TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY
Department of Mathematics and Computer Science
Data Mining Group

TU Delft
Department Intelligent Systems
Computer Graphics and Visualization Group

Deep Reinforcement Learning for Light Transport Path Guiding

Master's Thesis

Mauro Comi

Supervisors:

Dr. D.C. Mocanu
Prof. Dr. E. Eisemann

Committee Members:

Dr. D.C. Mocanu
Prof. Dr. E. Eisemann
Dr. R. Medeiros de Carvalho

Eindhoven, November 2019

Abstract

Computer graphics strongly relies on physically-based renderings to accurately model light behavior and simulate the real world. State-of-the-art approaches to obtain realistic results are based on path tracing and multiple variance reduction methods. However, these operations do not take into account the configuration of the scene to render; for this reason, they not always perform correctly for indirectly illuminated settings. In this regards, Machine Learning methods have been leveraged as variance reduction techniques for light transport path guiding. Specifically, a recent study introduced an importance sampling strategy that uses Reinforcement Learning to learn the optimal path of the light during rendering. This method, possible on the basis of the structural similarity between two cardinal equations in Reinforcement Learning and rendering, is limited by the discretization of the input data. Our proposed approach overcomes this constraint, learning and approximating the total incident radiance for a continuous set of points in a scene using Deep Reinforcement Learning instead of a tabular policy. The promising results obtained prove that this technique can lead to a better rendering quality than the previous work for specific settings.

Acknowledgment

This work has been an exciting experience that gave me the possibility to explore two fascinating fields, Machine Learning and Computer Graphics. Overcoming the many difficulties I faced would not have been possible without the help and support I received along the way. I first would like to thank my two supervisors, Dr. Decebal Mocanu and Prof. Elmar Eisemann, for their help, suggestions and insights. Their guidance went far beyond the scope of this thesis, inspiring me, boosting my confidence and concretely helping me to pursue a research career. Every discussion we had helped me immensely to develop new perspectives and ideas.

I also want to thank Jerry and Leonardo, who kindly listened to my doubts, my presentations and explained me concepts that shaped my research; I am truly grateful for the time they dedicated to me and their invaluable help.

I want to thank the EIT Digital Double Master's Program, which made these two amazing years in Spain and in the Netherlands possible. I am thankful for the great effort to coordinate such a big project that spans all over Europe, and for the organization of the numerous events that are leaving lifelong memories for all the EIT students.

I want to thank my Vega Friends, and all those who became my second family during these years; a well-deserved *gracias* is for Carlos. I also want to thank all the wonderful people I met at TNO, who welcomed me as a colleague and a friend since the first day.

Finally, I want to express my profound gratitude to the most important people in my life, my family, for their love and invaluable support, and the special person still dreaming with me against all the odds.
This work is dedicated to my grandmother, *"nonna Palla"*.

Mauro Comi, November 2019

Contents

List of Figures

List of Tables

List of Abbreviations

CGI Computer-Generated Imagery

PBR Physically Based Rendering

PDF Probability Density Function

NPG Neural Path Guiding

BRDF Bidirectional Reflectance Distribution Function

MC Monte Carlo

CDF Cumulative Density Function

SSIM Structural Similarity Index

SPP Sample Per Pixel

ML Machine Learning

RL Reinforcement Learning

MDP Markov Decision Process

MSE Mean Squared Error

DQN Deep Q-Network

ReLU Rectified Linear Unit

IQA Image Quality Assessment

SET Sparse Evolutionary Training

ADS Autonomous Driving Systems

TNO Netherlands Organisation for Applied Scientific Research

NEE Next Event Estimation

TD Temporal Difference

ANN Artificial Neural Networks

NICE Non-linear Independent Component Estimation

Summary of Notation

Reinforcement Learning

$\mathcal{S}$	Set of all states
$\mathcal{A}$	Set of all actions
$\mathcal{R}$	Set of all possible rewards
$\mathbb{R}^m$	Set of real numbers defined in a m-dimensional vector space
s, s'	States
a	An action
r	A reward
$\mathcal{T}(s, a, s')$	Transition function
π	The policy, or decision-making rule
$\pi(a\|s)$	The stochastic policy representing the probability of action a in state s
t	A discrete timestep
$Q(s, a)$	Value function
θ	Neural network's parameters
$J(\theta)$	Loss function
γ	Discount rate
α	Learning rate
σ	Standard deviation

Rendering

f_r	Bidirectional Reflectance Distribution Function in the rendering equation
$L_o(x, w_o)$	Reflected radiance from point x in direction w_o
$L_e(x, w_o)$	Emitted radiance from point x in direction w_o
$L_i(x, -w_i)$	Incident radiance, incoming to point x in direction w_i

$L_o(x, w_o)$	Reflected radiance from point x in direction w_o
θ	Co-latitude of the unit sphere
φ	Longitude of the unit sphere
Ω	Solid angle subtended by an area on the unit hemisphere
dw	Differential solid angle
n	Surface normal

Chapter 1

Introduction

Physically Based Rendering (PBR) is an approach in Computer Graphics that simulates accurately the light behavior in the real world to reproduce realistic images of a virtual scene. This technique relies upon Monte Carlo methods and Mathematical Optimization to compute the result. The fascinating aspect of this field is that it combines multiple disciplines, ranging from computer science, electronics and mathematics to art and visual design.

In this chapter, we introduce the main topic of the thesis. First of all, we describe the Business context of this work and its role in various Industrial sectors. This section is relevant for my Minor in Innovation&Entrepreneurship. Then, we motivate the defined goals and list the research questions to investigate. Next, we summarize the main contributions of the thesis and mention the system engineering challenges faced during the implementation of the proposed approach. Finally, we present the outline of the report.

1.1 Business context

Computer-rendered images are pervasive in our daily life. They are used in the Entertainment Industry, making possible the realization of stunning movies and videogames to maximize consumers' immersion in the virtual experience. Computer-Generated Imagery (CGI) is also applied in architecture or engineering, and it is revolutionizing the way products are designed and developed. In this work, we study and propose a novel approach to leverage Machine Learning as a numerical tool for PBR.

While pursuing this thesis, I worked part-time as a Research engineer in the field of autonomous driving at the Netherlands Organisation for Applied Scientific Research (TNO). Autonomous Driving Systems (ADS) rely upon the use of simulators, sensors, and cameras to train their navigation controls and improve object detection algorithms. Simulators have an essential role in ADS because smart agents need to learn the actions and conditions that lead to negative outcomes [49] in dangerous scenarios. Virtual environments help achieve this goal without the expenses of real-world applications. Besides this, cameras are employed on real

Figure 1.1: *Examples of images obtained with Physically-Based Rendering. On the left, an image from the project NVIDIA Drive Constellation, an autonomous vehicle simulation platform that allows photorealistic simulation of virtual scenarios. On the right, a frame from WALL-E (2008) by Walt Disney Pictures and Pixar Animation Studios.*

vehicles to acquire images that simulators cannot provide accurately at this stage. Although this solution is costly because of the devices and personnel required, it is necessary for some specific tasks. For instance, training an algorithm to detect objects on the street demands very precise images of the obstacles that the vehicle may encounter. Another example is lane detection, which is the process of training a vehicle to recognize its position on the road and on the lane. A possible solution to overcome the necessity of physical equipment is the realization of photorealistic simulators able to render the setting precisely and coherently with the physics of the environment. This work is being conducted by many companies and research centers, like the German Research Center for Artificial Intelligence, in German Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI). During a conference at TNO, Prof. Slusallek from DFKI, who also studies photorealistic rendering engines for autonomous driving [36], remarked that a major challenge in this regard is the correct approximation of the light behavior. The goal of this work is to research Machine Learning solutions to eventually overcome this obstacle.

Another extensive application of Computer Graphics techniques is found in the Animation Industry, which fully relies upon the research and development of new approaches to maximize the rendering quality and minimize costs of production. With a total value projected to reach US$ 270 billion by 2020 and an average growth rate of 2% [1], the Global Animation, VFX and Games Industry play a crucial role in the entertainment market. As technology advances, so does the quality of Computer-generated images and Computer Graphics techniques. To stay competitive in such a fast-growing market, Companies need to update continuously their software and hardware with the latest developments. For this reason, researching optimization techniques to render frames fast and accurately is essential for the Business. The entity of the costs of rendering is significant: a single frame can take up to 24 hours to render on a cluster of 24000 cores and 2000 computers [2]. This indicates that even small improvements in the rendering technique could reduce costs drastically.

Techniques developed for CGI are not only limited to image-based fields like Animation or Image processing for autonomous driving. When Machine Learning is used to reproduce the physics of a virtual character, the results obtained can be employed for various tasks. For instance, the Biomedical sector can benefit from it for prosthetic design. As additional proof of the cross-applicability of these studies, it was recently announced that scientists at CERN were interested in the results of light transport path guiding [29] to predict particle collision.

Finally, the need for computational power propels the electronics sector, which finds in the Graphic Industry an unlimited market. In Chapter 6, we report how the future progress with regards to the current GPU architecture might initiate a new research field for sparse matrix computations, enabling faster and more accurate renderings.

1.2 Research aims

Modern path tracing algorithms are based on the interaction between a camera and a virtual scene. Rays are shot from the camera towards the scene and bounce accordingly to the geometry of the environment, its physical properties, and the chosen scattering strategy. The amount of radiance emitted towards the viewing direction is described by the rendering equation [25]. Solving the rendering equation, which is the foundation of path tracing algorithms, is an intractable problem [33]. In other words, there is no algorithm that can precisely solve this equation. Currently, the common practice to approximate its value is using Monte Carlo importance sampling. For the last three decades, researchers have studied variance reduction strategies to improve the efficiency of this technique, such as importance sampling the geometry term of the rendering equation, or sampling light sources directly. However, these traditional schemes are not always efficient since they do not adapt to the scene, or they require before-hand the knowledge of all the light source contributions. In contrast, path guiding is a family of algorithms that adapts ray scattering to a given scene. Therefore, the amount of traced paths necessary for a converged image is reduced. Recently, Dahm et al. studied a methodology to approximate the radiance function inside a scene using Q-Learning [6]; this is a model-free off-policy Reinforcement Learning algorithm that learns a tabular policy to facilitates its decision-making process. The main limitation of this approach is the assumption of a discrete input space, which may hinder the learning process and forces the embedding of continuous coordinates into a discrete data structure.

The goal of this thesis is to propose a novel approach to approximate the incident radiance function in every point of a virtual scene using Deep Q-Learning. This method extends the Q-Learning algorithm and adopts a Deep Neural Network to generate the policy, which is the core of the algorithm's decision making process. The main advantage of a Deep Neural Network over a tabular policy is that it enables the input of a continuous space. We believe that this solution can improve the previous method, since the discretization of the input space

often entails loss of information. A detailed explanation of these two algorithms and their differences is reported in Section 2.2.

As the main objective depends on the Q-Learning method, this approach needs to be accurately studied, implemented and finally extended. In this regards, multiple secondary goals need to be realized as follows.

- Engineer a Q-Learning algorithm to sample scattering directions proportionally to the value function. The resulting PDF used to importance sample the value function obtained with Q-Learning is also consistent with the Deep Q-Learning approach.
- Study the possible integration between two components: the path tracing algorithm and the Machine Learning (ML) module. This is necessary to extend the path tracer with Deep Q-Learning, since these two software frameworks are independent and implemented in different programming languages.
- Study, develop and optimize a Deep Q-Learning algorithm for path guiding, which supports a continuous state space input. This represents the core of our proposed method.
- Investigate metrics to assess the effectiveness of the Deep Q-Learning training process and the quality of the images generated.
- Evaluate, test and analyze the results.

1.3 Research questions

Due to the nature of this work, which has both a significant theoretical component and a practical one, we identified two categories of research questions to reach our goals. While the first category refers to the fundamental research, the second one considers the technical challenges to overcome in order to successfully conduct the study. These questions and their context are addressed in detail in Section 1.4
Fundamental research:

1. *To what extent deriving a probability density function based on Q-values for importance sampling is beneficial for path tracing?* The probability density function based on Q-Learning importance sampling is a detail that is not explicitly articulated in the work conducted by Dahm et al. on Reinforcement Learning for path guiding [6]. In Chapter 3.2, a detailed explanation of the PDF we formulated is presented, along with its two main components. Furthermore, the effect that the derived PDF has on the Q-Learning path guiding approach is described.

2. *How can the incident radiance be learned from a continuous set of possible locations in a scene using Reinforcement Learning?* The use of Q-Learning for importance sampling is limited by the necessity to discretize the input space. Our proposed method approx-

imates the incident radiance function with a Deep Neural Network over a continuous state space. The proposed technique can be found in Section 3.3

3. *Can the application of Deep Reinforcement Learning as a light transport path guiding approach for physics-based simulators result in a better image quality than Q-learning, given the same Sample Per Pixel (SPP) ?* To answer this question, we first need to establish metrics to evaluate the result of the developed importance sampling strategies and estimate the effectiveness of the training process. In Section 4.3.2, these metrics are defined and results based on them are presented. As discussed in Chapter 5, there is not a conclusive answer to this question, because the performance of the Deep Reinforcement Learning algorithm studied depends on factors like the complexity of the scene, the chosen hyperparameters, and the considered metrics.

System and software engineering challenges:

1. *How can we combine a Machine Learning module in python with a C++ application for ray tracing?* The path tracing algorithm used for traditional importance sampling strategies and for Q-Learning was written using the C++11 Standard Library [18]. We made this decision because of the good performance of this coding language and its explicit memory management. On the contrary, the Deep Q-Learning module was designed using Python 3.6, one of the most popular Machine Learning languages. To combine the two software frameworks, many tools and strategies were experimented. In Section 1.4.2, details of the system engineering are illustrated.

1.4 Thesis contribution

Path guiding methods strive to adapt the ray scattering strategy to the characteristics of the scene to render. Figure 1.2 shows the effect of importance sampling based on the policy learned by our Deep Q-Learning algorithm. As one can see, the algorithm does not consider the light position before the training phase (Figure 1.2.b) and rays are randomly scattered according to a uniform sampling. In Figure 1.2.c, the learning process is completed and scattering follows a policy generated for this specific configuration. Therefore, the density of rays that reach the light source is significantly higher when importance sampling is performed on the incident radiance function generalized for each point of the scene.

The contributions of our work can be classified conforming to the same categories identified for our research questions.

1.4.1 Fundamental research

The core of a path tracing algorithm is the iterative process to approximate the rendering equation. Since the solution of this equation is intractable, it is commonly approximated

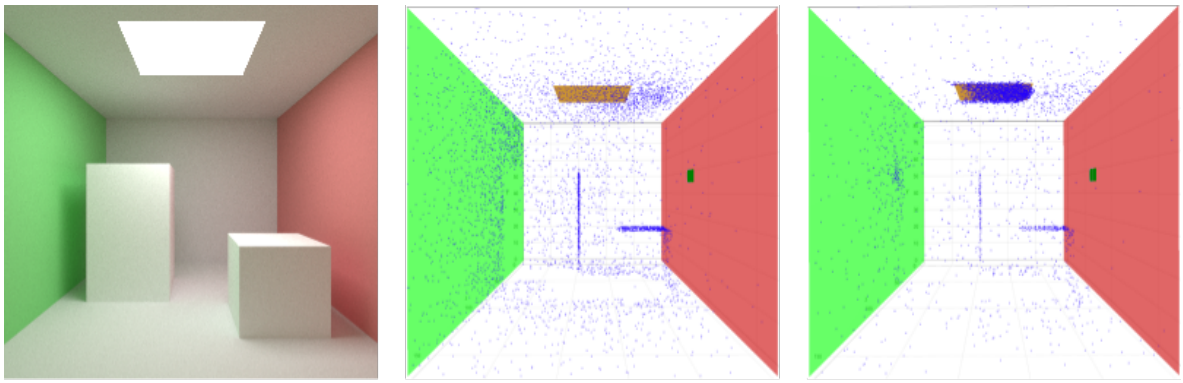

Figure 1.2: *Rays scattering in the scene before and after the Deep Q-Learning training. The source of rays is the dark green square located on the red wall, while the light source is the orange rectangle on the upper wall. Every blue point represents the coordinates of one ray scattered from the source of rays. The leftmost image represents the scene used to show the rays scattering. The figure in the middle refers to the scattering based on uniform sampling. On the right, the agent has correctly learned the incident radiance function inside the scene and scatters rays mainly towards the light source. The scenes displaying the scattering are presented from a different perspective than the reference image.*

by Monte Carlo importance sampling. As stated before, traditional strategies used for this purpose do not take into account specific characteristics of the environments, such as the light position, hence reducing the efficiency of the stochastic sampling. A solution to this problem is the use of Reinforcement Learning to approximate the incident radiance function in every point of a scene [6]. Consequently, rays are sampled proportionally to the learned function. Our proposed method contributes deriving and presenting the PDF for Q-Learning, which considers two different terms. The first is related to the distribution of the Q-values learned during the training phase, which reflects the incident radiance function approximation. The second concerns the uniform scattering probability over a physical patch on a unit hemisphere.

Among the different path guiding algorithms developed for variance reduction, the only one to our knowledge that approximates the incident radiance function with a Deep Neural Network is Neural Path Guiding (NPG) [29]. This method, that was presented during the execution of this thesis, is based on Non-linear Independent Component Estimation (NICE) [7], a framework that models high-dimensional densities faster and more efficiently than competing techniques used for this purpose. This approach was adapted to guarantee the applicability of Monte Carlo integration and it was extended with One-Blob encoding [29], multiple fully connected layers and a piecewise-polynomial warp to achieve higher performance. Although NPG is capable of handling a continuous action space, differently from our proposed method, the computational complexity of all the building blocks characterizing this approach makes it very expensive. Our method contributes to expanding the path guiding technique studied

by Dahm et al. with a Deep Q-Learning algorithm to approximate a PDF for importance sampling. Different from NPG, our proposed strategy embeds the input generated with the One-Blob encoding directly into the Neural Network and only consists of 4 fully connected layers.

To assess the validity of our approach, we present our results based on an algorithmic and an image quality perspective. To understand if the learning process is effective, we study and exhibit the results obtained with 4 different evaluation metrics for each importance sampling strategy: (1) the Structural Similarity Index (SSIM), (2) the Mean Squared Error (MSE), (3) the average amount of bounces inside the scene and (4) the evolution of the accumulated non-valid paths. These evaluators illustrate that Deep Q-Learning yields better results at equal sample count for two of the three settings tested.

