Prelude

A typical dissertation will be structured according to (somewhat) standard sections, described in what follows. However, it is hard and perhaps even counter-productive to generalise: the goal is *not* to be prescriptive, but simply to act as a guideline. In particular, each page count given is important but *not* absolute: their aim is simply to highlight that a clear, concise description is better than a rambling alternative that makes it hard to separate important content and facts from trivia.

You can use this document as a LaTeX-based [**?**, **?**] template for your own dissertation by simply deleting extraneous sections and content; keep in mind that the associated `Makefile` could be of use, in particular because it automatically executes to deal with the associated bibliography.

You can, on the other hand, opt *not* to use this template; this is a perfectly acceptable approach. Note that a standard cover and declaration of authorship may still be produced online via

`http://www.cs.bris.ac.uk/Teaching/Resources/cover.html`

Dissertation Type: research

DEPARTMENT OF COMPUTER SCIENCE

How effective are Temporal difference learning methods for reducing the number of zero contribution light paths while still accurately approximating Global Illumination in 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.

Tuesday 23rd April, 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, Tuesday 23rd April, 2019

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Executive Summary

In the field of Computer Graphics, Path tracing is an algorithm which accurately approximates global illumination in order to produce photo-realistic images. Path tracing has traditionally been known to trade speed for image quality. This is due to the lengthy process of accurately finding each pixels colour, whereby many light rays are fired through each pixel into scene, then directions for each ray are continually sampled until it intersects with a light source. Due to this, a variety of Importance sampling algorithms have been designed to avoid sampling directions which lead to rays contributing no light to the rendered image. The paths formed by sampling rays in these directions are known as zero contribution light paths. By not sampling zero contribution light paths, it is possible to significantly reduce the noise in rendered images using the same number of sampled rays per pixel in path tracing

Recently a Temporal Difference learning method was used by Nvidia to achieve impressive results in Importance sampling within a Path tracer. The algorithm essentially learns which directions light is coming from for a given point in the scene. It then uses importance sampling to favour shooting rays stored in those directions, reducing the number of zero contribution light paths sampled. With this success, there is plenty of potential to experiment with other Temporal Difference learning methods, particularly Deep Q-Learning. It is also important to assess both of these methods on their ability to accurately approximate Global Illumination to produce photo-realistic images. From this, my goal is to investigate the ability of two different temporal difference learning algorithms ability to reduce the number of zero contribution light paths in path tracing, whilst still accurately approximating global illumination. More specifically, the first temporal difference learning method will be that proposed by Nvidia, and the second will be my designed Neural-Q path tracing algorithm. I will be comparing these two methods in order to test the following hypothesis:

> The Neural-Q path tracer is further able to reduce the number of zero contribution light paths than an Expected SARSA Path tracer proposed by Nvidia, whilst still accurately simulating Global Illumination.

Outcomes

- Which is better able to reduce the number of zero contribution light paths expected SARSA or Deep Q-learning
- Can Expected SARSA learning handle multiple lights well in a scene & deep q-learning

Main areas of work

- I have written x lines of code to build a Path tracing engine from scratch which supports a variety of GPU accelerated Path tracing algorithms I have experimented with.
- I have spent x hours researching into the field of efficient light transport simulation for ray-tracing techniques.
- I have spent x hours researching into Reinforcement learning, particularly Temporal Difference learning and Deep Reinforcement learning, neither of which I have been taught before.
- I spent x hours implementing and validating the on-line Expected SARSA Path tracing algorithm proposed by Nvidia, which required me to implement the Irradiance Volume data structure as a prerequisite.
- I have spent x hours designing, implementing and analysing my own on-line Deep Q-learning Path tracing algorithm, along with a neural network architecture designed for the algorithm.

Supporting Technologies

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

Notation and Acronyms

TD learning : Temporal Difference learning

Acknowledgements

An optional section, of at most 1 **page**

It is common practice (although totally optional) to acknowledge any third-party advice, contribution or influence you have found useful during your work. Examples include support from friends or family, the input of your Supervisor and/or Advisor, external organisations or persons who have supplied resources of some kind (e.g., funding, advice or time), and so on.

0.0.1 Plan

1. Carl Henrik Ek - Validating my understanding of deep reinforcement learning
2. Neill Campbell - Deep reinforcement learning strategy

Chapter 1

Contextual Background

This chapter explains on a high level what path tracing is and how it accurately simulates light transport. Then importance sampling ray directions in light transport simulation is discussed, and how it can potentially reduce the number of zero contribution light paths and the associated benefits with this. Temporal difference learning as a branch of reinforcement learning is then introduced, along with how it can be used in importance sampling ray directions towards light sources. With a conceptual overview of theory my work is based on, I take a look at recent work which contributes to real-time accurate light transport simulation which my work aims to contribute to. Finally, an overview of the objectives and significant challenges of my investigation are described.

1.1 Path Tracing for Light Transport Simulation

Path Tracing is a Monte Carlo method for rendering photo-realistic images of 3D scenes by accurately approximating global illumination [7]. Figure 1.1 summarises on a high level how forward Path tracing produces a 2D image of a 3D 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 fired again. This process is repeated until all rays have intersected with an area light, at which point the pixel colour value can be found by averaging the colour estimate of each ray fired through that pixel. Each rays colour estimate is calculated based on the material surface properties it intersects with before intersecting with the light and the intersected area lights properties. The more rays shot through each pixel (also known as samples per pixel), the more visually accurate the rendered image becomes, but at a higher computational cost.

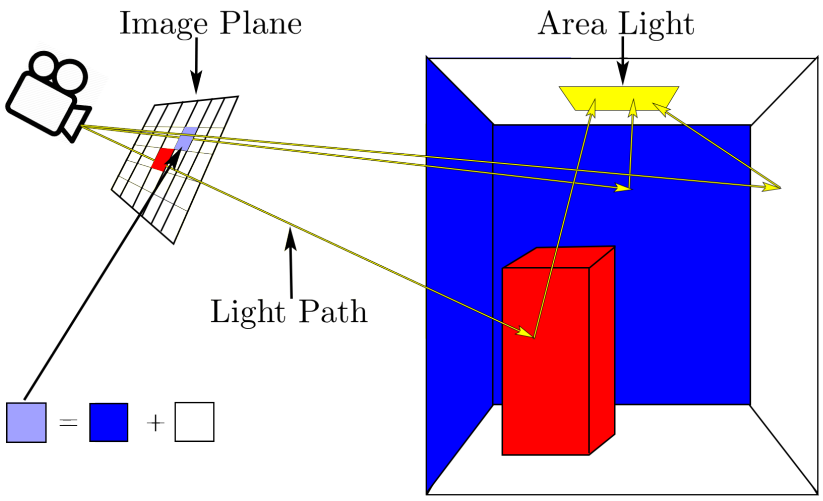


Figure 1.1: An illustration of path tracing, where three light paths are traced from from the camera through a pixel, to the light source in a simple 3D scene.

Path tracing simulates global illumination, meaning it accounts for both direct and indirect illumination. Direct illumination being rays of light emitted from a light source, which reflect off exactly one surface before reaching the camera in the scene. Whereas indirect illumination are ray of light which reflect 2 or times before reaching the camera. In **??**, 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 effects due to Path tracings ability to accurately simulate light transport, which is not the case for the directly illuminated scene. Where light transport simulation refers to firing and summing up the contributions of light transport paths that connect from the camera to light sources [18], such as those displayed in **??**. For example, effects such as (a) colour bleeding, (b) soft shadows, and (c) indirect diffuse lighting are a product of accurate light transport simulation.

