance Sampling

The theory of importance sampling for Monte Carlo integration underpins how noise in images rendered by path tracing can be reduced when using the same number of SPP. Therefore, it is necessary to have a good understanding of Monte Carlo integration and its properties, as well as importance sampling before applying it to path tracing.

2.1.1 Monte Carlo Integration

Monte Carlo Integration is a technique to estimate the value of an integral, equation 2.1 represents this integral for a one-dimensional function f between two points a, b.

$$F = \int_a^b f(x)dx \tag{2.1}$$

Monte Carlo integration is used to approximate an integral by uniformly sampling points (x_i) to evaluate the integral. These solutions to the integral are essentially averaged to find an approximation to the integral, or in other words, to find the area beneath the curve formed by the function (f) within the integral. More formally, basic Monte Carlo integration approximates a solution to an integral using the numerical solution in equation 2.2. Where $\langle F^N \rangle$ is the approximation of the value of the integral F, using N samples and x_i is a sample point [47].

$$\langle F^N \rangle = (b-a)\frac{1}{N}\sum_{i=0}^{N-1} f(x_i) \tag{2.2}$$

$$\langle F^N \rangle = \frac{1}{N}\sum_{i=0}^{N-1} \frac{f(x_i)}{\frac{1}{(b-a)}} = \frac{1}{N}\sum_{i=0}^{N-1} \frac{f(x_i)}{pdf(x_i)} \tag{2.3}$$

An important property of Monte Carlo Integration is that it produces an unbiased estimate of an integral, meaning the expectation of $\langle F^N \rangle$ is exactly the true value of the integral, F for any N [47].

This is presented in equation 2.4, where p_i is the probability of a given approximation $\langle F^N \rangle$. Basic Monte Carlo integration only produces a non-bias estimate when sample points x_i are randomly sampled from a uniform distribution. To extend this to *Generalized Monte Carlo* integration, where sample points may be sampled from any distribution, the function evaluated at point x_i must be divided by the Probability Density Function (PDF) over sample points evaluated at x_i ($pdf(x_i)$). Generalized Monte Carlo integration is shown in equation 2.3, which from here onwards we will refer to as MC integration. Dividing by the PDF ensures the estimate $\langle F^N \rangle$ is unbiased, as areas of the PDF with a high value will be sampled far more, but their contribution weighting ($\frac{1}{pdf(x_i)}$) to final estimate will be lower. Whereas areas of the PDF with a low value will be sampled less, but their contribution weighting to the final estimate will be higher to offset this.

$$\mathbf{E}[\langle F^N \rangle] = \sum_{i=0}^{k-1} \langle F^N \rangle_i * p_i = F \tag{2.4}$$

Another important property of MC integration is that by the law of large numbers, as the number of samples (N) approaches infinity, the probability of the MC approximation ($\langle F^N \rangle$) being equal to the true value of the integral (F) converges to 1. This law is stated in equation 2.5. By this property, MC Integration works well for multidimensional functions, as the convergence rate of the approximation is independent of the number of dimensions. Instead, it is only based on the number of samples used in the approximation (N). Whereas this is not the case for deterministic approximation methods, meaning they suffer from what is known as the curse of dimensionality [10]. For path tracing, the integral which is approximated is a 5 dimensional function, hence MC integration is used. This is further described in section 2.2.

$$Pr(\lim_{N \to \infty} \langle F^N \rangle = F) = 1 \tag{2.5}$$

The standard error of the MC integration approximation decreases according to Equation 2.7. Where the standard error describes the statistical accuracy of the MC approximation. Where σ_N^2 is the variance of the solutions for the samples taken, and is calculated by equation 2.6 using the mean of the solutions for the samples taken (μ_N). Due to equation 2.7, it takes four times as many samples to reduce the error of the MC approximation by a half. Also, the square root of the variance is equal to the error of the approximation. Therefore, from here on when we refer to reducing the variance, we are also implying a reduction in the error of the approximation.

$$\sigma_N^2 = \frac{1}{N-1} \sum_{i=0}^{N} (f(x_i) - \mu_N)^2 \tag{2.6}$$

$$\text{Standard Error} = \sqrt{Var(\langle F^N \rangle)} = \sqrt{\frac{\sigma_N^2}{N}} = \frac{\sigma_N}{\sqrt{N}} \tag{2.7}$$

2.1.2 Importance Sampling for Reducing Approximation Variance

Currently we have only discussed Monte Carlo integration by sampling points x_i to solve the integral using a uniform distribution. However, the purpose of introducing equation 2.3 was to create a custom *pdf* which can be used for importance sampling to reduce the variance of the MC approximation. To understand how and why importance sampling works, first observe figure 2.1 where a constant function is given with a single sample point evaluated for some function $f(x)$. This single sample is enough to find the true value for the area beneath the curve i.e. integrate the function with respect to x. This is shown in equation 2.8, where $p \in \mathbb{R}$ is a constant which can be substituted for $f(x)$.

$$\begin{aligned} \langle F^N \rangle &= (b-a)\frac{1}{N} \sum_{i=0}^{N-1} f(x_i) \\ &= (b-a)\frac{1}{N} \sum_{i=0}^{N-1} p \\ &= pb - pa \end{aligned} \tag{2.8}$$

However, figure 2.2 requires many samples to accurately approximate the integral when sampling from a uniform distribution. This is due to the functions complex shape, meaning many samples are required to approximate the area beneath the curve within the MC approximation. Generally, it requires fewer samples to approximate a function which is closer to being constant function [47].

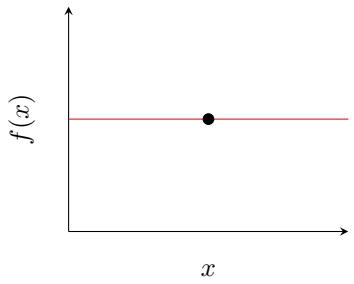


Figure 2.1: Constant function with a sample point

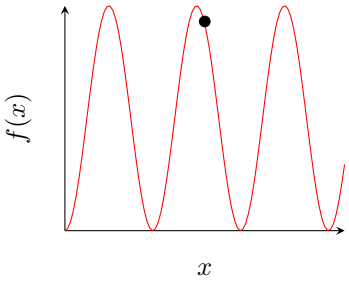


Figure 2.2: Non-linear function with a sample point

Most functions are not constant, however it is possible to turn any function into a constant one. This is exactly what can be done within MC integration. To convert a function f to a constant function, a function f' can be introduced which produces the same output as f for every input, but scaled by a constant c [62]. The function f is then divided by f' to produce a constant function, as shown in equation 2.9.

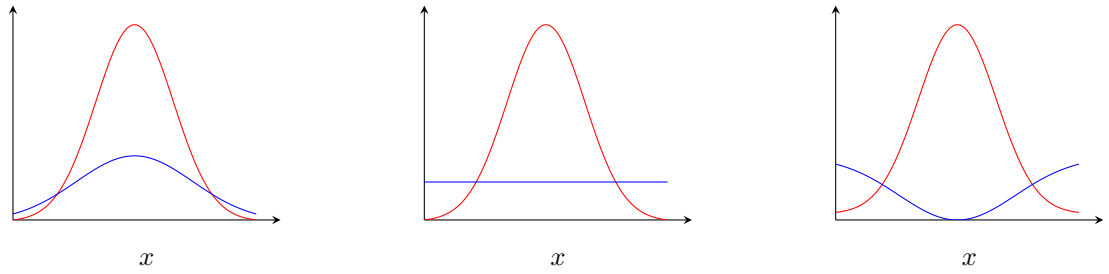
$$\frac{f(x)}{f'(x)} = \frac{1}{c} \tag{2.9}$$

This can be applied to MC integration stated in equation 2.3, by choosing a PDF which produces the same output as f for all inputs, but divided by some normalizing constant factor c to ensure the PDF is a probability distribution. Therefore, we are able to calculate the true value of the integral through MC integration as shown in equation 2.10. Where it turns out $\frac{1}{c}$ is the true value for the integral in equation 2.1.

$$\begin{aligned} \langle F^N \rangle &= \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(x)}{pdf(x)} \\ &= \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(x)}{cf(x)} \\ &= \frac{1}{N} \sum_{i=0}^{N-1} \frac{1}{c} \\ &= \frac{1}{c} \end{aligned} \tag{2.10}$$

For most cases, it's not possible to know the correct PDF to sample from which can normalize the function being integrated. However, if one has prior knowledge regarding 'important' regions of the functions input space, it is possible to create a PDF whose shape matches the functions more closely than a uniform PDF. By Important areas of the function input space, we mean areas of the input space which contribute a large amount to the integral of the function.

Figure 2.3a represents a PDF which has a similar shape to the function which is being integrated. Therefore, the variance in the MC integration approximation will be lower then that of the uniform distribution shown in figure 2.3b. Figure 2.3c presents an example where the created probability density function does not resemble the shape of the function which is being integrated. Using this PDF for MC integration would significantly increase the variance in the approximation compared to that from a uniform PDF in figure 2.3b. This is due to regions which have high importance according to the PDF actually contribute a low amount to the integral of the function f, causing the rise in variance of the MC approximation.



(a) Importance sampling reducing variance

(b) Uniform sampling

(c) Importance sampling increasing variance

Figure 2.3: Graphical representation of a function $f(x)$ (red) and the corresponding PDF $pdf(x)$ (blue) used in the MC integration approximation for the integral of $f(x)$.

2.2 Monte Carlo Path Tracing

In 1986 James Kajiya introduced the rendering equation and with it, a MC integration approximation to the equation [35]. This MC approximation is what is known as today as Monte Carlo path tracing which we refer to as path tracing. Here, we will give a detailed explanation of the rendering equation, what is represents, and how Monte Carlo Path Tracing approximates the equation by light transport simulation. As path tracing involves MC integration, importance sampling can be used to reduce the variance in its approximation, as described in section 2.1.2.

2.2.1 The Rendering Equation

Equation 2.11 is the rendering equation. It calculates the outgoing radiance from a point x in a direction ω. Radiance indicates the power of light emitted, transmitted, reflected or received by a surface from a given direction. The units of radiance are watts per steradian per square metre ($W \cdot \{sr\}^{-1} \cdot m^{-2}$). Therefore, by placing a camera in a scene, the radiance incident on the lens from a given surface determines the cameras perceived colour and power of light incident from the surface. These values are used to calculate pixel values in computer image generation. The rendering equation states how to correctly perform light transport simulation for rendering, and in turn how to accurately simulate global illumination. Therefore, methods which can accurately approximate the rendering equation for any given scene, are able to convert the incident radiance into pixel values. In turn, with enough SPP these pixel values form a high quality computer graphically generated image. The exact details of this process will be described in section 2.2.2.

$$\underbrace{L_o(x,\omega)}_{\text{Outgoing}} = \underbrace{L_e(x,\omega)}_{\text{Emitted}} + \underbrace{\int_\Omega L_i(x,\omega_i) \cdot f_r(\omega_i,x,\omega) \cdot \cos(\theta_i) d\omega_i}_{\text{Reflected}} \tag{2.11}$$

Where:

$L_o(x,\omega)$	= The total outgoing radiance from a 3D point x, in the direction ω
$L_e(x,\omega)$	= The emitted radiance from the point x, in the direction ω
Ω	= Hemisphere centred at x oriented to the normal $\mathbf{n}$ of the surface, containing all angles ω_i
$L_i(x,\omega_i)$	= The radiance incident on x in direction ω_i
$f_r(\omega_i,x,\omega)$	= The BRDF, describing the proportion of light reflected from ω_i in direction ω
$cos(\theta_i)$	= Cosine of the angle between surface normal at point x and the direction ω_i

The rendering equation is based on the physical law of the conservation of energy. Where the outgoing radiance in a given direction from a point (L_o), is equal to the emitted light (L_e) from that point in that direction, plus the reflected light (the integral) from that point in that direction. The emittance term L_e is simple, it is the light emitted from the point x which has been intersected with. If this is non-zero, a light source has been intersected with. However, the reflected light which is represented by the integral is generally analytically intractable, as it involves summing the contribution of incoming radiance from infinitely many directions in the hemisphere Ω around the point x. Also, the incident radiance function L_i is recursive [23], as to calculate the radiance incident in the direction ω_i on x, requires a solution to $L_o(h(x,\omega_i),-\omega_i)$. Where $h(x,\omega_i)$ is the hit-point function, which determines the closest intersected position $y \in \mathbb{R}^3$ in the scene by shooting a ray from x, in direction ω_i. This concept is represented Figure

2.4. Note, the functions L_o, L_e, L_i are all 5-dimensional functions as they both takes a position in the scene $x \in \mathbb{R}^3$ and a direction $\omega \in \mathbb{R}^2$.

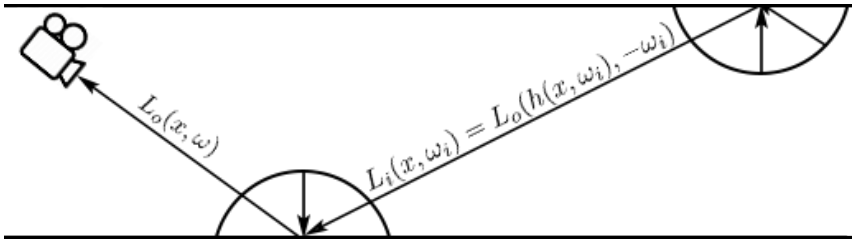
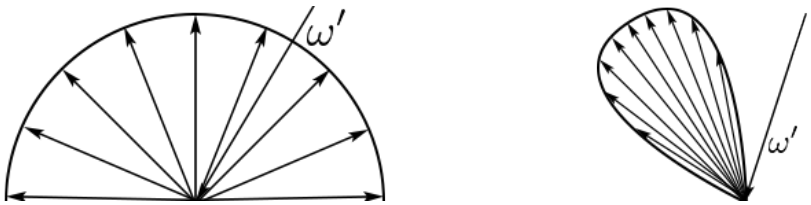


Figure 2.4: A diagrammatic representation of the recursive nature of the rendering equation. The outgoing radiance (L_o) in a given direction ω from a point x requires an estimation of the incident radiance coming from all angles in the hemisphere around the point, that is $L_i(x, \omega_i)\ \forall \omega_i \in \Omega$. To calculate $L_i(x, \omega_i)$ is identical to calculating the outgoing radiance $L_o(h(x, \omega_i), -\omega_i)$ as we assume no radiance is lost along a ray line, hence L_o, L_i are recursive functions.

The f_r term in Equation 2.11 is known as the bidirectional reflectance distribution function (BRDF). On a high level, the BRDF describes how a surface interacts with light [27]. Every surface has a BRDF which determines the ratio of reflected radiance in direction ω, when a ray intersects with that surface at a given incident direction ω_i. Therefore, querying the BRDF for a surface at point x with an incident ray direction ω' and given reflected direction ω, that is $f_r(\omega', x, \omega)$, a single scalar value is returned. Different materials can have vastly different BRDFs. For example, a diffuse material reflects light almost equally in all directions for any angle of incidence, an example of this is paper. Whilst for specular materials, incident rays are reflected in a narrow area around the perfect reflection direction, many metals exhibit specular reflections. Example BRDFs of both a diffuse and specular surface are depicted in figure 2.5.



(a) Diffuse BRDF (b) Specular BRDF

Figure 2.5: A representation of both a diffuse surface and specular surface BRDF's for a given angle of incidence ω'. The surface point is located where all outgoing arrows begin. The arrows indicate a subset of directions possible for the incident ray to be reflected in. All possible directions an incident ray be reflected in are defined by any vector which can be formed from the surface point to the curved line for an incident direction ω'. The further away a point is on the curved line, the more likely a ray is to reflected in a direction from the surface point to that point on the curved line. The diffuse surface is equally likely to reflect a ray in any direction. Whereas, the specular surface favours a small subset of directions in the hemisphere surrounding the surface point.

Another way to distinguish between diffuse and specular materials is their appearance from different viewing angles. For example, surface of paper appears to be identical no matter the viewing angle. However, a shiny metal ball would appear to reflect what was in front of it, which changes depending on the viewing angle, just like a mirror. These differences can be scene in figure 2.6.

Diffuse materials are generally more computationally expensive to approximate the outgoing radiance L_o from. This is because we have to simulate rays reflecting in all directions around the intersection point on the surface, compared to a specular surface where only a small subset of directions need to be sampled for each intersection. So, from here on whenever a surface or BRDF is mentioned, assume it is diffuse as our descriptions can be extended to specular materials by restricting ray reflections to a limited set of angles.

Finally, as mentioned $\cos(\theta_i)$ is the cosine of the angle between ω_i and the normal of the intersected surface. The normal of a surface $\mathbf{n}$ is the normalized vector that is perpendicular to the surface [76]. The $\cos(\theta_i)$ is a weighting for reflected radiance from a point, where the larger the angle from the normal

(a) La Table aux Amours, Marble Sculpture

(b) Cloud Gate, Stainless Steel Sculpture

Figure 2.6: Two sculptures, one made from a diffuse material (a) and the other from a specular material (b). The specular sculpture appearance is dependent upon the viewing angle, whereas the diffuse sculpture is not.

the smaller the reflected radiance. This simulates how light is spread across the surface, causing less light to be reflected in a direction which is further away from being perpendicular to the surface. The combined BRDF and cosine term in the rendering equation uphold the physical law of conservation of energy, meaning more radiance cannot be reflected from the surface then incident on the surface. This is formally described by equation 2.12 [27], where ω_r represents a direction of reflection.

$$\forall \omega_i, \int_{\Omega} f_r(\omega_i, x, \omega_r) \cos(\theta_r) d\omega_r \leq 1 \tag{2.12}$$

2.2.2 Path Tracing

Monte Carlo Path Tracing

In section 1.1 we gave a high level overview of the path tracing algorithm. To summarise, many light paths are sampled for each pixel, each sampled light path has a colour estimate. To determine a pixel's colour, the colour estimates for the light paths sampled for the pixel are averaged. It turns out that the process we have just described represents a MC solution to the rendering equation. The proof behind this is detailed in [30], but conceptually it is simple.

Previously the reflected radiance for a point x in direction ω was given by the integral in equation 2.11 with respect to all possible incident angles $\omega_i \in \Omega$ in the hemisphere surrounding x. To calculate this integral one can trace infinitely many light paths from the intersection point x in all possible directions Ω until they intersect with a light source. Each light path approximates the incident radiance in the direction they were sampled. The sum of all these light paths gives the total amount of incident radiance on point x. The MC approximation to the rendering equation using the process we have just described is given in equation 2.13. Where L_o^N represents the outgoing radiance estimate using N sampled light paths.

For a given pixel which is intersected with by a direction ω from point x in the scene, an approximation for $L_o^N(x, \omega)$ directly gives the estimated colour value for that pixel. Therefore, we can render an image by approximating $L_o^N(x, \omega)$ for every pixel in the image.

$$L_o^N(x,\omega) = \tfrac{1}{N} \textstyle\sum_{k=0}^{N-1} L_e(x_0, \omega_0) + (L_i(x_0, \omega_1) \cdot f_s(\omega_1, x_0, \omega_0) \cdot cos(\theta_{\omega_1}))/\rho_1$$

Such that

$$L_i(x_i, -\omega_i) = \begin{cases} L_e(x_i, \omega_i) & \text{if } x_i = \text{Light Source} \\ L_e(x_i, \omega_i) + (L_i(x_i, \omega_{i+1}) \cdot f_s(\omega_{i+1}, x_i, \omega_i) \cdot cos(\theta_{\omega_{i+1}}))/\rho_i & \text{otherwise} \end{cases} \tag{2.13}$$

Where:

x_i = Intersection location of the light path after i reflections in the scene
ω_i = Direction of the light path after i reflections in the scene
ρ_i = PDF over reflected ray directions for position x_i evaluated at the angle of incidence ω_i

In equation 2.13 the recursive L_i is still present, but the recursion is terminated when the light path intersects with a light source. By the law of large numbers in equation 2.5, the larger the number of sampled light paths (SPP) N for solving equation 2.13, the closer the approximated colour value for each pixel will be to its true colour value. Meaning, the more samples used in the MC approximation in equation 2.13, the lower the amount of noise in the image [17]. To visualise this concept, figure 2.7 presents seven rendered images using an increasing number of SPP N for the solution of L_o^N. The more samples used, the lower the noise in the rendered image. The pseudo code for a simple path tracer which was used to produce the images in figure 2.7 is given in algorithm 1.

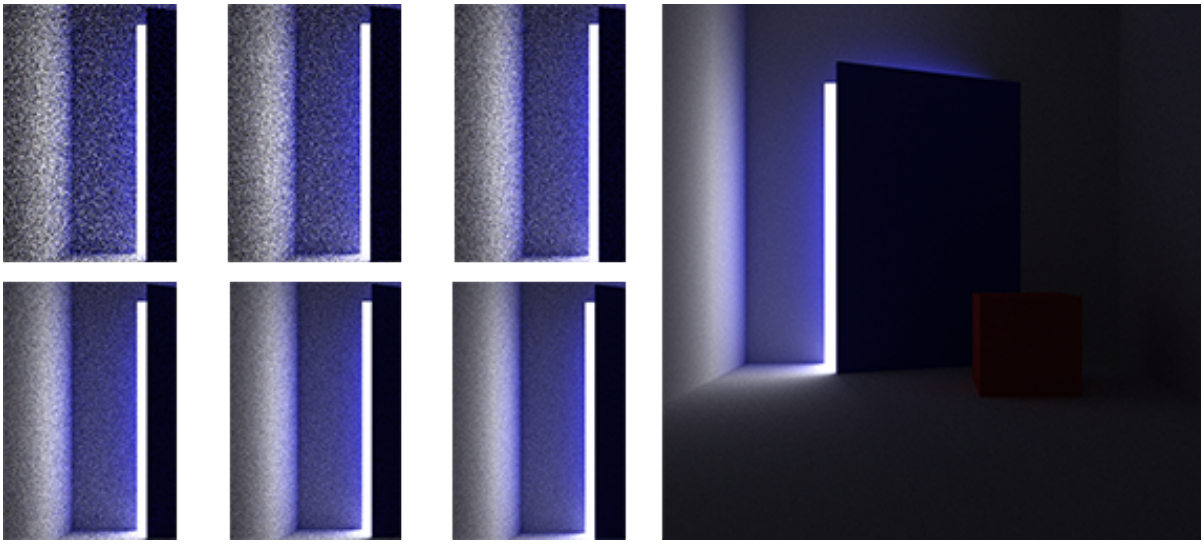


Figure 2.7: An indirectly illuminated scene rendered by path tracing. The grid of image sections show how increasing the number of SPP reduces image noise. Beginning in the top left with 16 SPP, to the bottom right with 512 SPP by doubling the number of SPP each time. The full image on the right is a reference image with 4096 SPP where the MC approximation has almost converged for all pixel values.