1.4.2 Software and system engineering

The Computer Graphics community released over the years multiple complete ray tracing environments. Open-source path tracers, such as PBRT [32] or Mitsuba [16], simulate light physics very accurately, and reflection, refraction, different types of shadows and generation of caustics are reproduced with great detail. However, we decided to implement the algorithm from scratch to guarantee full control of every aspect of it. Indeed, the scope of this work is to investigate whether Deep Reinforcement Learning can be employed to improve traditional importance sampling methods in path tracing, regardless of the complexity of the environment to render. The possibility to only render a very limited set of geometries and basic visual effects do not affect the conclusion drawn with the implementation of our approach, which can be extended to a more complex configuration once it proves to work on simpler ones. Thus, testing our proposed method in a simple setting suffices.

Our main contribution regarding the engineering of the proposed technique is the introduction of two open-source frameworks. The first application is a path tracer written in pure C++11 to train a Q-Learning algorithm, which discretizes automatically the state space depending on the user's parameters. Moreover, every aspect of the training process can be easily customized, and the tabular policy generated is stored locally to be loaded at any time. It will be released soon here: *https://github.com/maurock/Q-tracer*.

The second framework is a path tracer written in Python and Pybind11 [17], a header-only library developed by Jakob et al. that exposes C++ types in Python. The use of this library reduced the path tracing running time by 8 times. The algorithm is based on Deep Q-Learning, encodes the input with One-Blob encoding and stores the policy in the HDF5 format for re-use. It will be released soon here: *https://github.com/maurock/DQN-tracer*. Moreover, we provide a Python script that generates graphs to visualize the scattering inside a scene. The visualization library used for this purpose is Plotly, and an instance of the

obtained results can be seen in Figure 1.2. [13]

1.5 Outline

This work combines two different fields: Computer Graphics and Machine Learning. This is reflected by the structure of the thesis, which is organized as follows.

Chapter 2 shows the background necessary to understand the context of this work. In particular, it is divided into two main sections. Section 2.1 gives an overview of stochastic ray tracing and traditional techniques used for variance reduction. In this section, the mathematical framework required to support the integration of Monte Carlo methods to our proposed approach is described. Section 2.2 first focuses on the basics of Reinforcement Learning to introduce the main concepts of this family of algorithms. Then, a detailed explanation of two model-free off-policy approaches, namely Q-Learning and Deep Q-Learning, is provided. Next, common strategies employed for the optimization of these two methods are clarified. Finally, the state-of-the-art for modern path guiding algorithms based on Reinforcement Learning is reviewed and illustrated.

Chapter 3 presents our proposed methods on a high level. Here, we introduce the mathematical procedure to define the probability density function and the scattering equations. Furthermore, the novelty introduced by our work is stated.

The results of our methodology are reported in Chapter 4. The structure of this chapter reflects the combination of the two aforementioned research areas. Indeed, we propose an algorithmic perspective regarding the validity of the Reinforcement Learning algorithms and a Graphics perspective that addresses the performance of our approach with regards to the image quality.

In Chapter 5, the steps followed to obtain our results are summarized in chronological order. Then, a detailed discussion in view of the outcomes is outlined. The conclusions advanced in this chapter are particularly important because of the intrinsic research-oriented nature of this work.

Finally, in Chapter 6 the limitations of this work are indicated, along with the possible further works.

Chapter 2

Background

This chapter provides the conceptual and mathematical framework to fully understand the context of the thesis. The aim of this chapter is twofold. First, we give an overview on stochastic ray tracing algorithms (Section 2.1). Second, we define the main concepts in Reinforcement Learning, and how this family of algorithms can be used in ray tracing (Section 2.2).

In the first section, we begin with a brief overview on the history of Physically-Based Rendering approaches and the main developments in the last few decades. This description is important because the fundamentals are still used today in the Graphics Industry, approximately unaltered since their introduction. Then, we define the rendering equation in its hemispherical formulation and discuss its features. Next, we outline the basics of Monte Carlo methods, starting from Monte Carlo integration to derive Monte Carlo importance sampling. Finally, we conclude with a high-level introduction of the original naive path tracing algorithm, as well as two common improvements towards variance reduction: cosine-weighted importance sampling and Next Event Estimation (NEE).

In the Reinforcement Learning section, we first describe the key concepts of Reinforcement Learning and the notation we use in this thesis. Then, we present the state-of-the-art application of Q-Learning for path tracing. Finally, we show how Deep Neural Networks are used in Physically Based Rendering.

2.1 Stochastic Ray Tracing

2.1.1 Physics-based approaches

Physics-based approaches used in rendering simulate the light behavior to generate realistic images. Global Illumination algorithms operate in this domain by computing and combining the light directly coming from light sources with its indirect contribution, caused by refraction and reflection. Global lighting effects and the idea of light transport simulation through

ray tracing were first introduced by Turner Whitted in 1980 [47]. Whitted's approach was significantly different from any previous work proposed in Computer graphics, and it marks the beginning of PBR.

A few years later three notable researchers, Cook, Torrance, and Goral, suggested improvements to Whitted's technique taking into account new reflection models [10] and energy exchange. The main problem in these solutions was the high computational complexity that made their use very inefficient. To reduce this issue, ray tracing based on Monte Carlo integration was advanced. Even though this methodology was received at the time with skepticism due to the inevitable noise in the results, it was a breakthrough [32]. Still today, all the main approaches in PBR are based on Monte Carlo integration.

In 1986, Kajiya published one of the most important works in the history of ray tracing [19], introducing the idea of path tracing and formalizing the rendering equation, explained in detail in Section 2.1.2. The main idea behind this work was to compute the incoming radiance for each point in a three-dimensional scene recursively.

After 1997, Veach introduced in his dissertation [43] novel methods such as Monte Carlo Multiple importance sampling, bidirectional path tracing and Metropolis. Research started then to focus on noise reduction and realistic simulation for specific physical phenomena. Over the last few years, the use of Machine learning as a preferred numerical tool, along with the advances in GPU performance, opened the possibility to explore real-time path tracing.

2.1.2 The rendering equation

The rendering equation was introduced by Kajiya in 1986 [19], and it is considered since then one of the most important concepts of Global Illumination algorithms. Equation (2.1) describes the light transport mechanism as a recursive integral equation, where the integrand contains material and visibility properties. The rendering equation computes the exitant radiance at point x and direction w_o. This definition is the *hemispherical formulation* of the equation since the domain of integration is the unit hemisphere aligned by the surface normal and centered in x (Figure 2.1).

$$L_o(x, w_o) = L_e(x, w_o) + \int_\Omega L_i(x, -w_i) \cdot f_r(x, w_i, w_o) \cdot cos\theta_i \, \mathrm{d}w_i \tag{2.1}$$

In the formula:

- f_r is the Bidirectional Reflectance Distribution Function (BRDF), explained in the following paragraph.
- $L_o(x, w_o)$ is the reflected radiance, which is the radiant energy reflected from point x in direction w_o.
- $L_e(x, w_o)$ is the emitted radiance, which is the radiant energy emitted from point x in direction w_o. This quantity is positive when x lies on a light source.

- $L_i(x, -w_i)$ is the incident radiance, which is the radiant energy incoming to point x in direction w_i.
- θ is the angle between the direction of the incident light and the surface normal. Its value lies between 0 and $\frac{\pi}{2}$.
- Ω is the integration domain, representing the unit hemisphere centered in x.

The notation used in this thesis is shown in Figure 2.1.

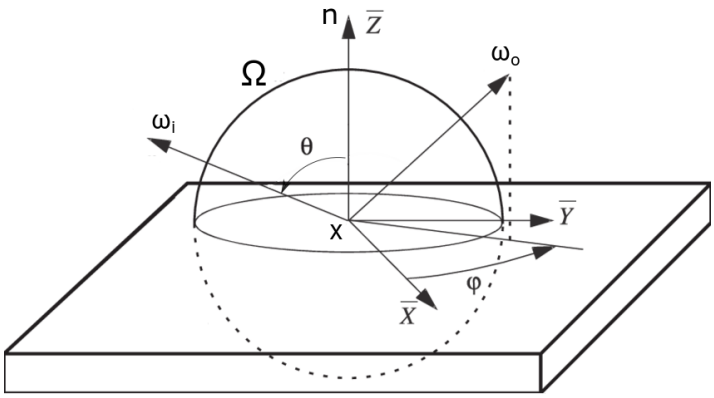

Figure 2.1: *Notation used for the variables in the rendering equation.*

The BRDF represents how light is reflected on a surface. Specifically, it is the ratio of the reflected radiance in a specific direction w, computed for a surface at a point x, and the irradiance through a differential solid angle dw. In our application, we consider all materials as Lambertian surfaces, for which reflection is constant and uniform in all the directions. In other words, a Lambertian surface looks equally bright from any point of view. It is important to mention that in reality surfaces are not ideal, and are all non-Lambertian to some degree.

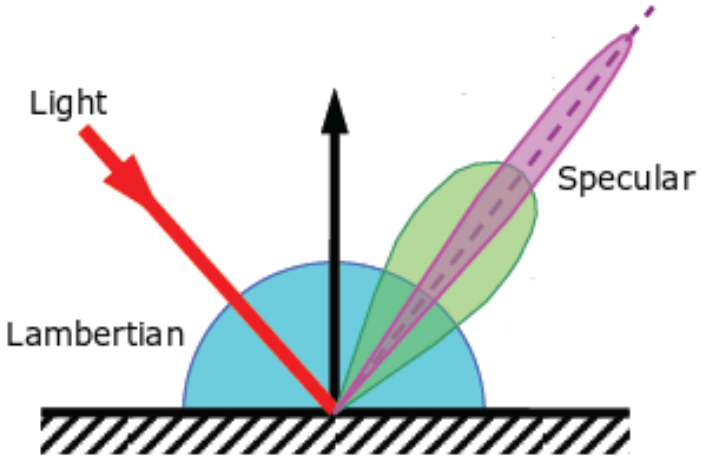

Figure 2.2: *Difference between diffuse BRDF (in blue) and specular BRDF (green and purple).*

The rendering equation was derived using the energy conservation law at a point x. Indeed, the total outgoing radiance in that point is equal to the sum of the emitted radiance and the total reflected radiance. Two important assumptions in this formalization are the absence of participating media and that light propagates instantaneously. These two elements make this formula a simplification of real light transport, even if it leads to an accurate approximation of the light behavior for the majority of applications.

2.1.3 Variance reduction methods for Monte Carlo

Monte Carlo integration

Monte Carlo methods are a family of mathematical techniques used to find approximated solutions through repeated random sampling. These methodologies are usually computationally intensive, and for this reason their popularity in science is strictly correlated with the rise of computational machines.

A powerful Monte Carlo technique, and arguably among the most important ones in Computer Graphics, is Monte Carlo integration. This method allows the approximation of an integrand $f(x)$ over a domain D, by evaluating the function at arbitrary points selected based on a probability density function. It is important to notice that this property guarantees applicability also to discontinuous functions. Specifically, the goal is to approximate the following integral of a function $f(x)$ over some domain D:

$$F = \int_D f(x)\,\mathrm{d}x \tag{2.2}$$

To achieve this result, two important concepts need to be defined: the PDF and the expected value. The PDF $p(x)$ over the domain D is a quantity that represents the density of samples as a function of x. The PDF has two important properties: it is always positive, and its integral over the sampling domain is equal to 1. Mathematically, these properties are formalized as follows:

$$\begin{gathered} p(x) > 0, \forall x \in D \\ \int_D p(x)\,\mathrm{d}x = 1 \end{gathered} \tag{2.3}$$

The expected value of a random variable x is defined as the average of the variable in the long run. The expected value of a function depending on a continuous variable $f(x)$, where the probability $p(x)$ of the variable x is known, is defined by the *law of unconscious statistician* as:

$$E[f(x)] = \int_D f(x)p(x)\,\mathrm{d}x \tag{2.4}$$

Combining Equations (2.2) and (2.4), one can see that the integral F can be also expressed

as:

$$F = \int_D f(x)\,\mathrm{d}x = \int_D \frac{f(x)}{p(x)} p(x)\,\mathrm{d}x = E[\frac{f(x)}{p(x)}] \tag{2.5}$$

The estimation of the integrand can be computed by generating random samples according to the probability $p(x)$. When increasing the number of samples, the average converges towards the expected value. This process, often used to approximate intractable integrals such as the recursive Fredholm integral in the rendering equation, is called Monte Carlo integration. In practice, the domain D is discretized to be computed by an algorithm:

$$< I >= \frac{1}{N} \sum_{i=0}^{N-1} \frac{f(x_i)}{p(x_i)} \tag{2.6}$$

where $< I >$ is called Monte Carlo estimator. An important measure in stochastic methods is the variance of the estimator. The variance σ^2 of $< I >$, described in [8], is:

$$\sigma^2 = \frac{1}{N} \int (\frac{f(x)}{p(x)} - I)^2 p(x) d(x) \tag{2.7}$$

An estimator is said unbiased if it converges towards the expectation as $N \to \infty$.
In contrast, a biased estimator is an estimator for which the expected result, that from now will be referred to as ζ , differs from the parameter it is estimating. Equation 2.8 summarizes these concepts:

$$E[< I >_{unbiased}] - \zeta = 0$$

$$E[< I >_{biased}] - \zeta \neq 0 \tag{2.8}$$

Ray tracers can be either biased or unbiased. Ray tracers based on biased estimators are not always worse than those related to unbiased ones: a biased estimator can introduce less variance, and if the bias is small enough, it might be barely noticeable. The ray tracer implemented for this work is unbiased. A considerable advantage of Monte Carlo integration is that is independent of the integral dimensionality. This property differs from other deterministic methods to approximate intractable integrals, such as the Riemann Sum, and makes this method greatly versatile for many different kinds of applications.

Even though this technique is powerful and vastly used in Computer Graphics, it introduces two main practical issues. First of all, Monte Carlo (MC) has slow $\frac{1}{\sqrt{N}}$ convergence – that is, four times the number of samples is necessary to halve the error. For this reason, finding techniques to reduce the variance independently of the number of samples is compelling.
Secondly, when samples corresponding to low likelihood are retrieved, $\frac{f(x_i)}{p(x_i)}$ is intrinsically very large. This increases the variance, since the sample mean would be skewed away from the true mean. There are various variance reduction methodologies to deal with these issues,

as explained in the following sections.

Monte Carlo importance sampling

A solution to slow convergence is to consider a PDF function $p(x)$ that minimizes the variance of the estimator in Equation 2.6. To do this, we use the Lagrange multiplier technique [24]. According to this method, if the variance of the estimator (Equation 2.7) is the function to optimize, and the boundary condition is Equation 2.1.3, the Lagrangian $Lm(p)$ is expressed as:

$$Lm(p) = \int (\frac{f(x)}{p(x)})^2 p(x)dx - \lambda \int (p(x)dx - 1) \tag{2.9}$$

To find the PDF that minimizes the variance, we need to solve the equation $\frac{\partial Lm}{\partial p(x)} = 0$. For the Chain rule, the derivative $\frac{\partial Lm(p)}{\partial p(x)}$ is equal to:

$$\frac{\partial Lm(p)}{\partial p(x)} = \frac{\partial Lm(p)}{\partial x}\frac{\partial x}{\partial p(x)} = \frac{Lm'(p)}{p'(x)} \tag{2.10}$$

The first component to calculate is then $Lm'(p)$:

$$Lm'(p) = \frac{\partial Lm(p)}{\partial x} = \frac{\partial}{\partial x}\Big[\int (\frac{f(x)}{p(x)})^2 p(x)dx - \lambda(\int p(x)dx - 1)\Big] = (\frac{f(x)}{p(x)})^2 p(x) - \lambda \cdot p(x) \tag{2.11}$$

The second component is simply $p'(x)$. Combining all together, and setting $\frac{\partial Lm(p)}{\partial x} = 0$:

$$\Big((\frac{f(x)}{p(x)})^2 p(x) - \lambda \cdot p(x) \Big)\frac{1}{p'(x)} = 0 \tag{2.12}$$

This means:

$$(\frac{f(x)}{p(x)})^2 p(x) - \lambda \cdot p(x) = 0 \tag{2.13}$$

Finally, the PDF can be expressed as:

$$p(x) = \frac{|f(x)|}{\sqrt{\lambda}} \tag{2.14}$$

For a detailed explanation of this derivation, we refer to [31, 8]. Equation 2.14 shows that, when the PDF is proportional to $f(x)$, the variance is minimized. Intuitively, since this is not always possible, $p(x)$ should match the integrand $f(x)$ as closely as possible [8]. That is, peaks and valleys in the function $f(x)$ correspond to peaks and valleys in the function $p(x)$. This technique is called *importance sampling* and can greatly reduce the estimator variance, which results in image noise. It is important to highlight that an incorrect PDF can result in higher variance, so uniform sampling would be preferred in those cases where the integrand is unknown. This concept is represented in Figure 2.3, where three different sampling methods

are shown.

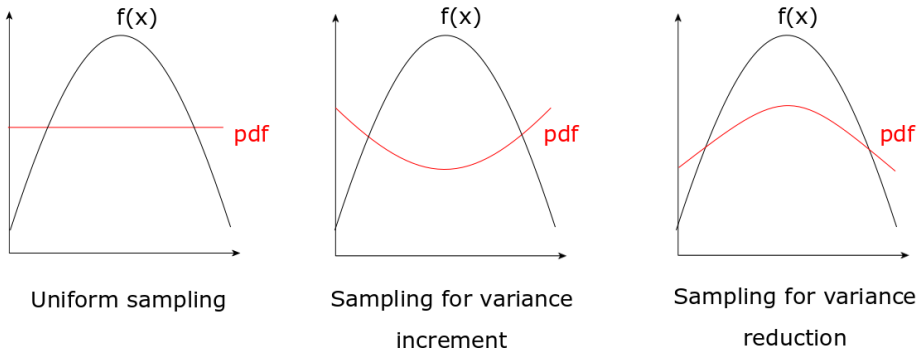

Figure 2.3: *Comparing three different sampling importance functions*

In Computer Graphics, Monte Carlo importance sampling can be applied to approximate the integral equation in the rendering equation:

$$\int L_i(x, -w_i) \cdot f_r(x, w_i, w_o) \cdot cos(\theta_i)\, \mathrm{d}w_i \tag{2.15}$$

2.1.4 Stochastic path tracing algorithms

Path tracing algorithms are a class of algorithms that simulate light bouncing to render a three-dimensional scene. Even though multiple variants of the original procedure have been implemented over the years, the standard set up is composed of three main components: a virtual camera, a scene to render and a single or multiple light sources. The strategy followed for ray scattering determines the efficiency of the algorithm. In this section, three different strategies are explained: uniform sampling in the naive path tracer, cosine-weighted importance sampling and NEE.

