Quality Assessment and Variance Reduction in Monte Carlo Rendering Algorithms

Whittle, Joss

Quality Assessment and Variance Reduction in Monte Carlo Rendering Algorithms

Joss Whittle

May 1st, 2018

Swansea University

Prifysgol Abertawe
Swansea University

Department of Computer Science
Computer Graphics Group

PhD Thesis

Quality Assessment and Variance Reduction in Monte Carlo Rendering Algorithms

Joss Whittle

1. Reviewer Kurt Debatista
Centre for Scientific Computing
University of Warwick

2. Reviewer Xianghua Xie
Department of Computer Science
University of Swansea

Supervisors Mark W. Jones and Benjamin Mora

May 1st, 2018

Joss Whittle

Quality Assessment and Variance Reduction in Monte Carlo Rendering Algorithms

PhD Thesis, May 1st, 2018

Reviewers: Kurt Debatista and Xianghua Xie

Supervisors: Mark W. Jones and Benjamin Mora

Swansea University

Computer Graphics Group

Department of Computer Science

Faraday Tower, Swansea University

Swansea, SA2 8PP

United Kingdom

Declaration

I declare that the work presented in this thesis has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Swansea, May 1st, 2018

Joss Whittle

That this thesis is the result of my own investigations, except where otherwise stated and that other sources are acknowledged by footnotes giving explicit references and that a bibliography is appended.

Swansea, May 1st, 2018

Joss Whittle

That I give consent for the thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Swansea, May 1st, 2018

Joss Whittle

Abstract

Over the past few decades much work has been focused on the area of physically based rendering which attempts to produce images that are indistinguishable from natural images such as photographs. Physically based rendering algorithms simulate the complex interactions of light with physically based material, light source, and camera models by structuring it as complex high dimensional integrals [Kaj86] which do not have a closed form solution. Stochastic processes such as Monte Carlo methods can be structured to approximate the expectation of these integrals, producing algorithms which converge to the true rendering solution as the amount of computation is increased in the limit.

When a finite amount of computation is used to approximate the rendering solution, images will contain undesirable distortions in the form of noise from under-sampling in image regions with complex light interactions. An important aspect of developing algorithms in this domain is to have a means of accurately comparing and contrasting the relative performance gains between different approaches.

Image Quality Assessment (IQA) measures provide a way of condensing the high-dimensionality of image data to a single scalar value which can be used as a representative measure of image quality and fidelity. These measures are largely developed in the context of image datasets containing natural images (photographs) coupled with their synthetically distorted versions, and quality assessment scores given by human observers under controlled viewing conditions. Inference using these measures therefore relies on whether the synthetic distortions used to develop the IQA measures are representative of the natural distortions that will be seen in images from domain being assessed.

When we consider images generated through stochastic rendering processes, the structure of visible distortions that are present in un-converged images is highly complex and spatially varying based on lighting and scene composition. In this domain the simple synthetic distortions used commonly to train and evaluate IQA measures are not representative of the complex natural distortions from the rendering process. This raises a question of how robust IQA measures are when applied to physically based rendered images.

In this thesis we summarize the classical and recent works in the area of physically based rendering using stochastic approaches such as Monte Carlo methods. We develop a modern C++ framework wrapping MPI for managing and running code on large scale distributed computing environments. With this framework we use high performance computing to generate a dataset of Monte Carlo images. From this we provide a study on the effectiveness of modern and classical IQA measures and their robustness when evaluating images generated through stochastic rendering processes. Finally, we build on the strengths of these IQA measures and apply modern deep-learning methods to the No Reference IQA problem, where we wish to assess the quality of a rendered image without knowing its true value.

Acknowledgement

To everyone who has helped me throughout my time at Swansea University, thank you. This experience would not have been what it has without you all.

I would like to offer my sincerest gratitude and thanks for the support of my advisor, Professor Mark Jones, who during my undergraduate studies at Swansea encouraged me to pursue my interest in computer graphics and mentored me along the path to becoming a researcher.

To my friends and colleagues at Swansea, I offer my sincerest thanks. The wealth of knowledge and experience you hold have been a tremendously valuable resource throughout my research. It has truly been a privilege to work with all of you.

This research has been supported by EPSRC through Doctoral Training Award EP/K502935/1 whom I thank for the amazing opportunity they have afforded me.

Finally, I would like to offer my deepest thanks to my parents, John and Catheryne Whittle, and my sisters, Claire Whittle and Sarah Warrick, whose unwavering support and encouragement I could not have done this without. I would also like to give my thanks to my brother-in-law Dr. Edmund Warrick, who originally inspired me to pursue a doctorate when I was still in high school, and whose detailed feedback and advice have benefited this thesis immeasurably.

Contents

1 Introduction

Any one who considers arithmetical methods of producing random digits is, of course, in a state of sin. For, as has been pointed out several times, there is no such thing as a random number — there are only methods to produce random numbers, and a strict arithmetic procedure of course is not such a method.

— John von Neumann

Physically based rendering methods allow for the generation of high quality synthetic images which can often be indistinguishable from natural images such as photographs. Due to the complex nature of light interactions with different mediums and the high volume with which light energy flows through natural environments, the simulation of such light transport problems can quickly become intractable to compute through classical numerical methods.

Through the application of stochastic methods such as Monte Carlo sampling, algorithms for the simulation of light transportation problems can be derived which faithfully capture the complexities and subtleties of light phenomena observed in nature. These algorithms are structured such that they converge to a true solution to the posed simulation problem as increasingly more computation is performed, only achieving full convergence in the limit as the number of random samples goes to infinity. Early halting of the simulation after a finite amount of computation has the effect of creating a coarse approximation of the true solution.

A significant benefit to this approach is it allows for the accuracy of the final simulation to be balanced against the sensitivity of the observer to errors present in the coarse approximation and the practicality of continuing the computation to further reduce simulation error. When machine precision and hardware limitations of output devices (such as computer monitors with finite colour depths) are considered, this allows for previously intractable problems to be faithfully simulated with a finite amount of computation without introducing any perceivable error.

Over the previous four decades our increased access to large scale computational resources such as increased memory capacity, processor speeds, thread counts, and the use of acceleration devices such as GPUs and co-processors have allowed us to tackle problems of increasing complexity and scale. Much work has been done in the area of developing reusable and extensible systems for managing hardware and the mapping of software onto hardware in large scale, often distributed, computing environments. However, even as the power and scale of the available hardware increases, our desire to keep pushing the limits of the hardware currently at our disposal requires the development of new and more efficient algorithms which are able to better use the information which is readily available within the simulation.

The use of a finite amount of computation in the stochastic simulation of complex light transportation problems has the effect of generating a coarse approximation of the true solution. The coarseness of the approximate solution presents itself as error compared to the true solution. In order to compare and contrast the relative performances of competing rendering algorithms robust measures for comparing the coarse outputs of different algorithms to the true solutions are needed. These measures, termed Image Quality Assessment (IQA) measures, have in the past been based on simple numerical divergences. However, such methods do not reflect the sensitivities of the human visual system to different types of distortions. In recent years much work has been focused on the task of developing efficient algorithms which accurately reflect the sensitivities of the human visual system when comparing images.

1.1 Research Methodology

This thesis investigates the use of IQA measures when applied to images generated with Monte Carlo rendering algorithms, exploring the robustness of existing IQA methods, and developing novel ones which are able to faithfully predict image quality when presented with only a distorted test image. When we review images reported in the physically based rendering literature we observe that in many cases visible noise is present in images labelled as Ground Truth (GT) references, which should by definition be completely distortion free. We seek to answer the question of *by how much do these types of distortions in reference images effect the accuracy of results reported by IQA measures?* We also consider whether many existing IQA measures report quality assessments that are representative of the perceived quality of distortions observed in rendered images. These measures have largely been developed with the aim of assessing the quality of specific distortion types. Most commonly these are synthetic distortions that are applied to datasets of natural images such as photographs. We ask the question of *whether these synthetic* *distortions are representative of the types of complex and spatially varying distortions that we observe naturally as a result of under-sampling within Monte Carlo rendering processes?*

To this end, we summarize the classical and recent works in the areas of image quality assessment, machine learning, and physically based rendering using stochastic approaches such as Monte Carlo methods. We present a modern C++ framework on top of Message Passing Interface (MPI) for managing and running code on large scale distributed computing environments, with particular care made to improve compile-time error handling and type-checking over previous works. Leveraging this framework, we implement Monte Carlo rendering algorithms in High Performance Computing (HPC) environments, and use this to generate a large dataset of rendered images containing varying amounts of natural distortion. Using the dataset we provide an ensemble study on the effectiveness of modern and classical IQA measures used in comparing digital images, and their robustness when evaluating images generated through stochastic rendering processes. Finally, we build on the strengths of these IQA measures and apply modern deep-learning methods to the NR-IQA problem, where we wish to assess the quality of a rendered image without knowing its true value.

1.2 Thesis Structure

Chapter 1

The remainder of chapter 1 summarizes the structure of this thesis and the contributions that are presented.

Chapter 2

In chapter 2 we review the literature on Bayesian Statistics, Monte Carlo Simulations, Physically Based Rendering, Image Quality Assessment, and Machine Learning. This chapter is intended to give an overview of the topics upon which this thesis builds. At beginning of chapters 3, 4, and 5 there are additional literature reviews pertaining specifically to each chapter.

Chapter 3

Monte Carlo rendering algorithms are slow to converge and require a large amount of computation to produce visually acceptable results. In order to generate large datasets of images to analyze we made use of HPC methods to implement rendering algorithms on large distributed computing environments. The implementation of these parallel rendering algorithms led to the development of a modern C++ framework which addressed several issues we encountered while working in HPC environments.

Chapter 3 describes the development of MPI Extension Library (MEL), a modern C++11 framework around the MPI 3.0 standard. MEL is a header-only library with the goal of creating a lightweight and robust framework for building distributed parallel applications for use in HPC environments. MEL is designed to introduce no (or minimal) overheads while *drastically* reducing code complexity and allowing for a greater range of common MPI errors to be caught at compile-time rather than during program execution when it can be far more difficult to debug.

One of the main goals of MEL is to give higher-level functionality that is not natively available within the MPI standard. To demonstrate this we tackle the issue of performing deep copy on complex hierarchical and potentially cyclical data structures between disjoint hosts in a distributed computing environment. We approach this problem by creating a set of generic deep copy semantics that can easily be applied to user data structures, and a traversal and transportation algorithm which can then walk the data structure performing the desired transport operations. Our semantic mark-up is generic both in that it can be applied simply to any user defined structure, but also in its abstraction of the transport operation that will later be performed on it. Once a data structure has been prepared for our deep copy algorithm it can be passed to all transport functions within the API, allowing for MPI send, receive, broadcast, and file IO operations to all be performed. Additionally we provide a variant of each of the above transport operations which internally buffer data into or from a contiguous block of memory during transmission, yielding considerable performance increases on appropriate data structures.

Chapter 4

In chapter 4 we provide and ensemble study on the robustness of IQA measures when evaluating images created through Monte Carlo rendering processes. When assessing image quality in Monte Carlo rendered images the use of a reference image or GT is a common method to provide a baseline with which to compare experimental results as we often need to determine the relative quality between images computed using different algorithms and with varying amounts of computation.

We show that if not chosen carefully the quality of reference images used for IQA can skew results leading to significant misreporting of error. We present an analysis of error in Monte Carlo rendered images and discuss practices to avoid or be aware of when designing an experiment.

The issue stems from the fact that ground truth images are never truly available, as they are the product of a stochastic rendering process just the like the test images we wish to evaluate. Monte Carlo rendering processes are known to converge in the limit, as the number of samples goes to infinity. For any finite number of samples used to approximate an image there will always be some distortion or bias introduced into the image estimate. By rendering reference images to a significantly higher visual quality than the test images they will be used to evaluate we can limit the bias introduced by distortions in reference images. However, this raises the question of by how much distortions in reference images affect the results of IQA measures when evaluating test images.

To answer this question we used our HPC implementations of several Monte Carlo rendering algorithms to render a large dataset of images containing a range of scene, material, and lighting compositions to increasing numbers of image samples. We constructed an experiment where we used this dataset of images at varying qualities to calculate the reported quality score of test images compared to each of the potentially noisy reference images drawn from the dataset, using a collection of IQA measures we gathered from the literature. We used the highest sampled images as approximate ground truth images, and use these "true" quality values to compute the amount by which each error metric under- or overestimates its quality score for each comparison, as a function of the quality of the potentially noisy reference image that was used.

By this method we can compute and visualize the robustness of each IQA measure we sample from the literature when we consider the scenario where the reference image used in image quality assessment is not a perfect representation of the ground truth image and contains some magnitude of distortion.

Chapter 5

Finally, in chapter 5 we investigate the use of deep-learning models applied to the task of NR-IQA in the context of evaluating images rendered with Monte Carlo rendering processes. As these models require a large corpus of data in order to learn robust and generalized strategies for inference we again make use of the image dataset we generated using distributed computing.

In Full Reference Image Quality Assessment (FR-IQA), images are compared with ground truth images that are known to be of high visual quality. These metrics are utilized in order to rank algorithms under test on their image quality performance, usually using an equal time or equal sample comparison. However, during an intermediate stage of a Monte Carlo rendering process we do not have access to ground truth images with which to compare our current image estimate, as this would imply the availability of the final image. To evaluate our current image estimate we need to utilise NR-IQA methods which compare the image under evaluation to the distribution of natural images we are likely trying to compute.

When reviewing existing NR-IQA methods we observe that the vast majority of measures in the literature are trained on images from publicly available datasets such as Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], and IRCCyN/IVC [AB09], which contain clean reference images (photographs) and their synthetically distorted test images, coupled with subjective quality scores given by and pooled over a set of human observers. We find that in many cases synthetic distortions do not present a representative target for training and evaluating metrics that will be applied to naturally distorted images. In our experiments we show that we can train models directly on images containing naturally occurring distortions from unconverged Monte Carlo rendering processes.

We propose a deep-learning approach to NR-IQA trained specifically on noise from Monte Carlo rendering processes using the dataset of images we created, which significantly outperforms existing NR-IQA methods, and produces performance close to the approximated FR-IQA measure.

Chapter 6

Lastly, in chapter 6 we conclude with a summary of the contributions made in each of the three main chapters of the thesis, and present a summary of further work.

1.3 Contributions

The major contributions of this work are:

- A review of classical and recent works on Monte Carlo Simulations, Physically Based Rendering, Image Quality Assessment, and Machine Learning.

- A modern C++ framework for HPC built on top of MPI, designed to introduce no (or minimal) overheads while *drastically* reducing code complexity, increasing type-safety, and adding higher-level functionality such as deep-copy semantics.

- A dataset of images rendered with Monte Carlo rendering algorithms, computed using HPC, containing varying degrees of image quality intended for use in analyzing and developing IQA methods on domain specific image distortions.

- An ensemble study on the robustness of existing IQA measures when evaluating images created through Monte Carlo rendering processes using the aforementioned dataset.

- A deep-learning approach to the NR-IQA problem aimed at assessing the quality of Monte Carlo rendered images containing natural image distortions from the rendering process.

2 Related Work

In this chapter we will review the literature on Bayesian Statistics, Monte Carlo Simulations, Physically Based Rendering, Image Quality Assessment, and Machine Learning that form the foundation of the topics discussed throughout the remainder of this thesis.

2.1 Probability Theory

As a prerequisite to much of the information covered in this chapter relating to Monte Carlo methods and rendering processes we will first introduce some of the key concepts and relations from probability theory which appear frequently in those areas. Probability is a measure of the likelihood of an event occurring and can be represented as a numerical value between 0 and 1. For an event $\mathcal{A}$ we can say its probability of occurring independently is $\mathcal{P}(\mathcal{A}) \in [0, 1]$. An example of this could be the outcome of flipping a fair coin, we can say that $\mathcal{P}(\text{head}) = {}^1\!/_2$ and similarly $\mathcal{P}(\text{tail}) = {}^1\!/_2$. Since there are only two outcomes of flipping a fair coin it makes sense that the probabilities of each possible outcome should sum to 1.

We can formalize this idea by defining the joint probability of one event *or* another occurring, $\mathcal{P}(\mathcal{A}\ \textit{or}\ \mathcal{B}) = \mathcal{P}(\mathcal{A} \cup \mathcal{B}) = \mathcal{P}(\mathcal{A}) + \mathcal{P}(\mathcal{B})$. In our coin flipping example we can now write that $\mathcal{P}(\text{head} \cup \text{tail}) = \mathcal{P}(\text{head}) + \mathcal{P}(\text{tail}) = {}^1\!/_2 + {}^1\!/_2 = 1$. This definition of *or* holds because our events are mutually exclusive of one another, which is to say that both cannot occur at the same time — either the coin has landed on heads or on tails but never heads and tails. Or more formally, $\mathcal{P}(\text{head}\ \textit{and}\ \text{tail}) = \mathcal{P}(\text{head} \cap \text{tail}) = 0$.

In order to describe potentially overlapping events where $\mathcal{P}(\mathcal{A} \cap \mathcal{B}) > 0$ we need to extend our definition to account for the probability of events in the overlap between distributions being double counted when we sum the probabilities of the independent events, $\mathcal{P}(\mathcal{A} \cup \mathcal{B}) = \mathcal{P}(\mathcal{A}) + \mathcal{P}(\mathcal{B}) - \mathcal{P}(\mathcal{A} \cap \mathcal{B})$. In the case of mutually exclusive events such as our coin example we can see that the intersection of events occurs with 0 probability, meaning the definition simplifies to what we saw previously.

To calculate the probability of both events occurring for the above definition of *or* we need to define another logical construct, *and*. To illustrate this let's modify our coin flip example to be the outcome of flipping two fair coins independently of one another, with events head_0 and tail_0 for the first coin flip, and events head_1 and tail_1 for the outcome of the second coin flip. Given the small scale of this example we can enumerate the four possible outcomes as $(\text{head}_0, \text{head}_1)$, $(\text{head}_0, \text{tail}_1)$, $(\text{tail}_0, \text{head}_1)$, $(\text{tail}_0, \text{tail}_1)$. From this enumeration it is trivial to see that the probability of both coins landing on heads is $^1/_4$. Formally, we can say that the probability of two independent events both occurring is equal to the product of each of the events occurring, $\mathcal{P}(\mathcal{A} \cap \mathcal{B}) = \mathcal{P}(\mathcal{A})\mathcal{P}(\mathcal{B})$. In our two coin example this is, $\mathcal{P}(\text{head}_0 \cap \text{head}_1) = \mathcal{P}(\text{head}_0)\mathcal{P}(\text{head}_1) = {}^1/_2 * {}^1/_2 = {}^1/_4$.

Events are not always independent of one another, however. In many instances we would like to be able to compute the probabilities of events which are dependent on other events. An illustrative example of this is drawing cards from a deck of playing cards without replacement. By not replacing cards as they are drawn the distribution of possible outcomes from subsequent draws changes each time a card is removed from the deck, making each draw dependent on the outcome of previous draws. Imagine we want to compute the probability of two subsequent draws yielding aces, event ace_0 is the first draw and ace_1 is the second draw. The probability of drawing an ace as the first card is simply $\mathcal{P}(\text{ace}_0) = {}^4/_{52}$ as there are four aces in a 52 card deck. For the second card the distribution of the deck has now changed, we now have three aces left and only 51 cards in the deck giving probability $\mathcal{P}(\text{ace}_1) = {}^3/_{51}$ from the updated deck. Given that the change in distribution of the deck was conditional on the first card being an ace we say that the probability of the second draw is a conditional probability distribution $\mathcal{P}(\text{ace}_1|\text{ace}_0) = {}^3/_{51}$. Formally we say, $\mathcal{P}(\mathcal{B}|\mathcal{A})$ is the conditional probability of event $\mathcal{B}$ occurring, given that event $\mathcal{A}$ has already occurred. Putting this together we can compute the probability of the two non-independent events both occurring by taking the product of the first event and conditional second event, $\mathcal{P}(\text{ace}_0 \cap \text{ace}_1) = \mathcal{P}(\text{ace}_1|\text{ace}_0)\mathcal{P}(\text{ace}_0) = {}^3/_{51} * {}^4/_{52} \approx {}^9/_{2000}$. Formally this gives us, $\mathcal{P}(\mathcal{A} \cap \mathcal{B}) = \mathcal{P}(\mathcal{B}|\mathcal{A})\mathcal{P}(\mathcal{A})$.

2.1.1 Bayes' Theorem

As we have just seen the joint probability of two non-independent events can be modelled as the product of the probability of one event happening and probability of the other event happening given the first one occurred. Because logical *and* is commutative, $\mathcal{P}(\mathcal{A} \cap \mathcal{B}) = \mathcal{P}(\mathcal{B} \cap \mathcal{A})$, this implies that the order of events in our joint probability can be swapped while achieving the same result, equation 2.1.

$$\mathcal{P}(\mathcal{A} \cap \mathcal{B}) = \mathcal{P}(\mathcal{A}|\mathcal{B})\mathcal{P}(\mathcal{B}) = \mathcal{P}(\mathcal{B}|\mathcal{A})\mathcal{P}(\mathcal{A}) = \mathcal{P}(\mathcal{B} \cap \mathcal{A}) \tag{2.1}$$

This equality is a powerful tool as it allows us to model the relationship between a conditional event and its inverse. In the above equation it is trivial to see we can solve for the conditional probability $\mathcal{P}(\mathcal{A}|\mathcal{B})$ by dividing both sides of the equation by $\mathcal{P}(\mathcal{B})$, and similarly we can solve for $\mathcal{P}(\mathcal{B}|\mathcal{A})$ by dividing both sides by $\mathcal{P}(\mathcal{A})$. Equation 2.2 shows the case where we solve for $\mathcal{P}(\mathcal{A}|\mathcal{B})$.

$$\mathcal{P}(\mathcal{A}|\mathcal{B}) = \frac{\mathcal{P}(\mathcal{B}|\mathcal{A})}{\mathcal{P}(\mathcal{B})}\mathcal{P}(\mathcal{A}) \tag{2.2}$$

This equation is known as Bayes' Theorem, also called Bayes' Rule or Bayes' Law, and it formalizes a method which allows us to update our beliefs given an initial assumption and new evidence. In this equation $\mathcal{P}(\mathcal{A})$ is the prior distribution which represents our initial assumptions about event $\mathcal{A}$ occurring without any other knowledge. $\mathcal{P}(\mathcal{B})$ in the denominator represents the marginal distribution of the condition occurring independently without any other knowledge. And $\mathcal{P}(\mathcal{B}|\mathcal{A})$ represents the likelihood of the condition occurring given we know the outcome. The fraction $\frac{\mathcal{P}(\mathcal{B}|\mathcal{A})}{\mathcal{P}(\mathcal{B})}$ is referred to as the likelihood ratio which represents the "support" for our change in assumption. Finally, the result $\mathcal{P}(\mathcal{A}|\mathcal{B})$ forms the posterior distribution which represents our updated beliefs after taking the new evidence into account.

As an illustrative example of Bayes' Theorem let us explore the following problem. Imagine we have two bags, $\text{bag}_\mathcal{A}$ and $\text{bag}_\mathcal{B}$. $\text{bag}_\mathcal{A}$ contains two black balls and four white balls, $\text{bag}_\mathcal{B}$ contains three black balls and one white ball. If we randomly draw a single black ball from either bag, what is the probability the ball came from $\text{bag}_\mathcal{A}$? Formally we are asking to compute the conditional probability $\mathcal{P}(\text{bag}_\mathcal{A}|\text{black})$. With Bayes' Theorem in hand we begin by rewriting the formula with our variables and conditions (equation 2.3).

$$\mathcal{P}(\text{bag}_\mathcal{A}|\text{black}) = \frac{\mathcal{P}(\text{black}|\text{bag}_\mathcal{A})}{\mathcal{P}(\text{black})}\mathcal{P}(\text{bag}_\mathcal{A}) \tag{2.3}$$

To determine what we need to compute each part of our equation we can rewrite our problem as a table listing black versus white balls as the columns, and the contents of $\text{bag}_\mathcal{A}$ versus $\text{bag}_\mathcal{B}$ as the rows.

	black	white
$\text{bag}_\mathcal{A}$	2	4
$\text{bag}_\mathcal{B}$	3	1

Because the problem stipulated that the drawn ball could come from either bag we can see that the prior probability that "a" ball came from $\text{bag}_\mathcal{A}$ is simply $\mathcal{P}(\text{bag}_\mathcal{A}) = 1/2$. The likelihood of a ball being black given that it came from $\text{bag}_\mathcal{A}$ is simply the proportion of balls in $\text{bag}_{\mathcal{A}}$ that are black. From our table we can see that $\text{bag}_{\mathcal{A}}$ contains six balls, two of which are black — this gives us $\mathcal{P}(\text{black}|\text{bag}_{\mathcal{A}}) = {}^{2}/_{6}$. Finally, the marginal probability that a randomly drawn ball from either bag is black is simply the proportion of black balls out of all the balls. We can compute this by summing the black column of our table and dividing the sum of all values by it, giving $\mathcal{P}(\text{black}) = {}^{5}/_{10}$. Equation 2.4 shows the result of substituting in our computed probabilities and computing the answer to our question.

$$\begin{aligned}\mathcal{P}(\text{bag}_{\mathcal{A}}|\text{black}) &= \frac{\frac{2}{2+4}}{\frac{2+3}{2+3+4+1}} * \frac{1}{2} \\ &= \frac{{}^{2}/_{6}}{{}^{5}/_{10}}{}^{1}/_{2} \\ &= \frac{{}^{1}/_{3}}{{}^{1}/_{2}}{}^{1}/_{2} \\ &= {}^{1}/_{3}\end{aligned} \tag{2.4}$$

The resulting probability that a randomly drawn black ball came from $\text{bag}_{\mathcal{A}}$ is $\mathcal{P}(\text{bag}_{\mathcal{A}}|\text{black}) = {}^{1}/_{3}$. This is in contrast to our original prior assumption that because we drew the ball randomly from either bag that the probability would be ${}^{1}/_{2}$. The prior assumption did not take into account the additional information we had about the compositions of each bags contents. By utilizing this additional information we were able to correctly answer the posed question.

2.1.2 Discussion

Bayesian statistics has powerful applications in a broad range of problems that are of interest in computer graphics and machine learning. We can apply techniques from this area whenever we have initial assumptions about the distribution of our variables and would like to update our beliefs to take account for additional data which may challenge our initial assumptions. This framework allows us to account for variable change and correctly report the meaning of our statistics. In the following sections of this chapter we will see the repeated use of Bayesian statistics applied in areas such as Monte Carlo sampling methods and their subsequent application to physically based light transport simulations.

2.2 Monte Carlo (MC) Sampling

A Monte Carlo (MC) process is a well defined tool in mathematics that allows for the evaluation of difficult multidimensional integrals that would be prohibitively expensive or impossible to compute by alternative means. A simple Monte Carlo Estimator is given by equation 2.5. The expectation of a function f and its Probability Density Function (PDF) p over a given (n-dimensional) sample space Θ can be approximated by drawing N independent and identically distributed (i.i.d.) random samples $\theta_{1..N} \propto \Theta$ distributed uniformly within the sample space Θ, and taking the average of the function f for each of these samples scaled by the probability p of the sample being drawn. The Central Limit Theorem tells us that in the limit, as the number of samples goes to infinity, $N \to \infty$, the error of the estimator goes to zero.

$$\mathbb{E}[\![f(\Theta)]\!] = \mu = \int_\Theta f(\theta)p(\theta)d\theta \quad \approx \quad \mu_N = \frac{1}{N}\sum_{i=1}^{N} f(\theta_i)p(\theta_i) \qquad \text{where} \quad \theta \propto \Theta \tag{2.5}$$

The basic Monte Carlo sampler gives us a framework with which to approximate the expectation of arbitrary integrals with a finite number of samples. While it is true that as the number of samples used goes to infinity, $N \to \infty$, the error of the approximation goes to zero, in practice when the function being evaluated is expensive to compute, or has high variance, the number of samples needed to even roughly approximate the true expectation quickly becomes intractable. Another scenario where this can occur is when the probability distribution $p(\Theta)$ has a finite (and potentially small) volume compared to its domain (i.e. Θ is unbounded in $\mathbb{R}^n$). In such situations drawing samples uniformly from the full domain $\theta \propto \Theta$ might yield an inordinately large number of samples with little to no contribution to the final expectation. By instead drawing samples proportionally to the distribution $\theta \propto p(\Theta)$ we explicitly focus the computation to relevant areas under the integral. To compensate for oversampling regions of high contribution we need to divide the out the probability with which samples are generated. Simplifying through this has the effect of dropping the $p(\theta)$ term, yielding equation 2.6.

$$\begin{aligned} \mathbb{E}_p[\![f(\Theta)]\!] = \mu &= \int_\Theta \frac{f(\theta)p(\theta)}{p(\theta)}d\theta = \int_\Theta f(\theta)d\theta \\ \approx \mu_N &= \frac{1}{N}\sum_{i=1}^{N} \frac{f(\theta_i)p(\theta_i)}{p(\theta_i)} = \frac{1}{N}\sum_{i=1}^{N} f(\theta_i) \\ &\text{where} \quad \theta \propto p(\Theta) \end{aligned} \tag{2.6}$$

Variance of the random variable $f(\Theta)$ can be modelled as equation 2.7. This relies on the availability of the final expectation, meaning that when the functions f and p have unknown distributions the true variance of the random variable can only be determined after the simulation has converged.

$$\sigma^2 = \mathbb{V}_p[\![f(\Theta)]\!] = \mathbb{E}_p[\![f(\Theta)^2]\!] - \mu^2 \tag{2.7}$$

The variance of the final estimate μ is given as:

$$\mathbb{V}_p[\![\mu]\!] = \frac{\sigma^2}{\sqrt{N}} \tag{2.8}$$

which goes to zero as the number of samples goes to infinity, $N \to \infty$. Before convergence is achieved the sample variance [Zwi02] (equation 2.9) of the random variable $\{f(\theta_i)|i \in 1..N\}$ can be used to approximate the variance of $f(\Theta)$ where N is the number of samples currently used in the simulation and μ_N is the current estimate of μ.

$$\sigma^2 \approx \sigma^2_{N-1} = \mathbb{V}_p[\![\{f(\theta_i)|i \in 1..N\}]\!] = \frac{1}{N-1}\sum_{i=1}^{N}(f(\theta_i) - \mu_N)^2 \tag{2.9}$$

By applying equation 2.8 to the sample variance we can approximate the variance of μ as:

$$\mathbb{V}_p[\![\mu]\!] \approx \mathbb{V}_p[\![\mu_N]\!] = \frac{\sigma^2_{N-1}}{\sqrt{N}} \tag{2.10}$$

This relies on the ability to draw samples proportionally to the probability distribution $p(\Theta)$. Figure 2.1 shows an exemplar target distribution we will use to demonstrate several methods for generating samples with desired probability. The distribution is bivariate Gaussian $p(\Theta) = \mathcal{N}_2(\mu, \Sigma)$ with mean $\mu = [0, 0]$ and covariance matrix $\Sigma = \begin{bmatrix} 0.33 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$.

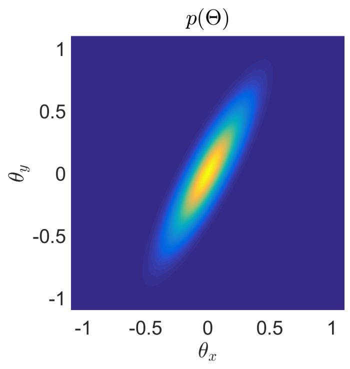


Fig. 2.1.: An exemplar probability distribution we will use to compare and contrast Monte Carlo sampling strategies. The distribution is a bivariate Gaussian $p(\Theta) = \mathcal{N}_2(\mu, \Sigma)$ with mean $\mu = [0, 0]$ and covariance matrix $\Sigma = \begin{bmatrix} 0.33 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$.

In this simple case of an n-dimensional multivariate Gaussian distribution, samples $x \propto \mathcal{N}_n(\mu, \Sigma)$ can be drawn in a closed form manner by transforming values drawn from independent univariate normal distributions $z \propto \mathcal{N}(0, 1)$ [Pap85]. First as a preprocessing step the covariance matrix Σ is decomposed via a Cholesky decomposition [DK03] yielding a row vector of n elements A, where $AA^\top = \Sigma$. For each sample to be drawn from the multivariate distribution a column vector of n elements z is constructed where each element is drawn from an independent normal distribution, $z_i \propto \mathcal{N}(0,1)$. Finally, the independent vector z is transformed by A and the multivariate mean μ by $x = \mu + Az$ giving a vector of n elements x where $x \propto \mathcal{N}_n(\mu, \Sigma)$. Figure 2.2 shows the results of drawing $10,000$ random i.i.d. samples from $p(\Theta)$ using the above method. The middle (b) and right (c) sub-plots visualize the correlation from one sample to the next, where correlation in each axis is signified by a linear trend along the diagonal of the respective plots. In the case of the closed form sampling method used here samples are drawn independently of one another and so no linearity is visible in the plots.

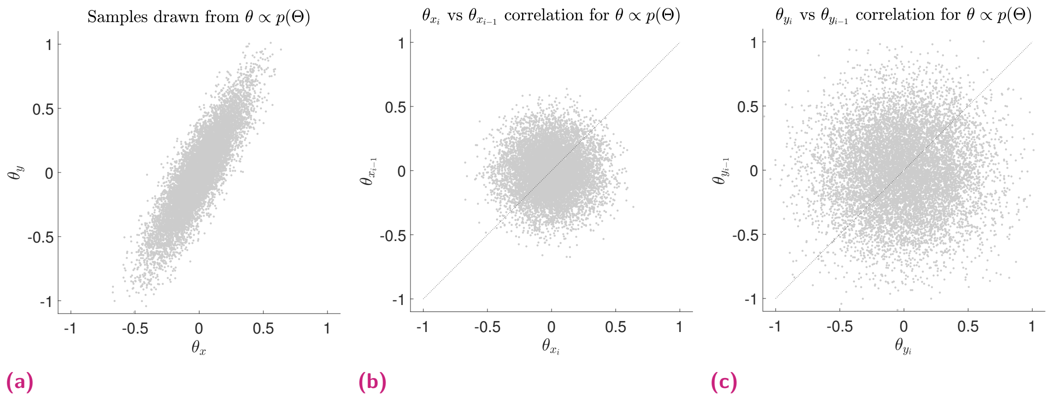


Fig. 2.2.: An example of the scenario where there is a closed form method for drawing samples directly from the distribution $p(\Theta)$. (a) $10,000$ random i.i.d. samples are drawn from $p(\Theta)$. (b) The x component of the i^{th} random sample is plotted w.r.t. the x component of the $(i-1)^{th}$ sample. (c) The y component of the i^{th} random sample is plotted w.r.t. the y component of the $(i-1)^{th}$ sample. In (b) & (c) no linear trend emerges indicating that samples are independently distributed and uncorrelated w.r.t. one another.

2.2.1 Accept Reject Sampling (ARS)

A closed form solution for drawing samples proportionally to the desired distribution $p(\Theta)$ may not always be available, or may be prohibitively expensive to evaluate. In such cases we can use a coarse approximation $q(\Theta) \approx p(\Theta)$ which can be sampled from more efficiently to draw samples which we can then transform into the target distribution $p(\Theta)$. Accept Reject Sampling (ARS) [Cas+04] works by conditionally discarding samples drawn proportionally to the auxiliary proposal distribution $q(\Theta)$ according to an acceptance probability based on the ratio between between the target and proposal distributions. Equation 2.11 gives probability of accepting a sample drawn from $q(\Theta)$ where M is a hand tuned value giving the upper-bound for the probability of acceptance.

$$P_{accept} = \min\left(1, \frac{p(\theta)}{Mq(\theta)}\right) \quad \text{where} \quad \theta \propto q(\Theta) \tag{2.11}$$

The full ARS scheme is given by algorithm 1. To generate a batch of N i.i.d. samples the algorithm loops to N only incrementing when a sample is accepted. At each iteration a sample $\theta' \propto q(\Theta)$ is proposed using the auxiliary proposal distribution and its probability of acceptance is calculated. A trigger value $\xi \propto \mathcal{U}(0,1)$ is sampled from the uniform distribution, and the sample is conditionally accepted or rejected based on whether the trigger value is less than the probability of acceptance.

Algorithm 1: ARS algorithm for drawing random samples proportional to the target distribution $p(\Theta)$ using the auxiliary distribution $q(\Theta)$ which coarsely approximates the target and is simpler to sample from.

Input : The number of samples N to draw from the target distribution $p(\Theta)$, using the auxiliary proposal distribution $q(\Theta)$ which can be directly sampled from, and the tunable upper-bound for the ratio M between the $p(\Theta)$ and $q(\Theta)$ distributions.

Output : A set of N samples $\{\theta_i | i \in 1..N\}$ drawn proportionally to $p(\Theta)$.

```
function ARS $(N, p(\Theta), q(\Theta), M)$
    $i = 1$                                              // Initialize sample index.
    while $(i \leq N)$ do                                // While more samples are needed.
        $\theta' \propto q(\Theta)$                      // Draw sample from proposal distribution.
        $P_{accept} = \min\left(1, \frac{p(\theta')}{Mq(\theta')}\right)$   // Compute probability of acceptance.
        $\xi \propto \mathcal{U}(0,1)$                   // Sample acceptance trigger.
        if $(\xi < P_{accept})$ then                     // Accept or reject new sample?
            $\theta_i = \theta'$
            $i = i + 1$                                  // Accept new sample and increment index.
        end
    end
    return $\{\theta_i | i \in 1..N\}$                   // Result samples.
end
```

To apply ARS to sample from the exemplar distribution we first define the auxiliary proposal distribution $q(\Theta) = \mathcal{U}_2(-1,1)$ to be a 2-dimensional uniform distribution with support between $[-1,1)$ in both dimensions, and rejection rate $M = 20$. Figure 2.3 shows the effect of ARS with the above settings. ARS produces i.i.d. samples which are uncorrelated w.r.t. one another. A limitation of ARS is that its efficiency is bounded by how closely the auxiliary proposal distribution $q(\Theta)$ approximates the target distribution. When the target distribution $p(\Theta)$ has unknown distribution, perhaps due to being conditional on many parameters unknown before simulation time, an auxiliary distribution $q(\Theta)$ might be impractical to select reliably making ARS difficult to apply effectively.

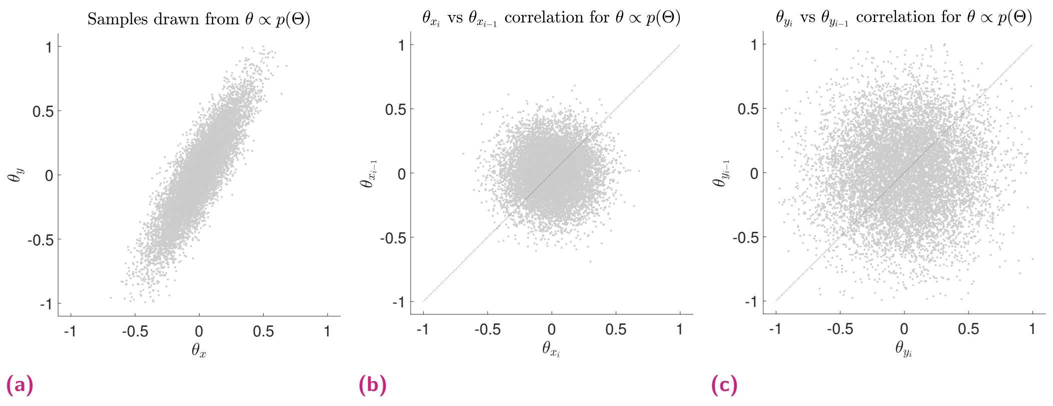


Fig. 2.3.: An example of ARS for drawing samples from the distribution $p(\Theta)$ using the auxiliary proposal distribution $q(\Theta)$ which coarsely approximates $p(\Theta)$, and a rejection rate M. (a) $10,000$ random i.i.d. samples are drawn from $p(\Theta)$. (b) The x component of the i^{th} random sample is plotted w.r.t. the x component of the $(i-1)^{th}$ sample. (c) The y component of the i^{th} random sample is plotted w.r.t. the y component of the $(i-1)^{th}$ sample. In (b) & (c) no linear trend emerges indicating that samples are independently distributed and uncorrelated w.r.t. one another. This is consistent with the closed form solution shown in figure 2.2 as both methods draw i.i.d. samples.

2.2.2 Markov Chain Monte Carlo (MCMC) Sampling

Markov chains represent states in a system and the probabilities of transitioning between them. In the context of Monte Carlo sampling methods Markov chains can be used to model a deterministic sampling process whereby each random sample can be drawn given the location of the previous sample. By iteratively drawing samples in this manner the location of the sampler appears to randomly walk around the sample space in a semi-deterministic fashion.

2.2.2.1. Metropolis-Hastings Monte Carlo (MHMC) Sampling

With ARS we saw that an auxiliary proposal distribution $q(\Theta)$ could be used to sample from the target distribution $q(\Theta)$. However, this relied on the availability of a distribution which coarsely approximated the entire target distribution to a reasonable degree of accuracy. In practice such an auxiliary distribution may not be available or may be as expensive to sample from as the original target distribution. In Metropolis-Hastings Monte Carlo (MHMC) [Has70] independent samples drawn from the full proposal distribution are substituted with Markov chain updates drawn from a local conditional proposal distribution centred around the previous sample. Drawing from Bayesian statistics, the conditional proposal distribution $q(\theta'|\theta)$ represents the probability of a new sample θ' being drawn given the previous sample θ, and can be sampled directly to yield $\theta' \propto q(\theta)$.

As with ARS proposed samples are accepted or rejected with a probability determined by the relationship between the target and proposal distributions. Due to the deterministic nature of Markov Chain Monte Carlo (MCMC) methods, in order to keep the sampler unbiased time reversibility must be maintained. This constrains the proposal distribution to be symmetric meaning the probability of transitioning from state $\theta \rightarrow \theta'$ must equal the probability of transitioning from $\theta' \rightarrow \theta$. In practice this may not be the case and so the transition probability for both directions is incorporated into the acceptance probability as shown in equation 2.12.

$$P_{accept} = \min\left(1, \frac{p(\theta')q(\theta|\theta')}{p(\theta)q(\theta'|\theta)}\right) \tag{2.12}$$

The full MHMC sampling method is shown in algorithm 2. It is almost identical in implementation to the ARS algorithm with the main difference being that each proposal sample is drawn from the conditional distribution centred around the previously accepted sample. The initial seed location for the sampler is drawn from the full sample space and the choice of this initial location can often be the cause of bias early in the sampling process. A common way to avoid this issue is through the use of a burn-in sampling phase, where the first n samples drawn from the sampler are discarded with the aim that by the n^{th} sample the effects of this initial bias will have been dissipated.

The efficiency and quality of the MHMC sampler is controlled by the choice of conditional proposal distribution $q(\theta'|\theta)$. If the support of proposal distribution is too wide around the currently accepted sample then the MHMC sampler will reduce to that of an ARS sampler, conversely if the support is too narrow the sampler will spend an inordinate amount of time exploring only a narrow region of the target distribution and will be slow to converge properly.

Applying MHMC sampling to the exemplar target distribution we select a conditional proposal distribution with $q(\theta) = \mathcal{N}_2(\theta, \sigma\mathbf{I})$ where $\sigma = 0.01$ which is a small bivariate Gaussian distribution with zero covariance centred around the previous sample θ. The variance of the independent Gaussian distributions was hand tuned to find an appropriate balance between sample space exploration and rate of proposal acceptance.

Algorithm 2: MHMC algorithm for drawing random samples proportional to the target distribution $p(\Theta)$ by constructing a Markov Chain of samples using the conditional proposal distribution $q(\theta'|\theta)$ which models the symmetric probability of transitioning from one sample to another.

Input : The number of samples N to draw from the target distribution $p(\Theta)$, using the conditional proposal distribution $q(\theta'|\theta)$ which models the symmetric probability of transitioning from one sample to another.

Output : A set of N samples $\{\theta_i | i \in 1..N\}$ drawn proportionally to $p(\Theta)$.

```
function MHMC $\left(N, p(\Theta), q(\theta'|\theta)\right)$
    $\theta \in \Theta$          // Initial sample location drawn uniformly or hardcoded.
    $i = 1$                                        // Initialize sample index.
    while $(i \leq N)$ do                       // While more samples are needed.
        $\theta' \propto q(\theta)$       // Draw sample from conditional proposal distribution.
        $P_{accept} = \min\left(1, \frac{p(\theta')q(\theta|\theta')}{p(\theta)q(\theta'|\theta)}\right)$     // Compute probability of acceptance.
        $\xi \propto \mathcal{U}(0,1)$                         // Sample acceptance trigger.
        if $(\xi < P_{accept})$ then                   // Accept or reject new sample?
            $\theta_i = \theta = \theta'$
            $i = i + 1$                 // Accept new sample and increment index.
        end
    end
    return $\{\theta_i | i \in 1..N\}$                          // Result samples.
end
```

Figure 2.4 shows the results of generating $10,000$ samples from the accepted states of the MHMC random walk. In (a) the green line denotes the path of the sampler over the 50 most recent accepted proposals. In (b) and (c) we see a strong linear trend along the diagonal which signifies the correlation from one sample to the next. At low sample counts this correlation has a large effect on the quality of the sampler as it is likely to have only sampled a small sub region of the full sample space, introducing bias into the estimate of the equation being approximated. This bias only goes away as more samples are added, and the sampler is able to traverse the full sample space in a representative manner.

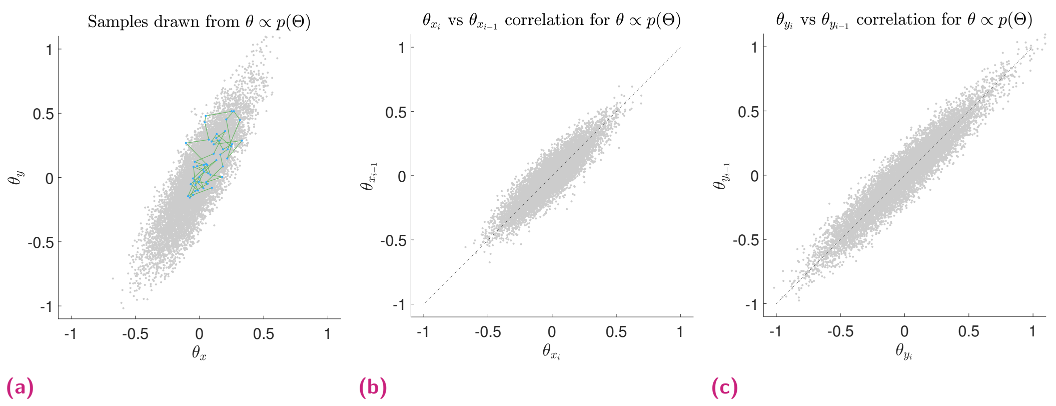


Fig. 2.4.: An example of MHMC for drawing samples from the distribution $p(\Theta)$ using the auxiliary conditional proposal distribution $q(\theta'|\theta)$ which proposes a sample in the local neighbourhood of the previous sample. (a) $10,000$ random Markovian samples (grey dots) are drawn from $p(\Theta)$, where the trajectory of the final 50 samples are highlighted with blue dots and a green line. (b) The x component of the i^{th} random sample is plotted w.r.t. the x component of the $(i-1)^{th}$ sample. (c) The y component of the i^{th} random sample is plotted w.r.t. the y component of the $(i-1)^{th}$ sample. In (b) & (c) a strong linear trend emerges along the diagonal indicating that samples are not independently distributed and are strongly correlated w.r.t. one another. This is due to the conditional proposal distribution $q(\theta'|\theta)$ which proposes a new sample θ' from a symmetric distribution centred around the current sample θ.

2.2.2.2. Hamiltonian Monte Carlo (HMC) Sampling

Hamiltonian Monte Carlo (HMC), sometimes referred to as Hybrid Monte Carlo is a random walk sampling scheme derived from the modelling of a physical process [Nea11]. In MHMC, new samples were proposed by drawing them from a conditional proposal distribution centred around the previously accepted sample. The choice of proposal distribution therefore needed to be tuned to the problem at hand. In HMC, we instead envision the target distribution $p(\Theta)$ in terms of the potential $U(x) = -\ln(p(x))$ and kinetic $K(v) = \frac{1}{2}\sum\limits_i v_i^2$ energy of a frictionless puck sliding on the surface of the distribution. By simulating the puck's movements over a fixed interval we find that the puck will gravitate towards regions of low potential energy (high probability). By sampling from the end points of these simulated runs a valid MCMC sampler can be derived. To sample the end points, at the end of a simulated run the samplers position is conditionally accepted using a variant on the MHMC acceptance formula, which balances the change in total energy of the system from the start point to the end point of the simulated run (equation 2.13).

$$P_{accept} = \min\left(1, \exp((U(x_0) + K(v_0)) - (U(x_s) + K(v_s)))\right) \tag{2.13}$$

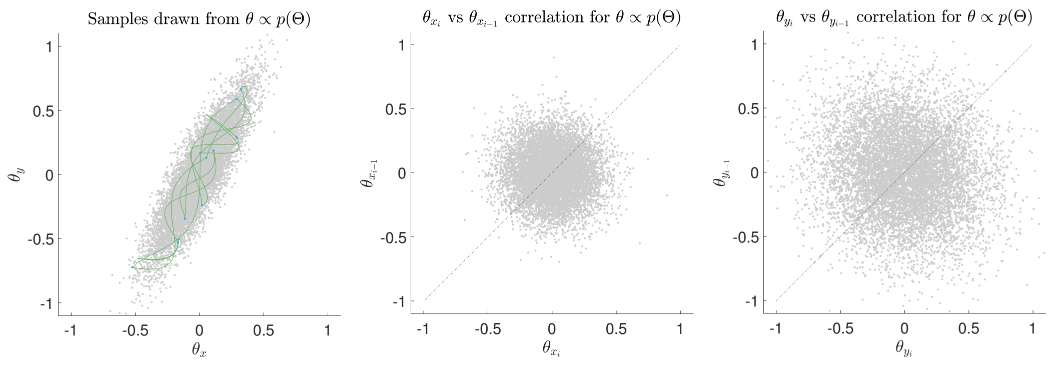
Algorithm 3: HMC algorithm for drawing random samples proportional to the target distribution $p(\Theta)$ by constructing a Markov Chain of samples from the end points of Hamiltonian Dynamics run trajectories.

Input : The number of samples N to draw from the target distribution $p(\Theta)$, and the HMC run length $\mathcal{L}$ and step size $\mathcal{E}$.

Output : A set of N samples $\{\theta_i | i \in 1..N\}$ drawn proportionally to $p(\Theta)$.

```
function HMC $\left(N, p(\Theta), \mathcal{E}, \mathcal{L}\right)$
    $U(x) = -\ln(p(x))$                 // Potential Energy of the target distribution.
    $dU(x) = \frac{\partial U(x)}{\partial x}$               // Jacobian of the Potential Energy distribution.
    $K(v) = \frac{1}{2}\sum_i v_i^2$                                 // Kinetic Energy Distribution.
    $\theta \in \Theta$            // Initial sample location drawn uniformly or hardcoded.
    $i = 1$                                               // Initialize sample index.
    while $(i \leq N)$ do                              // While more samples are needed.
        $x_0 = x_s = \theta$                 // Run starts from previously accepted sample.
        $v_0 = v_s \propto \mathcal{N}_n(0, \mathbf{I})$             // Momentum sampled from zero mean Gaussian.
        $j = 1$
        while $(j \leq \mathcal{L})$ do              // While more leapfrog steps are needed.
            $v_s = v_s - \frac{\mathcal{E}}{2} dU(x_s)$                 // First half step for momentum.
            $x_s = x_s + \mathcal{E} v_s$                           // Full step for position.
            $v_s = v_s - \frac{\mathcal{E}}{2} dU(x_s)$                  // Last half step for momentum.
            $j = j + 1$
        end
        $P_{accept} = \min\left(1, \exp((U(x_0) + K(v_0)) - (U(x_s) + K(v_s)))\right)$
        $\xi \propto \mathcal{U}(0, 1)$                                // Sample acceptance trigger.
        if $(\xi < P_{accept})$ then                          // Accept or reject new sample?
            $\theta = x_s$                                  // Update sample to be recorded.
        end
        $\theta_i = \theta$
        $i = i + 1$                                // Record sample and increment index.
    end
    return $\left\{\theta_i | i \in 1..N\right\}$                                  // Result samples.
end
```

The full HMC sampling method is shown in algorithm 3. As with MHMC the sampler is initialized at a random or hard-coded location. Each simulated run starts at the location of the previous sample and a random momentum is sampled from the kinetic energy distribution $K(v) = \frac{1}{2}\sum_i v_i^2$. This usually takes the form of a multivariate Gaussian with zero covariance, $v_0 \propto \mathcal{N}_n(0, \mathbf{I})$. The simulation is then evaluated for a fixed number of steps $\mathcal{L}$ and a fixed step size $\mathcal{E}$ which are hand tuned parameters for the sampler. Each step is evaluated in a leapfrog fashion, first the momentum of the puck is updated halfway using the local gradient of potential energy distribution, then position is updated using the adjusted momentum, and finally the momentum is updated the remaining halfway using the new local gradient of the potential energy distribution. By repeating the leapfrog steps the puck traverses the target distribution proportionally to its potential energy, spending more time in high probability regions which have low potential energy. In MHMC when a proposed sample was rejected the sampler continued to propose new samples until one was accepted. In order to converge to the correct target distribution, HMC sampling slightly alters this, reusing the previously accepted sample again when a new sample is rejected. The next simulated run begins from this location also, with a newly sampled momentum.



(a) (b) (c)

Fig. 2.5.: An example of HMC for drawing samples from the distribution $p(\Theta)$ using leapfrog updates. (a) $10,000$ random samples (grey dots) are drawn from $p(\Theta)$, where the trajectory and sub-steps of the final 10 accepted samples are highlighted with blue dots and a green line. (b) The x component of the i^{th} random sample is plotted w.r.t. the x component of the $(i-1)^{th}$ sample. (c) The y component of the i^{th} random sample is plotted w.r.t. the y component of the $(i-1)^{th}$ sample. In (b) and (c) no strong linear trend emerges indicating that despite new samples being generated from the prior one, the properly tuned HMC sampling scheme is able to generate samples that appear independently distributed and uncorrelated w.r.t. one another.

Figure 2.5 shows the results of generating $10,000$ samples using the HMC sampling algorithm with $\mathcal{L} = 20$ and $\mathcal{E} = 0.03$. In (a) the green trajectory denotes the path taken by the sampler over the 50 most recent simulated runs. The green dots show the end points of the runs on the trajectory. It can be observed that the trajectory varies smoothly except at these points where the momentum values are randomly resampled. In (b) and (c), even though the HMC method is still drawing samples according to a Markov chain random walk like MHMC, no discernible linearity presents itself. This allows the sampler to be representative even at low sample counts.

The correctly tuned HMC sampler offers a good balance between Markov chain style exploration of difficult regions of the target distribution and the low correlation between the samples of a non-Markov chain sampler. This comes at the cost of needing to evaluate the gradient of the potential energy distribution at each step of the leapfrog update. When it is computationally expensive to evaluate $p(\Theta)$, the derivative of its potential energy distribution, or to compute offset sample locations for use in a finite difference approximation of the derivative, HMC can be impractical or intractable to use.