Light transport simulation methods are able to produce many complex light transport effects by a simple single pass of a rendering algorithm. 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 light transport simulation rendering methods in the past decade [19].

My work in this thesis focuses on developing and assessing importance sampling techniques using Temporal Difference learning methods for light transport simulation in forward Path tracing. In particular, More specifically, for any intersection point in a 3D scene, I attempt to create an AI agent that learns and samples in directions light is coming from, reducing the total number of zero contribution light paths. A zero contribution light path is one whose estimated colour values are almost zero for all (R, G, B) components, hence, they contribute almost no visible difference to the rendered image. We should instead focus our sampling on light paths which do contribute to the image, reducing the noise in pixel values and bringing them closer to their true values for the same number of sampled rays per pixel. Meaning, Importance sampling can reduce the number of rays needed to be sampled per pixel in order to receive a photo-realistic (also known as converged) image from Path tracing. An example of this reduction in noise can be see in **??**, where the naive forward Path tracing algorithms output is compared to Nvidia's on-line reinforcement learning Path tracer using Importance sampling. Note, any light transport simulation algorithm can benefit from the Temporal DIfference learning schemes which will be described [16, 18], as they are all derived from what is known as the rendering equation. This equation is used as a mathematical basis of modelling light transport.

It is paramount that Importance sampling Path tracing algorithms continue to accurately simulates global illumination in order to produce photo-realistic images in a single rendering pass, as this is the major selling point of Path tracing over other methods. Therefore, I will also be assessing the accuracy of the global illumination approximation made by the Importance sampling algorithms compared to that of the naive forward Path tracing algorithm.

1.2 Temporal Difference Learning for Importance Sampling Ray Directions

There are three important unanswered questions up to this point; 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, are 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 [27]. 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 a position in the world it is exploring, and its action are the directions to move in

for a given distance. If it discovers a new area, it receives a positive reward signal. 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.

As mentioned TD learning methods are model free methods, meaning the methods do not require a model of the system dynamics they are placed in, instead they learn over time by interacting with the system. In other words, they learn from raw experience [27]. TD methods update their current estimates based on a combination of data received from interacting with the environment, and partly on their current learned estimates without waiting for the final outcome of events, this is known as bootstrapping. To illustrate the concept of bootstrapping, imagine you are driving home from work and you wish to estimate how long it will take you get home. By following a TD learning method, if you hit traffic you can update your current estimate of the time it takes you to drive home based on this new data, and your pre-existing estimate. Whereas compared to another set of Reinforcement learning methods known as Monte Carlo methods, you would have to wait until you got home to update your current estimate of how long it takes to get home from work. Meaning you have to wait for the final outcome before learning can begin, which is not the case for TD learning.

1.2.2 Temporal Difference learning methods for Efficient Light Transport Simulation

One of my main aims to reduce the number of zero contribution light paths sampled in Path tracing by the use of TD learning methods. In order to do so I must formulate the problem a reinforcement learning problem, which is done in detail in Chapter 2. However for a conceptual overview it suffices to explain what a state, action, and reward signal will be in the case of light transport simulation within Path tracing:

- **State**: A 3D intersection position in the scene for a given ray to sample the rays next direction from.
- **Action**: Firing the ray in a given direction (3D vector) from the current state.
- **Reward Signal**: The amount of light incident from the direction the ray was sampled in.

In this reinforcement learning setting, we can use TD-learning methods to create an AI agent which learns by taking different actions in different states and observes their reward signals to find out for each state which actions have the highest valuations. By then converting the action space into a probability distribution weighted by each actions learned valuation, the AI agent will more likely sample non-zero contribution light paths, reducing noise in rendered images. Note, the term valuation means the total expected reward for taking a given action, meaning valuation not only accounts for the immediate reward, but the expected reward for taking all future actions to come until the ray intersects with a light. Also, for the proposed AI agent, current actions can affect future rewards, as when the ray intersects a surface it looses some energy. Therefore, future rewards received after many intersections will be discounted compared to the reward of received immediately to match this behaviour. This means the agent will aim to minimise the average number of intersection a ray makes before intersecting with a light source, making it a good metric to test evaluate against to determine how well the AI agent is performing.

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. A light blocker is shown in 1.2, 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 zero contribution light paths. Therefore, scenes which are significantly affected by blockers will not receive the benefits from traditional Importance sampling and can even benefit more from an uniform sampling scheme [24].

Temporal difference learning methods are better equipped to tackle this problem [9]. As the AI agent described in the previous section learns which directions light is coming from in the scene and concentrates its sampling towards these directions. Directions leading to blockers will have a low value, hence it is unlikely the AI agent will sample rays in these directions.

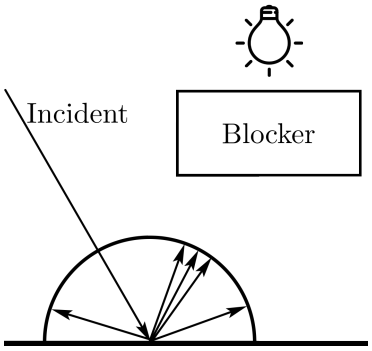


Figure 1.2: An illustration of a light blocker for an importance sampling scheme which does not consider visibility. Each arrow represents a possible direction the light path will be reflected in. Clearly the reflected light path is likely to hit the blocker increasing the likelihood of it becoming a zero-contribution light path.

1.3 Motivation

Rendering time of my graphics engine is not something I have tried to heavily optimise. I instead focus on producing higher quality images using the same number of samples per pixel in light transport simulation in hope that future work will find ways of optimising my methods for speed. Therefore, my work still aims to contribute to the wider goal seen in computer graphics to use accurate light transport simulation in the rendering of photo-realistic images for complex scenes in real-time. Speeding up the methods I use is a large topic in itself, requiring a deep investigation into the best software, hardware, and parallel programming paradigms to use.

1.3.1 Real time Rendering using Accurate Light Transport Simulation

The motivation for using accurate light transport simulation in real-time comes from the clear superior visual quality of images rendered using this techniques, compared to that of scanline methods which are currently used. Where scanline rendering, also known as rasterizing, is the current computer graphics industry standard method for real-time rendering. Not only are renders for a wide range of scenes clearly superior from methods which accurately simulate light transport, but they also scale far better with the number of polygons used to build the scenes surfaces. Therefore, scanline rendering for scenes with extremely complex geometry in real-time is currently not and option. Accurate light transport simulation methods therefore have great potential to be used in ultra realistic simulations for applications such as scenario planning and virtual reality learning environments [22]. Also, many games sell realism as one of their most important features, therefore developing photo-realistic graphics in real-time has clear economic incentive for the video games industry which was valued at over $136 by the end of 2018 [4]. An economic incentive can also be seen for the film industry, where reductions in render times lead to a direct saving on compute time, as well as the hardware required to render full length films.

1.3.2 Recent Developments

Due to the incentives, a large amount of research and investment has been focused on purpose built hardware and Deep learning post-processing methods in an attempt to bring accurate light transport simulation into real-time. NVIDIA's Turing Ray Tracing Technology [21] represents a significant leap in the 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, and 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 produced by a render which simulates light transport, and then reconstruct the image to remove the noise present in the image. Generally these methods rely on pre-trained deep neural networks to reconstruct the image far quicker then it would

take for the renderer to produce an image of the same visual quality [1]. 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 [5]. OptiX has been successfully integrated in to many of the top rendering engines which accurately simulate light transport, such as RenderMan [6] and Arnold [12]. 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.