Naive Path Tracing

The simplest version of the algorithm attempts solving the rendering equation using basic Monte Carlo integration, explained in detail in Section 2.1.3. The integral component of the rendering equation (Equation 2.1) is evaluated by generating N random directions over the unit hemisphere built on the point in the scene for which the radiance is computed iteratively. Figure 2.4 shows two different paths scattered inside a scene. For each pixel, one or multiple rays are traced from a predefined point representing the camera. The color value of that pixel is then computed through the rendering equation to simulate global illumination. In the image, the ray hits the scene in point p and the algorithm randomly chooses a direction to extend the light path. This operation is repeated until the ray hits the light source, or a stopping condition is met. Path γ bounces once before hitting the light source. Differently, path Φ does not reach the light source. For this reason, since the value L_e representing the

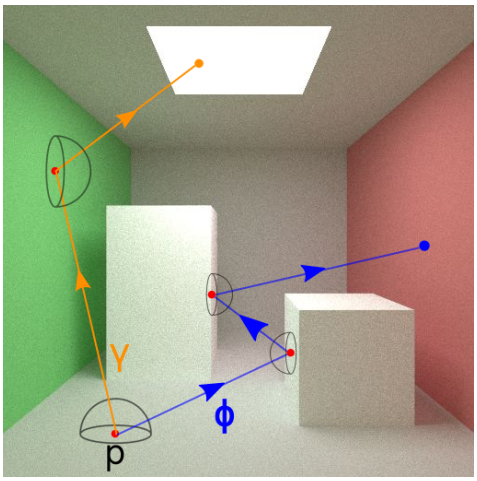

Figure 2.4: *Overview of the path tracing algorithm. From point p, two different paths are traced. Path γ reaches the light, yielding a positive contribution to the pixel color. On the other hand, path Φ stops before hitting the light and results in zero-contribution.*

emitted radiance is zero, this path is said to have zero contribution. The color of the pixel is the average of all the paths starting from p. Smaller *area lights* are harder to reach, and this increases the amount of noise and black pixels generated in the image. One of the main research areas in Computer Graphics, and the ambition of this work, is to reduce the number of paths having zero contribution. Indeed, as one can infer, these paths lead to a significant waste of computational power, since they need to be computed but do not account to the final radiance.

The number of rays scattered inside each pixel is called SPP and the higher this number, the more accurate the approximation of the integral. The next described approaches aim at improving the image quality at constant or lower SPP.

Cosine-weighted importance sampling for Global Illumination

A simple optimization of the naive path tracing algorithm can be obtained through cosine-weighted importance sampling. The mathematical framework of this variance reduction method is illustrated in Section 3.1. As described previously, using the naive path tracing algorithm rays are scattered following a random distribution over the hemisphere aligned by the surface normal. Since the integrand (Equation 2.15), representing the reflected radiance, depends on the cosine of the angle between the incoming ray and the surface normal, scattering with uniform probability over the hemisphere is not efficient, as explained in Section 2.1.3. Indeed, the contribution of the incoming radiance is lower for those rays scattered with higher θ angles. To prioritize those directions that yield higher contributions, we can generate rays according to a probability distribution proportional to the rendering equation, thus to the cosine of θ. As one can see in Figure 2.5, this technique results in more rays scattered

towards the upper section of the hemisphere, while random sampling yields a uniform scattering distribution.

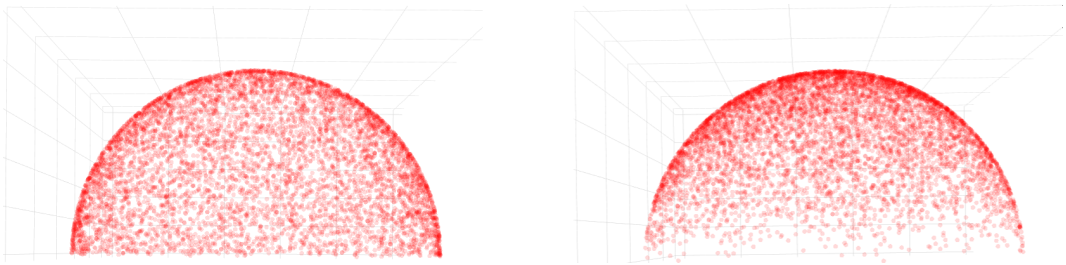

Figure 2.5: *Distribution of rays scattered over the hemisphere. On the left, rays are scattered based on a uniform distribution. Here, rays are distributed evenly on the surface. On the right, rays are scattered based on a PDF proportional to the cosine of the angle formed with the surface normal. Here, it is possible to appreciate a higher density in the upper part of the hemisphere.*

It is essential to mention that cosine-weighted importance sampling does not affect the number of paths with zero contribution. Indeed, the probability density function followed to continue the path is independent of the light source position.

Next Event Estimation

As previously stated, cosine-weighted importance sampling ensures the probability density function to follow the integrand in the rendering equation, but does not take into account the position of light sources. On that account, convergence remains problematic for small light sources. Moreover, since only paths that hit a light source contribute to the computation of the radiance, a long path where no material with positive emittance was hit returns a radiance equal to zero. Explicit light sampling, also known as Next Event Estimation, is a different importance sampling technique developed to overcome these issues.

Figure 2.6 shows a schematic overview of the path tracing algorithm with the implementation of explicit light sampling. The concept of NEE is based on the computation of direct and indirect light contributions separately. The main difference with the naive path tracer is that for every cast ray, one shadow ray is directly scattered towards the light source. This requires to know the exact position of the emittance materials, and it is trivial for few light sources, but becomes computationally very expensive for many light sources.

When computing the direct incident light at a specific point x in the scene, it is more convenient to express the integral over the hemisphere above x as an integral over the visible area seen from this point. If Ω is the solid angle subtended by an area A on the hemisphere with radius r, its value is defined as:

$$\Omega = \frac{A}{r^2} \tag{2.16}$$

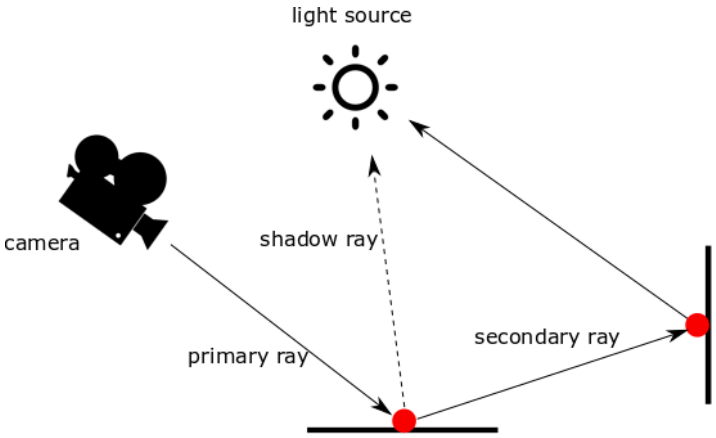

Figure 2.6: *Overview of the path tracing algorithm with explicit light sampling*

This relation allows the approximation of the solid angle subtended by a projected surface area, such as a light source. Specifically, we are interested in the transformation of a hemispherical integral into an area integral. Using Definition (2.16), the differential solid angle dw can be expressed as:

$$dw = \frac{cos\theta dA}{d^2} \tag{2.17}$$

where $cos\theta dA$ is the differential projected area, and d^2 is the distance between the point P and a point on the light surface.

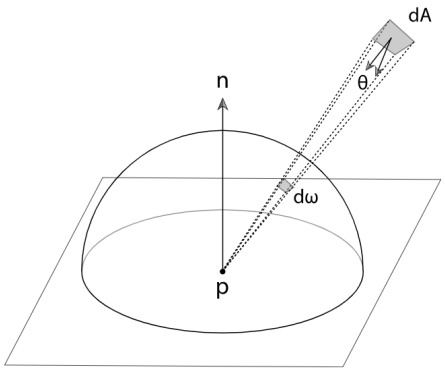

Figure 2.7: *Conversion from differential solid angle to differential projected surface*

This element, combined with the rendering equation, gives the contribution of the direct light $L_{o,d}$ (Equation 2.18). The integral is estimated choosing a random point Q uniformly distributed on the surface of the light source.

$$L_{o,d} \approx L_e(x, w_o) \cdot f_r(x, w_i, w_o) \frac{A \cdot cos(\theta_{iL})}{r^2} \tag{2.18}$$

In the equation, $cos\theta_{iL}$ is the cosine of the angle between the normal to the light surface

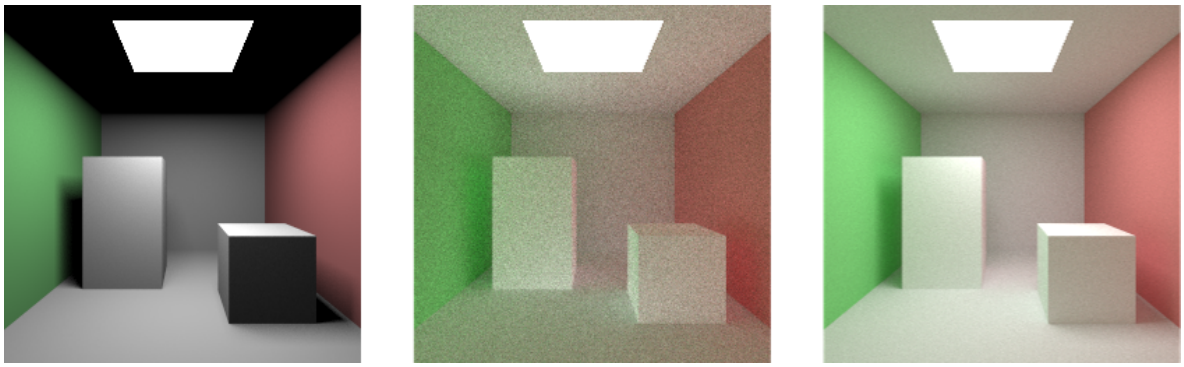

Figure 2.8: *On the left, only direct light contribution. In the middle, only indirect light contribution. On the right, Next Event Estimation with both contributions summed. Every image is rendered with 1024 SPP.*

and the vector linking point P to point Q. Since we only want the direct light contribution, the algorithm returns no radiance if the light is not visible from point P. The contribution of the indirect light is given by the rendering equation after the first bounce into the scene. These two values of radiance are then summed, and the final scene is returned. Figure 2.8 shows the two different images produced by this procedure, that takes into account both the contribution of direct and reflected light. This approach is widely used in the Industry not only for its simplicity and faster convergence, but also because it allows managing direct and reflected contributions separately. It also adds value for artistic purposes, and can help to debug the path tracer.

Among all the techniques used so far, NEE is the one that results in the lowest image noise. Indeed, since for each point in the scene the light source is directly sampled, rays always hit a light area if direct illumination exists for that point. This is the reason why the reference images used for Image Quality Assessment are obtained with this method.

Both cosine-weighted importance sampling and NEE do not consider blockers between the point for which the integration is computed and the light source. This inevitably leads to a high number of scattered rays with zero contribution, unless visibility is considered. The goal of this work is to implement a technique based on Reinforcement Learning to guide the path towards the light taking blockers into account.

2.1.5 Full-reference Image Quality Assessment

Image Quality Assessment (IQA) is a field composed of different methods to predict perceived image quality. It is mainly divided into two research areas: reference-based evaluation, also called full-reference (FR) IQA, and no-reference evaluation. While the former approach needs a base image, the latter estimates image quality without any reference. In this work, we assess the quality of the generated images with reference-based evaluation methods, namely MSE and SSIM. The reference images are generated with Next event estimation using 5120 SPP.

MSE is still today one of the most common metrics to evaluate the quality of an image. It consists in calculating the average of the squared differences between all the reference pixels and the target ones.

$$MSE(x,y) = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{j-1} [x(i,j) - y(i,j)]^2 \tag{2.19}$$

In the formula:

- m, n are respectively the height and width of the images
- x, y are respectively the test and reference images

The main assumption behind the use of this evaluator is that the loss of perceptual quality is directly related to the error signal. Wang et al. [45] proved that this hypothesis does not hold, generating pictures with a very similar MSE score, but very different perceptually. To overcome this limitation, many perceptual image quality assessment approaches have been developed.

To compute the SSIM, structural information is extracted from the image based on the dependencies among spatially proximate pixels. Then, changes in perceived structural information are evaluated and compared.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{2.20}$$

In the formula:

- x, y are respectively the test and the reference image
- μ_x is the average of x
- μ_y is the average of y
- σ_x^2 is the variance of x
- σ_y^2 is the average of y
- σ_{xy}^2 is the covariance of x and y
- c_1, c_2 are two variables to stabilize the division when the denominator is very small

The SSIM ranges between -1 and +1. The lower bound indicates that no similarity can be computed between the two images, while +1 is only scored when the test and reference images are the same. We argue that this method can perform better for our tasks, and we employ it in this work as an alternative evaluation metrics to the MSE, commonly used in Graphics research.

2.2 Reinforcement Learning for Path Tracing

2.2.1 Key concepts

Reinforcement Learning (RL) is a family of Machine Learning algorithms, and its goal is to find a policy that maximizes a reward signal [38]. The main difference with traditional supervised learning is the significantly less amount of information available to the system [12].

The environment where RL operates is simulated as a Markov Decision Process (MDP). Within this framework, the agent is the decision maker that interacts with the environment, performing actions and receiving numerical rewards. An MDP is defined by:

- a set of possible states $\mathcal{S}$ characterizing the agent in the environment.
- a set of possible actions $\mathcal{A}$.
- the reward function $\mathcal{R}(s)$.
- (optionally) a transition model $\mathcal{T}(s,a,s')$.

The role of the reward function is to provide a quantitative measure of the desirability of some states compared to others. A policy $\pi(a|s)$ defines the stochastic agent's behavior, and is a mapping from each state $s \in \mathcal{S}$ to the probability distribution of all the possible actions $\{a_1, a_2, ...a_n\} \in \mathcal{A}$. A very important concept in RL is the balance between exploration vs. exploitation. In order to gather the necessary information, the algorithm needs to explore the environment as much as possible. Usually, the policy is designed to maximize exploration during the initial phase of the training. Over time, the policy starts to take advantage of the gathered information, prioritizing the exploitation of this knowledge over random exploration. A practical application of this notion is described later on in this section. To reach the optimal policy, a large number of different approaches have been derived over the last decade according to the configuration of the environment, and the type of policy to optimize.

One of the most famous RL methods is Q-Learning. This approach was developed by Prof. Watkins [46] and is an *off-policy* tabular learning method, meaning that $Q(s,a)$ will eventually converge towards its real value regardless of the policy that is being followed. To find the optimal policy, the value for each state-action pair, called Q-value, is updated following the Bellman equation:

$$Q'(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \cdot argmax_{a'} Q(s',a') - Q(s,a)) \tag{2.21}$$

where r is the reward for taking the action a resulting in a transition to a future state s', γ the discount rate, and $Q(s,a)$ is the Q-function, often referred to as action-value function. Q-Learning is a *model-free* method, because it does not need the transition function $\mathcal{T}(s,a,s')$ for the MDP environment.

In this work, we use the Bellman equation formalized for continuous action space, which is structurally similar to the rendering equation:

$$Q'(s,a) = Q(s,a) + \alpha \cdot (r + \gamma \int max_{a'} Q(s',a')da' - Q(s,a)) \tag{2.22}$$

Likewise supervised learning, Reinforcement Learning requires a target in order to compute the loss function. In Q-learning, if s is not the terminal state, the target is defined as:

$$target = r + \gamma \cdot argmax_{a'} Q(s',a') \tag{2.23}$$

In case s is the terminal state, which means that by design the episode ends for that iteration, the target is:

$$target = r \tag{2.24}$$

As one can deduce from Equations 2.23,2.24 the algorithm is always trained with the estimation of future expected rewards except for the terminal state. In this event, the target is equal to the reward obtained, and that is the only instance when it corresponds to the ground truth.

Q-Learning proved to work well in many implementations where the state-space and action-space are discrete and low dimensional. The need for limited input and output size stems from the fact that the algorithm stores information in a table containing every combination of the action-state pairs. Deep Reinforcement Learning was introduced to solve this *curse of dimensionality*. Mnih et al. proved that a Deep Neural Network, combined with the principles of Q-learning, could tackle and outperform humans in multiple tasks, even keeping the same architecture for different problems [26]. In Deep Q-Learning, a fully connected artificial neural network approximates the Q-function. Using a gradient descent method, the weights θ of the network are updated to minimize a loss function. A common loss function $J(\theta)$ is the MSE of the Temporal Difference (TD) error between the target and the current estimate of Q-value:

$$J(\theta) = [(r + \gamma max_{a'} Q(s',a')) - Q(s,a)]^2 \tag{2.25}$$

Previously, we mentioned the concept of exploration vs. exploitation. In practice, this can be achieved by a decaying ϵ policy. This technique, formulated by DeepMind researchers, works as follows: first of all, a random quantity q in the range [0, 1] is uniformly sampled. Then, if q is lower than a decaying ϵ value, a random action is selected, hence prioritizing exploration. Otherwise, in case q is higher than ϵ, the action predicted by the policy is chosen, exploiting the knowledge acquired. It is trivial to observe that, if ϵ is initialized to 1, the policy most likely chooses to explore during the initial phase of the training, and gradually increases the importance of the policy over time. The decaying ratio is a critical parameter, and usually it is set to lead to an exponential decrement.

In Deep Q-Learning, the action-value function $Q(s,a)$ is approximated by a Deep Neural Network, different from Q-learning, where Q-values are stored in a state-action matrix. This

feature entails two significant differences during the action-value iteration update. The first distinction is the exact values cannot be retrieved by a look-up table, as in tabular Q-learning, but can only be approximated. Hence, there is no guarantee of convergence [3]. The second difference consists in a simultaneous update of multiple state-action values for each timestamp. In Q-Learning this does not happen, since only one state-action value is updated each time. This characteristic fosters a situation analogous to a dog chasing its own tail: every time the network acts to reduce the TD error, the target moves as a result of its own action. To tackle this and additional issues, we implemented two semi-standard solutions: target network and experience replay.