2.2.3 Importance Sampling (IS)

So far we have investigated methods for drawing samples from the target distribution $p(\Theta)$ for the purpose of evaluating a Monte Carlo estimator over an integral sampled w.r.t. to the probability distribution $p(\Theta)$. In ARS this was accomplished by sampling directly from an auxiliary proposal distribution $q(\Theta)$ which closely approximated $p(\Theta)$, and by conditionally accepting or rejecting these proposed samples to compensate for the differences between the two distributions. This strategy worked, but caused a large number of samples to be discarded when the probability of acceptance was low, wasting the computational effort used to generate the proposal. In Importance Sampling (IS) we aim to use every proposed sample, even those that occur with low probability, directly within the Monte Carlo estimator itself [Ken16].

We begin with the standard Monte Carlo estimator (equation 2.14) which is sampled w.r.t. to the uniform sample space $\theta \propto \Theta$. By introducing an the auxiliary proposal distribution $q(\Theta)$ from ARS we can see that the equation remains consistent if we both apply and compensate for the auxiliary proposal distribution by having it in both the numerator and the denominator (equation 2.14).

$$\begin{aligned}\mathbb{E}[\![f(\Theta)]\!] = \mu &= \int_{\Theta} f(\theta)p(\theta)d\theta = \int_{\Theta} \frac{f(\theta)p(\theta)}{q(\theta)}q(\theta)d\theta \\ \approx \mu_N &= \frac{1}{N}\sum_{i=1}^{N} f(\theta_i)p(\theta_i) = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\theta_i)p(\theta_i)}{q(\theta_i)}q(\theta_i) \\ &\text{where} \quad \theta \propto \Theta\end{aligned} \tag{2.14}$$

Recall that in order to sample w.r.t. the target distribution $p(\Theta)$ in the original Monte Carlo sampler (equation 2.6) we had to divide by the probability of sampling a given point $p(\theta)$ to compensate for over sampling. In order to sample the estimator w.r.t. the auxiliary proposal distribution we now need to divide by the probability of the sample being drawn from the proposal distribution (equation 2.15).

$$\begin{aligned}\mathbb{E}_q[\![f(\Theta)]\!] = \mu &= \int_{\Theta} \frac{\frac{f(\theta)p(\theta)}{q(\theta)}q(\theta)}{q(\theta)}d\theta = \int_{\Theta} \frac{f(\theta)p(\theta)}{q(\theta)}d\theta \\ \approx \mu_N &= \frac{1}{N}\sum_{i=1}^{N} \frac{\frac{f(\theta_i)p(\theta_i)}{q(\theta_i)}q(\theta_i)}{q(\theta_i)} = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\theta_i)p(\theta_i)}{q(\theta_i)} \\ &\text{where} \quad \theta \propto q(\Theta)\end{aligned} \tag{2.15}$$

2.2.3.1. Self Normalized Importance Sampling

The importance sampling scheme discussed above can be used when both the target $p(\theta)$ and auxiliary $q(\theta)$ distributions can be evaluated directly, and the auxiliary distribution can be sampled from. In many cases these distributions are only known up to a constant scaling factor such that $p(\theta) = c_p p_0(\theta)$ and $q(\theta) = c_q q_0(\theta)$ [Ken16]. c_p and c_q can be derived from the integral on each distribution by computing $c_p = {}^1/_{\int_\Theta p_0(\theta)d\theta}$ and $c_q = {}^1/_{\int_\Theta q_0(\theta)d\theta}$ but this may be prohibitively expensive to compute. The need to compute the scaling constants can be avoided by recognizing that the ratio between distributions is more useful than the values themselves.

$$
\begin{aligned}
\mathbb{E}_q[\![f(\Theta)]\!] = \mu = \int_\Theta \frac{f(\theta)p(\theta)}{q(\theta)} d\theta &= \frac{\int_\Theta f(\theta)\frac{p(\theta)}{q(\theta)}d\theta}{\int_\Theta \frac{p(\theta)}{q(\theta)}d\theta} \\
&= \frac{\int_\Theta f(\theta)\frac{c_p}{c_q}\frac{p_0(\theta)}{q_0(\theta)}d\theta}{\int_\Theta \frac{c_p}{c_q}\frac{p_0(\theta)}{q_0(\theta)}d\theta} \\
&= \frac{\int_\Theta f(\theta)\frac{p_0(\theta)}{q_0(\theta)}d\theta}{\int_\Theta \frac{p_0(\theta)}{q_0(\theta)}d\theta} \\
\approx \mu_N = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\theta_i)p(\theta_i)}{q(\theta_i)} &= \frac{\frac{1}{N}\sum_{i=1}^{N} f(\theta_i)\frac{p(\theta_i)}{q(\theta_i)}}{\frac{1}{N}\sum_{i=1}^{N}\frac{p(\theta_i)}{q(\theta_i)}} \\
&= \frac{\frac{1}{N}\sum_{i=1}^{N} f(\theta_i)\frac{c_p}{c_q}\frac{p_0(\theta_i)}{q_0(\theta_i)}}{\frac{1}{N}\sum_{i=1}^{N}\frac{c_p}{c_q}\frac{p_0(\theta_i)}{q_0(\theta_i)}} \\
&= \frac{\frac{1}{N}\sum_{i=1}^{N} f(\theta_i)\frac{p_0(\theta_i)}{q_0(\theta_i)}}{\frac{1}{N}\sum_{i=1}^{N}\frac{p_0(\theta_i)}{q_0(\theta_i)}} \\
&= \frac{\sum_{i=1}^{N} f(\theta_i)\frac{p_0(\theta_i)}{q_0(\theta_i)}}{\sum_{i=1}^{N}\frac{p_0(\theta_i)}{q_0(\theta_i)}} \\
\text{where} \quad \theta &\propto q(\Theta)
\end{aligned}
\tag{2.16}
$$

Equation 2.16 shows the derivation of self-normalized importance sampling whereby the constraint that $p(\theta)$ and $q(\theta)$ each integrate to 1 is removed. It begins by recognizing that if $\int_\Theta p(\theta)d\theta$ and $\int_\Theta q(\theta)d\theta$ both integrate to 1, then the integration of the ratio $\int_\Theta \frac{p(\theta)}{q(\theta)}d\theta$ also integrates to 1. By rewriting the original importance sampling integral over 1 and then substituting the denominator for the integral of the ratio a valid estimator can still be constructed. Rewriting instances of $p(\theta)$ and $q(\theta)$ as the product their scaling constants and un-normalized distributions $c_p p_0(\theta)$ and $c_q q_0(\theta)$ the constant ratio $\frac{c_p}{c_q}$ appears in both the numerator and denominator of the final estimator. Removing the constant terms we can see that the ratio of the integrals on the normalized distributions $p(\theta)$ and $q(\theta)$ is equal to the same ratio of integrals on the un-normalized distributions $p_0(\theta)$ and $q_0(\theta)$, eliminating the need to compute the scaling constants. When substituting the integrals with the numerator and denominator estimators over a discrete set of samples a minor optimization can be made by recognizing that the constant $\frac{1}{N}$ term in both the numerator and denominator can also be dropped yielding a ratio between summations rather than averages.

2.2.4 Multiple Importance Sampling (MIS)

In IS and ARS we saw that we could sample according to the target distribution $p(\Theta)$ using an auxiliary proposal distribution $q(\Theta)$ which closely approximates the target distribution but is easier to sample from. However, in many cases a single proposal distribution that approximates the entire target distribution may not be available. Rather, proposal distributions which accurately represent different regions of the target distribution may be more easily constructed such that the union of these distributions closely approximates the full target distribution. It is therefore desirable to construct an estimator which combines samples generated with all available sampling techniques such that the combined estimator still correctly converges to the target distribution. Such a method is proposed in the seminal work on Multiple Importance Sampling (MIS) by Veach and Guibas [VG95] and later formalized in Veach's doctoral thesis [Vea97].

Equation 2.17 shows the general form of the multi-sample estimator. For a set of K proposal distributions $q_i(\Theta)$ which are potentially overlapping, the estimator is defined as the weighted summation of estimates computed from the samples drawn from each proposal distribution independently. The weighting function $w_i(\theta)$ determines the amount of contribution a sample makes to the final estimate, given the proposal distribution it was drawn from and where it is located within the sample space. From each proposal distribution $q_i(\Theta)$ a fixed number of samples n_i are drawn and combined with the familiar importance sampling equation (equation 2.14).

$$\begin{gathered}\mathbb{E}[\![f(\Theta)]\!] = \mu \approx \mu_N = \sum_{i=1}^{K} \frac{1}{n_i} \sum_{j=1}^{n_i} w_i(\theta_{i,j}) \frac{f(\theta_{i,j}) p(\theta_{i,j})}{q_i(\theta_{i,j})} \\ \text{where} \quad \theta_i \propto q_i(\Theta) \quad \text{and} \quad N = \sum_{i=1}^{K} n_i\end{gathered} \tag{2.17}$$

In the original work by Veach and Guibas the number of samples drawn using each proposal distribution was constrained to be equal between all sampling distributions. The justification for this was that the gain in sampling efficiency from drawing varying number of samples with each technique would only matter up to a theoretical limit, and that a greater impact on efficiency could be made by constraining the number of samples proposed and instead improving the efficiency with which samples were drawn. This claim was made in the context of Bidirectional Path Tracing (BDPT) (section 2.3.6), where drawing samples generated by ray tracing is a severe performance bottleneck. In their work, Veach and Guibas propose to draw a single sample from all proposal distributions in the multi-sample estimator simultaneously by constructing a single eye and light sub-path by ray tracing, and extracting individual samples from the possible combinations of the path vertices. The individual samples can then be recombined with optimum weights computed using multiple-importance sampling.

In recent work, Sbert and Havran [SH17] have shown that in the general case for multi-sample estimators, a large increase in sampling efficiency can be achieved by adaptively tuning the number of samples taken from each proposal distribution as the estimator is evaluated. In their work the sample variance of each estimator is tracked using a secondary estimator on the optimal proportion of samples to take from each proposal distribution, constructed using the Harmonic Mean [LP08] which is robust to outliers, and the tracked variances.

The choice of weighting function used to combine samples generated with different proposal distributions is at the core of the multi-sample estimator and can have a large impact on its performance. Weighting functions are afforded a large amount of flexibility and are only subject to two constraints on their behaviour. Equation 2.18 shows the constraints for a valid weighting function. **W1** ensures that for a given sample generated from any of the proposal distributions the weights assigned to it for each proposal distribution should sum to 1. This ensures that the multi-sample estimator is a weighted average of individual estimators. **W2** ensures that where the i^{th} proposal distribution is zero the weight assigned to samples by the i^{th} weighting function is also zero. In this way it ensures that proposal distributions that only represent a sub-space for the full integration space do not bias the estimate of the integral in regions outside of their support.

$$\begin{aligned} &\textbf{(W1)} \quad \sum_{i=1}^{K} w_i(\theta) = 1 \\ &\textbf{(W2)} \quad w_i(\theta) = 0 \quad \text{where} \quad q_i(\theta) = 0 \end{aligned} \tag{2.18}$$

The above weighting function constraints allow for a variety of sampling techniques to be performed by the multi-sample estimator. For example, if we slice the full sample space Θ into separate non-overlapping regions Θ_i for $1 \leq i \leq K$, and define a weighting function such that a given sample has weight one if and only if it was sampled from that region of the sample space (equation 2.19), then this represents the stratified sampling technique for non-overlapping strata. We can see that this weighting function fulfils both constraints as only one of the weighting functions will have a value of one for a given sample, and because each proposal distribution only produces samples within its non-overlapping window where the weight for it is non-zero.

$$w_i(\theta) = \begin{cases} 1 & \text{if } \theta \in \Theta_i \\ 0 & \text{else} \end{cases} \tag{2.19}$$

The real power of the multi-sample integrator comes from its ability to combine the benefits of stratified sampling, which allows for a good distribution of samples to be generated across the entire sample space; and the benefits of importance sampling, which allows for samples to be generated proportionally to the distribution being estimated. When there is no single representative proposal distribution for drawing samples proportionally to the target distribution it can be advantageous to instead sample from multiple distributions which are each representative of different regions within the sample space. These regions may overlap, and in such cases it is necessary that the weighting function reflects the quality of the contribution that can be made by each of the overlapping proposal distributions at that point.

The Balance Heuristic (equation 2.20) is one such weighting function. It is defined as the ratio between the value of the i^{th} proposal distribution and the sum of all the proposal distributions at the sample point. This ratio is scaled by the proportion of the total samples that were drawn explicitly from the i^{th} distribution to remove bias from oversampling on one or more of the proposal distributions. We can see that over all weighting functions $w_i(\theta)$ this satisfies both constraints **W1** and **W2**.

$$w_i(\theta) = \frac{n_i q_i(\theta)}{\sum_{k=1}^{K} n_k q_k(\theta)} \tag{2.20}$$

In scenarios where one proposal distribution performs better than its peers, exacerbating the effect of the best performing proposal can lower the overall variance of the estimator. By raising both the numerator and denominator of the Balance Heuristic to the power of a constant β we increase the effect of the strongest proposal distribution at the given sample location. This leads to the Power Heuristic (equation 2.21) which is most commonly used with $\beta = 2$.

$$w_i(\theta) = \frac{\left(n_i q_i(\theta)\right)^{\beta}}{\sum_{k=1}^{K} \left(n_k q_k(\theta)\right)^{\beta}} \tag{2.21}$$

2.2.4.1. One Sample Estimator

As discussed above in Veach and Guibas' original work on bidirectional path tracing the estimator was constrained to use a single sample from each proposal distribution so that all samples could be generated by a single ray tracing pass. This constraint leads to the simplified one sample estimator shown in equation 2.22.

$$F = \sum_{i=1}^{K} w_i(\theta_i) \frac{f(\theta_i)p(\theta_i)}{q_i(\theta_i)} \qquad \text{where} \quad \theta_i \propto q_i(\Theta) \tag{2.22}$$

In this form a coarse approximation of the expectation F is computed from a single sample from each simultaneously generated proposal. In order to ensure convergence multiple estimates of F are computed and combined using a standard Monte Carlo estimator (equation 2.23). It is important to note that the number of samples N now represents a different quantity from previous estimators as each MIS sample is the ensemble of a single sample drawn from each proposal. However, when samples are drawn simultaneously as in bidirectional path tracing, the efficient use of existing information otherwise wasted by the single sample estimators above makes a compelling argument for measuring performance of MIS w.r.t. to the number of samples in the primary estimator, N.

$$\mathbb{E}[\![f(\Theta)]\!] \approx \frac{1}{N}\sum_{j=1}^{N} F_j \tag{2.23}$$

Finally, due to the above simplifications the weighting function no longer needs to consider the proportion of samples generated with each proposal distribution. Equation 2.24 shows the simplified form of the Power Heuristic for the one sample estimator.

$$w_i(\theta) = \frac{q_i(\theta)^\beta}{\sum_{k=1}^{K} q_k(\theta)^\beta} \tag{2.24}$$

2.2.5 Discussion

Monte Carlo methods offer us a way to approximate the expectation of complex integrals by expending a finite amount of computation. These methods work by averaging many samples drawn randomly from the integration domain. As we increase the amount of computation dedicated to the approximation the variance of the estimator goes to zero. In the limit, as the number of samples goes to infinity, this guarantees that the estimator will converge to the true expectation.

The standard Monte Carlo estimator draws samples with uniform probability from the integration domain. This means that when the function being integrated has small volume and exists on a large, potentially unbounded, domain a vast majority of the samples computed will have little to no contribution to the final expectation. In this section we have studied several strategies for overcoming such issues, with particular focus given to MCMC methods such as MHMC which iteratively update the position of a random sample by drawing new proposal samples from a conditional distribution centred around the previous sample. This allows the sampler to randomly “walk” the integration space, often being able to visit regions of the function space which were extremely unlikely to be sampled but can potentially have a large contribution to the final expectation. HMC presents an alternate sampling strategy adopted from the physics literature which treats the probability distribution being sampled as an energy potential and the head of the sampler as a frictionless puck riding on the surface of that energy potential. For each subsequent sample, the puck is simulated as it slides across the function space, spending proportionally more of its time in regions of high probability (low potential energy). HMC computes well distributed and low variance samples at the cost of computing a first order derivative of the target function multiple times per update. In certain domains computing the derivative of the target function is prohibitively expensive, making this method infeasible.

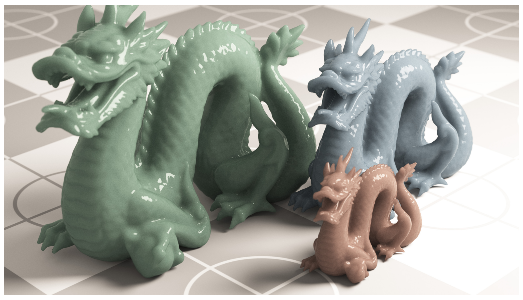
Due to the way in which sample proposals are stochastically accepted or rejected by an acceptance probability in MCMC methods, it is possible for samples which are expensive to draw from the proposal distribution (path-space samples drawn iteratively via ray tracing) to be discarded due to the random chance of not being accepted. Wasting important information about the function estimate despite the computational expense used to draw the sample. A powerful class of techniques for improving sampling efficiency comes in the form of IS and the more generalized MIS. These methods provide a solution to the problem of throwing away computation by considering the contribution of every sample that is drawn and combining these contributions in a non-uniform weighted average based on the ratio between the proposal distribution to the target distribution.

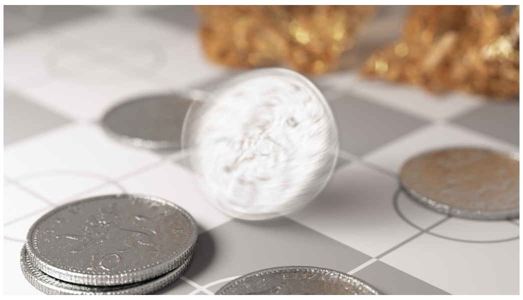
In the following section we will introduce physically based rendering and see how Bayesian statistics and Monte Carlo sampling techniques are used in numerous places to approximate the expectation of complex high-dimensional integrals modelling the way in which light interacts with simulated environments.

2.3 Physically Based Rendering

Physically based rendering algorithms allow for a plethora of photo-realistic lighting phenomena to be simulated such as indirect illumination, depth of field, participating media, caustics, and physically based materials (figure 2.6). These algorithms can be used to create synthetic images which exhibit a high degree of visual fidelity and can often be indistinguishable from natural images (photographs) when shown to human observers. Due to the complex nature of light interactions with different materials and the high volume of light particles (photons) which are present within a real world environment, the number of calculations we need to consider when simulating the light interactions in even a simple scenario can quickly become intractable as we consider longer and more complex chains of interactions. To handle this complex simulation domain stochastic methods are commonly used which can decompose an intractable problem into a series of tractable sub-problems. The equilibrium or expectation of these sub-problems can be used to coarsely approximate the original intractable problem, and the error of the approximation can decrease as more sub-problems are considered and allowed to contribute to the approximation.

(a)


(b)


(c)

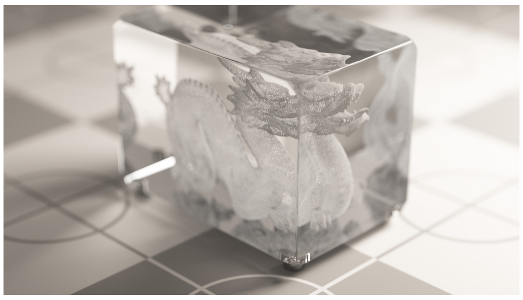

(d)

Fig. 2.6.: A selection of physically based rendered images created with software developed for our research. (a) Geometric instancing is used to duplicate a high-polygon Stanford Bunny mesh without additional memory requirements. The bunnies are rendered with a glass material leading to caustic illumination patters on the floor beneath them. (b) A sub-surface-scattering model is used to simulate a rubbery material on instanced dragon meshes. (c) Temporal lens sampling is used to simulate a finite length shutter-speed in front of a spinning coin. (d) Diffuse transmission and scattering is used to simulate a dragon laser etched inside of a crystal.

The Rendering Equation

The rendering equation was originally presented by Kajiya [Kaj86] and built off previous work on radiance transport from Whitted [Whi80], Cook [Coo+84], and

Goral [Gor+84]. The equation defines the amount of energy leaving a point x laying on a surface within an environment in direction ω_o as the summation of light emitted by the surface at x in direction ω_o, and the integral of all light reflected or transmitted from x in direction ω_o incident from all directions around x, (equation 2.25). The equation is additionally parameterized over a light wavelength λ and a time value t. In practice the wavelength λ is usually replaced with a wavelength distribution (spectrum) or a finite tuple over important wavelengths (RGB triple).

$$\begin{aligned} \mathcal{L}_o\left(x, \omega_o, \lambda, t\right) = {} & \mathcal{L}_e\left(x, \omega_o, \lambda, t\right) \\ & + \int_\Omega f\left(x, \omega_i, \omega_o, \lambda, t\right) \mathcal{L}_i\left(x, \omega_i, \lambda, t\right) \left(\omega_i \bullet n\right) d\omega_i \\ \text{where} \quad & \mathcal{L}_i\left(x, \omega_i, \lambda, t\right) = \mathcal{L}_o\left(x', -\omega_i, \lambda, t\right) \end{aligned} \tag{2.25}$$

In the integral over possible incoming directions the function $\mathcal{L}_i\left(x, \omega_i, \lambda, t\right)$ represents the amount of light incident on point x from direction ω_i. This function can be re-parameterized as the amount of light leaving the point x', visible from x in direction ω_i, in the direction towards x. Through this re-parameterization a recursive relationship presents itself. We can see that to integrate the equation from point x we will look in all surrounding directions and traverse to the point x' visible in each direction. From the new points we will look in all directions surrounding them and again traverse to each one in turn yielding point x'', and so on. This formulation of the equation is both infinitely recursive and infinitely branching at each level of the recursion making standard integration techniques computationally intractable as the depth of the recursion increases. If we sample a single direction to traverse from each point and recur for a random number of levels the sequence of points generated can be seen as the vertices of a random path drawn from the sampling domain over all possible paths of all possible lengths within the environment. We refer to this domain as the path-space of the environment.

Path vertices can be sampled on the surface of sensing elements such as virtual cameras, emitting elements such as light sources, and scattering elements representing solid surfaces. Scattering elements are defined by their Bidirectional Scattering Distribution Function (BSDF) function $f\left(x, \omega_i, \omega_o, \lambda, t\right)$ which defines the proportion of light leaving in direction ω_o which was incident to x from direction ω_i. Emitting elements are primarily defined by their emission distribution function $\mathcal{L}_e\left(x, \omega_o, \lambda, t\right)$ which represents the amount of light they emit in direction ω_o from point x. A scattering element which does not emit light would define its emission distribution function as $\mathcal{L}_e\left(x, \omega_o, \lambda, t\right) = 0$ and similarly an emitting element which does not scatter incoming light would define $f\left(x, \omega_i, \omega_o, \lambda, t\right) = 0$. In this formulation, allowing x to vary over the union of all surface elements will naturally account for the differences between emitting and scattering materials in the environment. Lastly, the $\left(\omega_i \bullet n\right)$ term in the integral part of the equation accounts for the decreased probability of light being scattered or emitted from a surface in a direction approaching a glancing angle [Lam92].

2.3.1 Bidirectional Scattering Distribution Function (BSDF)

The BSDF of a surface represents the proportion of light that is scattered out from a point for a given incident direction. The BSDF is the general form of the Bidirectional Reflectance Distribution Function (BRDF) which represents the proportion of light reflected back from a surface and the Bidirectional Transmittance Distribution Function (BTDF) which represents the proportion of light transmitted through a surface from one medium to another. For opaque surfaces the BRDF is sufficient to model the material as light cannot penetrate beneath the surface of the object. For translucent materials such as dielectrics like glass and various plastics both the BRDF and BTDF must be considered to correctly model the surface as light can be reflected off either side of the object boundary or transmitted through the boundary.

Lambert BRDF

A simple BRDF model which is commonly used in physically based rendering is the Lambertian BRDF. This model simulates the interaction of light with a matte, perfectly diffuse surface which scatters light incident from any direction to any outgoing direction.

In nature there are no known examples of a perfectly diffuse surface [Hab10], though synthetic materials such as Spectralon [AES93] have been developed which can approach $\geq 99\%$ of the diffuse reflectivity expected from a perfectly diffuse surface. Unless viewed from near glancing angles the Lambertian model is, in general, a good enough approximation of many real world matte materials (such as paper, unglazed ceramics, and wall paint) that it can be used in the creation of high quality synthetic imagery without causing visually implausible artefacts to be introduced.

The BRDF of the Lambertian model is defined uniformly for all pairs of incoming and outgoing directions as $\frac{K_d}{\pi}$, where K_d is the diffuse albedo (reflectant colour) of the surface and π is the projected area of the hemisphere above the path vertex x. While the amount of light reflected by the surface is uniform for all directions the probability of a photon being scattered in a given direction is not uniform. Specifically the model scatters light with a smaller probability towards glancing angles following Lambert's Cosine Law [Lam92] which states that the observed light coming from a point on a perfectly diffuse surface is proportional to the cosine of the angle between the surface normal and observer. Equation 2.26 defines the BRDF and PDF for the Lambertian model in terms of a path vertex x_i we wish to shade given the previous path vertex x_{i-1} and subsequent path vertex x_{i+1}. The PDF of the model is a conditional distribution $\mathcal{P}(x_i \to x_{i+1}|x_{i-1} \to x_i)$ representing the probability of scattering from x_i to x_{i+1} given that we previously scattered to x_i from x_{i-1}.

$$\begin{aligned} f(x_{i-1} \to x_i \to x_{i+1}) &= \frac{K_d}{\pi} \\ \mathcal{P}(x_i \to x_{i+1}|x_{i-1} \to x_i) &= \frac{\cos\theta_{+i}}{\pi} \\ \text{where} \quad \omega_{+i} &= \|x_i \to x_{i+1}\| \\ \cos\theta_{+i} &= |\omega_{+i} \bullet n_i| \end{aligned} \tag{2.26}$$

Because the right-hand side of the PDF does not reference the previous vertex x_{i-1} the definition of the Lambertian PDF can reduce to a non-conditional distribution $\mathcal{P}(x_i \to x_{i+1}) = \mathcal{P}(x_i \to x_{i+1}|x_{i-1} \to x_i)$, and similarly the definition of the BRDF can reduce to $f(x_i) = f(x_{i-1} \to x_i \to x_{i+1})$. However, for the sake of consistency when considering other more advanced BRDF models we will keep the notation of the conditional distribution throughout.

Lastly, we need a way to draw samples proportionally to the PDF of the model for use when sampling paths through the environment. Here we adopt the notation ω_{+i} to refer to the direction from vertex x_i to x_{i+1} in world space and similarly $\omega_{+i}^{\text{local}}$ to refer to that direction in the local coordinate system around vertex x_i which is transformed so that the surface normal n_i is aligned to the y-axis. Moving the evaluation and sampling of BRDF and PDF values into the local coordinate system allows for shading computations to be greatly simplified. Equation 2.27 shows the process of drawing a random direction proportionally to the PDF. To sample a direction from the PDF in the local coordinate system we draw two uniform random values ξ_0 and ξ_1 in the range $[0, 1)$ and warp them through a polar-coordinate transformation onto the surface of a cosine weighted hemisphere aligned with the y-axis as the up-vector. The resulting directional vector $\omega_{+i}^{\text{local}}$ can then be transformed from the local coordinate system at x_i to world coordinates yielding ω_{+i}. The next path vertex x_{i+1} can then be found by tracing a ray from x_i in direction ω_{+i} to the first intersection of the ray with a surface in the environment.

$$\begin{aligned} \omega_{+i} &\propto \mathcal{P}(\cdot|x_{i-1} \to x_i) \\ \\ \omega_{+i}^{\text{local}} &= \{v * \sin(u), \quad \sqrt{\xi_1}, \quad v * \cos(u)\} \\ \omega_{+i} &= \text{LocalToWorld}\left(\omega_{+i}^{\text{local}}, \quad x_i\right) \\ \\ \text{where} \quad u = 2\pi\xi_0 \quad v &= \sqrt{1-\xi_1} \quad \text{and} \quad \xi_0, \xi_1 \propto \mathcal{U}(0,1) \end{aligned} \tag{2.27}$$

2.3.2 Light Emitters

As we have previously shown, the rendering equation is defined as the summation of light emitted from a point in a direction towards the observer and the integral over the proportion of light reflected towards the observer, incident to the point from all directions. Having handled the latter case through the definition of the BRDF and the re-parameterization of $\mathcal{L}_i\left(x, \omega_i, \lambda, t\right)$ as an instance of $\mathcal{L}_o\left(x', -\omega_i, \lambda, t\right)$ exposing the recursive nature of the equation, we now turn our attention to the definition of the light emitted from a point towards the observer, $\mathcal{L}_e\left(x, \omega_o, \lambda, t\right)$.

Diffuse Area Light

A light emitting surface, also called an area light, is one of the most simple light sources to consider when implementing a physically based light simulation. An area light can emit energy from any point on its surface and the direction of the emitted energy can be modelled by an emission distribution function over the outgoing directions on the hemisphere around the point. When we consider the scenario of a light source which can emit light in any direction over the hemisphere orientated to the normal of the surface we have a diffuse area light as modelled in equation 2.28. For a point sampled on the surface of the light source y_0 we define $\mathcal{L}_e^0(y_0) = K_e$ as the amount of energy emitted by the light source at that location, where K_e is an intensity or distribution of intensities related to the spectrum of light frequencies being accumulated in the simulation. This value includes the integral of energy emitted in all directions over the hemisphere orientated to the emitting surface. To determine the amount of energy leaving the surface in a specific direction we define the emission distribution function at y_0 as $\mathcal{L}_e^1(y_0 \rightarrow y_1) = \frac{1}{\pi}$. This is similar in definition to the Lambertian BRDF discussed in section 2.3.1 with the minor modification that the numerator is now set to one so that the emission distribution function does not tint the light emitted from the light source.

$$\begin{aligned}
\mathcal{L}_e^0(y_0) &= K_e \\
\mathcal{P}_A(y_0) &= \frac{1}{\text{Area}} \\
\\
\mathcal{L}_e^1(y_0 \rightarrow y_1) &= \frac{1}{\pi} \\
\mathcal{P}(y_0 \rightarrow y_1 | y_0) &= \frac{\cos\theta_{+0}}{\pi} \\
\\
\text{where} \quad \omega_{+0} &= \|y_0 \rightarrow y_1\| \\
\cos\theta_{+0} &= |\omega_{+0} \bullet n_i|
\end{aligned} \tag{2.28}$$

As with the throughput of the light source we also split the computation of the PDF into two parts to aid in its implementation. For the point y_0 on the surface of the light source we can compute the probability of uniformly sampling that point $\mathcal{P}_A(y_0) = \frac{1}{\text{Area}}$, as simply being one over the surface area of the light source. For the conditional directional probability we borrow the PDF definition from the Lambertian model giving $\mathcal{P}(y_0 \rightarrow y_1 | y_0) = \frac{\cos\theta_{+0}}{\pi}$ where $\cos\theta_{+0}$ is the cosine of the angle between the surface normal at y_0 and the outgoing direction. With the PDF of the emission matching that of the Lambertian PDF we can reuse the method for sampling directions proportionally to the emission PDF by drawing samples from the cosine weighted hemisphere orientated to the surface normal at y_0.

2.3.3 Camera Models

While it is not explicitly shown in the definition of the rendering equation we also need to define a model of the camera being used during image synthesis. The camera model can be seen as the inverse of an emitting surface, and represents the proportion of light incoming from each direction that will contribute to the image estimate. Different camera models can be used to achieve varying visual aesthetics by modelling physically plausible distortions such as depth of field where image regions become out-of-focus and blurred, chromatic aberration where different wavelengths of light diverge while refracting through lens elements within the camera resulting in visible colour banding, and field of view distortions like those exhibited by fish-eye, wide-angle, or telephoto lenses.

Pin-hole Camera

A simple but physically plausible camera model is the pin-hole camera. This model gets its name from the idea of using a camera aperture with infinitesimally small area to project incident light onto a virtual film plane. Because the aperture is infinitesimally small, distortions such as depth of field cannot be captured in the resulting images. As with the above definition of a diffuse area light we split the evaluation and sampling of the camera model into spatial and directional components to make the model easier to implement and integrate into rendering algorithms (equation 2.29). Sampling a point on the aperture of the camera z_0 we can see that the single point must be able to absorb all incoming light $\mathcal{W}_e^0(z_0) = 1$ given that the aperture is infinitesimally small and all of the light will therefore be incident on z_0. For the directional component, modelling light incident on z_0 coming from the next path vertex z_1 we can define the proportion of light reaching z_0 from this direction as the area of a single pixel in a $W \times H$ resolution image, $|D| = \frac{1}{WH}$, if the direction passes through a point on the image plane and a value of zero otherwise. The probability distributions for the model are defined similarly and for this simplified camera model are identical to the equations for the throughput.

$$\mathcal{W}_e^0(z_0) = 1$$
$$\mathcal{P}_A(z_0) = 1$$

$$\mathcal{W}_e^1(z_0 \rightarrow z_1) = \begin{cases} |D| & \text{if } z_0 \rightarrow z_1 \text{ hits the image plane} \\ 0 & \text{else} \end{cases} \tag{2.29}$$
$$\mathcal{P}(z_0 \rightarrow z_1 | z_0) = \begin{cases} |D| & \text{if } z_0 \rightarrow z_1 \text{ hits the image plane} \\ 0 & \text{else} \end{cases}$$

2.3.4 Path Tracing (PT)

To evaluate Kajiya's rendering equation [Kaj86] we need a sampling strategy that can draw samples from an infinitely recursive formula which branches infinitely at each level of the recursion. Path Tracing, also called forwards ray-tracing, is a method of approximating the rendering equation while limiting the exponential explosion of computational complexity that occurs as paths become longer. Applying the standard Monte Carlo estimator (equation 2.5) to the inner integral of the rendering equation we see that the luminance incoming from the hemisphere over the point x can be coarsely approximated using a fixed number of rays sampled proportionally to the PDF of the BRDF at that point. This coarse approximation will only converge as the number of samples approaches infinity, however, it can be shown that the approximation will also converge if a single branch is followed at each level in the recursion and the entire path is re-sampled repeatedly. Through this transformation the complexity of evaluating the contribution of paths to the integral scales linearly by their length rather than exponentially in the original formulation.

A random path $\bar{z}$ is sampled from the path-space by first sampling a point of the camera lens z_0 with probability $\mathcal{P}_A(z_0)$. The first point in the scene z_1 is sampled by drawing a direction proportional to the PDF at vertex z_0, $\omega_{+0} \propto \mathcal{P}(\cdot|z_0)$, and tracing a ray from z_0 in direction ω_{+0} to find the next intersection z_1. From this point we can iteratively sample the path to generate the $(i+1)^{th}$ vertex by drawing a direction $\omega_{+i} \propto \mathcal{P}(\cdot|z_{i-1} \rightarrow z_i)$ from the BRDF around vertex z_i given the previous vertex z_{i-1} and tracing from z_i in direction ω_{+i} to find the next intersection z_{i+1}.

Paths can sampled to an arbitrary fixed length, however, this introduces bias to the estimate as paths longer than the arbitrary length can never contribute to the image. An alternative strategy is to halt path generation by Russian Roulette sampling. At each vertex we randomly choose to stop tracing with probability q_i (equation 2.30). If the path is continued we modify our definition of the probability for the event to account for the random choice that was made, yielding $\mathcal{P}(z_i \rightarrow z_{i+1} | z_{i-1} \rightarrow z_i) = q_i \mathcal{P}^*(z_i \rightarrow z_{i+1} | z_{i-1} \rightarrow z_i)$.

$$q_i = \min\left(1, \frac{f_t(z_{i-1} \rightarrow z_i \rightarrow z_{i+1})}{\mathcal{P}(z_i \rightarrow z_{i+1} | z_{i-1} \rightarrow z_i)}\right) \tag{2.30}$$

At this point we have sampled a path $\bar{z} = z_0, \cdots, z_{t-1}$ of t vertices starting at the camera lens and we need to define a method for evaluating the contribution of light that occurs along this path. Because we are going to re-sample the entire path multiple times to build our estimate of the final contribution we can define the expectation for the j^{th} pixel of the image as being a standard Monte Carlo estimator over paths sampled going through the j^{th} pixel (equation 2.31).

$$I_j^E = \frac{1}{N_j} \sum_{i=0}^{N_j} F_j^E \tag{2.31}$$

For a given full path, the contribution to the image is a summation over the contributions of the sub-paths of each length. We define this path contribution in equation 2.32 where we split the contribution of each sub-path into a precomputed value α_t^E which represents the contribution of the path before vertex z_{t-1} and an interaction contribution c_t which represents the unique part of the sampling strategy for the t^{th} sub-path.

$$\begin{aligned} F^E &= \sum_{t \geq 0} C_t \\ \text{where} \quad C_t &= c_t \alpha_t^E \end{aligned} \tag{2.32}$$

The reason we split the contribution of each sub-path is to leverage the fact that the portion of the sub-path α_t^E is dependent on the value of the previous sub-path α_{t-1}^E. This allows us to compute all the α^E values in a single forward pass along the path without having to recompute previous values for each sub-path. Equation 2.33 shows the iterative method of computing the sub-path contributions.

$$\begin{aligned} \alpha_0^E &= 1 \\ \alpha_1^E &= \frac{\mathcal{W}_e^0(z_0)}{\mathcal{P}_A(z_0)} \\ \alpha_2^E &= \alpha_1^E \frac{\mathcal{W}_e^1(z_0 \rightarrow z_1)}{\mathcal{P}(z_0 \rightarrow z_1 | z_0)} \\ \alpha_i^E &= \alpha_{i-1}^E \frac{f_t(z_{i-3} \rightarrow z_{i-2} \rightarrow z_{i-1})}{\mathcal{P}(z_{i-2} \rightarrow z_{i-1} | z_{i-3} \rightarrow z_{i-2})} \end{aligned} \tag{2.33}$$

With the sub-path contributions up to the last vertex in the sub-path computed, the only thing left is to construct full sub-paths by combining the precomputed contributions with the contributions of the interaction with the last vertex in the sub-path. For standard forward path tracing we only consider one sampling strategy, where the path randomly intersects a material which emits light ($\mathcal{L}_e\,(x, \omega_o, \lambda, t)$ in the original rendering equation). To evaluate this we simply take the light emitted by the sub-path end vertex z_{t-1} modelled by $\mathcal{L}_e^0$, and scale it by the proportion of that light that is emitted in the direction from $z_{t-1} \rightarrow z_{t-2}$ modelled by the emission distribution function $\mathcal{L}_e^1$ (equation 2.34). If the vertex z_{t-1} does not lie on a light emitting entity then the contribution of the sub-path is naturally zero, and the computation of the PDF associated with $\mathcal{L}_e^1$, $\mathcal{P}(z_{t-1} \rightarrow z_{t-2}|z_{t-1})$, is also zero.

$$c_t = \mathcal{L}_e^0(z_{t-1})\mathcal{L}_e^1(z_{t-1} \rightarrow z_{t-2}) \tag{2.34}$$

2.3.4.1. Direct Lighting

A shortcoming of the standard path tracing algorithm is that it requires that paths traced starting from the camera randomly intersect with light emitting surfaces within the scene. When light sources are small, distant, or heavily occluded such random intersections will occur with low probability meaning that the vast majority of paths computed will contribute no energy to the image. Another scenario where this can occur is when light sources are represented without using geometry such as a point- or directional-light source. In these cases it is impossible for a path generated from the camera to randomly intersect the light source.

A strategy to improve rendering performance in scenes where this is the case is to explicitly sample a point on a known light emitting entity in the scene and to compensate for having explicitly forced the connection to the sampled point [Shi+96]. To allow for this we introduce two helper functions which will be used repeatedly in the process of combining paths sampled through different strategies. Equation 2.35 shows the definition of function $\mathcal{V}(y \Leftrightarrow z)$ which gives a binary value denoting if there is an occluding object between path vertices y and z, and function $\mathcal{G}(y \Leftrightarrow z)$ which is referred to as the geometric attenuation function. This function measures the projected area from one vertex to another over the hemispheres of both vertices which are assumed to both lie on locally flat surfaces with normal vectors n_y and n_z respectively.

$$\begin{aligned}
\mathcal{V}(y \Leftrightarrow z) &= \begin{cases} 1 & \text{if the path from } y \text{ to } z \text{ is unobstructed} \\ 0 & \text{else} \end{cases} \\
\mathcal{G}(y \Leftrightarrow z) &= \mathcal{V}(y \Leftrightarrow z)\frac{|n_y \bullet v||n_z \bullet v|}{|v|} \\
&\text{where} \quad v = y \rightarrow z
\end{aligned} \tag{2.35}$$

Previously our path tracing definition only considered paths with vertices sampled from the eye path, which is equivalent to saying paths which contain some number of eye vertices and zero light vertices. However, with direct light sampling we also must consider paths which contain one vertex on the light path. Equation 2.36 shows the updated version of the path contribution formula where we now also sum over the combinations of zero or one light path vertices denoted by subscript s. The sub-path contribution C_t has also now been modified to $w_{s,t}C^*_{s,t}$ where $C^*_{s,t}$ is the un-weighted contribution of a path with s light path vertices and t eye path vertices, and $w_{s,t}$ is a weighting function satisfying the constraints from equation 2.18 from the section 2.2.4 on MIS. Computation of the weighting function will be covered more in-depth in section 2.3.6.1 but for now it suffices to say that the weighting function must sum to one over all combinations of sub-path lengths s and t which sum to the same path length. That is, our final estimator is over samples which are the sum of the weighted averages of paths of each length.

$$\begin{aligned} F^E &= \sum_{1\geq s\geq 0}\sum_{t\geq 0} w_{s,t}C^*_{s,t} \\ \text{where} \quad C^*_{s,t} &= \alpha^L_s c_{s,t} \alpha^E_t \end{aligned} \tag{2.36}$$

The un-weighted contribution $C^*_{s,t}$ is split into several parts so that we can leverage the fact that parts of the contribution are shared between paths of different lengths. As before we have the eye path contribution up to the t^{th} vertex on the eye path α^E_t, the contribution of the combination event $c_{s,t}$, and now we also have the contribution up to the s^{th} vertex on the light path α^L_s (equation 2.37).

$$\begin{aligned} \alpha^L_0 &= 1 \\ \alpha^L_1 &= \frac{\mathcal{L}^0_e(y_0)}{\mathcal{P}_A(y_0)} \\ \\ \alpha^E_0 &= 1 \\ \alpha^E_1 &= \frac{\mathcal{W}^0_e(z_0)}{\mathcal{P}_A(z_0)} \\ \alpha^E_2 &= \alpha^E_1 \frac{\mathcal{W}^1_e(z_0 \to z_1)}{\mathcal{P}(z_0 \to z_1 | z_0)} \\ \alpha^E_i &= \alpha^E_{i-1} \frac{f_t(z_{i-3} \to z_{i-2} \to z_{i-1})}{\mathcal{P}(z_{i-2} \to z_{i-1} | z_{i-3} \to z_{i-2})} \end{aligned} \tag{2.37}$$

Finally the contribution of the full sub-path can be computed by calculating the event at $c_{s,t}$. As in standard path tracing we have the event $c_{0,t}$ which is computed identically to event c_t in the original algorithm and computes the contribution of the eye path randomly intersecting a light emitting entity. With the direct lighting calculation we now also consider $c_{1,t}$ which is the contribution of the sampled point on the light source y_0 shining on point z_{t-1}. Because the emittance of point y_0 is calculated in α_1^L the contribution of $c_{1,t}$ starts with the directional throughput $\mathcal{L}_e^1$ between y_0 and z_{t-1}. This is scaled by the value of the BRDF at z_{t-1} coming from z_{t-2} and going towards y_0, and again scaled by the geometric attenuation function $\mathcal{G}$ to account for the mutual visibility between vertices z_{t-1} and y_0 (equation 2.38).

$$\begin{aligned} c_{0,t} &= \mathcal{L}_e^0(z_{t-1})\mathcal{L}_e^1(z_{t-1} \to z_{t-2}) \\ c_{1,t} &= \mathcal{L}_e^1(y_0 \to z_{t-1})f_t(y_0 \to z_{t-1} \to z_{t-2})\mathcal{G}(y_0 \Leftrightarrow z_{t-1}) \end{aligned} \tag{2.38}$$

2.3.5 Light Tracing (LT)

Even with the addition of direct lighting the standard forwards path tracing algorithm can often have difficulties sampling paths where there is heavily occlusion of emitters or when hard to sample effects such as caustic illumination are dominant in the converged image. The backwards ray tracing algorithm [AC86] is able to sample caustic illumination paths efficiently by generating multiple vertices on the light path and relating them back to the image plane to accumulate their contribution.

A random path $\bar{y}$ is sampled from the path-space by first sampling a point of a light source y_0 with probability $\mathcal{P}_A(y_0)$. The first point in the scene y_1 is sampled by drawing a direction proportionally to the PDF at vertex y_0, $\omega_{+0} \propto \mathcal{P}(\cdot|y_0)$, and tracing a ray from y_0 in direction ω_{+0} to find the next intersection y_1. From this point we can iteratively sample the path to generate the $(i+1)^{th}$ vertex by drawing a direction $\omega_{+i} \propto \mathcal{P}(\cdot|y_{i-1} \to y_i)$ from the BRDF around vertex y_i given the previous vertex y_{i-1} and tracing from y_i in direction ω_{+i} to find the next intersection y_{i+1}.

A key difference between the path tracing and light tracing algorithms is in the method by which sample contributions are accumulated. In path tracing, paths always contained at least two vertices on the eye path, meaning for a given path we always knew which pixel in the image that path contributed to. With light tracing, however, this is not the case. As the path is sampled outwards from the light source each bounce has the ability to contribute to a different pixel as it can lie anywhere on the projected image plane. To account for this the accumulation method used in light tracing forms a sum over all samples which eventually reach the j^{th} pixel of the image. This sum is scaled by the projected area of the pixel on the image plane $|D|$ divided by the total number of paths sampled from the light source N (equation 2.39).

$$I_j^L = (|D|/N)\sum F_j^L \tag{2.39}$$

The contribution of a path is now computed as a sum over the contributions C_s of possible sub-paths generated from the light source. As before, the evaluation of C_s is split into a precomputed component α_s^L and a contribution event c_s (equation 2.40).

$$\begin{aligned} F^L &= \sum_{s \geq 0} C_s \\ \text{where} \quad C_s &= \alpha_s^L c_s \end{aligned} \tag{2.40}$$

In the previous section on direct lighting (section 2.3.4.1) we defined the first two terms in α_s^L for use when the light path contained zero or one vertices. For the full light tracing algorithm we continue to define the remaining α_s^L for paths of two vertices or longer. A key difference is in the computation of the BRDF. When computing the eye path we used the version of the BRDF f_t and now in light tracing we use the notation f_s to denote the subtle difference between eye and light paths. For most BRDF models $f_t(x_{i-1} \rightarrow x_i \rightarrow x_{i+1}) = f_s(x_{i+1} \rightarrow x_i \rightarrow x_{i-1})$, however, in certain scenarios such as when dealing with refractive surfaces there is a small difference in light transmission between paths tracing forwards and backwards through material. To account for this the definition of f_s for non symmetric BRDF's contains a small correction term which is not present in f_t.

$$\begin{aligned} \alpha_0^L &= 1 \\ \alpha_1^L &= \frac{\mathcal{L}_e^0(y_0)}{\mathcal{P}_A(y_0)} \\ \alpha_2^L &= \alpha_1^L \frac{\mathcal{L}_e^1(y_0 \rightarrow y_1)}{\mathcal{P}(y_0 \rightarrow y_1 | y_0)} \\ \alpha_i^L &= \alpha_{i-1}^L \frac{f_s(y_{i-3} \rightarrow y_{i-2} \rightarrow y_{i-1})}{\mathcal{P}(y_{i-2} \rightarrow y_{i-1} | y_{i-3} \rightarrow y_{i-2})} \end{aligned} \tag{2.41}$$

Finally we need to calculate the contribution of a sub-path to the image. In standard forward path tracing this was done by assuming the last vertex on the sub-path was on a light emitting entity, and calculating the energy output from the last vertex in the direction of the preceding one. For light tracing we want to compute the opposing connection, by assuming that the last vertex on the sub-path y_{s-1} lies on the camera lens, and that light incoming to the lens from the direction of the preceding vertex y_{s-2} contributes to the image at a given pixel. This is done through the spatial and directional components of the camera model, $\mathcal{W}_e^0$ and $\mathcal{W}_e^1$ respectively, which we saw used in the computation of α_1^E and α_1^E in the path tracing algorithm.

$$c_s = \mathcal{W}_e^0(y_{s-1}) \mathcal{W}_e^1(y_{s-2} \rightarrow y_{s-1}) \tag{2.42}$$

2.3.5.1. Direct Pixel Contribution

As with forwards path tracing there is an inherent shortcoming of standard light tracing which makes it difficult to utilise for a variety of scene compositions. In light tracing, paths generated from the light source must randomly intersect with the camera lens in order to contribute their energy to the estimation of the image. This requires that the camera lens is modelled as a geometric component of the scene. In reality, camera lenses are often small compared to the size of the environment or subject being captured meaning a similarly sized geometric representation of a camera lens is unlikely to be randomly intersected during path generation. Further, only paths which intersect the lens from a narrow cone of directions will find their way through the lens to the image sensor, making the chance of contributing to the image even more unlikely. Often in computer graphics a much simpler camera model such as a pin-hole camera is employed as it is much simpler to compute and in many cases has little effect on the final image. Like the point- or directional-light sources discussed in the section on direct lighting (section 2.3.4.1) a pin-hole camera has no geometric representation and therefore cannot be used with the standard light tracing algorithm.

To improve sampling performance on hard to sample paths including small lens elements or pin-hole cameras we can extend the standard light tracing algorithm in an analogous manner to the addition of direct lighting to standard path tracing. We do this by explicitly sampling a point z_0 on the camera lens and computing the contribution of paths composed of zero or one eye path vertices and some number of light path vertices. We start by updating the equation for the full contribution of a path to be the summation over an arbitrary number of light path vertices and zero or one eye path vertices (equation 2.43). The sub-path contribution C_s is swapped for $w_{s,t}C^*_{s,t}$ where $C^*_{s,t}$ is the un-weighted contribution of the composed sub-path and $w_{s,t}$ is a weighting function like the one used for direct lighting (section 2.3.4.1) and satisfying the constraints from equation 2.18 from the section 2.2.4 on MIS. The un-weighted contribution $C^*_{s,t}$ is composed of the precomputed α^L_s from the light path vertices and α^E_t from the eye path vertex, and the combined contribution given by $c_{s,t}$.

$$\begin{aligned} F^L &= \sum_{s\geq 0} \sum_{1\geq t\geq 0} w_{s,t}C^*_{s,t} \\ \text{where} \quad C^*_{s,t} &= \alpha^L_s c_{s,t} \alpha^E_t \end{aligned} \tag{2.43}$$

Borrowing from the definition for standard path tracing we define α_0^E and α_1^E for the vertex sampled explicitly on the camera lens (equation 2.44).

$$\begin{aligned}
\alpha_0^L &= 1 \\
\alpha_1^L &= \frac{\mathcal{L}_e^0(y_0)}{\mathcal{P}_A(y_0)} \\
\alpha_2^L &= \alpha_1^L \frac{\mathcal{L}_e^1(y_0 \to y_1)}{\mathcal{P}(y_0 \to y_1 | y_0)} \\
\alpha_i^L &= \alpha_{i-1}^L \frac{f_s(y_{i-3} \to y_{i-2} \to y_{i-1})}{\mathcal{P}(y_{i-2} \to y_{i-1} | y_{i-3} \to y_{i-2})}
\end{aligned} \tag{2.44}$$

$$\begin{aligned}
\alpha_0^E &= 1 \\
\alpha_1^E &= \frac{\mathcal{W}_e^0(z_0)}{\mathcal{P}_A(z_0)}
\end{aligned}$$

Lastly we define the new case for the combined contribution of a path with one eye path vertex. This is defined similarly to the combined contribution for direct lighting. The spatial component of sampling the point on the lens $\mathcal{W}_e^0$ is computed as part of the computation of α_1^E, meaning the combined contribution can start with the computation of the directional component $\mathcal{W}_e^1$ from the sampled lens vertex z_0 to the point at the end of the light sub-path y_{s-1}. This is scaled by the BRDF at the end of the light sub-path coming from the preceding vertex y_{s-2} and going towards the sampled lens vertex z_0, and again scaled by the geometric attenuation function between the connected vertices.

$$\begin{aligned}
c_{s,0} &= \mathcal{W}_e^0(y_{s-1})\mathcal{W}_e^1(y_{s-2} \to y_{s-1}) \\
c_{s,1} &= \mathcal{W}_e^1(z_0 \to y_{s-1}) f_s(y_{s-2} \to y_{s-1} \to z_0)\mathcal{G}(y_{s-1} \Leftrightarrow z_0)
\end{aligned} \tag{2.45}$$

2.3.6 Bidirectional Path Tracing (BDPT)

While light tracing effectively solves the problem of hard-to-intersect light sources and difficult-to-sample caustic paths it has a number of shortcomings which severely limit its applicability in a large number of scenarios. One main limitation is the inability to allow path contributions from mirror- or glass-like surfaces which are directly visible in the image. In order for such events to contribute to the image estimate a light path vertex on a diffuse surface must scatter in a randomly sampled direction before going through one or more specular bounces consisting of reflections and refractions, finally intersecting the camera lens in a direction which contributes to the image estimate. Because specular surfaces form Dirac distributions (equation 2.46) where the probability of the event equals 1 if and only if the outgoing direction is a perfect reflection or refraction from the incoming direction and 0 everywhere else, this means direct pixel contribution methods as described in section 2.3.5.1 cannot be applied as the probability of the forced connection will be zero.

$$\mathcal{P}^{\delta}(x_i \to x_{i+1} | x_{i-1} \to x_i) = \begin{cases} 1 & \text{if } x_{i-1} \to x_i \to x_{i+1} \text{ forms} \\ & \text{ a perfect specular reflection around } n_i \\ 0 & \text{else} \end{cases} \quad (2.46)$$

For the same reason, forwards path tracing struggles to account for the contribution of caustic paths, which result from a path vertex on a diffuse surface scattering through one or more specular bounces before intersecting a light source. For small light emitting elements the probability of randomly intersecting the light source is prohibitively small, and for non-geometric emitters such as point- or directional-light sources the probability is zero.

Another limitation is shared by both the path tracing and light tracing algorithms. In the same manner an eye path may struggle to explore beyond a large occluding body in front of a light source, so too will light tracing struggle to explore beyond a large occluding body in front of the camera. Even with direct lighting and direct pixel contributions considered the probability of generating paths connected to the light sources or the camera lens becomes prohibitively low when using these algorithms in complex scenes.