Algorithm 1: Pseudo code for a path tracer. Given a camera position, scene geometry, this algorithm will render a single image by finding the colour estimate for each pixel using Monte Carlo path tracing. Where N is the pre-specified number of sampled light paths per pixel.

```
Function renderImage(camera, scene)
    for i = 1 to N do
        for p in camera.screen do
            /* Set the ray to be traced from the camera, through the pixel and into the scene  */
            ray ← initializeRay(p, camera)
            for j = 1 to ∞ do
                /* Find the rays closest intersection with the scene geometry                   */
                (y, n, L_e) ← closestIntersection(ray, scene)
                if noIntersection(y) or areaLightIntersection(y) then
                    ray.throughput ← ray.throughput · L_e
                    updatePixelColourEstimate(p, ray.throughput)
                    break
                end
                (ω_i, ρ_i, f_s) ← sampleRayDirRandomly(y)
                ray.throughput ← ray.throughtput · f_s · (ω_i · n)/ρ_i
                ray ← (y, ω_i)
            end
        end
    end
end
```

Importance Sampling in Path Tracing

As path tracing is a MC method for solving the rendering equation, Importance sampling can be applied in order to reduce the variance in pixel colour estimates. In section 2.1.2, we showed that by using a PDF which closely matches the shape of the function being integrated, the variance in the Monte Carlo approximation can be significantly reduced.

The term ρ_i in equation 2.13 represents the PDF over directions to continue a light path in for intersection location x_i, evaluated at direction ω_i. The path tracer in figure 2.7 uses a uniform PDF, such that each direction in the hemisphere surrounding an intersection point has a probability of $\rho_i = \frac{1}{2\pi}$ of being sampled. But this can be modified with prior knowledge of which directions are more important for continuing a light path in for a given position in the scene. Important directions are those which leads to a high contribution of incident radiance.

The question now is, do we have any knowledge of which directions contribute the most radiance for every possible point in the scene? Or in other words, do we have any knowledge of the incident radiance function $L_i(x, \omega)$ for a scene? The answer is yes, and there has been a large amount of research in this topic. The simplest example lies within the rendering equation itself, $\cos(\theta_i)$. As previously discussed, this term acts as a weighting for the contribution of radiance from a given direction. So, the probability density function ρ_i can also be weighted by $\cos(\theta_i)$, which is likely to reduce the pixel value variance, as our PDF will be closer to the true incident radiance function L_i.

There exists many other methods of retrieving knowledge from the scene to use in importance sampling during rendering. For example, irradiance cahcing [9], table-driven adaptive importance sampling [18], and sequential Monte Carlo adaptation [57]. However, these methods suffer from the problem described in 1.2.3 where they do not account for light blockers. This is because they do not actually attempt to learn the incident radiance function, instead they make assumptions about its general shape. Meaning if these assumption are incorrect, there is potential that the PDF used by these methods for importance sampling is a far different shape to the true incident radiance function L_i. Hence, they may actually increase the presence of noise rendered in images (see section 2.1.2) when compared to using a uniform PDF.

To avoid this issue, NVIDIA proposed that a reinforcement learning method can be adapted and used to learn the incident radiance function. This approximation of the incident radiance function is then used for importance sampling directions to continue light paths in [20], leading to a reduction in image noise. We will present NVIDIA's path tracer in chapter 3

2.3 Reinforcement Learning and TD-Learning

In this section we give a quick introduction to reinforcement learning and TD-learning. TD-learning learning rules are what both we and NVIDIA propose to use for approximating the incident radiance function. This approximation is used to importance sample directions to continue light paths in, in order to reduce the noise present in images rendered by path tracing. Exactly how TD-learning can be applied to learn the incident radiance function is discussed in chapter 3.

2.3.1 Markov Decision Processes

Reinforcement learning is one of the three archetypes of machine learning. It is concerned with finding what action should be taken in a given situation, in order to maximise a numerical reward [66]. This problem is formalized by a finite Markov Decision Process (MDP), which is designed to capture the most important aspects of the problem a learning agent faces when interacting over time with its environment to achieve a goal. An MDP is summarised in figure 2.8 and can be described in terms of the following:

- **Agent** - The learner and decision maker which takes an action A_t in an observed state S_t (where t is the current time step), receiving an immediate numerical reward R_{t+1} and the next observed state S_{t+1}
- **Environment** - What the agent interacts with when taking an action A_t in state S_t and produces both R_{t+1} & S_{t+1}

An MDP comprises of the following the following tuple:

$$(\mathcal{S}, \mathcal{A}, p, \gamma)$$

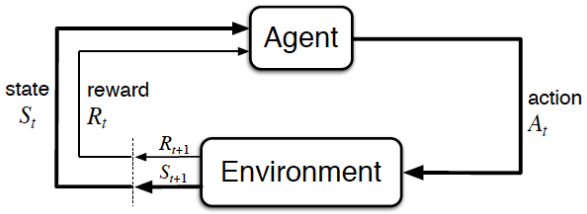


Figure 2.8: Markov Decision Process [66]

Where:

$\mathcal{S}$ = The set of all states
$\mathcal{A}$ = The set of all actions
p = Probability of receiving reward r and state s' when in the previous state s and action a was taken
γ = The discount factor which makes the agent value immediate rewards higher than later ones

An important detail of an MDP is that any problem modelled by an MDP assumes the Markov property.

> "The future is independent of the past, given the present." - *Hado van Hasselt, Senior Research Scientist at DeepMind* [70]

This is expressed mathematically for an MDP in equation 2.14. Put simply, the Markov property means the current state captures all relevant information from the history of all previous states the agent has experienced. Meaning the history of all previous states are not needed if we have the current state.

$$p(R_{t+1} = r, S_{t+1} = s'|S_t = s) = p(R_{t+1} = r, S_{t+1} = s'|S_1, .., S_{t-1}, S_t) \tag{2.14}$$

2.3.2 Goals and Rewards

The goal of a reinforcement learning agent can change significantly depending on the problem. For example, in the case of a game it may be to maximise the total score in one play-through. Or for a robotic space rover, it may be to discover the most amount of unseen terrain. However, in terms of an MDP all AI agents goals are described as maximising the total amount of cumulative reward received. This is more formally described by the reward hypothesis [66]:

> Any goal can be formalized as the maximisation of the expected value of the cumulative sum of a received scalar reward signal.

Once again, in the case of an agent learning the best possible action to take for any state in a game, a reward signal could be the points gained by making a certain move. Therefore, to maximise the expected return would be to maximise the number of points received in a play-through of the game in this case. The return is formally defined in equation 2.15. It consists of a sequence of reward signals (R_{t+i}), combined with the discount factor (γ), which as previously mentioned trades off later rewards for more immediate ones. If the discount factor is closer to 1, the agent is said to be far sighted, as it gives future rewards a high weighting. Whereas a myopic agent is one which has a discount factor closer closer to 0, as it gives a lower weighting to future rewards for their contribution towards the return G_t [70].

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.15}$$

This formulation works well if the agents interactions with the environment break down easily into sub-sequences [66]. Where an agent starts in one of a given set of *starting states* and takes a series of actions to reach a *terminal state*, the process of taking a series of actions from the starting state to the terminal state is known as an *episode*. From the terminal state the agent can be reset to one of the starting states to begin learning once again. This applies to path tracing, where the terminal state is one in which the light path has intersected with a light. This will be discussed in detail in chapter 3.

2.3.3 Value Function and Optimality

All reinforcement learning algorithms we will be considering involve the concept of a value function. There are two kinds of value functions, one which determines the value of being in a given state, the other determines the value of being in a certain state and taking a certain action, known as a state-action pair. We will be focusing on state-action pair value functions, where the value of a state-action pair is defined in terms of the expected return G_t from that state-action pair.

An agent follows a policy π, which determines how the agent will act in a given state. More formally, a policy is a mapping from states to probabilities of selecting a particular action. When an agent is following policy π at time t, then $\pi(a|s)$ is the probability of the agent taking action $A_t = a$ in state $S_t = s$. The reinforcement learning algorithms we shall discuss state how an agents policy changes from experience. In other words, how the agents behaviour changes depending on previous experiences.

The value of a state-action pair (s, a) under a policy π, is given in equation 2.16 denoted as $q_\pi(s, a)$,. This value function is commonly known as the action value function for policy π. Stating 'under policy π' is important as the value of a given state-action pair depends upon the actions we take onwards from taking action a in state s while following π. $\mathbf{E}_\pi$ denotes the expected value of a random variable, given that the agent follows policy π. From this, if we were to keep track of the actual returns received for taking a state-action pair, then as the number of times all state-action pairs are chosen tends to infinity, the average of the returns for the state action pair (s, a) will converge on the true expected value of the return for that state-action pair $q_\pi(s, a)$.

$$q_\pi(s, a) = \mathbf{E}_\pi[G_t|S_t = s, A_t = a] = \mathbf{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.16}$$

Now, if we had an AI agent the best way it could perform would be to maximise the true expected reward it receives in an episode. The policy which does this is known as the *optimal policy*, which is provably better than all other policies an agent can follow. Formally, the optimal policy is π if $\pi \geq \pi'$ for al possible policies π'. Where, $\pi \geq \pi'$ if and only if $q_\pi(s, a) \geq q_{\pi'}(s, a)$ for all $s \in \mathcal{S}$ and $a \in \mathcal{A}$. The optimal policy is denoted as π_* and the value function following the optimal policy, which is the *optimal value function*, is denoted $q_*(s, a)$. The optimal value function is defined in equation 2.17.

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.17}$$

$$= \mathbf{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{2.18}$$

Equation 2.18 defines the *Bellman optimality equation*, which states that the value of a state-action pair under an optimal policy must be equal to the expected return of the immediate reward plus the highest valued state-action pair available from the next state. Intuitively, if the optimal policy is available which is essentially a cheat sheet of what action is most valuable to take in each state. Then the true value of a given state-action pair should equal the immediate reward received by taking the action in the current state, plus the value of the best action to take in the next state given by the cheat sheet/optimal policy. Similarly, if we have the optimal value function, the optimal policy can easily be found by choosing the action a with the highest value $q_*(s, a)$ in state s [66].

To summarize, the aim from here on is to build an agent which is able to learn the optimal value function. But whilst this is provably possible, it rarely happens in practice. However, the learning methods I will discuss in the next section on TD-learning are able to learn a good value function in practice [65, 40, 67, 45].

2.3.4 Temporal Difference Learning

TD-Learning is combination of Monte Carlo and Dynamic Programming reinforcement learning methods for learning the optimal value function from equation 2.17. We will not discuss the details of Monte Carlo and Dynamic Programming methods as they are not investigated as part of our work. However, the reasoning for choosing to study TD-learning approaches over these two alternative approaches are as follows; TD-learning can perform learning updates of the value function throughout an episode, unlike Monte Carlo approaches which wait until the end [72]. This means TD-learning algorithms can be implemented as online learning algorithm [66], meaning they are able to learn during an episode. TD-learning can learn directly from experience, as it does not require a true model of the environment in order

to learn the optimal value function, unlike Dynamic Programming methods [71]. This means TD-learning is model-free, avoiding the expense of building the true model of the environment.

We will now introduce three different TD-learning methods which are required knowledge for the rest of our work.

Sarsa

Sarsa is an on-policy TD method which learns a state-action pair valuation function $q_\pi(s, a)$. The Sarsa learning rule is presented in equation 2.19, and we have chosen to present this method first to explain some key concepts TD-learning methods share. Firstly, Q denotes the current approximation of the optimal value function under policy π, q_π. Therefore the left arrow indicates an update in the currently estimated value of state-action pair (s, a), $Q(s, a)$. Also notice, how the current estimate is update upon every time step t, this means Sarsa like other TD-learning methods can learn during an episode as previously mentioned. The α term is the current learning rate, where $\alpha \in [0, 1]$ and γ is the discount factor as previously described. Finally, Sarsa performs what is known as bootstrapping in the context of reinforcement learning [66]. Bootstrapping is where the current estimate of the value function (Q) is updated based on some new data received by experience, which is the immediate reward R_{t+1}. As well as, the current estimate of the value function Q. Sarsa therefore learns from experience after every time step t, whereby an action A_t is taken in state S_t, leading to an immediate reward R_{t+1} which is used to update the current estimate Q.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.19}$$

The reasoning behind the name Sarsa is that the method uses every element in the quintuple, $(S_t, A_t, R_{t+1}, S_{t+1}, A_{t_1})$, which makes up a transition between each time step of state-action pairs. Sarsa is an on-policy TD-learning method, as to choose the next action to take in the next state $Q(S_{t+1}, A_{t+1})$ the policy is used. Sarsa is proven to converge on the optimal value function q_* when the policy π remains constant. Sarsa is a tabular learning method for approximating the optimal value function q_*. Meaning, a table with an entry for every state-action pair is required for the algorithm to converge on the optimal value function by using the update rule in equation 2.19.

To make Sarsa learning and TD-learning in general more concrete, imagine a robot with a camera whose goal it is to manoeuvre to the end of a corridor. Each time step is the point when a new frame is rendered on the camera, and the state is the image displayed by the camera. The robots actions consist of moving a short distance in a set of four different directions. If the robot were to learn using Sarsa, the robot would have a large table storing the value of each state-action pair $Q(S_t, A_t)$, which represents the current approximation of the value function. The robot would then select an action at each time step according to the policy π to receive a reward signal based on its distance to the end of the corridor. The robot would then perform a lookup on the large table, indexing with the action it took in the state it was in to perform the update rule in 2.19. This large table representing the current estimate of the optimal value function is also know as a *Q-table*, where each value in the table is known as a *Q-value*, $Q(S_t, A_t)$. By following a suitable policy, the robot will over time keep updating its Q-values to improve its estimate of $q_\pi(s, a)$.

Q-Learning

Q-learning is very similar to Sarsa except it is an off-policy TD-learning algorithm. Also, if all state-action pairs are visited infinitely many times, it is proven that Q-learning can converge on the optimal policy $q_\pi(s, a)$ faster than Sarsa, making it a generally a preferred method to Sarsa. The learning rule is given in equation 2.20, where rather than following a policy π to select the action to update with, the maximum value of the highest valued action in the next state is selected $\max_a Q(S_{t+1}, a)$. This means the agent following policy π will still choose its actions from a state based on π, however when it updates its value function Q, the action it chooses to update with may not necessarily be the same as the action it took. Hence, Q-learning is an off-policy TD-learning algorithm.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.20}$$

Expected Sarsa

Expected Sarsa is a TD-learning algorithm which is generally found to be superior to both Q-learning and Sarsa in practice [66]. It is very similar to Q-learning, but instead of using the maximum value of the

next state-action pairs to update its estimate, it uses the expectation over them. This means Expected Sarsa takes into account how likely each action is under the current policy as shown in equation 2.21. Where $\pi(a|S_{t+1})$ returns the probability of selecting action a in state S_{t+1}, while following the policy π. Note, Expected Sarsa may be used as either an on-policy or an off-policy algorithm. For example, if the policy π was set to the greedy-policy used in Q-learning, the learning rule would become identical to that of Q-learning. Therefore, Expected Sarsa generalizes over Q-learning. Expected Sarsa also reduces the variance in the value function approximation compared to that of Sarsa, as the expectation is taken over the current value estimate for all state-action pairs available from the current state.

$$\begin{aligned} Q(S_t, A_t) &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbf{E}[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)] \\ &\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \end{aligned} \tag{2.21}$$

2.3.5 Exploration vs Exploitation

Up to now we have formally introduced what reinforcement learning is, including what the optimal value function is and different ways to learn it. In the case of off-policy methods like Q-learning, we have not yet discussed any details about the kind of policy an agent should use to select actions for learning the optimal value function. Deciding on this policy is very influential on our agents performance, as the agent needs to gather enough information about its environment to make the best overall decisions. Therefore, online decision-making requires a fundamental choice to be made by the agent every time it chooses to take an action [68]. This is between the following:

- **Exploitation**: Maximise the agents performance based on the current knowledge available
- **Exploration**: Gain more knowledge about the environment

This means a good learning strategy for an agent may be to sacrifice short-term performance to maximise it in the long-term. This applies directly to the policies used for both on-policy and off-policy TD-learning methods. Initially exploration is the most important to quickly gain a broad amount of knowledge about the environment, opening up more specific areas for further exploration. Then over time the policy should favour exploitation more and more by taking known higher valued actions. An example of this kind of policy is the decaying ϵ-greedy policy [66]. This policy maintains the current value of $\epsilon \in [0, 1]$ and involves sampling a random number $x \in [0, 1]$, then if $x > \epsilon$ exploitation occurs, else exploration. By exploitation it could choose the current highest valued action from a state, whereas exploration involves choosing one at random. Overtime ϵ is decayed by a value δ as more episodes of training are accumulated, achieving the behaviour of increasing exploitation as more knowledge is gained.

Summary

The story up to this point can be summarised as follows; Monte Carlo integration is a numerical technique for approximating integral. It can be used to find an approximation of pixel values in path tracing by approximating the radiance from a point on a surface in a given direction. Rather then uniformly sampling directions in light path construction for determining pixel values, a whole field is dedicated to importance sampling these directions to reduce the variance in the Monte Carlo approximation. However, most traditional methods do not learn the incident radiance function, limiting their effectiveness for reducing image noise. Instead, reinforcement learning, specifically temporal difference learning techniques can be applied to learning the outgoing radiance from a point in a given direction, which are in turn used for importance sampling ray directions where the approximated radiance is high. We are specifically interested in TD-learning methods which are used to approximate the optimal value function, which in turn is used to make a better decision on what action an agent should take from a given state.

Chapter 3

The Expected Sarsa and Neural-Q Path Tracers

Monte Carlo integration, importance sampling, path tracing, and reinforcement learning have all been introduced. So now it is finally time to combine all of these concepts to build path tracing algorithms which importance sample directions to continue light paths in to reduce image noise.

In this section we introduce two modified path tracers. The first is based on the NVIDIA path tracer, which we refer to as the Expected Sarsa path tracer. The other is our newly proposed path tracer which we refer to as the Neural-Q path tracer. Both path tracers progressively learn the incident radiance function for importance sampling directions to continue light paths in. This reduces the variance in the Monte Carlo approximation of a pixels colour, ultimately reducing the noise in rendered images. However, the Expected Sarsa path tracer learns the incident radiance function for a discrete set of locations in the scene, whereas our Neural-Q path tracer learns the function for the continuous set of locations in a scene.

3.1 The Expected Sarsa Path Tracer

In this section we introduce the Expected Sarsa path tracer proposed by NVIDIA in [20]. In doing so, we will cover how light transport can be linked to the rendering equation for learning the incident radiance function, the Irradiance Volume data structure, and the modified path tracing algorithm itself.

3.1.1 Linking TD-Learning and Light Transport Simulation

Here, we will link together the concepts introduced in chapter 2 to derive a way of learning the incident radiance on any position in the scene x from any direction ω. In other words, we will step through the derivation of the learning rule given in [20] for learning the incident radiance function based on the TD-learning method Expected Sarsa.

First, the Expected Sarsa learning rule's summation over the set of all actions $\mathcal{A}$ can be represented as an integral with respect to an action a over all actions $\mathcal{A}$, as shown in Equation 3.3.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{3.1}$$

$$= (1-\alpha) \cdot Q(S_t, A_t) + \alpha \cdot \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) \right) \tag{3.2}$$

$$= (1-\alpha) \cdot Q(S_t, A_t) + \alpha \cdot \left(R_{t+1} + \gamma \int_{\mathcal{A}} \pi(a|S_{t+1})Q(S_{t+1}, a)da \right) \tag{3.3}$$

Recall the rendering equation from section 2.2.1 describes the radiance in an outgoing direction ω from point x is equivalent to the emitted radiance in ω from x plus the reflected radiance in the ω from x. Note, we have set the incident radiance function to $L_i(x, \omega_i) = L_o(h(x, \omega_i), -\omega_i)$ (see section 2.2.1).

$$L_o(x, \omega) = L_e(x, \omega) + \int_\Omega L_o(h(x, \omega_i), -\omega_i) \cdot f_r(\omega_i, x, \omega) \cdot \cos(\theta_i) d\omega_i$$

Now, by matching terms from the rendering equation to the Expected Sarsa learning rule in equation 3.3, equation 3.4 is formed. Where this new learning rule is designed to approximate the incident radiance in direction ω on a point x in the scene. Therefore, the Q-value of the state-action pair $Q(x = S_t, \omega = A_t)$ is determined by the amount of radiance incident in direction ω on x. An important detail which may be overlooked is that the substitution of $\gamma \cdot \pi(a|S_{t+1})$ for $f_s(\omega_k, y, -\omega) \cdot \cos(\theta_i)$ ensures that a trade off of long term rewards for more immediate rewards is made as $f_s(\omega_k, y, -\omega) \cdot \cos(\theta_i) \leq 1$. Meaning, the learning rule accurately accounts for a light paths loss of energy as it reflects off surfaces within the scene.