The introduction of a target network is primarily used to stabilize the learning. In the standard Deep Q-Learning procedure, training to predict the expected reward for the next state and updating the value of executing an action in the current state are achieved by the same network. An improved variation of this algorithm is realized performing backpropagation by the main network and using a copy of it, the target network, to estimate the expected reward related to the next state. The weights of the main network are copied to the target network every n iterations. This technique, applied to our work, proved to improve convergence, and reduce the image noise. The principal drawback of target networks is that they slow down learning because of the delayed value function updates [22].

Lack of convergence is only one among many issues that can occur when dealing with Deep Q-Network (DQN). In RL methods, *catastrophic forgetting* refers to the tendency of a network to forget the acquired knowledge of previously learned tasks. To avoid this, experience replay is used [27]. This technique consists in collecting a tuple of $< s_{t+1}, a_t, r_t, s_{t+1} >$ at the end of each iteration, where t is the current timestamp. Then, a fixed-sized batch of these entries is randomly sampled and the network is trained. This procedure guarantees to train on past experiences and reduce the bias due to the high correlation between subsequent observations. Taking into consideration the improvements presented so far, Deep Q-Learning proved to be suitable for problems characterized by a high-dimensional state and action space.

2.2.2 Q-Learning for importance sampling

In 2018, Dahm and Keller proposed a novel approach to optimize the light transport simulation with a Q-Learning algorithm [6]. The rationale behind it is that traditional techniques, such as cosine-weighted Monte Carlo importance sampling, do not take into account the position of the light source. The proposed approach, on the contrary, attempts at learning the incident radiance in each point of the scene, and guiding the path towards those voxels with the highest expected radiance. It is based on the correspondence between the Bellman equation for continuous action space 2.22, and the rendering equation, described in Section 2.1.2. As one can see, both equations display similarities that can be investigated to match terms. Specifically, this is obtained linking the reward with the emitted radiance $L_e(x, w_i) = r(s, a)$,

and the incoming radiance with the Q-value in the future hitting point $L_i(x, -w_i) = Q(y, w_i)$. Thus, the formula for updating the Q-value is:

$$Q'(x,w) = (1-\alpha)\cdot Q(x,w) + \alpha\Big(L_e(y,-w_i) + \int_D Q(y,w_i) f_s(y,w_i,-w) cos\theta_i \, \mathrm{d}w_i\Big) \quad (2.26)$$

where α is the learning rate, y is the position of the future hitting point, and $cos\theta_i$ is the cosine of the angle between the surface of the future hitting point and the scattering ray for each specific action. The resulting images present lower noise levels compared to those generated with cosine-weighted importance sampling, proving the potential of such an approach. Being Q-learning a tabular method, one of its main limitations is the need for the state-action matrix to be limited in size. The discretization of the input space leads to significant information loss and incorrect results. Indeed, the policy would result in the same distribution of actions for two points belonging to the same state, represented by physical tiles in the real environment. Figure 2.9 shows why this is a limitation: point A and B belong to the same state s, thus rays scattered from these two points follow the same policy. The scattering is performed correctly from point A, but not from point B, due to the distance between them.

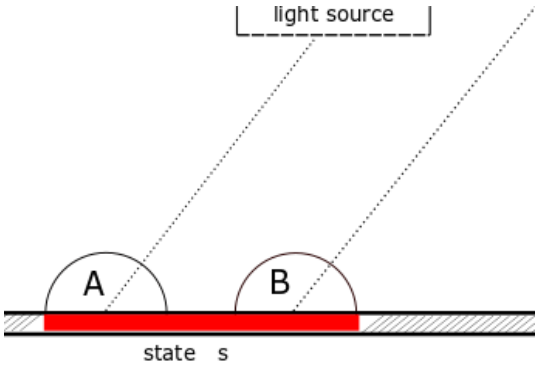

Figure 2.9: *Visualization of rays scattering from two different points belonging to the same state. Since the direction of the scattered ray depends on the same policy, the distance between results in different outcomes. In the case A, the ray hits the light source, thus rewarding the agent. In the case B, the ray does not hit the light source, even though the same action is chosen for the same state.*

2.2.3 Deep Neural Networks for path guiding

The major limitation in the use of Q-Learning to approximate the incident radiance function is the need for a discrete input space. To overcome this restriction, Dahm and Keller investigated a possible method to approximate the solution of the integral in the rendering equation using Deep Learning [21]. As the authors stated, learning the incident radiance was not very practical due to the difficulty in mapping correctly continuous states into a probability distribution over actions in such a setting. Instead, the scene is discretized in a finite number

of states, and multiple fully connected Artificial Neural Networks (ANN) are trained on each of them. The input for each network encodes the relative position inside a voxel, the incoming ray and the orientation of the voxel. The output of the networks is not the radiance estimation, as in the aforementioned work, but the RGB color estimation. This approach is strongly dependent on the geometry and the size of the environment, and it is computationally very expensive because of the multiple networks necessary. This work proved that it is possible to learn the importance of light sources in settings with multiple emitting surfaces. While their results are very promising, additional research needs to be done to reduce the computation necessary to apply deep neural networks in path tracing. Our methods extend this work, evaluating the local light path for importance sampling, in order to learn and approximate the incident radiance in every point.

Additional use ANN for variance reduction was applied by Zheng et al. [50] to learn importance sampling in the primary sample space. In this research, training samples are first obtained from the renderer, and used to generate a target density in the primary sample space. Samples from this density are then drawn and provided to the renderer for path tracing. Similarly to the majority of the research done in Graphics, this method focuses on the distribution of initial samples in the scene.

To our knowledge, the only paper that presents a solution to the approximation of the incident radiance function in a scene with Deep Neural Networks was published by Müller et al. at the beginning of our work. Authors introduced the concept of NPG [29], a methodology that leverages conditional probability densities learned with Deep neural Networks for path guiding. The first building block of the procedure is to apply NICE [7] to transform data to learn the non-linear distribution of the incident radiance function faster. Data generated by this operation are then encoded through the one-blob encoder, which is described as a generalization of the one-hot encoder with no information loss. This encoding method consists in placing a kernel, such as a Gaussian kernel with n-bins ($\sigma = \frac{1}{n}$), at a specific quantity and discretizing it. The encoding generates an array whose size is equal to the number of bins, containing the kernel value in each bin. This procedure is based on the so-called *kernel trick* [48], and allows expressing non-linear functions over the coordinates much more easily. In our work, encoding the coordinates via the One-Blob encoding helped achieve better results. The advantage of NPG is the ability to learn the distribution of the incident radiance over a continuous domain, while our proposed approach is limited by the discretization of the action space. Still, NPG is a very computationally expensive approach that consists of multiple modules, while Deep Q-Learning only embeds a single Deep Neural Network.

Chapter 3

Proposed methods

In this chapter, the main steps of our research are outlined at a high level and linked to their theoretical framework. We first lay out the steps followed to apply importance sampling in practice. Here, we also describe how to generate random samples following a specific probability distribution function from uniform generators. Then, we present how the Q-learning algorithm is used to produce a PDF to follow for importance sampling. Finally, we describe how to extend the aforementioned procedure with Deep Q-Learning, and briefly picture the challenges it entails.

3.1 Cosine-weighted importance sampling

In Section 2.1.4, we presented Monte Carlo integration and explained how importance sampling can help reduce the variance when solving an integral through stochastic samples. In this section, we describe how this technique is adopted in our domain.

To estimate the value of an integral as close as possible to the arithmetic solution, it is essential to get samples from those regions that lead to the highest contribution. As a first step, the approximation of the intractable rendering equation requires to find a proper PDF, which will be used to generate samples. Function 2.1 needs to be expressed in dependency of the angle θ; a common technique in Computer Graphics is to define it as:

$$p(w) = f(\theta) = \frac{cos\theta}{\pi} \tag{3.1}$$

The mathematical derivation of this PDF is reported in Appendix A.2. The reason why this PDF is often chosen for calculate importance sampling is that it is proportional to $cos\theta$ in the rendering equation. Thus, following the equation closely, it helps to reduce the variance, as explained in Section 2.1.3. For ideal Lambertian surfaces, the BRDF represented in the rendering equation as $f_r(x, w_i, w_o)$ is equal to $\frac{\rho}{\pi}$, where ρ is the radiance term. Hence, the integral component of the rendering equation can be approximated using Monte Carlo integ-

ration with the PDF defined in Equation 3.1. This operation also allows micro-optimization of the integrand as follows:

$$\begin{aligned}\int L_i(x, w_i) \cdot f_r(x, w_i, w_o) \cdot cos(\theta_i)\, \mathrm{d}w_i \approx \frac{1}{N} \sum_{j}^{N-1} \frac{L_i(x_j, w_{i_j}) \cdot f_r(x_j, w_{i_j}, w_{o_j}) \cdot cos\theta}{p(w_{i_j})} \\ = \frac{1}{N} \sum_{j}^{N-1} L_i(x_j, w_{i_j}) \cdot \rho_j\end{aligned} \tag{3.2}$$

Once the PDF is defined, the objective is to sample random directions w_{i_j} following this function. In order to do so, it is convenient to express the differential solid angle dw in spherical coordinates. This operation is obtained combining the definition of the differential solid angle with the differential area of the segment of a unit sphere dA according to trigonometry:

$$\mathrm{d}w = \frac{dA}{r^2} = sin\theta\, \mathrm{d}\theta\, \mathrm{d}\varphi \tag{3.3}$$

where θ is the co-latitude of the sphere, φ the longitude, and r the radius of the sphere. These notations are graphically reported in Figure 2.1.
The probability density function can be expressed in polar spherical coordinates as well:

$$p(w)\ \mathrm{d}w = p(\theta, \varphi)\mathrm{d}\theta\, \mathrm{d}\varphi \tag{3.4}$$

and combining it with (3.3), we obtain:

$$p(\theta, \phi) = p(w) sin\theta \tag{3.5}$$

This equation represents the transformation from the distribution expressed in terms of spherical coordinates to solid angle. In order to draw samples according to a non-uniform density, such as a cosine-weighted function, we first need to express the joint probability distribution $\mathrm{p}(\theta, \varphi)$ as two different probability functions only dependent on θ and φ. To compute them, we calculate the marginal and conditional PDF for the two different variables.

$$p(\theta) = \int_0^{2\pi} p(\theta, \varphi)\mathrm{d}\varphi = \int_0^{2\pi} p(w)\ sin\theta d\varphi = 2\,\pi\, sin\theta\, f(\theta) = 2 sin\theta cos\theta \tag{3.6}$$

$$p(\varphi) = \frac{p(\theta, \varphi)}{p(\theta)} = \frac{1}{2\pi} \tag{3.7}$$

Equation 3.7 reveals that the PDF in terms of the angle φ is uniform.
As explained at the beginning of Section 3.1, the goal of this procedure is to generate random samples from a specific non-uniform density, namely a cosine-weighted PDF. Since generating uniformly-distributed samples is computationally efficient, we need to draw samples based on the distributions $p(\theta)$ and $p(\varphi)$ from a set of uniform random values $\xi \in [0, 1]$. To do so,

we apply the inverse Cumulative Density Function (CDF) sampling technique . First, we calculate the CDF for both variables:

$$\xi_1 = \int_0^{\varphi} p(\varphi)\mathrm{d}\varphi = \frac{\varphi}{2\pi} \tag{3.8}$$

$$\xi_2 = \int_0^{\theta} p(\theta)\mathrm{d}\theta = \int_0^{\theta} 2\ sin\theta cos\theta d\theta = sin^2\theta = 1 - cos^2\theta \tag{3.9}$$

These equations relate uniformly-distributed values to our chosen variables. From those, it is finally possible to sample and express φ and $cos\theta$ as:

$$\varphi = 2\pi\xi_1 \tag{3.10}$$

$$cos\theta = \sqrt{1 - \xi_2} \tag{3.11}$$

In order to draw randomly generated samples with the chosen probability density functions for our use case, we need to transform Cartesian coordinates into spherical coordinates and express them as functions of the uniformly-distributed ξ_1 and ξ_2 just retrieved.

$$\begin{cases} x = cos\varphi sin\theta = cos\,(2\pi\xi_1)\sqrt{\xi_2} \\ y = sin\varphi sin\theta = sin\,(2\pi\xi_1)\sqrt{\xi_2} \\ z = cos\theta = \sqrt{1 - \xi_2} \end{cases} \tag{3.12}$$

This system of equations computes the three-dimensional coordinates (x, y, z) of a point on the surface of a unit hemisphere, chosen according to a distribution depending on $cos\theta$ and sampled using a uniformly-distributed generator ξ. Once the coordinates are calculated, a ray is scattered from the center of the hemisphere through this point. This operation is called cosine-weighted importance sampling, and applied to the rendering equation guarantees, in general, a faster convergence compared to sampling from a constant probability density function. Indeed, more rays are scattered towards the upper part of the hemisphere, where the radiance contribution is higher since it depends on $cos\theta$.

To compare this strategy with a uniform sampling over the hemisphere, we also calculate the Cartesian coordinates setting $p\,(\omega) = \frac{1}{2\pi}$. As proven in Appendix A.1, this is the PDF for uniform sampling on a hemisphere. Following the same procedure already applied to compute the coordinates for cosine weighted importance sampling, the system of equations obtained is:

$$\begin{cases} x = cos\varphi sin\theta = cos\,(2\pi\xi_1)\sqrt{\xi_2\,(2 - \xi_2)} \\ y = sin\varphi sin\theta = sin\,(2\pi\xi_1)\sqrt{\xi_2\,(2 - \xi_2)} \\ z = cos\theta = 1 - \xi_2 \end{cases} \tag{3.13}$$

3.2 Q-Learning

3.2.1 Concept

The methodology applied for the implementation of the Q-learning algorithm follows the work conducted by Dahm et al.[6]. The objective of this approach is to find light transport paths that lead to significant contributions for the image and prioritize them. In other words, the objective is to guide rays towards the light sources to minimize their number with zero contribution. This approach is based on the correspondence between the Bellman equation for continuous action space (Equation 2.22), and the rendering equation, described in Section 2.1.2. The resulting equation (Equation 2.26) is used to compute the normalized Q-values for importance sampling and represents the probability density function based on which a specific action is chosen. Indeed, instead of following a greedy approach and always choosing the maximum Q-value per state, actions are chosen proportionally to the Q-values. To understand how this approach works in practice, we first need to define two important elements of our MDP framework: the state and the action space.

3.2.2 State and action space

Due to the limitation of the algorithm, both the state and action space characterizing our setting need to be discretized. The states in our Q-learning approach are represented by a regular grid in the scene. Each tile in the environment represents a state, and its dimension affects the quality of the final image, as well as the duration of the training phase. Figure 3.1 shows an example of states generated in our environment.

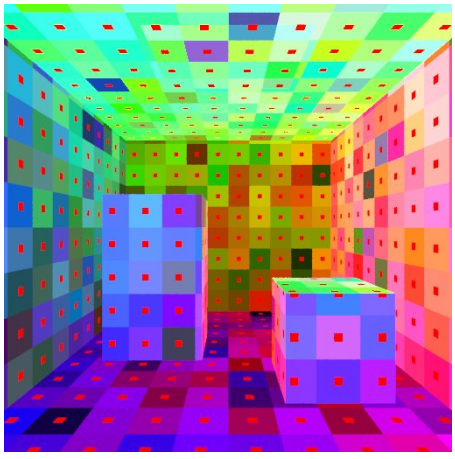

Figure 3.1: *Visualization of the states in the scene. Colors in this figure do not hold any intrinsic meaning, but they are only used to differentiate the states.*

The actions selected from the distribution mapped to each state represent the possible scattering discretized directions. Based on the aforementioned definition, actions are a finite

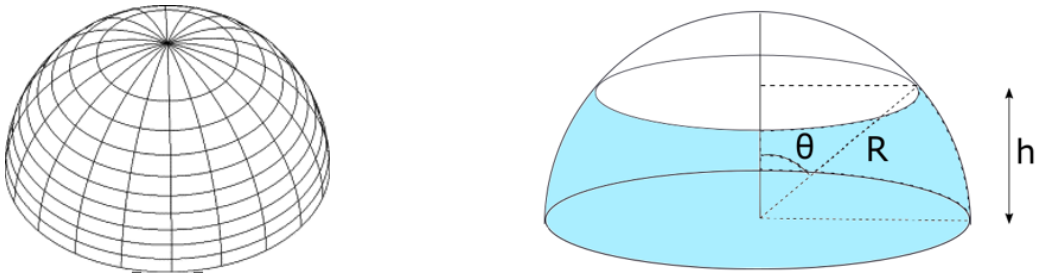

Figure 3.2: *Physical representation of the actions. On the left, the division of the hemisphere in equally sized patches. On the right, the notation used to derive the area of the patches, starting from the spherical zone.*

number of equally-sized patches in the hemisphere aligned with the surface normal. In Figure 3.2, one can see the physical representation of the actions. To divide the hemisphere in equally sized patches, we first consider the integral formulation of the surface area of the hemisphere in spherical coordinate:

$$A = 2\pi R \int_{\theta 1}^{\theta 2} Rsin\theta d\theta = 2\pi R[Rcos\theta]_{\theta 1}^{\theta 2} \tag{3.14}$$

Equation 3.14 shows that the area depends on $Rcos\theta$, which in Figure 3.2 is the height h of the spherical zone. For this reason, if the hemisphere is divided into spherical zones with equal height, the surfaces of these zones are the same. Furthermore, we can split each zone into the same amount of spherical segments to obtain equally sized patches.

3.2.3 PDF for Q-values importance sampling

As explained at the beginning of this section, rays are scattered proportionally to the value function. For this reason, the probability of choosing a specific action a out of all the possible actions is the normalized Q-value for that particular state-action pair:

$$pdf_1 = \frac{Q(s,a)}{\sum_{n=0}^{N} Q(s,a_n)} \tag{3.15}$$

Figure 3.3.a reveals a graphical intuition of this variable: pdf_1 represents the probability of choosing a specific patch, colored in red in the picture, over all the possible patches. The Q-values cannot go below a certain small positive threshold, since all the values need to be non-zero to guarantee exploration and convergence. As far as exploration is possible for all the actions, Q-learning has been proven to converge given an infinite amount of iterations [15, 4].