To account for the limitations of both approaches bidirectional path tracing samples multiple vertices on both the eye and light paths and considers all possible paths that can be constructed using sub-paths of varying lengths. First proposed by Lafortune *et. al.* [LW93], bidirectional path tracing allows for the efficient generation of complex paths and has given a greatly improved our ability to render complex scenes. Bidirectional path tracing was later formalized by Veach *et. al.* [Vea97] who developed MIS as a way to compute near optimum weights for combining the contributions of sub-paths.

Because we will now consider paths which can contribute to a specific pixel or to any pixel, based on how they are sampled, we split the accumulation of path contributions into two separate images to ease the implementation. Paths with two or more vertices on the eye path are accumulated into an eye image as in forward path tracing, and those with zero or one vertex on the eye path are accumulated and scaled into a light image as in backward light tracing. The final bidirectional image is computed as the sum of the eye and light images (equation 2.47). As we will see shortly, due to the nature of the weighting function used for combining sub-path contributions, both the eye and light images will be missing illumination from the sampling strategies accumulated into the other image and will therefore look incorrect on their own.

$$\begin{aligned} I_j^E &= \frac{1}{N_j}\sum_{i=0}^{N_j} F_j^E \\ I_j^L &= (|D|/N)\sum F_j^L \\ I_j^B &= I_j^E + I_j^L \end{aligned} \tag{2.47}$$

With two images for different sampling strategies we now have two accumulation equations. However, just as with the direct lighting extension on path tracing and the direct pixel contribution extension to light tracing the contribution is still of the form $w_{s,t}C^*_{s,t}$ where the un-weighted contribution $C^*_{s,t}$ represents the throughput of a specific sampling strategy, and the weighting function $w_{s,t}$ ensures the sum of all strategies forms a summation over the weighted averages of all considered strategies of each length (equation 2.48).

$$\begin{aligned} F^E &= \sum_{s\geq 0}\sum_{t\geq 2} w_{s,t}C^*_{s,t} \\ F^L &= \sum_{s\geq 0}\sum_{1\geq t\geq 0} w_{s,t}C^*_{s,t} \\ \text{where}\quad C^*_{s,t} &= \alpha_s^L c_{s,t}\alpha_t^E \end{aligned} \tag{2.48}$$

We bring in the full eye and light path sampling steps from path tracing and light tracing to compute the α_s^L and α_t^E values respectively for the sub-path contributions (equation 2.49).

$$\begin{aligned} \alpha_0^L &= 1 \\ \alpha_1^L &= \frac{\mathcal{L}_e^0(y_0)}{\mathcal{P}_A(y_0)} \\ \alpha_2^L &= \alpha_1^L \frac{\mathcal{L}_e^1(y_0 \to y_1)}{\mathcal{P}(y_0 \to y_1|y_0)} \\ \alpha_i^L &= \alpha_{i-1}^L \frac{f_s(y_{i-3} \to y_{i-2} \to y_{i-1})}{\mathcal{P}(y_{i-2} \to y_{i-1}|y_{i-3} \to y_{i-2})} \\ \\ \alpha_0^E &= 1 \\ \alpha_1^E &= \frac{\mathcal{W}_e^0(z_0)}{\mathcal{P}_A(z_0)} \\ \alpha_2^E &= \alpha_1^E \frac{\mathcal{W}_e^1(z_0 \to z_1)}{\mathcal{P}(z_0 \to z_1|z_0)} \\ \alpha_i^E &= \alpha_{i-1}^E \frac{f_t(z_{i-3} \to z_{i-2} \to z_{i-1})}{\mathcal{P}(z_{i-2} \to z_{i-1}|z_{i-3} \to z_{i-2})} \end{aligned} \tag{2.49}$$

Finally we can compute the combination contribution for the sub-path. We start by copying the definitions for $c_{0,t}$ from path tracing, $c_{1,t}$ from direct lighting, $c_{s,0}$

from light tracing, and $c_{s,1}$ from direct pixel contribution. Now we need to define a combination strategy for a path composed of an arbitrary number of eye and light path vertices $c_{s,t}$. In this strategy both sub-path endpoints will lie on the surfaces of objects within the scene. The combined throughput between the two vertices is therefore the product of the BRDF for each vertex given the preceding vertex on each sub-path and going towards the vertex at the end of the other sub-path, scaled by the geometric attenuation function between the two end vertices (equation 2.50).

$$\begin{aligned}
c_{0,t} &= \mathcal{L}_e^0(z_{t-1})\mathcal{L}_e^1(z_{t-1} \rightarrow z_{t-2}) \\
c_{1,t} &= \mathcal{L}_e^1(y_0 \rightarrow z_{t-1})f_t(y_0 \rightarrow z_{t-1} \rightarrow z_{t-2})\mathcal{G}(y_0 \Leftrightarrow z_{t-1}) \\
\\
c_{s,0} &= \mathcal{W}_e^0(y_{s-1})\mathcal{W}_e^1(y_{s-2} \rightarrow y_{s-1}) \\
c_{s,1} &= \mathcal{W}_e^1(z_0 \rightarrow y_{s-1})f_s(y_{s-2} \rightarrow y_{s-1} \rightarrow z_0)\mathcal{G}(y_{s-1} \Leftrightarrow z_0) \\
\\
c_{s,t} &= f_s(y_{s-2} \rightarrow y_{s-1} \rightarrow z_{t-1})f_t(y_{s-1} \rightarrow z_{t-1} \rightarrow z_{t-2})\mathcal{G}(y_{s-1} \Leftrightarrow z_{t-1})
\end{aligned} \tag{2.50}$$

2.3.6.1. Weighting Function

Until now we have avoided discussing the details of the weighting function used in direct lighting, direct pixel contribution, and bidirectional path tracing. We know that to be a valid MIS estimator the weights of paths of equal length must sum to one so that our estimator forms a summation of weighted averages over paths of each length. For each sub-path that was sampled this requires us to compute the probability that the sub-path occurred using all of the other sampling strategies we consider. To help simplify the computation of the weighting function it is helpful to re-parameterize the path composed of s light path vertices and t eye path vertices as a path $\bar{x}_{s,t} = x_0, \cdots, x_k$ where $k = s + t - 1$, (equation 2.51).

$$\bar{x}_{s,t} = x_0, \cdots, x_k = \begin{cases} z_{t-1}, \cdots, z_1, z_0 & \text{if } s = 0 \\ y_0, y_1, \cdots, y_{s-1} & \text{if } t = 0 \\ y_0, \cdots, y_{s-1}, z_{t-1}, \cdots, z_0 & \text{if } s > 0, t > 0 \end{cases} \tag{2.51}$$
$$\text{where} \quad k = s + t - 1$$

Like the α_s^L and α_t^E values used in computing the throughput of sub-paths we can make use of shared information to pre-compute part of the full probability for each sub-path. Equation 2.52 shows the progression of computing the partial probabilities for the sub-paths that were actually sampled during creation of the eye and light paths. The final probability $p_{s,t}(\bar{x}_{s,t})$ for one of the sampled paths $\bar{x}_{s,t}$ is simply the product of the two partial probabilities from each sub-path.

$$\begin{aligned}
p_0^L &= 1\\
p_1^L &= \mathcal{P}_A(y_0)\\
p_2^L &= p_1^L \mathcal{P}(y_0 \to y_1 | y_0) \mathcal{G}(y_0 \Leftrightarrow y_1)\\
p_i^L &= p_{i-1}^L \mathcal{P}(y_{i-2} \to y_{i-1} | y_{i-3} \to y_{i-2}) \mathcal{G}(y_{i-2} \Leftrightarrow y_{i-1})\\
\\
p_0^E &= 1\\
p_1^E &= \mathcal{P}_A(z_0)\\
p_2^E &= p_1^E \mathcal{P}(z_0 \to z_1 | z_0) \mathcal{G}(z_0 \Leftrightarrow z_1)\\
p_i^E &= p_{i-1}^E \mathcal{P}(z_{i-2} \to z_{i-1} | z_{i-3} \to z_{i-2}) \mathcal{G}(z_{i-2} \Leftrightarrow z_{i-1})\\
\\
p_{s,t}(\bar{x}_{s,t}) &= p_s^L \, p_t^E
\end{aligned} \tag{2.52}$$

For the path $\bar{x}_{s,t}$ which was sampled with probability $p_{s,t}(\bar{x}_{s,t})$ we need to compute the probability that the same vertices that form path $\bar{x}_{s,t}$ were sampled using the $s+t$ other valid sampling strategies for paths of that length. Because we know that no matter how $\bar{x}_{s,t}$ was sampled it contains $s+t$ vertices, it is useful to define the probability that the vertices in $\bar{x}_{s,t}$ were in fact sampled using some other number of light vertices i and implicitly determining that the resulting number of eye path vertices used should be $(s+t)-i$. Equation 2.53 defines the notation used in the following equations — p_i represents the probability that a path actually sampled with s light vertices and t eye vertices was instead sampled with i light vertices and $(s+t)-i$ eye vertices.

$$\begin{aligned}
p_s &= p_{s,t}(\bar{x}_{s,t})\\
p_i &= p_{i,(s+t)-i}(\bar{x}_{s,t})
\end{aligned} \tag{2.53}$$

When $i = s$ we have the special case p_s which is simply the probability $p_{s,t}(\bar{x}_{s,t})$ with which we actually sampled the path. Because we have access to the value p_s but not the remaining values for p_i when $i \neq s$ it is efficient to define the remaining p_i values in terms of p_s. Equation 2.54 shows the relationship between values of p_i and p_{i+1}. Because both methods of sampling path $\bar{x}_{s,t}$ only differ in how they sample vertex x_i the relationship between them simplifies to just the difference in probability and geometric attenuation used to sample the specific vertex as a member of either the eye or light path.

$$\frac{p_1}{p_0} = \frac{\mathcal{P}_A(x_0)}{\mathcal{P}(x_0 \rightarrow x_1|x_0)\mathcal{G}(x_1 \Leftrightarrow x_0)}$$

$$\frac{p_{i+1}}{p_i} = \frac{\mathcal{P}(x_{i-1} \rightarrow x_i|x_{i-2} \rightarrow x_{i-1})\mathcal{G}(x_i \Leftrightarrow x_{i-1})}{\mathcal{P}(x_{i+1} \rightarrow x_i|x_{i+2} \rightarrow x_{i+1})\mathcal{G}(x_i \Leftrightarrow x_{i+1})} \tag{2.54}$$

$$\frac{p_{k+1}}{p_k} = \frac{\mathcal{P}(x_{k-1} \rightarrow x_k|x_{k-2} \rightarrow x_{k-1})\mathcal{G}(x_k \Leftrightarrow x_{k-1})}{\mathcal{P}_A(x_k)}$$

Another way of interpreting this is given the known probability of sampling a path using i light path vertices and $(s+t)-i$ eye vertices p_i, the probability of instead sampling the path using $i+1$ light path vertices and $(s+t)-(i+1)$ eye vertices p_{i+1} is simply the original probability p_i divided by the probability of sampling vertex x_{i+1} on the eye path, multiplied by the probability of instead sampling it as part of the light path. Equation 2.55 shows the steps for computing the value of p_i for increasing i covering $p_{s+1}, p_{s+2}, \cdots, p_k, p_{k+1}$. The ratio defined in equation 2.54 is applied repeatedly to yield the next term in the sequence. The reciprocal ratio between p_i and p_{i-1} is defined similarly to equation 2.54 and allows for the remaining $p_{s-1}, p_{s-2}, \cdots, p_0$ to be computed iteratively.

$$p_s = p_{s,t}(\bar{x}_{s,t})$$

$$p_{s+1} = p_s \frac{\mathcal{P}(x_{s-1} \rightarrow x_s|x_{s-2} \rightarrow x_{s-1})\mathcal{G}(x_s \Leftrightarrow x_{s-1})}{\mathcal{P}(x_{s+1} \rightarrow x_s|x_{s+2} \rightarrow x_{s+1})\mathcal{G}(x_s \Leftrightarrow x_{s+1})}$$

$$p_{s+2} = p_{s+1} \frac{\mathcal{P}(x_s \rightarrow x_{s+1}|x_{s-1} \rightarrow x_s)\mathcal{G}(x_{s+1} \Leftrightarrow x_s)}{\mathcal{P}(x_{s+2} \rightarrow x_{s+1}|x_{s+3} \rightarrow x_{s+2})\mathcal{G}(x_{s+1} \Leftrightarrow x_{s+2})} \tag{2.55}$$

$$\ldots$$

$$p_{k+1} = p_k \frac{\mathcal{P}(x_{k-1} \rightarrow x_k|x_{k-2} \rightarrow x_{k-1})\mathcal{G}(x_k \Leftrightarrow x_{k-1})}{\mathcal{P}_A(x_k)}$$

With values of p_i computed for all i the weighting function is simply defined as the relative probability with which we actually sampled the path, out of the sum of all the probabilities with which we could have sampled the path using different sampling strategies. Here we adopt the commonly used power heuristic weighting function discussed previously in equation 2.20. An important distinction here is that the sum in the denominator is only over the sampling methods that were actually used during sub-path creation. For instance, when using a minimal camera model such as a pin-hole camera it is impossible to randomly intersect a lens element of the camera as it does not have a geometric representation. In this scenario we must ensure that the weighting function reflects the fact that we can never accumulate contribution using sampling strategy $c_{s,0}$. To accommodate this we make the elements of the denominator a product of the probability of sampling each sub-path and a value $u_i = u_{i,(s+t)-i}$ with value 0 or 1 denoting whether we consider paths using strategy $\bar{x}_{i,(s+t)-i}$ during the path generation stage of the algorithm, (equation 2.56). As an illustrative example, the weighting scheme used for the direct lighting extension of standard path tracing would only consider sampling strategies of the form $u_{0,t} = u_{1,t} = 1$ for all eye path lengths t, and $u_{s,t} = 0$ for all $s \geq 2$.

$$w_{s,t} = \frac{p_s^2}{\sum_i (p_i \, u_i)^2} \tag{2.56}$$

2.3.7 Advanced Sampling Methods

Since the development of bidirectional path tracing [LW93; Vea97] much work has been done in the area of improving the efficiency and effectiveness of sampling strategies based on this approach. In this section we will review some of the significant contributions in this area.

2.3.7.1. Metropolis Light Transport (MLT)

In addition to the contribution of using MIS to compute near optimum path weightings in bidirectional path tracing, Veach also presented a novel application of MHMC sampling applied to the sampling of paths during the rendering process [VG97]. Starting with an initial “seed” eye and light path sampled using the standard bidirectional path tracing procedure outlined in section 2.3.6 a Markov chain is constructed by applying a mutation operator to the current path composed of the full eye and light paths, yielding a new path proposal. This proposed new path is then accepted or rejected using the MHMC acceptance probability where the target distribution is based on the luminance of the throughput between the full MIS contributions of the original and mutated paths, and the conditional proposal distribution is written as the transition probability of the mutation occurring in either direction in the manner that it did, (equation 2.57).

$$\mathcal{A}(\bar{x} \rightarrow \bar{x}') = \frac{f_{\text{lum}}(\bar{x}')\mathcal{T}(\bar{x}' \rightarrow \bar{x})}{f_{\text{lum}}(\bar{x})\mathcal{T}(\bar{x} \rightarrow \bar{x}')} \tag{2.57}$$

Mutations can be designed in a flexible way such that certain mutations focus on sampling specific types of paths. Examples of mutation strategies include lens mutations which update the eye path, caustic mutations which update the light path, and manifold mutations [JM12] which update multiple vertices surrounding and on specular surfaces in coordination with one another so as to respect the dependencies imposed by Dirac distributions on specular surfaces.

A key aspect of the Metropolis Light Transport (MLT) algorithm is that path mutations are performed in path-space. For instance, a lens mutation which inserts a new vertex on the eye path between two existing vertices would be sampled by rotating the outgoing direction from the existing vertex (in polar-coordinates) before the insertion point, tracing to the first intersection in that direction to find the new vertex location, and then connecting the new vertex to the existing next vertex on the path. The implementation of such mutation strategies can be complex and error prone and computation of the probabilities with which mutations occur can also be difficult to validate the correctness of.

2.3.7.2. Primary Sample Space Metropolis Light Transport (PSSMLT)

A greatly simplified variant on MLT is proposed by Kelemen *et. al.* [Kel+02], who propose to move the computation of path mutations from path-space to the primary sample space from which paths are generated. When generating the initial seed paths Primary Sample Space Metropolis Light Transport (PSSMLT) recognizes that in order to sample the next vertex on the path a tuple of uniform random numbers (usually two or three) is sampled which are transformed onto the target sampling distribution of the PDF at the vertex. By treating this sequence of random values as the seed for generating the path we can see that the same path can be re-generated by tracing it out again and using the same random values to sample the outgoing direction at each vertex. By performing simple numerical mutations on the stream of random numbers directly and then using the mutated streams to re-generate the eye and light paths we can sample mutated paths without having to worry about the many edge cases that occur when implementing MLT.

2.3.7.3. Multiplexed Metropolis Light Transport (MMLT)

Another improvement on MLT and PSSMLT is proposed by Hachisuka *et. al.* [Hac+14]. In MLT, seed eye and light paths are sampled to a fixed maximum length or to a length determined by Russian roulette path termination. In PSSMLT the sequence of random values tracked for path generation is usually stored to a maximum length representing an implicit upper bound on the sampled path length, though it is allowed that the path corresponding to a given random stream may terminate before all of the values in the stream have been used.

However, there are many scenarios where the path will naturally mutate into high throughput regions of the path-space, where lighting predominantly contributes to the image through short eye and light paths in directly visible regions near light sources. In these regions it is wasteful to spend time computing full-length eye and light paths that will predominantly contribute to the image through the shortest of their MIS weighted sub-paths. In Multiplexed Metropolis Light Transport (MMLT) the length of path traced is also sampled as part of the mutation and proposal scheme with a transition probability between sampling paths of discrete lengths. This allows the target path length to grow or shrink as needed while the mutated path walks its way around the path-space. This significantly decreases the amount of time the algorithm spends constructing proposal paths for each mutation.

2.3.7.4. Energy Redistribution Path Tracing (ERPT)

There are significant advantages to using MHMC based rendering algorithms such as MLT, PSSMLT, and MMLT as they allow for difficult to sample paths which occur in low probability but high contribution regions of the path-space to be efficiently explored. However, there are also several drawbacks to these methods inherited from the underlying MHMC sampling strategy. One drawback is that paths sampled through the mutation process exhibit high correlation with one another, while their contributions to the image estimate during accumulation is assumed to be independent. This means that for any finite length sub-sequence of the random walk the expectation over the sequence of sample contributions will be a biased estimate of the true expectation. This inter-sample correlation combined with the arbitrary seed path sampled by bidirectional path tracing (which was not sampled w.r.t. the transition function of the random walk) can result in significantly biased image estimates at low sample counts.

A commonly used method to remove the start-up bias is to use a “burn-in” phase after seeding the sampler with a path sampled by bidirectional path tracing. During the burn-in the first N steps of the random walk will be discarded and do not contribute to the image. After a sufficiently large number of the burn-in steps the random walk of the sampler should have traversed to a representative region of path-space where accumulation can begin without affecting the stability of the estimate. For short sequences after the burn-in phase there is still an issue caused by high correlation between samples, but this is greatly reduced and will allow the estimate to converge to the expected value as more mutation steps are computed.

Energy Redistribution Path Tracing (ERPT) [Cli+05; Bat05] takes a different approach to the start-up bias problem of MHMC samplers by combining the strengths of both standard Monte Carlo and MHMC methods. In ERPT a seed path sampled with bidirectional path tracing is used to compute an image contribution. A MHMC style random walk is then performed for a small number of mutation steps — at each step a fraction of the seed path's contribution is accumulated to the pixel which the current mutated path corresponds to. By using a small number of mutation steps this strategy leverages the initial bias of the random walk being highly correlated to the seed path to smear its contributions over the local highly correlated region of the path-space. This region projected onto the image plane can correspond to a local anisotropically distributed neighbourhood of pixels in the accumulated image which can cause light bleeding and other visual distortions when a small number of seed paths are used.

At a high level, the relationship between MLT based methods and ERPT is that the former attempts to construct long mutation sequences from a small number of seed paths per-pixel, allowing the sampler to spend more time in high contribution regions of the path-space to avoid bias, while ERPT uses short mutation sequences initialized from a large number of seed paths per-pixel to leverage the high correlation between the seed path and neighbouring regions of the path-space.

2.3.7.5. Anisotropic Gaussian Mutations for Metropolis Light Transport Through Hessian-Hamiltonian Dynamics

The flexible way in which mutation strategies are designed for MLT based algorithms allows strategies to be specialized, focusing on sampling specific types of paths. As we saw previously in section 2.2.2.2, HMC offers an alternative approach to MHMC based on Hamiltonian Dynamics style particle simulations. Adopted from the physics literature, HMC uses long and curved arcs across the sampling domain to generate a Markov chain of samples with low temporal correlation, greatly improving the robustness of estimates at low sample counts.

A draw back of HMC is that at each time-step during the tracing of an arc to find the new sample proposal we need to compute the derivative of the target function in order to update the momentum of the simulated particle. In problems where the derivative is expensive to compute this makes HMC sampling impractical to apply as many derivative calculations are needed to generate a new proposal.

Li *et. al.* [Li+15] present Hessian Hamiltonian Monte Carlo (HHMC), a novel method for approximating Hamiltonian Dynamics based on the Hessian of the target function in problem domains where the derivative is expensive to compute. The Hessian of a function is a matrix representation of all of its second-order partial derivatives. The Jacobian is a vector of the first-order partial derivatives of a function and is often referred to as the function gradient due to its representation of the local linear slope of the function at a given point. It follows that the Hessian, representing second-order partial derivatives can therefore be thought of as the local quadratic curvature of the function at that point.

Applying Taylor's Theorem (equation 2.58) we can show that given the value of a function f at a given vector valued point x we can approximate the value of the function at a nearby point $\hat{x}$. If the function is locally flat then we can say that $f(\hat{x}) = f(x)$ as there is no change in the function value between the two points. Assuming the function is linear w.r.t. its inputs we can interpolate from our known point x to $\hat{x}$ by using the local gradient $\mathcal{J}_f(x)$ scaled by the distance between the points $(\hat{x}-x)$, this gives the first-order Taylor expansion $f(\hat{x}) = f(x)+\mathcal{J}_f(x)(\hat{x}-x)$.

While the assumption of linearity across the whole function is unlikely to be valid, a local assumption of linearity in the region surrounding x can be sufficient to use the first-order Taylor expansion to approximate $\hat{x}$ when it is nearby to x which is what we see in the original HMC algorithm. When the function f has increasingly high curvature, the region around x which can be approximated accurately with the first-order Taylor expansion diminishes. By adding higher-order terms to the expansion the region which can be trusted increases. Adding a term using the Hessian matrix $\mathcal{H}_f(x)$ the second-order Taylor expansion is now given by $f(\hat{x}) = f(x) + \mathcal{J}_f(\hat{x} - x) + \frac{1}{2}(\hat{x} - x)\mathcal{H}_f(x)(\hat{x} - x)^\top$.

$$\begin{aligned} f(\hat{x}) &\approx f(x) \\ &\approx f(x) + \mathcal{J}_f(x)(\hat{x} - x) \\ &\approx f(x) + \mathcal{J}_f(x)(\hat{x} - x) + \frac{1}{2}(\hat{x} - x)\mathcal{H}_f(x)(\hat{x} - x)^\top \\ &\dots \end{aligned} \tag{2.58}$$

In the work of Li *et. al.* [Li+15] the computation of the full MIS path contribution used in the PSSMLT target function for mutations is implemented using an Automatic Differentiation framework [Nei10]. This allows for the first- and second-order derivatives of the target function to be computed w.r.t. the values used to generate the path. By sampling paths from within a local Gaussian window around the current path w.r.t. the local curvature of the path-space they are able to perform an approximate Hamiltonian Dynamics update arc in a single step. Despite the computation of the Hessian matrix being slower to compute than the Jacobian, this is mitigated by the need to only compute a single Hessian matrix per-mutation, as opposed to many individual Jacobian computations per mutation in the standard HMC sampling process.

2.3.7.6. Gradient Domain Rendering

Recent work has aimed at applying Monte Carlo rendering algorithms to estimate the gradient of the contribution across the image plane rather than the direct contribution estimated by standard Monte Carlo rendering algorithms. By designing algorithms that estimate the gradients of path contributions while also accumulating a coarse approximation of the direct image estimate, this data can be fed into a "screened" Poisson solver [CK11] in order to reconstruct the underlying image. Further it can be shown that as the estimate of the gradients converges along with the coarse approximation of the direct image, this strategy computes an unbiased estimate of the expected true image.

In the seminal work on gradient domain rendering, Lehtinen *et. al.* [Leh+13] proposed Gradient Domain Metropolis Light Transport (G-MLT), a modification of MLT where the target function used for path mutations is designed to drive the Markov chain sampler towards regions of path-space where the image gradient is high. This drives computation into regions where fine details and complex light interactions are prevalent, maximizing the usefulness of the samples which are computed.

To mutate paths, their work adopts the lens mutation strategy from standard MLT along with the specular manifold exploration strategy from [JM12]. These strategies are extended to form a shift-mapping which mutates a path going through a given pixel to a path going through one of the neighbouring pixels which is highly correlated with the original path. By applying the constraint that shifted paths are similar to the original paths through half-vector constraints, it ensures that the magnitude of the finite difference gradient of the contributions between paths will be a more robust estimate of the underlying image gradient.

While G-MLT greatly increases the efficiency of rendering compared to standard MLT there are also drawbacks to this approach. The complexity of the MLT implementation due to its dependencies on path-space mutation strategies is shared by G-MLT, and even exacerbated by the additional constraints of the shift-mapping, making independent implementation of the method a complex task. Another issue shared with MLT is the need for a burn-in phase to avoid start-up bias, and the unreliability of the estimate at low sample counts.

More recently, Kettunen *et. al.* [Ket+15] from the same research group, present Gradient Domain Path Tracing (G-PT) as a hybrid approach between standard path tracing and G-MLT to address the issues that arise from the use of MLT based methods. In G-PT, eye paths traced through each pixel are used to form a coarse image of direct path contributions. For each path, a shift-mapping is used to create an offset path going through a neighbouring pixel which is used solely for a finite difference approximation of the image gradient between paths. The shift-mapping is based on sampling a pixel adjacent to the one used in the seed path and attempting to reconnect the shifted path to the seed path as soon as possible. Specifically, this occurs when the shifted path intersects a diffuse surface, and the corresponding next vertex on the seed path is diffuse, allowing for an explicit connection to be made. This approach is considerably more simple to implement than the G-MLT algorithm and behaves more stably at low sample counts.

Lastly, in a follow-up to their work on G-MLT and G-PT, Manzi *et. al.* [Man+15] develop Gradient Domain Bidirectional Path Tracing (G-BDPT) as a similar hybrid approach between bidirectional path tracing, G-MLT, and G-PT. In this work the shift-mapping used to create offset paths is based on the mapping used in G-MLT. This is preferable to the mapping developed for G-PT as the latter does not consider the use of a light path which can allow for the offset eye path to be connected back to the remaining path after a single diffuse vertex, as opposed to after two diffuse vertices for G-PT. This helps minimize the difference between offset and seed paths allowing for more robust gradients to be accumulated.

2.3.7.7. Selective Progressive Rendering

By default, rendering algorithms such as path tracing (section 2.3.4) dedicate and equal computation budget to each pixel, usually parameterized by the number of samples evaluated per pixel. Conversely, the dual algorithm light tracing (section 2.3.5) allows evaluated samples to contribute to any pixel through which the currently considered path vertex is visible through. This allows proportionally more samples to contribute in image regions which have a higher volume of light energy flowing through them. This concept is extended in algorithms such as MLT and PSSMLT which iteratively mutate seed paths to form a Markov Chain of proposal paths which spends proportionally more or its time in regions of the path space with high contribution to the image.

Selective progressive rendering methods offer a means of balancing a computational budget over the pixels of an image generated with a progressive rendering algorithm such as path tracing. The heuristic used to guide load balancing must be chosen carefully, as to proportionally increase the computation on one area of the image is also to reduce the computation of other areas. If the method used to distribute the computation budget is not representative of perceived image distortion then the resulting images may appear to be of lower visual quality than uniformly sampled images.

Another application of such techniques is used to recognize programmatically when an image has reached a desired quality to halt the rendering process before the maximum computation budget is reached. By evaluating the change in images as samples are progressively added, Myszkowski *et. al.* [Mys98] and Bolin *et. al.* [BM98] showed that FR-IQA, in the form of Visual Difference Predictor (VDP) [Dal93] and Visual Discrimination Model (VDM) [Lub95] respectively, could be applied to the task of early halting in a path tracing software. A drawback of their approaches was that the computational expense of computing the FR-IQA measures often outweighed efficiency gains from early halting.

Ramasubramanian *et. al.* [Ram+99] apply perceptual measures to pre-compute an initial sample density map based on direct illumination and an ambient approximation term. This is used to progressively direct a computational budget during the computation of indirect illumination. As the render progresses, the sample density map is updated periodically by comparing the current rendering solution to the one from the previous iteration. Rendering continues until convergence is achieved.

A similar scheme applied to animated sequences was proposed by Myszkowski *et. al.* [Mys+00]. In their work, predefined camera paths (non-interactive rendering) through a virtual scene allow for the sparse computation of high quality key-frames to be interpolated between using Image Based Rendering (IBR) [KS00] techniques. Using an adapted form of VDP termed Animation Quality Metric (AQM), which was modified to work on temporal image sequences, they determine whether the IBR interpolated images are of sufficient visual quality for use in the animated sequence. When they are found not to be of sufficient quality a new key-frame is recursively computed between the two existing ones and the surrounding frames are re-interpolated through IBR. This process continues until the entire sequence has been generated to a sufficient visual quality.

Debattista [Deb06] provides a thorough review of selective rendering approaches and develops techniques for selective progressive rendering based on time and visual quality constraints. This work also investigates the use of component-based rendering where geometric entities are rendered to different qualities based on their visual persistence in resulting images.

Recent work from Harvey *et. al.* [Har+17] studies the use of selective progressive rendering in multi-modal rendering domains such as virtual environments containing both audio and visual stimulus. In a controlled user study using offline rendering, they show that the sampling density from the interactions of simulated acoustics in virtual environments can be used as an indicator of user attention, allowing selective rendering techniques to drive more computation into image regions where the user was predicted to be focussing without a perceivable loss in quality.

2.3.8 Discussion

When applied to physically based rendering, Monte Carlo methods give us a robust and extensible framework to build rendering algorithms which converge to the true solution as the number of samples used to approximate a rendered image is increased in the limit.

In the path tracing algorithm we see Monte Carlo methods applied in several places to approximate an integral formulation of the rendering equation [Kaj86] which is both infinitely branching and recursive. To sample individual paths, a Markov Chain is constructed to compute each light bounce by iteratively drawing the reflected ray direction for the next bounce from the conditional probability distribution, predicated on having arrived at the current intersection from the direction of the previous intersection. For each pixel, a standard Monte Carlo sampler is computed by averaging the contributions of all paths sampled through that pixel in the image. By re-sampling individual paths continually, the overall estimator converges to the underlying integral without incurring the issues associated with infinite branching and recursion. Importance sampling is commonly applied when sampling the conditional distributions to get the reflected direction at each bounce during path construction, this technique greatly reduces variance in the overall estimator by focusing computation towards high-probability regions of the local path-space that are likely to contribute highly to the estimator.

Throughout this section we have explored the application of advanced Monte Carlo sampling strategies to various parts of the rendering process. These methods are appropriate in specific situations but often can perform poorly when sampling regions of the path-space which do not lend themselves to one sampling strategy or another. Rendering algorithms based on BDPT [LW93; Vea97; VG97; Kel+02; Cli+05; Bat05] try to resolve this issue using multiple importance sampling by constructing a unified estimator which draws samples from multiple, potentially overlapping, regions of the sample space while being sure to maintain an unbiased estimator.

Algorithms such as MLT [VG97] and PSSMLT [Kel+02] extend bidirectional sampling strategies using Markov Chains, by constructing seed paths which are repeatedly mutated to walk around the path-space. In such methods, the sampler ends up spending proportionally more of its time in regions of the path-space with high contribution to the final expectation; this has the effect of lowering the overall variance of the estimator because each sample has a higher likelihood of being representative of the final expectation. By using a wide range of different mutation and sampling strategies together in a single estimator [JM12; Li+15] MLT based methods allow for rapid and stable convergence across a wide variety of challenging material, lighting and scene compositions.

The novel application of Poisson solvers to Monte Carlo rendering algorithms led to the development Gradient Domain [Leh+13; Ket+15; Man+15] rendering techniques. These methods showed that unbiased estimators can be constructed from coarse rendering approximations combined with estimates of the 1^{st} order image gradient computed from path sampling strategies similar to ERPT [Cli+05; Bat05] and Manifold Exploration [JM12], greatly reducing the impact of impulse noise in under-sampled image regions while still converging to the correct rendering solution.

Finally we saw how the above rendering algorithms could be augmented with selective progressive rendering [Mys98; BM98; Ram+99; Mys+00; Deb06; Har+17] techniques. These methods aim to control the distribution computational resources by delegating them to image regions with lower perceived quality. By reducing the computation requirements of image regions with low error significant performance gains can achieved within a fixed computation budget over uniform sampling techniques.

2.4 Machine Learning

Machine learning as a sub-field of computer science is the study of designing and training algorithms to extract robust strategies directly from data that are representative of the chosen problem domain. Machine learning includes a variety of tasks such as image classification and segmentation, natural language processing, generative models which synthesize images, text, and audio data, and many others. In recent years, the increased accessibility of massively parallel compute devices such as GPU has allowed for an unprecedented surge in machine learning research. This surge has also been leveraged by the increased scale of data that we are able to store and process.

While there are many types of machine learning algorithm, one area has benefited particularly well from the recent renaissance in machine learning. Artificial Neural Network (ANN) [MP43] were the result of early work which aimed to improve our understanding of how biological processes involving neurons work within the brain. These early methods showed compelling results but were infeasible to work with on the hardware available at the time. This is analogous to the recent surge of interest in Monte Carlo rendering algorithms. When the rendering equation was first introduced by Kajiya in 1986 [Kaj86] the applicability of the proposed path tracing algorithm was prohibitively expensive to evaluate on the available hardware. This had the effect of severely stalling the development of improved methods until access to more powerful hardware became feasible for researchers to utilize.

Contributions to the field of machine learning generally fall into one of two camps: supervised learning, where example input data is labelled with the desired output data that the model should learn to predict; and unsupervised learning, where unlabelled data is used to train models which (generally) attempt to find representations for the data which have lower dimensionality but are still representative of the higher dimensional source data. Unsupervised methods can be thought of as clusterings, dimensionality reductions, or data embeddings and have a wide range of applications due to the abundance and availability of raw unlabelled data. Supervised methods are also applicable to a broad range of applications and can be thought of as a class of function approximators which potentially both accept and predict high dimensional values.

2.4.1 Multi-Layer Perceptron

Rosenblatt [Ros57] formalized much of the early work on neural networks, developing the idea of a perceptron as a simple function approximator that can be fit to observed data and and provide inference on unseen data. Perceptrons were later generalized to Multi-Layer Perceptron (MLP) [Rum+85] which stacked multiple perceptrons together, feeding the output from one perceptron as input to the next in a feed-forward architecture. Deep models like MLP are able to efficiently model arbitrary functions, allowing them the capability to extract complex trends directly from raw data or from hand-crafted features.

Equation 2.59 shows the general form of a single perceptron unit. Each element j in the output feature vector a_j^{l+1} is the result of a unique weighted summation over each value from the incoming feature vector a_i^l. When $l = 0$, a^l is the model input and for $l > 0$ it is the output from the previous layer. To compute j^{th} output feature, the i^{th} input feature is scaled by weight W_{ij}^l when summed. The summation is added to a scalar bias value B_j^{l+1} which shifts the result before it is fed to an activation function $f(\cdot)$ which often contains some form of non-linearity.

$$a_j^{l+1} = f\left(B_j^{l+1} + \sum_{i=1}^{N^l} a_i^l W_{ij}^l\right) \tag{2.59}$$

The weights W_{ij}^l form a rectangular matrix of size $N^l \times N^{l+1}$. By arranging the input features a^l as a $1 \times N^l$ vector the weighted summations for each element can be rewritten as a single matrix multiplication of $a^l W^l$. Equation 2.60 shows the vector form of the perceptron.

$$a^{l+1} = f\left(B^{l+1} + a^l W^l\right) \tag{2.60}$$

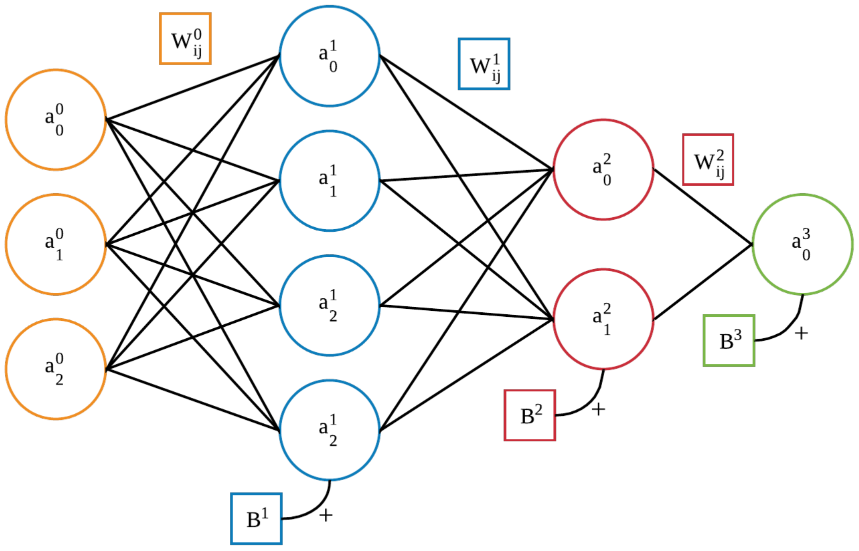


Fig. 2.7.: An example of a Multi-Layer Perceptron with two hidden layers. An input vector of three features a^0 (orange) feed into a vector of four features a^1 (blue) through a dense matrix multiplication with weight matrix W^0. Bias vector B^1 is added to the result and an activation function $f(\cdot)$ is applied element-wise, yielding the output feature vector a^1. This process is repeated for the remaining layers, feeding in the feature vector a^l to compute a^{l+1}.

Figure 2.7 shows the general structure of an MLP with two hidden layers and one output neuron. The hidden layers represent the learned intermediate representation of the model. We call them hidden layers because their values are extracted from data during the training process and are not readily interpretable to us. Features in these layers have complex non-linear relationships with one another, and with features in the surrounding layers. This complexity is what allows MLP to accurately model the relationships in complex data. Equation 2.61 shows the vector form of the MLP in figure 2.7 along with the sizes of the bias vectors and weight matrices.

$$\begin{aligned}
a^1 &= f\left(B^1 + a^0 W^0\right) \quad \text{where} \quad B^1 = \{1 \times N^1\},\ W^0 = \{N^0 \times N^1\} \\
a^2 &= f\left(B^2 + a^1 W^1\right) \quad \text{where} \quad B^2 = \{1 \times N^2\},\ W^1 = \{N^1 \times N^2\} \\
a^3 &= f\left(B^3 + a^2 W^2\right) \quad \text{where} \quad B^3 = \{1 \times N^3\},\ W^2 = \{N^2 \times N^3\}
\end{aligned} \tag{2.61}$$

$$\begin{aligned}
a^3 &= f\left(B^3 + f\left(B^2 + f\left(B^1 + a^0 W^0\right) W^1\right) W^2\right) \\
&\text{where} \quad N^0 = 3,\ N^1 = 4,\ N^2 = 2,\ \text{and } N^3 = 1
\end{aligned}$$

2.4.1.1. Activation Functions

The activation function $f(\cdot)$ is a non-linearity that is applied element-wise to the output feature vector of a layer. The choice of function is somewhat arbitrary. Many activations functions have been proposed and used over the years, each with varying degrees of effect on model accuracy when working with different types of data. Figure 2.8 shows four common activation functions.

The classic choice in non-linearity was the Sigmoid function (equation 2.62) which is a smooth stepped function in the range $[0, 1]$ with a smooth falloff as it approaches the extremes. An issue with this function as that the truncation of values as the function approaches 0 or 1 causes the gradient of the activation to go to zero. When multiple layers are stacked together in an MLP these truncated gradients compound with each other, leading to the so called vanishing gradient problem. When training these models the gradient of the network w.r.t. an objective function is used to optimize the values of the weight matrices and bias vectors. If the gradient of an individual parameter becomes too small it will take an extremely long time for it to move away from its current value to help minimize the objective function. This can stall training and effect stability.

$$f_{\text{Sigmoid}}(x) = \frac{1}{1 + e^{-x}} \tag{2.62}$$

The TanH function (equation 2.63) is a similar smooth step activation, this time in the range $[-1, 1]$. This function can also suffer from the vanishing gradient problem but allows output features to carry negative values to the next layer.

$$f_{\text{TanH}}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{2.63}$$

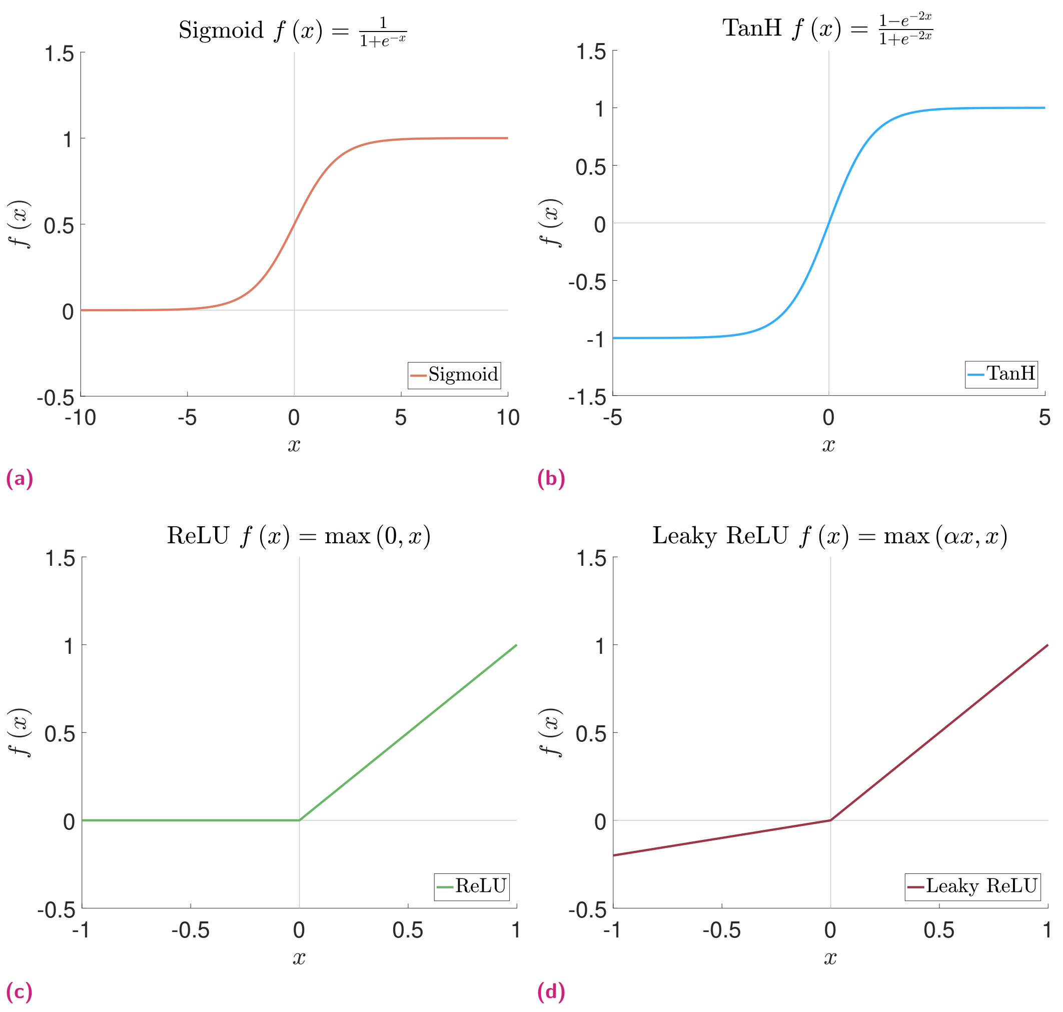


Fig. 2.8.: A selection of activation functions commonly used with neural networks. (a) Sigmoid activation which outputs in the range $[0, 1]$, (b) TanH activation which outputs in the range $[-1, 1]$, (c) ReLU activation which outputs in the range $[0, \infty]$, and (d) Leaky or Parametric ReLU activation which outputs in the range $[-\infty, \infty]$

In recent years, Rectified Linear Unit (ReLU) activation (equation 2.64) has seen a surge in popularity due to its efficiency and performance in many problem domains. ReLU is defined as being linear for positive inputs, and 0 for all negative inputs. This can be implemented efficiently by computing the maximum value of each feature x and zero.

$$f_{\text{ReLU}}(x) = \max(0, x) \tag{2.64}$$

One issue with ReLU is that when activations are negative the gradient is zero. This can cause training slow or for neurons to "die" because the model learns a strategy that never utilizes some of its capacity. A solution to this problem is to use a "leaky" ReLU that has a small positive slope in the range $[0, 1]$. This limits the extent to which negative activations contribute to the next layer, but allows for a well defined gradient that is easier to optimize over. Leaky-Rectified Linear Unit (LReLU) [Maa+13a] can be implemented efficiently like ReLU by taking an element-wise maximum value, this time between the input feature x and that feature scaled by a multiplicative gain factor αx (equation 2.65). If α is in the range $[0, 1]$ then the maximum will cause the unscaled value to be used for positive inputs, and the scaled value to be used for negative inputs. A commonly used α value or "leak" is a value of 0.2.

$$f^{\alpha}_{\text{Leaky ReLU}}(x) = \max(\alpha x, x) \quad \text{where} \quad 0 \leq \alpha \leq 1 \tag{2.65}$$

He *et. al.* [He+15b] propose a subtle variant on LReLU which makes the leak value α a trainable parameter optimized by gradient descent along with the other weights and biases. A separate α value can be learnt for every activation in each layer. This has been shown to increase accuracy in a number of applications [Xu+15], at the expense of adding a large number of trainable parameters which slow the efficiency by which the model can be optimized.

2.4.1.2. Loss Functions

In order to train MLP models we must first define a measure of goodness-of-fit with which to drive our optimization process. This is commonly structured as the minimization of a function representing the current accuracy or quality of prediction our model is making. The two main forms of loss function are regression losses, which attempt to synthesize real valued and possibly high dimensional output predictions that accurately fit trends in the input data; and classification losses that attempt to make predictions which are used to model a decision boundary between a discrete set of class labels.

Regression In the regression setting the most commonly used loss functions are the $\mathcal{L}_1$ and $\mathcal{L}_2$ losses due to their simplicity and empirical performance in many applications. The main difference between these objective functions is that the $\mathcal{L}_1$ loss weights deviations from the expected value linearly while the $\mathcal{L}_2$ loss weights deviations quadratically. For the $\mathcal{L}_2$ loss, this has the effect of exacerbating the contribution to the overall loss from examples in the data which are poorly represented by the model, implying that a good way to minimize the objective function is to work on better predicting these misrepresented examples. A limitation of the $\mathcal{L}_2$ loss is that by weighting the deviations quadratically the gradient of the loss function becomes large for values that far from their expected value. This can cause instability during training leading to an "exploding gradient" problem. A drawback of the $\mathcal{L}_1$ loss is that its derivative is discontinuous as the loss function passes over origin with a value of ± 1 without a smooth transition in between these extremes. This can also have the effect of causing instability during training as it is desirable for the gradient of parameters to suppress as they become close to their ideal values.

$$\begin{aligned} \mathcal{L}_1 &= \sum_{i=1}^{C} |y_i - \hat{y}_i| \\ \mathcal{L}_2 &= \frac{1}{2} \sum_{i=1}^{C} (y_i - \hat{y}_i)^2 \end{aligned} \tag{2.66}$$

Equation 2.66 shows the general form of the $\mathcal{L}_1$ and $\mathcal{L}_2$ loss functions for a set of predicted values output from the model $\hat{y}$ and their expected values from the dataset y. These loss functions are defined as a summation over the C output neurons from the model. This generalizes the loss functions from working on scalar values to operate on higher dimensional regression targets such as signals, images, volumes, and more.

Classification For classification tasks the goal of our model is to predict a probability for each of our possible output classes that represent the models confidence that the given input is an example of that class. To do this, we structure our models to have as their output a feature vector containing the same number of features as we have classes. Our model is trained to predict a greedy decision boundary between these features such that the feature with the biggest activation for a given input is considered to be intended classification choice. We refer to this task as "logistic regression" where the output features of our model are termed "logits". For a given input the predicted logits can have any real numbered value making them difficult to interpret w.r.t. one another. To transform these raw values into predicted class probabilities we can transform them using a Softmax activation function (equation 2.67). This non-linearity forces all logits to become in the range $[0, 1]$ where a flat or multi-modal distribution between values represents low certainty and a uni-modal distribution represents high certainty.

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{j=1}^{C} e^{a_j}} \tag{2.67}$$

With our logits transformed into predicted class probabilities we define the Categorical Cross-Entropy, $\mathcal{L}_{\text{Categorical}}$, loss for a model with C output classes as a measure of mutual information between the expected and predicted probability vectors. In this formulation, the true values from the dataset are constructed as "one-hot" vectors where the expected value for each output class is 0 except for the intended class which is given probability 1. Equation 2.68 shows the general form of the $\mathcal{L}_{\text{Categorical}}$ loss operating on Softmax activated logits.

$$\mathcal{L}_{\text{Categorical}} = -\sum_{i=1}^{C} y_i \ln \hat{y}_i \tag{2.68}$$

A drawback of this formulation for the $\mathcal{L}_{\text{Categorical}}$ loss is that it forces the intermediate evaluation of the sum over the constant e is raised to a power. By combining equations 2.67 and 2.68 and simplifying the result we can formulate the $\mathcal{L}_{\text{Categorical}}$ loss to operate directly on un-activated logits. Equation 2.69 shows the combined activation and loss which now only contains two natural logarithms per output class. In this configuration, the true class labels y are still represented as one-hot vectors.

$$\mathcal{L}_{\text{Categorical}} = -\sum_{i=1}^{C} y_i \ln a_i + (1 - y_i) \ln(1 - a_i) \tag{2.69}$$

2.4.1.3. Training by Back-Propagation

Until now we have only loosely discussed how ANN models such as MLP are trained to perform a given task. Given a large corpus of labelled data containing example inputs and their desired outputs, and a randomly initialized set of model weights, Rumelhart *et. al.* [Rum+85; Rum+95] propose a method of training ANN models by minimizing an objective function by gradient descent. For this training scheme we begin by extending our definition of the loss function to be an average loss over the N examples in our dataset (equation 2.70).

$$\mathcal{L} = \frac{1}{N} \sum_{n=1}^{N} \mathcal{L}_n \tag{2.70}$$

At each training step, we compute the average loss over all examples in the the dataset and use this value to compute the derivative of each trainable parameter that we would like to optimize w.r.t. the loss. For each parameter we update its value by stepping a small amount downhill in the opposite direction of the gradient of our parameters. The distance we step for each parameter is determined by scaling its gradient by a learning rate η which is tuned to give stable and consistent training performance. Equation 2.71 shows the general form of a gradient update for each of the trainable weight matrices and bias vectors described in equation 2.61.

$$V' = V - \eta \frac{\partial \mathcal{L}}{\partial V} \quad \text{where} \quad V \in \left\{ W^0, B^1, W^1, B^2, W^2, B^3 \right\} \tag{2.71}$$

To compute the gradient of our parameters w.r.t. the loss function we recursively apply the chain-rule to step backwards through the network, propagating the gradient calculation from one intermediate result to its predecessors until we have evaluated the gradients for all of the parameters we want to update. Recently, Automatic Differentiation [Nei10] frameworks have come into widespread use in popular programming frameworks such as Tensorflow [Aba+15]. These frameworks model arbitrary mathematical formulas as static computation graphs which know how to compute the derivatives of their constituent parts. For performance critical derivative calculations hand coded and optimized implementations can be specified, however, in many cases this is not necessary. By simplifying the process of defining the gradients of our models w.r.t. their parameters it is now easier than ever to experiment with complex and exotic model architectures.

A limitation of training by gradient descent is that in order for models to accurately learn strategies for complex tasks a large amount of training data, on the order of hundreds of thousands or millions of examples, is needed for the model to extract a robust and generalized strategy. This often causes severe limitations on memory complexity and model size when we try to implement our models on GPU devices. Stochastic Gradient Descent [Bot10] offers a solution to this problem by drawing random batches of training samples from the dataset without replacement and performing a gradient descent step for each batch of samples. When the dataset has been exhausted the process repeats, sampling batches from the dataset in different combinations and order without replacement. Each full pass of the dataset is generally referred to as an "epoch" of the training procedure. In this setting we can see that the standard gradient descent algorithm is simply an instance of stochastic gradient descent where the batch size is equal to the size of the dataset. Recent work has focused on the development of more advanced optimization algorithms which build off of stochastic gradient descent [TH12; KB14; Duc+11]. These methods draw inspiration from work in the physics domain, applying concepts such as momentum, velocity, and friction to structure gradient descent as a physical simulation of a mass sliding on a high dimensional surface.

2.4.2 Convolutional Neural Networks (CNN)

A major limitation of models such as MLP is that the number of weights needed grows proportionally to the product of the number of input and output features in each layer. When the input features are the pixels of images, potentially with multiple colour channels, the number of parameters can become infeasible to optimize over. The seminal work by LeCun *et. al.* [LeC+98] developed a powerful generalization on MLP type models, replacing individual scalar features with tensors (single channel images in the 2D case) and scalar entries in the weight matrices with convolutional kernels. This constrains the model such that individual pixels in the feature maps of one layer are only connected to a small local neighbourhood of pixels in the feature maps of the next layer. This greatly reduces the number of parameters needed, making the training of extremely large models that operate on signal, image, or volumetric data computationally feasible to train [Kri+12; SZ14; Sze+14; He+15a; Hua+16].