Below, we have listed what each term in the Expected Sarsa learning rule is matched to for deriving equation 3.4:

S_t	= 3D position in the scene, $x \in \mathbb{R}^3$
A_t	= Sampling a direction to continue the light path in from location x, in direction ω
S_{t+1}	= 3D position of a light ray reflected from x in direction ω, $y = h(x, \omega)$
R_{t+1}	= Emitted radiance from point y in direction $-\omega$, $L_e(y, -\omega)$
$\mathcal{A}$	= All directions in the hemisphere at x, oriented to the surface normal at x, Ω
$\gamma \cdot \pi(a\|S_{t+1})$	= BRDF and the cosine of the angle between y and ω_i, $f_r(\omega_i, y, \omega) \cdot \cos(\theta_i)$
$Q(S_t, A_t)$	= Approximated radiance incident on x from direction ω, $L_i(x, \omega) = Q(x, \omega)$

$$Q(x, \omega) \leftarrow (1 - \alpha) \cdot Q(x, \omega) + \alpha \cdot \left(L_e(y, -\omega) + \int_\Omega Q(y, \omega_i) f_s(\omega_i, y, -\omega) \cos(\theta_i) d\omega_i \right) \tag{3.4}$$

Finally, Monte Carlo integration with a uniform distribution PDF can be used to approximate the integral in equation 3.4. This converts the action space from continuous to m discrete angles and provides a numerical solution for approximating the incident radiance on x from direction ω, as shown in equation 3.5.

$$Q(x, \omega) \leftarrow (1 - \alpha) \cdot Q(x, \omega) + \alpha \cdot \left(L_e(y, -\omega) + \frac{2\pi}{m} \sum_{k=1}^{m-1} Q(y, \omega_k) f_s(\omega_k, y, -\omega) \cos(\theta_k) \right) \tag{3.5}$$

The estimated incident radiance function for a point x consists of m approximated incident radaince values from directions $\omega_i \forall i = 1, ..., m$. These approximated incident radiance values can be converted into a distribution. This distribution is an apporximation of the *radiance distribution* for a point [29]. A good approximation of the radiance distribution at a point x will have a similar shape to the true function of incident radiance at point x, $L_i(x, \omega)$ $\forall \omega \in \Omega$. Therefore, by MC importance sampling, sampling directions to continue light paths in from the learned radiance distribution and using it as the PDF in the MC approximation, will lead to a significant reduction in the variance of the approximation. Resulting in a significant reduction in image noise. Figure 3.1 gives an illustration of the radiance distribution for a point.

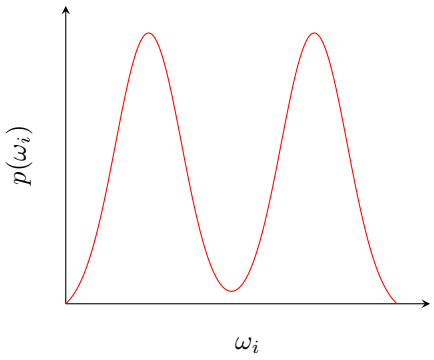


Figure 3.1: An illustration of a simple incident radiance distribution for a given point x in a scene. Where ω_i $\forall \omega_i \in \Omega$ are all possible incident directions in the hemisphere around the point x oriented to the surface normal at x. The scene contains two area lights with the same power which are visible from x in the scene, causing two equal level peaks in the incident radiance distribution for x.

3.1.2 The Irradiance Volume

The Expected Sarsa learning rule derived in equation 3.5 requires some sort of data-structure for looking up and updating the incident radiance on a point x from direction ω_i, which is also known as a Q-value. Therefore, the main requirement of the data structure is that it has some way of representing a discrete set of angles ($\omega_k \ \forall k = 1, ..., m$) in a hemisphere located at a position x and oriented to the surface normal at x. The Irradiance Volume data structure [29] meets this requirement.

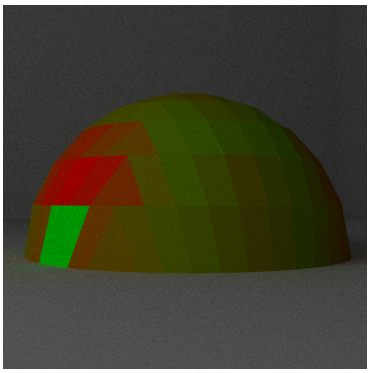
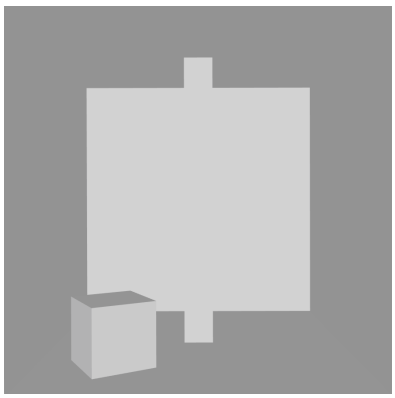
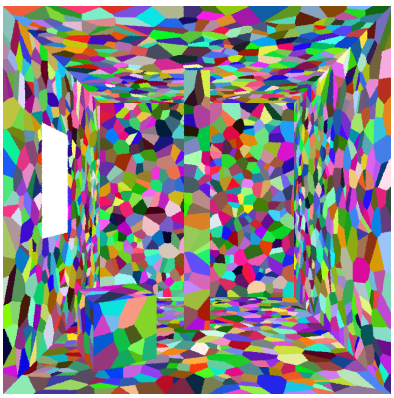


Figure 3.2: An Irradiance Volume. Each sector holds the incoming radiance $L_i(x, \omega_k)$, the more green a sector is the lower the stored radiance in that sector, the more red a sector is the higher the stored radiance in that sector.

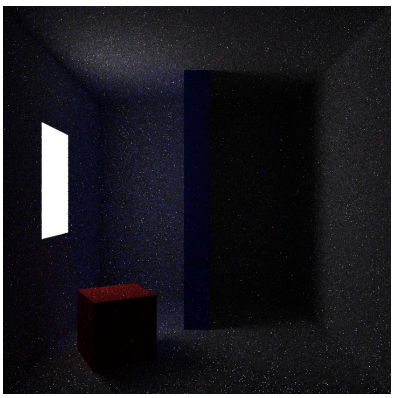
Originally designed to be used for pre-computation of radiance values which are looked up at runtime to approximate global illumination, the Irradiance Volume data structure is essentially a discretized version of a hemisphere which is visually represented in figure 3.2. The image shows the discrete sectors which make up a hemisphere. This was implemented by converting a 2-dimensional square grid into the 3-dimensional hemisphere shown, which is known as an adaptive quadrature. Where all sectors in the 2-dimensional grid have an equal area and a mapping which we refer to as an *adaptive quadrature* mapping introduced in [63], is used to convert the 2-dimensional grid coordinates into a hemisphere made up of sectors in 3-dimensional space. The mapping ensures the hemisphere sectors remain equal to one another, meaning the discrete set of directions represented by the hemisphere are of equal angles apart from one another.



(a) Representation of the scenes geometry meshes



(b) Voronoi Plot of Irradiance Volume locations



(c) Expected Sarsa path tracer with 16 SPP

Figure 3.3: An example of discretizing location in the scene into Irradiance Volume locations. The geometry mesh (a) is used to uniformly sample Irradiance Volume positions. Image (b) shows a voronoi plot for the Irradiance Volumes in the scene, where each pixel is coloured to the represent its closest Irradiance Volume, so each sector of colour in (b) represents a different Irradiance Volume location. Finally (c) gives a render using the Expected Sarsa path tracer based on algorithm 2.

Each sector of the Irradiance Volume is then used to store the current approximation of incident

radiance in the direction formed by the unit vector travelling from centre of the sector to the centre of the hemisphere. Therefore, an Irradiance Volume stores the incident radiance (Q-value) for a given position x (state) in the scene, from each direction ω_k (action), for all sectors $\forall k = 1, ..., m$ in the hemisphere located at x.

In order to store Q-values across the scene, Irradiance Volumes can be uniformly sampled over the scenes geometry as shown in figure 3.3. Then, to lookup a radiance/Q-value for a given position x in direction ω_k, a nearest neighbour search is performed to find the closest Irradiance Volume to position x, followed by a lookup to retrieve the Q-value from the sector at index k. Using this method results in a lookup and update time of $O(\log n) + O(1) = O(\log n)$ when using the KD-Tree data structure for nearest neighbour search [11]. A KD-Tree is essentially a binary tree for nearest neighbour search in n-dimensional space.

Lookup and update procedures from the Irradiance Volumes in the scene are all that are needed to apply the Expected Sarsa learning rule in equation 3.5. The entire collection of Irradiance Volumes in a scene can be thought of as a large 5-dimensional table ($x \in \mathbb{R}^3, \omega \in \mathbb{R}^2$) to store the incident radiance approximations. This table is our Q-table (see section 2.3.4) which is used by the Expected Sarsa learning rule to approximate the incident radiance function.

Each radiance volume also stores the radiance distribution approximated for its centre point. This is calculated by normalizing its stored incident radiance values (Q-values) into a probability distribution. As described in section 3.1.1, a direction to continue a light path in can be sampled proportionally to the radiance distribution of the closest radiance volume to the a light paths intersection point. This distribution is also used as the PDF for calculating the MC approximation of a pixels colour value in path tracing (equation 2.13).

To summarize, all Irradiance Volumes combined (Q-table) store both the current approximation of the incident radiance function, as well as a PDF which is a normalized version of the approximated incident radiance function. Meaning, if the approximation of the incident radiance function becomes more accurate, the stored PDF will come closer to normalizing the the true incident radiance function. This will lead to a reduction in variance of the MC approximation of pixel colour values, leading to a fall in image noise. Exactly how we go about updating the incident radiance distribution is what we will cover in section 3.1.3.

3.1.3 Expected Sarsa Path Tracing

The Expected Sarsa path tracing algorithm is very similar to the original forward path tracer introduced in algorithm 1. The algorithm learns online, meaning after every rendered frame, pixel values are likely to have a lower variance due to an improvement in the approximation of the learned incident radiance function L_i stored in the Q-table formed by all Irradiance Volumes in the scene. Initially, Irradiance Volumes are sampled uniformly across the room with all Q-values initialised to a small constant proportional to the number of sectors in each Irradiance Volume m. This encodes the assumption that initially the radiance incident in all directions at any given point in the scene is equal. This a reasonable assumption, as initially we have no prior knowledge of any radiance values $Q(x, \omega_k)$.

With the radiance volumes set up, every frame N light paths are sampled from the camera, through each pixel and into the scene. This process determines the pixels average colour estimate using algorithm 2. Algorithm 2 updates the radiance distribution stored in each Irradiance Volume after every rendered frame by normalizing the updated $Q(x, \omega_k)$ values stored. Meaning the next frame will importance sample from the Irradiance Volumes updated radiance distributions. The three additions to algorithm 1 to form the Expected Sarsa path tracer are given below.

Addition 1: Sample directions proportional to closest radiance distribution

The direction to continue the light path in is sampled proportional to the radiance distribution held in the nearest Irradiance Volume to the intersection location y. In practice, the stored radiance distirbution at every Irradiance Volume is a set of discrete normalized Q-values corresponding to the discrete directions ω_k $\forall k = 1, ..., m$. Therefore inverse transform sampling [21] is used to sample a direction from the stored radiance distribution. Inverse transform sampling is where a random number $r \in [0, 1]$ is sampled, then the largest number x from the domain of the cumulative distribution $P(X)$ is returned where $P(-\infty < X < x) \leq r$. The cumulative distribution for an Irradiance Volume in our case is built from the normalized Q-values stored in the Irradiance Volume.

Addition 2: Update stored Q-values

Once the ray has intersected with a position in the scene y from a position x, the incident radiance estimate for $Q(x, \omega_k)$ is updated using the Expected Sarsa learning rule derived in equation 3.5. This is based on the radiance emitted from the next intersection point y in direction $-\omega$ (L_e) and the current approximation of the radiance incident on y. The summation over all Q-values for closest radiance volume to y gives the approximated incident radiance on the point y. The $\alpha(x, \omega_k)$ term is the current learning rate for the entry of the Q-table at (x, ω_k), we will discuss how this is calculated later in this section.

Addition 3: Update Irradiance Volume Distributions

Update every Irradiance Volumes radiance distribution in the scene, by normalizing the Irradiance Volumes updated $Q(x, \omega_k)$ values. Without this step, ray directions would be continued to be sampled uniformly at random.

Algorithm 2: Expected Sarsa path tracer pseudo code following NVIDIA's method in [20]. Given a camera position, scene geometry, this algorithm will render a single image using a tabular Expected Sarsa approach to progressively reduce image noise. Where N is the pre-specified number of sampled light paths per pixel.

Function *renderImage(camera, scene)*
  for $i = 1$ **to** N **do**
    for p **in** *camera.screen* **do**
      $ray \leftarrow$ initializeRay$(p, camera)$
      for $j = 1$ **to** ∞ **do**
        $(y, \mathbf{n}, L_e) \leftarrow$ closestIntersection$(ray, scene)$
        if $j > 1$ **then**
          /* Addition (1) */
          $(\omega_i, \rho_i, f_s) \leftarrow$ sampleRayDirFromClosestRadianceDistribution(y)
          /* Addition (2) */
          $Q(ray.x, ray.\omega) \leftarrow (1 - \alpha(ray.x, ray.\omega)) \cdot Q(ray.x, ray.\omega) + \alpha(ray.x, ray.\omega) \cdot \left(L_e + \frac{2\pi}{n} \sum_{k=1}^{n-1} Q(y, \omega_k) f_s(\omega_k, y, -ray.\omega) \cdot (\omega_k \cdot \mathbf{n})\right)$
        end
        if *noIntersection*(y) **or** *areaLightIntersection*(y) **then**
          $ray.throughput \leftarrow ray.throughput \cdot L_e$
          updatePixelColourEstimate$(p, ray.throughput)$
          break
        end
        $ray.throughput \leftarrow ray.throughtput \cdot f_s \cdot (\omega_i \cdot \mathbf{n})/\rho_i$
        $ray \leftarrow (y, \omega_i)$
      end
    end
  end
  /* Addition (3) */
  updateIrradianceVolumeDistributions()
end

Monte Carlo Integration

Due to the importance sampling from addition (1), the PDF over all possible sampled directions Ω at location x is no longer uniform. Instead it is equal to the stored radiance distribution in the nearest Irradiance Volume to the point x. Therefore, the evaluated PDF ρ_i at intersection point x, at incident direction ω_i must also be evaluated using the radiance distribution to approximate the MC integral using equation 2.13. To do so, we have derived an equation 3.6 to evaluate ρ_i for an intersection location x and outgoing direction ω_k.

$$\rho_i = Q_p(x, \omega_k) \cdot m \cdot \frac{1}{2\pi} = \frac{Q_p(x, \omega_k) \cdot m}{2\pi} \tag{3.6}$$

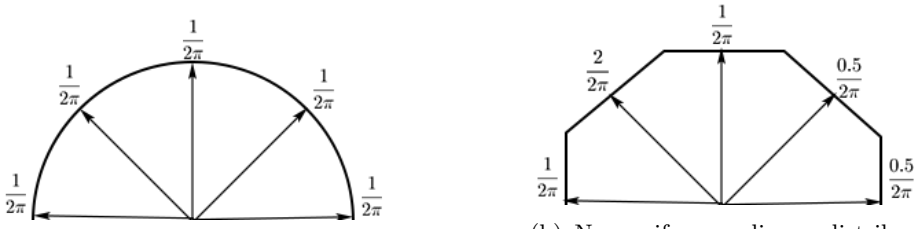
Where:

$Q_p(x, \omega_k)$ = Normalized Q-value from the Irradiance Volume closest to x at sector k
m = Total number of sectors in an Irradiance Volume

The reasoning behind this value ρ_i is that $\frac{1}{2\pi}$ represents the uniform PDF evaluated at any point when directions are randomly sampled in the hemisphere Ω. However, the Irradiance Volume splits the hemisphere into discrete sectors, but each sector still represents a continuous set of angles. Therefore, if the probability of sampling a ray in each sector were constant c, $Q_p(x, \omega_k) = c \; \forall k = 1, ..., m$, the PDF would remain constant:

$$\frac{Q_p(x, \omega_k) * m}{2\pi} = \frac{1}{2\pi}$$

However, the approximated radiance may vary across sectors. Causing the evaluated PDF to change across sectors proportional to $Q_p(x, \omega_k)$. The diagram in Figure 3.4 highlights these differences, where the non-uniform radiance distribution (PDF) is what Expected Sarsa uses. Whereas, the uniform radiance distribution is what was previously used in the default path tracer in algorithm 1.



(a) Uniform radiance distribution

(b) Non-uniform radiance distribution

Figure 3.4: A 2-dimensional view of a subset of values from two radiance distributions. One for a unit hemisphere (left) with a radiance distribution. One for an Irradiance Volume (right) with non-uniform radiance distribution. Where the arrows represent sampled directions and the values at the end of the arrows are the radiance distribution evaluated in the associated directions.

Consistency

It is important for our modified path tracing algorithm to converge in order to ensure consistency, meaning all newly introduced artefacts by the Expected Sarsa algorithm are guaranteed to vanish over time [20]. A decaying learning rate for α can be used to do so, see equation 3.7. Where visits(v, ω_k) is the number of times the rendering algorithm has updated the Q-value of the Irradiance Volume v for sector k representing angle ω_k.

$$\alpha(x, \omega) = \frac{1}{1 + \text{visits}(v, \omega_k)} \tag{3.7}$$

Now, while $Q(x, \omega_k)$ does converge, it does not necessarily converge on the true incident radiance function $q_*(x, \omega_k) = L_i(x, \omega)$. This is due to the discretization of both the continuous set of locations and directions the incident radiance function is approximated for. This has to be done as the Q-table cannot be infinitely large. In section 3.2 we present our new algorithm which does not require the discretization of the continuous set of locations in a scene to approximate the incident radiance function.

Another issue with this algorithm is that Q-values will not be visited the same number of times during rendering. For example Irradiance Volumes located in places which are in the cameras view will be visited far more, so images rendered of the scene which have been in the cameras view for the longest are likely to have the lowest variance in pixel colours. This is a problem, as parts of the scene may look particularly noisy compared to others as the camera begins to move round the scene.

3.2 The Neural-Q Path Tracer

This section is dedicated to presenting the details of our newly designed Neural-Q path tracing algorithm. Up to now we have only spoke of TD-learning techniques which use a tabular approach to approximate the the optimal value function. However, we show it is possible to instead use an artificial neural network as a non-linear function approximater for the optimal value function [69]. Following this, we introduce an artificial neural network architecture capable of learning the incident radiance function for the continuous set of possible locations in a scene. We then run through the pseudo code of the Neural-Q path tracing algorithm, which we designed and implemented for importance sampling directions to continue light paths in during path tracing.

3.2.1 Introduction to Deep Reinforcement Learning

Value Function Approximation

Observe the optimal value function which was first introduced in equation 2.17:

$$q_*(s,a) = \max_{\pi} q_\pi(s,a)$$

Recall this is simply a function which given a state and action, outputs a scalar representing the value of that state action pair. Therefore, rather than approximating it using a tabular form, instead we can learn the functions parametrized functional form which is a weight vector $\theta = (\theta_0, ..., \theta_n)$ where $\theta_i \in \mathbb{R}$. This turns the task of approximating the value function into a function approximation problem, for which there are many possible methodologies. To name some, Coarse coding, Decision Trees, Nearest Neighbour, Fourier basis, and Artificial Neural Networks (ANNs) [66]. Function approximators other than ANNs have been successful for a range of reinforcement learning problems, whilst maintaining both data and computational efficiency [65, 40, 67]. However, I will be using an ANN for value function approximation due to its capabilities of learning a non-linear function, and its good performance in the presence of a large amount of training data [42]. How these benefits are capitalized on will become more apparent later. By using a ANN for function approximation, we are now describing what is known as deep reinforcement learning.

Stochastic Gradient Descent

The goal of the ANN is to learn the optimal parameters for approximated value function θ, such that the approximated functions loss over all possible state-action pair inputs is minimised. In the case of ANNs the parameters θ are the weights for the connections between neurons. For stochastic gradient descent, a differentiable loss function which takes the parameter vector θ as input and outputs a scalar loss value is required [69]. The method from here on is to move the parameter values θ in the direction of the negative gradient to minimise the loss function $J(\theta)$, more formally:

$$\triangle\theta = -\frac{1}{2}\alpha\nabla_\theta J(\theta)$$

Where α is the step-size parameter. An example loss function for approximating the optimal value function is given in Equation 3.8. Note, it is common in literature to denote the optimal value function as q_π instead of q_*, we will stick to this convention from here on.

$$J(\theta) = \sum_{s\in\mathcal{S}}\sum_{a\in\mathcal{A}}(q_\pi(s,a) - \hat{q_\theta}(s,a))^2 \tag{3.8}$$

Where:

$J(\theta)$ = The loss function for the current parameter values θ
$q_\pi(s,a)$ = Value of the state-action pair under the optimal policy π
$\hat{q_\theta}(s,a)$ = Current approximation of the value for the state-action pair, using parameters θ

If plenty of data was available for $q_\pi(s,a)$ for many state actions pairs (s,a), it would be possible to train an ANN to approximate the optimal value function q_π. This can be done by simply running a forward pass on the ANN to compute $q_\theta(s,a)$ for a given state-action pair as input, then calculate the loss $J(\theta)$ using the ground truth $q_\pi(s,a)$. The backpropagation algorithm can then be used to calculate the partial derivatives w.r.t the loss. The parameters values θ can be updated using an optimizer such as

Adam. The issue is q_π is unknown and no training data is initially available for the training procedure just described. Instead, deep reinforcement learning uses online training procedures closely resembling the TD-learning methods introduced in section 2.3.4 for function approximation.

Bootstrapping

Following TD-learning, as the optimal value function q_* (or q_π in the previous section) is not available, it is possible to instead bootstrap using the current estimate of the value function [69]. Recall that bootstrapping for the value function is when the updated current estimate of the optimal value function for a state-action pair is partially based on new experience data, and the current estimated value of the next selected state-action pair when following a given policy. An example of a method which uses bootstrapping is the off-policy method Q-learning presented in equation 2.20. Equation 3.9 gives what is known as the TD error (δ_t) for Q-learning at time step t. It is called an error as it gives the difference between the current estimate $Q(S_t, A_t)$ and the better estimate $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$. Notice that the TD error is an estimate that has been made at time step t, meaning the error depends on the next state and the next reward which can both change across time steps.

$$\delta_t = \left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a)\right) - Q(S_t, A_t) \tag{3.9}$$

It is this TD error that can be used to form the loss function for an ANN to approximate the optimal value function, as shown in equation 3.10. The gradient of this loss function can be calculated by equation 3.11 which is used with an optimizer for updating the parameters θ during learning. This is the deep reinforcement learning rule which Neural-Q's loss function is based on, hence the name Neural-Q.

$$J(\theta) = \left(R_{t+1} + \gamma \left[\max_a \hat{q}_\theta(S_{t+1}, a)\right]\right) - \hat{q}_\theta(S_t, A_t) \tag{3.10}$$

$$\nabla_\theta J(\theta) = \left(\left(R_{t+1} + \gamma \left[\max_a \hat{q}_\theta(S_{t+1}, a)\right]\right) - \hat{q}_\theta(S_t, A_t)\right) \nabla_\theta \hat{q}_\theta(S_t, A_t) \tag{3.11}$$

Where:

∇_θ = Gradient w.r.t θ
$\hat{q}_\theta$ = The current approximation of the value function given by an ANN
$[.]$ = Stop gradient, meaning the value is taken as just a scalar input during backpropagation

Unfortunately due to the use of function approximators for TD-learning methods, the learning rule is no longer proven to converge on the optimal policy q_*/q_π. That said, in general these methods perform well in practice for appropriate use cases [43, 45]. They also generally learn faster than the other option of using Monte Carlo reinforcement learning with function approximation [66].