By using importance sampling by TD learning to reduce the number of samples required for accurate light transport simulation, the same standard of noisy image can be fed into an AI accelerated denoiser with fewer samples per pixel in light transport simulation. Running a rendering engine optimised in this way on purpose built hardware could make accurate light transport simulation for rendering photo-realistic images closer than it ever has been to real-time.

1.4 Challenges and Objectives

As previously mentioned, there already exists an example of TD learning used for importance sampling ray directions in a forward Path tracer [9]. However, further methods of analysis need to be conducted upon this new method to determine its performance for reducing the number of zero contribution l ight paths for different scenes with different settings. It is difficult to assess this as there are infinitely many scenes the method can be used to render, so coming to a clear conclusion is difficult. Another difficult task is that of designing an algorithm for an AI agent to learn what are the favourable directions to sample in a scene are using the deep Q-learning method. This includes some important unanswered questions, such as; is it possible for a deep neural network to model all Q values for a continuous scene space? If so, what is a suitable network architecture? All of which I will describe in more depth in Chapter 3. Then the actual task of implementing such an algorithm in a graphics engine written from scratch is non-trivial due to the technologies which will need to be combined together. The algorithm must also run fast enough to collect large amounts of data from, otherwise a justified conclusion on its performance cannot be made. Therefore, the algorithm will have to be parallelized and run on a GPU.

As previously mentioned, my main goal is to investigate the ability of two different temporal difference learning algorithms ability to reduce the number of zero contribution light paths in path tracing, whilst still accurately approximating global illumination. Which can be broken down in to the following objectives:

1. Reimplement Nvidia's state of the art on-line Temporal Difference learning Path Tracer in order to further investigate its ability to reduce the number of zero contribution light paths.
2. Design and implement an on-line Deep Q-Learning variant of the Path tracing algorithm and investigate its ability to reduce the number of zero contribution light paths sampled.
3. Assess both Nvidia's state of the art on-line Temporal Difference learning Path tracer, and the Deep Q-Learning Path tracer' on their ability to accurately simulate Global Illumination.

Chapter 2

Technical Background

The goal of this section is to give you as the reader a deep understanding of the technical concepts which build on top of one another to reduce the number of zero contribution light paths in path tracing, whilst still accurately approximating global illumination. Initially I introduce Monte Carlo integration to approximate and integral, as well as importance sampling and how it reduces variance in the approximation. Monte Carlo integration is the fundamental method which path tracing relies on, I describe this in detail within the section on Monte Carlo path tracing and physical laws path tracing relies on. With the method of path tracing for rendering images clear, I move on to an introduction on reinforcement learning, as this is what I will be using for importance sampling in path tracing. Here the important concept of a learning the optimal value function is introduced and methods in doing so. In the next section a temporal difference learning rule is applied to the context of light transport simulation. Finally, the Expected Sarsa path tracer introduced by Nvidia [9] is presented, tying in all what has been learnt so far for reducing the number of zero-contribution light paths in path tracing.

2.1 Monte Carlo Integration and Importance Sampling

The theory of Monte Carlo integration and importance sampling underpins how the noise in images rendered by path tracing can be reduced when using the same number of sampled rays per pixel. 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.

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

The idea behind Monte Carlo integration is to approximate the integral by uniformly sampling points (x_i) to evaluate the integral at, and then averaging the solution to the integral for all the sampled points. More formally, basic Monte Carlo integration approximates a solution to this integral using the numerical solution in Equation 2.2. Where $\langle F^N \rangle$ is the approximation of F using N samples.

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

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

An important property of Monte Carlo Integration is that it produces an unbiased estimate of an integral, meaning average of $\langle F^N \rangle$ is exactly the true value of the integral,F for any N [20]. This is presented in Equation 2.4, where p_i is the probability of a given 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*) evaluated at x_i. This is known as generalized Monte Carlo integration and is shown in Equation 2.3, which from here onwards I will refer to as Monte Carlo integration. Dividing by the *pdf* ensures the estimate $\langle F^N \rangle$ is unbiased, as areas with a high *pdf* will be sampled far more, but their contribution weighting ($\frac{1}{pdf}$) to final estimate will be lower. Whereas areas with a low *pdf* 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 Monte Carlo integration is that by the law of large numbers, as the number of samples (N) approaches infinity, the probability of the Monte Carlo 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 Monte Carlo Integration works well for multidimensional functions, as convergence rate of the approximation is independent of the number of dimensions, it is just based on the number of samples using in the approximation. Whereas this is not the case for deterministic approximation methods, meaning they suffer from what is known as the curse of dimensionality. For path tracing, the integral which is approximated is a 2 dimensional function, hence Monte Carlo integration is used.

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

The standard error of the Monte Carlo integration approximation decreases according to Equation 2.7. Where the standard error describes the statistical accuracy of the Monte Carlo 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). Due to Equation 2.7, in practice four times as many samples are required to reduce the error of the Monte Carlo integration approximation by a half. Also, the square root of the variance is equal to the error of the approximation, so from here on when I refer to reducing the variance I am also implying a reduction in the error of the approximation.

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

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

2.1.2 Importance Sampling for Reducing Approximation Variance

Currently I 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 Monte Carlo integration 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 $f(x)$. 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 $p = f(x)\ \forall x \in \mathbb{R}$.

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

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 calculate the are beneath the curve within the Monte Carlo integral approximation. Generally, it requires fewer samples to approximate a function which is closer to being constant function [20].

Most functions are not constant, however it is possible to turn any function into one, and this is exactly what can be done within Monte Carlo 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 [25]. The function f is then divided by f' to produce a constant function, as shown in Equation 2.9.

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

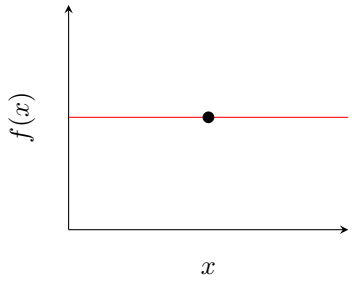


Figure 2.1: Constant Function with a sample point

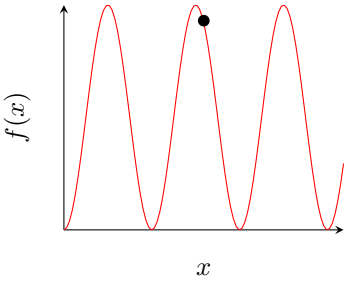
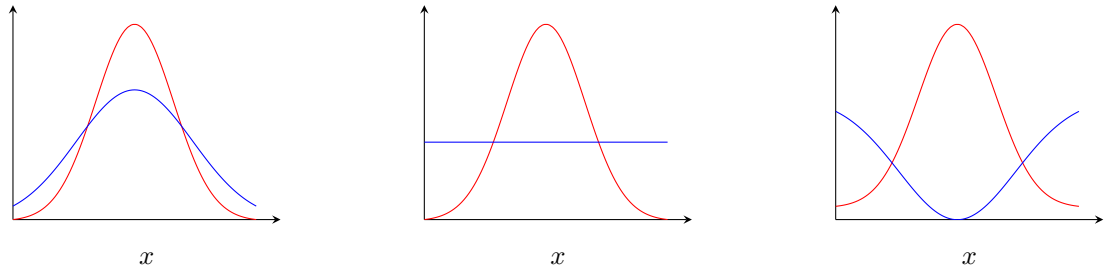


Figure 2.2: Non-linear Function with a sample point

This can be applied to Monte Carlo integration stated in Equation 2.3, by choosing a probability density function (*pdf*) which produces the same output as f for all inputs, but divided by some normlazing constant factor c, keeping *pdf* as a probability distribution. Therefore, we are able to calculate the true value of the integral through Monte Carlo integration as shown in Equation 2.10. Where it turns out $\frac{1}{c}$ is true value for the integral in Equation 2.1.