To convolve a single feature map a with a kernel K, the kernel is placed centred over each pixel of a in a sliding window over the image. As the kernel slides over each pixel, the value of the output pixel in a' is the summation of element-wise multiplication between each element in the kernel and the pixel value currently beneath it. Equation 2.72 shows the general form of convolution on a single channel image. The auxiliary variables mn are the offsets indices of the local pixel neighbourhood around the current pixel ij.

$$\begin{aligned} a' &= a * K \\ a'_{ij} &= \sum_{u=1}^{K_w} \sum_{v=1}^{K_h} a_{mn} K_{uv} \\ \text{where} \quad m &= (i - \frac{K_h}{2}) + (u-1) \\ \text{and} \quad n &= (j - \frac{K_w}{2}) + (v-1) \end{aligned} \tag{2.72}$$

The extension of the element-wise form of equation 2.59 from MLP to operate on feature maps, and using convolution rather than multiplication, can be seen in equation 2.73.

$$\begin{aligned} a_j^{l+1} &= f\left(B_j^{l+1} + \sum_{i=1}^{N^l} a_i^l * W_{ij}^l\right) \\ \text{where} \quad B^{l+1} &= \{1 \times N^{l+1}\} \\ \text{and} \quad W^l &= \{N^l \times N^{l+1} \times K_h^l \times K_w^l\} \end{aligned} \tag{2.73}$$

In this form, N^l input feature maps are transformed into N^{l+1} output feature maps. The bias vector B^{l+1} still contains only a single scalar additive gain for each output feature map, now applied to each of its pixels. However, the weights W^l are now a 4D tensor of convolutional kernels with size $N^l \times N^{l+1} \times K_h^l \times K_w^l$, where W_{ij}^l represents the $K_h^l \times K_w^l$ kernel convolved with the i^{th} input feature map as part of the computation of the j^{th} output feature map.

2.4.3 Discussion

Machine learning offers us a way to develop strategies for training systems to perform complex tasks and to extract meaningful information directly from the raw data of the problem domain. Early work on ANN led to the development of MLP [Ros57; Rum+85] models and the algorithms needed to fit them to training data such as stochastic gradient descent [Bot10] via back-propagation of error gradients [Rum+95] (section 2.4.1.3). These models were shown to be effective for both continuous domain prediction through regression (section 2.4.1.2) and in discrete predictions tasks such as classification (section 2.4.1.2).

Models such as MLP, which operate on scalar valued features suffer from scalability issues which make them infeasible for application on tasks with large numbers of input or output features. The extension and generalization of these models to Convolutional Neural Network (CNN), which replace scalar features with multi-dimensional feature maps and weight multiplications with kernel convolutions, has had a paradigm shifting impact on the focus of deep learning research. When provided with enough data from the task domain, these models have sufficient capacity to learn strategies for extracting meaningful features directly from the raw data. With the recent prevalence of high performance compute devices such as GPU this has allowed for the development of increasingly large and expressive models [Kri+12; SZ14; Sze+14; He+15a; Hua+16] that perform well on a variety of complex tasks.

2.5 Image Quality Assessment

When working with image data we often need to compare and contrast the differences between images, or to assign them with a measure of quality w.r.t. their intended values. Such measures of quality should ideally reflect the sensitivities of the Human Visual System (HVS) and the perceptibility of distortions, making them an inherently subjective measure.

Over the years, IQA measures have been proposed which approach this task from a number of different perspectives. Early work focused on the development of simple numerical divergences that compare values at each pixel within the input images and pool these divergences to give a single scalar value representing the average divergence. Recently, much work has been targeted at understanding and modelling the relationship between image distortions and the perceived changes in quality that are seen by human observers. These methods often approximate our sensitivities to effects such as contrast masking were colour intensities appear distorted by their surroundings, and effects where interference patterns in spatial frequency cause us to perceive structures that are not actually present with images.

IQA measures can be roughly classified into three categories, commonly referred to as FR-IQA, Reduced Reference Image Quality Assessment (RR-IQA), and NR-IQA.

2.5.1 Full-Reference Image Quality Assessment

In FR-IQA, distorted test images are compared to GT reference images which are known to be correct and distortion free. Formally a quality score of the form shown in equation 2.74 is computed where I is the image we wish to evaluated and R is the GT reference image. $\mathcal{E}$ is a function designed to measure the differences between the pixels of the test and reference images, or more generally between the local neighbourhoods around each pixel.

$$\mathcal{Q} = \mathcal{E}\left(I, R\right) \tag{2.74}$$

In many cases, simple numerical divergences such as Mean Squared Error do not accurately reflect the sensitivities of the HVS to image distortions, causing the reported error values to be a poor indicator of visual quality and fidelity. To resolve this issue, many methods have been proposed [CH07; DV+00; SB06; She+05a; WB02; Wan+04a; Wan+03; WL11; Zha+11c; BK16; BK16; Man+11] that use advanced techniques such as approximations of the HVS to identify image regions that human observers are more sensitive to. These methods are commonly applied at multiple spatial resolutions by decomposing input images through Multi-scale Geometric Analysis (MGA) methods. An in-depth review of prior FR-IQA methods and their design principles is given in section 4.2.

FR-IQA measures are most commonly used in the physically based rendering literature as a means of comparing the quality of images produced with competing rendering algorithms to GT reference images that are rendered to a significantly higher visual quality than test images. This is most commonly performed in either equal time or equal sample count comparisons to display the efficiency of computation and rate of convergence for the proposed methods.

An issue with this experimental design is that it assumes the availability of the GT to use as reference image in FR-IQA. When the reference image is the product of Monte Carlo rendering process we are faced with the scenario that our reference will only converge to the GT in the limit, as an infinite amount of computation is used in its creation. For any reference image computed using a finite amount of computation some magnitude of distortion will present. In chapter 4 we investigate the robustness of existing FR-IQA measures when we consider the scenario where the reference image is not a perfect representation of the GT. To this end we provide an in-depth study using an array of FR-IQA measures sampled from the literature together with a large dataset of images rendered with Monte Carlo rendering algorithms to varying degrees of quality.

2.5.2 Reduced-Reference Image Quality Assessment

In some circumstances we may not have access to the full GT reference image, instead only having an approximation of the GT. In these cases we can apply RR-IQA measures which work on the assumption that while the reference image does contain some magnitude of distortion, it is in general a good enough approximation of the GT to be used to evaluate the quality of a test image. Formally, an approximate quality score of the form shown in equation 2.75 is computed where $\hat{R}$ is an approximate reference image that is representative of the unknown GT R.

$$\mathcal{Q} \approx \mathcal{E}\left(I, \hat{R}\right) \tag{2.75}$$

In this configuration, $\mathcal{E}$ is a function which does not explicitly compare the values at each pixel or pixel neighbourhood, but rather compares the distribution of features extracted from the test image to the distribution of those features in the reference image [Tao+09; ER04]. This distinction allows for subtle distortions in the reference image to be accommodated as their effect on the overall distribution of extracted features should be small when the chosen features are sufficiently robust to the intended distortion domain. A review of RR-IQA methods and their design principles is given in section 4.2.

2.5.3 No-Reference Image Quality Assessment

Lastly, we consider the scenario where we would like to evaluate the quality of a potentially distorted image without knowing its true value. In NR-IQA test images are compared to a distribution just as with RR-IQA measures, however for NR-IQA compared distribution is not extracted from a reference image, but is instead a distribution learned from a corpus of distorted and clean images containing a specific class of distortions [Saa+10; MB10; Mit+12a; Kun+16a; She+05b; Mit+13a; Liu+16]. Formally, a predicted quality score of the form shown in equation 2.76 is computed where the function $\mathcal{E}$ contains parameters which are fit to a specific distortion profile extracted from representative image data. An in-depth review of prior methods in NR-IQA is given in section 5.2.

$$\mathcal{Q} \approx \mathcal{E}\left(I\right) \tag{2.76}$$

In the context of physically based rendering, NR-IQA measures give us a way of predicting the visual fidelity of rendered images before they have converged, without needing to know their true values. In chapter 5 we investigate the use of machine learning techniques such as CNN to develop NR-IQA measures trained directly on domain distortions using a large dataset of Monte Carlo rendered images. We develop novel objective functions and data augmentation schemes to enable the stable and robust training of such models and show that they can approximate, to within a high degree of accuracy, the quality assessment given by FR-IQA measures without having access to the GT reference image.

2.6 Summary

In this chapter we have reviewed the classical and recent works which lay the foundation for the contributions made in this thesis. The use of Bayesian methods to construct unbiased estimators of complex and high-dimensional integral equations is leveraged heavily in the fields of physically based rendering and machine learning.

Path tracing as a convergent solution to the rendering equation [Kaj86] (section 2.3.4) opened the door for the application of Monte Carlo sampling techniques throughout all aspects of the rendering process. From the use of simple Monte Carlo estimators (section 2.2) for the primary pixel estimates, to the construction of path-space samples using Markovian random walks (section 2.2.2), and the use of importance sampling (section 2.2.3) techniques to accurately draw reflected directions from the interactions of rays with physically based material models.

This was later extended upon through the development of bidirectional path tracing [LW93; Vea97] (section 2.3.6) which developed multiple-importance sampling (section 2.2.4) as a means of combining multiple, potentially biased, estimators to create a unified unbiased estimator with lower variance and well stratified sampling properties. From this work, several advanced rendering algorithms were developed which applied increasingly sophisticated sampling strategies in order to make effective use of information that was being discarded during the rendering process. Algorithms such as MLT [VG97], PSSMLT [Kel+02], MMLT [Hac+14], and ERPT [Cli+05; Bat05] (section 2.3.7) apply a variety of techniques based on MHMC (section 2.2.2.1) and HMC [Li+15] (section 2.2.2.2) to mutate path samples drawn through BDPT. This allows them to explore locally within hard to sample regions of the path-space, greatly improving rendering convergence.

A key stage in the rendering process is the evaluation of differences in perceived quality between images generated with different rendering algorithms. IQA (section 2.5) is an inherently subjective task which requires the development of systems that accurately model the sensitivities of the HVS to the types of distortions we expect to see in rendered images. When evaluating the relative performance of competing rendering techniques FR-IQA (section 2.5.1) is commonly employed, to compare images rendered with competing methods to GT reference images that are rendered to a significantly higher visual quality. In chapter 4 we provide a large ensemble study on the effects that distortions from Monte Carlo rendering processes have on the quality scores given by existing FR-IQA measures. Specifically, we study their robustness when we consider that the GT reference image is also the result of a Monte Carlo rendering process and may contain some magnitude of natural distortion, and we give recommendations on FR-IQA measures which are appropriate to use in this setting.

Before the rendering process has converged, and when the GT is not available, NR-IQA (section 2.5.3) methods can be used to predict the visual quality of images by comparing them to a known distribution of distortions extracted from a corpus of labelled images. In chapter 5 we investigate the use of deep learning applied to the NR-IQA. With this work we aim to faithfully predict a quality map for each pixel in a distorted image produced by Monte Carlo rendering methods without access to the unknown GT reference image. We develop CNN (section 2.4.2) architectures, objective functions, and data augmentation schemes that allow us to train our proposed models directly from a large dataset of images rendered with Monte Carlo rendering algorithms to varying levels of visual quality. We show that such models are capable of approximating robust FR-IQA measures to within a high degree of accuracy.

3 A modern C++ approach to High Performance Computing with MPI

In order to study the robustness of IQA measures when evaluating images from Monte Carlo rendering processes we required a large dataset of images containing varying degrees of distortion with which to perform our analysis. To generate this data we looked to HPC methods as a means of parallelizing our implementations of Monte Carlo rendering algorithms across distributed computing environments such as HPC Wales.

The development of these implementations, which used a combination of modern C++ and the C MPI framework, exposed a number of limitations and vulnerabilities of MPI with regards to type-safety and code complexity. This led to the development of a modern C++ wrapper framework around MPI, which aims to protect against these vulnerabilities and add higher level functionality not provided by the MPI framework.

3.1 MEL - The MPI Extension Library

MEL is a C++11, header-only library, being developed with the goal of creating a lightweight and robust framework for building parallel applications on top of MPI. MEL is designed to introduce no (or minimal) overheads while *drastically* reducing code complexity. It allows for a greater range of common MPI errors to be caught at compile-time rather than during program execution when it can be far more difficult to debug.

A good example of this is type safety in the MPI standard. The standard does not dictate how many of the object types should be implemented leaving these details to the implementation vendor. For instance, in Intel MPI 5.1 `MPI_Comm` objects and many other types are implemented as integer handles, `typedef int MPI_Comm`, to opaque data that are managed by the MPI run-time. A drawback with this approach is it causes compile time type-checking of function parameters to not flag erroneous combinations of variables. The common signature `MPI_Send(void*, int, MPI_Datatype, int, int, MPI_Comm)` is actually seen by the compiler as `MPI_Send(void*, int, int, int, int, int)`, allowing any ordering of the last five variables to be compiled as valid MPI code, while potentially causing catastrophic failure at run-time. In contrast, Open MPI 1.10.2 implements these types as structs which are inherently type-safe.

With MEL we aim to:

- Remain true to the underlying design of MPI, by keeping to an imperative function interface that does not fundamentally change the way in which the programmer interacts with the MPI run-time.
- To provide a type-safe, consistent, and unified function syntax that allows distributions of MPI from all vendors to behave in a common and predictable way at both compile-time and run-time.
- To be soluble, allowing the compiler to remove the abstractions MEL provides to achieve the same performance as native MPI code.
- To be memory efficient by minimizing the use of intermediate buffers whenever possible.
- To make use of modern C++ language features and advanced template meta-programming to both ensure correctness at compile-time and to generate boiler-plate values that programmers would otherwise have to provide themselves with native MPI code.
- To give higher-level functionality that is not available from the MPI standard such as deep-copy Semantics.

Work on MEL's deep-copy module was originally published in the journal PeerJ-CS [Whi+16] in November of 2016, by the thesis author alongside Prof Mark Jones and Dr Rita Borgo.

3.2 Static Safety and Implicit Parameter Deduction

As described above, in certain MPI distributions `MPI_Comm` objects and many other types are implemented as integer handles, `typedef int MPI_Comm`, to opaque data that are managed by the MPI run-time. MEL solves this issue by providing lightweight wrappers around all MPI handle types which provide the missing type safety and compile time parameter checking. These wrappers are stripped away by the compiler after type checking and have no impact on performance regardless of whether the underlying MPI distribution uses opaque integer handles or well defined C structures.

Minimal wrappers for all MPI defined handle types such as `MPI_Comm`, `MPI_Group`, `MPI_Request`, and `MPI_Datatype` are provided by MEL to give a consistent and robust programming interface.

Template meta-programming is used heavily throughout MEL to apply static type checking to ensure code correctness and in order to infer function parameters which normally have to be stated explicitly by the programmer. These checks and inferences are made at compile time and do not impact the efficiency of the compiled code. Rather, they simply remove possible sources of error that come from the programmer being tasked to manage the many interdependencies within their code.

A demonstrative example of this can be seen in the memory allocation routines of MPI and MEL. In MPI, the programmer is responsible for allocating the correct number of bytes for allocated memory, this requires scaling the number of elements desired by the size of the type being allocated. This is a simple but easy to miss detail in the code that the compiler has no means of distinguishing the correctness of; the result of which yields invalid memory access, but may not cause the program to outright halt, instead allowing the program to continue trying to execute in an undefined and unstable state. In MEL, template meta-programming is used to specify the type being allocated, and the conversion from element count to byte count is done implicitly using this information, removing the chance of programmer error.

These design patterns and parameter inferences are used throughout MEL and cover the entire MPI standard. MEL also provides helper functions which greatly simplify the creation of Derived Datatypes, the configuration of Process Topology communicators, and the addition of User Defined Operations for reduction style operations.

3.3 MEL - Deep-Copy

Message passing is an established communication paradigm for both synchronous and asynchronous communication in distributed or parallel systems. Using MPI with Object Orientated (OO) languages is not always an easy task — while control on memory locality and data distribution represent extremely valuable features, dealing with the ever growing and sophisticated features of OO languages can be cumbersome.

This problem is particularly challenging for data structures employing abstractions (e.g, inheritance and polymorphism) and pointer indirection, since transferring these data structures between disjoint hosts requires deep-copy semantics. For user defined objects MPI adopts shallow copy semantics, whereby default copy constructors and assignment operators perform shallow copies of the object leaving memory allocation, copy, and de-allocation to be the responsibility of the programmer, not the implementation. A similar policy is applied to MPI objects, represented as handles to opaque data that cannot be directly copied. Copy constructors and assignment operators in user defined objects that contain an MPI handle must either ensure to invoke the appropriate MPI function to copy the opaque data (deep-copy) or use a reference counting scheme that will provide references to the handle (reference counted shallow copy). Shallow copy is acceptable for shared-memory programming models where it is always legal to dereference a pointer with the underlying assumption that the target of member pointers will be shared among all copies. Users often require deep-copy semantics, as illustrated in Fig. 3.1, where every object in a data structure is transferred. Deep-copy requires recursively traversing pointer members in a data structure, transferring all disjoint memory locations, and translating the pointers to refer to the appropriate device location (also referred to as object serialization or marshalling), commonly used for sending complex data structures across networks or writing to disk. MPI has basic support for describing the layout of user defined data types and sending user-defined objects between processes [Mes14].

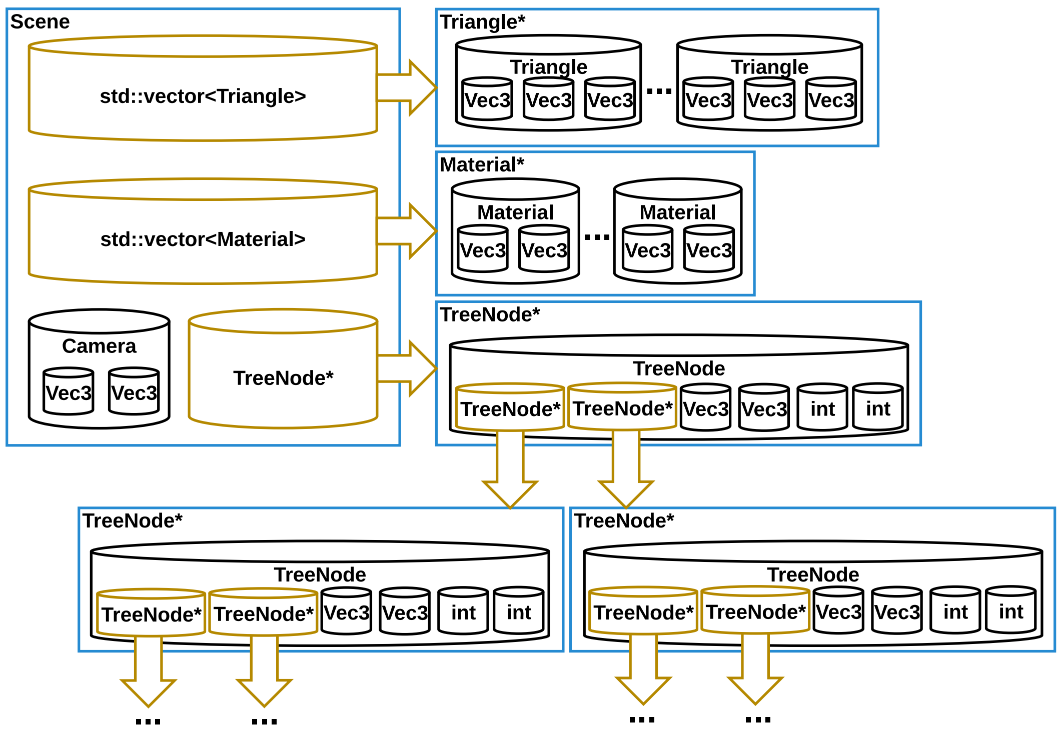


Fig. 3.1.: An example of a structure that requires deep-copy semantics. Arrows represent pointer traversals to disjoint regions of memory.

The directives we propose provide a mechanism to shape and abstract deep-copy semantics for MPI programs written in C++. Along with elegantly solving the deep-copy problem, this mechanism also reduces the level of difficulty for the programmer who only needs to express the dependencies of an object type, rather than explicitly programming how and when to move the memory behind pointers.

As a motivating example, we show that comparable performance can be achieved when using a simple and generic algorithm to implement deep-copy compared to hand coded native MPI implementations. We provide generic implementations of deep-copy semantics that can be easily applied to existing code to enable complex structured data to be deep copied transparently as either a send, receive, broadcast, or file access operation with minimal programmer intervention. The latter can also be used for the purpose of check-pointing when writing fault tolerant MPI code.

3.3.1 Related Work

Message passing as a style of parallel programming enables easy abstraction and code composition of complex inter-process communications. Existing MPI interfacing libraries [McC+96; Hua+06; BC15b] by default rely on the underlying standard shallow copy principle, where data contains no dependencies on memory outside the region directly being copied; and that where dependencies do exist that they are explicitly resolved by the programmer using subsequent shallow copies. However, this simplified model of communication comes at the cost of having to structure computations that require inter-process communication using low-level building blocks, which often leads to complex and verbose implementations [Fri+13]. Similar systems, such as the generic message passing framework [LL03] resolve pointers to objects, but do not follow dynamic pointers (data structure traversal) to copy complete complex dynamic structures possibly containing cycles.

MPI works on the principle that nothing is shared between processes unless it is explicitly transported by the programmer. These semantics simplify reasoning about the program's state [HS11] and avoid complex problems that are often encountered in shared-memory programming models [Lee06] where automatic memory synchronization becomes a significant bottleneck.

Autolink and Automap [Gou+98] work together to provide similar functionality. Automap creates objects at the receiver. Autolink tags pointers to determine whether they have been visited or not during traversal. The user must place directives in their code and carry out an additional compilation step to create intermediate files for further compilation. Extended MPICC [Ren07] is a C library that converts user-defined data types to MPI data types, and also requires an additional compilation. It can automate the process, but also in some cases requires user input to direct the process. Tansey and Tilevich [TT08] also demonstrate a method to derive MPI data types and capture user interaction via a GUI to direct the marshalling process.

Autoserial Library [GNA08] gives a C++ interface for performing serialization to file as binary or XML, or to a raw network socket as binary data. Their library also offers a set of convenience functions for buffering data to a contiguous array with MPI communications to move the data. Their method makes extensive use of pre-processor macros to generate boilerplate code needed for deep traversal of objects. For MPI, this library only handles the use case of fully buffered deep-copy in the context of `MPI_Send` and `MPI_Recv` communications.

OpenACC [Bey+14] tackles the deep-copy problem in the context of transferring structured data from host machines to on node hardware such as GPUs and Accelerators. Their approach is based on a compiler-implemented `#pragma` notation similar to OpenMP while our method is implemented as a header-only template library.

TPO++ [Gru+00] requires serialize and deserialize functions to be defined. The paper highlights good design goals which we also follow in this work.

Compared to the above approaches, we place much lighter requirements on the user and do not require additional signposting (usually implemented as preprocessor macros wrapped around variable declarations) that other methods require. We do not require an additional compilation step or GUI compared to the above as will be demonstrated in the following sections. We also provide an analysis of our approach. We explicitly demonstrate and analyze our approach on a wide variety of complex dynamic data structures. Our analysis shows that our approach has low time and memory overhead and also requires less user direction to achieve deep-copy. It provides this extra functionality at no loss of performance over hand coded approaches. We avoid the in-place serialize that some approaches utilize, resulting in our approach having a low memory overhead. We also evaluate our methods in comparison to Boost Serialization Library [Cog05] and demonstrate that Boost introduces a performance penalty which our method avoids. Boost also requires more intervention from the user / programmer to achieve the same capability. Therefore the main benefit of our approach over others is that it is a true deep-copy approach where the user only has to pass in the root object / node of the data structure.

In CHARM++ [KK93; Mil15] messages are by default passed by value, however CHARM++ provides support for deep-copy via definition of serialization methods for non contiguous data structures. It is a user task to define the proper serialization methods including the explicit definition of memory movement and copy operations.

If the serialization methods are implemented correctly for a user-defined type, a deep-copy will be made of the data being serialized. CHARM++ distinguishes between shared-memory and distributed-memory scenarios, where shared-memory data within a node can be directly passed by pointer. The programmer must explicitly specify the policy to be adopted by indicating if the data should be *conditionally packed* or not. Conditionally packed data are put into a message only when the data leaves the node. In an MPI environment processes within the same node do not share a common address space making such an optimization unavailable.

Generally the more desirable solution is to avoid deep-copy operations to maintain efficiency in message transmission. This is straightforward to achieve by converting user-defined types with pointer members to equivalent user-defined types with statically-sized arrays. This approach of restructuring and packing a data structure is often used by shared-memory programming paradigms where structures with pointers are manually packed and unpacked across the device boundary to reduce transfer costs for data structures used on the device.

When memory isolation (e.g., avoiding cross boundary references) is not a requirement other approaches might be possible. For operations executed within sequential or shared memory multi-core processors, hardware can be used more efficiently by avoiding deep-copy operations and relying instead on pointer exchange. This requires messages to have ownership transfer semantics with calls to send (pass) ownership of memory regions, instead of their contents, between processes [Fri+13]. In the context of the present work we do not focus on ownership passing but on the traditional approach of refactoring code. MEL provides an efficient and intuitive alternative to implementing object packing by hand. Porting an object type to use MEL deep-copy only requires adding a member function to the type containing directives that describe the dependencies of the type. In this case, the additional effort to rewrite data structures to allow communication using the standard MPI shallow copy principles is much larger, making refactoring an application to avoid deep-copy an undesirable solution.

Deep-copy semantics are not only relevant when dealing with inter-process communication. When recovering from process or node failure in fault tolerant MPI, applications often incur problems very similar to the ones dealt with by deep-copy operations. Fault tolerance plays an important role in high performance computing applications [HR15] and significant research has focused on its development in MPI [GL04; Vis+10; Bou15]. While the library itself does not provide explicit fault tolerance support, MPI can provide a standard and well structured context for writing programs that exhibit significant degrees of fault tolerant behaviour. Several approaches have been investigated in literature to achieve fault tolerance in MPI [GL04; Lag+14], with check-pointing being one of the most commonly used compared to more sophisticated approaches involving direct manipulation of the MPI standard to support fault tolerance [Fag+01; FD04], or modifying semantics of standard MPI functions to provide resilience to program faults.

In check-pointing, a process will periodically cache its work to disk so that in the event of a crash or node failure, a newly spawned process can load back the last saved state of the failed process and continue the work from there. When the data a process is dependent on is deep in structure, the implementation challenges associated with reading and writing the data to disk are the same ones encountered when handling the communication of such types. MEL provides support for fault tolerance by leveraging deep-copy semantics to transparently target file reads and writes in the same manner it handles the sending and receiving of inter-process communications.

3.3.2 When to use Deep-Copy

It is important that programmers be aware of the dangers of shallow-copying deep types without also resolving any dependencies of that type. For example, if an object contains a pointer and is copied by its memory footprint to another MPI process the value of the contained pointer on the receiver is now dangling and accessing the pointed to memory erroneous. Listing 3.1 shows an example of performing such an MPI shallow-copy when a deep-copy was needed. Throughout the rest of this chapter code listings with a shaded background will denote example use cases and usage of our algorithm while listings without shading denote the implementation details of the algorithm.

```
struct SomeStruct {
    int *ptr = nullptr, len = 0;
};

//----------------------------------------------------------------//
// On sending process
SomeStruct myVar;

// Allocate sub array
myVar.len = 10;
MPI_Alloc_mem(myVar.len * sizeof(int), MPI_INFO_NULL, &(myVar.ptr));

// Populate sub array with values...
for (int i = 0; i < myVar.len; ++i) myVar.ptr[i] = i;

MPI_Send(&myVar, sizeof(SomeStruct), MPI_BYTE, dst_rank, tag, comm);

//----------------------------------------------------------------//
// On receiving process
SomeStruct myVar;
MPI_Recv(&myVar, sizeof(SomeStruct), MPI_BYTE, src_rank, tag, comm);

```

```
// Error! myVar.ptr is a dangling reference to memory of the sending
// process!
```

Listing 3.1: User Example - Error from not resolving the data dependencies of an object when copying with MPI.

While accessing the pointed to memory is invalid, if we declare as a rule that if a pointer is not allocated it will be assigned to `nullptr` (and we strictly adhere to this rule), we can use the value of the dangling pointer to determine if an allocation needs to be made and data received on the receiving process. Listing 3.2 gives a corrected example of Listing 3.1, by deep-copying a struct containing a pointer safely using native MPI commands.

```
struct SomeStruct {
    int *ptr = nullptr, len = 0;
};

//--------------------------------------------------------------//
// On sending process
SomeStruct myVar;

// Allocate sub array
myVar.len = 10;
MPI_Alloc_mem(myVar.len * sizeof(int), MPI_INFO_NULL, &(myVar.ptr));

// Populate sub array with values...
for (int i = 0; i < myVar.len; ++i) myVar.ptr[i] = i;

// Send the footprint of the struct, allowing the receiver to check
// if ptr == nullptr or len == 0
MPI_Send(&myVar, sizeof(SomeStruct), MPI_BYTE, dst_rank, tag, comm);

// Resolve the dependency of the struct
if (myVar.ptr != nullptr && myVar.len > 0) {
    MPI_Send(myVar.ptr, myVar.len, MPI_INT, dst_rank, tag, comm);
}

//--------------------------------------------------------------//
// On receiving process
SomeStruct myVar;

// Receive the footprint of the struct so we can check if the array
// needs receiving
MPI_Recv(&myVar, sizeof(SomeStruct), MPI_BYTE, src_rank, tag, comm);

// Resolve the dependency of the struct
if (myVar.ptr != nullptr && myVar.len > 0) {
    MPI_Alloc_mem(myVar.len * sizeof(int), MPI_INFO_NULL, &(myVar.ptr));

    MPI_Recv(myVar.ptr, myVar.len, MPI_INT, src_rank, tag, comm);
}
```

Listing 3.2: User Example - Hand coded deep-copy using a dangling pointer from the sending process to determine if data needs to be received.

If an object which implements its own memory management through copy / move constructors and assignment operators, such as `std::vector`, is used, heap corruption can occur in a manner that can be difficult to debug. An example of this is shown in Listing 3.3. If a `std::vector` is copied by footprint its internal pointer, just like the raw pointer previously, is no longer valid. The vector class works on the assumption that its internal pointer is always valid, and that it needs to be de-allocated or re-allocated if any of the assignment, resize, or destructor functions are called. If the vector goes out of scope and its destructor is called the incurring segfault will often not be caught correctly by a debugger and the error will be reported "nearby", leaving the programmer to hunt down the true source of the error. Short of using the C++ placement-new operator to force the vector to be recreated without calling its destructor there is no way of "safely" recovering in this situation.

```cpp
struct SomeStruct {
    std::vector<int> someVec;
};

//------------------------------------------------------------------//
// On sending process
SomeStruct myVar;

// push_back into myVar.someVec a few times...
for (int i = 0; i < 5; ++i) myVar.someVec.push_back(i);

MPI_Send(&myVar, sizeof(SomeStruct), MPI_BYTE, dst_rank, tag, comm);

// Resolve the dependency of the struct
if (myVar.someVec.size() > 0) {
    MPI_Send(&(myVar.someVec[0]), myVar.someVec.size(), MPI_INT,
           dst_rank, tag, comm);
}

//------------------------------------------------------------------//
// On receiving process
SomeStruct myVar;
MPI_Recv(&myVar, sizeof(SomeStruct), MPI_BYTE, src_rank, tag, comm);

// If myVar goes out of scope we segfault!
//myVar.someVec.clear();              // Segfault!
//myVar.someVec.resize(10);           // Segfault!
//myVar.someVec.reserve(10);          // Segfault!
//myVar.someVec = std::vector<int>(); // Segfault!
// ect...

// It is safe to access .size() of the vector even if its internal
// pointer is invalid, we can use this to create a new vector in place,
// and to determine if we need to receive data.

// Force a new vector to be constructed at the memory address of the
// existing one without calling the existing vector's destructor.
new (&(myVar.someVec)) std::vector<int>(myVar.someVec.size());

// Resolve the dependency of the struct
if (myVar.someVec.size() > 0) {
    MPI_Recv(&(myVar.someVec[0]), myVar.someVec.size(), MPI_INT,
        src_rank, tag, comm);
}
```

Listing 3.3: User Example - The dangers of copying deep types by their footprint in memory without fixing them properly on the receiving processes.

3.3.2.1. Buffered vs. Non-Buffered

So far we have discussed methods for deep-copying object types by recursively traversing the data-structure and performing discrete message operations to resolve each dependency. While often small there is a performance cost associated with beginning and ending a communication between processes, and this cost is exacerbated when communication occurs between processes on different physical nodes connected by a network interface. In many cases it is beneficial to pack a deep structure into a contiguous buffer on the sending process and to transport it as a single communication, the buffer can then be received and unpacked to reconstruct the target data structure. Listing 3.4 demonstrates a variant on Listing 3.2 where data is packed into a buffer before being transported and unpacked on the receiving process.

While buffered deep-copy enables greater performance when communicating large structures made up of many small objects between processes, this speed comes at the cost of increased code complexity and limitations on the size of data that can be transferred. In the scenario where the data to be deep copied occupies more than half of the available system memory buffering into a contiguous buffer is no longer applicable as there is no remaining space in memory to allocate the buffer. Additionally, for programs that make many small allocations and deallocations during normal execution system memory can become fragmented, leading to a situation where there is more than enough available memory to allocate the buffer but it is split up in many small pieces meaning no one contiguous allocation can be made. In these scenarios there is no alternative but to perform a non-buffered deep-copy to move the data.

```
struct SomeStruct {
    int *ptr = nullptr, len = 0;
};

//------------------------------------------------------------------//
// On sending process
SomeStruct myVar;

// Allocate sub array
myVar.len = 10;
MPI_Alloc_mem(myVar.len * sizeof(int), MPI_INFO_NULL, &(myVar.ptr));

```

```
// Populate sub array with values...
for (int i = 0; i < myVar.len; ++i) myVar.ptr[i] = i;

// Calculate buffer size and allocate space
int buffer_size = sizeof(SomeStruct);
if (myVar.ptr != nullptr && myVar.len > 0) {
    buffer_size += (sizeof(int) * myVar.len);
}

char *buffer, *pos;
MPI_Alloc_mem(buffer_size, MPI_INFO_NULL, &buffer);
pos = buffer;

// Pack the struct itself to move non-deep members
memcpy(pos, &myVar, sizeof(SomeStruct));
pos += sizeof(SomeStruct);

// Pack the array of the struct
if (myVar.ptr != nullptr && myVar.len > 0) {
    memcpy(pos, myVar.ptr, sizeof(int) * myVar.len);
    pos += sizeof(int) * myVar.len;
}

// Send the buffer
MPI_Send(buffer, buffer_size, MPI_BYTE, dst_rank, tag, comm);

// Free the buffer
MPI_Free_mem(buffer);

//----------------------------------------------------------------//
// On receiving process
SomeStruct myVar;

// Calculate buffer size and allocate space
MPI_Status status;
MPI_Probe(src_rank, tag, comm, &status);

int buffer_size;
MPI_Get_count(&status, MPI_BYTE, &buffer_size);

char *buffer, *pos;
MPI_Alloc_mem(buffer_size, MPI_INFO_NULL, &buffer);
pos = buffer;

// Receive the buffer
MPI_Recv(buffer, buffer_size, MPI_BYTE, src_rank, tag, comm);


// Unpack the struct itself to move non-deep members
memcpy(&myVar, pos, sizeof(SomeStruct));
pos += sizeof(SomeStruct);

// Unpack the array of the struct
if (myVar.ptr != nullptr && myVar.len > 0) {
    MPI_Alloc_mem(myVar.len * sizeof(int), MPI_INFO_NULL, &(myVar.ptr));

    memcpy(myVar.ptr, pos, sizeof(int) * myVar.len);
    pos += sizeof(int) * myVar.len;
}
```

```

// Free the buffer
MPI_Free_mem(buffer);
```

Listing 3.4: User Example - Hand coded buffered deep-copy using a dangling pointer from the sending process to determine if data needs to be unpacked.

Buffering may also perform worse than non-buffered methods when the data to be deep copied consists of a small number of large objects, such as a struct containing several pointers to large buffers. In this case it may be detrimental to force the local copying of the large buffers into a single message only to unpack them on the receiving process when it would have been faster to transport them separately while taking the hit on the overheads associated with setting up multiple communications.

3.3.3 MEL - Deep-Copy Algorithm Design

Our algorithm is implemented in four parts: a top-level interface of functions for initiating deep-copy as send/receive, broadcast, or file-IO operation; a Transport-API of functions that describe how data is to be moved within a deep-copy operation; a set of transport methods that describe generically how to move a region of memory; and a hash-map interface for tracking which parts of the data structure have already been traversed. Figure 3.2 shows the architecture of our algorithm.

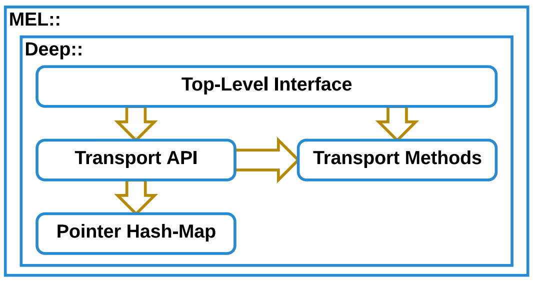


Fig. 3.2.: MEL Deep-Copy Architecture

In order to ensure correct memory management for deep structures the user must adhere to the following rules:

- Unallocated pointers are initialized to `nullptr`.
- Dynamic Memory must be allocated using `MPI_Alloc_mem` and freed using `MPI_Free_mem`, or the equivalent MEL calls:

```
T*   MEL::MemAlloc<T>(int len)
T*   MEL::MemAlloc(int len, T &value)
void MEL::MemFree(T *ptr)
```

```
T*   MEL::MemConstruct<T>(Args &&...args)
void MEL::MemDestruct(T *ptr, int len = 1)
```

- Pointers refer to distinct allocations. E.g. It is erroneous to have an allocation of the form `char *ptr = new char[100]` in one object, and to then have a weak-pointer into the array in subsequent objects: `char *mySubPtr = &ptr[50]` . In these situations it is best to store integer offsets into the array, rather than the pointer address itself.

3.3.3.1. Top-Level Interface

The top-level interface for our algorithm (Listing **??**) consists of functions for initiating a deep-copy as a send, receive, broadcast, or file access operation on a templated pointer `(T*)`, a pointer-length pair `(T*, len)`, an object reference `(T&)`, or an STL container `(std::vector<T>&, std::list<T>&)`. In the case of receiving methods (`Recv`, `Bcast`, and `FileRead`) the len parameter can either be passed by reference so that it can be modified to reflect the number of elements that were actually received, or captured from an integer literal or constant variable to provide a run-time assertion whether the correct number of elements were received. All methods are blocking and do not return until the entire data-structure has been transferred.

Buffered variants of the top-level interface initiate a local deep-copy to a contiguous buffer on the sender, this buffer is then sent as a single transport to the receiving processes where it can be unpacked. By decreasing the number of MPI communications or file accesses needed to transfer a deep structure significant reductions in latency can be achieved, at the cost of added memory overhead from packing and unpacking data before and after transport. In general, large structures of small objects (i.e. a tree with many nodes that are small in memory) benefit most from buffering while smaller structures of large objects (i.e. a struct containing large arrays) tend to benefit from non-buffered transport.

Another motivating reason for providing a non-buffered mechanism for deep-copy is the scenario where the deep structure occupies more than half of the available system memory. In such cases it is not possible to make a single contiguous allocation large enough to pack the structured data. An example of where this can happen is the use of MPI to distribute work to banks of Intel Xeon Phi Coprocessors which are exposed to the host system via a virtual network interface. While such hardware provides a large number of physical processor cores (60) on card memory is reduced (8-16GB). On larger systems with more available memory this is less likely to occur although the use of non-buffered methods may still be desirable for the reasons outlined above; and in any case, achieving low memory overhead is good practice.

3.3.3.2. Detecting objects that require Deep-Copy

Determining whether a given object is "deep" or not is performed at compile time using C++ template meta-programming to detect the presence of a member function of the form

```
template<typename MSG> void DeepCopy(MSG &msg)
```

that describes how to resolve the dependencies of a given object type. The template parameter `MSG` is a shorthand for `MEL::Deep::Message<TRANSPORT_METHOD, HASH_MAP>` where `TRANSPORT_METHOD` and `HASH_MAP` are types satisfying the constraints described in sections 3.3.3.5 and 3.3.3.6, respectively. A detailed example of the method used to detect the presence of a matching member function is given in Section 3.3.4.1.

The use of template meta-programming in C++ allows for the complete set of possible copy operations needed to transport a structure to be known at compile time, allowing the compiler to make optimizations that might otherwise not be possible if inheritance and virtual function calls were used. Template programming also opens up the future possibility of using more advanced C++ `type_traits` such as `std::is_pod<T>` (is-plain-old-data) and other similar type traits to help make informed decisions about how best to move types automatically at compile time.

Because we use the same function for sending / receiving, buffered / non-buffered, and for point-to-point / collective / file access communications we make use of a utility type, `Message`, that tracks which operation is being performed and where data is coming from or going to. The message object is created internally when one of the top-level functions is called and remains unmodified throughout the deep-copy.

3.3.3.3. Message Transport-API

The deep-copy function declares to our algorithm how data dependencies of a type need to be resolved in order to correctly rebuild a data structure on the receiving process. To keep the definition of this function simple the `Message` object exposes a small API of functions (Listing 3.5) that abstract the details of how data is sent and received between processes.

```
// Transfer a deep object. Only needed for deep types!
// Non-deep members are transported automatically
void Message::packVar(T &obj)

// Transfer a deep/non-deep pointer to len objects
void Message::packPtr(T *&ptr, int len = 1)

// Transfer a deep/non-deep pointer to len objects  where the pointer
// may also be referenced in other parts of the deep structure. (i.e.
// A graph structure where multiple nodes point to a shared neighbour)
void Message::packSharedPtr(T *&ptr, int len = 1)

// Transfer a std::vector of deep/non-deep objects.
void Message::packSTL(std::vector<T> &vec)
// Or a std::list (doubly linked list).
void Message::packSTL(std::list<T> &lst)

// Or use the shorthand operators
Message& Message::operator&(T &obj)
// ^ Calls packVar which is only defined for deep types
Message& Message::operator&(std::vector<T> &vec)
Message& Message::operator&(std::list<T> &lst)

// Only used in Top Level interface functions, these variants differ
// only from their standard counterparts (above) in that they do not
// assume the parent object has been transported as for the root
// object there is no parent.
void Message::packRootVar(T &obj)
void Message::packRootPtr(T *&ptr, int len = 1)
void Message::packRootSTL(std::vector<T> &vec)
void Message::packRootSTL(std::list<T> &lst)
```

Listing 3.5: MEL Implementation - Message Transport-API

Listing 3.6 gives an example usage of the `Message` Transport-API to move a complex data-structure. All of the functions provided work transparently with both deep and non-deep types, with the exception of `Message::packVar` which is intended only for the transport of deep types as non-deep member variables will be transported automatically. By comparison, Boost Serialization Library requires that all types except for language defined base types (e.g. `int`, `bool`, `double`) provide serialization functions regardless of whether they contain deep members, and that all member variables within the type (including non-deep members) are explicitly registered with the archive object.

```
struct SomeDeepStruct {
    // Non-deep members will be copied automatically.
    int a, b, c, len;
    SomeFlatStruct d;

    // Deep members must be declared in the Deep-copy function
    AnotherDeepStruct e, f, g;
    char *myArray = nullptr;
    GraphNode *mySharedPointer = nullptr;
    std::vector<int> v;
    std::vector<AnotherDeepStruct> w;

    template<typename MSG>
    void DeepCopy(MSG &msg) {
        // Pack a deep object by reference.
        msg.packVar(e);
        // A lighter syntax for non-pointer members.
        msg & f & g;

        // Transfer a char array of len elements.
        msg.packPtr(myArray, len);
        // Transfer a shared pointer that may also be pointed to elsewhere.
        msg.packSharedPtr(mySharedPointer);

        // Transfer a std::vector.
        msg.packSTL(v);
        // We can also transfer a std::vector or std::list using & syntax.
        msg & w;

        // In fact, we can simply replace all of the above code with:
        msg & e & f & g & v & w;
        msg.packPtr(myArray, len);
        msg.packSharedPtr(mySharedPointer);
    }
};
```

Listing 3.6: MEL Implementation - Registering dependencies using the Transport-API

3.3.3.4. An example copy

In essence, the deep-copy algorithm works by both sending and receiving processes entering a message loop or handshake with one another where they both expect to keep sending and receiving data until the entire structure has been transferred. The sending process determines how much data is to be sent, and this information is conveyed to the receiving processes transparently in such a way that when a receiving process determines there is nothing left to receive the sending process has returned.

```
// On sending process
int len = 10;
int *ptr = MEL::MemAlloc<int>(len);

// Fill ptr with some values... ptr = [0..len)
for (int i = 0; i < len; ++i) ptr[i] = i;

MEL::Deep::Send(ptr, len, dst_rank, tag, comm);

//---------------------------------------------------------------//
// On receiving process
int len;
int *ptr = nullptr;
MEL::Deep::Recv(ptr, len, src_rank, tag, comm);
// len = 10 and ptr now equals an address to len integers
// ptr = [0..len)
```

Listing 3.7: User Example - MEL deep-copy of non-deep type.

Listing 3.7 shows an example of using the deep-copy function to move an array of non-deep objects. Because the type, `int`, does not provide a member function for deep-copy the footprint of the array is sent in a single MPI message. On the receiving process memory is allocated into the pointer provided and the data is received.

```
struct SomeStruct {
    int len;
    int *array = nullptr;

    template<typename MSG> void DeepCopy(MSG &msg) {
        msg.packPtr(array, len);
    }
};

//-----------------------------------------------------------------//
// On sending process allocate array and subarrays
int len = 5;
SomeStruct *ptr = MEL::MemAlloc<SomeStruct>(len);

for (int i = 0; i < len; ++i) {
    // Allocate sub array
    ptr[i].len = i + 1;
    ptr[i].array = MEL::MemAlloc<int>(ptr[i].len);

    // Fill ptr[i].array with some values... ptr_i = [0..len)
    for (int j = 0; j < ptr[i].len; ++j) ptr[i].array[j] = j;
}

MEL::Deep::Send(ptr, len, dst_rank, tag, comm);

//-----------------------------------------------------------------//
// On receiving process
int len;
SomeStruct *ptr = nullptr;
MEL::Deep::Recv(ptr, len, src_rank, tag, comm);
// len = 5 and ptr equals an address to an array of 5 structures
// each having their respective lengths and subarrays
// ptr = [0..5) : { [0..1), [0..2), [0..3), [0..4), [0..5) }
```

Listing 3.8: User Example - MEL Deep-Copy of deep type.

An example of moving an array of structs containing pointers to dynamically allocated memory is given in Listing 3.8. In order to correctly reconstruct the data on receiving processes a deep-copy function has been implemented which tells the algorithm to copy a `char` array containing `len` elements. Because the type has a deep-copy function the receiving processes will allocate the memory for the array of structs and copy the footprint of the array as a single contiguous chunk resulting in non-deep member variables being transferred automatically. The receiving process makes the necessary allocations to receive its dependencies. Both sending and receiving processes will then loop over each element in their array and call the objects deep-copy function to resolve its data dependencies. If the struct contained variables which themselves required a deep-copy the algorithm would recurse on them until all dependencies are resolved. In this simple case, however, the struct contains a char array which does not require a deep-copy and as such the sub-array is transferred by allocating the needed memory and copying the entire sub-array as one contiguous chunk, as in Listing 3.7.

3.3.3.5. Transport Method

The `Message` object represents how our algorithm traverses the deep structure and ensures that both sending and receiving processes independently come to the same conclusion on what order objects are to be traversed with minimal communication. This traversal order is independent of, and identical for, all deep-copy operations. Because of this we template the `Message` object on a type that represents the specific nature of the data transportation we want to perform (i.e. `Message<TransportSend>` to perform deep-copy as an `MPI_Send` communication), allowing the same traversal scheme to be reused.

As a part of our implementation we provide transport methods for a wide variety of data movement scenarios:

TransportSend	Performs each transport call as a discrete `MPI_Send` communication.
TransportRecv	Performs each transport call as a discrete `MPI_Recv` communication.
TransportBcastRoot	Performs each transport call as a discrete `MPI_Bcast` communication, as a sender.
TransportBcast	Performs each transport call as a discrete `MPI_Bcast` communication, as a receiver.
TransportFileWrite	Performs each transport call as a discrete `MPI_FileWrite` operation.
TransportFileRead	Performs each transport call as a discrete `MPI_FileRead` operation.
TransportSTLFileWrite	Performs each transport call as a discrete `std::ofstream::write`.
TransportSTLFileRead	Performs each transport call as a discrete `std::ifstream::read`.
TransportBufferWrite	Performs each transport call as a discrete `std::memcpy` to a contiguous memory buffer.

TransportBufferRead	Performs each transport call as a discrete `std::memcpy` from a contiguous memory buffer.
NoTransport	This transport method acts as a sender but does not move any data. This method is used to implement the top-level interface functions for `MEL::Deep::BufferSize` which counts how many bytes need to be moved without performing any transportation.

Adding additional transport methods is as simple as implementing a class with a public-member function of the form

```
template<typename T> inline void transport(T *&ptr, const int len)
```

that describes how to move a region of memory, and a public-static-member variable `static constexpr bool SOURCE` which tells the compiler whether or not this is a sending or a receiving transport method. This boolean is important as it tells the `Message` object whether or not it needs to make allocations as it traverses the deep structure. The transport method should also store any state variables need to maintain the transport over the duration of the deep-copy. Such state variables may be but are not limited to an MPI communicator and process rank, a file handle, or a pointer to an array used for buffering.

3.3.3.6. Hashing shared pointers

When considering large structured data containing duplicate pointers the method used to track which parts of the structure have already been transported can have a significant impact on the traversal time. A hash-map is a natural choice for representing an unordered map between two pointers as it is efficient for random access lookups and insertions.

As with the transport method, the `Message` object is also templated on the hash-map to use for pointer tracking, namely `Message<TRANSPORT_METHOD, HASH_MAP = MEL::Deep ::PointerHashMap>`. This allows for the user to provide an adapter to their own implementation of a hash-map specifically optimized for pointers or to provide an adapter type to a third-party hash-map implementation.

To use a custom hash-map with any of the top-level functions simply override the default template parameter when initiating a deep-copy operation. E.g. `MEL:: Deep::Send<int, MyCustomHashMap>(ptr, len, dst, tag, comm);` where `MyCustomHashMap` exposes public-member functions of the form:

```
template<typename T> inline bool find(T* oldPtr, T* &ptr)
template<typename T> inline void insert(T* oldPtr, T* ptr)
```

These functions are templated on the pointer type, `T*`, so that user provided hash-map adapters are able to use this extra type information to optimize hashing if needed.

3.3.3.7. External Deep-Copy Functions

So far we have discussed the use of deep-copy functions and the Transport-API in cases where the deep-copy function was a local member function of the type being considered. In some use cases a structure may be defined in headers or libraries that cannot be modified easily (or at all). In such cases we still would like to be able to define the deep-copy semantics for the type without directly modifying its implementation. To enable this we provide an overload of all the functions in the Transport-API and top-level interface that take an additional template parameter that is a handle to a global-free-function of the form

```
template<typename MSG> inline void MyTypeDeepCopy(MyType &obj, MSG &msg)
```

that takes by reference an instance of the object to transport and a `Message` object to perform the deep-copy.

Listing 3.9, shows the usage of external free deep-copy functions with types needing deep-copy. `StructB` contains an internal member function for performing deep-copy, while `StructA` does not. Passing an instance of `StructA` to the top-level interface will result in incorrect results as its dependencies will not be resolved. By implementing a global-free-function that defines the deep-copy requirements of `StructA` we can then tell the top-level interface to explicitly use that function to resolve external dependencies of the type. If we provide an external free function for `StructB` which already has an internal deep-copy function, the internal function is ignored and the free function explicitly given is used.

```
struct StructA {
    std::vector<int> arr;
};

struct StructB {
    std::list<int> lst;

    // Internal - Local member deep-copy function
    template<typename MSG> void DeepCopy(MSG &msg) {
        msg & lst;
    }
};

// External - Global free deep-copy function
template<typename MSG> void StructA_DeepCopy(StructA &obj, MSG &msg) {
    msg & obj.arr;
}

// External - Global free deep-copy function
template<typename MSG> void StructB_DeepCopy(StructB &obj, MSG &msg) {
    msg & obj.lst;
}

// Example usage:
StructA sA;

MEL::Deep::Send(sA, dst, tag, comm);
// ^ Error! StructA does not have a deep-copy function

MEL::Deep::Send<StructA,
        MEL::Deep::PointerHashMap,
        StructA_DeepCopy>(sA, dst, tag, comm);
// ^ Correct. Uses external free function to perform the deep-copy

StructB sB;

MEL::Deep::Send(sB, dst, tag, comm);
// ^ Correct. Uses internal member function to perform the deep-copy

MEL::Deep::Send<StructB,
          MEL::Deep::PointerHashMap,
          StructB_DeepCopy>(sB, dst, tag, comm);
// ^ Correct. Uses external free function (overrides internal function)
//            to perform the deep-copy
```

Listing 3.9: User Example - Using external global-free-functions for deep-copy

The same rules apply for providing external free functions to the Transport-API. Listing 3.10, shows an example of this, where once again `StructA` is a deep type that does not provide an internal deep-copy function. `StructC` is also deep and contains a `std::list` of `StructA`. If the deep-copy function of `StructC` simply calls the ampersand operator or `Message::packSTL` function (Listing 3.10, lines 15,16) to transport the `std::list` then the instances of `StructA` will be transported incorrectly as a non-deep type. In the same manner as with the top-level interface the free function to use to deep-copy `StructA` is given explicitly to `Message::packSTL` so that it can correctly resolve the dependencies of the deep structure.

```
1  struct StructA {
2      std::vector<int> arr;
3  };
4
5  // External - Global free deep-copy function
6  template<typename MSG> void StructA_DeepCopy(StructA &obj, MSG &msg) {
7      msg & obj.arr;
}

struct StructC {
    std::list<StructA> lst;

    // Internal - Local member deep-copy function
    template<typename MSG> void DeepCopy(MSG &msg) {
        //msg & lst;        // <- Error - StructA has no deep-copy function
        //msg.packSTL(lst); // <- Error - ...
        msg.packSTL<StructA, StructA_DeepCopy>(lst); // <- Correct
    }
};

// Example usage:
StructC sC;
MEL::Deep::Send(sC, dst, tag, comm);
// ^ Correct. Uses internal member function of StructC and the external free
//              function StructA_DeepCopy for StructA.
```

Listing 3.10: User Example - Using external global-free-functions for deep-copy with the Transport-API

The option to use external deep-copy functions gives our method flexibility when we need to add deep-copy semantics to code that cannot be directly, or easily modified. However, this does not mean it will always be applicable as it requires intimate and low-level knowledge of the object's internal implementation and methods of allocation.

3.3.4 MEL - Deep-Copy Implementation Details

In the following section we provide a detailed discussion of the implementation of the MEL deep-copy algorithm.