3.2.2 Deep Reinforcement Learning Motivation

Unlike tabular TD-learning methods, we have created a deep reinforcement learning method to approximate the incident radiance function over the continuous set of possible locations in a scene in a discrete set of incident directions $\omega_k \ \forall k = 1, ..., m$. Then, when sampling directions to continue light paths during path tracing, the incident radiance estimate from each discretized direction is normalized, forming a distribution known as the radiance distribution, see section **??**. Therefore, a good approximation of the radiance distribution at any point can be used for importance sampling directions for light path directions during Monte Carlo path tracing in order to reduce image noise (see section 2.2.2).

Unlike the Expected Sarsa tabular approach, function approximation has the ability to generalize the value over state-action pairs. In other words, the tabular approach requires a single entry for every possible state-action. So, if an update is made to a state-action pair, it affects that state-action pair value alone. In the case where there are many states (potentially infinitely many), state-action pairs may not be updated often due to the number of them. This means they will be updated infrequently, making learning slow. Furthermore, it may not even be possible to store such a number of state-action pairs. Using an ANN or any function approximator allows one to model a potentially infinite state space and generalize on seen data value to approximate the value of unseen state-action pairs. This applies well to rendering, as the number of unique positions in the scene is technically infinite. However, this comes at the cost of updating weights in the ANN can affect the valuation of multiple states, making it impossible to predict the optimal value for every state-action pair, unlike the tabular case. Leading to proofs for convergence on the optimal value function $q_\pi(s, a)$, to break down for the function approximation case.

Previously, we mentioned that we chose an ANN as the function approximator for the Neural-Q path tracer, due to its ability to model non-linear functions well in the presence of a large amount of training data. When approximating the incident radiance $L_i(x, \omega)$ for all possible positions x in the scene, the radiance distribution at any point in the scene can be a non-linear function due to the way light paths randomly reflect off complex geometry in a scene. Therefore, learning the radiance distribution for all points in a arbitrary scene requires a non-linear function approximator. In terms of training data, the rendering process samples light paths to approximate radiance which can be used for training. This means as much training data as needed can be generated during the rendering process.

3.2.3 Deep Q-learning for Light Transport

In order to use deep reinforcement learning to learn the incident radiance function, we first derive a loss function for which the ANN will be trained with. However, instead of using Expected Sarsa, we derive our loss function from the deep Q-learning rule introduced in Equation 3.10. This choice was made primarily due to its proven success when used to approximate the optimal value function for a variety of Atari games [45].

Once again, the rendering equation from section 2.2.1 states the radiance in an outgoing direction ω from point x is equivalent to the emitted radiance in the direction plus the reflected radiance in the direction:

$$L_o(x, \omega) = L_e(x, \omega) + \int_\Omega L_o(h(x, \omega_i), -\omega_i) \cdot f_r(\omega_i, x, \omega) \cdot \cos(\theta_i) d\omega_i$$

By matching terms and adapting the rendering equation to the deep Q-learning loss function in Equation 3.10, the loss function for training an ANN to learn the incident radiance can be found by equation 3.12.

$$\triangle \hat{q}_\theta(x, \omega) = \left(L_e(y, -\omega) + \left[\max_{\omega_i} (\hat{q}_\theta(y, \omega_i) f_s(\omega_i, y, \omega)(\omega_i \cdot \mathbf{n})) \right] \right) - \hat{q}_\theta(x, \omega) \tag{3.12}$$

Where:

$\hat{q}_\theta(x, \omega)$	= Approximated incident radiance function from the ANN at (x, ω) using parameters θ
$\triangle \hat{q}_\theta(x, \omega)$	= The loss/error of the ANNs approximation of $q_\pi(x, \omega)$
θ	= Current parameter values of the ANN
ω_i	= The direction where the incident radiance is highest on y
$(\omega_i \cdot \mathbf{n})$	= Equivalent to the cosine of the angle between the normal $\mathbf{n}$ and direction vector ω_i
$(\cdot)$	= Denotes the dot product

3.2.4 Artificial Neural Network Architecture

With the loss function defined in equation 3.12 for learning the incident radiance function, a suitable ANN architecture must be developed for approximating the $\hat{q}_\theta(x, \omega)$. At first one might think the ANN would take a single position $x \in \mathbb{R}^3$ and an incident direction $\omega \in \mathbb{R}^2$ as input for a forward pass to calculate the approximated incident radiance under parameters θ. This way the ANN could take any arbitrary incident direction ω to calculate the incident radiance on position x. However, when the incident radiance needs to be estimated for each discrete direction ω_k in the hemisphere around a position x for all m directions, m forward passes must be made through the ANN. A single light path may include hundreds of reflections before intersecting with a light source, meaning potentially thousands of forwards passes would be needed for a single light path. Being conservative, imagine every light path reflected 30 times before intersecting with a light source, if we only sample $N = 16$ light paths per pixel for a $512x512$ image. The total number of forward passes for using only $m = 16$ different possible directions to continue a light path in is over a billion.

To avoid this situation, we followed the technique proposed for learning to play Atari games in [45]. Which in the context of reinforcement learning is to give the ANN as input the agents state, then a forward pass of the network gives the value of each state-action pair for the input state. More formally, given a state x, the ANN will evaluate $\hat{q}_\theta(s, a_k)\ \forall k = 1, ..., m$. Applying this to incident radiance function, a forward pass computes the incident radiance from the set of discrete directions $\omega_k\ \forall k = 0, ..., m$. In other words, a single forward pass of the ANN gives all the required information needed to importance sample a direction to continue the light path in upon intersection with a surface.

With the current process proposed, only a single 3-dimensional point x will be passed into the ANN to infer the incident radiance from directions $\omega_k \ \forall k = 0, ..., m$. The 3-dimensional position given is in the world coordinate system of the scene, hence it gives no information regarding where that point is relative to the geometry in the scene. In terms of reinforcement learning, the state is said to be only partially observable [66]. So, one of our key innovations was to instead pass the ANN all vertices in the scene converted into a coordinate system centred at the position x as input, rather than passing the position x itself. Without this modification, we found the ANN could not learn the incident radiance function using the learning rule in equation 3.12 for any scene. Formally, for any position $x \in \mathbb{R}^3$ in the scene, where the set of all vertices in the scene are denoted as $\mathbf{v} = (v_0, v_1, ..., v_n)$, where $v_i \in \mathbb{R}^3 \ \forall v_i \in \mathbf{v}$, an input vector $\mathbf{v^x}$ is formed:

$$\mathbf{v^x} = (v_0^{\{x\}}, v_1^{\{x\}}, ..., v_n^{\{x\}})$$
$$\text{where: } v_i^{\{x\}} = v_i - x \quad \forall i = 0, ..., n$$

This decision was inspired by [45], where a state in a game of Atari is represented by the raw image of the game state. This encodes information regarding where the player is relative to objects around it in 2-dimensions. Whereas $\mathbf{v^x}$ encodes where the current location x is, to all objects around it in 3-dimensions. The full process of approximating the incident radiance at a position x from discrete directions ω_k by using our ANN architecture is illustrated in Figure 3.5.

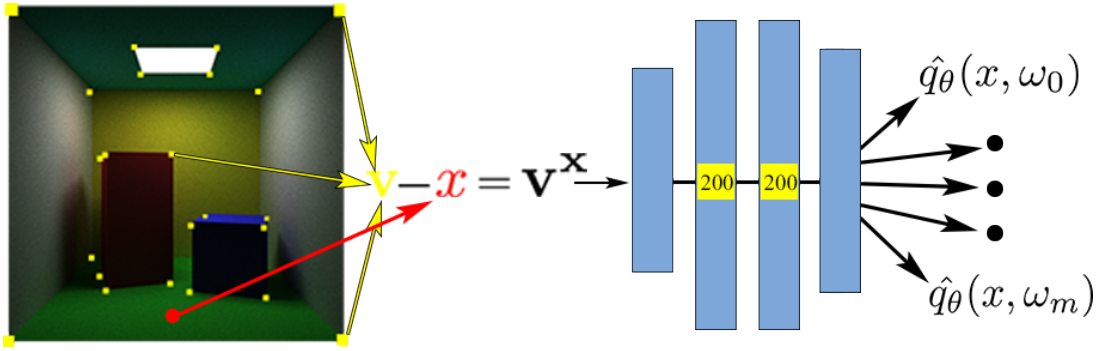


Figure 3.5: An illustration of the process for a forward pass on the ANN used for the Neural-Q path tracer. Starting with the scene on the left, all vertices in the scene (yellow points) are converted into a coordinate system relative to the red point x, producing $\mathbf{v^x}$. This is passed to the input layer of the ANN which then computes a forward pass through the two hidden layers fully connected layers, each with a width of 200 neurons. The output layers width is equal to the number of discrete incident directions m used in the approximation of the incident radiance function. Each output represents the approximated incident radiance on the position x for every discrete incident direction $\hat{q}_\theta(x, \omega_k) \ \forall k = 1, ..., m$, using parameter values θ.

As shown in the illustration in figure 3.5, the ANN consists of an input layer consisting of n neurons, where n is the the total number of vertices in the scene. The input is then all vertices in the scene converted into a coordinate system around the intersection position x, $\mathbf{v^x}$. Then there are 2 hidden fully connected layers, where each hidden layer is followed by a rectifier of non-linearity activation function (ReLU) [50], ensuring the network can learn a non-linear function. The output layer consists of m neurons to output the approximated incident radiance from every incident direction represented by discretized hemisphere around the intersection point x. The choice to use only two fully connected hidden layers was made to reduce time spent on computing forward passes and training the network during the rendering of an image, whilst still having two hidden layers allows the ANN to learn the 5-dimensional incident radiance function $L_i(x, \omega)$, where $x \in \mathbb{R}^3$ and $\omega \in \mathbb{R}^2$. However, scenes with more complex geometry than those experimented on in chapter 4 may require more hidden layers and/or neurons in hidden layers to accurately approximate $L_i(x, \omega)$ [60].

Training the ANN is where the main difference arises compared to the supervised learning case. Recall from section 3.2.1, in the reinforcement learning case there is no large training data set available for learning the optimal value function directly. Hence, bootstrapping is used in TD-learning methods in order to learn online by continually updating the estimate of the optimal value function as data is received. This is otherwise known as learning from experience [66]. The loss function in equation 3.12

does exactly that, where the loss is described in terms of some immediate reward which is the emitted radiance L_e term, and partially on the current estimate of incident radiance from direction ω:

$$\text{TD Error: } \triangle\hat{q}_\theta(x,\omega) = \left(L_e(y,-\omega) + \left[\max_{\omega_i}\left(\hat{q}_\theta(y,\omega_i)f_s(\omega_i,y,\omega)(\omega_i\cdot\mathbf{n})\right)\right]\right) - \hat{q}_\theta(x,\omega)$$

So to compute this loss, an initial forward pass on the network must be made to determine the value of $\hat{q}_\theta(x,\omega)$, then a second pass must be made to calculate $\max_{\omega_i}\hat{q}_\theta(y,\omega_i)$. All other terms are a result of experience and are already calculated as part of the path tracing algorithm. After the loss is calculated, backpropagation is used to compute the partial derivative of the loss with w.r.t to each parameter. Then another pass can be made through the network to update the networks weights using the partial derivatives calculated in the previous step with an optimizer. In our case we use the Adam optimizer to benefit from its use of individual learning rates for each parameter, which are adapted during learning [39]. While there is no guarantee, in practice over many iterations of the learning procedure described, the ANNs estimate of incident radiance for a discrete set of directions ω_k for any point x in the scene move towards a local optimum [69].

3.2.5 Neural-Q Path Tracing Algorithm

With our ANN and a method for training it to learn the the incident radiance function specified, we now present our full Neural-Q algorithm which uses the ANNs predictions for importance sampling directions to continue light paths in. Algorithm 3 renders a single image using N SPP to compute the colour estimate of each pixel by path tracing. Similarly to the Expected Sarsa path tracer detailed in algorithm 2, the Neural-Q path tracer estimates the radiance distribution at the intersection point of a light path and samples a direction to continue the light path in proportional to this distribution. This radiance distribution is also used as the PDF for MC integration in equation 2.13 to reduce the variance in the approximation of the outgoing radiance L_o, leading to a reduction in image noise. Neural-Q also trains online, meaning it progressively reduces noise in the image as it simulates more light paths. Once again the ρ_i term is the PDF over the hemisphere of directions Ω at intersection point x to continue a light path in, evaluated for the sampled direction ω_i. This can be calculated using equation 3.6. There are four additions made to default path tracing presented in algorithm 1 to create the Neural-Q path tracer.

Addition 1: Minibatching

In order to yield a smoother sampled gradient for training, a minibatching method is used where the gradient information for light paths is accumulated over a batch to train the network with. This was proven to be succesful for a range of physical control tasks in [43]. While algorithm 3 iterates through the batches to clearly represent the steps the algorithm takes, in practice we performed all computation within the batch loops in parallel using a GPU.

Addition 2: Sample direction using decaying ϵ-*greedy* policy

The Neural-Q path tracer follows a decaying ϵ-greedy policy (see section 2.3.5) to either exploit by sampling one of the discrete directions proportional to the estimated radiance distribution at the intersection position. Or explore, by randomly sampling one of the discrete directions at the intersection point. This choice is made depending on the value of ϵ. Where $\epsilon \in [0,1]$, a random number $r \in [0,1]$ is sampled. If $r > \epsilon$ then the so called *greedy strategy* is chosen. This is where a direction is sampled proportional to the radiance distribution formed by normalizing the m discrete Q-values produced by a forward pass on the network using $\mathbf{v^x}$ as input. Where each Q-value represents a discrete direction in the discretized hemisphere surrounding the intersection point x of the light path. Similarly to our implementation of the Expected Sarsa method, the hemisphere is once again discretized using the adaptive quadrature method in [63]. In the case of exploitation, a random direction ω_k is sampled from the discrete set directions ω_k for $i = 1, ..., m$ for the adaptive quadrature at the light paths intersection point.

Addition 3: Training the ANN

This modification trains the ANN to improve its approximation of the incident radiance at any point x in the scene, for the set of discrete angles ω_k in the adaptive quadrature centred at x oriented at the surface normal at x. To do so, the algorithm computes the loss function in 3.12, by running a forward pass on the ANN to get the approximated incident radiance in the discrete directions $\omega_k\ \forall k = 1, ..., m$ for

both the position x, as well as the next intersected position y. The loss is then used to train the network using the process described in section 3.2.4.

The policy followed for selecting a direction to continue a light path in is different to that of selecting a directions for training with in $\hat{q_\theta}(y, \omega)$ for bootstrapping. The direction for training with is instead chosen according to the direction with the highest incident radiance from y by following equation 3.12. For this reason, the Neural-Q path tracer uses off-policy learning.

Whilst a replay buffer was not used in the Neural-Q path tracer, what it hopes to achieve is partially covered by the Neural-Q path tracing algorithm. A replay buffer was originally introduced for deep Q-learning in [45] for training an AI agent to play a variety of Atari games by learning a good approximation of the optimal value function using deep Q-learning. The replay buffer strored transitions between state-action pairs and the corresponding reward received $e_t = (S_t, A_t, R_t, S_{t+1})$. The replay buffer is then sampled from to make up part of the minibatch to train the network with. This is known as experience replay, and is used to avoid giving the network highly correlated data to train on as a result of learning online. Training online means that taking one action in a particular state may affect the probability of selecting another action in the next state. ANNs make the assumption that the training data is independently and identically distributed (i.i.d) [66], which was not the case for consecutive states in Atari games. Therefore, by sampling from a replay buffer, the variance in consecutive weight updates is reduced as the updates are no longer highly correlated with one another compared to using standard Q-learning [66].

Rather than storing transitions in a buffer to make up a minibatch for training, a batch of rays are continually traced and trained together in the Neural-Q path tracer. This avoids strong correlations between training iterations, reducing the variance in consecutive weight updates. However, it is likely there is still some correlation in the initial consecutive training iterations for light paths sampled from spatially local pixel positions in the image plane, so using some form of replay buffer like that suggested in [49] may further reduce the variance in consecutive weight updates.

Addition 4: Decaying decaying ϵ-greedy

As part of the decaying ϵ-greedy policy, the value of ϵ is decreased after 1 sampled light path is computed through every pixel in the image. Meaning, according to algorithm 3, this represents a single epoch. This is a reasonable point to decay ϵ, as for an average light path length of 30, for each sampled pixel in a $512x512$ image with a minibatch size of 1024. Approximately $7,680$ training updates are made on the network. As discussed in section 2.3.5, the decaying ϵ-greedy policy makes sure the path tracer at first prioritises exploration of which directions contribute the most radiance. As training progresses, the path tracers policy alters more towards exploiting by sampling directions to continue light paths in based on the current estimated radiance distribution at their intersection points. This behaviour is desirable, as if we focused on exploitation too early, the path tracer may not sample seemingly unfavourable directions in the short term, which after more reflections off surfaces lead to a large amount of incident radiance to the point.

Algorithm 3: Neural-Q forward path tracer. Given a camera position, scene geometry, epsilon and epsilon decay, this algorithm will render a single image. Where N is the pre-specified number of sampled light paths per pixel. The algorithm trains the ANN online to progressively reduce noise in consecutive rendered images, by improving its approximation of the incident radiance function for the scene.

Function *renderImage(camera, scene, decay, ϵ)*
- $ANN = \text{loadANN}()$
- **for** $i = 1$ **to** N **do**
  - /* Addition (1) */
  - **for** $b = 1$ **to** *Batches* **do**
    - **for** $k = 1$ **to** *BatchSize* **do**
      - $p \leftarrow \text{getPixel(b,k)}$
      - $ray \leftarrow \text{initializeRay}(p, camera)$
      - **for** $j = 1$ **to** ∞ **do**
        - $(y, \mathbf{n}, L_e) \leftarrow \text{closestIntersection}(ray, scene)$
        - **if** $j > 1$ **then**
          - /* Addition (2) */
          - $(\omega_i, \rho_i, f_s) \leftarrow \text{sampleRayDirEpsilonGreedy}(\epsilon, y)$
          - /* Addition (3) */
          - $\hat{q}_\theta x \leftarrow ANN.getQValue(ray.x, ray.\omega, scene)$
          - $\hat{q}_\theta y \leftarrow ANN.getMaxQValue(y, scene)$
          - $\triangle \hat{q}_\theta x \leftarrow L_e + (\hat{q}_\theta y \cdot f_s \cdot (\omega_i \cdot \mathbf{n})) - \hat{q}_\theta x$
          - $ANN.\text{train}(\triangle \hat{q}_\theta x)$
        - **end**
        - **if** *noIntersection*(y) **or** *areaLightIntersection*(y) **then**
          - $ray.throughput \leftarrow ray.throughput \cdot L_e$
          - $\text{updatePixelColourEstimate}(p, ray.throughput)$
          - **break**
        - **end**
        - $ray.throughput \leftarrow ray.throughtput \cdot f_s \cdot (\omega_i \cdot \mathbf{n})/\rho_i$
        - $ray \leftarrow (y, \omega_i)$
      - **end**
    - **end**
  - **end**
  - /* Addition (4) */
  - $\epsilon \leftarrow \epsilon - decay$
- **end**

end

Chapter 4

Comparing the Expected Sarsa and Neural-Q Path Tracers

Recall, that our main goal is to find out whether learning the incident radiance function for the continuous set of locations in a scene is more advantageous for importance sampling in path tracing, compared to learning the function for a discrete set of locations. To the best of our knowledge, there are no other studies which have attempted to make this assessment. Therefore, we have decided to analyse both the Expected Sarsa and Neural-Q path tracers in the following areas which we believe to show the most important differences:

- Assess the noise reduction and the type of noise which remains in the rendered images.
- Assess the accuracy of the incident radiance approximation for scenes.
- Find the number episodes required for learning to approximately converge.
- Analyse the memory usage and compute time.
- Assess the ease of setting hyperparameters for rendering an arbitrary scene.

4.1 Experimental Setup

We built a path tracing renderer from scratch which supported algorithms 1, 2, 3 and was used to produce all rendered results seen in this thesis. The rendering engine used only OpenGL Mathematics library [1] for various operations that are common in the rendering pipeline, SDL2 [2] for displaying rendered images, and the Dynet neural network library [51] for the Neural-Q path tracer implementation. Algorithms 1, 2, 3 were all accelerated on an NVIDIA GPU by using the CUDA Toolkit [52] to retrieve experimental results in an acceptable time. The reasoning for choosing Dynet over more commonly used neural networks libraries such as Tensorflow [3], was due to its ability to be easily compiled with the CUDA `nvcc` compiler and it had a well documented C++ API. This was a requirement for the Neural-Q path tracer, as tracing light paths, ANN inference, and ANN training are all performed on a NVIDIA GPU via C++ API calls. All results were produced using a machine with an Intel i5-8600K CPU, NVIDIA 1070Ti GPU and 16GB of RAM installed.

We developed four different scenes using Maya [7] and created a custom object importer to import the scenes into our path tracing engine. A reference image of all four scenes is given in the very right column in figure 4.1, which have been rendered with 4096 sampled SPP using the default path tracing algorithm 1. They are known as reference images as they have minimal visible noise, meaning the MC approximations for each pixel's colour value have approximately converged. Due to path tracing accurately modelling physical light transport (see section 2.2), these images are the ideal case for which all other path tracing algorithms should aim to produce as closely as possible, with the smallest number of SPP possible. Each rendered image from any of the techniques has been rendered at a resolution of 720X720.