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

For most cases it is not possible to know the correct probability density function which can convert the Monte Carlo integration problem into integrating a constant function. However, if one has prior knowledge regarding 'important' regions of the functions input space, it is possible to create a probability density function whose shape matches f more closely then a uniform probability distribution. By Important areas of the function input space, I mean areas of the input space which produce a large contribution to the integral of the function. For example in Figure 2.3a, the most important regions are around the top of the functions peak.



(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 probability density function $pdf(x)$ (blue) used in the Monte Carlo integration approximation for the integral of $f(x)$.

As previously explained, Figure 2.3a represents a probability density function which has a similar shape to the function which is being integrated. Therefore the variance in the Monte Carlo 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* in Monte Carlo integration would significantly increase the variance in the approximation compared to that from a uniform *pdf* shown in Figure 2.3b. This is due to regions which have high importance according to the *pdf* contribute a low amount to the integral of the function f, causing the variance in the Monte Carlo integration approximation to rise.

2.2 Monte Carlo Path Tracing

In 1986 James Kajiya introduced the rendering equation and with a it a Monte Carlo integral approximation to the equation [17]. This Monte Carlo approximation is essentially what is known as today as

Monte Carlo Path Tracing. Here, I will give a detailed explanation of the rendering equation and how Monte Carlo Path Tracing approximates the equation by accurately simulating light transport. As Path tracing is a involves a Monte Carlo integral approximation, 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 radiance incident from a point x at a given viewing angle ω. Radiance indicates the power of light emitted, transmitted, reflected or received by a surface from a given viewing angle, with units 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 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 can convert the incident radiance into pixel values to produce realistic rendered images of any given scene. The exact details of how this is done will be described in the next section on the forward path tracing algorithm.

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

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
Ω	= Hemisphere centred around the normal n of the surface, containing all possible angles ω_i
$L_i(y,-\omega_i)$	= The radiance incident from the intersected position y in direction ω_i
$h(x,\omega_i)$	= Returns the closest intersected position by firing a ray from 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 (L_o) from a point is equal to the emitted light (L_e) from the point in the direction, plus the reflected light (the integral) from that point in the direction. The emittance term L_e is simple, it is the light emitted the point x which has been intersected, 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 (L_i). Also, the term L_i is recursive [11], as to calculate the radiance incident in the direction ω_i from some hit-point say $y = h(x,\omega_i)$, a solution is required for $L_o(y,\omega)$. This concept is represented Figure 2.4.

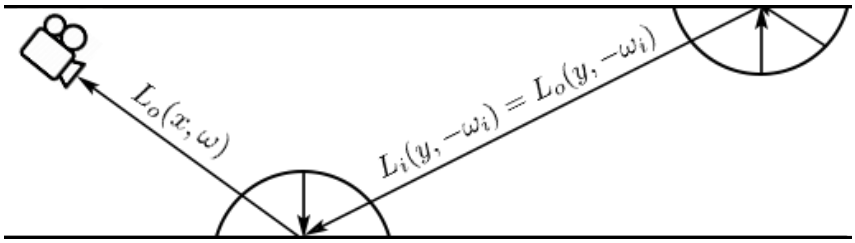
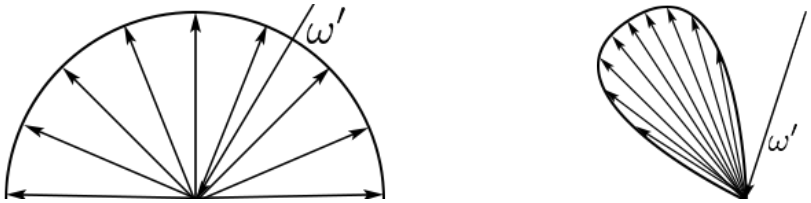


Figure 2.4: A diagrammatic representation of the recursive nature of the rendering equation. The outgoing radiance (L_o) in a given direction ω from a point x requires an estimation of the incident radiance coming from all angles in the hemisphere around the point, that is $L_i(h(x,\omega_i),-\omega_i)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

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 and upholds the law of conservation of energy [13]. Every surface has a BRDF which determines when a ray intersects with that surface at

a given incident direction, the probability distribution over the set of angles the ray can be reflected in. Therefore, querying the BRDF for a surface at point x with incident ray direction ω' and given reflected direction ω, that is $f_r(\omega', x, \omega)$, a single scalar probability value is returned. This probability represents how likely a ray is to be reflected in direction ω. A diffuse and specular surfaces BRDF's are depicted in 2.5. An example of a diffuse material is paper, as it reflects light almost equally in all directions for any angle of incidence. Whilst for specular materials, many metals exhibit specular reflections, where incident rays are reflected in a narrow area around the perfect reflection direction.



(a) Diffuse BRDF (b) Specular BRDF

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

Another way to think about diffuse and specular materials is do they change in appearance depending on the viewing angle? 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.

(a) La Table aux Amours, Marble Sculpture

(b) Cloud Gate, Stainless Steel Sculpture

Figure 2.6: Two sculptures, one made from a diffuse material (left) and the other from a specular material.

A scene comprising of only diffuse materials is generally more computationally expensive to simulate, as one has to simulate rays reflecting in all directions around intersections points with surfaces, compared to a specular scene 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 my 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 ω_i between the normal of the point of the surface intersected with and the angle of incidence. The normal of a surface is the normalized vector that is perpendicular to the surface [32]. The $cos(\theta_i)$ is a weighting for reflected radiance from a point, where the larger the angle from the normal 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.

2.2.2 Path Tracing

In section 1.1 I already gave a high level overview of the how the path tracing algorithm where many light paths are sampled which consist of shooting a ray from the camera, through a pixel and into the scene to calculate a colour estimate. A pixels colour is then determined by averaging all light paths colour estimates. However I did not detail how to get the colour estimate of a light path. This is exactly what the solution to the rendering equation gives, as $L_o(x, \omega)$ gives the outgoing radiance for each sampled light paths initial direction ω and intersection point x. The radiance is then converted into a pixel colour value. Put another way, $L_o(x, \omega)$ is a pixels colour value where ω is the direction of the ray when shot from the camera, through the pixel and into the scene. Then x is the position in the scene the ray first intersects with.