3.3.4.1. Detecting the Deep-Copy Function

To detect whether the type under consideration contains a deep-copy function at compile time we make use of *SFINAE* (*Substitution Failure Is Not An Error*) to create a compile-time boolean test for the existence of a member function with the desired signature. We encapsulate the usage of this method into a templated shorthand that uses `std::enable_if` to give us a clean and concise method for providing function overloads for deep and non-deep types.

```
template<typename T>
struct HasDeepCopyMethod {
    // This pseudo-type does not exist unless type U has a member function of
    // the desired form: template<typename MSG> void DeepCopy(MSG &msg)
    template<typename U, void(U::*)(MEL::Deep::Message<NoTransport>&)>
```

```cpp
    struct SFINAE {};

    // If this succeeds Test<T> will be a function that returns char
    template<typename U> static char Test(SFINAE<U, &U::DeepCopy>*);
    // Otherwise Test<T> will return an int
    template<typename U> static int  Test(...);

    // We can now test if type T has the desired member function by seeing if
    // the result is the size of a char or an int.
    static const bool value = sizeof(Test<T>(0)) == sizeof(char);
};

// Shorthands for when implementing functions
template<typename T, typename R = void>
using enable_if_deep = typename
      std::enable_if<HasDeepCopyMethod<T>::value, R>::type;
template<typename T, typename R = void>
using enable_if_not_deep = typename
      std::enable_if<!(HasDeepCopyMethod<T>::value), R>::type;

// Example usage in function definitions
template<typename T> enable_if_deep<T> someFunc(T &obj) {
    std::cout << "Called with deep type!" << std::endl;
}

template<typename T> enable_if_not_deep<T> someFunc(T &obj) {
    std::cout << "Called with non-deep type!" << std::endl;
}

// A deep type
struct StructA {
    template<typename MSG> void DeepCopy(MSG &msg) {}
};

StructA sA;
someFunc(sA); // Called with deep type!

int i;
someFunc(i);  // Called with non-deep type!
```

Listing 3.11: MEL Implementation - Detecting the deep-copy function

Listing 3.11, shows an implementation of the technique used to conditionally detect member functions of template types at compile time. The overloads of `void someFunc (T &obj)` for when `T` is or is not a type with a deep-copy function allows us specialize our implementation for deep types while allowing them to share identical function signatures.

3.3.4.2. Transport-API Implementation

Next we describe the implementation of the Transport-API which specifies the traversal order our algorithm uses when performing deep-copy.

Message::packVar The `Message::packVar` function will call the deep-copy function of the given variable to resolve its dependencies. This function works on the assumption that local member variables of the object have already been transported when the parent object was traversed. It is for this reason that `Message::packVar` is only defined for deep types, as a non-deep type will have been transported automatically with the parent. In all of the following listings for the implementations of the Transport-API the overloads for non-deep types have been omitted for space.

```cpp
// Transport a deep object
template<typename D>
inline enable_if_deep<D> Message::packVar(D &obj) {
    // Assumes that the footprint of obj has already been transported
    obj.DeepCopy(*this); // *this == the Message object
}
```

Listing 3.12: MEL Implementation - `Message::packVar`

Message::packPtr When transporting dynamically allocated memory special care must be taken to correctly allocate memory on the receiving processes. Listing 3.13 shows the implementation of `Message::packPtr` for deep types. This function offloads its work to the `transportAlloc` helper function of the `Message` object. On the receiving process, `transportAlloc` will make an allocation of `len` elements of the given type before receiving the data. On the sending process `transportAlloc` is identical to `transport` and simply moves the requested data. For a deep type, `Message::packPtr` will then loop over all the received elements and call their deep-copy functions to resolve any dependencies.

```cpp
// Transport a deep pointer to len objects
template<typename D>
inline enable_if_deep<D> Message::packPtr(D *&ptr, int len = 1) {
    // On sender   - If (len > 0) and (ptr != nullptr) send the memory
    // On receiver - If (len > 0) and (ptr != nullptr) then overwrite the
    //               dangling ptr with a new allocation of len elements
    //               and receive the memory
    transportAlloc(ptr, len);

    // Followed by the recursion for deep types
    if (ptr != nullptr) {
        for (int i = 0; i < len; ++i) ptr[i].DeepCopy(*this);
    }
}
```

Listing 3.13: MEL Implementation - `Message::packPtr`

Message::packSharedPtr In complex structured data there is often a requirement for data to be self referencing. That is, one part of the deep structure may be pointed to from multiple other points within the structure. In these situations a naïve deep-copy algorithm would traverse the shared object within the structure multiple times allocating a unique copy of it with each visit. If the shared object is deep itself and points to one of its ancestors within the structure then the deep-copy algorithm will become stuck in an infinite cycle within the data, allocating new memory with each loop. To avoid this and to allow complex self-referential data to be transported, we provide the `Message::packSharedPtr` function shown in Listing 3.14. This method checks the given pointer against a hash-map of type $(pointer \rightarrow pointer)$ to determine if the pointed to memory has already been transported.

```
// Transport a deep shared pointer to len objects
template<typename D>
inline enable_if_deep<D> Message::packSharedPtr(D *&ptr, int len = 1) {
    // Save the original pointer in case we modify it
    D *oldPtr = ptr;

    // Is the given pointer already in the hash-map?
    // If so, set ptr equal to the pointer stored in the hash-map and return
    if (pointerMap.find(oldPtr, ptr)) return;

    // Same as for packPtr
    transportAlloc(ptr, len);

    // Insert the (newly allocated, on receiver) ptr into the hashmap with
    // the original dangling pointer (from the sender) as the key
    pointerMap.insert(oldPtr, ptr);

    // Followed by the recursion for deep types
    if (ptr != nullptr) {
        for (int i = 0; i < len; ++i) ptr[i].DeepCopy(*this);
    }
}
```

Listing 3.14: MEL Implementation - `Message::packSharedPtr`

During deep-copy the first time a shared pointer is passed to `Message::packSharedPtr` on both the sending and receiving processes, it is transported in the same manner as in `Message::packPtr` by calling `transportAlloc`. On the sending process the pointer is then inserted into the hash-map so it can be ignored if it is visited again. On the receiving processes the call to `transportAlloc` will have caused the dangling pointer from the sender to have been overwritten with the newly allocated pointer. This new pointer is inserted into the hash-map with the original (dangling) pointer as the key, so that next time the receiver is asked to transport the same dangling pointer it can simply look up and return the existing allocation.

When a shared pointer that has already been visited is passed to `Message::packSharedPtr` and it is found within the hash-map then the sending process can simply return as no memory needs to be transported; the receiving process uses the dangling pointer passed to it to retrieve the valid pointer that was previously allocated and transported the last time the shared pointer was visited. All interaction with the hash-map is performed through the `pointerMap.find` and `pointerMap.insert` functions of the `Message` object. These functions are further discussed in Section 3.3.4.4.

A nice property of this scheme is that the hash-map is never communicated and is constructed independently on both the sending and receiving processes. This means that for non-buffered communications the sender and receiver can traverse the structure in parallel (lock-step), and for buffered communications or buffered/non-buffered file-access the processes can traverse the structure independently.

Message::packSTL As part of the Transport-API we provide helper functions for moving common C++ STL containers. Listing 3.15 shows the implementation of `Message::packSTL` for C++ `std::vector`s of both deep and non-deep types. This is very similar to the implementation of `Message::packPtr` discussed previously with the slight difference that instead of making a new allocation on the receiving processes via `transportAlloc` we instead repair the internal pointer of the given `std::vector` by calling the placement-new operator to recreate the vector in place (as discussed in Listing 3.3). The implementation of `Message::packSTL` for other STL containers is conducted in the same way and is omitted here.

```
// Transport a std::vector of deep types
template<typename D>
inline enable_if_deep<D> packSTL(std::vector<D> &obj) {
    // .size() is safe to access even if the internal pointer is invalid
    int len = obj.size();
    // If this is a recieving process, repair the dangling internal pointer
    if (!TRANSPORT_METHOD::SOURCE) {
        // std::vector forces construction of elements
        new (&obj) std::vector<D>(len, D());

        // we need to call the destructor explicitly in case any resources
        // were acquired upon default construction of each element
        for (int i = 0; i < len; ++i) (&obj[i])->~D();
    }

    // Transport the data within the vector
    D *p = &obj[0];
    if (len > 0) transport(p, len);

    // Followed by the recursion for deep types
    for (int i = 0; i < len; ++i) {
        obj[i].DeepCopy(*this);
    }
}
```

Listing 3.15: MEL Implementation - `Message::packSTL` for `std::vector`

Message::packRootVar, Message::packRootPtr, & Message::packRootSTL Finally, we provide a set of functions to simplify the implementation of the top-level interface. Recall that `Message::packVar` is only defined for deep types and assumes that the object's footprint is always transported with the parent object. This is not the case for the top-level functions as no parent has been transported; in this case we must explicitly transport the object footprint regardless of whether it is deep or not.

A similar scenario occurs for pointers passed to the top-level interface. In order to avoid duplicating all of the top-level functions to account for whether the root pointer is shared we always insert it into the hash-map as this is a small constant overhead that does not affect performance. Recall from the implementation of `Message::packSharedPtr` that on the receiving processes the dangling pointer from the sender is used as the key into the hash-map. Because of this, for the root pointer we must explicitly transport the address-value of the pointer from the sender to the receiving processes so they can insert it into their hash-maps.

Finally, when considering STL containers passed to the top-level interface, receiving processes cannot query `.size()` of the container as its footprint was not previously transported. Instead, we explicitly transport the size of the container and call `.resize()` on the receiving processes.

```cpp
// Transport the footprint of a non-deep object
template<typename T>
inline enable_if_not_deep<T> Message::packRootVar(T &obj) {
    transport(obj);      // Transport the footprint
}

// Transport the footprint of a deep object and call its DeepCopy function
template<typename D>
inline enable_if_deep<D> Message::packRootVar(D &obj) {
    transport(obj);      // Transport the footprint
    obj.DeepCopy(*this); // Recurse on the deep structure
}

// Transport a root pointer to len deep objects
template<typename D>
inline enable_if_deep<D> Message::packRootPtr(D *&ptr, int len = 1) {
    // Explicitly transport the pointer value for the root node
    // so the it can be hashed correctly on recieving processes
    size_t addr = (size_t) ptr;
    transport(addr);
    ptr = (D*) addr;

    // Same as packSharedPtr, except we don't need to check the pointer
    D *oldPtr = ptr;
    transportAlloc(ptr, len);
    pointerMap.insert(oldPtr, ptr);

    // Followed by the recursion for deep types
    if (ptr != nullptr) {
        for (int i = 0; i < len; ++i) ptr[i].DeepCopy(*this);
    }
}

// Transport a root stl container to len deep objects
template<typename D>
inline enable_if_deep<D> packRootSTL(std::vector<D> &obj) {
    // Explicitly transport the length of the container
    int len;
    if (TRANSPORT_METHOD::SOURCE) {
        len = obj.size(); transport(len);
    }
    else {
        transport(len); obj.resize(len, D());
        for (int i = 0; i < len; ++i) (&obj[i])->~D();
    }

    D *p = &obj[0];
    if (len > 0) transport(p, len);

    // Followed by the recursion for deep types
    for (int i = 0; i < len; ++i) {
        obj[i].DeepCopy(*this);
    }
}
```

Listing 3.16: MEL Implementation - `Message::packRootVar`, `Message::packRootPtr`, & `Message::packRootSTL`

3.3.4.3. Transport Method Implementation & Usage

A transport method is a class which provides a single public-member function of the form

```
template<typename T> inline void transport(T *&ptr, const int len)
```

which defines how to move `len` objects of type `T` from a given pointer `ptr`. Listing 3.17 shows the implementation of the `TransportSend` transport method, which defines how move data using a discrete `MPI_Send` for each transport. An instance of a transport method carries any state needed to represent the data movement over the duration of the deep-copy. In the case of `TransportSend` the state needed to represent the transfer are the MPI rank of the destination process, a tag to use for the communication, and the MPI communicator over which the data will be transferred. For other transport methods the state may be a file handle, or a pointer to an array used for buffering.

```
class TransportSend {
private:
    // Store any state or resources needed to maintain this transport method
    const int pid, tag;
    const MEL::Comm comm;

public:
    // A transport method is either a source or a destination
    // This is known at compile time
    static constexpr bool SOURCE = true;

    TransportSend(const int _pid, const int _tag, const MEL::Comm &_comm)
                    : pid(_pid), tag(_tag), comm(_comm) {}

    // Transport function describes how to move data, in this case by
    // performing an MPI_Send
    template<typename T>
    inline void transport(T *&ptr, const int len) {
        MEL::Send(ptr, len, pid, tag, comm);
    }
};
```

Listing 3.17: MEL Implementation - Transport method for `Message`

Listing 3.18 shows the implementation of one of the top-level interface functions for performing deep-copy as an `MPI_Send` operation. A `Message<TransportSend>` object is instantiated, and the parameters from the function are transparently forwarded to the instance of the transport method within the `Message` object using `std::forward<Args>(args)`. After creating the message object the pointer to the deep structure can be transported by calling `Message::packRootPtr` from the Transport-API.

```
template<typename P, typename HASH_MAP = MEL::Deep::PointerHashMap>
inline enable_if_pointer<P> Send(P &ptr,
                 const int dst,
                 const int tag,
                 const Comm &comm) {
    // Arguments to the Message constructor are std::forward'd to the
    // TransportSend constructor
    Message<TransportSend, HASH_MAP> msg(dst, tag, comm);

    // Transport the deep-structure
    msg.packRootPtr(ptr);
}
```

Listing 3.18: MEL Implementation - Usage of a transport method in the top-level interface.

When performing a buffered deep-copy the data is first packed into a contiguous buffer on the sending process before being transported as a single operation to the receiving processes where the data can then be expanded back into the deep structure. Listing 3.19 shows the implementation of `BufferedSend` and `BufferedRecv` which make use of the `TransportBufferWrite` and `TransportBufferRead` transport methods.

```
template<typename P, typename HASH_MAP = MEL::Deep::PointerHashMap>
inline enable_if_pointer<P> BufferedSend(P &ptr,
                                          const int dst,
                                          const int tag,
                                          const Comm &comm) {
    // Compute the buffer size for the deep structure and transport it
    MEL::Deep::BufferedSend(ptr, dst, tag, comm, MEL::Deep::BufferSize(ptr));
}

template<typename P, typename HASH_MAP = MEL::Deep::PointerHashMap>
inline enable_if_pointer<P> BufferedSend(P &ptr,
                                          const int dst,
                                          const int tag,
                                          const Comm &comm,
                                          const int bufferSize) {
    // Allocate the buffer for packing
17      char *buffer = MEL::MemAlloc<char>(bufferSize);
18  
19      // Deep-copy into the buffer
20      Message<TransportBufferWrite, HASH_MAP> msg(buffer, bufferSize);
21      msg.packRootPtr(ptr);
22  
23      // Send the buffer in one message. Uses Message<TransportSend>
24      // bufferSize represents an upper bound on how much data there is to
25      // transport, msg.getOffset() gives us how much data was actually
26      // packed into the buffer
27      MEL::Deep::Send(buffer, msg.getOffset(), dst, tag, comm);
28  
29      // Clean up the buffer
30      MEL::MemFree(buffer);
31  }
32  
33  template<typename P, typename HASH_MAP = MEL::Deep::PointerHashMap>
34  inline enable_if_pointer<P> BufferedRecv(P &ptr,
35                                           const int src,
36                                           const int tag,
37                                           const Comm &comm) {
38      // Recieve the packed buffer in one message. Uses Message<TransportRecv>
39      int bufferSize;
40      char *buffer = nullptr;
41      MEL::Deep::Recv(buffer, bufferSize, src, tag, comm);
42      // bufferSize on the receiving processes is equal to msg.getOffset()
43      // on the sending process
44  
45      // Deep-copy out of the buffer
46      Message<TransportBufferRead, HASH_MAP> msg(buffer, bufferSize);
47      msg.packRootPtr(ptr);
48  
49      // Clean up the buffer
50      MEL::MemFree(buffer);
51  }
```

Listing 3.19: MEL Implementation - Usage of a buffered transport method in the top-level interface

The last parameter to buffered transport methods on sending processes is an integer value representing the byte size of the contiguous buffer to use for packing the deep structure. If this value is omitted an overloaded version of the function computes the upper-bound of the buffer size needed by calling `MEL::Deep::BufferSize` before forwarding its parameters to the main function overload.

Note on the sending process for a buffered transport that `msg.getOffset()` is used as the length parameter when transporting the buffer (Listing 3.19, line 21) and not the `bufferSize` parameter. This means that if the sender blindly requests a large buffer because it does not know the size of the deep structure exactly, but only a part of the buffer is filled, only the used part of the buffer will be transported to the receiving processes. In the scenario where the buffer size given was not large enough to complete the deep-copy a run-time assertion occurs.

3.3.4.4. Hash-Map Implementation

The `Message` object is templated on a hash-map type that exposes public-member functions of the form:

```
template<typename T> inline bool find(T* oldPtr, T* &ptr)
template<typename T> inline void insert(T* oldPtr, T* ptr)
```

This allows the user to provide an implementation of a hashing scheme optimized for pointers or to provide an adapter to a third-party hash-map implementation. One of the goals of MEL is to be portable and to not introduce external dependencies in the user's code; because of this our default hash-map implementation (Listing 3.20) is simply a wrapper around a `std::unordered_map` container between two `void` pointers.

```
class PointerHashMap {
private:
    // Hashmaps for storing pointers to types of any size
    std::unordered_map<void*, void*> pointerMap;

public:
    // Pointer hashmap public interface

    // Returns true if oldPtr is found in the hash-map and sets ptr equal to
    // the stored value. Otherwise returns false and ptr is un-altered
    template<typename T>
    inline bool find(T* oldPtr, T* &ptr) {
        // Is oldPtr already in the hashmap?
        const auto it = pointerMap.find((void*) oldPtr);

        if (it != pointerMap.end()) {
            // If so set ptr equal to the value stored in the hashmap
            ptr = (T*) it->second;
            return true;
        }
        return false;
    }

    // Insert ptr into the hashmap using oldptr as the key
    template<typename T>
    inline void insert(T* oldPtr, T* ptr) {
        pointerMap.insert(std::make_pair((void*) oldPtr, (void*) ptr));
    }
};
```

Listing 3.20: MEL Implementation - Default hash-map interface for `MEL::Deep::Message`

3.3.5 MEL - Deep-Copy Performance Benchmarks

For benchmarking we used the Swansea branch of the HPC Wales compute cluster. Nodes contain two Intel Xeon E5-2670 processors for a total of 16 physical cores with 64GBs of RAM per-node, connected with Infiniband 40 Gbps networking. Benchmarks were run using Intel MPI 4.1 and compiling under Intel ICPC 13.0.1

3.3.5.1. Case Study: Ray-Tracing Scene Structure

To evaluate the performance of our algorithms relative to the equivalent hand coded MPI implementations and to other libraries that offer deep-copy semantics such as Boost Serialization Library [Cog05], we used the example of deep-copying a large binary-tree structure between processes in the context of a distributed ray-tracer. A 3D scene (Fig. 3.3) is loaded on one process, consisting of triangular meshes, cameras, materials, and a bounding volume hierarchy to help accelerate ray-triangle intersection tests.

Fig. 3.3.: Utah Teapot mesh used for Benchmarks

For each experiment a scene was loaded containing increasing numbers of the classic Utah Teapot mesh. The scene structure was then communicated using the various algorithms and the performance measured by comparing the times spent between `MPI_Barrier`s before and after the communication.

Broadcast - MPI vs. MEL For this example just 4 lines of code calling the Transport-API were added to the BVH TreeNode and Scene structs (appendix A.1.1) to enable both buffered and non-buffered deep-copy using our algorithm.

By comparison, the hand coded MPI non-buffered method (appendix A.1.2) took 34 lines of code, and 70 lines of code for the MPI buffered algorithm (appendix A.1.3), (not including comments, formatting, or trailing brackets), where pointers, allocations, and object construction had to be managed manually by the programmer. Also these implementations only handled the case of `Bcast` operations, while the MEL version works transparently with all operations.

Despite its generic interface and minimal syntax, our algorithm performs almost identically with hand coded MPI implementations in fewer lines of code and a fraction of the code complexity. Relevant code for this example is given in appendix A.1.

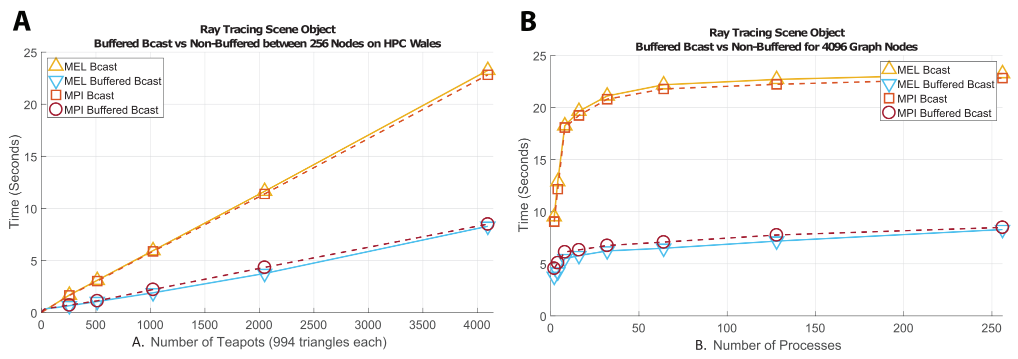


Fig. 3.4.: Time comparison of algorithms broadcasting large tree structures between processes within node and on separate nodes. MEL requires the addition of four simple lines of code which greatly accelerates programming time and vastly reduces the chance of user induced bugs.

Figure 3.4 (A) shows the resulting times from broadcasting increasingly larger scenes with each algorithm between 256 nodes on HPC Wales. We can see that the buffered methods that only send a small constant number of messages between processes are faster than non-buffered methods despite the added overheads from packing and unpacking the data. The scalability of our algorithm with respect to the number of MPI processes involved in the communication is only bounded by the scalability of the transport method itself. In the case of a broadcast operation, Fig. 3.4 (B) shows that varying the number of processes is of the same complexity as the underlying `MPI_Bcast` communication (logarithmic).

File Write / Read - MEL vs. Boost When fault tolerance is a concern one method for recovering from a failed process is to periodically cache the current state of the data being worked on to disk so that in the event of a failure the data can be reloaded on a new process (potentially on a different node) and the work continued from the point at which it was last saved. When the data needed to store the state of a process is deep we incur the same problems that arise during deep-copy. MEL implements file read and write operations for both buffered and non-buffered file access, utilizing the same user-defined deep-copy functions needed for the broadcast, send, and receive methods. For this experiment we also compared our performance to the Boost Serialization Library which is designed for saving and restoring structured data from file.

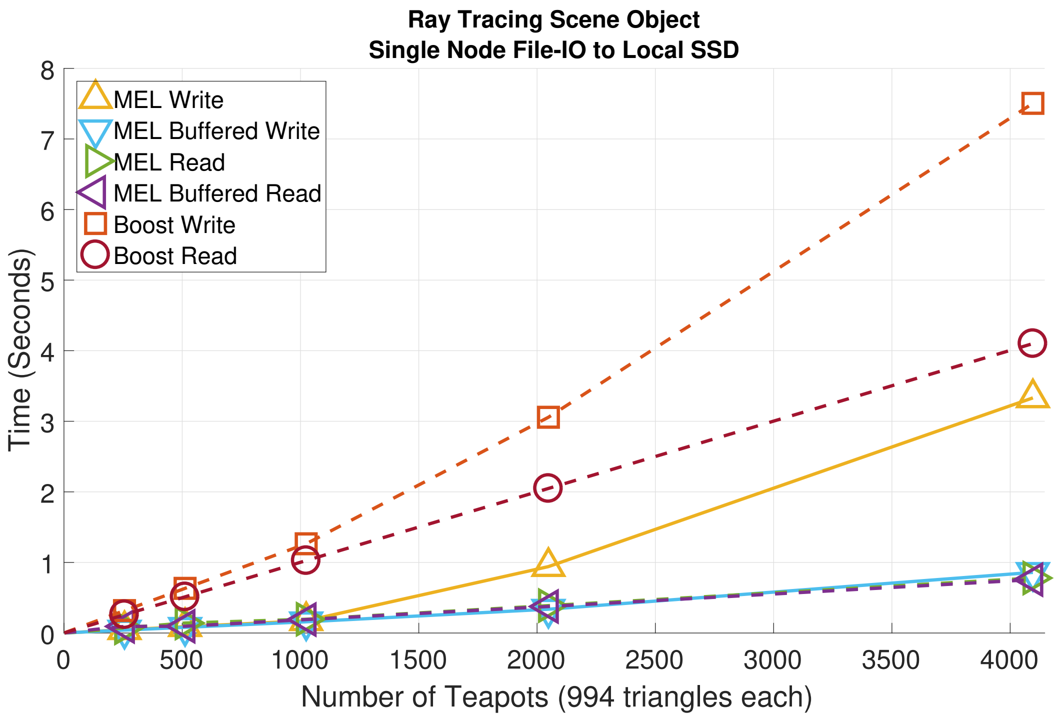


Fig. 3.5.: Time comparison of MEL to Boost Serialization Library for File Read/Write on a single node, to a within node Solid State Drive.

Figure 3.5 shows the results of using MEL to write/read a large tree structure to or from file. Unlike with MPI communications where MEL's buffered methods performed considerably faster than non-buffered variants due to the overheads from starting and ending network communications, with file access non-buffered reads perform almost identically to buffered methods. This is due to `std::fstream`'s use of an internal buffer to optimize file access, meaning that cost of starting and ending write/read operations is negligible compared to the cost of traversing the deep structure. While Boost Serialize also uses C++ streams its method of traversing the deep structure incurs significant overheads leading to poor and differing performance when reading and writing data. Finally, non-buffered writes perform slightly poorer then buffered writes due to file system having to allocate additional blocks as the file grows.

3.3.5.2. Case Study: Graphs with Cycles

In the previous example the implementation of `TreeNode` was simplified by the observation that tree nodes were only pointed to from a single parent. However, in many applications multiple objects may share a common child. To show how MEL copes with structures containing pointers to shared dependencies we use the example of communicating generic directed graph structures constructed in various connectivities (see Fig.3.6). Relevant code for this example can be found in appendix A.2.

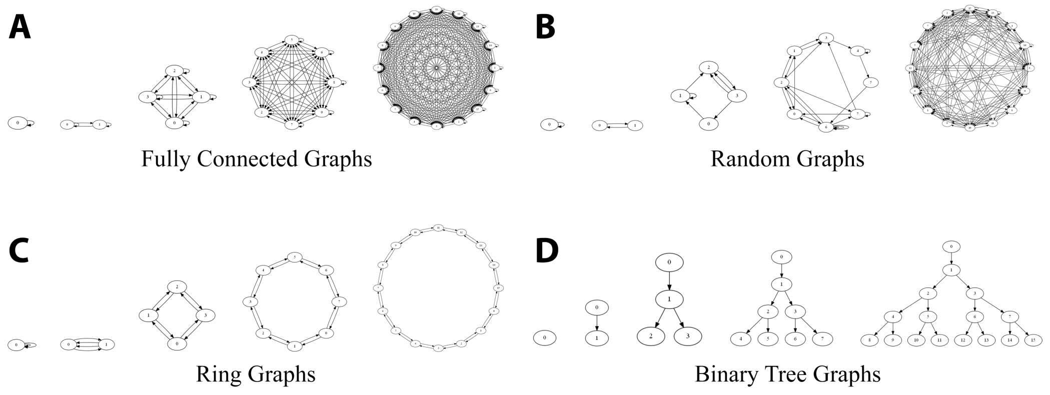


Fig. 3.6.: Graph Connectivities for $\{2^0, 2^1, 2^2, 2^3, 2^4, ...\}$ nodes.

Fully Connected Graphs Figure 3.7 shows the results for communicating fully connected graph structures of increasing size. In this example, n independent graph nodes will be traversed, each containing a list of pointers to all n nodes; during deep-copy the hash-map will be queried n^2 times and will grow to contain n entries. Compared to the previous broadcast example for the ray tracing case study (Section 3.3.5.1) where buffered communication showed better performance, with fully connected graphs we see the opposite effect. Non-buffered communication is consistently faster when the number of shared dependencies is high. Internally, shared pointers are tracked using a hash table to ensure that only distinct pointers are transported and duplicates linked correctly. Because of the overheads attached to insert and find operations on the hash table, when the number of shared dependencies is high the overhead from sending separate communications for each object in the structure is small compared to that of accessing the hash table. This has the effect of making the overhead from buffering the structure into a contiguous array for transport a bottleneck for deep-copy.

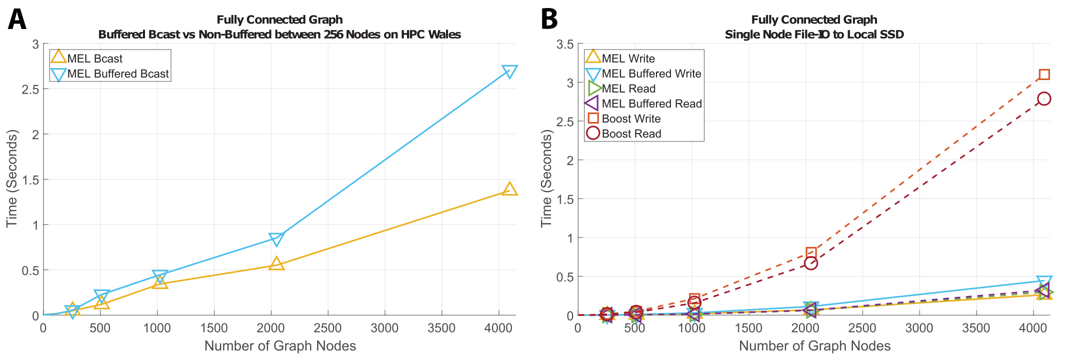


Fig. 3.7.: Time comparison for broadcast and file-IO operations on fully connected graph structures.

A similar trend is observed for file access, where non-buffered access is more efficient than buffered. In this example we also compare MEL to Boost Serialization library. Here shared pointer usage introduces significant overheads for Boost that our method avoids leading to significantly improved performance.

Random Graph Next we look at graphs with random connectivities. Figure 3.8 shows the results of communicating randomly generated graphs of different sizes. With this example, n independent graph nodes will be traversed, each containing a list of pointers to a random number of nodes (at least one); during deep-copy the hash-map will be queried between n and n^2 times and will grow to contain n entries. Again, we see that when the number of shared dependencies within the structure is large non-buffered communication performs consistently better than for buffered. We also see slightly better performance than with the fully connected graphs, showing that time complexity scales linearly with the number of graph edges. For file access the same trends emerge, where our method performs considerably faster than Boost Serialization.

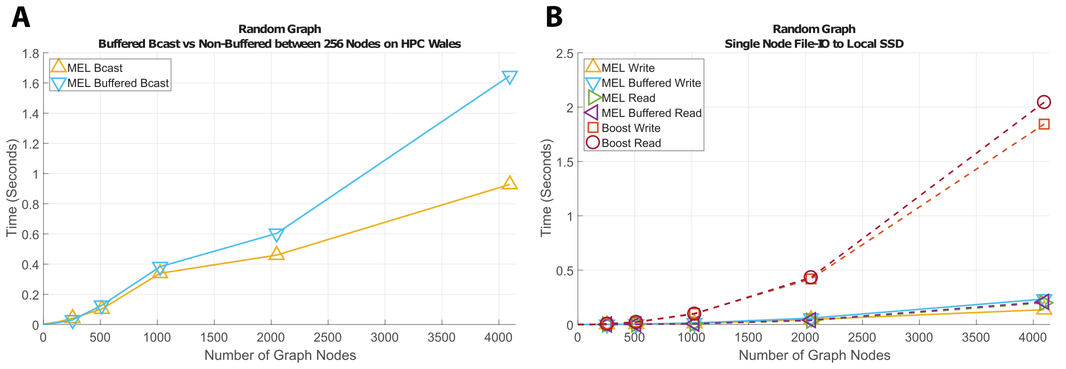


Fig. 3.8.: Time comparison for broadcast and file-IO operations on randomly connected graph structures.

Ring Graph A ring graph can be modelled as a doubly-linked list where the last element is connected back to the first element in the structure. For this example, n independent graph nodes will be traversed, each containing a list of two pointers to previous and next nodes; during deep-copy the hash-map will be queried $2n$ times and will grow to contain n entries. Figure 3.9 shows the results of communicating large ring structures. Because the number of shared edges is small we initially see that buffered communication is faster than non-buffered as with Section 3.3.5.1. As the number of graph nodes in the structure passes 2400 the amount of time needed to buffer the structure becomes larger then the overhead associated with starting and stopping separate MPI communications making the non-buffered method more efficient for larger structures. For file access we still see that our methods perform consistently faster than Boost's even when the number of shared dependencies is low.

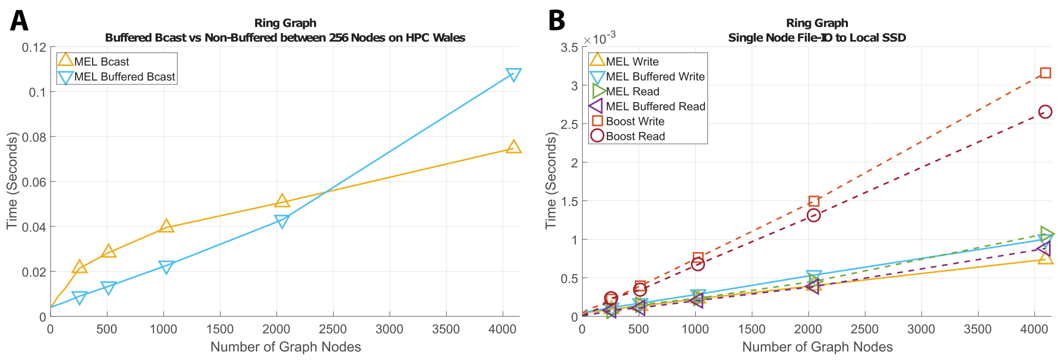


Fig. 3.9.: Time comparison for broadcast and file-IO operations on ring graph structures.

Binary Tree Finally, we look at the example of constructing a binary-tree-shaped graph where there are no shared dependencies. The generic container does not know this, and still must use `Message::packSharedPtr` to transport child nodes, meaning it still incurs the overheads of pointer lookup. In this example, n independent graph nodes will be traversed, each containing a list of one or two pointers to descending child nodes; during deep-copy the hash-map will be queried n times and will grow to contain n entries. Figure 3.10 shows the results of communicating binary trees of different sizes. Similarly to communicating ring graphs, buffered network communication is significantly faster non-buffered methods until the structure becomes large enough that buffering becomes the main bottleneck.

For file access the opposite is true, with non-buffered file access being slightly faster than buffered. We attribute this to `std::fstream`'s use of internal buffering, which renders the overheads from our fully buffered method unnecessary in this use case.

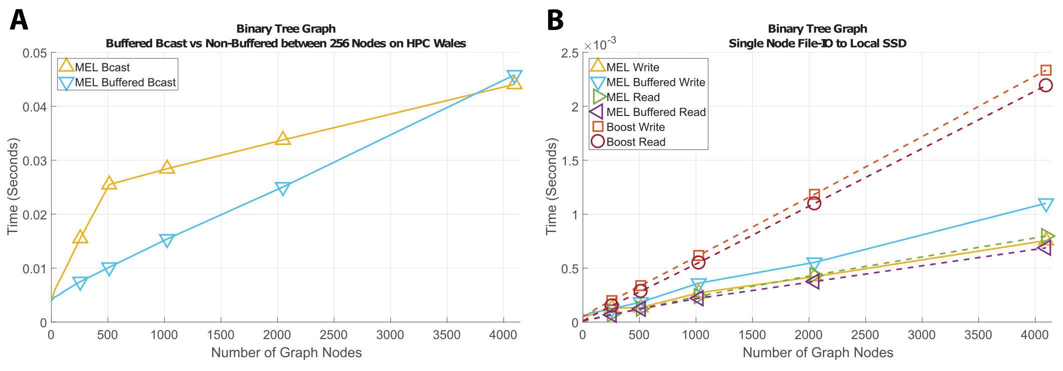


Fig. 3.10.: Time comparison for broadcast and file-IO operations on binary tree graph structures.

3.4 Conclusions

In this chapter we have presented MEL, our implementation of a C++ wrapper library for MPI being developed with the goal of creating a light weight, header-only C++ interface around the C-style functions exposed by the MPI-3 standard, with backwards compatibility for systems where only MPI-2 is available. MEL leverage's the power of modern C++ compilers through template meta-programming to add compile-time type checking and error detection, greatly improving the programmer's ability to write correct and bug free code.

As a part of MEL, we provide an implementation of deep-copy semantics that encapsulates both buffered and non-buffered methods for dealing with complex structured data in the context of MPI inter-process communication and file access. Users may choose shared versions for data structures containing cycles, or faster non-shared variants when the programmer knows that cycles will not be present in the data. We have shown that a generic implementation of such semantics can achieve like-for-like performance with hand-crafted implementations while dramatically reducing code complexity and decreasing the chance for programmer error. We also demonstrate our method is faster than utilizing Boost Serialization Library for the task of file-IO without requiring any changes to the user's code. MEL non-buffered methods provide a generic, low memory overhead, high performance (equal to hand crafted) solution to the deep-copy problem.

3.4.1 Further Work

As we continue the development of MEL there are several areas which present themselves for future work. MEL makes heavy usage of C++ template meta-programming to compute boiler-plate values, provide static analysis for error handling, and embed type-safety constraints on the underlying C MPI API at compilation-time. Due to the complexity of template meta-programming within modern C++ there are often multiple ways encode the same behaviour and type-safety at compilation-time. Because of this complexity, one direction of future work for MEL deep-copy is to further simplify the rules used to test if a structure needs deep-copying and to make the interface with STL containers more transparent. Additional transport methods can be added to MEL deep-copy allowing for deep structured data to be moved to/from accelerators such as GPU and co-processor devices through Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL).

Currently, MEL deep-copy only supports blocking communications and transport methods as the order of message passing is used to encode the traversal ordering and whether data-types within the structure require recursion to apply further deep-copy semantics. Applying non-blocking and asynchronous communications to this algorithm while maintaining the information ordering used throughout a deep-copy operation must be done carefully. Combining multi-threading with MEL deep-copy presents a strategy for further optimization. In hybrid MPI programs a single MPI process is instanced on each physical compute-node in the distributed environment, while threading within the node is performed using a secondary shared-memory threading model such as C++ `std::thread`, Open Multi-Processing (OpenMP), POSIX Threads (Pthreads), or another analogous framework. Deep-copy is not necessary between processes on the same compute-node as they share a common address space and can directly read from or write to one another's variables. When communicating off-node we can perform non-blocking asynchronous deep-copy by spawning an auxiliary thread within the same address space with which to perform the deep-copy. Within the auxiliary thread, deep-copy can be performed using the default synchronous algorithm as described in this chapter. Finally, synchronization of the distributed program to ensure that the deep-copy has completed can then be performed using a combination of MPI and the chosen threading framework's synchronization functions such as join, barrier, wait, and test.

MEL is open source and available on Github under the MIT license at:
`https://github.com/CS-Swansea/MEL`

4 Analysis of Error in Monte Carlo Rendered Images

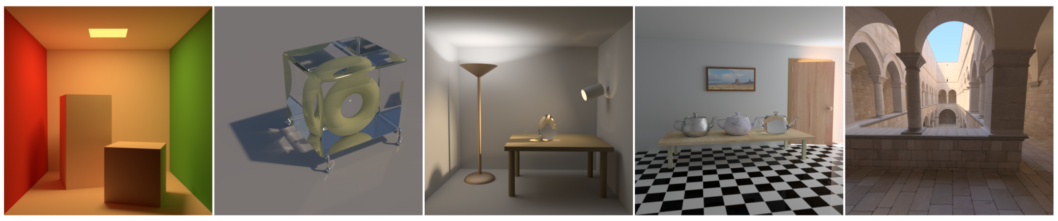


Fig. 4.1.: Scenes used for error analysis. From left to right: Cornell Box, Torus, Veach Bidir, Veach Door, Sponza.

Evaluating image quality in Monte Carlo rendered images is an important aspect of the rendering process as we often need to determine the relative quality between images computed using different algorithms and with varying amounts of computation. The use of a gold-standard, reference image, or ground truth (GT) is a common method to provide a baseline with which to compare experimental results. We show that if not chosen carefully the quality of reference images used for IQA can skew results leading to significant misreporting of error. We present an analysis of error in Monte Carlo rendered images and discuss practices to avoid or be aware of when designing an experiment.

In this chapter we leverage the MEL framework introduced in chapter 3 to implement Monte Carlo rendering algorithms in HPC environments, and use this to generate a dataset of rendered images containing varying amounts of natural distortion. Using this dataset we provide an ensemble study on the effectiveness of modern and classical IQA measures used in comparing digital images, and their robustness when evaluating images generated through stochastic rendering processes.

This work was originally published in the journal The Visual Computer and presented at the conference Computer Graphics International 2017 [Whi+17], by the thesis author alongside Prof Mark Jones and Dr Rafał Mantiuk.

4.1 Introduction

Monte Carlo rendering algorithms [Kaj86] allow for a plethora of photo-realistic and physically based lighting phenomena to be simulated; such as Indirect Illumination, Depth of Field, Participating Media, Caustics, and Physically Based Materials. A major problem is slow convergence — early termination of rendering can leave a large amount of undesirable noise in the images. Many methods have been proposed over the last three decades that attempt to minimize noise using as few samples as possible. These can be roughly classified into Path Space methods [LW93; VG97; Kel+02; JM12; Cli+05; Hac+14; Jen01; Hac+08b; Li+15; Doi+12] that use extra information available within the renderer to guide sampling in path space, and Image Filtering methods [RW94; JC95; TJ97; SW00; Kon+04; Hac+08a; PH10; Rou+12a; KS13; Bau+15; Kal+15; DJ13] that attempt to reconstruct the GT from a coarse un-converged image.

New methods need to be evaluated relative to existing ones. Often the increase in quality is not clear cut and is dependent on the test scenes used; while a strong improvement can be observed for suitable scenes it may be that others are ill-suited. This can cause the relative improvement in image quality to be small, though important nonetheless. In these cases where small improvements in quality are used to justify a method's performance, the accuracy of these measurements is important.

A commonly accepted methodology for evaluating images is **I:** to use a known GT which is noise free; **II:** that comparisons between the GT and test images use a metric such as Mean Absolute Error (MAE), Mean Square Error (MSE), Peak Signal to Noise Ratio (PSNR), or more recently Structural Similarity Index (SSIM) [Wan+04a]; and **III:** usually equal time and/or equal quality comparisons are reported as results. For these types of metrics to be effective it is a requirement that the reference image is correct and noise free.

In this chapter we present analysis of error reported when evaluating Monte Carlo rendered images. We look at the impact of reference image quality on results reported by IQA and highlight practices surrounding sample sets.

4.2 Image Quality Assessment (IQA)

Thorough analysis of 26 distance metrics applied to image data under varying distortions, spanning from pixel divergence methods such as MSE to those based on pixel correlation, structural features, and spectral measures [Avc+02], concluded that MSE most accurately described the level of distortion in images containing additive white noise. While for structural distortions such as blurring or block-artefacts measures based on edge similarity or weighted by models of the HVS were more robust. A similar study based on how closely different IQA compare to scores given by human test subjects was conducted [She+06]; with results indicating that the MSE based metrics can achieve comparable performance to more complex algorithms when images are distorted by additive white noise. From the literature, while there are differing opinions on its effectiveness, image quality metrics based on MSE appear to be most common and trusted when evaluating images corrupted predominantly by additive noise.

Relative quality in error assessment of the MSE and MAE metrics was investigated [WM05], showing that MSE's non-linear weighting with divergence can potentially lead to an exaggerated interpretation of error. Recent work, [CD14] has argued that MSE is in fact preferable over MAE when the error distribution is expected to fit a Gaussian model.

MGA works by decomposing image signals into sub-bands of spatial frequency [Gao+09]. In the IQA literature many MGA methods are used to extract structural information from input images. For the IQA considered in this work MGA appeared repeatedly in the form of Gaussian and Laplacian pyramids [BA83], Steerable pyramids [SP94], Contrast pyramids [Toe+89], Wavelet Transformations [Chu92], the Contourlet Transformation [DV05a], and the Wavelet Based Contourlet Transformation [VS05].

Study of the HVS has led to the creation of models that attempt to describe the likelihood that numerical distortions are actually perceivable by human observers under generalized viewing conditions. These models vary from a simple linear weighting of features in a multi-component error measure [Wan+04a; Wan+03], to models based on a non-linear Contrast Sensitivity Function [MV93] applied at multiple scales in an MGA decomposition.

Universal Quality Index (UQI) [WB02] splits image comparison into luminance, contrast, and structural components using statistics over the local neighbourhoods of each pixel. SSIM [Wan+04a] extends this idea by applying a linear weighting to each component using values derived from the HVS. The size of neighbourhood used in SSIM can alter its effectiveness at evaluating image quality; Multi Scale Structural Similarity Index (MS-SSIM) [Wan+03] addresses this by applying SSIM to each level of a Gaussian pyramid decomposition of images. Further discussion of the drawbacks of MSE based approaches compared to structural measures such as SSIM [WB09], shows that in many cases the same MSE score can be achieved for distorted images that are given vastly different quality assessments when viewed by human observers. In such cases, measures that consider structural features were significantly more robust and closely matched the assessments of human observers. More recently, an analysis of the mathematical properties of SSIM (and IQA based on it) compared to MSE derivative metrics showed they share several desirable qualities which make them well suited in the areas of parameter optimization and transform-domain noise reduction [Bru+12].

Information Weighting provides an interesting extension on several existing image metrics by applying a non-uniform weighting scheme to the pooling stage of IQA [WL11]. An information map is computed at each pixel that represents its relative importance with respect to visually perceivable distortions in the input. This is performed at multiple scales in a Laplacian pyramid decomposition of the input image. The resulting IW-MSE and IW-PSNR metrics perform comparably with several advanced IQA algorithms that take properties of the HVS into account. A third metric that benefits from information content weighting is IW-SSIM which extends the MS-SSIM algorithm making it an IQA that takes multi-scale and HVS information into account during both the distortion and pooling stages.

Visual Signal to Noise Ratio (VSNR) applies knowledge of the HVS to determine if image distortions would be noticeable to a human observer [CH07]. A spatially varying threshold on visible distortion is used to quickly determine if the comparison needs additional analysis which is performed by measuring perceived contrast and global precedence of structures within the images.

Noise Quality Measure [DV+00] fits input images to a HVS noise model using a contrast pyramid decomposition which has the effect of filtering out distortions the model is not sensitive to. Conventional SNR can then be applied to the model fitted images to provide a quality assessment.

Information Fidelity Criterion (IFC) [She+05a] and Visual Information Fidelity (VIF) [SB06] apply MGA by decomposing input images via the wavelet transformation. Statistics applied to the wavelet coefficients attempt to capture the mutual structural information between the inputs. By decomposing the images at multiple spatial sub-bands the effects of high frequency impulse noise can be directly measured. VIF can be considered a normalized variant on IFC [Bov09].

Recent work has been targeted at quantifying multi channel image distortions that do not present themselves when images are reduced to a single channel. FSIMc which is an extension of Feature Similarity Index (FSIM) [Zha+11c], considers images in the YIQ colour space [YK03]. This representation allows for luminance and chrominance features to be extracted and compared independently. Structural Contrast Quality Index (SC-QI) and Structural Contrast Distortion Metric (SC-DM) [BK16] perform feature extraction in the LMN colour space which has similar properties to YIQ. HDR-VDP-2 (Visual Difference Predictor) [Man+11] takes a different approach to multi channel image analysis by looking at the effects of inter-channel contrast masking in the sRGB colour space. The measure makes a per-pixel prediction on the likelihood a human observer would be able to detect the difference between reference and distorted images and is robust to a wide range of illumination conditions seen in natural images.

In Full Reference (FR) [She+06] IQA input images are compared against a GT image that is known to be correct. We also include two methods categorized as Reduced Reference (RR) IQA in our analysis. These methods are designed with the assumption that the reference image may contain some distortions but overall is still representative of the GT. Rather than directly measuring per-pixel deviation these methods measure the structural similarity of images by using the distribution of features extracted by MGA decomposition. The algorithms considered are based on the Contourlet transform [Tao+09] and Wavelet Based Contourlet Transform [ER04] respectively.

New IQA methods are often tested against image datasets such as Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], and IRCCyN/IVC [AB09] which couple distorted images with Mean Opinion Score (MOS) or Differential Mean Opinion Score (DMOS) on image quality given by human observers. In our exploration of the literature we have not found an analysis of how these algorithms (both FR and RR) perform when the reference image being used is the product of an un-converged rendering process, still containing impulse noise. We provide an extensive analysis here.

4.3 Computing Error

To compute an error value for a given image, it is compared to a GT that is known to be completely noise free. In computer graphics, error metrics that operate on single channel (gray-scale) images are most widely used in the literature with more recent research working to create IQA measures that operate on multi channel images. To extend single channel IQA metrics to multi channel (RGB) images the luminosity [And+96] of the RGB values is often used for error evaluation (equation 4.1). In this chapter all single channel IQA are performed on the luminosity channel of images.

$$\mathcal{L} = (0.2989 \cdot r) + (0.587 \cdot g) + (0.114 \cdot b) \tag{4.1}$$

In Monte Carlo rendering processes, images are rendered in High Dynamic Range (HDR) using physical units from the light transport simulation. This means the maximum value for a pixel is a physical upper-bound dependent on the composition of the scene being rendered. In order to present the image to a user it must be converted to a Low Dynamic Range (LDR) representation which accurately captures the distribution luminance within the original HDR image.

This raises the question of whether to perform IQA measures on images in the HDR or LDR representation. Most consumer displays have a colour-depth limited to 2^8 values per colour channel which suggests the application of IQA in LDR. Recently, high-end displays in the consumer domain and those used for viewing medical imagery have become available which have extended colour-depths in the range of 2^{12} values per colour channel. However, these devices are still displaying a LDR representation of the images as there is still a fixed upper-bound on colour intensity. As the colour-depth of display devices increases there is a growing need to be able to evaluate the quality of images that will be displayed with an arbitrary colour-depth, and additionally to be able to evaluate the quality of images directly in their physically based HDR representations. The application of IQA on HDR images is non-trivial as the lack of an upper-bound on pixel intensity makes it extremely difficult to design measures which are robust to a wide range of plausible natural inputs. In this chapter, due to dataset constraints and because the majority of IQA measures under test operate on LDR images, all IQA considered are computed on LDR images using their default hyper-parameters for an LDR configuration. In sections 4.6.1 and 5.5.1 we discuss how in the future we would like to revisit this to generate an expanded dataset of HDR images which can be used in a wider range of research applications.

While IQA measures can use a large variety of methods to compare image similarity they generally follow a two stage design pattern. In the first stage a distortion map is computed by comparing images at each pixel, or more generally at a local region around each pixel. Methods can use pixel divergence, structural similarity, statistical models for perceivable difference, or combinations of these and other measures. A secondary pooling stage then consolidates this information to a single representative value which most often takes the form of an average across image space, sometimes weighted further by additional perceptual information based on the HVS.

Other IQA based on natural image statistics leveraging decompositions such as the Wavelet transformation are more abstract in that image similarity is not compared on a per-pixel basis, but rather on an overall statistical measure of mutual information encoded by the decomposition coefficients.

In our experiment we chose IQA based on both of the above methodologies and those utilizing a variety of measures on per-pixel distortion to see how these various methods cope under the condition of a degraded and possibly non-representative reference image.

These metrics were chosen as they represent a wide range of techniques and design principles observed in the IQA literature. In multiple cases the chosen measures include the same or similar features in their algorithmic design but use them in different combinations. By analyzing the robustness of these similar measures we can drawn conclusions as to the effect of certain features or design principles on the quality of IQA. Table 4.1 shows a comparison of the techniques used by each of the considered IQA measures.

	Full Reference (FR-IQA)	Reduced Reference (RR-IQA)	Low Dynamic Range (LDR)	High Dynamic Range (HDR)	Single Channel	Multiple Channels	Multi-scale Geometric Analysis (MGA)	Human Visual System (HVS) Model
MSE	✓			✓	✓			
RMSE	✓			✓	✓			
MAE	✓			✓	✓			
PSNR	✓		✓		✓			
VSNR [CH07]	✓		✓		✓		✓	✓
NQM [DV+00]	✓		✓		✓			✓
VIF [SB06]	✓		✓		✓		✓	✓
IFC [She+05a]	✓		✓		✓		✓	✓
UQI [WB02]	✓		✓		✓			
SSIM [Wan+04a]	✓		✓		✓			✓
MS-SSIM [Wan+03]	✓		✓		✓		✓	✓
IW-MSE [WL11]	✓		✓		✓		✓	✓
IW-PSNR [WL11]	✓		✓		✓		✓	✓
IW-SSIM [WL11]	✓		✓		✓		✓	✓
Contourlet [Tao+09]		✓	✓		✓		✓	✓
WBCT [ER04]		✓	✓		✓		✓	✓
FSIM [Zha+11c]	✓		✓		✓		✓	✓
FSIMc [Zha+11c]	✓		✓			✓	✓	✓
SC-QI [BK16]	✓		✓			✓	✓	✓
SC-DM [BK16]	✓		✓			✓	✓	✓
HDR-VDP-2 [Man+11]	✓			✓		✓	✓	✓

Tab. 4.1.: A feature comparison of the 21 IQA measures sampled from the literature.

The metrics considered are:

- Single channel IQA: MSE, RMSE, MAE, PSNR, VSNR [CH07], NQM[DV+00], VIF [SB06], IFC [She+05a], UQI [WB02], SSIM [Wan+04a], MS-SSIM [Wan+03], IW-MSE, IW-PSNR, IW-SSIM [WL11], Contourlet [Tao+09] and WBCT [ER04] IQA, FSIM [Zha+11c];

- Multi channel IQA: FSIMc [Zha+11c], SC-QI [BK16], SC-DM [BK16], and HDR-VDP-2 [Man+11].

4.4 Our Experiment

Our experiment is motivated by practices we review in the literature. When examining reference images in some literature, we still see impulse noise, and we wish to explore the effect that reference image quality has on the results reported by IQA. Initially we performed our analysis using an image dataset rendered with a bespoke path tracing software developed for our research using the MEL framework introduced in chapter 3. We then validated our experiment by creating a new dataset of Monte Carlo rendered images using the widely trusted Mitsuba Renderer [Jak10] to give researchers confidence in the validity of our results. The analysis of this new dataset are the data we show in this chapter.

4.4.1 Monte Carlo Image Dataset Generation

We constructed an experiment where test scenes were rendered to increasing numbers of independent samples using each of the rendering algorithms considered. Images were generated on a 2^n samples per pixel (s.p.p.) sequence $\mathbb{N}$, for each of the test algorithms $\mathbb{A}$, and for each scene $\mathbb{S}$ (equation 4.2). Images were rendered in HDR using Mitsuba renderer, they were then tone-mapped to LDR using Mitsuba's Reinhard Tone-Mapping operator [Rei+02] which applies $\frac{L}{L+1}$ to the luminosity channel of the image in the LUV colour-space, and then gamma-corrected in the RGB colour-space using an exponent of $^1/_{2.2}$. The resulting LDR images stored as 24 bits-per-pixel RGB images are the ones used in this experiment.

$$\begin{aligned} \mathcal{N} &\in \mathbb{N} : \{2^n | 2 \leq n \leq \dots\} \\ \mathcal{A} &\in \mathbb{A} : \{\textit{PT}, \textit{BDPT}, \textit{PSSMLT}, \textit{MLT}, \textit{Manifold-MLT}, \textit{ERPT}, \textit{Manifold-ERPT}\} \\ \mathcal{S} &\in \mathbb{S} : \{\textit{Cornell Box}, \textit{Torus}, \textit{Veach Bidir}, \textit{Veach Door}, \textit{Sponza}\} \end{aligned} \tag{4.2}$$

This defines a set of images $\mathcal{I}_{\mathcal{SAN}}$ where $(\mathcal{SAN}) \in (\mathbb{S} \times \mathbb{A} \times \mathbb{N})$ parameterized by scene, rendering algorithm, and sample count with which to perform our analysis. For each scene we chose a rendering algorithm $\mathcal{A}^G$ to be the reference algorithm based upon its rate of convergence and the lack of structural artefacts at low sample counts. Path Tracing was chosen as the reference algorithm for the Cornell Box and Sponza scenes, while the caustic illumination in the Torus, Veach Bidir, and Veach Door scenes were better sampled using Bidirectional Path Tracing. Figure 4.2 shows exemplar image sequences from the dataset.