Algorithm 1: Path tracing based on Q-values importance sampling, following Dahm and Keller's approach [6]. As in stochastic ray tracing, rays are shot from the camera into the scene. The traditional approach is modified since a Q-function is updated for importance sampling during the image generation. The scattering directions are chosen proportionally to the Q-values.

```
function QPathTracer(camera, scene)
    for pixel in screen do
        for sample ← 0 to SPP do
            ray ← initializePrimaryRay(camera)
            for j ← 0 to ∞ do
                (ns, n, f_s, L_e) ← intersectScene(ray, scene) // ns: next state, n:
                 normal
                if noIntersection(ray) or LightIntersected(ray) then
                    radiance_sample ← L_e
                end if
                if j > 0 then
                    (fs, pdf, w_ns) ← scatterProportionalToQ(ns) // w_s: action next
                     state
                    Q'(s, w_s) = (1 − α) · Q(s, w_s) + α(L_e(ns, w_ns) +
                     Σ_{k=0}^{n} Q(ns, w_ns) f_s(ns, w_s, w_ns) cosθ_ns)
                end if
            end for
            ray ← (ns, w_ns)
            radiance_sample ← computeRadiance(f_s, n, pdf)
            s ← ns; w_s ← w_ns // s: state, w_s: action old state
        end for
    end for
end function
```

Figure 3.3: *Graphical intuition of the two components contributing to the PDF for Q-values importance sampling. Figure a) represents pdf_1, the probability of choosing a specific patch, colored in red in the picture, over all the possible patches. Figure b) is related to pdf_2, the probability of scattering a ray through a point p inside the chosen patch.*

The PDF formalized in Equation 3.15 does not completely describe the overall probability of scattering a ray based on the procedure conceived in [6]. Indeed, once an action is selected, a ray is scattered accordingly through a point lying on the surface of the hemisphere. This point is chosen via uniform sampling the area on the hemisphere mapped to the selected

action. This technique avoids biases in the path tracer because it introduces randomness in the process. The probability of uniform sampling a patch is equal to the PDF for uniform sampling on a hemisphere divided by the number of total patches:

$$pdf_2 = \frac{N}{2\pi} \tag{3.16}$$

where N is the number of actions. A detailed derivation of the PDF for uniform sampling on a hemisphere is reported in Appendix A. A graphical intuition of this quantity is displayed in Figure 3.3.b. The PDF used for Q-values-weighted importance sampling for action a in state s is obtained combining 3.15 and 3.16.

$$pdf = \frac{Q(s, w_a) \cdot N}{\sum_{n=0}^{N} Q(s, w_n) \cdot 2\pi} \tag{3.17}$$

The main difference with the previous work is the separation between a learning phase and an active phase. Indeed, while Dahm et al. performed on-line training, our version first preprocesses the tabular policy, and later uses the computed policy to generate the actual image. This operation allows faster results and less bias in the resulting image. The rationale behind this is that the vanishing learning rate decreases exponentially with each state-action visit. After some time, the Q-function converges because the adjustment of the Q-values becomes very small. Thus, updating the Q-function for the whole image generation is pointless since no significant changes are made to the policy. This concept is shown in Section 4.2, Figure 4.2 where the convergence of the value function after a certain number of episodes is displayed. The number of rays to scatter in order to reach a satisfactory policy is determined empirically and treated as a hyperparameter.

3.2.4 Hyperparameters

There are only three hyperparameters to set. The first one is the learning rate α, which in our work is defined as:

$$\alpha = \frac{1}{1 + number\_visits(s, a)} \tag{3.18}$$

where $number\_visits(s, a)$ is a counter taking into account the number of visits to each state-action pair. The vanishing learning rate is a common practice used in Reinforcement Learning, and this specific definition was explored by Keller et al. in similar work on consistency in the light transport simulation [20]. The second parameter is the dimension of the tiles representing the states. Since rays are scattered all over the state surface based on the same Q-value, a finer grid leads to more reliable results. The same reasoning applies to the third parameter, the number of actions. A small number of actions means that the hemisphere is divided into bigger patches, scattering the ray towards a wider area.

Algorithm 2: Path tracing based on Deep Q-Learning. As in stochastic ray tracing, rays are shot from the camera into the scene. The traditional approach is modified since a Q-function is updated for importance sampling during the image generation. The scattering directions are chosen proportionally to the Q-values.

```
function QPathTracer(camera, scene)
    for pixel in screen do
        for sample ← 0 to SPP do
            ray ← initializePrimaryRay(camera)
            for j ← 0 to ∞ do
                (ns, n, f_s, L_e) ← intersectScene(ray, scene) // ns: next state, n:
                 normal
                if noIntersection(ray) or LightIntersected(ray) then
                    radiance_sample ← L_e
                end if
                if j > 0 then
                    (fs, pdf, w_ns) ← scatterProportionalToQ(ns) // w_s: action next
                     state
                    ns_enc ← oneBlobEncoder(ns)
                    input ← createInput(ns_enc, n)
                    Q_s ← network.predictQ(old_input)
                    target_Q ← target_network.predictQ(input)
                    ∇Q ← (L_e + cosθ · Q_s * f_s) − target_Q
                    network.train(∇Q)
                end if
            end for
            ϵ ← ϵ · decay
            target_network ← network
            ray ← (ns, w_ns)
            radiance_sample ← computeRadiance(f_s, n, pdf)
            s ← ns; w_s ← w_ns; input ← old_input // s: state, w_s: action old state
        end for
    end for
end function
```

3.3 Deep Q-Learning

3.3.1 Concept

Deep Q-Learning, explained in Section 2.2, is an extension of the classical Q-Learning approach and for this reason it follows the same structure. The algorithm implemented is improved with the use of a target network and experience replay, both explained in Section 2.2. While the target network technique proved to lead to better results, standard experience replay was tested but did not help significantly. The reason for this might be that consecutive states are highly uncorrelated. Since standard experience replay is usually employed to reduce the linear correlation in the input space, this technique didn't provide any meaningful advantages. Instead, we employed an interesting variation of experience replay, which is presented in a recent work by Giannakopoulos et al. [9]. Here, the authors explore a methodology to deal

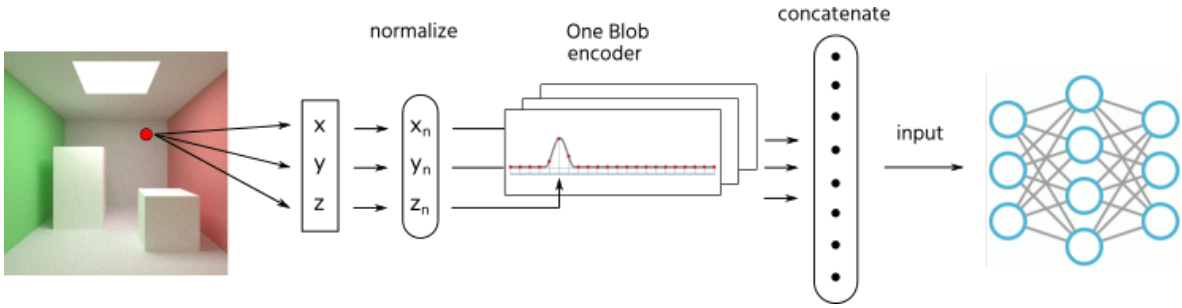

Figure 3.4: *Pipeline showing the input pre-processing of the deep neural network. This procedure consists in coordinates normalization, One-Blob encoding, and finally concatenation of the resulting values in a single space vector.*

with sparse episodic rewards. Specifically, the setting analyzed assumes a single reward given at the end of each episode, defined as a series of observations $\mathcal{O} = \{(s_1, a_1), (s_2, a_2)...(s_n, a_n)\}$, where s_n is the terminal state. Their approach first consists in storing a batch of multiple $< s_t, a_t, r_t, s_{t+1} >$ tuples encountered during an episode. Then, experience replay is applied to the whole sequence of tuples characterizing the randomly sampled episode. The application of this method to path tracing results in slight improvements, because it helps to deal with sparse and delayed rewards.

Due to the lack of computational resources, the policy is learned during the generation of a low-resolution image, usually 128x128. The same policy is then applied to high-resolution images to compute the final results.

3.3.2 State and action space

The rationale behind the use of Deep Q-Learning is its ability to handle a continuous state space. This aspect made this algorithm a popular choice for solving tasks in those applications, such as Robotics or Physics simulation, where coordinates in the three-dimensional space can be processed as continuous variables. In our work, the input of the DQN is not the combination of raw coordinates, but rather a pre-processed version of them. Specifically, the coordinates are first normalized; even though this operation is not always necessary, because of the ability of the ANN to scale the magnitude of the input variables, it is proven to help convergence [14]. Then, an encoding method recently proposed by Müller et al. is applied to the normalized vector. The authors introduced the One-Blob encoding, described in Section 2.2.3. This encoding technique is applied to the ANN input, which in our case is composed of the normalized x, y and z coordinates. Along with the aforementioned methods, this expedient further improved the final results. Figure 3.4 shows how raw coordinates are preprocessed before being received by the Deep Neural Network.

3.3.3 Hyperparameters

Correct hyperparametrization is a complex, delicate and crucial aspect of gradient-based approaches. Choosing the correct parameters for a given task is an open problem. To find the optimal architecture of the network, we need to find the proper number of layers and neurons. This operation follows a trial-and-error approach, starting from a low number of hidden layers and neurons, and increasing them depending on the obtained results. The core of gradient descent methods is to minimize the loss function, moving the parameters θ towards the negative gradient w.r.t the loss function. The learning rate determines to what extent we are adjusting the network parameters. As for the number of neurons and layers, this value is chosen via trial-and-error.

The hyperparametrization of the network's topology is even more involved because it strongly depends on the chosen optimizer. This function determines how weight parameters changes during backpropagation, and it takes into account multiple elements such as the learning rate and momentum. Our chosen optimizer is Adam [23], because it adapts the learning rate to the input parameters. As mentioned in Section 3.2, a common strategy to improve convergence is to set a vanishing learning rate (Equation 3.18), which is dependent on the number of visits to each state-action pair. Even though this is not possible in a continuous state space, where there are no discrete states that are constantly visited, we partially replace the benefits of the vanishing behavior with Adam's learning rate adaptation.

In order to ensure non-linearity in the training process, each layer needs to be mapped to an activation function. In our application, the selected one is Rectified Linear Unit (ReLU) [11]. ReLU is an activation function that works by thresholding the output of a hidden layer at 0, i.e $g(x) = max(0, x)$. This algorithm has an important advantage besides the introduction of non-linearity: its output is not limited to a specific range, as it happens using different activations, and this suits the goal of our network; we want to learn the incident radiance in each point of the scene, and avoid an upper and lower boundary of the learned values that would introduce bias in the rendering.

In Section 2.2, the concept of decaying ϵ policy was defined. The strategy described, as it is often used in Deep Q-Learning, is to decrease the ϵ value exponentially. In our work, the ϵ is decreased linearly in order to extend the exploration time. This approach is justified by the need to approximate the incident radiance for all the actions, and not only for those that maximize the reward signal.

The last hyperparameter is the number of Gaussian bins to encode the input through the One-Blob encoding. Even though it is true that increasing the input dimensionality can significantly help the network expressing non-linear functions, it may be harder to find the global optimum of the loss function when the input vector space is too big. The trade-off is reached via trial and error.

Chapter 4

Experimental setup and Results

In this chapter, we first describe the different scenes used to compare the algorithms and the details of our implementation. Then, we present the results from two different perspectives: Section 4.2 focuses on the algorithmic convergence of the two Reinforcement Learning approaches, Q-Learning and Deep Q-Learning, while Section 4.3.2 considers the Image Quality Assessment.

4.1 Setup

The methodology explained in the previous chapter has been applied to three different scenes, shown in Figure 4.1.

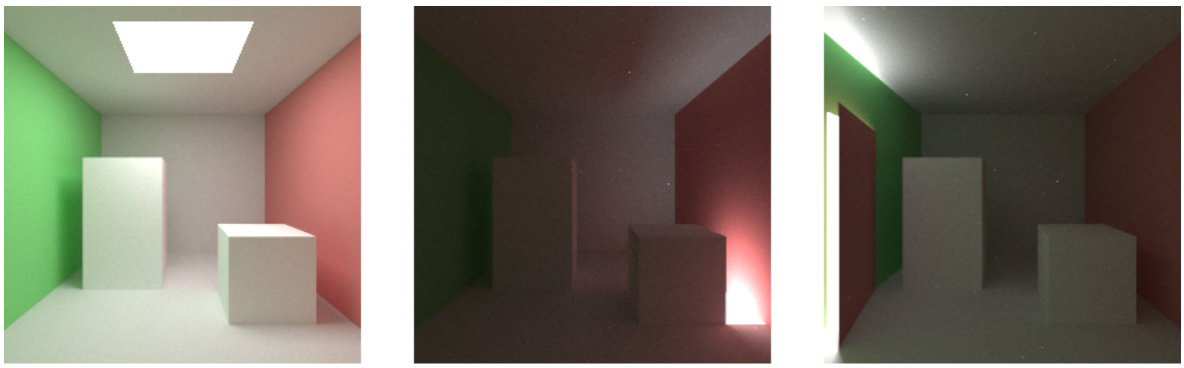

Figure 4.1: *Scenes designed to compare the path tracing algorithms. The different light position introduces increasing challenges for the path tracer. On the left (Box), the light source is directly reachable by the majority of coordinates in the scene. In the middle (Sunrise), the light source is partially blocked, but it can still be directly reached by a relevant set of coordinates . On the right (Door), a door blocks the majority of points to directly reach the light.*

The different light surfaces and positions in these scenes have been designed to introduce

increasing challenges for the path tracer. The characteristics of these configurations are outlined as follows:

- Box: the first scene from the left is the simplest one and the fastest to render. The light is directly reachable from almost every point in the environment. This results in a bright image, where few soft shadows are present. It's interesting to notice the red and green reflections on the sides of the cubes.
- Sunrise: in the second scene, the light source has a small surface and it is partially blocked by an obstacle, that prevents a significant amount of rays from reaching it directly. Nevertheless, the points on the right side of the scene can directly reach the light source.
- Door: the last scene is the hardest to render, even though it features the biggest light source area. Indeed, the light is almost completely blocked by a door and can only be reached through a narrow opening. The highly-stochasticity introduced by the state and action space discretization lowers the chances to reach the light-emitting surfaces despite the choice of best actions. This issue is detailed in Chapter 5.

In the last two cases, only a very limited percentage of rays would reach the light source if scattered according to a random distribution. Since the main contribution is given by indirect illumination (Section 2.1.4), path guiding can be very beneficial.

4.2 Results: An algorithmic perspective

4.2.1 Analysis of the selected parameters for Q-Learning

Section 3.2.2 reports a high-level description of the states and actions in the proposed methods. In this section, we explain the details of these values in our implementation of Q-Learning. Specifically, each state $s \in \mathcal{S} \subset \mathbb{R}^6$ is identified by the 3 normalized coordinates of the hit point and by the 3 coordinates of the surface normal; the virtual scene is discretized in 3644 states in total. Each state is mapped to 72 actions, $a \in \mathcal{A} \subset \mathbb{R}$ which represent the scattering directions from the center of the hemisphere aligned by the surface normal.

The learning process is stopped after that 3 rays are shot through every pixel in the image at 128x128 resolution. The amount of episodes for training is a hyperparameter, and 3 passes prove to perform well in the first scene. The main assumption of stopping the training after 3 passes is based on the fact that Q-values for every state in the scene converge on average at the same rate. Since the vanishing learning rate decreases by the same amount for all the states, this hypothesis is reasonable. We decided to keep the same number of passes for every scene, even if Door and Sunrise are more complex than Box, and additional training episodes might be necessary. In Chapter 5, we describe the rationale and consequences of this choice. Since we apply the same reasoning when Deep Q-Learning is employed, we believe that the

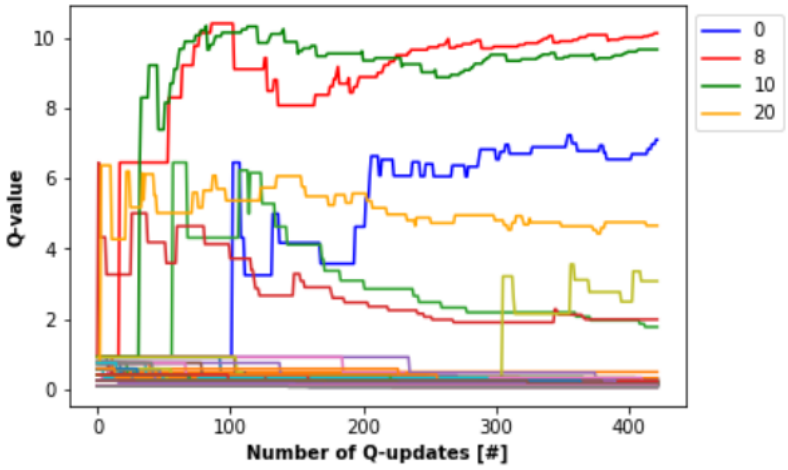

Figure 4.2: *Evolution of the Q-values during the Q-Learning training process for a specific state. In the plot, the Q-values against the number of Q-updates are shown. Every time the state is visited, the Q-values are updated. As every line in the plot refers to a specific action, there are 72 lines in total. The legend shows the 4 actions resulting in the highest values in the last iteration. It is possible to observe a satisfactory Q-values convergence due to the vanishing learning rate after about 400 updates, or 3 rays samples inside every pixel of an image at 128x128 resolution.*

comparison is still fair and unbiased.