But how does one solve the rendering equation, as often it cannot be done analytically? This is what Monte Carlo integrating in path tracing is used for. Path tracing solves a slightly different form of the rendering equation to that in 2.11. To calculate the reflected radiance at point x in the scene for the angle of incidence ω, it is possible to instead calcualte the integral of all light paths which start at the intersection x and reflect round the scene until a light source is intersected with. The proof behind this is detailed in [15], but conceptually it is simple. Previously the reflected radiance for (x, ω) was given by the integral of the incident radiance on x with respect to the angle of incidence. To calculate this integral one can trace infinitely rays from the intersection point x in all possible directions Ω until they high with a light source, the sum of which gives the total amount of incident light on point x. Therefore, path tracing solves a variant of the rendering equation to estimate $L_o(x, \omega)$ by integrating over all possible light paths starting from x with respect to the surfaces intersected with. It is this integral which is solved via Monte Carlo integration, the details of which are given in Equation 2.12.

$$L_o^N(x, \omega) = \frac{1}{N} \sum_{k=0}^{N-1} L_e(x_0, \omega_0) + (L_i(x_1, -\omega_1) \cdot f_s(\omega_1, x_1, \omega_0) \cdot cos(\theta_{\omega_1}))/\rho_1$$

Such that

$$L_i(x_i, -\omega_i) = \begin{cases} L_e(x_i, \omega_i) + (L_i(x_{i+1}, -\omega_{i+1}) \cdot f_s(\omega_{i+1}, x_{i+1}, \omega_i) \cdot cos(\theta_{\omega_{i+1}}))/\rho_i & \text{if } x_i = \text{Light Source} \\ L_e(x_i, \omega_i) & \text{otherwise} \end{cases} \tag{2.12}$$

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 = Probability density function over reflected ray directions for position x_i and ω_i angle of incidence

In Equation 2.12 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 (N), the closer each pixels approximation will be to the pixels true value as a result of solving the rendering equation. As known from section 2.2.1, the rendering equation follows the physical law of energy conservation, and due to this it accurately models light transport simulation for global illumination. Therefore, the more samples used in the Monte Carlo approximation in Equation 2.12, the more realistic the rendered image [7].

Algorithm 1 describes a forward path tracer which computes one sample inside the summation of Equation 2.12 to find a sampled light paths colour estimate for a given pixel. To render an entire image, this algorithm would be called for each pixel N times (where N is the number of samples per pixel) and the colour estimates of all N rays would be average to find the colour estimate of the pixel.

Algorithm 1: Forward path tracer

```
Function pathTrace(camera, scene, pixel)
    throughput ← 1
    ray ← initialiseRayForPixel(pixel, camera)
    for i = 0 to ∞ do
        (y, norm) = closestIntersection(ray, scene)
        if noIntersection(y) then
            return throughput · environmentLightRadiance(ray, y)
        end
        else if areaLightIntersection(y) then
            return throughput · areaLightRadiance(ray, y)
        end
        (ω, ρ_i, f_s) ← sampleRayDirRandomly(y)
        throughput ← throughput · f_s · cos(norm, ω)/ρ_i
        ray ← (y, ω)
    end
end
```

As path tracing is a Monte Carlo method for solving the rendering equation, Importance sampling can be applied in order to reduce the variance pixel colour estimates. In section 2.1.2 it was shown that by using a probability density function (*pdf*) which closely matches the shape of the function being integrated, the variance in the Monte Carlo estimate is significantly reduced. Applying this to Equation 2.12, the term ρ_i which represents the probability density function for sampling the next ray direction at intersection location x_i with angle of incidence ω_i. Currently you can assume that the probability density function ρ_i is uniform. But this can be modified with prior knowledge regarding which directions are more important for continuing a light path in, where an important direction is one which leads to a high contribution of radiance to the pixel estimate.

The question now is, can one have any knowledge for which directions contribute the most radiance to the pixels colours value? The answer is yes, and there has been a large amount of research in this which resides in the topic of light transport simulation. The simplest example lies within the rendering equation itself, $cos(\theta_i)$. As previously discussed, this term acts as a weighting for the radiance contribution of outgoing light paths. So, the probability density function ρ_i can also be weighted by $cos(\theta_i)$, which is likely to reduce the pixel value variance. There exists many other methods of retrieving knowledge from the scene to use in importance sampling during rendering. For example, irradiance cahcing [2], table-driven adaptive importance sampling [8], and sequential Monte Carlo adaptation [23]. However as discussed in section 1.3, these previous methods do not effectively reduce the number of zero contribution light paths, meaning their ability to image noise for certain scenes is very limited. Instead, Nvidiai proposed that reinforcement learning can be used for this [9] which is the main inspiration for my work. In the proceeding sections I will discuss in detail how it is possible to apply reinforcement learning for importance sampling in light path construction.

2.3 Reinforcement Learning and TD-Learning

Now that it is clear how Importance sampling light paths can be used to reduce variance in Monte Carlo path tracing, it is time to introduce the concept of reinforcement learning as I will be using this to gain knowledge for this Importance sampling. This section aims to give a quick introduction to reinforcement learning and TD-learning to cover all of the background material of the learning methods I will be using, before describing how they are applied to path tracing in the next section.

2.3.1 Markov Decision Processes

Reinforcement learning is one of the three archetypes of machine learning and it is concerned with finding what action should be taken in a given situation, in order to maximise a numerical reward [27]. 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. A MDP is summarised in 2.7 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}

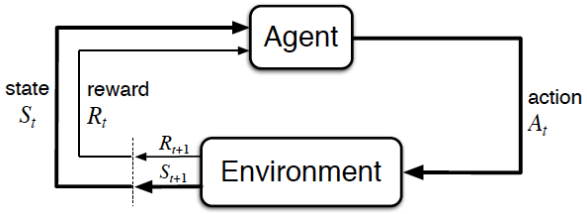


Figure 2.7: Markov Decision Process [27]

An MDP comprises of the following the following tuple:

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

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 which makes it far easier to implement in practice 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* [29]

This is expressed mathematically for an MDP in equation 2.13. 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 is not needed.

$$p(R_{t+1} = r, S_{t+1} = s'|S_t = s) = p(R_{t+1} = r, S_{t+1} = s'|S_1, .., S_{t-1}, S_t) \tag{2.13}$$

2.3.2 Goals and Rewards

The goal thought of for 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 a described as maximising the total amount of cumulative reward received. This is more formally described by the reward hypothesis [27]

> 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 the game (known as the optimal policy), 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. The return is formally defined in Equation 2.14 in terms of a reward sequence combined with the discount factor, which as previously mentioned trades off later rewards for more immediate ones. If a discount factor (γ) 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 [29].

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.14}$$

This formulation works well if the agents interactions with the environment break down easily into sub-sequences [27], where an agent starts in one of a given set of starting states and takes a series of actions to reach a terminal state. 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, but this will be discussed in detail in Section 2.4.

2.3.3 Value Function and Optimality

All reinforcement learning algorithms I 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. The methods I consider are those which use state-action pair value functions, where the value of a state-action pair is defined in terms of the expected return from that state-action pair.

An agent follows a policy π, which determines how the agent will act in a given state. 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 that $A_t = a$ if $S_t = s$. Reinforcement learning algorithms state how an agents policy changes from experience.

The value of a state-action pair (s, a) under a policy π, is given in Equation 2.15 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 due to π. $\mathbf{E}_\pi$ denotes the expected value of a random variable, given that the agent follows policy π. From this, if one were to keep track of the actual returns received for taking a state-action pair, then as the number of times the state-action pair is chosen tends to infinity, the average of the returns will converge on the true expected value of the return for the state-action pair $q_\pi(s, a)$.

$$q_\pi(s, a) = \mathbf{E}_\pi[G_t|S_t = s, A_t = a] = \mathbf{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.15}$$

Now, if you had an AI agent the best way it could perform would be to maximise the expected reward it receives in an episode. In terms of a policies, this is know as the optimal policy which is said to be better then all other policies and 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 2.16.

$$q_*(s, a) = \max_\pi q_\pi(s, a) \tag{2.16}$$

$$= \mathbf{E}[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a')|S_t = s, A_t = a] \tag{2.17}$$

Equation 2.17 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 value of a given state-action pair should be 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. Therefore, if one has the optimal value function, the optimal policy can easily be found by maximising the choice of action a for $q_*(s, a)$ in state s [27].

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 find a good value function for light path direction sampling.