4.2 Assessing the reduction in image Noise for Monte Carlo Path Tracing

Importance sampling techniques are used in path tracing to ultimately reduce image noise using the same number of SPP. Therefore, one of the most important points to assess is if our Neural-Q path tracer is able to reduce image noise to the same extent as the Expected Sarsa path tracer when using the same number of SPP.

4.2.1 Quantifying the Reduction in Image Noise

To quantify the amount of noise within images rendered by a default forward path tracer, the Expected Sarsa path tracer, and the Neural-Q path tracer, we will use the Mean Absolute Percentage Error (MAPE) given in equation 4.1 [49].

$$M = \frac{1}{P} \sum_{i=0}^{P-1} \left| \frac{A_i - F_i}{A_i} \right| \tag{4.1}$$

Where:

P = The total number of pixels in the image
A_i = The ith pixel value in the reference image
F_i = The ith pixel value in the image whose noise is being quantified

The MAPE value can then be used to quantify the average difference between pixel values from the reference image to each image rendered by the three path tracing algorithms with the same number of SPP. Therefore, a lower MAPE score is more desirable for a rendered image, as it means the image has a lower amount of noise. The MAPE score for images rendered by the three different path tracing algorithms for four different scenes is shown in 4.1. Each one of the four different scenes have been designed to exhibit different properties which affect light transport simulation. These can be summarised as:

- **Shelter** - A large scene which includes three different area lights, each with their own blockers. This makes it difficult to learn the contribution of radiance from each of the three sources to a given point in the scene. The scene is a sealed room, meaning all light paths will intersect with an area light as the number of reflections tend to infinity.
- **Cornell Box** - A ubiquitous scene to test the accuracy of a rendering techniques approximation of global illumination. The scene is small and includes two light blockers. The wall in the directions of the camera is missing, meaning there is a void which causes light paths to potentially never intersect with an area light.
- **Complex Pillars** - A scene which poses a true challenge to estimating the incident radiance on any point, due to the number of light blockers. The scene is sealed and the left, right, bottom, and top walls are all made to absorb most of the light incident on them. Meaning, light paths must navigate through the pillars to illuminate the back wall and the pillars themselves.
- **Door Room** - The door room is a sealed scene which is almost entirely lit by indirect illumination. Light paths must reach the light before their contribution to the final image becomes negligible.

From the MAPE score calculated for each rendered image of the four scenes using the three different path tracers, it is clear that the default forward path tracer in algorithm 1 is inferior for a fixed number of SPP, compared to the path tracers which leverage the power of importance sampling in light path construction. Meaning, the Expected Sarsa and Neural-Q algorithms have both learnt an approximation of the incident radiance distribution function $L_i(x, \omega)$ for each scene, such that it is closer to normalizing the numerator in the Monte Carlo integration computed by path tracing (see equation 2.13), compared to a uniform PDF. This causes a reduction in variance for the estimate of a pixel's colour value by using the two path tracers, leading to a lower amount of image noise within the renders displayed in figure 4.1.

The MAPE scores for the Neural-Q path tracer are significantly lower for the `Shelter` and `Complex Pillars` scenes compared to the Expected Sarsa algorithms, and a tie is essentially reached for the `Door Room` scene. Whereas, the Expected Sarsa clearly beats the Neural-Q path tracer for MAPE score on the `Cornell Box`. Based on these MAPE results, the Neural-Q path tracer outperforms the Expected Sarsa

path tracer for reducing image noise in renders of larger scenes with more complex geometry and a greater number of area light sources, while using less memory. The memory usage of the methods is discussed in detail in section 4.5, where the actual amount of memory used to produce each of the rendered images from both the Neural-Q and Expected Sarsa method are given in table 4.1.

For smaller scenes with less complex geometry and a lower number of area light sources, the Expected Sarsa algorithm performs comparably well if not better. Our hypothesis behind why this is, is that the reduction in noise is largely due to the Neural-Q's ability to generalize the approximation of the incident radiance distribution at any position in the scene, and that this is more important for larger and more complex scenes. We will discuss this in more detail within sections 4.2.2 and 4.4.

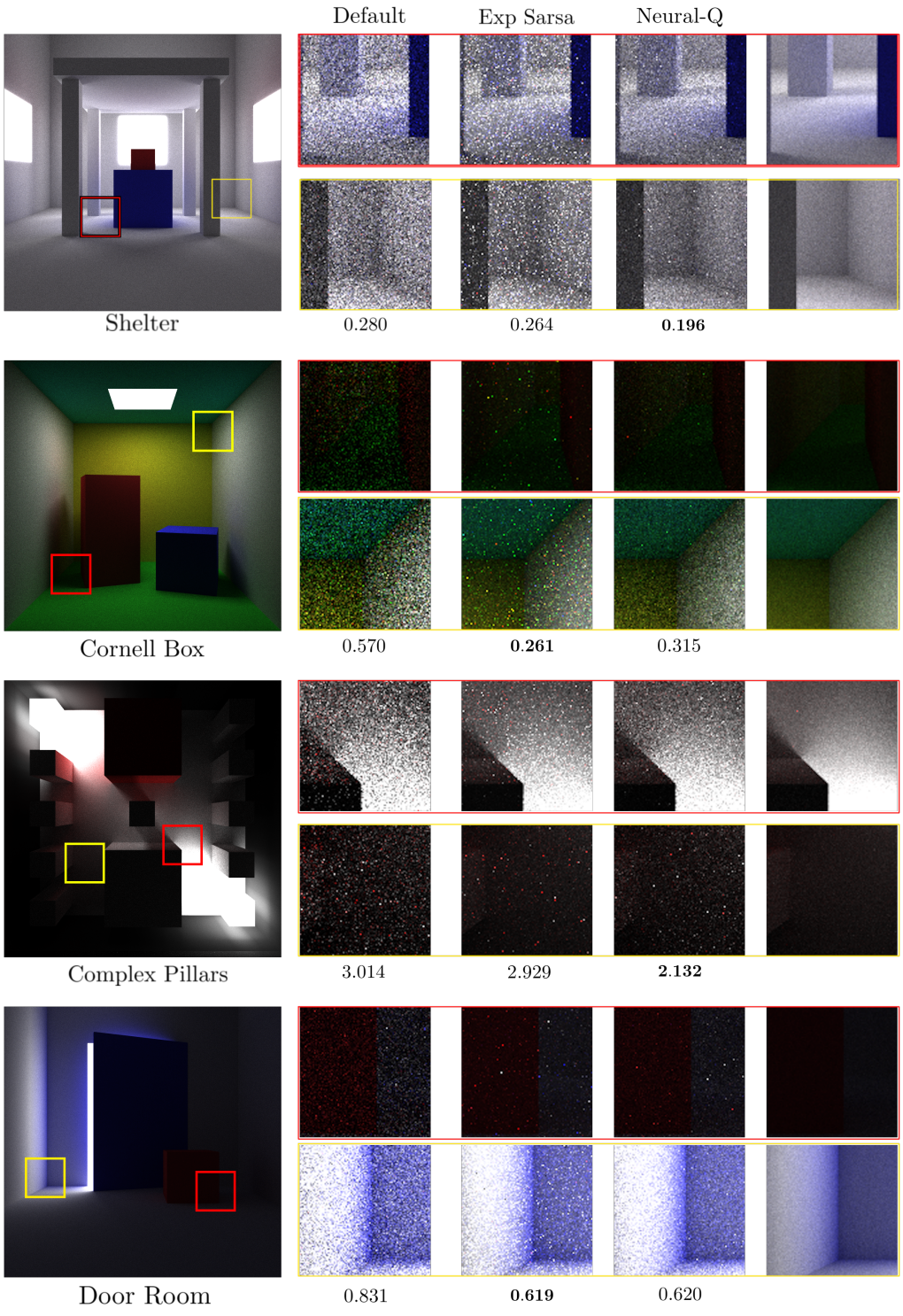


Figure 4.1: A comparison of the default forward path tracer, Expected Sarsa path tracer, and the Neural-Q path tracer rendered image noise for four different rendered scenes. All images were rendered with 128 SPP, apart from the reference images in the very right column which use 4096 SPP. The score under each column in an image row corresponds to the MAPE score for each path tracing algorithm for the particular scene.The Neural-Q and Expected Sarsa algorithms both used 144 discrete directions in each discretized hemisphere to estimate the incident radiance distribution at a given point. The Neural-Q path tracer used the ANN architecture described in 3.2.4 for all fours scenes with a decaying ϵ-greedy policy starting at $\epsilon = 1$, with a decay of $\delta = 0.05$ every epoch. The Expected Sarsa path tracer used just enough Irradiance Volumes (which varied depending on the scene) to facilitate a significant reduction in image noise in all four renders, such that it was competitive with the Neural-Q path tracer. See appendix A for the full images produced by all three path tracers.

4.2.2 A Closer Inspection of Pixel Error Values

By visual observation of the rendered images 4.1, the type of noise present in the Expected Sarsa and Neural-Q path tracer's renders are quite different. In particular, the noise resulting from the Expected Sarsa path tracer are pixels with very high RGB values compared to that of their neighbours, which are commonly referred to as 'fireflies' [17]. Whereas the noise present in the Neural-Q renders is more subtle, however its general presence can be seen when comparing its renders to that of the reference images. In order to investigate this further, we have provided histograms in figure 4.2 for the frequency of average RGB pixel error values of the rendered image compared to the reference image. Essentially, the greater the average RGB error for a particular pixel, the higher the amount of noise present in the pixel's colour estimate as a result of the path tracing algorithm used. Note, the histograms use a log scale, as the vast majority of pixel values when using 128 SPP for both algorithms have a low error, so the log scale allows us to compare the trend of high error estimates within the image.

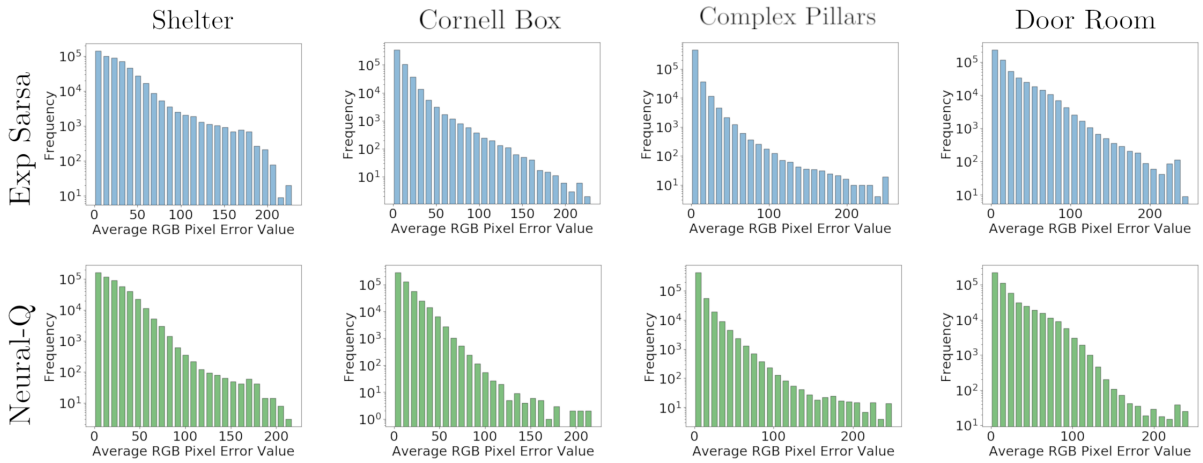
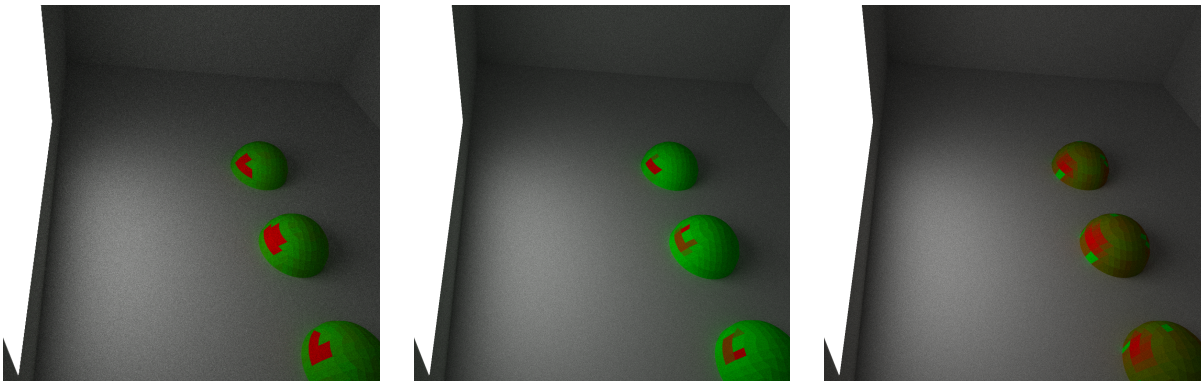


Figure 4.2: Histograms for the average RGB pixel error values for all four rendered scenes using both the Expected Sarsa and the Neural-Q path tracers. Where the average RGB pixel error value is the average difference in all RGB colour channels between a pixel in the reference image, and the corresponding pixel in the image rendered by either the Expected Sarsa or Neural-Q path tracers. The max average RGB pixel error value is 255, which corresponds to the case where the reference images pixels value is $(255, 255, 255)$ whereas the rendered images was $(0, 0, 0)$ or vice versa. The histograms are calculated using the rendered images presented in figure 4.1.

By comparing the tail end of the two rows of histograms, the reasoning behind the fireflies within the Expected Sarsa renders becomes clear. That is, on average the number of high average RGB error pixels is greater for the Expected Sarsa path tracer compared to the Neural-Q path tracer, which are the direct cause of fireflies. This validates the visual observation of fireflies in the renders shown in 4.1. For example, the `Door Room` scene render from the Expected Sarsa path tracer has many bright white and red pixels compared to the reference images, which corresponds to the high number of the large error values within the histogram.

The question now is, why are there more pixels with high error values in the Expected Sarsa path tracer renders compared to those produced by Neural-Q? To answer this question, first recall from section 2.1.2 the closer the PDF comes to normalizing the function being integrated, the lower the variance in the Monte Carlo integral approximation. However, if the PDF used has a different shape to the function, it can actually increase the variance in the Monte Carlo approximation compared to using a uniform PDF. To relate this concept to learning the incident radiance function $L_i(x, \omega)$ for importance sampling in path tracing, if the error in the approximated incident radiance function is high for a scene, the noise in the approximated pixel values will be higher. This is because a normalised version of the approximated incident radiance function for a point is used to importance sample a direction to continue a light path in. More specifically, the incident radiance distribution at the intersection point of a light path is used to importance sample a direction to continue the light path in. Recall, for Expected Sarsa, this incident radiance distribution is calculated by normalizing $Q(x, \omega_k)\ \forall k = 1, ..., m$. Whereas for the Neural-Q it is calculated by normalizing $\hat{q_\theta}(x, \omega_k)\ \forall k = 1, ..., m$. Therefore, a comparison needs to be made between the learned incident radiance functions $Q(x, \omega_k)$ and $\hat{q_\theta}(x, \omega_k)$ to understand why the Expected Sarsa

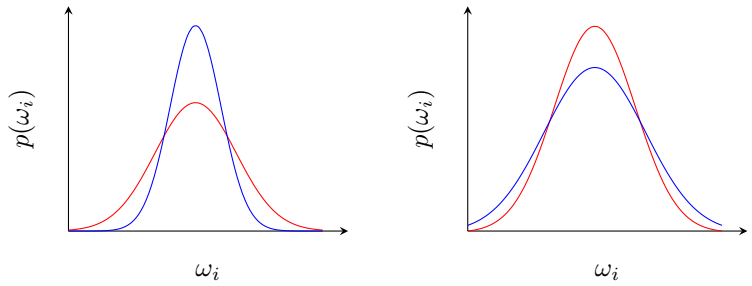
and Neural-Q pixel error values are different.



(a) Ground Truth (b) Expected Sarsa (c) Neural-Q

Figure 4.3: Visualisations of the incident radiance distribution for three different positions in a scene. The incident radiance is represented by placing an adaptive quadrature at each point with 144 different sectors, each representing an incident direction which the incident radiance has been calculated at. The positions are in front of a single large area light within a simple scene. (a) presents the ground truth incident radiance distribution, which has been calculated by firing 4096 sampled light paths per sector and averaging their incident radiance approximation. (b) The approximated incident radiance distribution for the three positions produced by the Expected Sarsa method when trained 1024 SPP, and (c) is the same but for the Neural-Q method trained with 1024 SPP. Due to the single large area light in front of the point, most of the incident radiance on the points will come from directions spanning from the area light on the left.

As previously explained incident radiance function $L_i(x, \omega)$ is a 5-dimensional function as $x \in \mathbb{R}^3$, $\omega \in \mathbb{R}^2$, making it difficult to visualize. Therefore, in figure 4.3 a visual representation of the true incident radiance distribution, along with the Expected Sarsa and Neural-Q approximated distributions for three points in the scene are given. The scene is simple, it has no light blockers and there is a single area light from which the majority of incident radiance on the three points comes from. Meaning, it clearly presents the differences in the shape of the radiance distribution formed by normalising the incident radiance values approximated by the two methods at a point. The Neural-Q's approximation is clearly closer to true radiance distribution in figure 4.3a, when compared to the Expected Sarsa method. The Expected Sarsa visualisation shows that the learned distribution gives far too much weighting to certain individual directions, making its approximation noticeably inaccurate, even for the simple scene shown. The Neural-Q approximated incident radiance distribution is not perfect, it gives slightly less weighting then it should do to directions facing the light. However it has clearly converged on which directions contribute the most incident radiance.



(a) Expected Sarsa (b) Neural-Q

Figure 4.4: An illustration of the incident radiance distribution for a given point directly in front of the light source shown in figure 4.3 for both the Expected Sarsa and Neural-Q methods. Where $\omega_i \in \Omega$ are discrete incident angles on the point, and $p(\omega_i)$ is the probability density function over incident radiance distribution evaluated at angle ω_i. The red line represents the true incident radiance distribution on the point, the blue represents the corresponding methods approximation of the incident radiance distribution on the point.

To give a better a conceptual understanding of why Expected Sarsa is less accurate compared to Neural-Q for the setting in figure 4.3, an illustration in figure 4.4 of an example incident radiance distribution for an intersection point in front of the area light is given, whereby normalising the incidence radiance prediction over all angles in a hemisphere Ω around an intersection point, a PDF over the predicted incident radiance on the intersection point is formed. The PDF for the Expected Sarsa approach is leptokurtic, meaning it contains an excess of extreme values causing it to incorrectly model the true radiance distribution for the point in the scene. Whereas the more mesokurtic curve for the Neural-Q path tracer more accurately models the true underlying radiance distribution for the point, however it is not perfect hence some noise is still present in rendered images in 4.1 for the Neural-Q path tracer.

This inaccuracy by Expected Sarsa means for some incident angles ω_i which are not at the peak of the approximated distribution will likely have a significantly higher contribution of radiance to a point x than the approximated radiance distribution states they will. This causes the evaulated PDF $pdf(x, \omega_i)$ to be small, and when used to divide the numerator in equation 2.13, the approximated pixel colour value will be much higher than its true value. This presents a severe limitation to the Expected Sarsa's ability to reduce the noise in renders of a scene, as it adds more 'fireflies' to rendered images. As previously discussed in section 3.1.3, the Expected Sarsa algorithm does not necessarily converge on the incident radiance [20], which is exactly what has been shown in figure 4.3.

4.3 From a Discrete to Continuous State Space

As a reminder, a state in the context of Neural-Q learning is an intersection location x of a light path within the scene. An action is choosing a direction to continue the light path in from the intersection point. As discussed in section 3.2.2, the Neural-Q algorithm uses an ANN as a function approximator for learning the incident radiance function $(L_i(x, \omega))$ where the approximation is denoted as $\hat{q_\theta}(x, \omega_k)$ for the discrete set of actions $\forall k = 1, ..., m$, over the continuous space of intersection positions $x \in \mathbb{R}^3$. Whereas Expected Sarsa uses a tabular approach denoted as $Q(x, \omega_k)$, to approximate the incident radiance function at a discrete set of locations in the scene, using $\forall k = 1, ..., m$ discrete directions for each location. Therefore, as the the approximation of $L_i(x, \omega)$ by Neural-Q is made by an ANN, to find the estimated incident radiance for an unseen intersection location $x' \in \mathbb{R}^3$ in direction ω_k, the ANN can use its current parameter values θ to generalize the incident radiance $\hat{q_\theta}(x', \omega_k) \approx L_i(x', \omega_k)$. Whereas the approach of Expected Sarsa is to find the closest Irradiance Volume to the point located at $\bar{x} \in \mathbb{R}^3$ and use its stored incident radiance estimate for the approximation, that is $Q(\bar{x}, \omega_k) \approx L_i(x', \omega_k)$.

We will no discuss how generalization for unseen intersection locations allows the ANN to learn a smooth approximation of the incident radiance for any point in the scene, compared to the tabular method used by Expected Sarsa. Observe figure 4.5. As stated, a single light path is sampled through each pixel using the two different path tracing algorithms. The difference is that the trained Expected Sarsa and Neural-Q path tracers continue the light path at every intersection point in the direction of maximum predicted radiance. Therefore, the goal of 4.5 is to not reduce image noise in the renders of the two scenes at 1 SPP. Instead, it's to visualize for any point in the scene, the accuracy of the two methods approximation for the direction of the highest incident radiance..

For both renders shown in figure 4.5 produced by the trained Expected Sarsa path tracer, there are small distinct darker regions of the image. Each one of these small dark regions represents an Irradiance Volume which has not made a good approximation of which incident direction contributes the highest amount radiance. This creates small distinct areas where the estimation of incident radiance is poor, which is likely to lead to higher levels of image noise upon intersection with these locations during path tracing. On the other hand, a vastly different pattern can be seen for the Neural-Q algorithms renders, where boundaries between regions with a better approximation for the highest incident radiance direction are much smoother than that of the Expected Sarsa's. This is due to the generalization for incident radiance over the continuous set of locations in the scene, as no longer are discrete incident radiance values stored, but instead the parameter values θ of the network encode an approximation of the incident radiance function $L_i(x, \omega)$.