Figure 4.2 shows the update of the Q-values for one specific state, referred to as s_Q, during 3 passes. As one can see, all the Q-values converge after about 400 updates. The two best actions in the image, colored in blue and yellow in the plot, converge towards 10. This value represents the cumulative expected reward that the agent would obtain choosing one of these two actions for that specific state. In Section 2.2.2, we stated that the reward $r \in \mathcal{R}$ obtained when the scattered ray hits a light source is equal to its emitted radiance $L_e(x, \omega)$. In our experiments, the emitted radiance L_e is set to 12 for light sources and 0 otherwise, thus the domain of the reward space $\mathcal{R}$ is formalized as $\mathcal{R} \subset \{0, 12\}$. Since the maximum reward that the agent can obtain from the MDP environment is 12, we can deduce that an expected reward close to 10 presumes a high probability of reaching the light source. Moreover, it is important to point out that the Q-values adopted for importance sampling cannot be null, as explained in Section 3.2.1. Therefore, a threshold equal to 0.1 is set and Q-values smaller than the threshold are automatically replaced with 0.1. This way, the full exploration of the action space is guaranteed. Taking into account that sampling is proportional to the value function $Q(s, a)$, the actions that lead to the highest expected reward, mapped to Q-values approximately equal to 10, are 100 times more likely to be chosen by the policy $\pi(a|s)$ over the least favorable actions in the instance presented in Figure 4.2, with a Q-value equal to 0.1.

Results are shown in Section 4.3, Figure 4.9, where the images are rendered using 32 SPP

at 256x256 resolution.

4.2.2 Analysis of the selected parameters for Deep Q-Learning

The fully connected Deep Neural Network developed to approximate the policy has an input size of 51, 4 hidden layers each with 1000 neurons, and an output layer with a size of 72 (Figure 4.3). Formally, this entails that each state is defined as $s \in \mathcal{S} \subset \mathbb{R}^{51}$ and each action as $a \in \mathcal{A} \subset \mathbb{R}$.

The learning rate is 0.0001, and the decaying ϵ-policy is linear, with the ϵ value decreasing from 1 to 0.01 in 28000 episodes. The total number of training episodes is 80000. The Gaussian kernel used for encoding the input data with the One-Blob encoding has 16 bins, and it is applied to each of the 3 normalized three-dimensional coordinates, accounting then for 48 variables out of the 51 in the state space. The additional values in the input space consist of the 3 coordinates of surface normal hit by the ray.

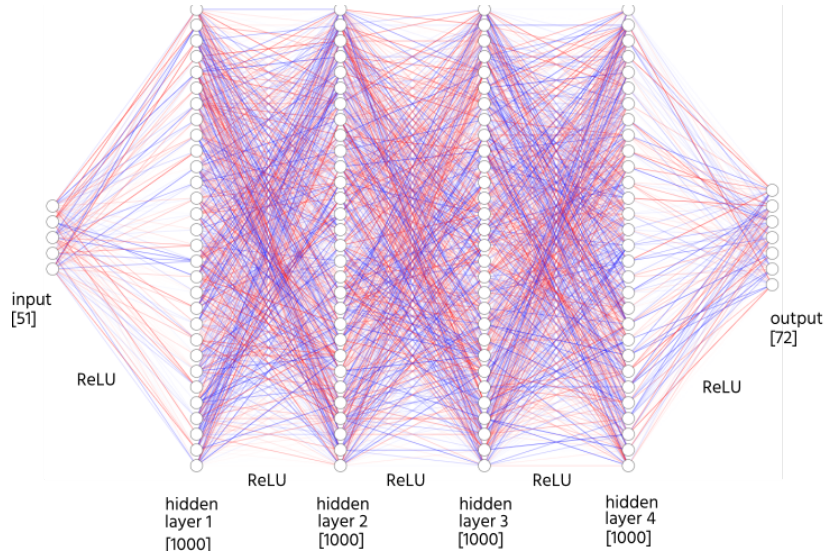

Figure 4.3: *Structure of the Deep neural Network: 4 hidden layers each with 1000 neurons, and all the layers are associated with a ReLU activation function. The picture only displays a high-level overview of the network architecture adopted, and the color of the connections does not hold any intrinsic meaning.*

In Section 2.2.1, we described the use of two identical Deep Neural Networks during the training process. The principal network is used to predict the value function for the current state and its weights are updated for every iteration, or in other words for every ray sampled. A second network, called target network, is employed to predict the target value and its weights are synchronized with the original network only at the end of each episode, which consists of multiple iterations. In Section 4.3, we demonstrate that this technique improves the overall image quality.

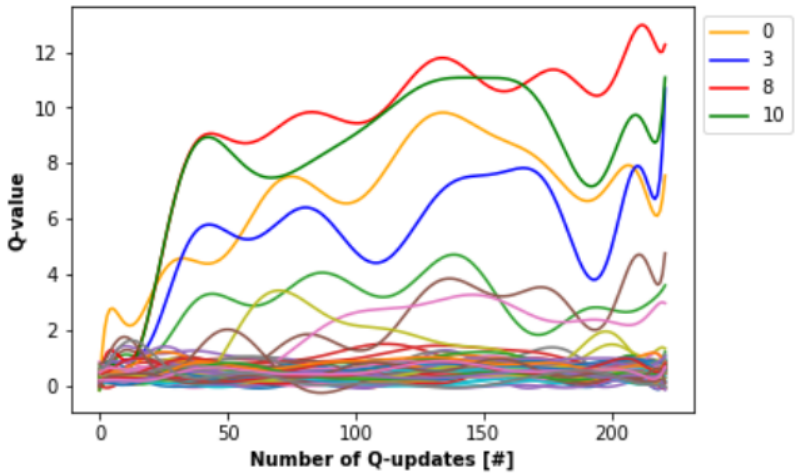

Figure 4.4: *Evolution of the Q-values during the Deep Q-Learning training process for a specific state s_{DQN}. Everytime a ray is scattered from a coordinate belonging to s_Q, the Q-values for all the actions are stored and plotted. Every line represent an action, and for readability each curve has been smoothed using a polynomial fitting with a degree of 15. This is necessary because of the high stochasticity of the framework that causes instability in the progress of the Q-values. The legend shows the 4 actions resulting in the highest values in the last iteration.*

.

Analyzing the evolution of Q-values during the training process is slightly different from the procedure performed for Q-Learning. Indeed, while the value function $Q(s, a)$ in Q-Learning is only updated when the agent visits s, in Deep Q-Learning the weights of the network change at every iteration regardless of the visited state. Figure 4.4 displays the evolution of the Q-values during the Deep Q-Learning training process: everytime a ray is scattered from one of the coordinates belonging to s_Q, the Q-values for all the actions are stored and plotted. This visualization choice allows the comparison between the two plots (Figure 4.4, 4.2), which would not be possible if the Q-values were stored at every Q-update due to the very noisy signal. The figure reveals a large number of actions leading to a high expected reward. The state s_{DQN} considered to elaborate this plot is obtained by sampling a random three-dimensional coordinate belonging to the state s_Q, mentioned in the previous section. Furthermore, because of the instability caused by the absence of the vanishing learning rate, the plot was smoothed using a polynomial fitting with a degree of 15 for readability. It is also important to observe that, for a readability purpose, the Q-updates on the *x-axis* do not represent every update to the value function, which is adjusted at every iteration. Instead, $Q(s, a)$ is plotted every time a ray hits a point included in the physical tile representing the formerly s_Q used for the Q-Learning analysis. These elements are discussed in Chapter 5.

4.2.3 Conclusion on algorithmic convergence

The main difference between the two Reinforcement Learning approaches with regards to the training process is their convergence rate. Indeed, Q-values learned using the Q-Learning algorithm feature a smoother evolution than those learned with Deep Q-Learning. The reason is that the learning rate used in Q-Learning decreases over time for every Q-update. As the learning rate converges towards zero, the contribution of the new information acquired by visiting new states becomes irrelevant. In Deep Q-Learning, estimating the influence of the step size during the training process is harder. Because the optimizer employed is Adam, the learning rate varies based on the mean and the uncentered variance of the weights' gradients [23].

Despite the distinct convergence rate for the two approaches, the value function displays a similar behavior for the specific states s_Q and s_{DQN} examined previously. This behavior can be seen in the distributions reported in Figure 4.5, that reveal peaks corresponding to the same directions. Moreover, to give a better intuition of the meaning of these distributions, the visualization of rays scattering for both methods from the considered s_Q and s_{DQN} are shown. Since the actions mapped to the highest Q-values are the same for both the approaches, rays are scattered in a very similar way. This information, which cannot be deduced from the evolution of Q-values due to very noisy signals in Deep Q-Learning, is clearly visible in this plot.

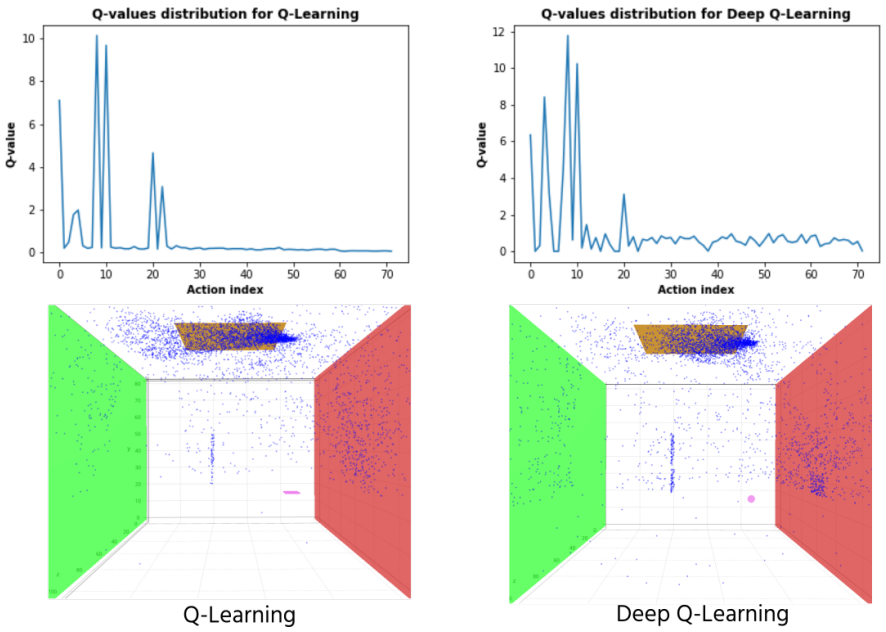

Figure 4.5: *Distribution of the Q-values for a specific state, both for Q-Learning (on the left) and Deep Q-Learning (on the right). The value function retrieved by the Deep Q-Learning algorithm overestimates the incident radiance for some actions. Overall, both distributions are strongly correlated, and present peaks in correspondence of the same actions.*

In both distributions, action 8 and action 10 are associated to the largest Q-values. The difference is their magnitude and these actions estimate a higher incident radiance when using Deep Q-Learning compared to Q-Learning. The consideration that the Deep Neural Network overestimates the Q-values is also exhibited in Figure 4.6. Here, the magnitude of the total incident radiance in each point of the Box scene is reflected by the intensity of the color gradient. It was calculated summing all the Q-values for every state encountered.

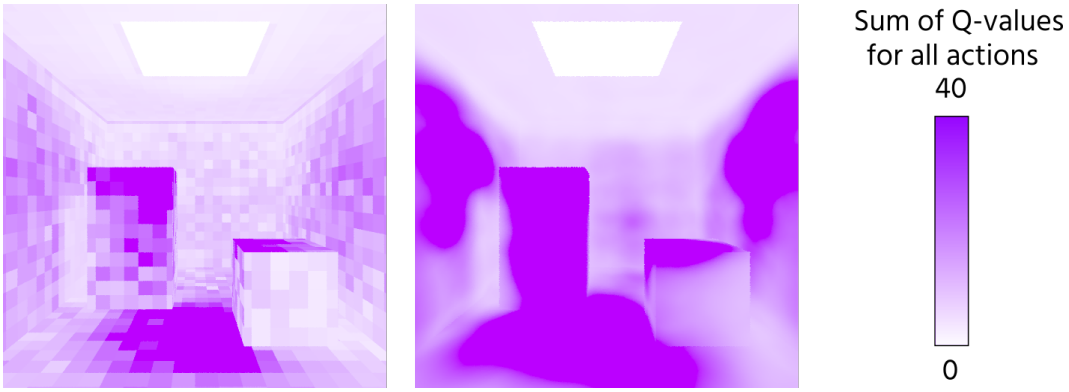

Figure 4.6: *Magnitude of the incident radiance inside the Box scene learned with Q-Learning (on the left) and Deep Q-Learning (on the right). For each point of the scene, the sum of the Q-values for all the actions, defining the total incident radiance, is calculated and its value is represented by the color intensity. The tiles visible in the Q-Learning approach are due to the discretization of the input space.*

Besides this visualization, a good measurement to discuss the comparison between the two Reinforcement Learning approaches is to estimate the quality of the generated images, as the next section explores.

4.3 Results: An image quality perspective

4.3.1 Evaluation metrics

The three importance sampling strategies compared are cosine-weighted importance sampling, importance sampling the value function approximated with Q-Learning and the value function retrieved with Deep Q-Learning. In order to compare the images generated by the these methods, four different metrics were evaluated. The metrics need to satisfy two important properties. The first requirement is to be independent of the software and the hardware utilized for the experiments. For this reason, the execution time was discarded. The second requirement is to capture the variance of the distribution of the pixels in the generated image.

To comply with these conditions, the first measure considered is the average number of bounces before hitting the light source. The rationale behind this measurement is that, when the path is not guided, rays need more bounces in order to hit the light. In case the ray

Metrics	Disadvantages	Advantages
Average amount of bounces	• Affected by Russian Roulette stopping strategy • No insight on the Q-value distribution	• Direct evaluation for path guiding towards the light source
MSE	• Poorly correlated with human perceptions • Not very reliable for low SPP • Images with same MSE have very different distortions	• Clear mathematical meaning
SSIM	• Not very reliable for low SPP • Significantly affected by nonlinear or multiple sources of distortions [44]	• Consistent with visual perception • Score is bounded
zero-contribution rays	• Affected by ϵ-decaying strategy • No insight on the Q-value distribution	• Clear evaluation of training performance

Table 4.1: *Advantages and disadvantages of the metrics used to assess the image quality and validity of the proposed importance sampling strategy*

does not reach the light after a certain amount of bounces, its contribution is set to zero. This operation is achieved through Russian Roulette [43], an adaptive cut-off path length technique. The generated rays need a stopping condition or path tracing would not come to a halt. Stopping the generation of rays at a fixed amount of bounces introduces biases, since potentially important paths would not be considered and the complexity of the scene would not be taken into account. In Russian Roulette, the radiance computed at each bounce is accumulated and stored. After a certain number of bounces, which in our case is set at 5, this accumulated factor is compared to a value randomly generated in the interval $[0, 1]$ for every recursion. If the radiance factor falls below this threshold, the rays generation is halted.

Even though the number of bounces gives a reliable insight into the scattered rays' behavior, no conclusions can be drawn regarding the approximation of the radiance function in the scene. This observation is evaluated via the quantification of the image noise. The metrics used in our comparison is the SSIM [45] and belongs to the referenced-based family of methods, described in Section 2.1.5. This evaluator is compared with the standard Mean Squared Error. Better image quality is expressed by a higher SSIM or a lower MSE.

The final metric is the number of paths with zero contribution. Especially, we are interested in the evolution of this value over time during the training process. If the training is

successful, the accumulated amount of paths with zero contribution should stop increasing at a constant rate.

In Table 4.1, the advantages and disadvantages of these metrics are outlined. For a detailed discussion on the limitations, please refer to Chapter 5.

4.3.2 Image Quality Assessment

In this section, we present the main results obtained during our work. Unless explicitly reported, every conclusion is based on the use of fine-tuned parameters and optimized techniques. Before showing these results, we express the reasons for some specific design choices.

Previously in this chapter, we mentioned the use of a target network to predict the target value function, instead of using a unique network in the training process. As we explained in Section 2.2.1 , the target network is a common strategy used in Deep Q-Learning to reduce the error variance in the target values and stabilize the training process. This confirms to be valid for our case and Figure 4.7 shows that this technique improves the SSIM score, while the MSE is not affected.

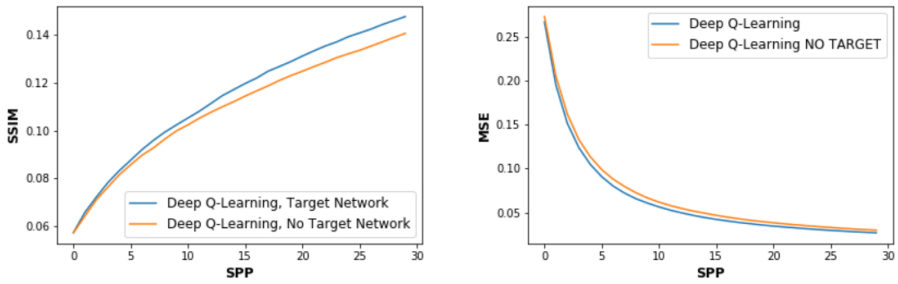

Figure 4.7: *SSIM and MSE of the approach based on Deep Q-Learning to render the Box scene. The metrics are plotted against the SPP. In one case, both the principal and the target network are used, while in the other case one single network is employed in the training process. The target network yields a better SSIM, but does not affect the MSE.*

A second important subject to address is the motive for using a large network, when most Deep Learning applications rely upon smaller architectures. As an example of this, Dahm and Keller predict the RGB color of a rendered image using multiple tiny networks composed of a single hidden layer with 9 neurons [21]. However, during the hyperparametrization, we observed that larger networks are associated with higher SSIM scores. In Figure 4.8, this dependency is depicted and three different architectures are used: 4 hidden layers with 200, 500 and 1000 neurons each. These plots substantiates that using 4 hidden layers of 1000 neurons each yields the best result. For the aforementioned reasons, two Deep Neural Networks with 4 hidden layers and 1000 neurons per layer are used during the training: the first is the main network and the second its copy with periodic weights update. After presenting these design

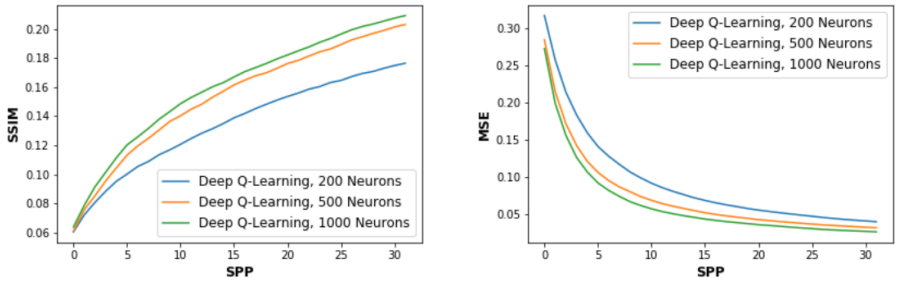

Figure 4.8: *SSIM and MSE for three different Deep neural Network structures, using 4 hidden layers of 200, 500 and 1000 neurons each for the Box scene. The metrics are plotted against the SPP. The images generated to assess the quality are renderings at 128x128 resolution of the scene Box. The best SSIM and MSE scores are those obtained with the largest network.*

choices, we focus on the results achieved with the tuned hyperparameters declared at the beginning of this chapter.