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 2.16. I will not discuss the details of Monte Carlo and Dynamic Programming methods as they are not investigated as part of my 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 [31]. This means TD-learning algorithms can be written in an online learning algorithm [27]. TD-learning can learn directly from experience, as it does not require a true model of the environment in order to learn, unlike Dynamic Programming methods [30]. This means TD-learning is model-free, avoiding the expense of building the true model of the environment. I will now introduce

I will now introduce three different temporal difference learning methods which are required knowledge for the rest of my work.

Sarsa

Sarsa is a on-policy TD method which learns a good state-action pair valuation function $q_\pi(s, a)$. The Sarsa learning rule is presented in Equation 2.18, and I have chosen to present this method first to explain some key concepts TD-learning methods share. Firstly, Q denotes the current value function under policy π, q_π. Therefore the left arrow indicates an update in the value of the current estimate Q. 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 discussed. Finally, Sarsa performs what is known as bootstrapping in the context of reinforcement learning [27]. Bootstrapping is where the estimate of the valuation function (Q), is updated based on some new data by experience, which is the immediate reward R_{t+1}. As well as, the current estimate of the valuation function Q. Sarsa therefore learns from experience, whereby an action is 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.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{2.18}$$

The reasoning behind the name Sarsa is that the method uses every element in the quintuple of events, $Q(S_t, A_t, R_{t+1}, S_{t+1}, A_{t_1})$, 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.

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 direction. 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 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 and perform the update rule in 2.18. 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 will 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 $q_\pi(s, a)$, therefore it is generally a preferred method. The learning rule is displayed in Equation **??**, where rather then 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 in a state based on π, however when it update its valuation function Q, the action chosen to update with may not necessarily be the same as the action chosen. Hence, Q-learning is an off-policy TD-learning algorithm.

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

Expected Sarsa

Expected Sarsa is a TD-learning algorithm which is generally found to be superior to both Q-learning and Sarsa in practice [27]. It is very similar to Q-learning, but instead of using the maximum value of the next state-action pairs it takes the expected value 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 of 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 taken over the current valuation estimate for state-action pairs used.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \mathbf{E}[Q(S_{t+1}, A_{t+1})|S_{t+1}] - Q(S_t, A_t)] \tag{2.20}$$

$$\leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1}) Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.21}$$

2.3.5 Exploration vs Exploitation

Up to now I have formally introduced what reinforcement learning is, including what the optimal value function is and different ways to learn it. As for the TD-learning methods presented so far, I have not discussed any details about the kind of policy an agent should use to select actions in the process of learning. 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 [28]. This is between the following:

- **Exploration**: 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 policy used in TD-learning methods. Initially exploration is the most important to quickly gain a broad amount of knowledge about the environment and 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 [27]. 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 is common practice to choose the current highest valued action, whereas exploration involves choosing one at random. Overtime ϵ is decreased to match the behaviour of increasing exploitation as more knowledge is gained.

2.4 Linking TD-Learning and Light Transport Simulation

With TD-learning learning introduced, I will now link TD-learning to light transport simulation for importance sampling light path directions. Specifically, due to the benefits discussed of Expected Sarsa over Q-learning and Sarsa, I will represent the rendering equation as an Expected Sarsa learning rule. This was orginally introduce by Nvidia in 2017 [9].

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, as shown in Equation 2.24. This implies the action space is continuous:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.22}$$

$$= (1-\alpha) \cdot Q(S_t, A_t) + \alpha \cdot \left(R_{t+1} + \gamma \sum_a \pi(a|S_{t+1})Q(S_{t+1}, a) \right) \tag{2.23}$$

$$= (1-\alpha) \cdot Q(S_t, A_t) + \alpha \cdot \left(R_{t+1} + \gamma \int_{\mathcal{A}} \pi(a|S_{t+1})Q(S_{t+1}, a)da \right) \tag{2.24}$$

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 the direction plus the reflected radiance in the direction.

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

Now, by matching terms from the rendering equation to the Expected Sarsa learning rule in Equation 2.24, Equation 2.25 is formed. Where this new learning rule is designed to approximate the outgoing radiance in direction ω from point x. Therefore, the value of the state-action pair $Q(x = S_t, \omega = A_t)$ is determined by the amount of radiance incident in direction ω from point 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 reflects off surfaces in the scene.

The following lists what each term in the Expected Sarsa learning rule is matched to:

S_t	= 3D position in the scene, $x \in \mathbb{R}^3$
A_t	= Sampling a direction to continue the light path from location x, in direction ω
S_{t+1}	= 3D position of a light ray from 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 direction 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 y, ω_i, $f_r(\omega_i, y, \omega) \cdot \cos(\theta_i)$
$Q(S_t, A_t)$	= Radiance incident on x from direction ω, $-L_i(x, \omega) = Q(x, \omega)$

$$Q(x, \omega) \leftarrow (1-\alpha) \cdot Q(x, \omega) + \alpha \cdot \left(L_e(y, -\omega) + \int_\Omega Q(y, \omega_i) f_s(\omega_i, y, -\omega) \cos(\theta_i) d\omega_i \right) \tag{2.25}$$

Finally, Monte Carlo integration with a uniform distribution for the probability density function can be used to approximate the integral in Equation 2.25. This converts the action space from continuous to n discrete angles and provides a numerical solution for approximating the outgoing radiance from x in direction ω, which is presented in Equation 2.26.

$$Q(x, \omega) \leftarrow (1-\alpha) \cdot Q(x, \omega) + \alpha \cdot \left(L_e(y, -\omega) + \frac{2\pi}{n} \sum_{k=1}^{n-1} Q(y, \omega_k) f_s(\omega_k, y, -\omega) \cos(\theta_k) \right) \tag{2.26}$$

To summarize, I have derived a learning rule for determining the outgoing radiance from a position in a given direction $Q(x, \omega)$. In the context of reinforcement learning this is an approximate value function Q, which attempts to approximate the true value function $q_*(s, a)$. So for light transport, the optimal value function $q_*(x, \omega)$ would be the true value of radiance outgoing from point x in direction ω. The purpose of approximating the radiance $Q(x, \omega)$ is to use this as knowledge for importance sampling ray directions in path tracing to avoid zero-contribution light paths. Exactly how this is done is specified in the next section 2.5.

2.5 The expected Sarsa Path tracer

Monte Carlo Integration, importance sampling, Monte Carlo path tracing, reinforcement learning and the link between temporal difference learning and light transport have all been introduced. So now it is finally time to combine all of these concepts by introducing Nvidia's Expected Sarsa path tracer which uses the learning rule derived in 2.26 for Importance sampling. This method progressively reduces the number of zero-contribution light paths sampled during rendering, reducing the variance in the Monte Carlo approximation of a pixels colour, reducing the noise in rendered images.

2.5.1 The Irradiance Volume

The learning rule introduced in 2.4 requires some sort of data-structure for looking up and updating the outgoing radiance at point x in direction ω, which from here on I will refer to as a Q-value. Therefore, the main requirement of the data structure is that it has some was of representing a discrete set of angles (ω_k) in a hemisphere located at a position x and oriented to the surface normal at x. The Irradiance Volume data structure [14] meets this requirement.

Originally designed to be used for pre-computation of radiance values which are looked up at run-time to approximate global illumination, the Irradiance Volume data structure is essentially a discretized version of a hemisphere which is visually represented in Figure **??**. The image shows the discrete sectors which make up a hemisphere, this was implemented by converting a 2D square grid into the 3D hemisphere shown, which is known as an adaptive quadrature. Where all sectors in the 2D grid have an equal area and a mapping introduced in [26] converts the 2D grid coordinates into a regular discretized hemisphere in 3D space. The mapping ensures the hemisphere sectors remain equal to one another, meaning the discrete set of direction represented by the hemisphere are of equal angles apart from one another.