The question now is, does the generalisation over the state space made by Neural-Q actually help to reduce image noise compared to the discretized approach by Expected Sarsa? Unfortunately there is no clear answer to this question from the data we have collected. The reason being, the ANN generalization improves the approximation of $L_i(x, \omega)$ in both the `Shelter` and `Complex Pillars` scenes, whereas it performs equally as well for the `Door Room` scene and worse for the `Cornell Box`. So at best we can conclude the ANNs generalization can be more beneficial for scenes with larger and more complex geometry whilst using less memory, as shown in table4.5. This is represented in figure4.5, where on average

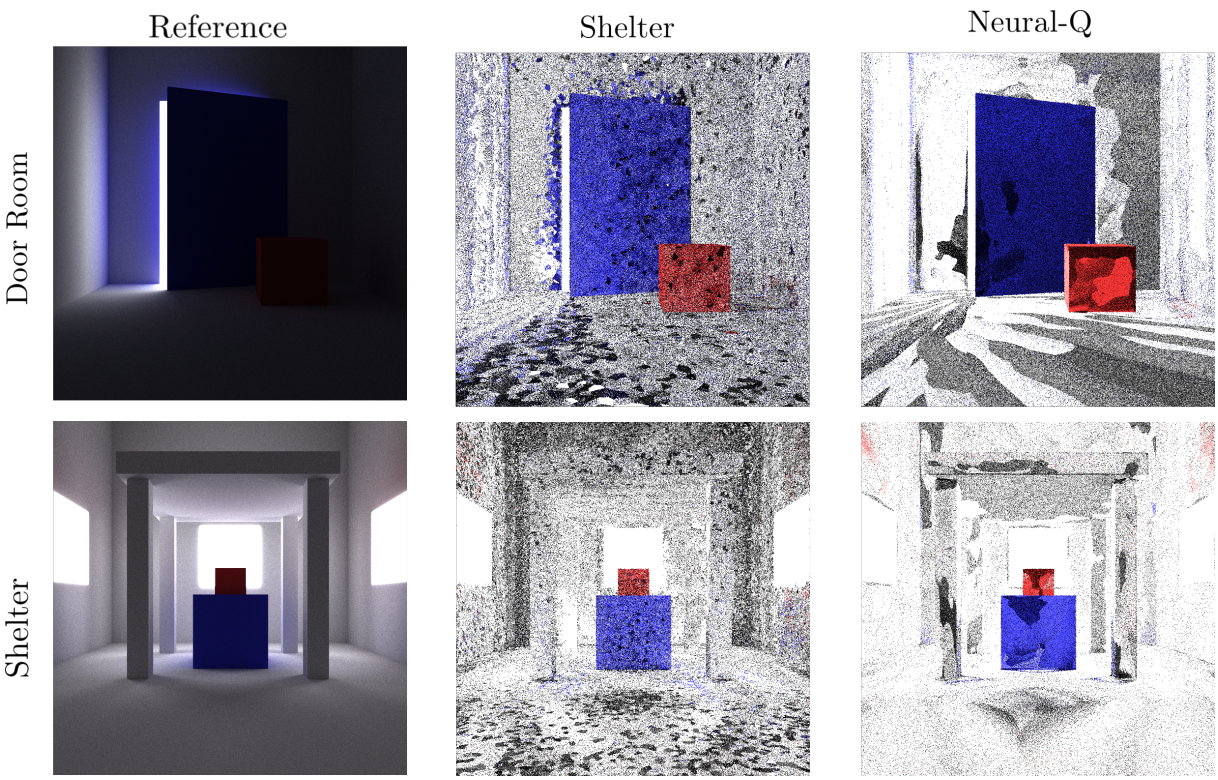


Figure 4.5: Reference Images of the Door Room and Shelter scenes next to a 1SPP render using an Expected Sarsa path tracer and a Neural-Q path tracer, both trained for 100 epochs. The single sample for each pixel in the Expected Sarsa and Neural-Q renders is constructed by reflecting the light path in the direction of highest estimated radiance upon every intersection point, until the light is intersected with.

for the Neural-Q's render of the `Shelter` scene shows a good approximation of the direction which the highest incident radiance is coming from, due to the smooth bright areas of the image. Whereas there are far more dark patches in the Expected Sarsa's render which represents incorrect approximations of the highest incident radiance direction. However, for smaller less complex scenes (`Cornell Box` and `Door Room` scenes), the ANN's approximation quality of incident radiance heavily changes across neighbouring regions in the scene. As a result, figure 4.1 shows Neural-Q's performance against the Expected Sarsa approach is slightly worse for reducing image noise for these scenes. In order to remedy the ANN's poor approximation in such situations, modifications in the ANN's architecture may be required. But this seems to be part of a wider problem which comes with a lack of research into how ANN architecture affects the quality of the approximated incident radiance function $L_i(x, \omega)$ for various types of scenes. Meaning more research is required before ANN's can be stably used to importance sample light paths in Monte Carlo path tracing to reduce image noise for any arbitrary scene. Luckily, some progress has been made in this area very recently, which we will discuss later in section 4.7.

4.4 Convergence

Another important property which has not yet been discussed is the number of SPP required for the Expected Sarsa and Neural-Q path tracers to convergence on their approximation of the incident radiance function $L_i(x, \omega)$. To assess this, the number of *zero contribution light paths* when rendering a single image has been collected, as well as the average number of reflections a light path undergoes before intersecting with a light source, known as the average *path length*. These are both plotted against accumlated rendered images using 1 SPP (epoch). A zero contribution light path is a sampled light path which ends up contributing a negligible amount to a pixel value.

For a scene where a light path is guaranteed to intersect with an area light (sealed scene), as the average path length reduces, the number of zero contribution light paths also decreases. This is due to light paths with a shorter length generally contribute more to the rendered image because of the

rendering equation. Following importance sampling for Monte Carlo integration, the lower the number of zero contribution light paths sampled in path tracing, the lower the image noise [20]. This is shown in figure 4.6, where the average path length curves correlate with the number zero contribution light paths across accumulated frames for the `Shelter` scene, leading to a reduction in noise from the first rendered image to the 100th rendered image for both algorithms using 1 SPP to render each image. Note, a similar pattern was seen for the `Cornell Box`, `Complex Pillars`, and `Door Room` scenes.

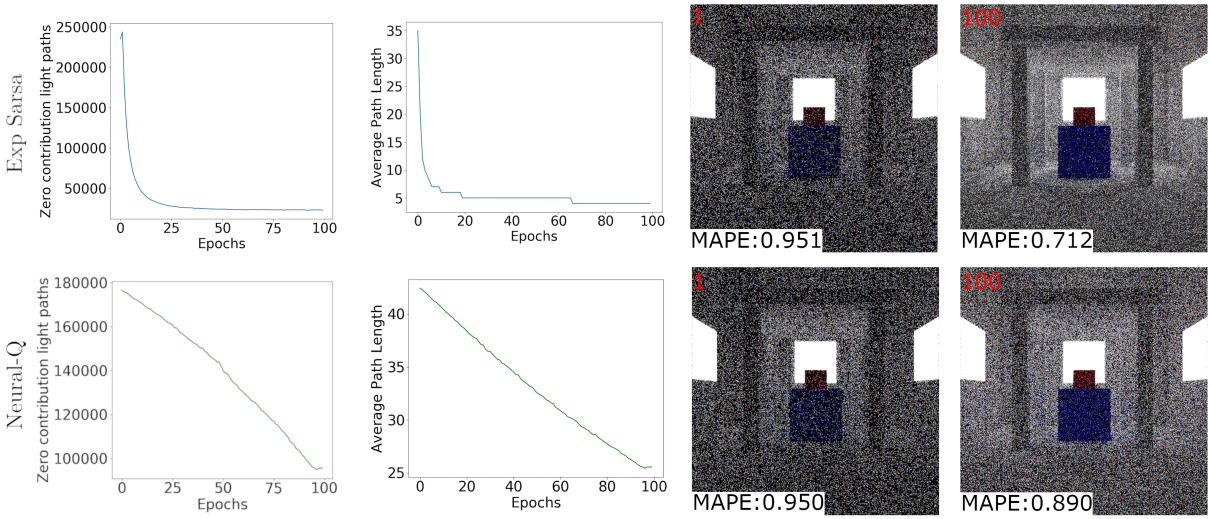


Figure 4.6: Training curves for the average path length and number of zero contribution light paths when rendering the Shelter scene for 100 epochs, as well as the 1st and 100th rendered frames when using 1 SPP for both images. This is shown for both the Expected Sarsa and Neural-Q path tracers. An epoch represents one sampled light path through every pixel in the image. The average path length is the number reflections a light path takes before intersecting with a light source. A zero contribution light path is one which contributes almost zero colour to the final image.

An interesting observation is how both the number of zero contribution light paths and the average path length for the Expected Sarsa algorithm fall far below that of the Neural-Q algorithm, yet the MAPE score for the `Archway` scene is significantly higher for Expected Sarsa in figure 4.1. This leads to the conclusion that the Expected Sarsa path tracer has the potential to further reduce the number of zero contribution light paths compared to the Neural-Q path tracer. However, this alone does not directly reduce image noise. Recall from the previous section 4.2.2 the leptokurtic shape of the incident radiance distribution approximated by the Expected Sarsa method for a given point in the room. Meaning, Expected Sarsa's approximated radiance distribution for this point did not accurately describe the true underlying radiance distribution. Therefore, the PDF used to normalise the MC prediction of a light paths colour estimate for Expected Sarsa caused a higher amount of image noise compared to that of the Neural-Q path tracer.

By observing the curves for the number of zero contribution light paths and average path length for both methods in figure 4.6, it is clear that the Expected Sarsa approach is able to more quickly learn the incident radiance function $L_i(x, \omega)$ compared to the Neural-Q path tracer. However, the almost linear decrease seen for both curves using the Neural-Q method is due to the use of an ϵ-greedy policy. As previously described in section 3.2.4, the ϵ-greedy policy ensures the Neural-Q algorithm initially prioritises exploration (ϵ is high) to find the approximate value of $L_i(x, \omega)$ by uniformly at random sampling directions to continue light paths in. Every time all pixels in the image have had a light path sampled through them, the value of ϵ is decayed to favour exploitation more. This causes the Neural-Q path tracer to importance light path directions based on its current estimate of $L_i(x, \omega)$ more over time. As the decay was set to $\delta = 0.05$ and initially $\epsilon = 1$, at first the Neural-Q path tracer is limited to just randomly sampling directions to continue light paths in. But gradually over time, it favours importance sampling directions which are more likely to lead to a light source. This causes the close to linear downwards trend for the average path length and number of zero contribution light paths, as ϵ is decreased linearly.

4.5 From Memory Bound to Compute Bound

Computational resources utilized by the path tracing algorithms introduced is a very important talking point when considering their potential to be integrated into renders used in industry such as [26, 16, 64]. In particular any bounds imposed on the algorithm from current available hardware are the most important to consider, as rendering algorithms should be designed to give as much power to artists as possible to create and render any arbitrary scene of their choice. A detailed investigation on both highly optimised implementations of the Expected Sarsa and Neural-Q path tracers performance using current available hardware warrants an investigation in itself. Instead for completeness, we present a high level analysis of how both memory usage and compute power limits the path tracers performance.

4.5.1 Memory Usage

The default forward path tracer is compute bound, meaning the only way to make the algorithm faster is by providing more compute power. For example, by parallelizing the rendering process using a GPU with a quicker processor clock speed. On the other hand, the performance of the Expected Sarsa algorithm for reducing image noise has been found to be bound by the amount of memory the algorithm has available to it from the underlying hardware. This is due to the requirement of storing the Q-table in RAM which holds the approximated incident radiance values for a scene.

	Memory (MB)	
Scene	Expected Sarsa	Neural-Q
Shelter	271	30
Cornell Box	44	30
Complex Pillars	300	30
Door Room	66	30

Table 4.1: Additional memory required in Megabytes (MB) to render each of the four scenes shown in figure 4.1 using both the Expected Sarsa and Neural-Q path tracers.

Expected Sarsa

The size of memory used by the Q-table is defined by the number of Irradiance Volumes sampled across the scenes geometry, as well as the number incident radiance values stored for directions $\omega_k \ \forall k = 1, ..., m$ within each Irradiance Volume. We have found the higher the number of sampled Irradiance Volumes the more accurate the incident radiance approximation is for a given intersection point, as shown in figure 4.7. Intuitively this makes sense, as for the incident radiance function $L_i(x, \omega)$, the position $x \in \mathbb{R}^3$ can be infinitely many positions in the scene. Therefore, the more Irradiance Volumes used to store approximated incident radiance values, the closer the approximation will be to the true function. In terms of the algorithm, the average distance from a light paths intersection point to the closest Irradiance Volume decreases as the number of Irradiance Volumes sampled in the scene increases. This becomes clear by observing figure 4.7, where the Voronoi plot of the scene with a lower number of sampled Irradiance Volumes on average has a larger sector size. Meaning, on average the nearest neighbour search will return an Irradiance Volume further away from the intersection point, compared to when more Irradiance Volumes are sampled. Recall that a nearest neighbour search is used to estimate the incident radiance on a point based on the stored values in the closest Irradiance Volume, therefore the closer the Irradiance Volume used for this approximation, the more accurate the approximation of incident radiance on the point will be. However, notice for an over a 45X increase in Irradiance Volumes sampled in the `Complex Pillars` scene, the difference in image noise by both observation and MAPE score is surprisingly small. From this, it seems that the number of Irradiance Volumes used in the Expected Sarsa method scales poorly with the reduction in image noise, even for a more complex scene such as the `Complex Pillars` in figure 4.7.

Another problem with the relationship between increasing memory usage and reduction in image noise, is the cap on available memory imposed by the underlying hardware. If the memory available is too low, the algorithm may not be able to significantly reduce the amount of noise in rendered images. Now, this may not be a problem for most small scenes with a polygon count of around 100, as common commodity graphics cards such as the NVIDIA 1070Ti GPU used for our experiments come with 8GB

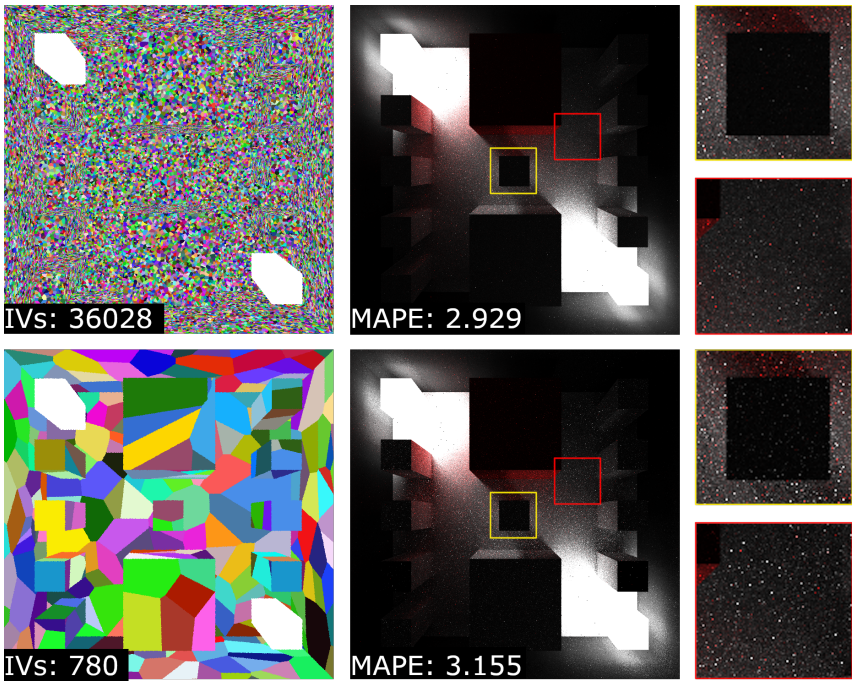


Figure 4.7: The Complex Pillars scene rendered using two Expected Sarsa path tracers, both trained for 1000 SPP and the rendered images in the middle column are produced using 128 SPP. The first row presents the Expected Sarsa path tracer with 36029 Irradiance Volumes. The second row is the Expected Sarsa path tracer with only 780 Irradiance Volumes. The left column is a voronoi plot of the Irradiance Volumes, where each pixel is given a colour to represent which Irradiance Volume it is closest to. The MAPE score and render is given for both renders in the middle column. The right column presents two close-ups of the corresponding image in the middle column to give a visual comparison of the noise between the two images.

of global memory (RAM) and NVIDIA claim they are able to fit the Q-table for these small scenes into only 2MB of memory to receive a significant reduction in noise [20]. However, films commonly render scenes with hundred of thousands, or even millions of polygons, potentially making it impossible to store a Q-table large enough to significantly reduce image noise in memory. For example, in Transformers Revenge of the Fallen, the robot 'Devastator' was made out of 11,716,127 polygons [61]. To make this memory scaling issue clear, figure 4.8 presents a render of the `Cornell Box` scene which uses only 1MB of memory to store the Q-table.

Neural-Q

From our experiments in figure 4.1, the Neural-Q path tracer does not require significantly more memory to render large scenes. This is due to the Neural-Q path tracer only requiring more memory to store the incident radiance values for the current batch of light paths and to store ANN parameters θ. The memory required to store the current batch of incident radiance values is controllable by the batch size parameter for the number of rays, therefore does not scale with the complexity of the scenes geometry. But a study from [60] suggests more layers may be required in ANNs to approximate the incident radiance $L_i(x, \omega)$ for scenes with more complex geometry, leading to more memory being required to store more weight parameters of the network. However, further experimentation must be done to investigate the scaling of memory usage to scenes with thousands of vertices for the two rendering methods.

The renders presented in figure 4.1 were produced using a constant ANN architecture as specified in section 3.2.4, which required only 30MB of memory to store the ANN as well as intermediate values computed during a forward pass (see table 4.1). Whilst the Expected Sarsa path tracer required the rendered scenes with different geometry to have significantly different Q-table sizes to reduce rendered image noise. This is due to the ANNs ability to generalize the approximated incident radiance function over its 5-dimensional space. Where a single weight in the ANN has the capacity to contribute to the approximated incident radiance of many positions and directions in the scene. Hence, an ANN with 2 hidden layers was able to make a good approximation of the incident radiance for any (x, ω) for all tested

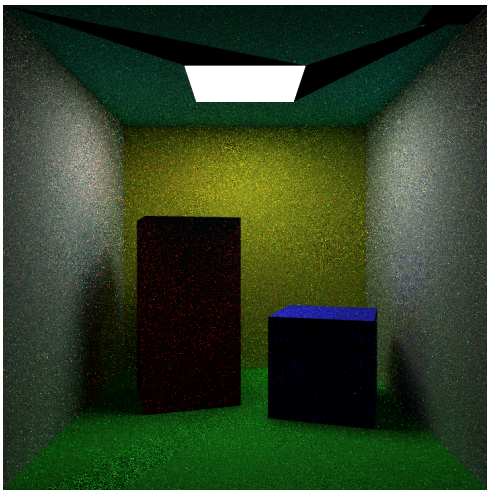


Figure 4.8: Cornell Box scene rendered using 1MB of memory to store the Irradiance Volumes for the Expected Sarsa path tracer. The large black artefacts and incorrect colouring of the red block is a result of using too few Irradiance Volumes to approximate the incident radiance function $L_i(x, \omega)$. Causing importance sampling to actually increase the variance in pixel value estimates, leading to an increase in overall image noise.

scenes. The Q-table on the other hand holds an individual value for each incident radiance estimate (x, ω), therefore the storage requirements rose for the scenes rendered with more complex geometry. For example, the Q-table required for Expected Sarsa rendering of the large archway scene without any artefacts used 272MB of memory, 9X more than that of the Neural-Q path tracer.

4.5.2 Parallel Processing

To ensure high quality, high frame rate rendering of scenes, rendering algorithms have become very computationally demanding [19]. In order to facilitate this high level of performance, parallel computing must be used to distribute the computational workload. This pattern follows for both scanline renderers and ray-tracing renderers (which path tracing falls into) [5, 24]. The default path tracing algorithm in particular is an embarrassingly parallel algorithm [25], as the task of finding the colour estimate of a single light path can be performed independently by a single process without any communication to other processes. A simple example of this is given in [6], whereby using a GPU the process of rendering an image using an algorithm which closely resembles path tracing is split among many cores. In industry path tracing algorithms take advantage of parallelism to an unprecedented scale, such as Disney's supercomputer for rendering full-length films using the Hyperion renderer [64]. Therefore, for any modified path tracing algorithm to gain industries attention it needs to be able to take full advantage of parallel computing hardware.

Expected Sarsa

The Expected Sarsa path tracer is no longer as trivial to parallelize as the default forward path tracing algorithm, due to the the first addition to the algorithm detailed in section 3.1.3, where the incident radiance estimate for a given entry $Q(x, \omega_k)$ is updated for each intersection point x and direction ω_k while constructing a light path. To parallelize the algorithm means to create an individual process to construct each light path to determine its colour estimate. However, the Q-table which stores the current approximation of all discrete incident radiance values in the scene is accessed by each thread, eaning, it is possible that two threads will attempt to write to the same location at the same time in the Q-table if they intersect near one another. This will cause the update rule provided in equation 3.5 to be applied incorrectly. To remedy this, the update rule can be placed within a critical code section [59] to ensure only one thread at a time can calculate and set the updated incident radiance estimate. For our purposes, all code written for parallel execution used the CUDA Toolkit [52], where CUDA atomic operations were used to ensure only one thread at a time can calculate and apply the Q-value update. The critical section added a minor performance overhead to each thread in both waiting to and performing the update rule. However, it was negligible when using a Q-table near the size of any used to render the images in figure

4.1. This is due to the number of Q-values stored, making it rare two threads attempt to write to the same address at the same time.

The significant performance penalty we found the Expected Sarsa approach adds to path tracing is the time taken to perform the nearest neighbour search for finding the closest Irradiance Volume to a given intersection point in the scene. Whilst we implemented a KD-Tree on the GPU in order to reduce the nearest neighbour computational complexity to $O(\log n)$, we found the Expected Sarsa path tracer ran nearly 10X times slower then that of the default path tracer. The cause of this slowdown was found to be in per thread memory read times to global CUDA memory [54].