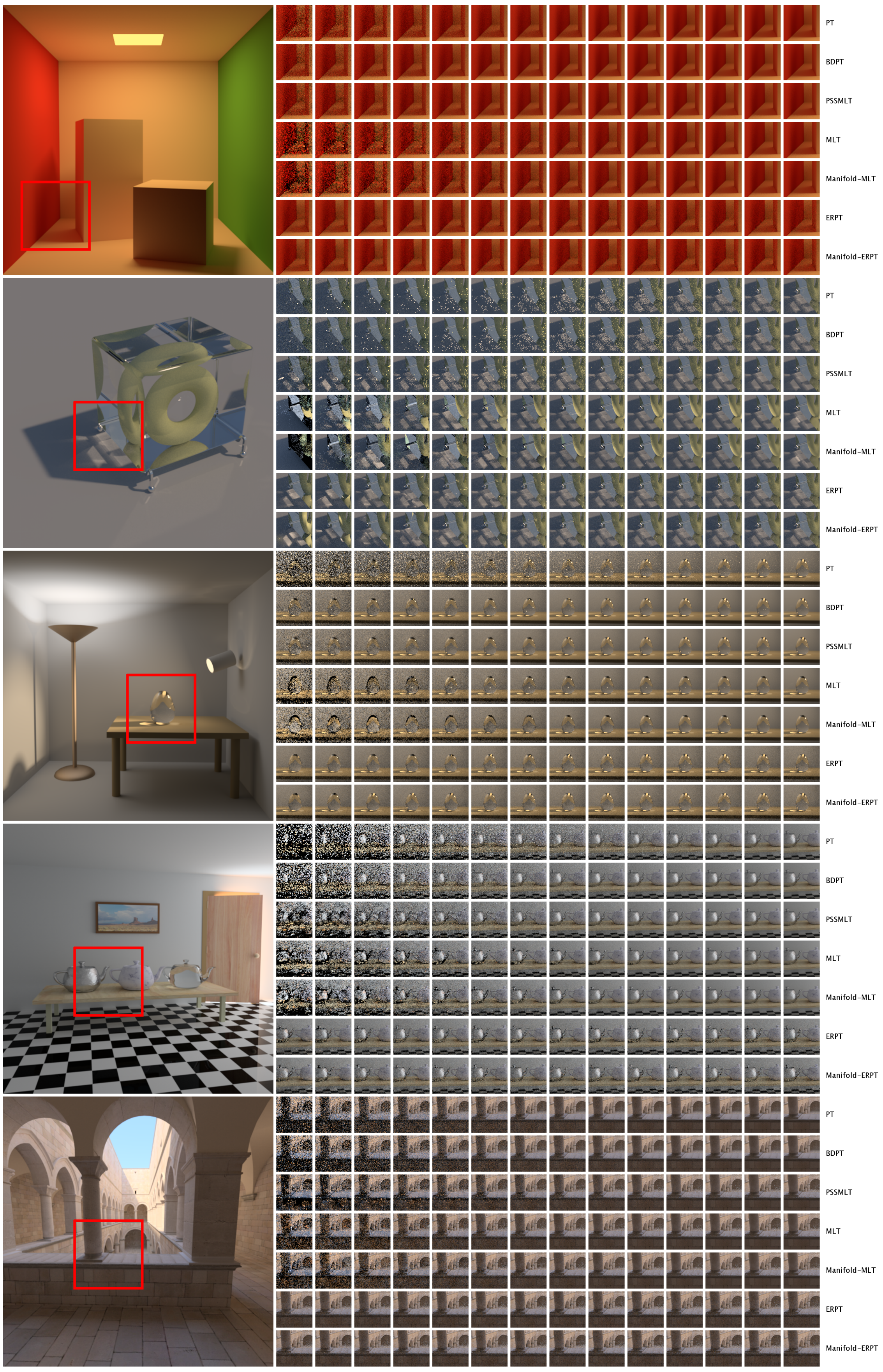


Fig. 4.2.: Image sequences from each scene, rendered with each algorithm. The left panels show the reference images used for IQA for each scene, where the superimposed red squares denote the crop window used in the right panels. Panels for each rendering algorithm for each scene are organized left-to-right on an ascending 2^n s.p.p. scale. For each sequence the first 14 images are shown.

For each error metric we compute the true error values to the GT reference image, and we wish to see how degrading the quality of the reference image affects these true error scores, (equation 4.3).

$$\begin{aligned}\mathcal{E} \in \mathbb{E} : \{&\mathit{MSE}, \mathit{MAE}, \mathit{PSNR}, \mathit{UQI}, \mathit{SSIM}, \mathit{MS\text{-}SSIM}, \mathit{IW\text{-}SSIM}, \mathit{IW\text{-}MSE}, \\ &\mathit{IW\text{-}PSNR}, \mathit{VSNR}, \mathit{Contourlet}, \mathit{WBCT}, \mathit{NQM}, \mathit{VIF}, \mathit{IFC}, \mathit{FSIM}, \\ &\mathit{FSIMc}, \mathit{HDR\text{-}VDP\text{-}2}, \mathit{SC\text{-}QI}, \mathit{SC\text{-}DM}\}\end{aligned} \tag{4.3}$$

4.4.2 Computing IQA Robustness

To do this we select the next highest sampled image as the reference image and recompute the error values. Only images with lower sample counts than the currently selected reference image are computed. By repeating this for all images in the sequence of the reference algorithm we end up with a triangular matrix for each error metric, algorithm, and scene; where one row represents the true error values, and the remaining rows represent the error values as the reference image is degraded. Formally, for all configurations $\mathcal{C}$ of an error metric, scene, and rendering algorithm we have a lower triangular matrix $\mathcal{M}^{\mathcal{C}}$ with elements indexed by the number of samples in the test image $\mathcal{N}_j$, and in the reference image $\mathcal{N}_i$, where each element is the error calculated between the reference image $\mathcal{I}_{\mathcal{S}\mathcal{A}^G\mathcal{N}_i}$ and the test image $\mathcal{I}_{\mathcal{S}\mathcal{A}\mathcal{N}_j}$ using an error metric $\mathcal{E}$ (equation 4.4).

$$\begin{gathered}\mathcal{M}^{\mathcal{C}}_{i,j} = \mathcal{E}\left(\mathcal{I}_{\mathcal{S}\mathcal{A}^G\mathcal{N}_i}, \mathcal{I}_{\mathcal{S}\mathcal{A}\mathcal{N}_j}\right) \\ \text{where } i > j \text{ and } \mathcal{C} = (\mathcal{E}\mathcal{S}\mathcal{A}) \;\; \forall \mathcal{C} \in (\mathbb{E} \times \mathbb{S} \times \mathbb{A})\end{gathered} \tag{4.4}$$

4.4.2.1. Log Accuracy Ratio

To compare the degraded error values to the true values we use Log Accuracy Ratio, $\ln \mathcal{Q}$ [Tof15], which measures the difference between an observed and expected value. Because the IQA measures considered report their quality results on different output scales we need a measure of relative change. Several such measures a widely used, most notably Percent Error. However, an issue with many of these measures is that the reported relative changes are asymmetric between positive and negative change. That is to say, if an predicted value P is 25% greater than the actual value A, then the actual value A is 20% less than the predicted value P, it is not 25% less (equation 4.5). This asymmetry makes it difficult to accurately compare and contrast relative performance when both under- and over-estimation are equally undesirable.

$$\frac{P - A}{A} \neq -\frac{A - P}{P} \quad \text{where} \quad P \neq A \tag{4.5}$$

The value Q is defined as the relative accuracy between the predicted value P and actual value A, $\frac{P}{A}$. This value on its own is still asymmetric, however it can be made symmetric by simply taking a logarithm of the ratio. Logarithms of ratios can be rewritten as the subtraction of two logarithms, through this transformation we can show that symmetry is preserved through the equality shown in equation 4.6.

$$\begin{aligned}\ln\left(\frac{P}{A}\right) &= \ln(P) - \ln(A) \\ &= -\left[\ln(A) - \ln(P)\right] \\ &= -\ln\left(\frac{A}{P}\right)\end{aligned} \tag{4.6}$$

Because both positive and negative values occur frequently within our data, we chose to use $\ln Q$ to show the relative change in reported error between different IQA measures. Figure 4.3 shows a visual depiction of the asymmetry present in Percent Error, and how this is avoided by using $\ln Q$.

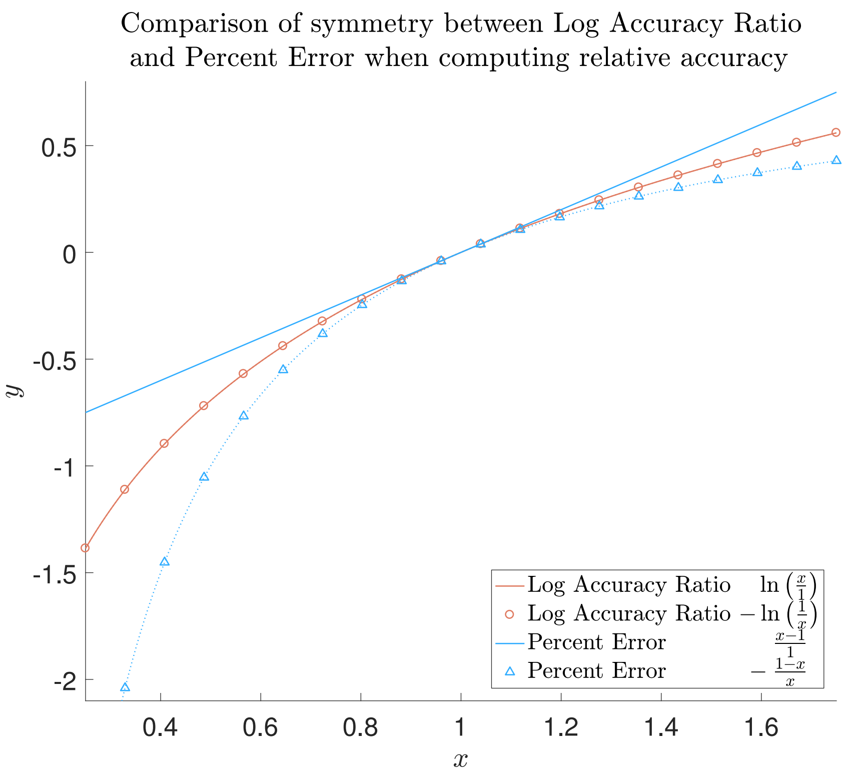


Fig. 4.3.: A visual comparison of the difference in symmetry between Log Accuracy Ratio and Percent Error. Both measures compute relative change between and observed and expected value. For both methods we fix the expected value to be 1 and sweep the observed value from -0.25 to 1.75, plotting the response from each function (solid blue and orange lines). We then swap the inputs, fixing the observed value to be 1 and sweeping the expected value across the same range and negate the result (dotted lines with orange and blue markers). We can see that Percent error is not symmetric as the flipped and negated inputs yield a different result. Log Accuracy Ratio does maintain symmetry, and both curves lie on top of one another.

This is applied to our triangular matrices by taking the natural logarithm of the values in each column divided by the true value in the $|\mathbb{N}|^{th}$ (bottom) row. This gives a matrix where the bottom row are zeros (referring to the $\ln \mathcal{Q}$ of true values versus themselves) and subsequent rows represent the quality of error evaluations as the reference image is degraded. Formally from the matrix $\mathcal{M}^{\mathcal{C}}$ for each configuration in the ensemble we define an equally sized matrix $\mathcal{P}^{\mathcal{C}}$ with elements defined by equation 4.7.

$$\mathcal{P}^{\mathcal{C}}_{i,j} = \ln\left(\frac{\mathcal{M}^{\mathcal{C}}_{i,j}}{\mathcal{M}^{\mathcal{C}}_{|\mathbb{N}|,j}}\right) \quad \text{where } i > j \text{ and } \mathcal{C} = (\mathcal{ESA}) \ \ \forall\mathcal{C} \in (\mathbb{E} \times \mathbb{S} \times \mathbb{A}) \tag{4.7}$$

Where $\mathcal{P}^{\mathcal{C}}$ has positive values this shows the IQA under test has **overestimated** the amount of error while negative values show the error was **underestimated**.

4.5 Discussion

For all scenes, rendering algorithms, and error metrics there are 735 separate $\mathcal{P}^{\mathcal{C}}$ matrices in the dataset. We present the full results in the supplementary material. Tables 4.2a-4.2e show $\mathcal{P}^{\mathcal{C}}$ for the Cornell Box scene rendered with Bidirectional Path Tracing, and tables 4.3a-4.3e show $\mathcal{P}^{\mathcal{C}}$ for the Veach Door scene rendered with Energy Redistribution Path Tracing, using error metrics VIF (top), MS-SSIM, SC-QI, HDR-VDP-2, and MSE (bottom) for both sets. A strong increase in values is visible for MSE showing that overestimation increases as the number of samples in the reference image decreases to the number of samples in the test image. The increase in misreporting also appears for VIF as a strong underestimation. MS-SSIM and SC-QI also exhibit underestimation but at a significantly lower magnitude. HDR-VDP-2 shows both under and overestimation at magnitudes comparable to VIF. Tables 4.4a-4.4e show $\mathcal{P}^{\mathcal{C}}$ for the Torus scene rendered with Metropolis Light Transport, using error metrics PSNR (top), NQM, VSNR, MAE, and Contourlet IQA (bottom).

To condense this to a manageable set of results table 4.5 displays the maximum magnitude of misreporting within a defined region of each matrix where the columns have been ordered left to right according to the average magnitude of under or overestimation for each error metric. The maximum magnitude is underlined in each table of results (tables 4.2a-4.2e, 4.3a-4.3e, 4.4a-4.4e, and supplementary material). The region is defined for reference images having sufficient samples that they exhibit good visual convergence as considered by a human observer. Reference images outside this region have lower sample counts and consequently an unacceptable amount of visible noise. The higher sample reference images are of sufficient visual quality that they are representative of image comparisons that are typically seen in the literature when evaluating rendering algorithms. We signify the start of this region in each table by a horizontal rule. Figure 4.4 shows the construction of table 4.5 w.r.t. a selection of $\mathcal{P}^{\mathcal{C}}$ configurations.

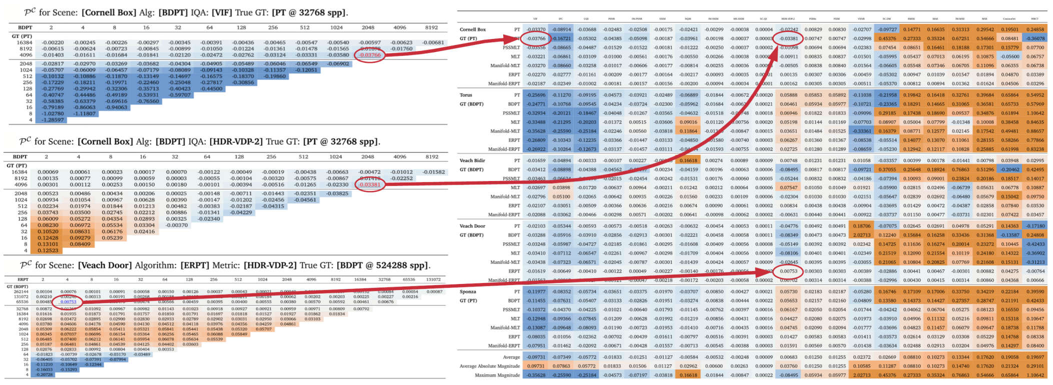

$\mathcal{P}^{\mathcal{C}}$ for Scene: [Cornell Box] Alg: [BDPT] IQA: [VIF] True GT: [PT @ 32768 spp].
$\mathcal{P}^{\mathcal{C}}$ for Scene: [Cornell Box] Alg: [BDPT] IQA: [HDR-VDP-2] True GT: [PT @ 32768 spp].
$\mathcal{P}^{\mathcal{C}}$ for Scene: [Veach Door] Algorithm: [ERPT] Metric: [HDR-VDP-2] True GT: [BDPT @ 524288 spp].
Cornell Box
Torus
Veach Bidir
Veach Door
Sponza
Average
Average Absolute Magnitude
Maximum Magnitude

Fig. 4.4.: A visual depiction of how table 4.5 is formed. For each configuration $\mathcal{P}^{\mathcal{C}}$ the value with maximum magnitude of misreported error using a visually acceptable reference image becomes the value carried over to the corresponding cell in table 4.5 for that configuration.

For an example configuration (Cornell Box, BDPT), the maximum magnitude of misreporting in MSE (table 4.2e) was $\ln \mathcal{Q}$ of 0.54666, in MS-SSIM (table 4.2b) -0.00037, and in SC-QI (table 4.2c) just -0.00003. The same trends and behaviour can also be seen for across other configurations, such as (Veach Door, ERPT) shown in tables 4.3a-4.3e.

For another representative configuration (Torus, MLT) we will look at some of the metrics which consistently performed poorly in our experiment — the maximum magnitude of misreporting in PSNR (table 4.4a) was $\ln \mathcal{Q}$ of -0.01372, in NQM (table 4.4b) 0.09016, in VSNR (table 4.4c) -0.07703, in MAE (table 4.4d) 0.07799, and in Contourlet IQA (table 4.4e) an $\ln \mathcal{Q}$ of 0.38458.

4.5.1 Critical Analysis of IQA Measures

For a large majority of configurations from all scenes IFC, NQM, HDR-VDP-2, FSIMc, FSIM, VSNR, SC-DM, IW-MSE, Contourlet IQA, and WBCT IQA all show multiple occurrences of severe under- and over-estimation of error within the same configuration and between configurations. This indicates they are highly sensitive to the presence of impulse noise in the reference image and are not able to accurately capture the distribution of distortions that are present under these circumstances. This is a surprising result for Contourlet IQA and WBCT IQA which are RR-IQA methods intended for use with partially corrupted reference images.

(a) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Cornell Box]** Alg: **[BDPT]** IQA: **[VIF]** True GT: **[PT @ 32768 spp]**.

BDPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GT (PT)													
16384	-0.00220	-0.00245	-0.00226	-0.00297	-0.00345	-0.00391	-0.00436	-0.00465	-0.00547	-0.00540	-0.00597	-0.00623	-0.00681
8192	-0.00615	-0.00624	-0.00723	-0.00845	-0.00899	-0.01050	-0.01224	-0.01361	-0.01478	-0.01565	-0.01678	-0.01760	
4096	-0.01403	-0.01611	-0.01684	-0.01841	-0.02120	-0.02472	-0.02762	-0.03124	-0.03331	-0.03580	-0.03766		
2048	-0.02817	-0.02970	-0.03269	-0.03682	-0.04304	-0.04905	-0.05489	-0.06046	-0.06549	-0.06902			
1024	-0.05707	-0.06009	-0.06457	-0.07179	-0.08089	-0.09143	-0.10328	-0.11357	-0.12051				
512	-0.10132	-0.10886	-0.11870	-0.13149	-0.14697	-0.16575	-0.18370	-0.19860					
256	-0.17229	-0.18211	-0.19971	-0.22460	-0.25048	-0.27817	-0.30856						
128	-0.27769	-0.29942	-0.32306	-0.35713	-0.40423	-0.44500							
64	-0.40747	-0.44486	-0.49189	-0.53931	-0.59707								
32	-0.58385	-0.63379	-0.69616	-0.76560									
16	-0.79189	-0.86063	-0.94063										
8	-1.02780	-1.11807											
4	-1.28597												

(b) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Cornell Box]** Alg: **[BDPT]** IQA: **[MS-SSIM]** True GT: **[PT @ 32768 spp]**.

BDPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GT (PT)													
16384	-0.00010	-0.00000	-0.00003	-0.00006	-0.00003	-0.00008	-0.00005	-0.00005	-0.00006	-0.00005	-0.00005	-0.00005	-0.00006
8192	-0.00012	-0.00004	-0.00015	-0.00014	-0.00013	-0.00012	-0.00015	-0.00016	-0.00015	-0.00015	-0.00016	-0.00016	
4096	-0.00007	-0.00025	-0.00025	-0.00027	-0.00026	-0.00035	-0.00034	-0.00036	-0.00035	-0.00037	-0.00037		
2048	-0.00010	-0.00033	-0.00059	-0.00063	-0.00069	-0.00076	-0.00073	-0.00076	-0.00075	-0.00076			
1024	-0.00104	-0.00105	-0.00119	-0.00120	-0.00141	-0.00133	-0.00148	-0.00155	-0.00152				
512	-0.00179	-0.00159	-0.00252	-0.00266	-0.00279	-0.00294	-0.00305	-0.00310					
256	-0.00306	-0.00351	-0.00468	-0.00539	-0.00561	-0.00583	-0.00620						
128	-0.00628	-0.00800	-0.00909	-0.01011	-0.01146	-0.01181							
64	-0.01118	-0.01467	-0.01825	-0.02007	-0.02134								
32	-0.02254	-0.02859	-0.03358	-0.03739									
16	-0.04154	-0.05122	-0.05902										
8	-0.07407	-0.08862											
4	-0.12584												

(c) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Cornell Box]** Alg: **[BDPT]** IQA: **[SC-QI]** True GT: **[PT @ 32768 spp]**.

BDPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GT (PT)													
16384	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	-0.00000
8192	0.00001	0.00001	0.00001	0.00001	0.00000	0.00000	0.00000	0.00000	0.00000	-0.00000	-0.00000	-0.00000	
4096	0.00003	0.00002	0.00002	0.00001	0.00001	0.00001	0.00000	0.00000	-0.00000	-0.00000	-0.00000		
2048	0.00005	0.00004	0.00003	0.00002	0.00001	0.00001	0.00000	0.00000	-0.00000	-0.00000			
1024	0.00008	0.00006	0.00004	0.00003	0.00002	0.00001	0.00000	-0.00000	-0.00001				
512	0.00011	0.00009	0.00006	0.00004	0.00002	0.00001	-0.00000	-0.00001					
256	0.00016	0.00012	0.00008	0.00005	0.00002	0.00000	-0.00001						
128	0.00022	0.00015	0.00009	0.00005	0.00001	-0.00002							
64	0.00027	0.00017	0.00009	0.00003	-0.00003								
32	0.00032	0.00017	0.00006	-0.00003									
16	0.00032	0.00013	-0.00004										
8	0.00024	-0.00003											
4	0.00002												

(d) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Cornell Box]** Alg: **[BDPT]** IQA: **[HDR-VDP-2]** True GT: **[PT @ 32768 spp]**.

BDPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GT (PT)													
16384	0.00069	0.00061	0.00023	0.00017	0.00070	-0.00122	-0.00049	-0.00019	-0.00438	-0.00663	-0.00472	-0.01012	-0.01582
8192	0.00135	0.00077	0.00099	0.00059	0.00003	-0.00055	-0.00104	-0.00320	-0.00575	-0.00867	-0.01418	-0.02252	
4096	0.00301	0.00112	0.00253	0.00150	0.00180	-0.00101	-0.00394	-0.00516	-0.01265	-0.02330	-0.03381		
2048	0.00523	0.00486	0.00434	0.00206	0.00025	-0.00148	-0.00711	-0.01443	-0.02351	-0.03825			
1024	0.00934	0.01054	0.00967	0.00628	0.00390	-0.00147	-0.01202	-0.02456	-0.04561				
512	0.02234	0.01974	0.01844	0.01213	0.00482	-0.00383	-0.02187	-0.04315					
256	0.03743	0.03500	0.02745	0.02212	0.00886	-0.01341	-0.04229						
128	0.06009	0.05272	0.04354	0.02893	0.00325	-0.02340							
64	0.08230	0.06972	0.05534	0.03304	-0.00370								
32	0.10520	0.08631	0.06176	0.02416									
16	0.12428	0.09279	0.05239										
8	0.13101	0.08409											
4	0.12523												

(e) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Cornell Box]** Alg: **[BDPT]** IQA: **[MSE]** True GT: **[PT @ 32768 spp]**.

BDPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
GT (PT)													
16384	0.00021	0.00044	0.00035	0.00146	0.00250	0.00604	0.01123	0.01857	0.03963	0.06006	0.10249	0.14676	0.20907
8192	0.00082	0.00089	0.00228	0.00439	0.00790	0.01515	0.03214	0.05903	0.10703	0.17634	0.27113	0.38223	
4096	0.00067	0.00301	0.00481	0.00984	0.01821	0.03760	0.07164	0.13536	0.23000	0.37515	0.54666		
2048	0.00206	0.00420	0.01003	0.01965	0.04178	0.08150	0.14686	0.26415	0.44191	0.66779			
1024	0.00577	0.01141	0.02103	0.04097	0.08008	0.14905	0.27787	0.48703	0.75710				
512	0.01046	0.02234	0.04352	0.08472	0.15813	0.29235	0.50361	0.81414					
256	0.02121	0.04110	0.08476	0.16638	0.30385	0.52198	0.85770						
128	0.04228	0.08528	0.16436	0.30002	0.54550	0.87442							
64	0.08224	0.16066	0.30742	0.53591	0.88474								
32	0.15806	0.29905	0.53027	0.87775									
16	0.29062	0.52153	0.86325										
8	0.50545	0.84654											
4	0.81929												

Tab. 4.2.: ln $\mathcal{Q}$ of various IQA measures as reference and test image quality are varied for the Cornell Box scene, rendered with Bidirectional Path Tracing. The vertical axis represents the number of s.p.p. in reference images while the horizontal axis denotes the number of s.p.p. in test images. Cells are highlighted from underestimation (blue) to overestimation (orange). The horizontal rule between 2048 and 4096 s.p.p. separates ground truths that exhibit good visual convergence (above) from sample counts that result in ground truths with visible noise (below). Maximum magnitude for reference images with good visual convergence shown with a black underline. The matrix has been flipped vertically and the zero row of reference values versus themselves has been omitted to aid in visualization. Full results for this experiment are available in a supplementary document: `https://static-content.springer.com/esm/art%3A10.1007%2Fs00371-017-1384-7/MediaObjects/371_2017_1384_MOESM1_ESM.pdf`

(a) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Veach Door]** Algorithm: **[ERPT]** Metric: **[VIF]** True GT: **[BDPT @ 524288 spp]**.

ERPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
GT (BDPT)																	
262144	-0.00238	-0.00224	-0.00245	-0.00244	-0.00266	-0.00217	-0.00278	-0.00259	-0.00260	-0.00256	-0.00239	-0.00210	-0.00266	-0.00257	-0.00250	-0.00262	-0.00252
131072	-0.00681	-0.00676	-0.00692	-0.00682	-0.00713	-0.00686	-0.00672	-0.00695	-0.00715	-0.00701	-0.00662	-0.00679	-0.00705	-0.00666	-0.00671	-0.00700	
65536	-0.01588	-0.01512	-0.01554	-0.01536	-0.01514	-0.01534	-0.01499	-0.01568	-0.01619	-0.01516	-0.01463	-0.01558	-0.01555	-0.01514	-0.01539		
32768	-0.03253	-0.03172	-0.03207	-0.03205	-0.03201	-0.03135	-0.03127	-0.03144	-0.03201	-0.03128	-0.03074	-0.03207	-0.03182	-0.03144			
16384	-0.05770	-0.05838	-0.05768	-0.05805	-0.05797	-0.05830	-0.05782	-0.05868	-0.05744	-0.05886	-0.05767	-0.05874	-0.05837				
8192	-0.10139	-0.09951	-0.10131	-0.10073	-0.10077	-0.10192	-0.10090	-0.10180	-0.10146	-0.10235	-0.10103	-0.10001					
4096	-0.16626	-0.16661	-0.16385	-0.16513	-0.16371	-0.16482	-0.16453	-0.16556	-0.16693	-0.16362	-0.16565						
2048	-0.25675	-0.25490	-0.25323	-0.25423	-0.25295	-0.25215	-0.25315	-0.25360	-0.25474	-0.25470							
1024	-0.37885	-0.37749	-0.37417	-0.37708	-0.37402	-0.37564	-0.37508	-0.37473	-0.37926								
512	-0.53061	-0.52879	-0.53008	-0.52773	-0.52737	-0.52966	-0.52757	-0.52944									
256	-0.71289	-0.70754	-0.70496	-0.70610	-0.70779	-0.70980	-0.71231										
128	-0.92762	-0.92693	-0.92687	-0.93014	-0.93220	-0.92879											
64	-1.17268	-1.16182	-1.16261	-1.17058	-1.16728												
32	-1.44034	-1.44247	-1.44341	-1.44428													
16	-1.73919	-1.74373	-1.73871														
8	-2.09441	-2.09284															
4	-2.49507																

(b) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Veach Door]** Algorithm: **[ERPT]** Metric: **[MS-SSIM]** True GT: **[BDPT @ 524288 spp]**.

ERPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
GT (BDPT)																	
262144	-0.00003	-0.00012	-0.00011	-0.00005	-0.00013	-0.00002	-0.00008	-0.00010	-0.00013	0.00002	-0.00010	-0.00002	-0.00005	-0.00009	-0.00006	-0.00011	-0.00005
131072	-0.00017	-0.00020	-0.00025	-0.00015	-0.00023	-0.00020	-0.00020	-0.00014	-0.00025	-0.00016	-0.00019	-0.00016	-0.00019	-0.00016	-0.00016	-0.00019	
65536	-0.00056	-0.00041	-0.00042	-0.00050	-0.00055	-0.00053	-0.00041	-0.00050	-0.00045	-0.00043	-0.00048	-0.00048	-0.00046	-0.00048	-0.00047		
32768	-0.00111	-0.00101	-0.00087	-0.00105	-0.00110	-0.00086	-0.00096	-0.00087	-0.00103	-0.00089	-0.00088	-0.00087	-0.00096	-0.00098			
16384	-0.00179	-0.00213	-0.00192	-0.00221	-0.00196	-0.00203	-0.00197	-0.00210	-0.00201	-0.00219	-0.00197	-0.00206	-0.00194				
8192	-0.00352	-0.00350	-0.00392	-0.00388	-0.00369	-0.00397	-0.00393	-0.00360	-0.00357	-0.00436	-0.00397	-0.00376					
4096	-0.00781	-0.00827	-0.00792	-0.00815	-0.00755	-0.00799	-0.00821	-0.00803	-0.00803	-0.00758	-0.00796						
2048	-0.01504	-0.01486	-0.01497	-0.01573	-0.01501	-0.01500	-0.01433	-0.01463	-0.01467	-0.01438							
1024	-0.02898	-0.02914	-0.02964	-0.02991	-0.02976	-0.02970	-0.02921	-0.02955	-0.02938								
512	-0.05454	-0.05395	-0.05500	-0.05427	-0.05407	-0.05587	-0.05409	-0.05478									
256	-0.09423	-0.09493	-0.09419	-0.09467	-0.09585	-0.09625	-0.09674										
128	-0.15676	-0.16176	-0.16117	-0.16171	-0.16262	-0.15936											
64	-0.24455	-0.24286	-0.25074	-0.24869	-0.25011												
32	-0.35649	-0.36080	-0.36149	-0.36439													
16	-0.48547	-0.49121	-0.49738														
8	-0.64860	-0.65719															
4	-0.85489																

(c) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Veach Door]** Algorithm: **[ERPT]** Metric: **[SC-QI]** True GT: **[BDPT @ 524288 spp]**.

ERPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
GT (BDPT)																	
262144	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
131072	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	0.00001	
65536	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002	0.00002		
32768	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003	0.00003			
16384	0.00004	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005	0.00005				
8192	0.00006	0.00006	0.00006	0.00006	0.00007	0.00007	0.00007	0.00007	0.00007	0.00007	0.00007	0.00007					
4096	0.00007	0.00007	0.00007	0.00008	0.00008	0.00008	0.00008	0.00008	0.00007	0.00008	0.00008						
2048	0.00005	0.00006	0.00006	0.00007	0.00007	0.00007	0.00007	0.00007	0.00006	0.00007							
1024	-0.00000	-0.00000	0.00000	0.00000	0.00002	0.00001	0.00002	0.00001	0.00001								
512	-0.00017	-0.00018	-0.00016	-0.00015	-0.00015	-0.00017	-0.00016	-0.00016									
256	-0.00050	-0.00048	-0.00049	-0.00046	-0.00047	-0.00047	-0.00048										
128	-0.00110	-0.00111	-0.00110	-0.00109	-0.00108	-0.00110											
64	-0.00206	-0.00203	-0.00205	-0.00206	-0.00204												
32	-0.00353	-0.00352	-0.00354	-0.00353													
16	-0.00550	-0.00555	-0.00556														
8	-0.00822	-0.00825															
4	-0.01160																

(d) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Veach Door]** Algorithm: **[ERPT]** Metric: **[HDR-VDP-2]** True GT: **[BDPT @ 524288 spp]**.

ERPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
GT (BDPT)																	
262144	0.00104	0.00076	0.00101	0.00091	0.00058	0.00150	0.00126	0.00037	0.00043	0.00021	0.00040	0.00125	0.00119	0.00132	0.00084	0.00054	0.00087
131072	0.00210	0.00246	0.00313	0.00191	0.00268	0.00188	0.00188	0.00182	0.00211	0.00184	0.00062	0.00202	0.00203	0.00225	0.00227	0.00216	
65536	0.00485	0.00753	0.00487	0.00473	0.00474	0.00566	0.00459	0.00395	0.00400	0.00555	0.00380	0.00570	0.00592	0.00461	0.00676		
32768	0.00872	0.01269	0.01092	0.00967	0.00982	0.01143	0.01023	0.00918	0.00927	0.00921	0.00959	0.00971	0.00809	0.00792			
16384	0.01616	0.01935	0.01873	0.01791	0.01757	0.01850	0.01791	0.01697	0.01818	0.01527	0.01927	0.01862	0.01834				
8192	0.02698	0.03472	0.02895	0.02900	0.02830	0.02933	0.02789	0.02902	0.03031	0.02950	0.03066	0.03103					
4096	0.03780	0.04606	0.04178	0.04590	0.04130	0.04512	0.04118	0.03976	0.04356	0.04259	0.04861						
2048	0.05309	0.06222	0.05854	0.05411	0.05321	0.05841	0.05441	0.05438	0.05320	0.05707							
1024	0.06345	0.07037	0.06696	0.06154	0.05364	0.06256	0.05465	0.06088	0.05849								
512	0.06485	0.07400	0.06212	0.06141	0.05954	0.06078	0.05634	0.05539									
256	0.05187	0.06481	0.04861	0.04539	0.04125	0.04402	0.03603										
128	0.02076	0.02833	0.00992	0.00804	0.00404	0.00353											
64	-0.01823	-0.00739	-0.02678	-0.03170	-0.03489												
32	-0.06405	-0.05702	-0.07391	-0.07994													
16	-0.11210	-0.10649	-0.12344														
8	-0.16033	-0.15293															
4	-0.20728																

(e) $\mathcal{P}^{\mathcal{C}}$ for Scene: **[Veach Door]** Algorithm: **[ERPT]** Metric: **[MSE]** True GT: **[BDPT @ 524288 spp]**.

ERPT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072
GT (BDPT)																	
262144	0.00056	0.00106	0.00123	0.00060	0.00146	0.00100	0.00196	0.00135	0.00166	0.00113	0.00145	0.00130	0.00194	0.00124	0.00203	0.00154	0.00110
131072	0.00321	0.00315	0.00391	0.00362	0.00432	0.00403	0.00485	0.00425	0.00460	0.00419	0.00359	0.00425	0.00484	0.00291	0.00324	0.00432	
65536	0.00610	0.00636	0.00863	0.00760	0.00758	0.00859	0.00820	0.00861	0.00839	0.00790	0.00676	0.00882	0.00798	0.00747	0.00790		
32768	0.01303	0.01300	0.01642	0.01591	0.01772	0.01581	0.01618	0.01542	0.01662	0.01661	0.01453	0.01627	0.01698	0.01546			
16384	0.02062	0.02444	0.02647	0.02935	0.03073	0.03080	0.02914	0.03058	0.02729	0.03193	0.02999	0.03126	0.02944				
8192	0.04174	0.04153	0.05472	0.05858	0.05706	0.06126	0.05993	0.06235	0.05539	0.06209	0.05971	0.05906					
4096	0.08841	0.09406	0.10474	0.11416	0.11293	0.11675	0.11460	0.12012	0.11452	0.11674	0.11821						
2048	0.16736	0.17095	0.19909	0.21436	0.21421	0.21454	0.21079	0.21801	0.21263	0.21586							
1024	0.30766	0.32456	0.36338	0.39176	0.38765	0.39776	0.38693	0.39463	0.38402								
512	0.53892	0.55911	0.63200	0.65907	0.66243	0.67095	0.65946	0.67653									
256	0.86061	0.88683	0.97186	1.02529	1.02840	1.03822	1.03397										
128	1.29715	1.34293	1.44793	1.50644	1.51028	1.51551											
64	1.81570	1.85525	1.97594	2.04208	2.04309												
32	2.39316	2.44020	2.56538	2.63477													
16	3.00176	3.05333	3.17783														
8	3.62338	3.67783															
4	4.19733																

Tab. 4.3.: $\ln \mathcal{Q}$ of various IQA measures as reference and test image quality are varied for the Veach Door scene, rendered with Energy Redistribution Path Tracing. The vertical axis represents the number of s.p.p. in reference images while the horizontal axis denotes the number of s.p.p. in test images. Cells are highlighted from underestimation (blue) to overestimation (orange). The horizontal rule between 32768 and 65536 s.p.p. separates ground truths that exhibit good visual convergence (above) from sample counts that result in ground truths with visible noise (below). Maximum magnitude for reference images with good visual convergence shown with a black underline. The matrix has been flipped vertically and the zero row of reference values versus themselves has been omitted to aid in visualization. Full results for this experiment are available in a supplementary document: `https://static-content.springer.com/esm/art%3A10.1007%2Fs00371-017-1384-7/MediaObjects/371_2017_1384_MOESM1_ESM.pdf`

(a) $\mathcal{P}^{C}$ for Scene: **[Torus]** Algorithm: **[MLT]** Metric: **[PSNR]** True GT: **[BDPT @ 524288 spp]**.

MLT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
GT (BDPT)														
262144	0.00028	0.00034	0.00013	0.00011	-0.00004	0.00001	-0.00020	-0.00041	-0.00017	-0.00042	-0.00040	-0.00044	-0.00051	0.00005
131072	0.00045	0.00007	-0.00051	-0.00071	-0.00101	-0.00121	-0.00136	-0.00174	-0.00170	-0.00194	-0.00221	-0.00216	-0.00245	-0.00259
65536	0.00087	0.00029	-0.00061	-0.00112	-0.00208	-0.00231	-0.00288	-0.00396	-0.00401	-0.00461	-0.00485	-0.00506	-0.00563	-0.00612
32768	0.00055	0.00002	-0.00204	-0.00304	-0.00491	-0.00504	-0.00619	-0.00906	-0.00869	-0.00983	-0.01034	-0.01054	-0.01205	-0.01372
16384	0.00065	-0.00037	-0.00347	-0.00501	-0.00948	-0.00930	-0.01137	-0.01508	-0.01549	-0.01777	-0.01834	-0.01926	-0.02151	
8192	0.00018	-0.00234	-0.00934	-0.01083	-0.01822	-0.01968	-0.02441	-0.03111	-0.03189	-0.03567	-0.03706	-0.03826		
4096	0.00012	-0.00450	-0.01589	-0.02077	-0.03267	-0.03441	-0.04246	-0.05295	-0.05382	-0.05972	-0.06197			
2048	-0.00033	-0.00904	-0.02765	-0.03452	-0.05442	-0.05749	-0.06775	-0.08239	-0.08417	-0.09176				
1024	-0.00259	-0.01640	-0.04141	-0.05264	-0.08109	-0.08659	-0.10115	-0.11891	-0.12060					
512	-0.00190	-0.02011	-0.05514	-0.06772	-0.10506	-0.11039	-0.12760	-0.15128						
256	0.00121	-0.02433	-0.06587	-0.08013	-0.12208	-0.13059	-0.15055							
128	0.00909	-0.02033	-0.06353	-0.08019	-0.12447	-0.13272								
64	0.01244	-0.02168	-0.06387	-0.08282	-0.13169									
32	0.01959	-0.01578	-0.05870	-0.07855										
16	0.02738	-0.01097	-0.05235											
8	0.02439	-0.01286												
4	0.02610													

(b) $\mathcal{P}^{C}$ for Scene: **[Torus]** Algorithm: **[MLT]** Metric: **[NQM]** True GT: **[BDPT @ 524288 spp]**.

MLT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
GT (BDPT)														
262144	-0.00178	-0.00239	0.02558	0.00919	0.00574	0.00543	0.00167	0.00216	0.00120	0.00146	0.00060	0.00131	0.00184	0.00329
131072	-0.00403	-0.00434	0.03315	-0.00335	0.00290	0.00051	0.00149	0.00199	0.00172	0.00195	-0.00000	0.00069	0.00091	-0.00032
65536	-0.00748	-0.00755	0.07999	-0.00017	0.00836	0.00317	0.00278	0.00217	0.00091	0.00027	-0.00108	-0.00132	-0.00121	-0.00293
32768	-0.00989	-0.01295	0.09016	-0.01833	0.01123	-0.00260	-0.00361	-0.00548	-0.00806	-0.00895	-0.00706	-0.00588	-0.00798	-0.01346
16384	-0.01580	-0.02173	0.15206	0.02603	0.00728	0.00353	0.00017	0.00007	-0.00077	-0.00577	-0.00631	-0.00656	-0.00953	
8192	-0.02660	-0.03046	0.09696	-0.02077	-0.00004	-0.01912	-0.03262	-0.03002	-0.03740	-0.04136	-0.03923	-0.04004		
4096	-0.04940	-0.05783	0.27174	-0.02488	-0.01392	-0.02828	-0.04340	-0.05169	-0.06037	-0.07110	-0.06945			
2048	-0.07674	-0.07762	0.28233	-0.05651	-0.04465	-0.08567	-0.08936	-0.09086	-0.10840	-0.10602				
1024	-0.10566	-0.08972	0.47525	-0.13504	-0.10748	-0.15695	-0.17663	-0.18308	-0.20133					
512	-0.14378	-0.12725	0.43507	-0.18601	-0.16076	-0.21466	-0.23041	-0.24039						
256	-0.17634	-0.13796	0.49602	-0.27835	-0.21416	-0.32051	-0.34187							
128	-0.20511	-0.15631	0.58976	-0.40263	-0.26811	-0.36911								
64	-0.20216	-0.11565	0.52206	-0.53104	-0.38080									
32	-0.20988	-0.12940	0.62785	-0.64428										
16	-0.21702	-0.10263	0.51593											
8	-0.20808	-0.11367												
4	-0.25986													

(c) $\mathcal{P}^{C}$ for Scene: **[Torus]** Algorithm: **[MLT]** Metric: **[VSNR]** True GT: **[BDPT @ 524288 spp]**.

MLT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
GT (BDPT)														
262144	0.00646	0.02739	-0.03375	-0.04039	0.00725	0.01184	0.00824	0.00614	0.00642	0.00586	0.00620	0.00579	0.00512	0.00226
131072	0.00910	0.05997	-0.02363	-0.02290	-0.00019	0.00917	0.00503	0.00047	0.00267	0.00148	0.00228	0.00225	0.01964	-0.00303
65536	0.00862	0.03377	-0.01203	-0.07703	-0.01074	0.00460	-0.00369	-0.00800	-0.00604	-0.00758	0.01497	-0.00505	0.01167	0.00048
32768	0.02104	0.05160	-0.04994	-0.02945	-0.02209	0.00076	0.01483	0.00254	0.00921	0.00590	0.00987	0.01053	0.02348	0.00370
16384	-0.00793	0.05844	-0.03425	0.00056	-0.01605	0.02262	0.02999	0.01588	0.02471	0.01859	0.04292	0.04225	0.03540	
8192	-0.04709	-0.06102	-0.02022	0.03856	-0.00305	0.02255	0.02599	0.02586	0.01995	0.01498	0.03797	0.03887		
4096	-0.18883	-0.17457	0.05772	0.07678	0.02768	0.05933	0.04579	0.02532	0.03951	0.03027	0.05459			
2048	-0.37557	-0.68707	0.16051	0.22561	0.05094	0.11790	0.05474	0.04633	0.04697	0.03725				
1024	-0.65146	-1.18979	0.30230	0.35456	0.06627	0.12698	0.05439	0.02559	0.04307					
512	-1.28784	-1.67452	0.42809	0.50934	0.10041	0.17379	0.08533	0.04298						
256	-2.41019	-1.05250	0.48484	0.56582	0.12047	0.19910	0.08172							
128	-2.83200	-0.95782	0.48224	0.56038	0.10516	0.17518								
64	-3.13895	-1.18795	0.48522	0.50033	0.05063									
32	-3.04854	-1.63145	0.44164	0.47009										
16	-1.58401	-3.02872	0.34577											
8	-1.27586	-1.64271												
4	-1.32869													

(d) $\mathcal{P}^{C}$ for Scene: **[Torus]** Algorithm: **[MLT]** Metric: **[MAE]** True GT: **[BDPT @ 524288 spp]**.

MLT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
GT (BDPT)														
262144	-0.00089	-0.00142	-0.00170	-0.00193	-0.00033	-0.00087	0.00054	0.00179	0.00222	0.00260	0.00379	0.00316	0.00368	0.00116
131072	-0.00149	-0.00174	-0.00082	-0.00071	0.00386	0.00367	0.00475	0.00845	0.01044	0.01036	0.01473	0.01349	0.01434	0.01773
65536	-0.00334	-0.00404	-0.00395	-0.00301	0.00606	0.00719	0.01009	0.01846	0.02241	0.02191	0.02944	0.02827	0.03093	0.03802
32768	-0.00410	-0.00534	-0.00320	0.00059	0.01699	0.01755	0.02555	0.04203	0.04705	0.04792	0.06110	0.05797	0.06559	0.07799
16384	-0.00584	-0.00737	-0.00201	0.00285	0.03679	0.03777	0.05131	0.07432	0.08588	0.09071	0.10890	0.10650	0.11970	
8192	-0.00917	-0.00982	0.00545	0.01246	0.06389	0.06918	0.09406	0.13370	0.14838	0.16013	0.18305	0.18110		
4096	-0.01297	-0.01205	0.01541	0.03197	0.10702	0.11590	0.14962	0.21000	0.22596	0.24607	0.27289			
2048	-0.01786	-0.01226	0.02971	0.05442	0.16397	0.17325	0.21605	0.30085	0.32153	0.35124				
1024	-0.02523	-0.01524	0.03319	0.07096	0.22206	0.23516	0.29003	0.40410	0.42571					
512	-0.03762	-0.02872	0.02916	0.07632	0.27415	0.28657	0.35298	0.49918						
256	-0.05574	-0.04506	0.01761	0.07895	0.31550	0.33717	0.41237							
128	-0.08104	-0.07499	-0.00513	0.07440	0.34295	0.36940								
64	-0.09719	-0.09176	-0.00263	0.09118	0.37975									
32	-0.11404	-0.11364	0.01147	0.11455										
16	-0.13144	-0.13108	0.04752											
8	-0.13628	-0.13390												
4	-0.14772													

(e) $\mathcal{P}^{C}$ for Scene: **[Torus]** Algorithm: **[MLT]** Metric: **[Contourlet]** True GT: **[BDPT @ 524288 spp]**.

MLT	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384
GT (BDPT)														
262144	0.00551	0.00871	0.01044	0.01606	0.01929	0.01638	0.02728	0.01285	0.01536	0.00548	0.03538	0.02266	0.00837	0.08320
131072	0.01670	0.02464	0.03070	0.04349	0.05535	0.04486	0.06370	0.05443	0.04801	0.06450	0.09450	0.10731	0.11164	0.13016
65536	0.03557	0.04869	0.06681	0.09001	0.11375	0.10754	0.14477	0.11495	0.13291	0.14771	0.19636	0.22158	0.20313	0.19967
32768	0.06479	0.09259	0.12664	0.16480	0.20539	0.18313	0.26468	0.22637	0.23949	0.26867	0.32612	0.38458	0.31422	0.21846
16384	0.11121	0.16127	0.21763	0.27892	0.38719	0.33936	0.48215	0.43651	0.51505	0.60292	0.56807	0.59327	0.61114	
8192	0.17749	0.26001	0.35584	0.45670	0.63339	0.54201	0.70791	0.71259	0.92235	0.79858	0.86416	0.59327		
4096	0.26106	0.38449	0.51994	0.67596	0.93150	0.78039	0.89992	0.82976	0.83098	0.68205	0.60506			
2048	0.37604	0.55322	0.77995	0.99991	1.21396	1.08099	0.98298	0.99996	0.77064	0.51731				
1024	0.48969	0.72353	1.07635	1.04960	1.14446	0.98824	0.93167	0.79801	0.57849					
512	0.59383	0.91726	1.11344	1.06017	1.00162	0.89710	0.72975	0.59544						
256	0.66642	1.00909	1.22062	1.10482	0.86044	0.78039	0.58570							
128	0.61059	0.96381	1.22062	1.07096	0.73647	0.69419								
64	0.52909	0.86061	1.17495	1.00945	0.71522									
32	0.47381	0.75539	1.15359	1.06017										
16	0.37706	0.58178	0.88262											
8	0.39383	0.59106												
4	0.49267													

Tab. 4.4.: $\ln \mathcal{Q}$ of various IQA measures as reference and test image quality are varied for the Torus scene, rendered with Metropolis Light Transport. The vertical axis represents the number of s.p.p. in reference images while the horizontal axis denotes the number of s.p.p. in test images. Cells are highlighted from underestimation (blue) to overestimation (orange). The horizontal rule between 16384 and 32768 s.p.p. separates ground truths that exhibit good visual convergence (above) from sample counts that result in ground truths with visible noise (below). Maximum magnitude for reference images with good visual convergence shown with a black underline. The matrix has been flipped vertically and the zero row of reference values versus themselves has been omitted to aid in visualization. Full results for this experiment are available in a supplementary document: `https://static-content.springer.com/esm/art%3A10.1007%2Fs00371-017-1384-7/MediaObjects/371_2017_1384_MOESM1_ESM.pdf`

Overall, two of the worst performing metrics were the WBCT and Contourlet IQA methods which consistently overestimated error, with an average maximum overestimation across all scenes and rendering algorithms of 0.19697 and 0.19058 respectively. These methods are the same measure performed on the different decompositions of the input images which is simply a distance between two coarse histograms over the proportion of visually important coefficients in a multi-scale image decomposition. These are classified as RR IQA methods, meaning they are designed to work with the assumption that the reference image may contain errors but is still representative. However these results show that the measures are highly sensitive to image distortions such as high frequency impulse noise that are prevalent in Monte Carlo rendered images, even at high sample counts. The commonly used MSE measure performs just as poorly, consistently overestimating error with an average maximum of 0.1762 overestimation. MAE performs slightly better with an average overestimation of 0.10273 which is to be expected as MSE weights deviations quadratically while MAE weights deviations linearly.

At the other end of the scale, VIF and IFC consistently underestimate error between images with an average maximum of -0.09731 and -0.07349 respectively. Both methods are based on approximating the two random fields of a GSM noise model. This model assumes that the reference image is correct and does not account of distortions within the reference. Other IQA methods that build off of the GSM model are the Information Content Weighting methods. IW-MSE on average performs slightly better than the standard MSE with an average maximum overestimation of 0.13344; however due to the poor ability of the GSM to handle noise in the ground truth, this performance is likely due to the addition of multi-scale image analysis rather than because of the GSM noise model. The performance of IW-SSIM which had an average maximum underestimation of -0.00532 supports this theory as it is marginally worse than that of MS-SSIM which scored an average maximum underestimation of -0.00248. These methods only differ in the use of the GSM noise model. UQI and SSIM which don't perform multi-scale image analysis also support this as they perform worse than MS-SSIM with average maximum underestimations by -0.05772 and -0.01127 respectively.

Out of the five scenes the Torus scene showed the largest magnitudes of misreported results, likely due to the slow convergence of caustic illumination. The Veach Bidir and Veach Door scenes also feature caustic illumination however these converge comparatively quickly compared to the Torus scene and this can be seen in reduced comparative misreporting between the scenes.