Each sector of the Irradiance Volume is then used to store the current approximation of radiance from the centre of the hemispheres location x, in the outgoing direction formed by the unit vector from x to the centre of the sector. Therefore, an Irradiance Volume stores the radiance (Q-value) for a given position x (state) in each direction ω_k (action), for all sectors k in the hemisphere located at x. In order to store Q-values across the scene, Irradiance Volumes can be uniformly sampled over the scene geometry. 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 Irradianc volume to position x, then retrieve the Q-value from the sector at index k. Giving a lookup and update time of $O(\log n) + O(1) = O(\log n)$ when using a data structure such as a KD-Tree for nearest neighbour search [3]. Lookup and update procedures from the Irradiance volumes in the scene is all that is needed to apply the Expected Sarsa learning rule in Equation 2.26.

2.5.2 Expected Sarsa Path Tracing

Algorithm

The Expected Sarsa based 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 a reduction in the number of zero contribution light paths. Initially radiance volumes are sampled uniformly across the room with all Q-values initialised to a small constant proportional to the number of sectors on each hemisphere k. This encodes the assumption that initially the radiance in all direction from any given point in the room is equal, as initially there is no prior knowledge of any radiance values $Q(x, \omega)$. With the radiance volumes set up, n sampled light paths are traced through each pixel from the camera and into the scene. The average colour estimate of the n sampled light paths per pixel is averaged to find the colour of each pixel, every rendered frame. The colour estimate of each sampled light path is found by Algorithm 2. The two additions to this algorithm from the forward path tracer in Algorithm 1, are as follows:

1. Once the ray has intersected with a position in the scene y from a position x, update the radiance estimate $Q(x, \omega)$ using the Expected Sarsa learning rule derived in Equation 2.26. This is based on the light emitted from y in direction $-\omega$ and the outgoing radiance estimate in direction $-\omega$ from point y described by the summation. The summation involves summing all Q-values for the closest radiance volume to position y. Each of which are multiplied by BRDF of the surface at y, as well as the cosine of the angle between the sector direction for the Q-value (ω_k) and the surface normal y.

2. The direction to continue the light path in is sampled proportional to the Q-values stored in the closest hemisphere to position y. This is achieved by normalizing the Q-values for the radiance volumes, converting them into a distribution which is appropriately known as the radiance distribution. Then inverse transform sampling [10] is performed to get a direction in the hemisphere to sample a ray in. 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 $P(-\infty < X < x) \leq r$.

Algorithm 2: Expected Sarsa forward path tracer [9]

Function *pathTrace(camera, scene, pixel)*
  $throughput \leftarrow 1$
  $ray \leftarrow initialiseRayForPixel(pixel, camera)$
  for $i = 0$ **to** ∞ **do**
    $(y, norm) = closestIntersection(ray, scene)$
    /* Addition (1) */
    if $i > 0$ **then**
      $Q(x,\omega) \leftarrow (1-\alpha) \cdot Q(x,\omega) + \alpha \cdot \left(L_e(y,-\omega) + \frac{2\pi}{n} \sum_{k=1}^{n-1} Q(y,\omega_k) f_s(\omega_k, y, -\omega) \cos(\theta_k)\right)$
    end
    if $noIntersection(y)$ **then**
      return $throughput \cdot$ environmentLightRadiance(ray, y)
    end
    else if $areaLightIntersection(y)$ **then**
      return $throughput \cdot$ areaLightRadiance(ray, y)
    end
    /* Addition (2) */
    $(\omega, \rho_i, f_s) \leftarrow$ sampleRayDirProportionalToQ(y)
    $throughput \leftarrow throughput \cdot f_s \cdot cos(norm, \omega)/\rho_i$
    $ray \leftarrow (y, \omega)$
  end
end

Monte Carlo Integration

It is important for the modified path tracing algorithm to converge to ensure consistency, meaning all introduced artefacts such as image noise are guarenteed to vanish over time [9]. A decaying learning rate for α can be used to do so, see Equation 2.27. Where i(rv, ω_k) is the number of times the rendering algorithm has updated the Q-value of the Irradiance Volume i for sector k representing angle ω_k.

$$\alpha(x,\omega) = \frac{1}{1 + \text{visits}(x,\omega_k)} \tag{2.27}$$

Due to the importance sampling from addition (2), the probability density function over sampled directions ω at location x is no longer uniform. Instead it is equal to the normalized Q-values for the closest Irradiance Volume to the point x. Therefore the evaluated probability density function ρ_i must also take the shape of the probability distribution to correctly apply Monte Carlo Integration. To do so I have derived an Equation 2.28 to determine the value ρ_i.

$$\rho_i = Q_p(x,\omega_k) \cdot n \cdot \frac{1}{2\pi} = \frac{Q_p(x,\omega_k) \cdot n}{2\pi} \tag{2.28}$$

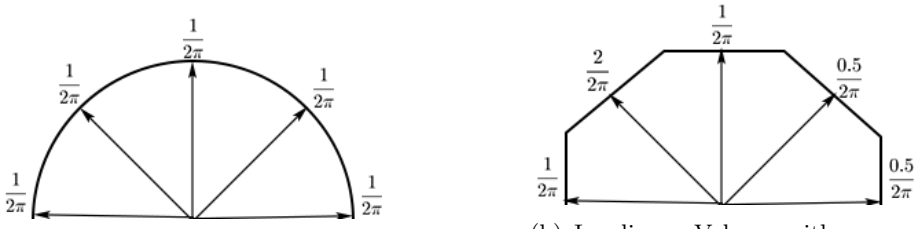
Where:

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

The reasoning behind this value ρ_i is that $\frac{1}{2\pi}$ represents the probability density function (*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 represents a continuous set of angles. Therefore if the probability of sampling a ray in each sector were constant c, $Q_p(x,\omega_k) = c \; \forall k < n$, the *pdf* would remain constant:

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

However, the approximated radiance may vary across sectors, causing the associated *pdf* to vary across sector due to $Q_p(x,\omega_k)$. This was not the case previously prior to importance sampling, where direction were sampled randomly over a unit hemisphere making a the *pdf* is uniform. The diagram in Figure 2.8 highlights these differences.



(a) Hemisphere with a uniform *pdf*

(b) Irradiance Volume with a non-uniform *pdf*

Figure 2.8: A 2 dimension view of a subset of values from two probability density functions (*pdf*). One for a unit hemisphere (left) with a uniform *pdf*. One for an Irradiance Volume (right) with non-uniform pdf. Where the arrows represent sampled directions and the values at the end are the evaluated *pdf* values for each direction.

Consistency

While $Q(x, \omega_k)$ does converge, it does not necessarily converge on the true radiance $q_*(x, \omega_k)$. This is due to the discretization of the action/direction into sectors which make up a hemisphere. If the number of sectors was infinite then the algorithm would converge on the true radiance by the law of large number applied to Equation 2.26. Clearly this is not possible, but later I discuss how increasing the number of sectors on the Irradiance Volume affects the number of zero-contribution light paths. Another issue is 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.

Chapter 3

The Neural-Q Path tracer

3.0.1 Plan

Breakdown

1. State learning rule for deep Q-learning and the difference from deep Q-learning to q-learning. Maybe some of the difficulties associated with deep q-learning versus q-learning, and some of the general advantages.