As a quick overview of CUDA's thread hierarchy; an execution thread in CUDA is contained within a group of threads known as a thread block. Each thread block is executed by a symmetric multiprocessor (SM) placed in a GRID. Following this, the memory hierarchy consists of per-thread local memory, per-block shared memory, and global memory which any thread can read and write to at any time [54]. The issue with the Expected Sarsa path tracer, is that the Q-table needs to be globally accessible to any thread in the GPU at any time, as every thread will be constructing a light path and during its construction it may need to update any $Q(x, \omega)$ in the Q-table. This compounded with the fact that the Q-table is far too large to store in per-thread or per-block memory for any commodity GPU, means nearest neighbour lookup performs $O(\log n)$ global memory lookups. Each uncached global memory lookup has roughly 100X higher latency then that of per-thread or per-block memory lookups [32]. Hence, the Expected Sarsa algorithm experiences a high performance penalty from idle time waiting for data to be returned from global memory to a core for processing.

Interestingly, NVIDIA claims that the Expected Sarsa path tracer is able to render images with the same number of SPP by adding only a 20% performance penalty on the default path tracers runtime to significantly reduce image noise [20]. However they do not specify the details of the machines that were running these experiments. Our implementation could be further optimised by ensuring aligned memory access and reducing the amount of data which must be retrieved during rendering. But we believe it is unlikely that any of these optimisations will be able to avoid the bottleneck of regularly querying global memory.

Neural-Q

As for the Neural-Q path tracer parallel execution is slightly different, whereby batches of light paths colour estimates are computed one at a time. Note, the batch size should be set to be large enough such that no processors are idle whilst computing a batch, whilst being small enough to fit the temporary approximated incident radiance values into memory. Recall, these temporary approximated incident radiance values are used for importance sampling directions to continue light paths in.

Also, mini-batches which are the same size as the batch of light paths can be used in querying and training the ANN by stochastic gradient descent. A problem which added a significant performance penalty on our approach compared to that in [49], is that there was no built in way to evaluate an ANN per-thread on the GPU using the Dynet framework. Only code from the CPU could call any of the Dynet API functions. Meaning, our implementation had to frequently send data produced from tracing a batch of light paths to their next intersection point (positions, normals, throughputs etcetera) from the GPU to the CPU, only to call a dynet function which moved the data back to the GPU again for evaluating the ANN. Within algorithm 3 there are a total of 3 calls to the Dynet API for evaluating the network for every batch of light ray paths, this means there is a total 6 batch data transfers from the host device memory (memory that is directly accessible by the CPU) to GPU memory.

The latency added by this design flaw totaled to over a 100X slow down for our experiments with the Neural-Q path tracer. This was primarily due to the peak bandwidth available for transferring data between memory in the GPU is 144 GB/s, while for transferring data from the CPU to the GPU (and vice versa) is only 8 GB/s [31]. Therefore, the 18x performance penalty received 6 times with each batch computation, makes evaluating our implementation of the Neural-Q path tracer on render time not a fair test. However, the performance penalty received by methods which use neural network based importance sampling for path tracing have found that the cost of evaluating the ANN without the data transfers from CPU to GPU (and vice verca) can still add a 10X peformance penalty, compared to other importance sampling methods in some cases [49, 36]. Clearly this needs to be alleviated before the neural network importance sampling methods are adopted for path tracing in industry. Answers to this problem may already lie in existing hardware, by leveraging the power of NVIDIA's TensorCores in Tesla GPUs [53], which claim to have 40X higher performance than CPUs for inference on ANNs.

4.6 Hyperparameters

The final point of comparison between the Expected Sarsa and Neural-Q path tracers is the hyperparameters which must be configured for the algorithms. One of the enticing parts of using path tracing in industry, are its lack of hyperparameters [26]. It turns out a lack of hyperparameters is very important to artists for rendering photo-realistic images of scenes, as they are able to iterate quicker in their development of a scene. For example, when developing a scene for a film, an artist will be constantly changing the geometry and will need a rendering algorithm which requires little configuration to quickly render a scene [26].

Expected Sarsa

The default path tracing algorithm (algorithm 1) only requires the number of SPP to be set for rendering. Along with the SPP, the Expected Sarsa path tracer requires the user to specify the amount of memory to use for the Q-table to approximate the incident radiance function [20]. As described in section 4.5, the higher the amount of memory given to the algorithm, the more Irradiance Volumes sampled around the scene. This leads to a more accurate approximation of the incident radiance function. Not only does this present the issue that for more complex scenes, more memory is required, but actually determining the minimal amount of memory required to significantly reduce the noise in rendered images without any artefacts is a difficult task in itself. By artefacts, we mean large areas of the image which have a significant amount of noise, such as that seen in figure 4.8.

Our approach for determining the amount of memory a scene required was to instead determine the number of Irradiance Volumes to sample per-polygon in the scene. This was done by determining a surface area to Irradiance Volume ratio. For example, say the ratio was $1 : 10$, this means every 1cm^2 of surface area in the scene would have 10 Irradiance Volumes sampled upon it. This changes the parameter from arbitrarily guessing the memory required by the scene to a ratio, which we found takes less time to find the optimal value for in practice. However, the trial and error approach required to find the correct ratio is still present, but just to a lesser degree. Clearly finding the optimal memory usage for the discretization of the scene in the Expected Sarsa method is a time consuming process. In practice, it may take longer to find an optimal amount of memory to render a scene with, than it does to just use the default path tracer with a higher SPP to get rid of noise. Note, this is more of a problem for cases where memory usage is a concern, such as rendering scenes which require many textures to be stored in memory, or when the scene has a very high polygon count. Otherwise, a large amount of memory can just given to the algorithm and it should generally perform well.

Neural-Q

The Neural-Q path tracer was shown to significantly reduce the noise in rendered images using a fixed amount of memory for the ANN architecture provided in figure 4.1. Therefore, the parameter for memory provisioned by the algorithm does not need to be specified by the user. However, as discussed in section 4.5, the ANN may require more layers to represent scenes with more complex geometry than those we have experimented with. For now, the Neural-Q algorithm only exposes two parameters; the beginning ϵ value and the decay rate δ, both defining the ϵ-greedy policy to follow. It is common practice to begin start with $\epsilon = 1$, as initially the reinforcement learning agent has no prior knowledge of which actions are more valuable to take [45, 73, 46]. The same goes for the Neural-Q path tracer, as initially the ANN has no prior knowledge of the incident radiance function. Meaning, $\epsilon = 1$ is a simple choice to make.

The less trivial parameter is the decay δ which should be high enough for the algorithm to quickly start exploiting its approximation of the incident radiance function $\hat{q}_\theta(x, \omega_k)$, by importance sampling light path directions. Yet low enough, such that the algorithm efficiently builds an accurate estimate of the incident radiance function. The choice of $\delta = 0.05$ for all renders presented in figure 4.1 was made as it fit the criteria specified for the four scenes experimented with. This is not to say this is the universal best decay rate for all scenes. It is likely scenes with a large number of blockers and complex geometry require more exploration, however this is something yet to be explored along with Neural-Q in general on scenes with a very large number of polygons.

The Neural-Q algorithm's parameters are clearly easier to configure for rendering an arbitrary scene, compared to the Expected Sarsa's. Having to manually decide on memory usage is no easy task and is far too dependent on the scene being rendered itself. Meaning, every change to geometry when building a scene using a production path tracer such as [26, 16, 64] will require the difficult process of estimating

the memory needed to render the scene without artefacts. This is something artists would definitely wish to avoid when designing a scene.

4.7 Recent Advancements in Neural Importance Sampling for Monte Carlo Path Tracing

Along with Nvidia's paper which suggested a reinforcement learning approach for approximating the incident radiance at any point in a scene [20], other reinforcement learning based methods for learning light transport in a scene have been engineered [48, 74]. But none of these investigated the potential of deep reinforcement learning for this problem. However, recently neural networks for both the global and local sampling of light paths have had a large amount of research [78, 49, 36, 33]. Global light path sampling is where the optimal density of SPP, for every pixel is determined by a learning algorithm. Therefore, the reinforcement learning agent only determines where light paths should initially be sampled from. While this method does not reduce noise in the path tracing to the extent local importance sampling does, both [48, 78] have applied this technique to reduce image noise with a much lower performance penalty compared to that of local importance sampling. We however, have been working on evaluating local light path importance sampling methods. Local light path importance sampling involves learning important directions to continue light paths in when they intersect with surfaces in the scene, in order to reduce image noise.

The follow up paper to the one which introduced the Expected Sarsa path tracer [20] is [36], which was published towards the end of the execution of our work. Here, ANNs are experimented with for rendering in three different ways; for a given intersection point in the scene an ANN is trained to determine which light source should be used to compute the direct light incident on that intersection point, for approximating the visibility of an intersection point to all light sources, and for approximating the radiance at a given point in the scene to directly approximate a pixel's colour value using many small ANNs. The results are promising for all three use cases. However, unlike the Neural-Q path tracer, none of the methods estimate the incident radaince from a set of discrete directions around a point for importance sampling.

Another paper published by Disney's Zurich research team during the implementation of our work in Febuary 2019 introduced *neural importance sampling* [49]. Neural importance sampling samples a direction to continue a light path in and Disney call their algorithm for doing so Neural Path Guiding (NPG). This is currently the only other method which is an alternative for using neural networks to importance sample directions to continue light paths in, based on a learned approximation of the incident radiance function. An ANN framework for modelling complex high-dimensional densities known as Non-linear independent Component Estimation (NICE) [22] is trained to learn the distribution of incident radiance at any given point in the scene. However, unlike our approach, the network takes in the intersection point $x \in \mathbb{R}^3$, the intersection point's surface normal and the outgoing direction ω_o of interest to directly evaluate $L_o(x, \omega_o)$. A one blob encoding is then applied to these inputs which improves the speed and performance of inference by the network to output a single direction ω to continue the light path in. The PDF over the hemisphere Ω of possible directions to sample from at an intersection point x, can also be evaluated at ω using their proposed ANN framework. This has the advantage over Neural-Q as it is able to learn the incident radiance function over both the continuous set of locations and directions in the scene, rather than discretizing the possible directions using the adaptive quadrature technique. However, the ANN framework they proposed is far more expensive to evaluate than that used by the Neural-Q path tracer, as it consists of a one-blob encoding, many fully connected layers, and a piecewise polynomial warp [49]. Whereas, the Neural-Q network in comparison only consists of two hidden layers, hence requires less memory for storing parameter values and evaluating the network compared to that of NPG.

Chapter 5

Conclusions

The objectives set out by this thesis were to find a way to accurately learn the incident radiance function for the continuous set of locations in a scene. Then to design an algorithm which used this approximation for importance sampling directions to continue light paths in as part of Monte Carlo path tracing to reduce image noise. This algorithm then needs to be compared against an existing state of the art method which instead discretizes the continuous set of possible locations in the scene to approximate the incident radiance function. With the new algorithm designed and evaluated, the question of, "Is learning the incident radiance for the continuous set of possible locations in a scene beneficial for importance sampling in Monte Carlo path tracing?" can be answered. This chapter describes our current state of achievement of these goals, and in turn the contributions we have added to the field of computer graphics and efficient numerical integration, followed by a discussion on the future research opportunities which have arisen due to the conclusions we have made.

5.1 Wider Motivation and the Problem

Monte Carlo path tracing is a method used in Computer Graphics capable of producing photo-realistic images. Traditionally, it is known to trade off rendering time for the superior image quality it can produce. However, increasing compute power, innovative algorithms, and post processing have been shown to accelerate the speed of the algorithm by a significant amount. This combined with the lack of set-up required to render any arbitrary scene has led to a large resurgence in the algorithms use in industry [38]. A larger goal held across industry at this point is to achieve real-time rendering for methods like path tracing which accurately simulate light transport [4], but more work is required to make this a reality.

Monte Carlo path tracing uses Monte Carlo integration to approximate the true pixel value for all pixels in an image to render a scene, where a sample is represented by a single light path. Therefore, importance sampling light paths for Monte Carlo path tracing has the potential to significantly improve every pixels colour estimate within the image, while using the same number of SPP. However, the task of finding correct information to importance sample light path directions with is a difficult one, and there have been many different approaches for doing so [74, 48, 20]. NVIDIA's technique for this is what we refer to as the Expected Sarsa path tracer, which introduced the application of reinforcement learning for approximating the incident radiance function, as well as a new path tracing algorithm to use this approximation for importance sampling light path directions [20]. Much like the other existing methods, their tabular TD-learning approach required a discretization of locations in the scene to approximate the incident radiance function, whereas the scene in actual fact contains an infinite number of discrete positions. This raised the question that is it possible to learn the incident radiance function over the continuous set of locations in the scene using a function approximator instead? Also, is this advantageous for reducing noise in images produced by Monte Carlo path tracing compared to existing methods?

5.2 Summary of Contributions

We have introduced an ANN loss function based on Deep Q-learning [45], along with an ANN architecture for learning the incident radiance function $L_i(x, \omega)$ [35] for an arbitrary scene. To make use of the proposed ANN and loss function, a new path tracing algorithm known as the Neural-Q path tracer has

been developed. Neural-Q uses the proposed learning rule to efficiently train the ANN online during the rendering process. Much like Expected Sarsa, the approximation of the incident radiance function is then used to importance sample directions to continue light paths in, and to calculate PDF needed for correctly evaluating Monte Carlo path tracing. As more light paths are sampled in the rendering process, the accuracy of the approximated incident radiance function improves, leading to more efficient importance sampling of directions to continue light paths in. In turn, the variance in the approximation of pixel values during Monte Carlo path tracing using the same number of SPP is reduced. Directly reducing image noise.

We have also introduced and compared the Expected Sarsa path tracer to our Neural-Q path tracer as a means of assessing the extension of approximating the incident radiance over the set of continuous locations in a scene. In doing so, we found this extension brought various advantages for importance sampling Monte Carlo path tracing. These include; improved incident radiance function approximation for scenes with more complex geometry, leading to noise reduction in images produced by Monte Carlo path tracing, as well as improved memory scaling for rendering scenes with more complex geometry, and simpler hyperparemeters for tuning. The investigation also uncovered that the Expected Sarsa algorithms applicability for rendering in industry is limited by the memory required to store the Q-table for approximating the incident radiance function. Whereas, the Neural-Q algorithm is limited by the time taken to evaluate a forward pass on an ANN. This conclusion was also reached by two studies published during the execution of this thesis [36, 49].

5.3 Discussion

In this section the key conclusions made by our work are summarised.

Reducing noise in Monte Carlo path tracing renders

The results presented so far for the newly developed Neural-Q algorithm are promising. It is able to significantly reduce noise for scenes rendered by Monte Carlo path tracing using the same number of SPP. Furthermore, it is able to reduce image noise more compared to that of the state of the art Expected Sarsa path tracer for scenes with more complex geometry. While at the same time using a smaller, constant amount of memory. Hyperparamter tuning was also a much simpler task, as determining the parameters for the decaying ϵ-greedy strategy used by Neural-Q for the scenes tested was found to be a far easier and quicker task, compared to deciding the amount of memory the Expected Sarsa algorithm requires to significantly reduce image noise. These factors combined with Neural-Q's online training to progressively reduce image noise across accumulated rendered frames for a scene, make it a promising addition to the field of computer graphics. However, it is important to realise that newly proposed rendering algorithms should be tested for a broad range of scenes to fully assess which ones they apply best to. The Neural-Q algorithm and ANN architecture we propose is no exception to this. While we have shown it is able to perform well for a selection of different scenes exhibiting different properties, scenes used in film and game production are yet to be experimented with for both the Neural-Q and Expected Sarsa path tracers.

Improving the approximation of the incident radiance function

The reasoning for the reduction in noise was found to be the more accurate approximation made by the Neural-Q algorithm for the incident radiance function $L_i(x, \omega)$. This conclusion was reached by both further analysing the noise present in images rendered by the Expected Sarsa method and visualising the incident radiance distribution on three different points in a simple scene within section 4.2. Expected Sarsa's poorer performance for renders was found to be a product of the more inaccurate incident radiance function approximation made compared to that of the Neural-Q path tracers. This was related back to the theory of Monte Carlo importance sampling, which also revealed why there were a large number of fireflies present in the Expected Sarsa renders.

Memory versus compute time

While both the Neural-Q and Expected Sarsa path tracers are able to significantly reduce noise in images rendered by Monte Carlo path tracing, they both come with costs. The Expected Sarsa was found to be limited in its ability to reduce noise for scenes by the amount of memory available in the underlying the hardware to store the Q-table for approximating the incident radiance function. The real concern from this was that more complex scenes required more memory to receive a significant reduction in image

noise. Meaning, Expected Sarsa may not be applicable to scenes used in the production of films which include millions of polygons, due to the memory requirements to store the Q-table being far too large.

On the other hand, the Neural-Q algorithms requirement to frequently evaluate forward passes on an ANN for tracing every light path is very computationally expensive. In fact so much so, new purpose built hardware offered for quick ANN inference running per thread on a GPU will likely be required to have any hope of using Neural-Q in industry.

Neural Importance Sampling

The benefits which are received by using ANNs for sampling directions to continue light paths in for Monte Carlo path tracing are now clear. Not only does this present an addition to the field of computer graphics, but also presents a clear application of using neural networks for importance sampling in Monte Carlo methods, otherwise known as neural importance sampling [36]. Meaning, other methods which require Monte Carlo integration to numerically solve integrals may benefit from an improvement in efficiency by adapting the approach outlined here to their problem. In fact, this is a hot topic in computer graphics right now, where leaders in the computer graphics industry including Disney and NVIDIA are making serious progress in the application of neural importance sampling to Monte Carlo rendering techniques [36, 49]. But, to the best of our knowledge, the Neural-Q algorithm and ANN architecture we introduced is the first which uses a shallow ANN for importance sampling light path directions in Monte Carlo rendering.

Assessing the data collected in recent investigations, it seems as though approximating the incident radiance using Monte Carlo integration with neural importance sampling, rather than using an ANNs approximation of the incident radiance function to directly infer a pixels colour value, produces higher quality images [78, 36, 49]. Hence, it is an area which we are likely to see a significant amount of research in within the near future.

5.4 Future Work

With the introduction of the new loss function and Neural-Q path tracer, as well as recent studies on neural importance sampling published during the execution of this thesis, many areas of future research have opened up. In this section we present the most interesting areas which we believe will lead to answers for the most important questions currently surrounding neural importance sampling techniques for Monte Carlo path tracing.

Scaling up Neural-Q

There are three areas we propose that the Neural-Q algorithm should be scaled up:

1. **Testing high polygon count scenes** - For the Neural-Q path tracer to be adopted in industry it needs to be tested on scenes with potentially millions of polygons. If Neural-Q is able to continue to significantly reduce the noise in rendered images of such scenes, then it is only a question of how to speed the algorithm up to make it competitive with existing algorithms. However, we believe it is more likely further development will have to be done on the ANN architecture used by the Neural-Q path tracer to learn the incident radiance function for a very complex scene. Our reasoning behind this thought is due to the complexity of the ANN architecture described in [49] to model scenes with complex geometry. However, our idea of using the vertices converted into a coordinate system centred around the input position may reduce the need for this complexity.

2. **Accelerating Neural-Q on optimised hardware** - The experiments conducted in this thesis for the Neural-Q path tracer used an NVIDIA 1070Ti GPU, which is not even the latest series of commercial GPUs released by NVIDIA with the introduction of their new 20 series GPUs [55]. Therefore, we believe it is necessary to test Neural-Q path tracer on a variety of the latest available graphics cards, prioritising those with NVIDIA Tensor Cores [53] in hope to significantly speed up inference during light path construction. Note, these tests should also be run on a more optimised path tracing engine than the one developed for this project, including common features such as bounding boxes [14] and practices like aligned memory accesses. Comparisons should then be made against existing methods for importance sampling in Monte Carlo path tracing, such as the Expected Sarsa algorithm and many others.

3. **Integrating Neural-Q into a production path tracer** - This point relies on the two previous ones being completed and receiving satisfactory data that proves Neural-Q can be applied to industry for rendering. By integrating Neural-Q to a production renderer, the algorithm can be assessed on rendering films and whether artists find it to be helpful or taxing to their work. This will also check if the algorithm combines well with post processing used in production renderer pipelines [26, 16]. For example image denoising by ANNs allowing the use of far less samples in Monte Carlo path tracing to begin with [8, 15].

Investigating the relationship between scene geometry and ANN requirements

The Neural-Q algorithm outperforms the Expected Sarsa method for scenes with more complex geometry, yet it performs comparably or worse for scenes with simpler geometry. Therefore, it is clear more research must be done to investigate how an ANN architecture relates to the accuracy of the approximated incident radiance function for a scene. Particularly, further investigation is needed into the relationship between the depth of the network used in Neural-Q path tracing and the accuracy of the incident radiance function learned.

Other function approximators such as Gaussian Processes [12] for learning the incident radiance function instead of an ANN may potentially be profitable. However, we believe due to the abundance of data available by continually sampling light paths as part of Monte Carlo path tracing, ANNs are likely to produce the best results.

Neural-Q versus Disney's Neural Path Guider

The only other work to our knowledge which uses ANNs for importance sampling light path directions in Monte Carlo path tracing is that of Disney Research, that was published during the execution of this thesis [49]. As previously described, the ANN framework they introduced known as NPG also learns online, hence the Neural-Q path tracer and NPG can be tested against one another for their performance in various aspects. We believe NPG is likely to be more successful in reducing noise for an arbitrary scene due to the promising results shown for complex geometry. However, it will be slower in doing so due to the smaller ANN architecture introduced by Neural-Q. We believe this is an important area to assess, as there are no current studies published regarding the comparison of multiple neural importance sampling schemes to the best of our knowledge.

Further investigate Deep Reinforcement learning techniques

The field of Deep Reinforcement learning also has plenty to offer for the further development of the Neural-Q path tracer. Other algorithms such double deep q-learning (DDQN) [73] which has been shown to improve the efficiency of the learning process could be applied to the Neural-Q path tracer for quicker convergence on a good approximation of the incident radiance function. Semi gradient one-step Expected Sarsa [66] could be used to derive an alternative loss function which more closely resembles the rendering function, by taking a similar approach to that derived in [36].

One area of the project which was blatantly not investigated was how both the Neural-Q and Expected Sarsa path tracers performed when increasing the number of directions that were used in the approximation of the incident radiance function. In theory from the law of large numbers, as the number of sampled directions used to approximate the incident radiance distribution at a point are increased, the more accurate the approximation will be. Furthermore, similarly to our initial goal of extending the approximation of incident radiance to the continuous set of locations in a scene, the directions modelled could be extended to the continuous case. For this, adapting deep reinforcement learning actor-critic methods [43, 44] to light transport simulation is a clear way of moving forward.