Figure 4.9 shows the SSIM for the three scenes. The reference image used for the comparison, displayed in the first column to the right, is generated with NEE and 5120 SPP. All the other images are generated using 32 SPP. In Box and Sunrise, the highest SSIM is obtained with Deep Q-Learning. In the Door scene, on the other hand, Q-Learning is the approach that leads to the highest image quality.

A comprehensive overview of the results obtained when assessing the quality of the generated images is summarized in Table 4.2. The MSE and the SSIM display a similar outcome for both Door and Box, where Deep Q-Learning compares favorably against the two other approaches. In contrast, it is interesting to notice that in Sunrise the SSIM score is significantly higher for Deep Q-Learning, while the MSE is lower for Q-Learning. In Chapter 5, this difference is analyzed.

	Box		Sunrise		Door	
	SSIM	MSE	SSIM	MSE	SSIM	MSE
Cosine	0.111	0.041	0.057	0.035	0.067	0.048
Q-Learning	0.135	0.028	0.077	**0.026**	**0.084**	**0.034**
Deep Q-Learning	**0.151**	**0.025**	**0.089**	0.027	0.067	0.048

Table 4.2: *SSIM and MSE of the different importance sampling approaches. A higher SSIM score and a lower MSE score represent better image quality. On one hand, considering SSIM as the principal metric, Deep Q-learning compares favorably against the other approaches for Box and Sunrise. In the last scene, Q-Learning yields the highest SSIM score. On the other hand, the MSE score is lower both for Sunrise and Door.*

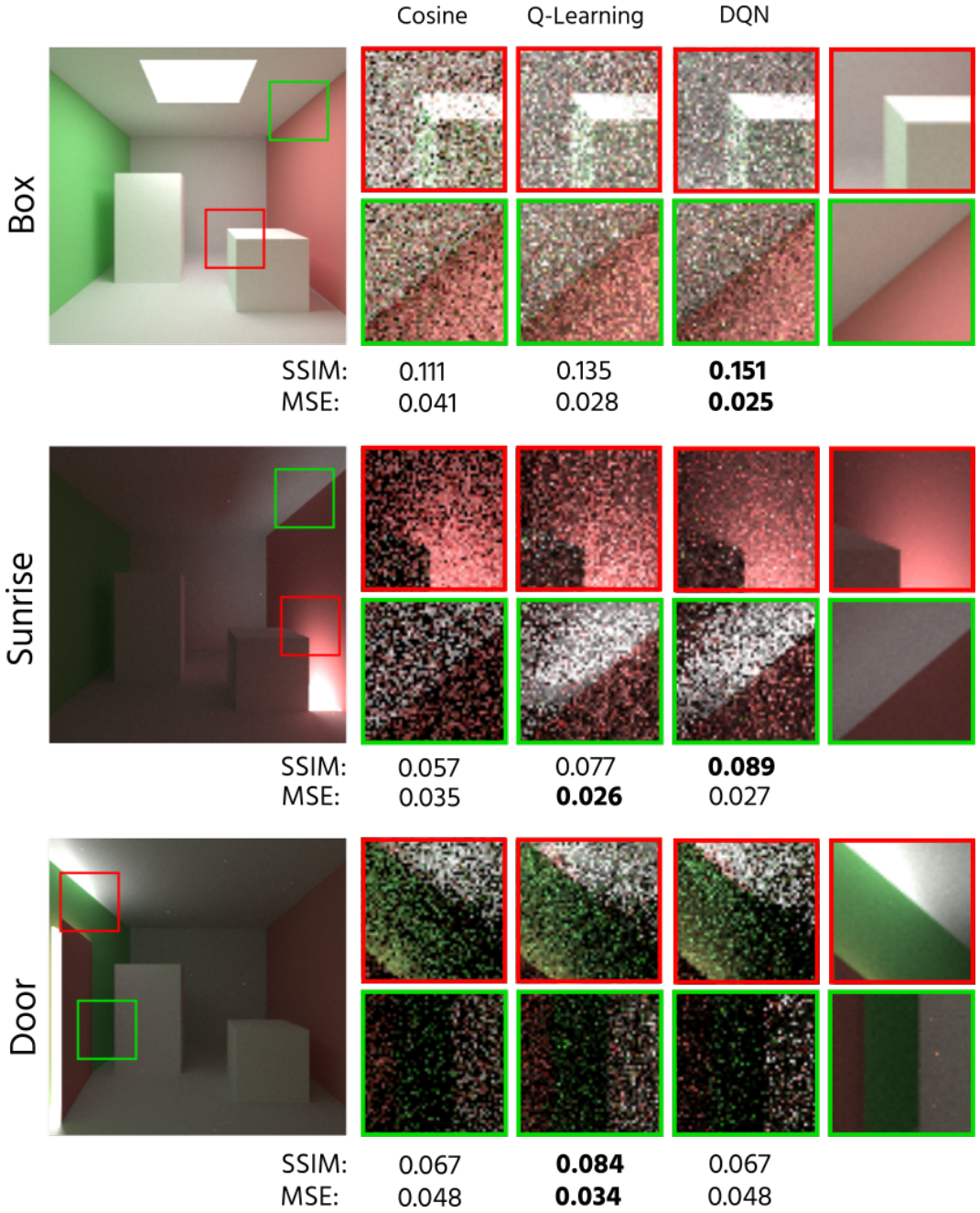

Figure 4.9: *Comparison among three approaches: cosine-weighted importance sampling, Q-Learning and Deep Q-Learning. The SSIM and MSE are reported for each scene and method. The images for the comparison are rendered using 32 SPP, while the reference image in the very right column is rendered with 5120 SPP and Next event estimation. All images are rendered at 256x256 resolution.*

Conforming to the results reported in Table 4.2, the SSIM and MSE are plotted against the SPP in Figures 4.10, 4.11. While for Sunrise and Box the use of both Reinforcement learning approaches produces better results, the application of Deep Q-Learning with the aforementioned architecture seems to yield no effect on the outcome in Door. Moreover, the SSIM value is substantially higher for the Box scene, endorsing the fact that simpler scenes are easier to model by a Markov Decision Process framework.

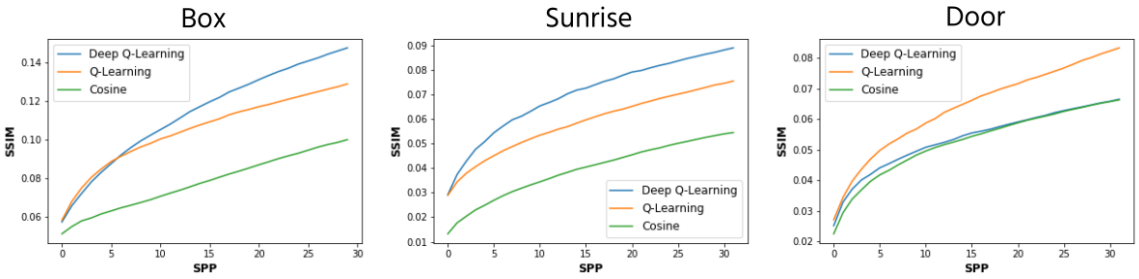

Figure 4.10: *The three plots show the SSIM of the three importance sampling strategies per scene against the SPP. In the first two scenes, Deep Q-Learning results in the best method, while in the last scene Q-Learning yields the best score.*

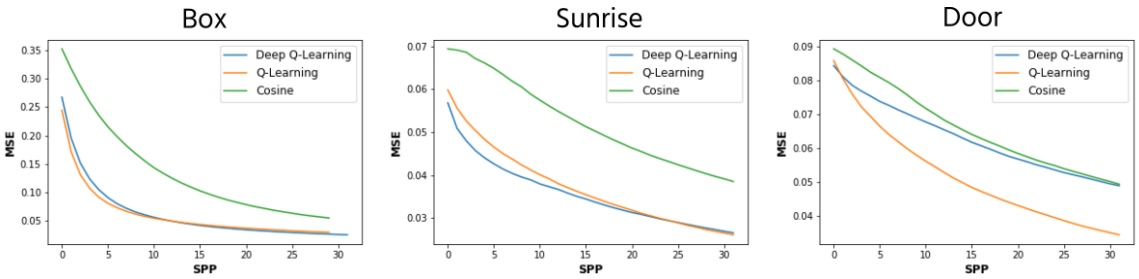

Figure 4.11: *The three plots show the MSE of the three importance sampling strategies per scene against the SPP. While in the first scene Q-Learning and Deep Q-learning almost lead to the same result, importance sampling based on discretized Q-values results in a lower MSE for the last two scenes.*

To analyze the impact of the Deep Neural Network architecture on the image quality of Door, we tested a more complex structure. Specifically, this new network is composed of 4 layers, each with 1500 neurons. All the other hyperparameters are maintained the same. Figure 4.12 shows that no gain is obtained using the new topology, suggesting that alternative changes might be needed for improvement.

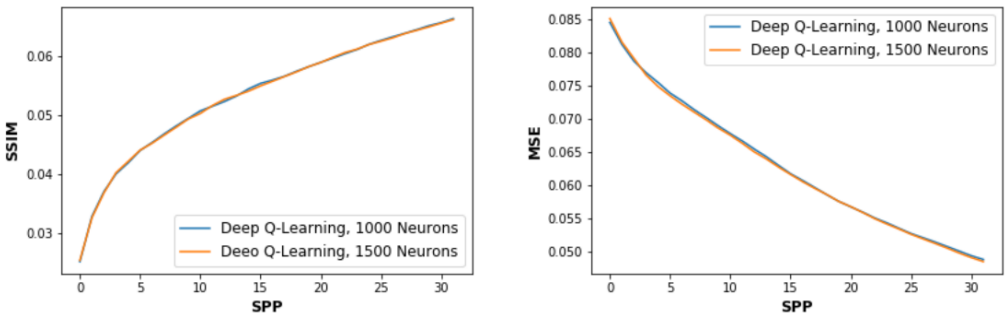

Figure 4.12: *SSIM and MSE against the SPP for two different Deep Neural Network architectures. The plots show that no improvement is obtained when the number of neurons is increased.*

When rays are scattered inside the scene, they bounce until they reach the light source, or until they are stopped by a Russian Roulette strategy, explained in Section 4.3.1. Intuitively, guiding the path towards the light source should reduce the average number of bounces. This behavior is confirmed by the results shown in Table 4.3. As one can see, Deep Q-Learning always leads to a considerable reduction in the average bounces, followed by Q-Learning. In this regard, it is essential to remark that no conclusive insight can be drawn by this result alone. Indeed, our goal is not only to choose those actions that increase the chance to hit the light source; the objective is to approximate the incident radiance accurately for all the possible actions. Even though the average number of bounces does not demonstrate that our aim is reached, it strongly suggests that Reinforcement Learning techniques can be correctly employed for path guiding.

	Box	Sunrise	Door
Cosine, mean±SD	7.66±0.005	10.29±0.007	10.21±0.005
Q-Learning, mean±SD	6.31±0.005	9.72±0.010	9.91±0.015
Deep Q-Learning, mean±SD	5.05±0.002	7.03±0.002	7.18±0.005

Table 4.3: *Average number of bounces inside each scene per approach. The results are reported in the format mean±standard deviation, obtained averaging 5 different computations for each result. Russian roulette allows to probabilistically stop rays scattering after 6 bounces.*

Finally, the number of paths with zero contribution against the episodes during the training phase is plotted in Figure 4.13. Comparing Q-Learning and Deep Q-learning to cosine-weighted importance sampling, it is possible to appreciate the significant reduction of non-valid paths when the former approaches are adopted . Even though the training is interrupted at 80 000 iterations, the trend lines suggest an increasing gap between the Reinforcement learning approaches and cosine-weighted importance sampling for longer training.

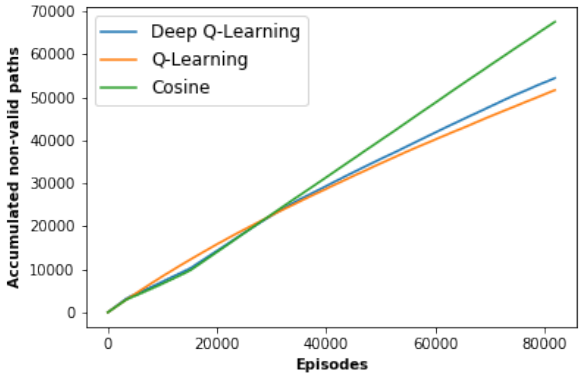

Figure 4.13: *Accumulated paths with zero contribution during training. In the beginning, rays scattering mainly follows a random distribution for both Q-learning and Deep Q-Learning. After about 35000 iterations, the number of non-valid paths starts to decrease with the episodes. Differently, cosine-weighted importance sampling is not a path guiding approach and the number of non-valid paths increases constantly.*

Chapter 5

Discussion

In this chapter, we first summarize the main steps observed in this work to answer the Research questions we initially formulated. Then, we describe and discuss the principal findings of the proposed approach and the challenges it posed.

5.1 Summary of our main steps

The primary goal of this thesis is to investigate how Reinforcement Learning techniques can be leveraged to learn the incident radiance function within a scene. The main assumption in this regard is that it is possible to frame the components of the naive path tracing algorithm as a Markov Decision Process. This hypothesis is based on the work conducted by Dahm and Keller on path guiding. To test this assumption, we first implemented a naive path tracer able to scatter secondary rays based on a random distribution. Then, we expanded this version to include a cosine-weighted importance sampling strategy and Next Event Estimation. While the former strategy is used in comparison with Reinforcement Learning driven approaches, the latter is employed to generate ground truth images. These images are then used to assess the quality of the renderings obtained adopting the various importance sampling strategy explained in detail in Chapter 3. Once the implementation of the path tracer was refined, and after comparing traditional variance reduction methods, we followed the approach proposed by Dahm et al. to guide the light path according to a tabular policy π generated with Q-Learning. Due to the lack of implementation details in the paper, we presented our solutions to technical and theoretical issues that arose during the work, such as the discretization of the action space or the formulation of the resulting PDF. Even though it is not possible to infer how the authors tackled these problems, a careful study of their work and the referred literature suggests that a very similar procedure was adopted. Next, we proposed a novel method to use a Deep Neural Network as a function approximator for learning the incident radiance function within a scene. The implementation of this method was very challenging

and required to develop a new software framework. Finally, we framed four evaluation metrics and compared the different importance sampling strategies.

5.2 Key insights

Among the numerous algorithms that belong to the Reinforcement Learning paradigm, the focus of our work is on Q-Learning and Deep Q-Learning. Although the use of these two particular algorithms introduces many challenges, we consider them to be suited for solving our problem since they are both off-policy approaches: this means that they can learn an optimal policy while following a different one. This factor is crucial, since it guarantees the validity of the learning process even when following an exploratory policy instead of an exploiting one. That is, we can approximate the incident radiance for all the actions and not only for the ones that lead to the highest reward. This behavior also enables the simultaneous use of two different networks, the target network and the principal one, that is not possible for on-policy algorithms.

The rationale behind the extension of Q-Learning and the use of Deep Neural Networks is to overcome the constraints imposed by the discretization of the state space. The policy produced by Deep Q-Learning relies on the generalization over the state space, since the algorithm needs to approximate the target function over a continuous set of points in the scene. This represents an improvement if the algorithm maps correctly continuous unseen input values into the distribution of the incident radiance. Differently, the policy obtained with the Q-Learning algorithm consists in a matrix where all the state-action pairs have been explored.

In light of the results reported in Section 4.3, we looked into whether a function approximation over a continuous space results in better image quality compared to a tabular discretized approach. To analyze this question, two different concepts need to be considered: the efficiency of path guiding towards the light source, and the reliability of the value function's approximation. Table 4.3 reveals that the Deep Q-Learning algorithm reduces the average number of bounces for all the scenes. This means that the ray is scattered faster and more efficiently towards the light source. To determine whether the approximation of the value function reflects reality, the quality of the images generated needs to be assessed. The results reported in Section 4.3 show two different metrics: the SSIM and the MSE. Evaluating which of these scores better represent the image quality is not an easy task, since it is highly subjective. The plots in Figures 4.10, 4.11 show that the MSE of the images obtained with Q-Learning and Deep Q-Learning is almost the same. As one can see looking closely at the two pictures in Figure 5.1, Deep Q-Learning seems to generate more uniform tones and shades, that can be appreciated looking at the zoomed detail on red wall.

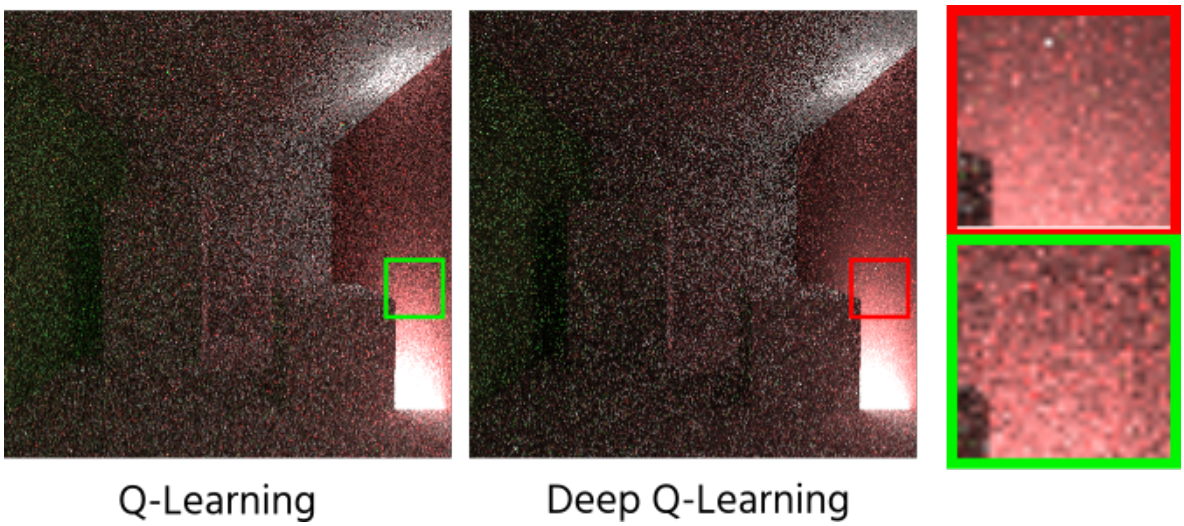

Figure 5.1: *Images obtained with Q-Learning (on the left) and Deep Q-learning (on the right). The details on the very right column show that Deep Q-Learning results in more uniform tones and shades.*

As stated, this result is subjective, but due to the differences between the images that the MSE is not able to capture, we consider SSIM more reliable in perceiving the overall image quality. However, given the low SPP rate, caution must be exercised when drawing conclusions.