2. Derive the learning rule for deep q-learning network which I used, once again justifying terms throughout the derivation.

3. Explain concept of eta-greedy policy used. Explain exploration vs exploitation but we will talk about this more later

4. Describe how the current method is used for diffuse surfaces. Introduce the pseudo code for the new algorithm. Give a description of each stage and what it does. Relating back to properties such as bias rendering and pointing out assumption made by the path tracer.

5. Present and explain the network architecture. Explain in depth about how the state was modelled as a point relative to all vertices to give the network information about the position of the vertex relative to the rest of the world compared to passing in a single position. Relate this to Atari games, we get an image showing where we are relative to the world rather than just a single position in the world.

6. Present some results side by side against a default path tracer and Nvidia's reinforcement learning approach. Pointing out aspects of the image and reasoning for certain parts.

Chapter 4

Critical Evaluation

- Here I roughly outline some important implantation details I had to consider which I found should be considered when building a path tracing engine using Algorithm 2.
- Firstly, no details regarding how this algorithm is parallelized were given. This is a very important to take into consideration as nearly all computer graphics algorithms leverage the power of Graphical Processing Units (GPUs) for superior speeds in practice. Therefore if the algorithm cannot be parallelized to high degree, it will never be able to perform as well as other existing rendering algorithms. Every pixel

A topic-specific chapter, of roughly 15 **pages**

This chapter is intended to evaluate what you did. The content is highly topic-specific, but for many projects will have flavours of the following:

1. functional testing, including analysis and explanation of failure cases,
2. behavioural testing, often including analysis of any results that draw some form of conclusion wrt. the aims and objectives, and
3. evaluation of options and decisions within the project, and/or a comparison with alternatives.

This chapter often acts to differentiate project quality: even if the work completed is of a high technical quality, critical yet objective evaluation and comparison of the outcomes is crucial. In essence, the reader wants to learn something, so the worst examples amount to simple statements of fact (e.g., "graph X shows the result is Y"); the best examples are analytical and exploratory (e.g., "graph X shows the result is Y, which means Z; this contradicts [1], which may be because I use a different assumption"). As such, both positive *and* negative outcomes are valid *if* presented in a suitable manner.

4.0.1 Plan

Data to collect

- Build 4 different scenes:
  - Simple geometry, Indirectly illuminated scene: Here both reinforcement learning methods should perform excellently
  - Simple geometry, Directly illuminated scene: Here all methods should perform well
  - Complex geometry, Indirectly illuminated scene: Can both methods do this - will take a lot of training, deeper NN potentially
  - Complex geometry, Directly illuminated scene: Can both methods do this - will take a lot of training, deeper NN potentially
- Number zero-contribution light paths/ light paths that do not intersect with a with a light after n bounces therefore they become irrelevant for all methods with accumulated frames on the x-axis

- Variance in points around the room to train network in order to make training batches as varied as possible (this is a weird one, essentially assessing the fact that we do not need a replay buffer).
- eta-greedy constant for loss curve for training the network & decaying eta-greedy policy graph for the loss as well
- Visual representation of Q-values being higher in directions near light source: Map q-values to hemispheres in the scene and get a close up, clearly indicating its ability to sample in the correct direction
- 1 SPP, 16 SPP, 32 SPP, 64 SPP, 128 SPP, 256 SPP for all three methods on 4 different scenes to evaluate their effectiveness: Assessing accuracy of global illumination approximation
- Limitations: Number of angles which can accurately be learned by the network, accuracy needs to be compared with expected SARSA approach for a single radiance volume at a given point in the scene. Size of the scene which can be learnt accurately.

Preliminary

1. Exploration vs Exploitation for both techniques, exploration can yield to better results plus exploitation does not accurately simulate light, relate to the rendering equation and how light works in the physical world.
2. Show for about 4 different scenes the results for a n different numbers of samples; the images, average path length, number of light paths which actually contribute to the image which are sampled between all techniques. I will have to analyse which reduces the number of zero contribution paths the most, but also still assess if the image is photo-realistic.
3. Also analyse default Q-learnings ability on top of expected SARSA
4. Justify reasoning for choosing to analyse Q-Learning, Expected SARSA and DQN (because they have good results for other cases and TD learning fits the online learning procedure)
5. Assess the number of parameters required, configuration is important for these algorithms, if it is very difficult to get right, then the time spent configuring may not be worth it compared to actually rendering the image. E.g. default path-tracing there are not other parameters apart from the number of samples per pixel, expected SARSA requires the user to specify the memory which is allowed to be used by the program, this requires careful consideration, as well as the threshold the distribution cannot fall below, the deep Q-learning algorithm requires less config but potentially different neural network architectures should be investigated to further reduce the number of zero-contribution light paths.
6. Ease of implementation
7. Parallelisability of each algorithm, path-tracing is far easier to parallelise as it requires minimal memory accesses by the program to infer pixel values, as opposed to expected SARSA which requires many. Deep-q learning has more customizability in terms of parallelizing (needs more research)
8. Memory usage: Path-tracing is minimal, Expected SARSA is unbounded, Deep Q-Learning is bounded by the size of the neural network, but the memory it requires is still significant (needs more research)
9. DQN vs Expected Sarsa: Do not have to wait for an iteration to begin importance sampling on the newly learned Q values for a given point, neural network is continually trained and inferred from. Continuous state space vs discretized required for storage in expected SARSA.

Chapter 5

Conclusion

A compulsory chapter, of roughly 5 pages

The concluding chapter of a dissertation is often underutilised because it is too often left too close to the deadline: it is important to allocation enough attention. Ideally, the chapter will consist of three parts:

1. (Re)summarise the main contributions and achievements, in essence summing up the content.
2. Clearly state the current project status (e.g., "X is working, Y is not") and evaluate what has been achieved with respect to the initial aims and objectives (e.g., "I completed aim X outlined previously, the evidence for this is within Chapter Y"). There is no problem including aims which were not completed, but it is important to evaluate and/or justify why this is the case.
3. Outline any open problems or future plans. Rather than treat this only as an exercise in what you *could* have done given more time, try to focus on any unexplored options or interesting outcomes (e.g., "my experiment for X gave counter-intuitive results, this could be because Y and would form an interesting area for further study" or "users found feature Z of my software difficult to use, which is obvious in hindsight but not during at design stage; to resolve this, I could clearly apply the technique of Smith [7]").

5.0.1 Plan

1. Summarise contributions:
   (a) Implementing a path tracer from scratch to analyse in depth the difficulties and issues that come with Ken Dahm's algorithm. Including memory usage, parallelisation and parameter usage.
   (b) Analysis of different reinforcement learning approaches pitched together clearly on a variety of scenes
   (c) Analysis of neural networks ability to learn the irradiance distribution function
   (d) Online deep-reinforcement learning algorithms effectiveness of learning irradiance distribution function
2. If DQN does not work well provide some further analysis on potential other alternatives which could be used.
3. Future Work: Policy learning to model continuous action & state space
4. DDQN and other deep reinforcement learning strategies

Bibliography

[1] 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.

[2] 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.

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

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

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

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

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

[19] 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.

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

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

[22] 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.

[23] 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.

[24] 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.

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

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

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

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

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

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

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

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

Appendix A

An Example Appendix

Content which is not central to, but may enhance the dissertation can be included in one or more appendices; examples include, but are not limited to

- lengthy mathematical proofs, numerical or graphical results which are summarised in the main body,
- sample or example calculations, and
- results of user studies or questionnaires.

Note that in line with most research conferences, the marking panel is not obliged to read such appendices.