Bibliography

[1] Opengl mathematics (glm).

[2] Simple directmedia layer. SDL version 2.0.9 (stable).

[3] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2018.

[5] Erik Alerstam, Tomas Svensson, and Stefan Andersson-Engels. Parallel computing with graphics processing units for high-speed monte carlo simulation of photon migration. *Journal of biomedical optics*, 13(6):060504, 2008.

[6] Roger Allen. Accelerated ray tracing in one weekend in cuda. NVIDIA Developer Blog.

[7] Autodesk. Autodesk maya api white paper.

[8] Steve Bako, Thijs Vogels, Brian McWilliams, Mark Meyer, Jan Novák, Alex Harvill, Pradeep Sen, Tony Derose, and Fabrice Rousselle. Kernel-predicting convolutional networks for denoising monte carlo renderings. *ACM Trans. Graph.*, 36(4):97–1, 2017.

[9] Thomas Bashford-Rogers, Kurt Debattista, and Alan Chalmers. A significance cache for accelerating global illumination. In *Computer Graphics Forum*, volume 31, pages 1837–1851. Wiley Online Library, 2012.

[10] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[11] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[12] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[13] Bloomberg. Peak video game? top analyst sees industry slumping in 2019.

[14] Solomon Boulos. Notes on efficient ray tracing. In *ACM SIGGRAPH 2005 Courses*, page 10. ACM, 2005.

[15] Chakravarty R Alla Chaitanya, Anton S Kaplanyan, Christoph Schied, Marco Salvi, Aaron Lefohn, Derek Nowrouzezahrai, and Timo Aila. Interactive reconstruction of monte carlo image sequences using a recurrent denoising autoencoder. *ACM Transactions on Graphics (TOG)*, 36(4):98, 2017.

[16] Per Christensen, Julian Fong, Jonathan Shade, Wayne Wooten, Brenden Schubert, Andrew Kensler, Stephen Friedman, Charlie Kilpatrick, Cliff Ramshaw, Marc Bannister, et al. Renderman: An advanced path-tracing architecture for movie rendering. *ACM Transactions on Graphics (TOG)*, 37(3):30, 2018.

[17] Per H Christensen, Wojciech Jarosz, et al. The path to path-traced movies. *Foundations and Trends® in Computer Graphics and Vision*, 10(2):103–175, 2016.

[18] David Cline, Daniel Adams, and Parris Egbert. Table-driven adaptive importance sampling. In *Computer Graphics Forum*, volume 27, pages 1115–1123. Wiley Online Library, 2008.

[19] Thomas W Crockett. Parallel rendering. Technical report, INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING HAMPTON VA, 1995.

[20] Ken Dahm and Alexander Keller. Learning light transport the reinforced way. *arXiv preprint arXiv:1701.07403*, 2017.

[21] Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

[22] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

[23] Philip Dutré, Henrik Wann Jensen, Jim Arvo, Kavita Bala, Philippe Bekaert, Steve Marschner, and Matt Pharr. State of the art in monte carlo global illumination. In *ACM SIGGRAPH 2004 Course Notes*, page 5. ACM, 2004.

[24] Kayvon Fatahalian, Edward Luong, Solomon Boulos, Kurt Akeley, William R Mark, and Pat Hanrahan. Data-parallel rasterization of micropolygons with defocus and motion blur. In *Proceedings of the Conference on High Performance Graphics 2009*, pages 59–68. ACM, 2009.

[25] Ian Foster. Designing and building parallel programs. Section 1.4.4.

[26] Iliyan Georgiev, Thiago Ize, Mike Farnsworth, Ramón Montoya-Vozmediano, Alan King, Brecht Van Lommel, Angel Jimenez, Oscar Anson, Shinji Ogaki, Eric Johnston, et al. Arnold: A brute-force production path tracer. *ACM Transactions on Graphics (TOG)*, 37(3):32, 2018.

[27] Andrew S Glassner. *Principles of digital image synthesis*. 2014.

[28] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH computer graphics*, volume 18, pages 213–222. ACM, 1984.

[29] Gene Greger, Peter Shirley, Philip M Hubbard, and Donald P Greenberg. The irradiance volume. *IEEE Computer Graphics and Applications*, 18(2):32–43, 1998.

[30] Pat Hanrahan. Monte carlo path tracing. Stanford Graphics.

[31] Mark Harris. How to optimize data transfers in cuda c/c++. NVIDIA Developer Blog.

[32] Mark Harris. Using shared memory in cuda c/c++. NVIDIA Developer Blog.

[33] Pedro Hermosilla, Sebastian Maisch, Tobias Ritschel, and Timo Ropinski. Deep-learning the latent space of light transport. *arXiv preprint arXiv:1811.04756*, 2018.

[34] Henrik Wann Jensen. Global illumination using photon maps. In *Rendering Techniques 96*, pages 21–30. Springer, 1996.

[35] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986.

[36] Alexander Keller and Ken Dahm. Integral equations and machine learning. *Mathematics and Computers in Simulation*, 2019.

[37] Alexander Keller, Ken Dahm, and Nikolaus Binder. Path space filtering. In *Monte Carlo and Quasi-Monte Carlo Methods*, pages 423–436. Springer, 2016.

[38] Alexander Keller, Luca Fascione, Marcos Fajardo, Iliyan Georgiev, P Christensen, Johannes Hanika, Christian Eisenacher, and Gregory Nichols. The path tracing revolution in the movie industry. In *ACM SIGGRAPH 2015 Courses*, page 24. ACM, 2015.

[39] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[40] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[41] Jaroslav Křivánek, Alexander Keller, Iliyan Georgiev, Anton S Kaplanyan, Marcos Fajardo, Mark Meyer, Jean-Daniel Nahmias, Ondrej Karlík, and Juan Canada. Recent advances in light transport simulation: some theory and a lot of practice. In *SIGGRAPH Courses*, pages 17–1, 2014.

[42] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[43] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[44] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

[45] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[47] William J Morokoff and Russel E Caflisch. Quasi-monte carlo integration. *Journal of computational physics*, 122(2):218–230, 1995.

[48] Thomas Müller, Markus Gross, and Jan Novák. Practical path guiding for efficient light-transport simulation. In *Computer Graphics Forum*, volume 36, pages 91–100. Wiley Online Library, 2017.

[49] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *arXiv preprint arXiv:1808.03856*, 2018.

[50] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[51] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[52] Nvidia. Cuda toolkit. Develop, Optimize and Deploy GPU-accelerated Apps.

[53] NVIDIA. Nvidia tensor cores. The Next Generation of Deep Learning.

[54] NVIDIA. Nvidia cuda c programming guide, 2012. NVIDIA CUDA.

[55] NVIDIA. *NVIDIA Turing Architecture Whitepaper*, 2018.

[56] Zhigeng Pan, Adrian David Cheok, Hongwei Yang, Jiejie Zhu, and Jiaoying Shi. Virtual reality and mixed reality for virtual learning environments. *Computers & graphics*, 30(1):20–28, 2006.

[57] Vincent Pegoraro, Carson Brownlee, Peter S Shirley, and Steven G Parker. Towards interactive global illumination effects via sequential monte carlo adaptation. In *2008 IEEE Symposium on Interactive Ray Tracing*, pages 107–114. IEEE, 2008.

[58] Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. A theory of monte carlo visibility sampling. *ACM Transactions on Graphics (TOG)*, 31(5):121, 2012.

[59] Michel Raynal. *Concurrent programming: algorithms, principles, and foundations*. Springer Science & Business Media, 2012.

[60] Peiran Ren, Jiaping Wang, Minmin Gong, Stephen Lin, Xin Tong, and Baining Guo. Global illumination with radiance regression functions. *ACM Transactions on Graphics (TOG)*, 32(4):130, 2013.

[61] Barbara Robertson. Weighty matters, computer graphics world. Transformers: Revenge of the Fallen.

[62] Scratchapixel. Monte carlo methods in practice, Apr 2015.

[63] Peter Shirley and Kenneth Chiu. Notes on adaptive quadrature on the hemisphere. Technical report, Technical Report 411, Department of Computer Science, Indiana University , 1994.

[64] Walt Disney Animation Studios. Hyperion. Disney's Hyperion Renderer.

[65] Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

[66] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 2011.

[67] William TB Uther and Manuela M Veloso. Tree based discretization for continuous state space reinforcement learning. In *Aaai/iaai*, pages 769–774, 1998.

[68] Hado van Hasselt. Exploration and exploitation. DeepMind: Advanced Deep Learning & Reinforcement Learning.

[69] Hado van Hasselt. Function approximation and deep reinforcement learning. DeepMind: Advanced Deep Learning & Reinforcement Learning.

[70] Hado van Hasselt. Introduction to reinforcement learning. DeepMind: Advanced Deep Learning & Reinforcement Learning.

[71] Hado van Hasselt. Markov decision processes and dynamic programming. DeepMind: Advanced Deep Learning & Reinforcement Learning.

[72] Hado van Hasselt. Model-free prediction and control. DeepMind: Advanced Deep Learning & Reinforcement Learning.

[73] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[74] Jiří Vorba, Ondřej Karlík, Martin Šik, Tobias Ritschel, and Jaroslav Křivánek. On-line learning of parametric mixture models for light transport simulation. *ACM Transactions on Graphics (TOG)*, 33(4):101, 2014.

[75] Ingo Wald and Philipp Slusallek. State of the art in interactive ray tracing. *State of the Art Reports, EUROGRAPHICS*, 2001:21–42, 2001.

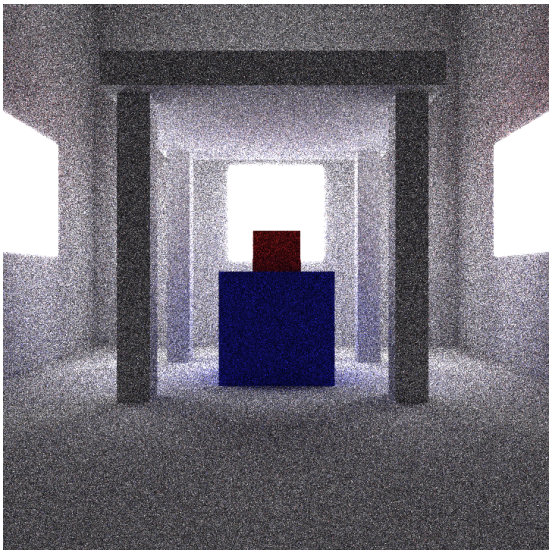
[76] Eric W Weisstein. Normal vector. MathWolrd–A Wolfram Web Resource.

[77] Turner Whitted. An improved illumination model for shaded display. In *ACM Siggraph 2005 Courses*, page 4. ACM, 2005.

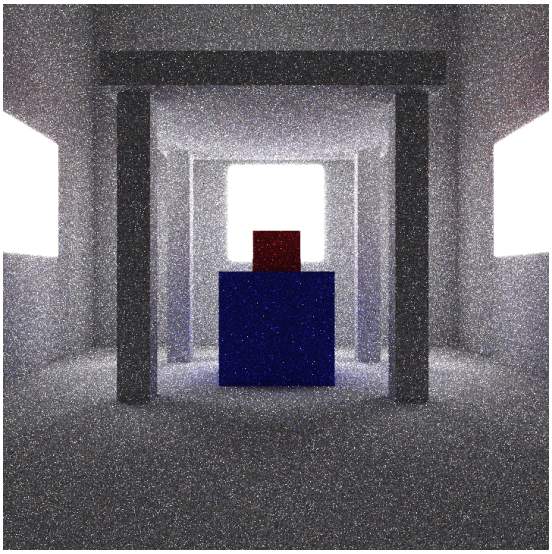
[78] Quan Zheng and Matthias Zwicker. Learning to importance sample in primary sample space. *arXiv preprint arXiv:1808.07840*, 2018.

Appendix A

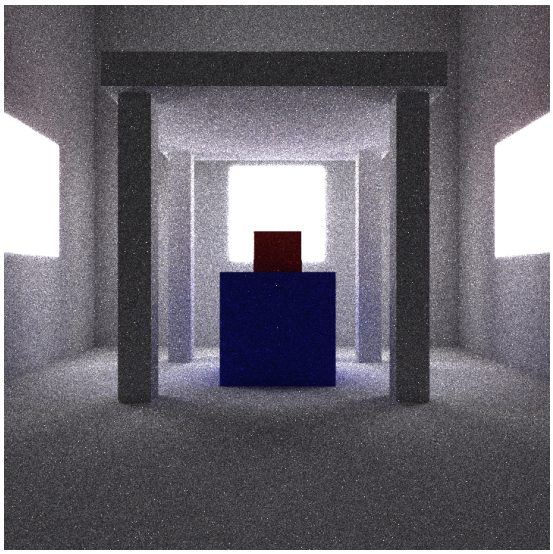
Appendix



(a) Default

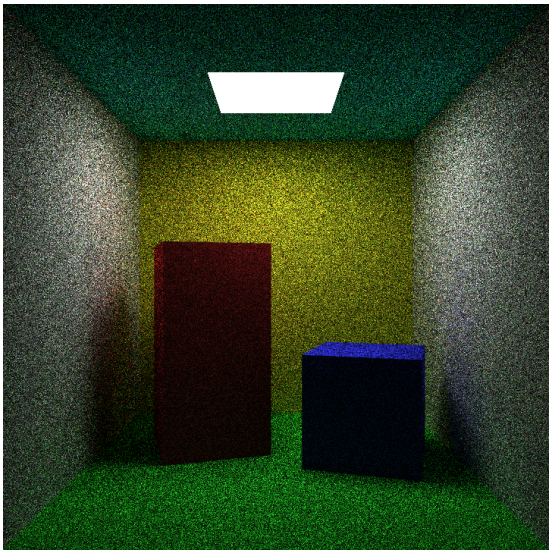


(b) Expected Sarsa

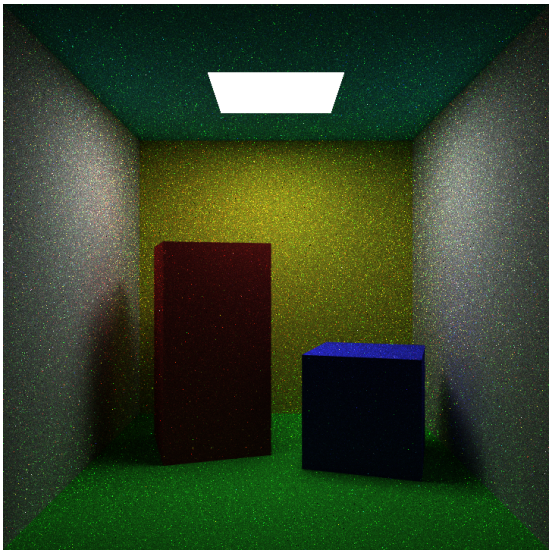


(c) Neural-Q

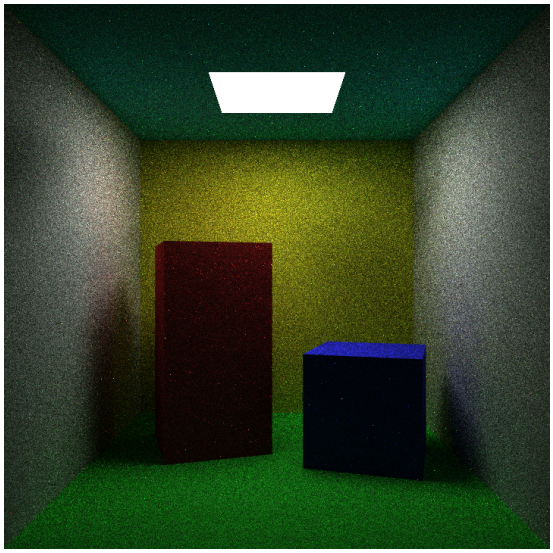
Figure A.1: 128 SPP renders for the Shelter scene.



(a) Default

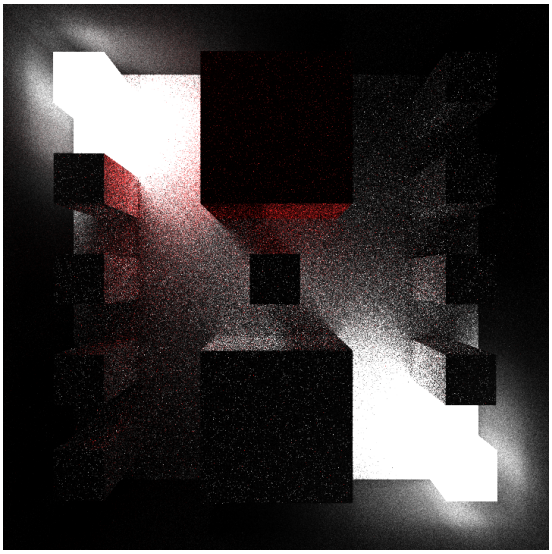


(b) Expected Sarsa

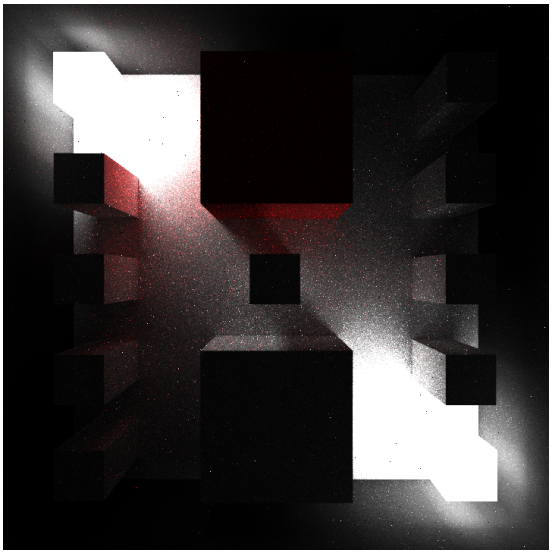


(c) Neural-Q

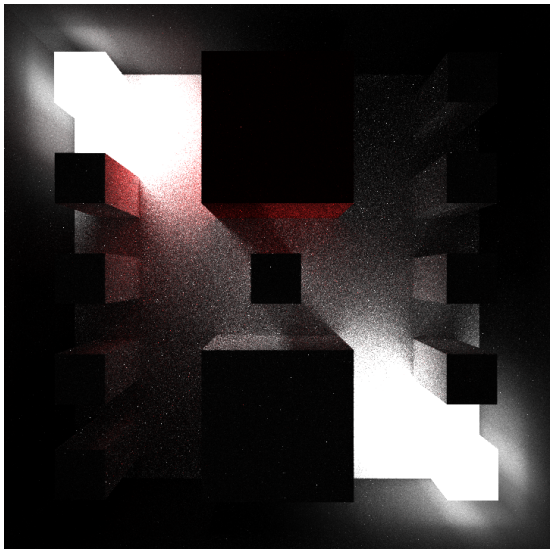
Figure A.2: 128 SPP renders for the Cornell Box scene.



(a) Default

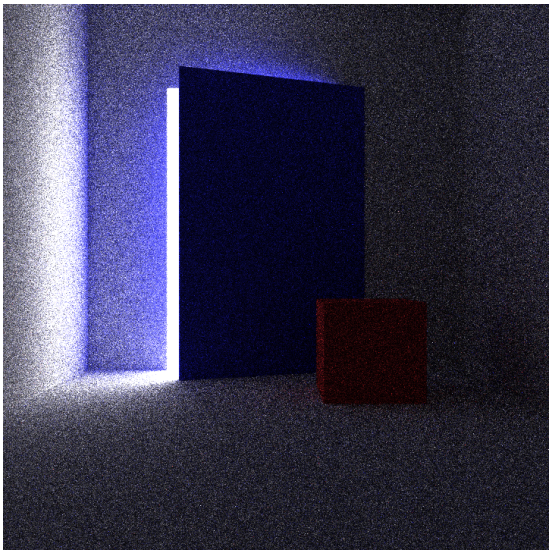


(b) Expected Sarsa

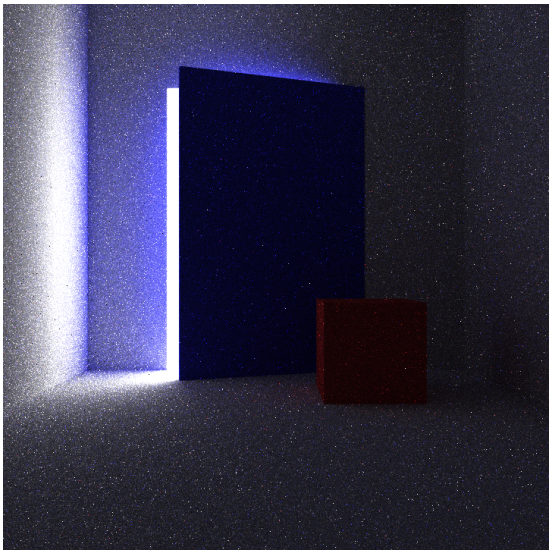


(c) Neural-Q

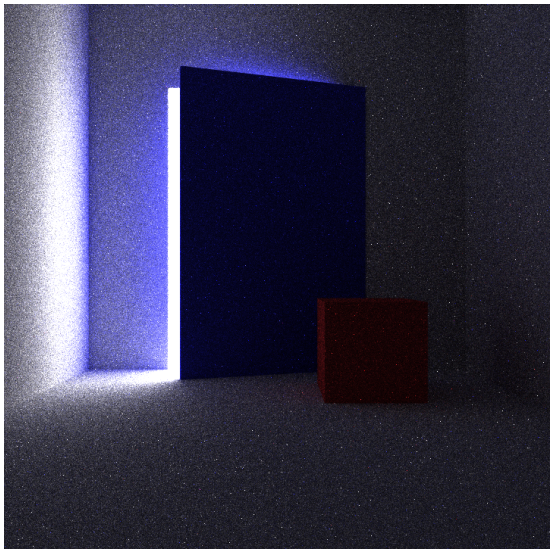
Figure A.3: 128 SPP renders for the Complex Pillars scene.



(a) Default



(b) Expected Sarsa



(c) Neural-Q

Figure A.4: 128 SPP renders for the Door Room scene.