The SSIM scores visible in Table 4.2 show that Deep Q-Learning improves Monte Carlo convergence in some cases, namely in the first two scenes. Despite the lack of data to ascertain which setting better fits the use of Deep Neural Networks, it is reasonable to think that the scene complexity represents a key factor. Since, as mentioned previously, Deep Q-Learning improves path guiding, we believe that its weakness can be attributed to the difficulty in modeling the correct distribution of the incident radiance function. Indeed, if the algorithm importance samples over an improper distribution, it computes a wrong PDF: this results in the incorrect estimation of the pixels' final radiance, since the PDF is an essential component of Monte Carlo integration (Section 2.1.3). According to the lower average number of bounces required to reach the light source and the higher estimated Q-values (Figure 4.5), we can then conclude that the employed architecture causes an overestimation of the incident radiance.

The amount of shorter paths resulting from the use of Deep Q-Learning seems to contradict the fact that this method also leads to a higher number of accumulated non-valid path, as revealed in Figure 4.13. We argue that this might be explained by network overfitting. Indeed, overfitting causes a lack of generalization which results in two possible extreme cases. On the one hand, the network maps the most visited states into accurate distributions, leading to very short paths. On the other hand, multiple invalid paths can occur when proper inference is not correctly achieved for those input values that are far from the commonly seen ones. This unbalance could be generated by the configuration of the scene, which favors the visit of those

states with higher visibility from the camera. Moreover, the function approximator for all the scenes is a Deep Neural Network with the same structure. Hence, we cannot exclude that re-iterating the hyperparametrization process on the network used for the last scene would improve the SSIM score and positively reduce overfitting. The use of the same agent for all the scenes is not ideal, since the hyperparametrization should be scene-dependent. However, the amount of possible parameters combinations is extremely large, and hypertuning for the first scene took weeks due to the current software framework. Under the current limitations, it has not been possible to adapt the agent to every scene.

The difficulties in training the algorithm for the third scene can be also explained by two characteristics of our setting: the sparse episodic reward signal and the highly-stochastic environment. The sparse episodic reward signal is caused by the way our Reinforcement Learning environment is framed. Based on Equation 2.26, derived from the combination between the Bellman equation and the rendering equation, the reward given to the agent is equal to the emitted radiance of the hit surface. This entails that the reward function is episodic, since it is provided to the agent only when the whole episode is concluded. An episode is considered the sequence of states and actions starting when the primary ray is scattered and ending when the ray hits the light source or it is stopped by the Russian Roulette strategy explained in Section 4.3.1. Consequently, as the ray can be stopped before hitting an emitting surface, the reward can be null even at the terminal state. This introduces additional sparsity in the reward signal. We calculated that the light source is only reached about 1.5% of the times a ray is scattered inside the scene. These factors are not ideal for the learning process and account considerably for the lack of convergence of the value function. As a possible solution, replay memory (Section 3.3.1) was implemented. Due to the lack of correlation among successive tuples $< s, a, r, s' >$, it did not significantly help.

In Section 2.2.2, we showed how the discretization of the state space in physical tiles in the scene could negatively affect the performance of the path tracer based on Q-Learning. Namely, when rays belonging to the same state are scattered from different coordinates, their direction is sampled from the same distribution. This factor causes different outcomes depending on the coordinates from where the rays are bounced (Figure 2.9). This is not the only element that introduces high stochasticity to the approach. Similarly to the state space, also the action space is discretized in virtual patches over the unit hemisphere. When an action is chosen by the importance sampling strategy, a point is randomly selected inside the patch mapped to that action, and the new ray is scattered through it. Thus, multiple rays scattered from the same coordinates and based on the same actions can follow different directions, as one can see in Figure 5.2.

When the metrics to assess the proposed approaches were investigated, one of the fundamental principles was to consider the evaluators as independent from the algorithms' running time. Indeed, the engineering performance of the methods presented is out of scope and for

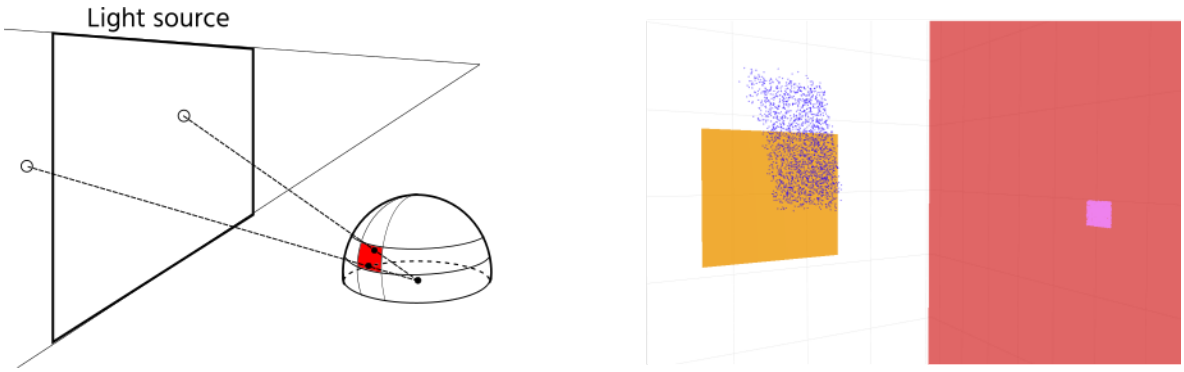

Figure 5.2: *On the left, a scheme showing how scattering rays through points evenly distributed inside a patch leads to different outcomes. On the right, the real scattering distribution of multiple rays from the same state, represented by the purple tile, according to the same action. Half the rays do not hit the light source, thus increasing stochasticity in the learning process.*

this reason was not examined. This being said, we can consider the use of Deep Q-Learning in the production pipeline unfeasible at this stage. To obtain a better rendering quality than the one achieved with Q-Learning, the network needs to be very large. Hence, the computational cost is highly increased and the gain in quality, for complex scenes, is very limited. The adoption of two Deep Neural Networks, necessary to adopt the target network technique, does not slow down noticeably the performance of the algorithm, since only the principal network is used to backpropagate the error and compute the gradients to minimize the loss. The bottleneck of the proposed approach is the gradient calculation, which in the case of our network is composed of $3.12e+6$ parameters. Moreover, the difficulties that made impractical to introduce Deep Neural Networks in the C++ pipeline suggest that research and development need to be undertaken before this approach can be considered useful for production purposes.

Regardless of these considerations, the good results obtained in the first two scenes are very promising and can be further developed, as discussed in the next chapter.

Chapter 6

Conclusions

6.1 Conclusions

This research was mainly undertaken to design a Deep Q-Learning approach and evaluate its efficiency for importance sampling the approximation of the incident reward function in every point of a virtual scene. Our findings suggest that the proposed method is capable of mapping accurately a continuous set of input variables into a distribution over a discrete action space. Indeed, in two out of the three settings we tested, our method outperformed the previous work on Q-Learning for path guiding. To obtain this result, the approach proved to model correctly a highly-stochastic environment with a sparse episodic reward signal. The relevance of this element is supported by the MSE and SSIM evaluators used to assess the quality of the generated images, that are better than or equal to the measurements for the discrete version. In addition, further analysis showed that the average amount of bounces is reduced for every scene when Deep Q-Learning is used. This also validates the usefulness of the proposed method. Taken together, these findings could conceivably lead to the use of Deep Reinforcement Learning as an importance sampling technique for path guiding.

Furthermore, the previous work on Q-Learning has been slightly altered to include a short training phase employed to create a policy, followed by an active phase that generates the image based on the knowledge acquired. The derivation of important aspects of this method, that were not explicitly reported in the previous work on Q-Learning, are also detailed in our research. This is especially important for the formulation of the PDF, used for both Q-Learning and Deep Q-Learning, since it guarantees the reproducibility of our results.

6.2 Limitation and Future work

Finally, a number of caveats regarding the present study need to be noted. It is recommended that further research be undertaken in the following areas: hyperparametrization, Policy

Gradient methods, and sparseness.

In one case, namely in the third scene presented, our Deep Q-Learning approach does not prove to contribute meaningfully to the importance sampling strategy. There is, therefore, a definite need for additional hyperparametrization to estimate whether this is caused by a technical limitation or by the methodology adopted. In this regard, further research might explore the network's topology and the influence of the learning rate on the final results. Then, different experience replay strategies can be investigated to improve the convergence of the value function. Specifically, Prioritized Experience Replay (PER) proved to perform successfully in multiple tasks [34]. Moreover, research is also needed to determine the optimal input size and kernel characterizing the One-Blob encoding. A larger input might be beneficial for complex scenes, since an increased dimensionality can help the inference of non-linear mapping. Automatic hypertuning, which is currently performed with Genetic Algorithms, might ease the procedure.

Besides the hyperparametrization, additional adjustments can be considered for further improvements. Deep Q-Learning is not usually the preferred option for highly-stochastic environments, and changing paradigma to adopt Policy Gradients methods might advance numerous benefits [39]. We believe that the need to discretize the action space might be among the major limitations of our work, and Policy Gradients approaches are specifically suitable for settings with a continuous action space [5]. Among all the Policy Gradients algorithms, REINFORCE [39], CACLA [42] and DDPG [35] are the most investigated techniques in this field.

In Chapter 5, we discuss the feasibility of implementing the proposed approach beyond the scope of research investigation. As mentioned, the large number of network parameters to update at every iteration increases significantly the computation necessary for path guiding. An intuitive solution might be to reduce the number of network's weights, while trying to keep the performance unaltered. The Deep Neural Networks developed in our method are characterized by fully connected multilayer perceptrons [30], where nodes have full connections to all activations in the previous layer. Sometimes, this can be unnecessary since different parts of the network are adapted to diverse subtasks during the training. A concept of efficiency in this regard is sparseness. Sparse connectivity, opposed to full connectivity, means that each neuron is connected to a limited amount of other neurons [40]. Mocanu et al. introduced Sparse Evolutionary Training (SET) [28], a procedure closely related to NeuroEvolution [37] that yields sparse adaptive connectivity. The authors presented the promising results obtained with this technique, which could be further tested for Reinforcement Learning methodologies and might lead to an optimized structure to be used for path guiding.

Bibliography

[1] Global animation, vfx and games industry: Strategies, trends and opportunities, 2019. https://www.researchandmarkets.com/research/hh3jc7/global_animation?w=5. Accessed: 10-02-2019. 2

[2] Rendering. https://sciencebehindpixar.org/pipeline/rendering. Accessed: 10-02-2019. 2

[3] Leemon C Baird. Reinforcement learning through gradient descent. 1999. 23

[4] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*, volume 5. Athena Scientific Belmont, MA, 1996. 30

[5] Po-Wei Chou, Daniel Maturana, and Sebastian Scherer. Improving stochastic policy gradients in continuous control with deep reinforcement learning using the beta distribution. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 834–843. JMLR. org, 2017. 56

[6] Ken Dahm and Alexander Keller. Learning light transport the reinforced way. *arXiv preprint arXiv:1701.07403*, 2017. 3, 4, 6, 23, 29, 31

[7] Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014. 6, 25

[8] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced global illumination*. AK Peters/CRC Press, 2006. 13, 14

[9] Petros Giannakopoulos and Yannis Cotronis. A deep q-learning agent for the l-game with variable batch training. *arXiv preprint arXiv:1802.06225*, 2018. 33

[10] Cindy M Goral, Kenneth E Torrance, Donald P Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *ACM SIGGRAPH computer graphics*, volume 18, pages 213–222. ACM, 1984. 10

[11] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947, 2000. 35

[12] Verena Heidrich-Meisner, Martin Lauer, Christian Igel, and Martin A Riedmiller. Reinforcement learning in a nutshell. In *ESANN*, pages 277–288, 2007. 21

[13] Plotly Technologies Inc. Collaborative data science, 2015. 8

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 34

[15] Tommi Jaakkola, Michael I Jordan, and Satinder P Singh. Convergence of stochastic iterative dynamic programming algorithms. In *Advances in neural information processing systems*, pages 703–710, 1994. 30

[16] Wenzel Jakob. Mitsuba renderer, 2010. http://www.mitsuba-renderer.org. 7

[17] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 — seamless operability between c++11 and python, 2016. https://github.com/pybind/pybind11. 7

[18] Nicolai M Josuttis. *The C++ standard library: a tutorial and reference*. Addison-Wesley, 2012. 5

[19] James T Kajiya. The rendering equation. In *ACM SIGGRAPH computer graphics*, volume 20, pages 143–150. ACM, 1986. 10

[20] Alexander Keller and Nikolaus Binder. Deterministic consistent density estimation for light transport simulation. In *Monte Carlo and Quasi-Monte Carlo Methods 2012*, pages 467–480. Springer, 2013. 32

[21] Alexander Keller and Ken Dahm. Integral equations and machine learning. *Mathematics and Computers in Simulation*, 161:2–12, 2019. 24, 44

[22] Seungchan Kim, Kavosh Asadi, Michael Littman, and George Konidaris. Deepmellow: Removing the need for a target network in deep q-learning. 23

[23] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 35, 41

[24] Huijuan Li. Lagrange multipliers and their applications. *Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, TN*, 37921, 2008. 14

[25] Morgan McGuire. *The Graphics Codex*. Casual Effects, 2.14 edition, 2018. 3

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 22

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015. 23

[28] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):2383, 2018. 56

[29] Thomas Müller, Brian McWilliams, Fabrice Rousselle, Markus Gross, and Jan Novák. Neural importance sampling. *arXiv preprint arXiv:1808.03856*, 2018. 3, 6, 25

[30] Fionn Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2(5-6):183–197, 1991. 56

[31] Jacopo Pantaleoni and Eric Heitz. Notes on optimal approximations for importance sampling. *arXiv preprint arXiv:1707.08358*, 2017. 14

[32] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2016. 7, 10

[33] Georgios Sakas, Peter Shirley, and Stefan Müller. *Photorealistic rendering techniques*. Springer Science & Business Media, 2012. 3

[34] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015. 56

[35] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. 2014. 56

[36] Philipp Slusallek. Photo-realistic rendering–recent trends and developments. 2

[37] Kenneth O Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002. 56

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 21

[39] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000. 56

[40] Markus Thom and Günther Palm. Sparse activity and sparse connectivity in supervised learning. *Journal of Machine Learning Research*, 14(Apr):1091–1143, 2013. 56

[41] Hado Van Hasselt. Reinforcement learning in continuous state and action spaces. In *Reinforcement learning*, pages 207–251. Springer, 2012.

[42] Hado Van Hasselt and Marco A Wiering. Reinforcement learning in continuous action spaces. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 272–279. IEEE, 2007. 56

[43] Eric Veach. *Robust Monte Carlo methods for light transport simulation*, volume 1610. Stanford University PhD thesis, 1997. 10, 43

[44] Haiyan Wang, Diego Maldonado, and Sharad Silwal. A nonparametric-test-based structural similarity measure for digital images. *Computational Statistics & Data Analysis*, 55(11):2925–2936, 2011. 43

[45] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, et al. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 20, 43

[46] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. 21

[47] Turner Whitted. An improved illumination model for shaded display. In *ACM Siggraph 2005 Courses*, page 4. ACM, 2005. 10

[48] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006. 25

[49] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. *arXiv preprint arXiv:1906.05113*, 2019. 1

[50] Quan Zheng and Matthias Zwicker. Learning to importance sample in primary sample space. In *Computer Graphics Forum*, volume 38, pages 169–179. Wiley Online Library, 2019. 25

Appendix A

Derivation of the PDF for different importance sampling strategies

A.1 PDF for uniform sampling

To derive the PDF for uniform sampling on a hemisphere, the sample space can be conveniently expressed in terms of solid angle w. Since the solid angle subtended by an hemisphere is 2π, the integration of the PDF over the entire sample space results:

$$\int_0^{2\pi} p(w)dw = 1 \tag{A.1}$$

As the PDF is constant, $p(w) = C$:

$$1 = C \int_0^{2\pi} dw = C \cdot 2\pi \tag{A.2}$$

Thus:

$$C = p(w) = \frac{1}{2\pi} \tag{A.3}$$

A.2 PDF for cosine-weighted importance sampling

In cosine-weighted importance sampling, the PDF $p(w)$ is proportional to $cos\theta$, where θ is the angle between the direction of the incident light and the surface normal.

$$p(w) = f(\theta) = C \cdot cos\theta \tag{A.4}$$

The differential solid angle, as seen in Section 3.1, can be expressed as:

$$dw = sin\theta \ d\theta \ d\phi \tag{A.5}$$

The two properties of the PDF are the following:

$$p(x) > 0, \forall x \in D$$

$$\int_D p(x)\,\mathrm{d}x = 1 \tag{A.6}$$

Expressing the domain D in spherical coordinates, and combining Equations A.4, A.5, and A.2:

$$1 = \int_0^{2\pi} \int_0^{\frac{\pi}{2}} C \cdot cos\theta \ sin\theta \ d\theta d\phi = 2\pi C \int_0^{\frac{\pi}{2}} cos\theta \ sin\theta \ d\theta = 2\pi C \ [-\frac{1}{2} cos\theta]_0^{\frac{\pi}{2}} = \pi C \tag{A.7}$$

Thus:

$$C = \frac{1}{\pi} \tag{A.8}$$

The resulting PDF is obtained combining Equations A.4 and A.8:

$$p(w) = \frac{cos\theta}{\pi} \tag{A.9}$$