		VIF	IFC	UQI	PSNR	IW-PSNR	SSIM	NQM	IW-SSIM	MS-SSIM	SC-QI	HDR-VDP-2	FSIMc	FSIM	VSNR	SC-D
Cornell Box	PT	-0.03370	-0.08914	-0.03688	-0.02483	-0.02508	-0.00175	-0.02421	-0.00299	-0.00038	0.00004	-0.02242	0.00829	0.00830	-0.02707	-0.09
GT (PT)	BDPT	-0.03766	-0.16721	-0.05302	-0.04385	-0.05098	-0.00187	-0.03961	-0.00198	-0.00037	0.00003	-0.03381	0.00747	0.00747	-0.02998	0.45
	PSSMLT	-0.03556	-0.08665	-0.04487	-0.01529	-0.01522	-0.00181	-0.01222	-0.00250	-0.00037	0.00002	-0.01098	0.00694	0.00694	-0.02383	0.07
	MLT	-0.03221	-0.06861	-0.03109	-0.01000	-0.00561	-0.00176	-0.00550	-0.00266	-0.00038	0.00003	-0.00911	0.00835	0.00837	-0.01501	-0.05
	Manifold-MLT	-0.03270	-0.08660	-0.03258	-0.01017	-0.00606	-0.00177	-0.00814	-0.00255	-0.00036	0.00003	-0.00505	0.00838	0.00840	-0.01564	-0.06
	ERPT	-0.02270	-0.02777	-0.01161	-0.00209	-0.00177	-0.00164	-0.00217	-0.00093	-0.00035	0.00001	0.00135	0.00307	0.00306	-0.00459	-0.05
	Manifold-ERPT	-0.02187	-0.02349	-0.01002	-0.00181	-0.00157	-0.00156	-0.00200	-0.00094	-0.00034	0.00001	0.00162	0.00305	0.00305	-0.00511	-0.05
Torus	PT	-0.25696	-0.11270	-0.09195	-0.04573	-0.03921	-0.02489	-0.06889	-0.01844	-0.00672	0.00020	0.05888	0.05853	0.05892	-0.11038	-0.21
GT (BDPT)	BDPT	-0.24771	-0.10768	-0.09545	-0.04234	-0.03724	-0.02300	-0.05962	-0.01684	-0.00625	0.00021	0.06461	0.05934	0.05977	-0.10721	-0.23
	PSSMLT	-0.32934	-0.20121	-0.18467	-0.04048	-0.01267	-0.03565	-0.04632	-0.01518	-0.00748	0.00018	0.06946	0.01822	0.01833	-0.09096	0.29
	MLT	-0.33488	-0.21295	-0.20203	-0.01372	0.00515	-0.03606	0.09016	-0.01120	-0.00756	0.00020	0.05198	0.01144	0.01169	-0.07703	0.08
	Manifold-MLT	-0.35628	-0.25590	-0.25184	-0.02246	0.00560	-0.03818	0.11864	-0.01310	-0.00847	0.00015	0.03651	0.01484	0.01525	-0.33361	0.16
	ERPT	-0.26809	-0.10343	-0.12235	-0.03366	-0.01447	-0.03133	-0.04850	-0.01580	-0.00740	0.00003	0.06267	0.01360	0.01367	-0.08538	-0.05
	Manifold-ERPT	-0.26922	-0.10264	-0.12673	-0.03137	-0.01457	-0.03111	-0.04444	-0.01593	-0.00755	0.00002	0.02725	0.01280	0.01289	-0.08659	-0.05
Veach Bidir	PT	-0.01659	-0.04894	-0.00333	-0.00107	0.00227	-0.00148	0.16618	0.00274	0.00089	0.00009	0.00748	0.01231	0.01231	0.01058	-0.03
GT (BDPT)	BDPT	-0.03412	-0.08898	-0.04388	-0.04562	-0.07197	-0.00234	-0.04159	0.00196	-0.00063	0.00006	-0.08495	0.00817	0.00817	-0.09721	0.37
	PSSMLT	-0.03463	-0.06634	-0.04196	-0.02015	-0.02454	-0.00242	-0.01531	0.00176	-0.00060	0.00005	-0.05244	0.00832	0.00832	-0.04186	-0.07
	MLT	-0.02697	0.03898	-0.01720	-0.00637	0.00964	-0.00211	0.01242	0.00212	0.00064	0.00006	0.07547	0.01050	0.01049	0.01921	-0.05
	Manifold-MLT	-0.02796	0.05100	-0.02065	-0.00642	0.00935	-0.00226	0.01560	0.00233	0.00109	0.00006	-0.02304	0.01030	0.01030	-0.02151	-0.05
	ERPT	-0.02107	-0.03051	-0.00509	-0.00366	0.00636	-0.00216	0.00674	0.00090	-0.00061	0.00002	0.00834	0.00438	0.00438	-0.00904	-0.03
	Manifold-ERPT	-0.02088	-0.03062	-0.00466	-0.00298	0.00571	-0.00202	0.00641	-0.00098	-0.00062	0.00002	-0.00631	0.00440	0.00441	-0.00922	-0.03
Veach Door	PT	-0.02103	-0.05344	-0.00593	-0.00573	-0.00518	-0.00263	-0.00632	-0.00454	-0.00053	0.00011	-0.04776	0.00492	0.00491	0.18706	-0.07
GT (BDPT)	BDPT	-0.03288	-0.05916	-0.03910	-0.02856	-0.02913	-0.00301	-0.02221	-0.00458	-0.00058	0.00011	-0.08349	0.00473	0.00473	2.02713	0.12
	PSSMLT	-0.03248	-0.05987	-0.04727	-0.02185	-0.01861	-0.00295	-0.01608	-0.00409	-0.00056	0.00008	-0.05149	0.00392	0.00392	0.02342	0.14
	MLT	-0.03410	-0.07112	-0.06547	-0.02261	-0.00967	-0.00298	-0.01709	-0.00404	-0.00056	0.00009	-0.08106	0.00401	0.00401	-0.02349	0.21
	Manifold-MLT	-0.03438	-0.07323	-0.06571	-0.02045	-0.00787	-0.00301	-0.01439	-0.00424	-0.00057	0.00009	-0.02645	0.00395	0.00395	-0.03055	0.21
	ERPT	-0.01619	-0.00649	-0.00410	-0.00122	0.00049	-0.00227	-0.00140	-0.00176	-0.00056	0.00002	0.00753	0.00303	0.00303	-0.00389	-0.02
	Manifold-ERPT	-0.01563	-0.00862	-0.00327	-0.00121	-0.00047	-0.00218	-0.00172	-0.00203	-0.00058	0.00002	0.00702	0.00314	0.00314	-0.00388	-0.02
Sponza	PT	-0.11977	-0.08352	-0.05734	-0.03651	-0.03375	-0.01970	-0.03707	-0.00850	-0.00427	0.00021	0.05730	0.02183	0.02187	-0.05280	0.16
GT (PT)	BDPT	-0.11455	-0.07631	-0.05407	-0.03133	-0.02826	-0.01951	-0.03274	-0.00838	-0.00444	0.00022	0.05653	0.02157	0.02160	-0.04809	0.13
	PSSMLT	-0.10372	-0.04370	-0.04225	-0.01021	-0.00640	-0.01793	-0.01145	-0.00762	-0.00397	0.00016	0.06167	0.02050	0.02054	-0.01744	-0.04
	MLT	-0.12948	-0.09366	-0.07845	-0.01209	-0.00628	-0.01992	-0.01219	-0.00856	-0.00431	0.00016	0.04427	0.02080	0.02075	-0.01973	-0.03
	Manifold-MLT	-0.13087	-0.09648	-0.08093	-0.01190	-0.00723	-0.01953	-0.01410	-0.00716	-0.00435	0.00016	0.04745	0.02090	0.02094	-0.01777	-0.03
	ERPT	-0.08035	-0.01056	-0.02362	-0.00702	-0.00439	-0.01611	-0.00797	-0.00516	-0.00391	0.00003	0.01427	0.00583	0.00583	-0.01411	-0.03
	Manifold-ERPT	-0.07951	-0.01462	-0.02092	-0.00671	-0.00428	-0.01557	-0.00713	-0.00545	-0.00388	0.00003	0.01591	0.00569	0.00570	-0.01438	-0.03
	Average	-0.09731	-0.07349	-0.05772	-0.01833	-0.01251	-0.01127	-0.00584	-0.00532	-0.00248	0.00009	0.00683	0.01250	0.01255	0.02372	0.02
	Average Absolute Magnitude	0.09731	0.07863	0.05772	0.01833	0.01506	0.01127	0.02962	0.00600	0.00263	0.00009	0.03760	0.01250	0.01255	0.10585	0.11
	Maximum Magnitude	-0.35628	-0.25590	-0.25184	-0.04573	-0.07197	-0.03818	0.16618	-0.01844	-0.00847	0.00022	-0.08495	0.05934	0.05977	2.02713	0.45

Tab. 4.5.: A condensed table showing all 735 result tables, showing max $\mathcal{P}^C$ for all Algorithms, Scenes, and Me
estimation in each configuration for IQA. These worst cases are from the high sample count ground tru
in the supplementary materials. A value of zero represents an ideal result showing error has not bee
column Average. Cells are highlighted from underestimation (blue) to overestimation (orange). Full re
supplementary document: `https://static-content.springer.com/esm/art%3A10.1007%2Fs`
`371_2017_1384_MOESM1_ESM.pdf`

4.6 Recommendations and Conclusions

It is difficult to find a balance between the desire for a purely numerical distance metric as we are evaluating the quality of a numerical simulation, and the desire to measure only the perceivable noise as observed by the HVS. We argue that a good balance of these features is for a proposed error metric to be monotonic with respect to a simple numerical divergence like MSE such that a reduction in numerical distance always corresponds to a reduction of reported error. Of the IQA considered in this work that were more advanced than a numerical distance MS-SSIM, SC-QI, SC-DM, and NQM were all monotonic with respect to MSE for the types of distortion that are prevalent in Monte Carlo rendered images. The other IQA tested all showed non-monotonicity in the presence of strong impulse noise, primarily from caustic illumination.

IQA which measured per-pixel structural information seemed to be more robust to the effects of impulse noise in the reference image; however, a stronger divide was seen between methods that applied multi-scale geometric analysis and those that did not. By isolating high frequency noise in one level of a multi-scale decomposition its effects on image assessment can be bounded or minimized effectively.

Metrics which used perceptual models of the HVS were highly sensitive to the noise in reference images and quickly became unreliable as the quality of the reference was degraded.

Rendering algorithms such as Path Tracing and Bidirectional Path Tracing, which uniformly sample path space, are better suited to the task of producing reference images than rendering algorithms which use a Markov based random walk such as Metropolis Light Transport or Energy Redistribution Path Tracing. While in certain situations Markov based algorithms exhibit faster convergence than uniform sampling methods, before the simulation has fully converged a uniform method will have independently distributed error while a Markov algorithm will exhibit noise distributed deterministically with respect to the trajectory the random walk has followed. The result of this is that when we consider the possibility of noise in reference images, noise from Markov processes are more likely to form structural artefacts in the reference, exacerbating misreported error when IQA consider structural features and similarity.

Our recommendations are that MS-SSIM or SC-QI be used for image quality assessments when evaluating images produced by Monte Carlo rendering algorithms as these methods were the most robust when we consider noise in reference images. Reference images should ideally be rendered with uniform sampling methods to avoid the introduction of structural artefacts in IQA. It is important that the reference used is not only visually noise free, but also that it is of sufficiently higher numerical quality than images tested against it. Reference images should therefore be rendered to at least an order of magnitude higher sample count than test images to minimize the possibility of noise in the reference causing a significant deviation in reported error. Finally, we recommend that the sample count and method of production of the reference image should be stated clearly to give researchers every confidence in reported results.

4.6.1 Further Work

As the research continues with the aim of creating more robust IQA measures targeting Monte Carlo rendered images there is an increasing need for standardized and trusted datasets for comparing and contrasting measures on domain relevant data. In this work we have shown how natural image datasets coupled with synthetic distortions such as those used by Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], and IRCCyN/IVC [AB09] are not representative of the naturally occurring distortions from stochastic rendering processes.

We propose to expand the Monte Carlo image dataset presented in this work by adding additional scenes, both including scenes which aim to model photo-realistic image compositions, and those which aim to model more synthetic based images of single material models under controlled lighting and viewing conditions. The addition of these scenes both increases the visual fidelity of the dataset by including a larger range of natural distortions, but also allows for controlled experimentation with how specific material, lighting, and viewing conditions effect the perception of visible distortions. The expanded dataset should be computed in high-dynamic range so that both HDR and LDR quality measures can be trained and evaluated.

Finally, the dataset should be coupled with a user study to produce MOS or DMOS scores for images. Subjective DMOS would allow for the magnitude of quality misreporting of each IQA measure to be correlated against the change in differential quality that was perceived by human observers. Further, having subjective quality scores for the rendered images will allow for the development of novel IQA measures that are able to accurately model human perception to noise from Monte Carlo rendering processes. It would also allow existing IQA measures to be re-trained on the Monte Carlo image dataset to produce a comparison of their performance when considering this type of distortion.

Full results for our experiment are available in a supplementary document: `https://static-content.springer.com/esm/art%3A10.1007%2Fs00371-017-1384-7/MediaObjects/371_2017_1384_MOESM1_ESM.pdf`

5 A Deep Learning Approach to No-Reference Image Quality Assessment using Convolutional Neural Networks

In FR-IQA, images are compared with ground truth images that are known to be of high visual quality. These metrics are utilized in order to rank algorithms under test on their image quality performance. During Monte Carlo rendering processes we wish to determine whether enough samples have been produced such that the rendered image is sufficiently noise free, or to determine where in the image computational resources should be focused to maximize work throughput. Without the availability of a ground truth image FR-IQA metrics are not applicable and we need to utilise NR-IQA.

In this chapter we propose a deep learning approach to NR-IQA trained specifically on noise from Monte Carlo rendering processes, which significantly outperforms existing NR-IQA methods, and produces performance close to the approximated FR-IQA measure. These experiments make use of the Monte Carlo image dataset described in chapter 4 to learn a noise profile for the spatially varying image distortions which occur due to under-sampling within the light transport simulation before it has converged.

5.1 Introduction

Over the last three decades much work has been conducted in the area of Monte Carlo physically based light transport simulations for scientific and rendering applications. Such simulations are capable of modelling the complex interactions of light with a wide range of physically based materials, participating media, and camera models which can be used to synthesize images that are indistinguishable from photographs. The trade-off for producing images with high visual quality is slow computation time as the algorithms are only guaranteed to converge in the limit as the number of samples approaches infinity. Early termination of the rendering process can leave an undesirable amount of visual distortions in the form of impulse noise, and missing illumination from complex interactions such as caustic and indirect illumination. When comparing the performance and visual quality of images produced by Monte Carlo rendering processes, a commonly accepted methodology is to render a reference image to a large number of samples to ensure that the reference has high visual quality. This reference image is then assumed to be representative of the Ground Truth (GT) image and can be used experimentally to compare images rendered by algorithms under test, either in an equal time or equal quality benchmark. To compare images to the assumed GT, FR-IQA methods such as MAE, MSE, SSIM [Wan+04a], or more recently MS-SSIM [Wan+03], HDR-VDP-2 [Man+11], or SC-QI [BK16] can be used to evaluate the relative quality of test images. Outside of algorithm benchmarking, the GT image is often not available. In such cases NR-IQA measures provide a way to infer the approximate quality of images or image regions for the purpose of terminating the rendering process once the image is of sufficient quality, or to direct additional computational resources to specific image regions.

In this chapter we present a deep learning approach to NR-IQA, specifically aimed at evaluating the noise that is found in un-converged Monte Carlo rendered images. We show that a convolutional neural network architecture is capable of modelling the spatially varying nature of natural image distortions that arise from under-sampling of complex light transport simulations; and that predictions made by the model can approach, within a high degree of accuracy, to FR-IQA methods.

These models are designed to estimate image quality "blindly" by comparing features extracted from the local regions around each pixel in an un-converged input image to the distribution of those features which occur naturally in clean images from the target domain. This representation is learned by regression from a corpus of images corrupted with varying magnitudes of distortion. The regression target for a given noisy image is defined as the result of computing an FR-IQA measure between the un-converged image and its associated GT image.

5.2 Related Work

In image processing tasks IQA algorithms aim to measure the quality or similarity of images. In FR-IQA algorithms have access to a potentially corrupted image and a GT image which is known to be correct and distortion free; the algorithm is tasked with evaluating the similarity between the corrupt and GT images. The meaning of "similarity" can vary between algorithms; ranging from simple definitions of geometric distance, to complex models of the Human Visual System (HVS) and perceived distortion. In contrast, NR-IQA algorithms are tasked with approximating the quality of a test image without outside input on what the image should look like. In this formulation, "quality" can also take on different meanings based on the context and intent of the NR-IQA algorithm; most often the problem is formulated as approximating the similarity between the test image and a predicted ground truth image. The method of prediction tends to be based on the statistics of natural scenes that can be modelled explicitly or extracted from representative labelled image data through machine learning.

A recent survey of 38 NR-IQA measures [KB15] exploring the current state of the art in NR-IQA has been conducted. The metrics considered were clustered into (overlapping) groups signifying the types of distortions the measures were designed to be applied to. The distortion types are:

Additive Gaussian Noise	[CY10; Lu+10; MB10; Saa+10; MB11; Mit+12b; YD12; Saa+12; Mit+12c; Mit+13b; Hua+14; Jia+13]
Gaussian Blur	[Mar+04; ZMS07; FK09; Wu+09; Lu+10; MB10; Saa+10; CY10; Cia+11; NK11; MB11; Ser+13; Mit+12b; YD12; Saa+12; Mit+12c; Mit+13b; Jia+13]
JPEG-2000 Compression	[Mar+04; She+05c; Saz+08; Lu+10; MB10; Saa+10; ZL10; Lia+10; MB11; Mit+12b; YD12; Saa+12; Mit+12c; Mit+13b; Jia+13]
JPEG Compression	[Bab+07; BQ08; Zha+08; Sur+09; Lu+10; MB10; Saa+10; MB11; Mit+12b; YD12; Saa+12; Mit+12c; Mit+13b; Jia+13]
Blockiness	[Sut09; LP12]
Fast Fading	[Lu+10; MB10; Saa+10; MB11; Mit+12b; YD12; Saa+12; Mit+12c; Mit+13b; Jia+13]
Sub-band Truncation	[Du+05]
Ringing Artefacts	[Liu+10]

Mittal *et al.* [Mit+12a] propose an NR-IQA metric based on natural scene statistics designed for natural images under synthetic distortions, trained against subjective quality scores given by human observers from the Live [She+14] and TID2008 [Pon+09] databases. Mittal *et al.* [Mit+13a] also develop a "completely blind" NR-IQA based on natural scene statistics which does not need to be fitted against subjective quality scores. Kundu *et al.* [Kun+16a] develop a NR-IQA metric based on spatial and gradient domain features. Specifically their method is targeted at high dynamic range images under varying tone-mapping methods. Their method is evaluated on tone-mapped HDR images from the ESPL-HDR [Kun+16b] database and also on LDR synthetically distorted images from the Live [She+14] database. Sheikh *et al.* [She+05b] apply a simplified form of natural scene statistics to develop an NR-IQA metric for natural images distorted by JPEG 2000 block encoding from the Live [She+14] database.

In the above works, distorted images are sampled from publicly available datasets, namely Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], and IRCCyN/IVC [AB09]. These datasets contain clean reference images and their synthetically distorted test images, coupled with subjective quality scores given by and pooled over a set of human observers. The reference images are natural photographs covering a wide range of scene compositions and subject matters. Much of the work contained within the current of state of the art in IQA methods focus on fitting models to the subjective quality scores given for these synthetically distorted images. The types of distortion considered attempt to approximate the types of distortions that can occur during compression, transmission, or image synthesis. In our work we find that in many cases synthetic distortions do not present a representative target for training and evaluating metrics that will be applied to naturally distorted images. Thus, in this work we propose a model trained directly on images containing naturally occurring distortions from un-converged Monte Carlo rendering processes.

5.2.1 IQA in Monte Carlo Rendering

Monte Carlo rendering algorithms [Kaj86] allow for photo-realistic images to be computed that can faithfully recreate complex physically based lighting phenomena. These methods approximate an integral to create an image using an average over many discrete samples. In such light simulations each sample represents the energy contribution of individual photons or regions of path-space and as such no one sample is representative of the integral's final expectation. Rather, only in the limit as the number of samples goes to infinity will the error of the approximation go to zero, and the average of the samples converge to the expectation of the integral. By limiting the number of samples used in the rendering computation the approximate image will naturally contain a certain amount of error compared to the final expectation. When benchmarking the rendering algorithms themselves it is a commonly accepted methodology to render an image to an high sample count to ensure it will have high visual quality, and to use this image as a GT with which to compare images generated with competing algorithms in either an equal time or computation comparison using a well established FR-IQA measure. In this formulation the rendered GT image can potentially contain distortions as it is also the product of an integral approximated using a finite number of samples. In the previous chapter 4 we surveyed of the robustness of 19 FR-IQA and 2 RR-IQA (Reduced Reference) algorithms and showed that Multi-Scale Structural Similarity Index (MS-SSIM) [Wan+03] and Structural Contrast Quality Index (SC-QI) [BK16] give the most consistent results when the GT image itself contains visible distortions, and is therefore not representative of the final expectation image.

Another use of IQA methods in Monte Carlo rendering is to evaluate the quality of images during the rendering process — either to evaluate when the simulation has converged to its true value, or to direct proportionally more computation into regions with lower quality. In such cases, NR-IQA measures can allow for image quality to be approximated without explicitly knowing what the final value should be.

Ferwerda *et al.* [Fer+97] propose a model for predicting the effects of visual masking in rendered images. Their method allows for textures and mesh details in Monte Carlo rendered images to be chosen to optimally hide the effects of noise and distortions that are prevalent when light simulations are yet to converge.

Herzog *et al.* [Her+12a] present an NR-IQA method designed to capture visual sensitivity to rendering artefacts, such as those arising from Virtual Point Lights [Kel97], and shadow map discretisation. In addition to the rendered images, the method also uses information such as the surface albedo and depth map, which are available within the rendering framework. Yee *et al.* [Yee+01a] use animation sequences rendered without global illumination to approximate where viewers will focus their attention throughout the sequence. By directing proportionally more computational effort to regions determined to be visually important high quality can be achieved in the final global illumination pass using a significantly reduced number of samples.

Lavoué *et al.* [LM15a] show that visible distortions in Monte Carlo rendered images are highly correlated with artefacts in the 3D meshes composing the rendered scene.

5.2.2 Machine Learning

The NR-IQA problem is closely related to the problem of blind denoising, where we wish to recover a clean image from a corrupted input image. Intuitively, if we could perfectly detect noise in a distorted image, then subtracting the noise map would result in a perfectly denoised image. Similarly, if a perfectly denoised image could be extracted from the distorted image, then the difference map would perfectly capture the distortion at each pixel. In this way we can see that there is a shared feature space of information relevant to both tasks.

Recently, machine learning has been applied to the task of blind image denoising under synthetic and naturally occurring distortions. Kalantari *et al.* [Kal+15] apply deep learning to denoising Monte Carlo rendered images. Their method uses a set of 7 primary features such as pixel colour, screen-space coordinates, world-space coordinates, shading normals, texture albedo at the first and second bounces, and the direct illumination shadow-map; which can all be extracted directly from the rendering pipeline without loss of rendering performance. From this they compute 36 secondary features by computing per-pixel statistics over the sample set and over local neighbourhood regions of the primary feature maps. These secondary features are fed, for each pixel independently, to an MLP to predict a set of variances for each dimension in a cross bilateral or cross non-local means filter which is to be to the distorted input image. Inference on full images is performed by considering each pixel batch-wise. Formally their method calls for an MLP with 36 input units for the secondary features, a single hidden layer with 10 units with Sigmoid activation, and lastly 6 output units with Softplus activation representing the 6 control parameters of the filter model being used for denoising. Their model is trained using the RPROP optimization algorithm [RB93] to minimize the Relative-MSE [Rou+12b] of the reconstructed pixel colour compared to the ground truth pixel colour, scaled by the sample-rate of the pixel to account for the rate of convergence of the rendering simulation.

Deep learning has also shown promise in blind denoising of natural images corrupted by additive Gaussian noise [Zha+17b]. In that work the residual image $\mathcal{R}(x)$ of the noisy input is predicted such that $\hat{y} = x - \mathcal{R}(x)$ gives a denoised image that approximates y. Corrupted image patches are generated by distorting each pixel by a value drawn from a Gaussian distribution $\mathcal{N}(0, \sigma)$ where σ is held constant for pixels in the same patch. The distinction that σ is constant for each patch is important as it means a robust strategy for denoising is to approximate the distribution that was used to corrupt the input. In the case of naturally occurring distortions this is a considerably more difficult task as natural distortions are often spatially varying with complex underlying distributions. Formally their model is a CNN referred to as a Denoising Convolutional Neural Network (DnCNN) consisting of: 3 input feature maps for RGB image input; a 3×3 convolutional layer with 64 output feature maps with ReLU activation; 18 blocks consisting of a 3×3 convolutional layer with 64 output feature maps, batch normalization [IS15], and ReLU activation; and a final 3×3 convolutional layer with 3 output feature maps representing the residual RGB image that denoises the input image when summed. Their model is trained to minimize the $\mathcal{L}_2$ loss between reconstructed and true images using the Adam [KB14] optimizer.

The application of deep learning to image de-raining [Zha+17a; Fu+17] represents a denoising task on natural images under natural distortion. In [Fu+17] a convolutional neural network is trained using clean images with rain synthetically added to them, evaluated using standard FR-IQA methods such as SSIM [Wan+04a], PSNR, and VIF [SB06]. Their model is then validated against competing methods on real-world rainy images where a ground truth is not available using Blind Image Quality Index [MB10]. Their model starts by splitting the input RGB image into a low- and high-pass image. The high-pass RGB image is de-rained by feeding it to a CNN with 3 convolutional layers, the first consisting of 16×16 convolutions outputting 512 feature maps, the second consisting of 1×1 convolutions outputting 512 feature maps, and the third layer with 8×8 convolutions outputting 3 feature maps for the RGB de-rained high-pass image. Both the de-rained high-pass and low-pass images are colour enhanced and summed to produce the final de-rained image. The model is trained to minimize the $\mathcal{L}_2$ loss between de-rained and true images using the Stochastic Gradient Descent (SGD) [RM51] optimizer.

An alternative method using conditional adversarial training [Zha+17a] also shows promising results. Rainy/snowy images are fed through two networks, the first of which attempts to reconstruct the de-rained image. De-rained images are then concatenated with clean images labelled as fake and real respectively, and fed to a second discriminator network. The generative model is a CNN with residual connections built around blocks of 3×3 convolutional or de-convolutional layers coupled with batch normalization and ReLU activation. The input RGB image is fed to four subsequent convolutional blocks with 64 feature maps each. This is followed by a pair of blocks with 32 and 1 feature maps respectively. The generator then switches to deconvolution with the first block outputting 32 feature maps. Then, four de-convolutional blocks with 64 feature maps are applied in sequence, with a final block outputting 3 feature maps for the RGB channels of the de-rained image. Additive residual connections are added between the input image and output feature maps, between second convolutional block and the second to last de-convolutional block, and finally between the fourth convolutional block and the second de-convolutional block. TanH activation is applied to the result of the outermost residual connection yielding the final de-rained image. The discriminator model compresses an input image down to a single Sigmoid activated neuron representing whether the discriminator thinks the image is a real or fake input. The discriminator down-samples the input image by sequentially applying three convolutional blocks with stride 2 and 48, 96, and 192 output feature maps respectively. This is followed by a final two blocks with stride 1 with 384 and 1 feature channels each. Training the networks in lock-step, the discriminator network learns to tell real clean images from fake de-rained images, while the reconstruction network learns to fool the discriminator by producing de-rained images with high visual fidelity. This is done by freezing the weights in the discriminator and updating the generator weights to minimize the loss from lying to the discriminator.

Li *et al.* [Li+11] apply General Regression Neural Networks (GRNN) to NR-IQA on natural images under synthetic distortion. In their formulation hand crafted features are extracted from distorted images based on phase congruency, information entropy, and the image gradient; these features are fed, independently for each pixel, to an MLP with two hidden layers to predict the DMOS from the Live [She+14] database. Liu *et al.* [Liu+16] employ a similar strategy using neural nets with AdaBoosting, which improves the robustness of quality predictions.

Bosse *et al.* [Bos+16a; Bos+16b] apply deep convolutional networks to NR-IQA. Their model is a sequential CNN without residual or skip connections. Starting from a 32×32 RGB input this data is fed through five blocks consisting of two 3×3 convolutions followed by a 2×2 max-pooling layer [Bou+10]. Each block halves the width and height dimensions, and doubles the number of output feature maps. Starting with 32 feature maps of size 16×16 after the first block and ending with 512 feature maps of size 1×1 after the fifth block. The model then has a fully-connected hidden layer with 512 units and dropout [Sri+14] regularization with a probability of 50%, before a final hidden layer with a single output neuron with linear activation which represents the quality score for the input image patch. All convolutional layers and the first fully-connected layer have ReLU activation. Their method was trained using the Adam optimizer [KB14] to minimize the $\mathcal{L}_1$ loss to the predicted DMOS quality score for an image from the Live [She+14] database, given a set of randomly sampled 32×32 RGB patches from the image. For each full image assessment the network is only presented with a random subset of possible patches; this forms an implicit dropout regularization which helps prevent memorization of the training set. When the loss is computed using ground truth IQA values for individual patches (as is the case with our data) the ability of the network to memorize the input data becomes prohibitive to the models ability to learn a generalized strategy for NR-IQA.

5.3 Experiment I

In our first experiment we wish to test whether a deep convolutional neural network can capture the distribution of error in Monte Carlo rendered images using only local neighbourhood information from the distorted image. To accommodate this, our model directly predicts the Mean Absolute Error (MAE) of 32×32 RGB image patches with pixels in the range $[0, 1]$ sampled from images rendered with Monte Carlo rendering algorithms to varying degrees of quality. The choice of RGB as the input colour-space is motivated by the work of Reddy *et. al.* [Red+17] who studied the effect of colour-space on CNN learning performance, concluding that the

RGB and LUV colour-spaces lead to the greatest prediction accuracy compared to alternative representations.

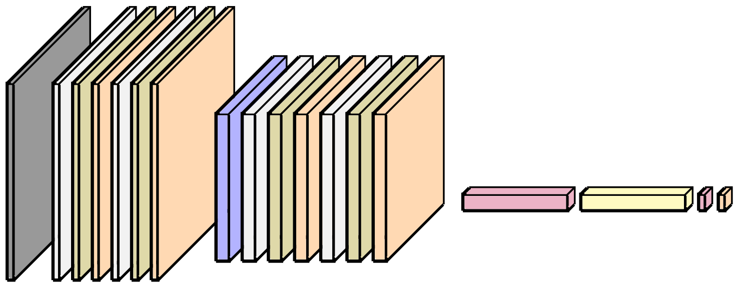


Fig. 5.1.: Experiment I: Our patch based model. The model consists of four convolutional blocks, each with batch normalization and PReLU activation, separated by max pooling; followed by two fully connected layers with 70% dropout. The output neuron has ReLU activation to ensure the resulting quality score is positive. The full model is: Input$(32 \times 32, 3)$, Conv$(3 \times 3, 32)$, BNorm, PReLU, Conv$(3 \times 3, 32)$, BNorm, PReLU, Max(2×2),Conv$(3 \times 3, 64)$, BNorm, PReLU, Conv$(3 \times 3, 64)$, BNorm, PReLU,Full(512), Dropout(70%), Full(1), ReLU.

Our model architecture is motivated by the work of Bosse *et al.* [Bos+16a; Bos+16b]. Early experimentation with their architecture showed that the model had an extremely high capacity for memorization. In order to improve training stability and avoid over-fitting we use a smaller model of only four blocks of 3×3 convolutions with max-pooling [Bou+10] after the second convolution. On the convolutional layers we apply batch normalization with momentum 0.9 [IS15] and Parametric-Rectified Linear Unit (PReLU) activations [He+15b]. After the convolutional blocks the feature space is flattened and fed to a fully connected layer with 512 units. An aggressive dropout [Sri+14] probability of 70% is used to encourage generalization and prevent memorization from the training set. Finally, the fully-connected layer is brought down to a single unit for output. ReLU activation on the output neuron ensures the resulting quality estimate is positive. In total the model contains $8,553,889$ trainable parameters where the majority of these parameters are components of the first fully-connected layer. Figure 5.1 shows the full architecture.

5.3.1 Training

Using the Monte Carlo image database from chapter 4 we construct a training regime for our NR-IQA method. We apply leave-one-scene-out cross validation whereby the model is trained from random initialization using each scene in turn as the validation set while training on the remaining scenes. In this way we can evaluate whether the model is not over-fitting and can generalize to unknown scene compositions, materials, textures, and colour palettes. Table 5.1 shows the sizes of the training and validation sets for each cross validation run.

Validation	Training			Validation		
Scene	Images	Patches	Percent	Images	Patches	Percent
Cornell Box	348	$80,179,200$	76%	109	$25,113,600$	24%
Veach Bidir	337	$77,644,800$	74%	120	$27,648,000$	26%
Veach Door	334	$76,953,600$	73%	123	$28,339,200$	27%
Sponza	352	$81,100,800$	77%	105	$24,192,000$	23%

Tab. 5.1.: Experiment I: Training and Validation set sizes for **leave one scene out** cross validation, before random jitter is applied. For example, when validating on the Cornell Box scene, the training set is comprised of images from the Veach Bidir, Veach Door, and Sponza scenes. The subset of the Monte Carlo image database used contains 457, 512×512 images, with $(512-32)^2 = 230400$ 32×32 patches per image, distributed over four scenes and seven different Monte Carlo rendering algorithms.

At each training step we sample a random mini-batch [Li+14] of 32×32 noisy patches from the training set and compute the true MAE $y_i = MAE(x_i, x'_i)$ of the i^{th} sampled noisy image patch x_i compared to its ground truth patch x'_i. The noisy patches are fed to the network and the approximate MAE of the sampled noisy patch $\hat{y}_i = \mathcal{Q}(x_i, \Theta)$ predicted by the network $\mathcal{Q}$ using the set of trainable weights Θ. We then update the weights Θ via gradient descent w.r.t. a loss function using the Adam optimizer [KB14]. We chose Adam because it adaptively anneals the learning rate as training progresses based on the steepness of the gradient, leading to good convergence without the need to manually tune hyper-parameters.

Formally we train for 32 epochs consisting of 100 minibatches of 1024 32×32 RGB image patches in the range $[0, 1]$ randomly sampled from the training scenes. We train using the Adam optimizer [KB14] with a base learning rate of 0.0005. Training was performed on a single NVidia GTX 1070 GPU and took roughly 6 hours per cross validation fold. Through experimentation we found that when images in the validation scene contained vastly different structural compositions to images in the training set the model was susceptible to over-fitting by memorizing the structural features surrounding image regions of low quality in the training set. To encourage generalization on the naturally distorted image patches we apply random jitter to the inputs in the form of a horizontal flip with 50% probability, and random rotations in jumps of $90°$ with 25% probability each.

However, the model is still prone to over-fitting when validating on image patches from scenes with dramatically different colour palettes than those in the training set. To counter this we also apply a random jitter to the patches in the HSV colour space before calculating the ground truth MAE for the example. The HSV jitter applies a common hue, saturation, and brightness shift to all pixels within the patch and its associated ground truth patch (equation 5.1), where the shift is sampled independently for every training example in the minibatch. During validation batches no jitter is applied.

$$\begin{aligned} H, S, V &= \text{RGBtoHSV}(R, G, B) \\ H' &= (H + \xi_H) \ mod\ 1 \\ S' &= \text{clip}(S + \xi_S, 0, 1) \\ V' &= \text{clip}(V + \xi_V, 0, 1) \\ R', G', B' &= \text{HSVtoRGB}(H', S', V') \\ \text{where} \quad \xi_H \in \mathcal{U}(0,1) &\quad \text{and} \quad \xi_S, \xi_V \in \mathcal{U}(-0.3, 0.3) \end{aligned} \tag{5.1}$$

Figure 5.2 shows the effect of the above rotational and flip jitters in combination with the proposed HSV jitter. Image patches are sampled from the Veach Bidir scene of the Monte Carlo image database. In the top row we see the non-jittered images and in each subsequent row the same source images after random jitter has been applied.

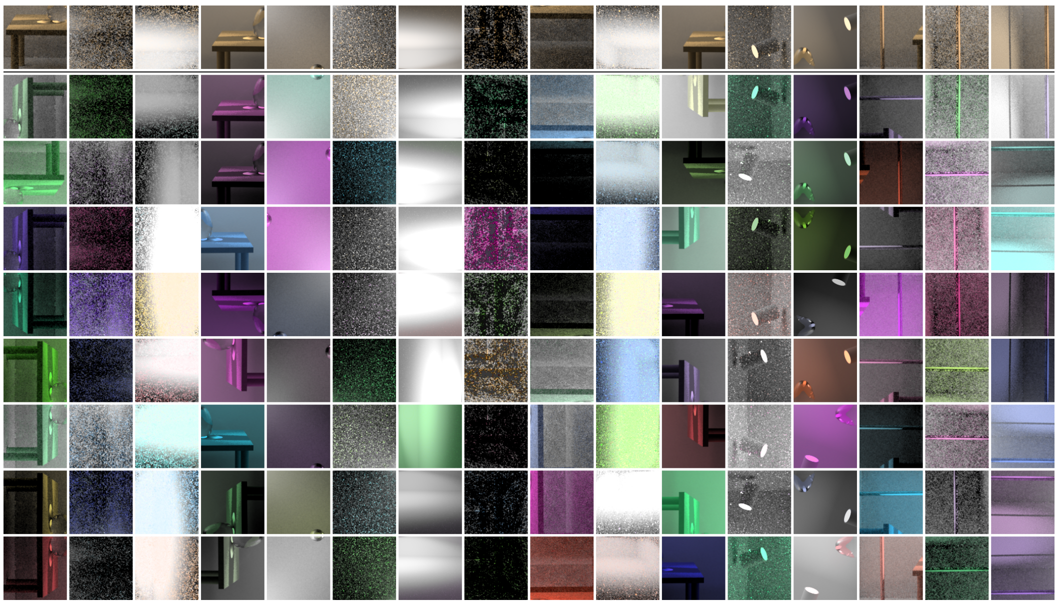


Fig. 5.2.: An example of the proposed jitter exhibiting rotations, flips, and HSV perturbations. Image patches are sampled from the Veach Bidir scene of the Monte Carlo image database. The top row shows the non-jittered images and each subsequent row contains the same source images after random jitter has been applied.

A review of robust loss functions [Bar17] makes a case for the use of the Charbonnier or Pseudo-Huber loss function. The $\mathcal{L}_2$ loss works well for regression problems where the data is balanced and does not contain outliers. However, when this is not the case the square weighting of the $\mathcal{L}_2$ loss can place considerably more importance on outlier training examples, stalling convergence. The $\mathcal{L}_1$ loss avoids this by scaling linearly with divergence, meaning strong outliers do not have as much of an impact on performance. However, the $\mathcal{L}_1$ loss has a discontinuous gradient jump from -1 to 1 at origin which is undesirably coarse and can introduce variance into the training. The Charbonnier loss combines the best of both worlds and behaves like the $\mathcal{L}_1$ loss far from the origin, scaling linearly with divergence, and like the $\mathcal{L}_2$ loss near the origin, giving it a well defined derivative which smoothly transitions from -1 to 1.

$$\begin{aligned} \mathcal{L}_C &= \sqrt{(y-\hat{y})^2+\epsilon^2} \\ \frac{\partial \mathcal{L}_C}{\partial (y-\hat{y})} &= \frac{y-\hat{y}}{\sqrt{(y-\hat{y})^2+\epsilon^2}} \end{aligned} \tag{5.2}$$

The Charbonnier loss and its first derivative parametrised by expected and predicted values y and $\hat{y}$ are shown in equation 5.2; where smoothness of the gradient transition is controlled by an auxiliary ϵ parameter. Experimentally, a small ϵ value of 1×10^{-3} has been shown to produce good results [Bar17]. Figure 5.3 shows a comparison of the Charbonnier, $\mathcal{L}_1$, and $\mathcal{L}_2$ losses and their first order derivatives. For the Charbonnier loss the ϵ parameter has been exaggerated to 1×10^{-1} to highlight the smooth transition that occurs at origin.

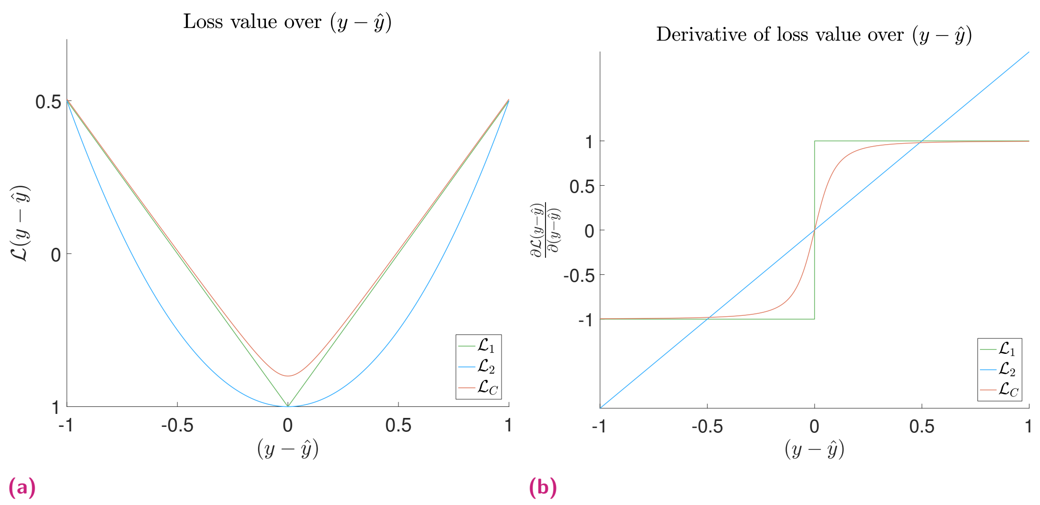


Fig. 5.3.: An example of the Charbonnier, $\mathcal{L}_1$, and $\mathcal{L}_2$ losses (a) and their first order derivatives (b). For the Charbonnier loss the ϵ parameter has been exaggerated to 1×10^{-1} to highlight the smooth transition that occurs at origin.

In the context of an IQA metric a relevant measure of goodness of fit is the Pearson's Correlation Coefficient (PCC). Incorporating this into the loss function we jointly minimize the Charbonnier loss and maximize the absolute batch-wise PCC as shown in equation 5.3.

$$\mathcal{L}_{\text{Joint}} = \delta(1 - |PCC(y, \hat{y})|) + \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}_{\mathcal{C}}(y_i, \hat{y}_i) \tag{5.3}$$

The contribution of PCC to the joint loss is scaled up by $\delta = 1$ to give it equal importance with the Charbonnier loss which is operating on images in the range $[0, 1]$. Incorporating PCC into the loss function has the effect of increasing accuracy and stability during training.

5.3.2 Results and Discussion

After training for 32 epochs as described in section 5.3.1 we report the accuracy of our model on patches from the training and validation sets. Accuracy is calculated using the Pearson's Correlation Coefficient (which measures the relationship between linearly related variables), Spearman's Rank Order Correlation Coefficient (SROCC) (which measures the relationship between monotonically related variables), and Kendall's Tau Rank Order Correlation Coefficient (Kendall's Tau) (which measures the relationship between dependent variables) between the expected and predicted quality values for each patch in the training and validation sets for each cross validation fold.

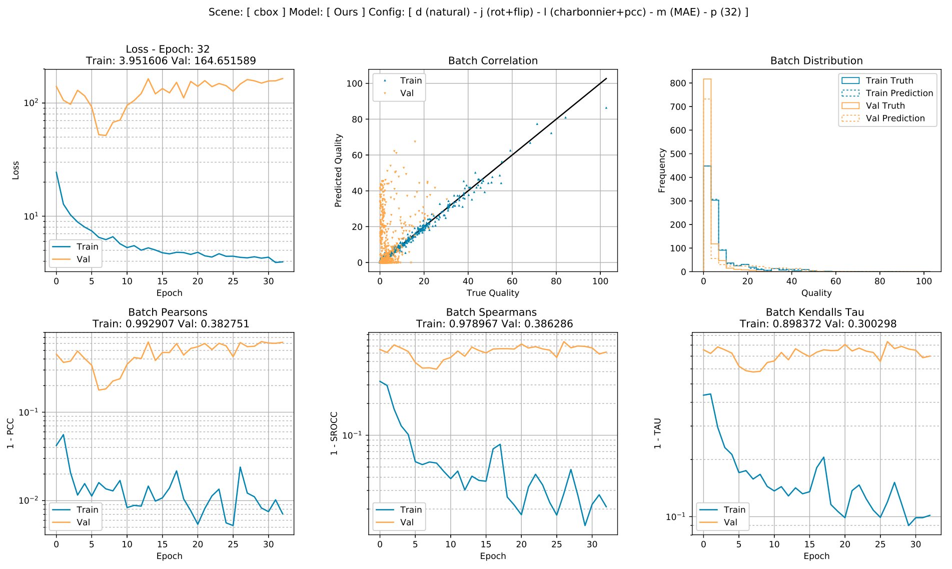
In our first training run we use only the rotational and flip jitter. For the $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\mathcal{C}$ losses the model did not train past the first epoch and suffered from a severe exploding gradient problem, yielding NaN predictions for all inputs. Using the joint loss $\mathcal{L}_{\text{Joint}}$ the model was able to train stably, but was prone to over-fitting on the testing set.

	Validation	Training (Rot+Flip Jitter)			Validation		
Loss	Scene	PCC	SROCC	Tau	PCC	SROCC	Tau
$\mathcal{L}_{\text{Joint}}$	Cornell Box	0.989778	0.972695	0.901166	0.398593	0.410840	0.313313
	Veach Bidir	0.994436	0.971958	0.887453	0.984717	0.978086	0.890768
	Veach Door	0.994979	0.970777	0.885271	0.943044	0.842643	0.698404
	Sponza	0.993877	0.974384	0.894393	0.985955	0.925607	0.813046
	$\mu \pm \sigma$	0.9933 ± 0.0024	0.9725 ± 0.0015	0.8921 ± 0.0072	0.8281 ± 0.2870	0.7893 ± 0.2584	0.6789 ± 0.2562

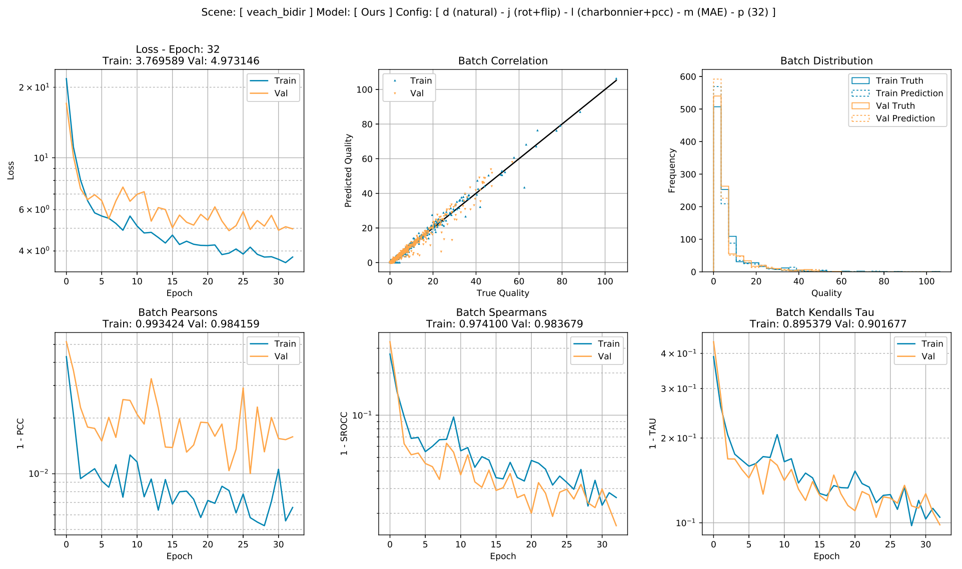
Tab. 5.2.: Experiment I: Training and Validation set accuracies after 32 **epochs** for **leave one scene out** cross validation using **rotation, and flip jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The mean and standard deviation in the reported correlations across the folds are reported in the bottom row. For the $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_\mathcal{C}$ losses the model did not train past the first epoch and suffered from a severe exploding gradient problem.

Table 5.2 shows the result of the first training run using the $\mathcal{L}_{\text{Joint}}$ loss. For the testing sets all four validation folds achieved good and consistent prediction accuracies, with an average and standard deviation of 0.9933 ± 0.0024 for PCC, 0.9725 ± 0.0015

for SROCC, and 0.8921 ± 0.0072 for Kendall's Tau. However, the model did not generalize well to the validation sets which have significantly lower accuracy and variance between folds.



(a) Validation Scene: Cornell Box



(b) Validation Scene: Veach Bidir

Fig. 5.4.: Experiment I: Training and Validation set accuracies curves over 32 **epochs** for **leave one scene out** cross validation using **rotation, and flip jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 32 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 32 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training. Here we show the results for using the (a) Cornell Box and (b) Veach Bidir scenes are the validation scene. It is clearly visible that the model has failed to generalize to the Cornell Box scene when it is excluded from the training set. Full results for all cross validation folds are shown in appendix A.3.1.

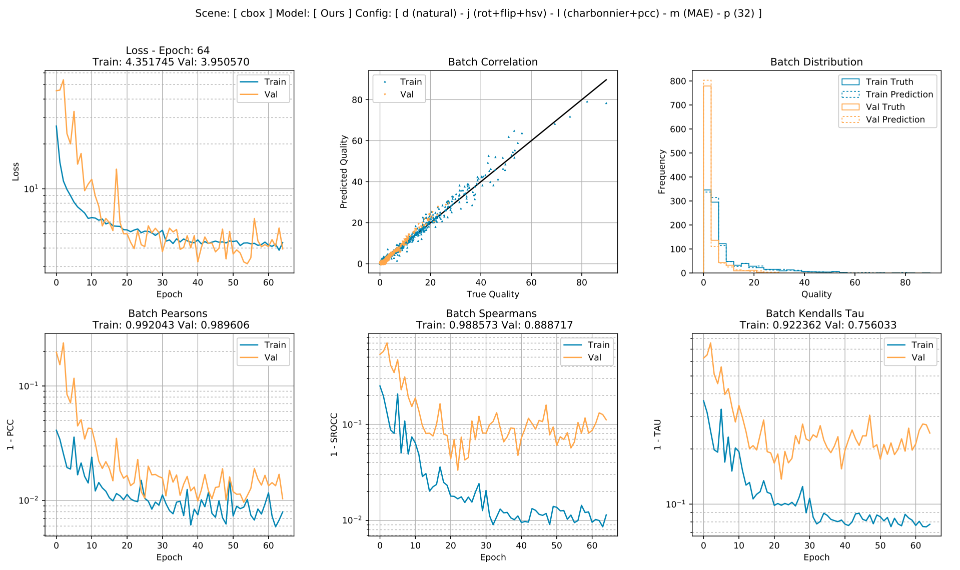
The validation accuracies were significantly lower with averages and standard deviations of 0.8281 ± 0.2870 for PCC, 0.7893 ± 0.2584 for SROCC, and 0.6789 ± 0.2562 for Kendall's Tau. While the accuracies are consistently lower across all cross validation folds, the Cornell Box scene stands out with especially low scores of 0.398593 for PCC, 0.410840 for SROCC, and 0.313313 for Kendall's Tau.

The Cornell Box scene contains a vastly different colour palettes compared to the Veach Bidir, Veach Door, and Sponza scenes which are more muted, containing a variety of grays, soft oranges, and blues. The strong reds and greens of the Cornell Box scene make it difficult for the model to generalize to these image patches, implying that the model has memorized certain colour palettes as being more or less likely to contain certain quality scores given the patches it has seen from the training set. Figure 5.4 shows the training progression of the cross validation folds for the Cornell Box and Veach Bidir scenes. Here we can see that for the Cornell Box scene (figure 5.4a) after the first couple of epochs the model has failed to generalize and is not able to accurately predict the quality of validation image patches. This is visible in the validation loss curve (top-left) and in PCC, SROCC, and Kendall's Tau plots (bottom row). For the Veach Bidir scene (figure 5.4b) the model appears to generalize but this is sporadic across cross validation scenes. Full results for all cross validation folds are shown in appendix A.3.1.

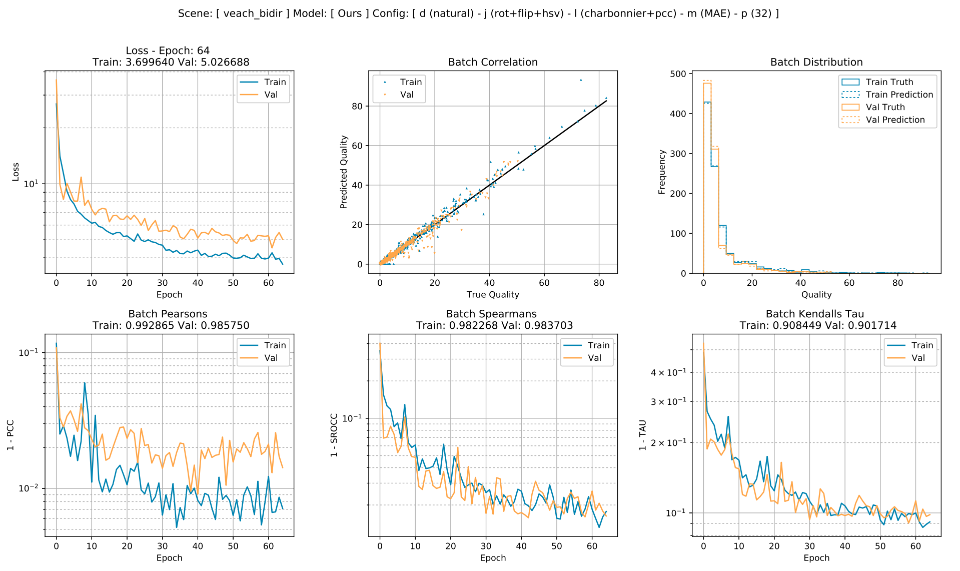
In the second training run we add the HSV jitter in combination with the rotational and flip jitters from the first run. With the $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_C$ losses the model still did not train past the first epoch and again suffered from a severe exploding gradient problem, yielding NaN predictions for all inputs. Using the joint loss $\mathcal{L}_{\text{Joint}}$ the model was able to train stably, and with the addition of the HSV jitter was able to generalize much better to the validation sets. We hypothesize that the Pearson's Correlation Coefficient in the joint loss function is acting as a regularization term, limiting the extrema of the gradient in minibatch updates which stabilizes training. Now that the model is generalizing properly, we extend the number of epochs used for training each cross validation fold to 64 epochs.

Table 5.3 shows the result of the second training run using the $\mathcal{L}_{\text{Joint}}$ loss. For the testing sets all four validation folds achieved good and consistent prediction accuracies, with an average and standard deviation of 0.9903 ± 0.0015 for PCC, 0.9789 ± 0.0069 for SROCC, and 0.9019 ± 0.0103 for Kendall's Tau; showing a modest improvement on training set accuracies without the HSV jitter.

The validation set accuracies, however, show a major improvement in quality and consistency. For all four scenes (including the Cornell Box scene) accuracy was largely consistent with that seen on the training sets. Across cross validation folds validation accuracy had a average and standard deviation of 0.9871 ± 0.0031 for PCC, 0.9593 ± 0.0390 for SROCC, and 0.8604 ± 0.0663 for Kendall's Tau.



(a) Validation Scene: Cornell Box



(b) Validation Scene: Veach Bidir

Fig. 5.5.: Experiment I: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 64 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 64 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training. Here we show the results for using the (a) Cornell Box and (b) Veach Bidir scenes are the validation scene. We can see that with the addition of the HSV jitter that the model is now able to generalize to the Cornell Box scene when it is excluded from the training set. Full results for all cross validation folds are shown in appendix A.3.2.

	Validation	Training (Rot+Flip+HSV Jitter)			Validation		
Loss	Scene	PCC	SROCC	Tau	PCC	SROCC	Tau
$\mathcal{L}_{\text{Joint}}$	Cornell Box	0.990588	0.988802	0.915202	0.989007	0.900845	0.761066
	Veach Bidir	0.991008	0.973419	0.893524	0.982550	0.976555	0.892524
	Veach Door	0.991432	0.978147	0.904963	0.987315	0.977666	0.891403
	Sponza	0.988167	0.975147	0.893937	0.989405	0.982082	0.896622
	$\mu \pm \sigma$	0.9903 ± 0.0015	0.9789 ± 0.0069	0.9019 ± 0.0103	0.9871 ± 0.0031	0.9593 ± 0.0390	0.8604 ± 0.0663

Tab. 5.3.: Experiment I: Training and Validation set accuracies after 64 **epochs** for **leave one scene out** cross validation using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The mean and standard deviation in the reported correlations across the folds are reported in the bottom row. For the $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_{\mathcal{C}}$ losses the model did not train past the first epoch and suffered from a severe exploding gradient problem.

Figure 5.5 shows the training progression of the cross validation folds for the Cornell Box and Veach Bidir scenes. We can see that the addition of HSV jitter has encouraged the model to generalize when the Cornell Box scene is used as the validation scene, making it consistent with the other cross validation folds (figure 5.4a). The validation loss curve (top-left) and the PCC, SROCC, and Kendall's Tau curves (bottom row) are smoother and more consistent with the training set curves. For the Veach Bidir scene (figure 5.4b) the model performs just as well. An interesting observation is that for the Cornell Box scene the model has better validation set PCC and worse SROCC and Kendall's Tau than the training set values, and for the Veach Bidir scene the model has worse PCC and better SROCC and Kendall's Tau values. Full results for all cross validation folds are shown in appendix A.3.2.

From this experiment we conclude that the HSV jitter is successful in discouraging the network from using the colour palettes of images from the training sets as an indication of quality. Encouraging it to instead use variational and structural cues to determine this information. This allows the model to generalize well to scene compositions drastically different from those in the training set.

5.3.2.1. Patch Based NR-IQA

Our model operates on fixed sized 32×32 RGB image patches. In order to apply the model to images of arbitrary size we use a strided sliding window approach. A prediction is made for each patch extracted from the noisy input image, approximating the MAE of the noisy patch compared to its unknown ground truth patch. These predictions can then be used either as a map of relative quality across the image or the predictions can be pooled and used as an overall quality index for the image.

There is a trade-off on the choice of stride value, between increasing computational complexity for small strides, and decreasing IQA accuracy for larger stride values. For a stride of one pixel, the network must make a prediction for all $(512-32)^2 = 230400$ patches in the input image. This can be accelerated by grouping the extracted patches into fixed sized batches and processing them simultaneously. However, this can still lead to undesirably slow evaluation times. Using a more conservative stride of 8 pixels, the network only needs to process $\left(\frac{512-32}{8}\right)^2 = 3600$ patches per image, greatly increasing evaluation efficiency at the expense of IQA robustness.

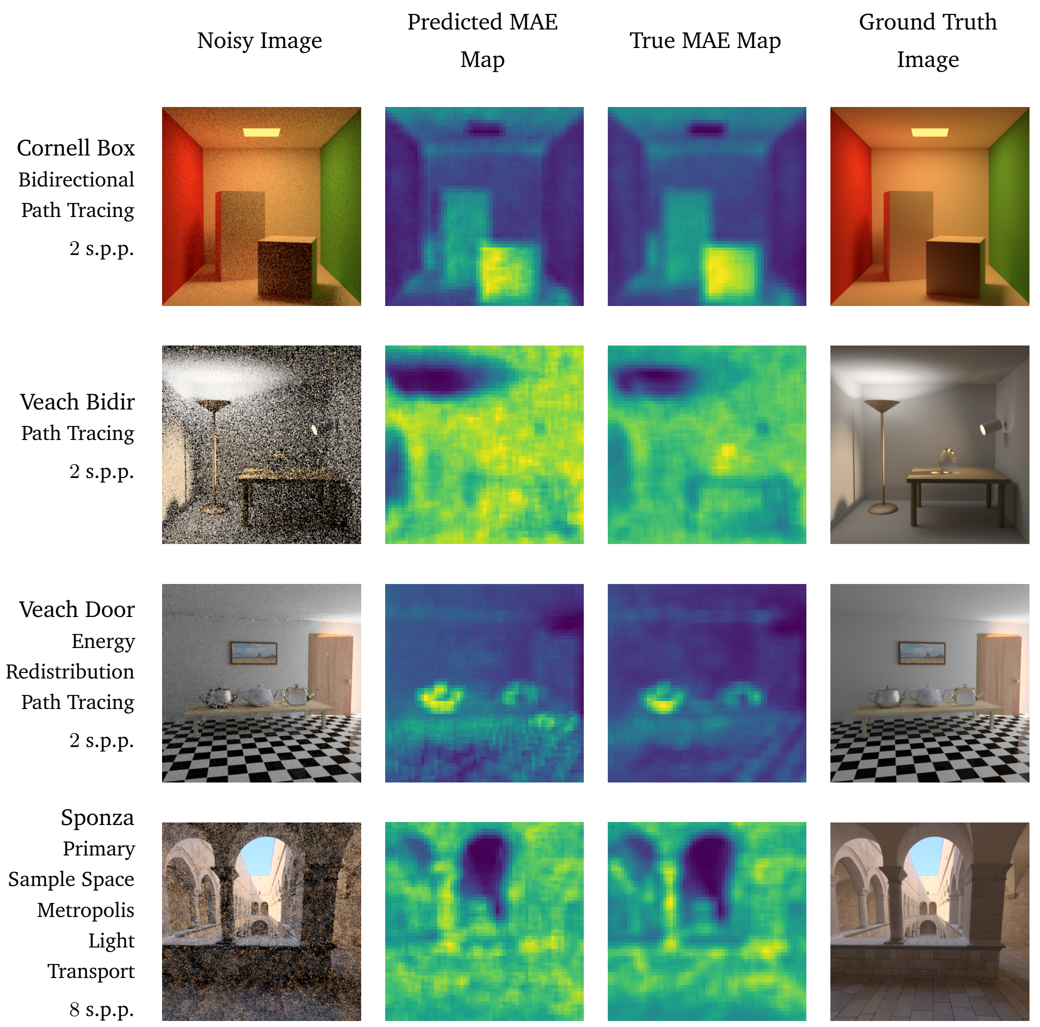

Noisy Image
Predicted MAE Map
True MAE Map
Ground Truth Image
Cornell Box Bidirectional Path Tracing 2 s.p.p.
Veach Bidir Path Tracing 2 s.p.p.
Veach Door Energy Redistribution Path Tracing 2 s.p.p.
Sponza Primary Sample Space Metropolis Light Transport 8 s.p.p.

Fig. 5.6.: Experiment II: Example IQA prediction from our model on images from the validation sets. From left to right: The input noisy image to evaluate. The predicted MAE of the noisy image computed patchwise with stride 8. The true MAE of the noisy image compared to the ground truth image, computed patchwise with stride 8. The ground truth image.

Figure 5.6 shows the predicted MAE maps for images from the validation scenes of each cross validation fold. The top row shows an image of the Cornell Box scene, rendered with Bidirectional Path Tracing at 2 s.p.p.. The prediction map accurately captures the distribution of error across the image, specifically in regions such as the near face of the short box (right), and the soft shadow behind the tall box (left).

In the second row we have an image for the Veach Bidir scene, rendered with Path Tracing at 2 s.p.p.. The predicted map shows some overestimation of error in areas lit primarily by diffuse indirect illumination. The third row, we see an image of the Veach Door scene rendered with Energy Redistribution Path Tracing at 2 s.p.p.. For this scene the prediction map is able to correctly capture the visible distortions around the teapot models and in corner regions of the room. Finally, in the fourth row we see an image of the Sponza scene rendered with Primary Sample Space Metropolis Light Transport at 8 s.p.p.. The prediction map accurately captures the distribution of error across the image with some minor overestimation on the floor in the foreground of the image.

In order to compare our proposed NR-IQA model to existing methods we apply mean pooling to the resulting prediction maps from patch-wise strided evaluation of target images to give a single scalar valued quality score. Using the predicted values from our method and from the existing methods, and the ground truth quality scores computed using an FR-IQA method such as MAE, we compute the correlation of the predicted values compared to the FR-IQA values and report these results for each validation scene, using the version of our method trained on each cross validation fold, respectively.

Table 5.4 shows the results of our model applied patch-wise with stride 8 compared to existing NR-IQA algorithms. Our method consistently performs best across all four validation scenes. The BIQI [MB10], BRISQUE [Mit+12a], HIGRADE 1 [Kun+16a], HIGRADE 2 [Kun+16a], JP2K-NR [She+05b], NIQE [Mit+13a], OG-IQA [Liu+16] NR-IQA measures all exhibit both positive and negative correlations for different validation scenes which indicates that they are not accurately describing the underlying distribution of visible distortions present in the test images.

	Cornell Box			Veach Bidir			Veach Door			Sponza		
Metric	PCC	SROCC	Tau	PCC	SROCC	Tau	PCC	SROCC	Tau	PCC	SROCC	Tau
Ours - Experiment I	**0.9951**	**0.9919**	**0.9425**	**0.9963**	**0.9928**	**0.9501**	**0.9965**	**0.9887**	**0.9258**	**0.9990**	**0.9949**	**0.9623**
BLIINDS [Saa+10]	-0.9912	-0.9870	-0.9297	-0.9289	-0.9525	-0.8701	-0.7388	-0.9770	-0.8941	-0.9706	-0.9596	-0.8581
BIQI [MB10]	0.1087	0.0396	0.0010	-0.0515	0.3481	0.2508	0.8700	0.3204	0.2085	0.4092	0.4945	0.3491
BRISQUE [Mit+12a]	-0.3953	-0.6787	-0.5327	-0.0872	-0.3458	-0.2258	0.2516	-0.4082	-0.3391	0.6609	0.3566	0.3036
HIGRADE 1 [Kun+16a]	-0.2284	0.0986	0.0651	0.5495	0.6105	0.4420	-0.7064	0.3329	0.2860	-0.2662	-0.0004	-0.0284
HIGRADE 2 [Kun+16a]	-0.3893	-0.4237	-0.3150	0.5235	0.8505	0.6573	-0.6338	-0.1623	-0.1039	-0.2010	-0.2593	-0.1770
JP2K-NR [She+05b]	0.4325	0.9257	0.8087	0.3941	0.9095	0.7449	-0.0315	-0.2605	-0.1386	0.5007	0.8064	0.6397
NIQE [Mit+13a]	-0.6218	-0.9465	-0.8197	-0.4744	-0.8691	-0.7123	0.4141	-0.4561	-0.3817	0.7032	0.2801	0.2289
OG-IQA [Liu+16]	-0.3272	-0.4793	-0.3219	0.2447	0.1993	0.1833	0.7769	0.2168	0.1660	0.8336	0.6145	0.4481

Tab. 5.4.: Experiment I: Validation set NR-IQA performance for **leave one scene out** cross validation. Each correlation is computed relative to the full reference MAE of each 512×512 image in the validation set compared to its ground truth image.

5.4 Experiment II

A shortcoming of the network design from experiment I is that each patch in an image must be shown to the network individually. For a 512×512 image this results in

262144 image patches which need evaluation. Even when batch processing multiple patches simultaneously and when accelerating evaluation with GPU hardware this is slow compute.

One solution to this problem is to use a Fully-Convolutional Neural Network (FCNN) architecture. This class of model were first proposed for the task of dense image segmentation [Lon+15] where the goal is to perform multi-class classification on each individual pixel of an input image. In this work the authors took existing pre-trained models from AlexNet [Kri+12], VGG [SZ14], and GoogLeNet [Sze+14] and modified the final layers to make the networks fully convolutional. The weights on the original networks were frozen and the new output layers trained to perform dense segmentation before a final fine tuning was performed on all layers of the network. Because the output of a fully convolutional network does not have a fixed size, and whose size is only dependent on the input tensor size this means that once the network is trained it can perform segmentation on images of arbitrary size. This, conveniently, means that during training images can be of a fixed size and the can be grouped into minibatches, greatly accelerating and stabilizing training performance.

In our second experiment we wish to see whether a fully convolutional neural network can be trained to densely regress the error in Monte Carlo rendered images at each pixel simultaneously using only local neighbourhood information from the distorted image. Additionally we address a shortcoming of the original experiment where the target of the regression was the MAE of 32×32 image patches. Local patch wise MAE was a convenient target for the regression due to its simplicity and efficiency of computation. However, when tiling this measure at each pixel and mean pooling the result over a full image this does not form a robust IQA measure (chapter 4). In the second experiment we move to the SSIM [Wan+04a] IQA measure as the target of the regression. Internally, SSIM uses (by default) an 11×11 pixel neighbourhood over which it performs local statistics on the means, variances, and covariance between the reference and test images. This yields an SSIM map which is the same resolution as the input images which can be mean pooled to provide a robust single valued IQA measure. Given a noisy test image of an arbitrary resolution we aim to predict the value of the SSIM map at each pixel without access to the ground truth image. The predicted SSIM map can then be mean pooled to provide a single value representing predicted image quality.

Reducing the size of the receptive field from 32 to 11×11 pixels and moving to a more complex regression target increases the learning difficulty for the network. Because of this, our proposed network architecture is substantially larger than the one used for experiment I.

Our proposed model is motivated by prior work in image denoising and NR-IQA. In the work by Zhang *et al.* [Zha+17b] image denoising is performed using an FCNN by densely regressing a residual image. Their simple feed-forward architecture, composed from blocks of convolutional and regularization layers, is able to capture complex information about the structure of natural images. We also draw from the work of Li *et al.* [Li+11] who applied an MLP with hand crafted features independently to the features extracted from each pixel, in order to regress the DMOS of natural images under synthetic distortions.

From this we propose a feed-forward FCNN model with two stages. In the first stage we extract image features from the local neighbourhood around each pixel, then in the second stage we apply a series of dense layers to each pixel independently. Normally, dense layers force the size of the output tensor to be a constant size. However, this is undesirable for our use case as we would like to be able to perform dense inference on images of any size after the model has been trained. In FCNN models applying convolutions with a filter size of 1×1 has the same effect as applying a dense layer to the features of each pixel individually. By phrasing this as a convolution the operation remains invariant to the size of the input, meaning inputs of arbitrary size can still be fed through the trained network to attain a prediction.

The model begins with four convolutional blocks, each using 3×3 convolutions with 256 output feature channels, each using batch normalization and ReLU activations. This forms the feature extractor part of the network and raises the receptive field of the network up to 11×11. This part of the network is responsible for extracting and combining information about the local structure and composition of the input image.

The next part of the network takes these features which are defined independently on each pixel and combines them through a series of non-linear combinations to compute a predicted image quality score for every pixel in the image in parallel. To combine our feature channels down to a single quality score for each pixel, we apply six convolutional layers using 1×1 with 128 output feature channels, each using batch normalization and ReLU activations.

Finally we apply a last convolutional layer using 1×1 filters with a single output feature channel to reduce the feature channels for each pixel down to a final regression prediction. Due to the large size of the new network architecture, we chose standard ReLU activations as opposed to PReLU used in experiment I because of the large number of trainable parameters PReLU adds to the network and their effect on training time. The SSIM map can contain values in the range $[-1, 1]$, however in practice values below zero rarely appear in our training data and greater network performance can be achieved by clipping the target SSIM maps to be in the range $[0, 1]$ before computing the loss function. Because we only consider positive values for the regression we apply a ReLU activation on the output prediction map from the final convolution to ensure all network predictions are positive. In total the model contains $1,896,577$ trainable parameters, just 22.17% the number of parameters used by the patch based model in experiment I. 93.9% of weights are used in the initial set of 3×3 feature extracting convolutions in the first four blocks of the network, while the remaining six 1×1 convolutional blocks account for just 6.1% of the parameters. Figure 5.7 shows the fully convolutional model architecture.

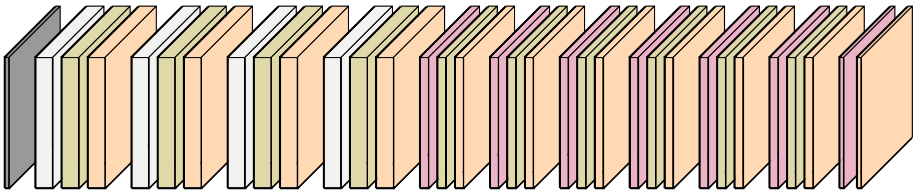


Fig. 5.7.: Experiment II: Our fully convolutional model. The model consists of four 3×3 convolutional blocks, each with batch normalization and ReLU activation; followed by six dense 1×1 convolutional blocks, each with batch normalization and ReLU activation. The output neuron has ReLU activation to ensure the resulting quality scores are positive. The full model is: Input$(32 \times 32, 3)$, Conv$(3 \times 3, 256)$, BNorm, ReLU, Conv$(3 \times 3, 256)$, BNorm, ReLU, Conv$(3 \times 3, 256)$, BNorm, ReLU, Conv$(3 \times 3, 256)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 128)$, BNorm, ReLU, Conv$(1 \times 1, 1)$, ReLU

5.4.1 Training

Training is performed in that same manner as in experiment I. Using the Monte Carlo image database we apply leave-one-scene-out cross validation whereby the model is trained from random initialization using each scene in turn as the validation set while training on the remaining scenes.

At each training step we sample a random mini-batch [Li+14] of 64×64 noisy patches from the training set and compute the true SSIM map $y_i = SSIM_{map}(x_i, x'_i)$ of the i^{th} sampled noisy image patch x_i compared to its ground truth patch x'_i. The noisy patches are fed to the network and the approximate SSIM map of the sampled noisy patch $\hat{y}_i = \mathcal{Q}(x_i, \Theta)$ predicted by the network $\mathcal{Q}$ using the set of trainable weights Θ. We then update the weights Θ via gradient descent w.r.t. a loss function using the Adam optimizer [KB14].

Validation Scene	Training Images	Training Patches	Training Percent	Validation Images	Validation Patches	Validation Percent
Cornell Box	348	$69,844,992$	76%	109	$21,876,736$	24%
Veach Bidir	337	$67,637,248$	74%	120	$24,084,480$	26%
Veach Door	334	$67,035,136$	73%	123	$24,686,592$	27%
Sponza	352	$70,647,808$	77%	105	$21,073,920$	23%

Tab. 5.5.: Experiment II: Training and Validation set sizes for **leave one scene out** cross validation, before random jitter is applied. For example, when validating on the Cornell Box scene, the training set is comprised of images from the Veach Bidir, Veach Door, and Sponza scenes. The subset of the Monte Carlo image database used contains 457, 512×512 images, with $(512-64)^2 = 200704$ 64×64 patches per image, distributed over four scenes and seven different Monte Carlo rendering algorithms.

Formally we train for 1024 epochs consisting of 256 minibatches of 16 64×64 RGB image patches in the range $[0, 1]$ randomly sampled from the training scenes. We train using the Adam optimizer [KB14] with a base learning rate of 0.001. Training was performed on a single NVidia GTX 1070 GPU and took roughly 4 hours per cross validation fold. As with experiment I we found that when images in the validation scene contained vastly different structural compositions to images in the training set the model was susceptible to over-fitting by memorizing the structural features and colour information surrounding image regions of low quality in the training set. To encourage generalization on the naturally distorted image patches we apply random jitter to the inputs in the form of a horizontal flip with 50% probability, and random rotations in jumps of $90°$ with 25% probability each. We also apply the HSV jitter which applies a common hue, saturation, and brightness shift to all pixels within the patch and its associated ground truth patch (equation 5.1), where the shift is sampled independently for every training example in the minibatch. During validation batches no jitter is applied. Table 5.5 shows the sizes of the training and validation sets for each cross validation run.

5.4.2 Results and Discussion

As with the results of experiment I (section 5.3.2) we report the accuracy of our model on patches from the training and validation sets. Accuracy is calculated using the PCC, SROCC, and Kendall's Tau between the expected and predicted quality values for each patch in the training and validation sets for each cross validation fold.

In the first training run for experiment II we use the rotational, flip, and HSV jitters. With the fully convolutional model the $\mathcal{L}_1$, $\mathcal{L}_2$, and $\mathcal{L}_C$ losses were able to train reasonably stably without the exploding gradient problem seen in the model from experiment I. As an exploratory step we trained our new model for 128 epochs using each of the four loss functions so as to compare their effects on training stability and efficiency.

Table 5.6 shows the result of the first training run using each of the four loss functions. All four losses perform reasonably well and do not show signs of major over-fitting. On the validation sets, the $\mathcal{L}_{\text{Joint}}$ loss has the highest average accuracy and has consistently small standard deviations across cross validation folds for each correlation measure.

	Validation	Training (Rot+Flip+HSV Jitter)			Validation		
Loss	Scene	PCC	SROCC	Tau	PCC	SROCC	Tau
$\mathcal{L}_1$	Cornell Box	0.972193	0.973177	0.867047	0.982244	0.860295	0.700377
	Veach Bidir	0.966749	0.968851	0.865146	0.978072	0.975443	0.888591
	Veach Door	0.978891	0.969311	0.859785	0.944208	0.953709	0.835708
	Sponza	0.977722	0.982417	0.897448	0.961138	0.961118	0.838531
	$\mu \pm \sigma$	0.9739 ± 0.0056	0.9734 ± 0.0063	0.8724 ± 0.0170	0.9664 ± 0.0174	0.9376 ± 0.0523	0.8158 ± 0.0807
$\mathcal{L}_2$	Cornell Box	0.975399	0.968849	0.855512	0.988992	0.858504	0.703183
	Veach Bidir	0.974397	0.933385	0.792890	0.982448	0.958809	0.842332
	Veach Door	0.979344	0.951626	0.822898	0.937276	0.933206	0.805445
	Sponza	0.967688	0.947931	0.820128	0.956717	0.960155	0.828935
	$\mu \pm \sigma$	0.9742 ± 0.0048	0.9504 ± 0.0146	0.8229 ± 0.0256	0.9664 ± 0.0239	0.9277 ± 0.0477	0.7950 ± 0.0631
$\mathcal{L}_\mathcal{C}$	Cornell Box	0.965823	0.964170	0.846650	0.984243	0.896299	0.741704
	Veach Bidir	0.964259	0.960643	0.846469	0.977131	0.981226	0.895550
	Veach Door	0.975903	0.979250	0.887376	0.925470	0.931493	0.821929
	Sponza	0.970519	0.962865	0.848396	0.957923	0.956690	0.825556
	$\mu \pm \sigma$	0.9691 ± 0.0052	0.9667 ± 0.0085	0.8572 ± 0.0201	0.9612 ± 0.0263	0.9414 ± 0.0363	0.8212 ± 0.0629
$\mathcal{L}_{\text{Joint}}$	Cornell Box	0.974092	0.975930	0.874233	0.988965	0.954679	0.830864
	Veach Bidir	0.973101	0.961212	0.840273	0.975710	0.972310	0.862354
	Veach Door	0.979507	0.978489	0.883210	0.942559	0.950619	0.848439
	Sponza	0.975069	0.968283	0.859585	0.964857	0.968420	0.850224
	$\mu \pm \sigma$	0.9754 ± 0.0028	0.9710 ± 0.0078	0.8643 ± 0.0188	0.9680 ± 0.0196	0.9615 ± 0.0105	0.8480 ± 0.0130

Tab. 5.6.: Experiment II: Training and Validation set accuracies after 128 **epochs** for **leave one scene out** cross validation using **rotation, flip, and HSV jitter**. The cross validation folds were computed for each of the four loss functions under test, and for each of these configurations the mean and standard deviation in the reported correlations across the folds are reported in the bottom row of each section. The $\mathcal{L}_\mathcal{C}$ loss shows only a minor improvement over the $\mathcal{L}_2$ and $\mathcal{L}_1$ losses; however, the $\mathcal{L}_{\text{Joint}}$ loss shows significantly more consistent results and has lower standard deviations over the cross validation folds.

While all four loss functions perform similarly when comparing validation PCC values with 0.9664 ± 0.0174 for $\mathcal{L}_1$, 0.9664 ± 0.0239 for $\mathcal{L}_2$, 0.9612 ± 0.0263 for $\mathcal{L}_\mathcal{C}$, and 0.9680 ± 0.0196 for $\mathcal{L}_{\text{Joint}}$; the difference between SROCC and Kendall's Tau values are more pronounced. For SROCC we observe values of 0.9376 ± 0.0523 for $\mathcal{L}_1$, 0.9277 ± 0.0477 for $\mathcal{L}_2$, 0.9414 ± 0.0363 for $\mathcal{L}_\mathcal{C}$, and 0.9615 ± 0.0105 for $\mathcal{L}_{\text{Joint}}$, and for Kendall's Tau we see values of 0.8158 ± 0.0807 for $\mathcal{L}_1$, 0.7950 ± 0.0631 for $\mathcal{L}_2$, 0.8212 ± 0.0629 for $\mathcal{L}_\mathcal{C}$, and 0.8480 ± 0.0130 for $\mathcal{L}_{\text{Joint}}$. For these two measures we see that the $\mathcal{L}_{\text{Joint}}$ loss is on average better by $2.01 - 3.38\%$ for SROCC and $2.68 - 5.3\%$ for Kendall's Tau over the cross validation folds and has a lower standard deviation between folds. This shows that even when the regularization properties of incorporating PCC into the loss function are not explicitly needed to avoid the exploding gradient problem, it is still effective in improving training stability and accuracy.

Selecting $\mathcal{L}_{\text{Joint}}$ as the best performing loss function we continued training the model to 1024 epochs to improve the accuracy to effective convergence (table 5.7). Doing so only gave a modest improvement in accuracy on the validation sets of $+0.39\%$ in PCC, $+0.56\%$ in SROCC, and $+2.21\%$ in Kendall's Tau.

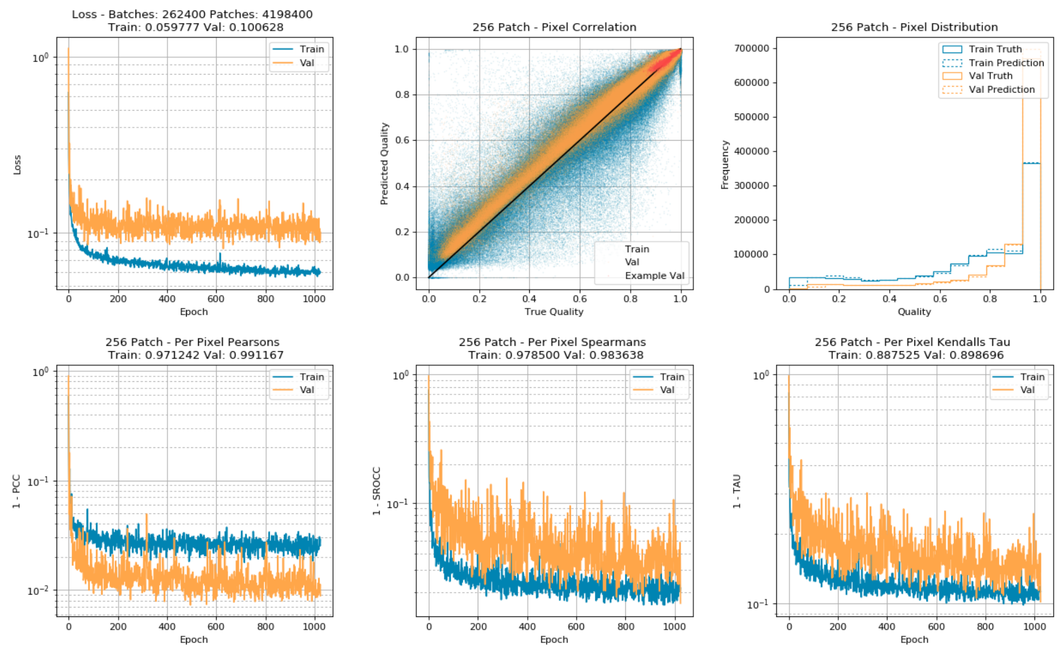
	Validation	Training (Rot+Flip+HSV Jitter)			Validation		
Loss	Scene	PCC	SROCC	Tau	PCC	SROCC	Tau
$\mathcal{L}_{\text{Joint}}$	Cornell Box	0.974825	0.979942	0.887806	0.988769	0.963563	0.858154
	Veach Bidir	0.978509	0.982749	0.894389	0.988179	0.983878	0.914428
	Veach Door	0.982456	0.986308	0.909126	0.951672	0.957310	0.864841
	Sponza	0.981203	0.986476	0.915918	0.958848	0.963735	0.842833
	$\mu \pm \sigma$	0.9792 ± 0.0034	0.9839 ± 0.0031	0.9018 ± 0.0130	0.9719 ± 0.0194	0.9671 ± 0.0116	0.8701 ± 0.0310

Tab. 5.7.: Experiment II: Training and Validation set accuracies after 1024 **epochs** for **leave one scene out** cross validation using **rotation, flip, and HSV jitter**.

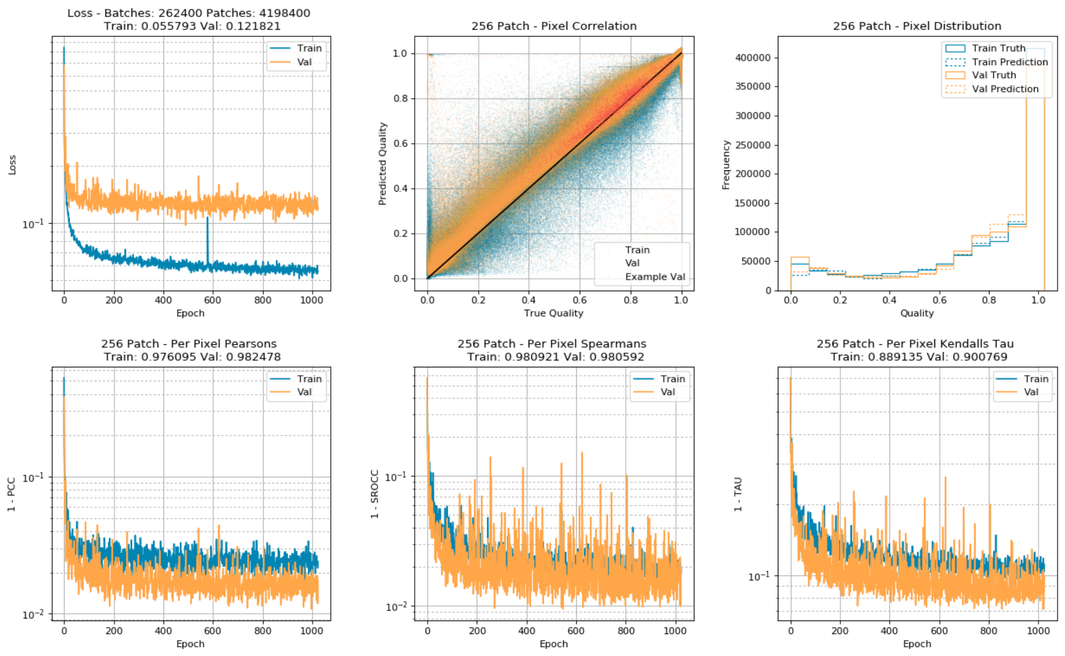
Figure 5.8 shows the training progression of the cross validation fold for the Cornell Box scenes using the $\mathcal{L}_{\text{Joint}}$ loss. We can observe that compared to the scatter plots for the same configuration in experiment I (figure 5.5) the new fully convolutional model appears to have higher variance. This is largely due to the fact that in experiment I quality scores were averaged over 32×32 sized image patches, while in experiment II we show the prediction for individual pixels as data points in the scatter plots (top-centre). Another difference can be seen in the utilization of the output domain. In experiment I predicted quality values appear bunched up towards zero, with a long tailed falloff to the right, and an unbounded maximum value. Due to the use of SSIM as the regression target the new model consistently predicts quality values in the range $[0, 1]$ where the output space is used uniformly w.r.t. image quality. This has the benefit of being more readily interpretable than MAE. Full results for all loss functions and cross validation folds are shown in appendix A.4.

In figure 5.9 we see the predicted SSIM maps for images from the validation scenes of each cross validation fold. The images shown are the same ones used in figure 5.6 of experiment I for comparison purposes. Compared to the results of experiment I we can see that by moving from a 32×32 receptive field sampled with stride 8 from the image to a smaller 11×11 receptive field sampled at stride 1, the new fully convolutional model is able to capture a much clearer understanding of the distribution of noise in rendered images.

With this increased clarity, fine details such as quality around the edges of geometry are more faithfully predicted by the network, even when they are heavily occluded by noise in the region. In the bottom row of the grid we see an image of the Sponza



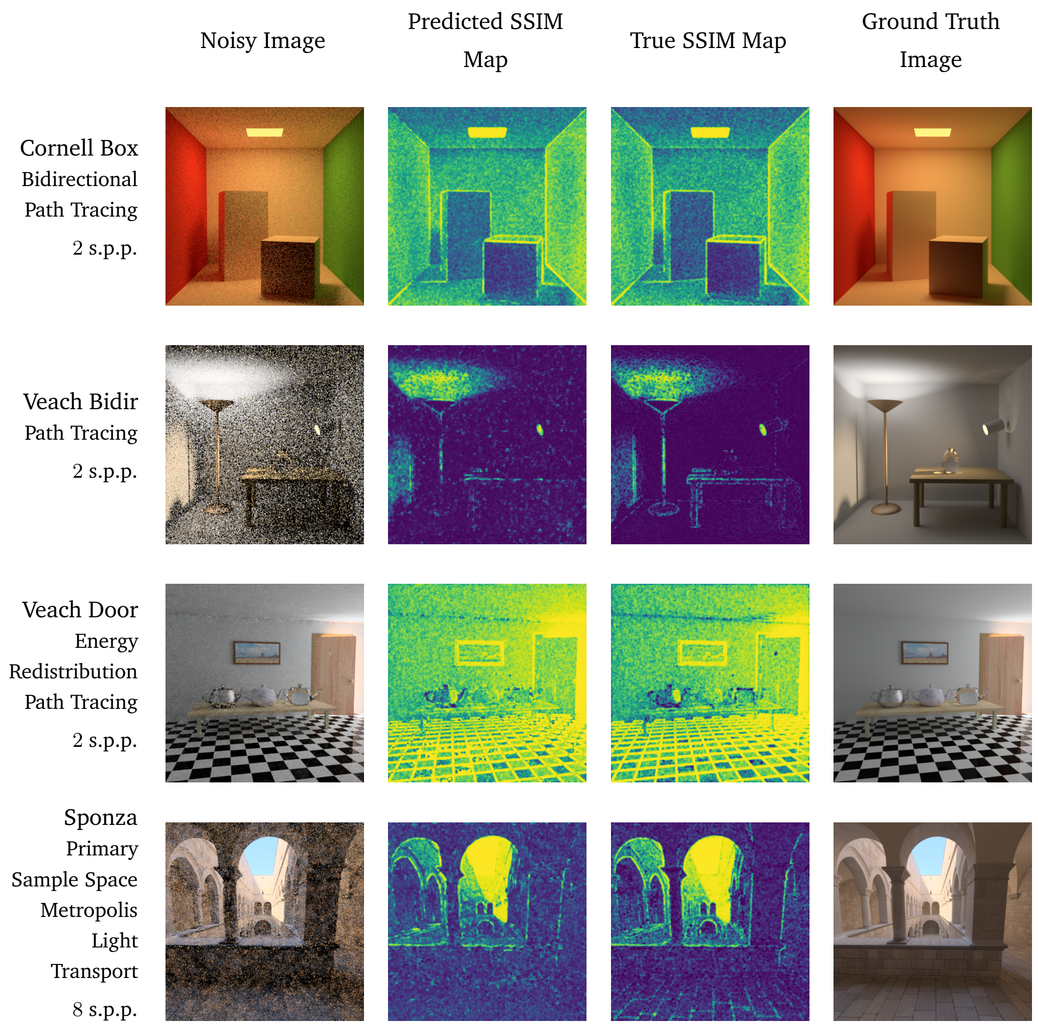
(a) Scene: Cornell Box



(b) Scene: Veach Bidir

Fig. 5.8.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for the Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 1024 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 1024 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training. The scatter plots show higher variance than those from experiment I (figure 5.5) due to lack of 32×32 average pooling accross patches from the original experiment. Full results for all loss functions and cross validation folds are shown in appendix A.4.

scene rendered with Primary Sample Space Metropolis Light Transport at 8 s.p.p.. In this image we see the edges of floor tiles which are heavily occluded by noise. For these edges the network consistently underestimates the SSIM quality in the local region. This is potentially due to subtle differences in the distribution of noise on and around geometric edges and texture induced edges from the Monte Carlo rendering process. A possible way to improve the prediction accuracy in these regions would be to feed the network and additional input of local texture albedo for geometry visible within the image.



Noisy Image
Predicted SSIM Map
True SSIM Map
Ground Truth Image
Cornell Box Bidirectional Path Tracing 2 s.p.p.
Veach Bidir Path Tracing 2 s.p.p.
Veach Door Energy Redistribution Path Tracing 2 s.p.p.
Sponza Primary Sample Space Metropolis Light Transport 8 s.p.p.

Fig. 5.9.: Experiment II: Example IQA prediction from our model on images from the validation sets. From left to right: The input noisy image to evaluate. The predicted SSIM of the noisy image. The true SSIM of the noisy image compared to the ground truth image. The ground truth image.

Finally, in order to compare our proposed fully convolutional NR-IQA model to existing methods we feed each 512×512 image in the validation sets to the network and apply mean pooling to the resulting prediction maps to give a single scalar valued quality score. Using the predicted values from our method and from the existing methods, and the ground truth quality scores computed using the SSIM FR-IQA method; we compute the correlation of the predicted values compared to the FR-IQA values and report these results for each validation scene, using the version of our method trained on each cross validation fold, respectively.

	Cornell Box			Veach Bidir			Veach Door			Sponza		
Metric	PCC	SROCC	Tau	PCC	SROCC	Tau	PCC	SROCC	Tau	PCC	SROCC	Tau
Ours - Experiment II	**0.9996**	**0.9959**	**0.9688**	**0.9982**	**0.9921**	**0.9393**	**0.9928**	**0.9916**	**0.9331**	**0.9989**	**0.9964**	**0.9686**
BLIINDS [Saa+10]	0.9840	0.9858	0.9287	0.9619	0.9544	0.8650	0.9640	0.9870	0.9144	0.9066	0.9668	0.8723
BIQI [MB10]	-0.2150	-0.0472	-0.0145	-0.0365	-0.3395	-0.2411	-0.7170	-0.3449	-0.2462	-0.4219	-0.5132	-0.3693
BRISQUE [Mit+12a]	0.3596	0.6854	0.5414	0.2211	0.3402	0.2172	0.1214	0.4154	0.3546	-0.5622	-0.3427	-0.2879
HIGRADE 1 [Kun+16a]	0.2321	-0.1136	-0.0779	-0.7522	-0.6113	-0.4465	0.3415	-0.3348	-0.2901	0.2163	-0.0028	0.0246
HIGRADE 2 [Kun+16a]	0.4008	0.4236	0.3167	-0.7526	-0.8591	-0.6761	0.4064	0.1528	0.1009	0.4042	0.2663	0.1867
JP2K-NR [She+05b]	-0.3977	-0.9352	-0.8264	-0.5410	-0.9139	-0.7517	-0.0234	0.2408	0.1091	-0.6635	-0.8156	-0.6524
NIQE [Mit+13a]	0.6013	0.9490	0.8283	0.6573	0.8651	0.7089	-0.0037	0.4418	0.3581	-0.4974	-0.2841	-0.2371
OG-IQA [Liu+16]	0.2863	0.4868	0.3278	-0.2013	-0.1947	-0.1759	-0.5076	-0.2150	-0.1673	-0.7798	-0.6177	-0.4556

Tab. 5.8.: Experiment II: Validation set NR-IQA performance for **leave one scene out** cross validation. Each correlation is computed relative to the full reference SSIM of each 512×512 image in the validation set compared to its ground truth image.

Table 5.8 shows the results of our fully convolutional model applied to whole images compared to existing NR-IQA algorithms. Our method consistently performs best across all four validation scenes. As with experiment I, the BRISQUE [Mit+12a], HIGRADE 1 [Kun+16a], HIGRADE 2 [Kun+16a], JP2K-NR [She+05b], NIQE [Mit+13a], OG-IQA [Liu+16] NR-IQA measures all exhibit both positive and negative correlations for different validation scenes which indicates that they are not accurately describing the underlying distribution of visible distortions present in the test images.

5.5 Conclusion and Discussion

In this chapter we have presented the results of our experiments into applying deep learning to NR-IQA. We have shown that convolutional neural networks acting as feature extractors are sufficiently able to detect and extract noise present in Monte Carlo rendered images, and that such features can be used for the purpose of IQA.

When reviewing existing NR-IQA methods we observe that the vast majority of measures in the literature are trained on images from publicly available datasets such as Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], IRCCyN/IVC [AB09], which contain clean reference images (photographs) and their synthetically distorted test images, coupled with subjective quality scores given by and pooled over a set of human observers. We find that in many cases synthetic distortions do not present a representative target for training and evaluating metrics that will be applied to naturally distorted images. In our experiments we show that we can train models directly on images containing naturally occurring distortions from unconverged Monte Carlo rendering processes.

In experiment I we develop a convolutional neural network which accepts a noisy fixed sized 32×32 RGB image patch from a Monte Carlo rendered image, and predicts the quality of the patch to approximate the FR-IQA score that would be computed for the patch if its associated ground truth patch was available. This model initially suffered from stability issues such as the exploding gradient problem. We were able to solve this problem by incorporating the Pearson's Correlation Coefficient into the loss function which acts as a regularization term, allowing the model to train stably. The model also had a tendency to over-fit on the training data by memorizing colour palettes, which caused it to struggle to correctly predict the quality of image patches from scenes which were drastically different to those in the training sets. We solved this problem by augmenting the training data by developing a jitter which operates in the HSV colour space. By randomly shifting the hue of input patches and modulating their contrast and saturation we discourage the model from using colour information as a feature and encourage it to look for structural information which remains invariant under these distortions. Even with a solution to the colour palette issue it is clear from the difference in training stability between cross validation folds, that model stability and generalization is tightly tied to the range and variation of lighting and material compositions visible in scenes from the training set. On leave one out cross validation folds using the Cornell Box scene the model is not able to generalize well due to the difference of images of that scene compared to those available in the training set. The only way to significantly improve on this is to expand the Monte Carlo image dataset to contain additional and more varied scenes to sample from the training sets.

A shortcoming of the model from experiment I is that individual patches of the images must be extracted and processed separately by the network. This is not efficient and causes many duplicated convolutions between image patches which overlap in the source image. For experiment II we develop a fully convolutional model which is able to perform a regression for all pixels in the source image simultaneously. In the fully convolutional architecture an initial set of feature extraction layers using 3×3 convolutions are used to increase the models receptive field up to 11×11 pixels. We chose a receptive field of 11×11 pixels to match the default neighbourhood size used in the SSIM [Wan+04a] FR-IQA metric which was used to compute the ground truth values for the experiment. The feature channels are then put through several layers of 1×1 convolutions which are the equivalent of branching the network independently for every pixel and applying fully connected layers on the branches, where the fully connected layers all have shared weights. This allows for the network to have a deep structure for regression while still only having a 11×11 pixel receptive field.

In both experiments we justify our method by comparing the accuracy on validation sets to values computed using existing state of the art NR-IQA methods, namely: BLIINDS [Saa+10], BIQI [MB10], BRISQUE [Mit+12a], HIGRADE 1 [Kun+16a], HIGRADE 2 [Kun+16a], JP2K-NR [She+05b], NIQE [Mit+13a], and OG-IQA [Liu+16]. In experiment I we compared our results to the MAE FR-IQA method and in experiment II we used SSIM as the FR-IQA method, computed on full images from the validation sets. In both experiments our method was able to accurately capture the distribution of image quality to a much higher degree than existing methods which often had low correlation or inconsistently showed both positive and negative correlation for different validation scenes. This implies that the synthetic distortions used in the publicly available datasets are not representative of the distribution of naturally occurring distortions we observe in Monte Carlo rendered images.

5.5.1 Further Work

As we have previously discussed, NR-IQA measures have a range of applications when applied to Monte Carlo rendered images. NR-IQA can be used to inspect the current quality of images during the rendering process, it can be used as a heuristic for an early stopping criteria to determine when the image has reached a sufficient quality, and it can be used as a heuristic for adaptively directing computational efforts towards regions of the image with lower quality. As further work we want to investigate the use of deep learning models like the one described in experiment II applied to the task of adaptive rendering by integrating the output quality prediction maps into a system for balancing computational resources like those defined in [Mys98; BM98; Ram+99; Mys+00; Deb06] by considering the quality map as showing regions which need proportionally more computation. This is a difficult task as the model must be able to robustly predict the regions of the image with the lowest quality, and be able to do this efficiently so that the computational gains of the adaptive sampling are not outweighed by the expense of determining the regions of low quality. In a potential failure case, if the predicted quality maps are not representative enough of true error distribution such a method could hinder convergence leading to worse performance than uniform sampling.

In section 4.6.1 we discussed our plans to expand the dataset of Monte Carlo rendered images with additional scenes containing a larger range of lighting and material compositions, and to provide this data in HDR to allow more flexibility in future experimental design. As further work we would also look to train models on this expanded dataset and for these new models to make predictions on the raw HDR data that is available within the rendering software. These models could also take as input additional meta data from the rendering software [Kel97; Kal+15] such as pixel colour, screen-space coordinates, world-space coordinates, shading normals, texture albedo at the first and second bounces, and the direct illumination shadow-map.

The code for the two experiments discussed in this chapter are released open source, implemented in Tensorflow [Aba+15] and Keras [Cho+15], and are available on Github under the MIT license at:
`https://github.com/CS-Swansea/MC-NR-IQA`

6 Conclusion

In conclusion we will now summarize the contributions made in each of the three main chapters of this thesis.

Chapter 3

Chapter 3 describes the development of MEL, a modern C++11 framework around the MPI 3.0 standard. MEL is a header-only library with the goal of creating a lightweight and robust framework for building distributed parallel applications for use in HPC environments. MEL is designed to introduce no (or minimal) overheads while *drastically* reducing code complexity and allowing for a greater range of common MPI errors to be caught at compile-time rather than during program execution when it can be far more difficult to debug.

Prior work on providing C++ abstractions to MPI have focused on applying OO design patterns to the way users interact with the MPI runtime [McC+96; Gru+00; BC15b]. For example, an OO wrapper around an `MPI_Comm` object will provide functionality such as a destructor and copy and move constructors. While on the surface this would appear to make code more robust by implicitly instructing the underlying MPI runtime to correctly manage opaque resources, in practice this leads to unnecessary interaction with the MPI runtime which can lead to poor and inconsistent performance when scaling up the number of processes involved in a distributed setting.

For a copy constructor to be properly implemented on an `MPI_Comm` object an implicit call to `MPI_Comm_dup` to duplicate the existing communicator would need to be triggered. Similarly for a destructor to be implemented on a communicator an implicit call to `MPI_Comm_disconnect` would be needed. Such operations are known to not scale well in large systems, especially when network latency is a concern, and are ultimately unnecessary in most use cases. By strictly adhering to certain best practices in OO regimes users can attempt to avoid these kinds of scenarios by passing such OO wrapped objects by reference or by using a form of reference counting scheme. But even with care taken, something as simple as assigning a wrapped communicator object to a temporary variable within a function could silently add an implicit call to a poorly scaling function, making completely valid and pragmatic OO code have confusing and inconsistent runtime characteristics which are difficult to debug.

By fundamentally changing the way in which the user's program interacts with the MPI runtime unnecessary overheads and performance bottlenecks are introduced along with the need to relearn how one is supposed to interact with the MPI runtime, which may hinder adoption of such frameworks. The MPI runtime is designed to sit adjacent to the user's program and to abstract the details of how code interacts with the rest of the distributed system. Invoking OO design principles on this relationship has the effect of injecting implicit and unnecessary micro-management on the MPI runtime.

In our work we opt not apply an OO model and instead leverage the power of the modern C++ compiler to provide much needed type-safety and consistency to the existing imperative model with which MPI was designed to be used.

Another area of MPI we aim to improve upon, which has not been a strict focus of prior modernization frameworks, has been on tightening the consistency of code behaviour and compile-time error handling between MPI distributions. As a motivating example we explore type safety in the MPI standard. With MPI being intended as a C library certain design patterns emerge in the standard and its implementations which have the unintentional side effect of nullifying key parts of the type-system provided by the C and C++ languages.

For example, the standard does not dictate how many of the object types should be implemented, leaving these details to the implementation vendor. In Intel MPI 5.1 `MPI_Comm` objects and many other types are implemented as integer handles, `typedef int MPI_Comm`, to opaque data that are managed by the MPI run-time. This causes compile-time type-checking of function parameters to not flag erroneous combinations of variables. The common signature `MPI_Send(void*, int, MPI_Datatype, int, int, MPI_Comm)` is actually seen by the compiler as `MPI_Send(void*, int, int, int, int, int)`, allowing any ordering of the last five variables to be compiled as valid MPI code, potentially causing catastrophic failure at run-time. In contrast, Open MPI 1.10.2 implements these types as structs which are inherently type-safe.

In MEL we impose the necessary type-safety on the opaque data handles returned by the MPI runtime regardless of the individual MPI vendor's implementation. These abstractions largely take the form of simple structs with the underlying MPI type as a member variable, and an explicit cast operator for removing the abstraction when the time is right. Such abstractions are designed to be soluble, existing only as far as the intermediate representation during compilation, allowing the compiler to remove the abstractions MEL provides to achieve the same performance as native

MPI code while imposing the type-safety that was originally lacking. MEL makes use of modern C++ language features such as `const` types, pass by reference, and advanced template meta-programming to both ensure correctness at compile-time and to generate boiler-plate values that programmers have to provide themselves with native MPI code.

Lastly, one of the main goals of MEL is to give higher-level functionality that is not natively available within the MPI standard. To demonstrate this we tackle the issue of performing deep copy on complex hierarchical and potentially cyclical data structures between disjoint hosts in a distributed computing environment.

For user defined objects MPI natively adopts shallow copy semantics, where memory allocation, copy, and de-allocation are the responsibility of the programmer. Shallow copy is acceptable for shared-memory programming models where it is always legal to dereference a pointer with the underlying assumption that the target of member pointers will be shared among all processes. In environments which do not employ a shared-memory model such as communication between processes on a single machine or between processes on disjoint hosts connected by a network interface deep copy is needed to recursively traverse pointer members in a data structure, transferring all disjoint memory locations, and translating the pointers to refer to the appropriate device location.

The task of writing code to perform such deep copy transfers is a complicated process which must be implemented for each data structure that needs to be transferred, and for each method by which it will be transferred. This is an error prone task that can be difficult to debug as errors can easily exist on both sending and receiving processes, and often present themselves as undefined behaviour stemming from memory overruns or from failing to correctly traverse the entire structure. Such errors often do not cause the program to outright terminate; rather, they leave the still executing program in an undefined and potentially unstable state which can further confuse debugging efforts.

In MEL we approach this problem by creating a set of generic deep copy semantics that can easily be applied to user data structures, and a traversal and transportation algorithm which can then walk the data structure performing the desired transport operations. Our semantic markup is generic both in that it can be applied simply to any user defined structure, but also in its abstraction of the transport operation that will later be performed on it. Once a data structure has been prepared for our deep copy algorithm it can be passed to all transport functions within the API, allowing for MPI send, receive, broadcast, and file IO operations to all be performed. Additionally we provide a variant of each of the above transport operations which internally buffer data into or from a contiguous block of memory during transmission, yielding considerable performance increases on appropriate data structures.

To justify our approach we provide code examples and run-time performance measurements computed using the HPC Wales compute cluster.

In our first experiment we show that in just four simple lines of code (section 3.3.5.1) we can apply our deep copy semantics to a BVH-Tree data structure in the context of a distributed ray tracing program. These four lines are able to convey all the information needed by our algorithm to replace 34 and 70 lines of hand written code needed to perform the same deep copy operation for non-buffered and buffered MPI broadcast operations, respectively. Further we show that we can accomplish this with no loss in performance. We also show that our method scales with the number of processes involved in the communication at the same rate as the underlying broadcast operation (logarithmically). In our second experiment we show the performance of our method in reading and writing the BVH-Tree structure to disk compared to Boost Serialization Library [Cog05], showing that our method scales significantly better than Boost as the size of the structure increases. Finally, we show the results of non-buffered versus buffered MEL deep copy broadcast operations, and of MEL deep file IO operations versus Boost Serialization Library on arbitrary generic graph structures of various connectivities (fully connected, randomly connected, rings, binary-tree) and sizes, showing again that our method naturally scales well without any tuning or modification.

MEL is Open-Source and available on Github under the MIT license at:
`https://github.com/CS-Swansea/MEL`

Chapter 4

In chapter 4 we provide an ensemble study on the robustness of IQA measures when evaluating images created through Monte Carlo rendering processes. When assessing image quality in Monte Carlo rendered images the use of a reference image or GT is a common method to provide a baseline with which to compare experimental results as we often need to determine the relative quality between images computed using different algorithms and with varying amounts of computation.

We show that if not chosen carefully the quality of reference images used for IQA can skew results leading to significant misreporting of error. We present an analysis of error in Monte Carlo rendered images and discuss practices to avoid or be aware of when designing an experiment.

The issue stems from the fact that ground truth images are never truly available, as they are the product of a stochastic rendering process just the like the test images we wish to evaluate. Monte Carlo rendering processes are known to converge in the limit, as the number of samples goes to infinity. For any finite number of samples used to approximate an image there will always be some distortion or bias introduced into the image estimate. By rendering reference images to a significantly higher visual quality than the test images they will be used to evaluate we can limit the bias introduced by distortions in reference images.

However, this raises the question of by how much distortions in reference images affect the results of IQA measures when evaluating test images. To answer this question we constructed an experiment where we rendered images of various scene, material, and lighting compositions to increasing numbers of image samples using several Monte Carlo rendering algorithms; we then used this dataset of images of varying quality to calculate the reported error of test images compared to each of the potentially noisy reference images drawn from the dataset, using a collection of IQA measures we gathered from the literature. We used the highest quality images as approximate ground truth images, and use these "true" values to compute the amount each error metric had under- or overestimated its quality score for each comparison, as a function of the quality of the potentially noisy reference image that was used.

By this method we can show the robustness of each IQA measure we sample from the literature when we consider the scenario where the reference image used in image quality assessment is not a perfect representation of the ground truth image and contains some magnitude of distortion.

The results of our study show that some IQA measures are significantly more susceptible to under- or overestimating the amount of error in test images as a result of distortions in reference images.

From our results, we recommend that MS-SSIM [Wan+03] or SC-QI [BK16] be used for image quality assessments when evaluating images produced by Monte Carlo rendering algorithms as these methods were the most robust when we consider noise in reference images. Reference images should ideally be rendered with uniform sampling methods such as Path Tracing (PT) or BDPT to avoid the introduction of structural artefacts in IQA. We show that it is crucial that the reference used is not only visually noise free, but also that it is of sufficiently higher numerical quality than images tested against it. Reference images should therefore be rendered to at least an order of magnitude higher sample count than test images to minimize the possibility of noise in the reference causing a significant deviation in reported error. Finally, the sample count and method of production of the reference image should be stated clearly to give researchers attempting replication every confidence in reported results.

Chapter 5

Finally, in chapter 5 we investigate the use of deep learning models applied to the task of NR-IQA in the context of evaluating images rendered with Monte Carlo rendering processes.

In FR-IQA, images are compared with ground truth images that are known to be of high visual quality. These metrics are utilized in order to rank algorithms under test on their image quality performance. However, during an intermediate stage of a Monte Carlo rendering process we do not have access to ground truth images with which to compare our current image estimate, as this would imply the availability of the final image. To evaluate our current image estimate we need to utilise NR-IQA methods which compare the image under evaluation to the distribution of natural images we are likely trying to compute.

When reviewing existing NR-IQA methods we observe that the vast majority of measures in the literature are trained on images from publicly available datasets such as Live [She+14], TID2008 [Pon+09], TID2013 [Pon+13], Kodak Lossless True Colour [Fra99], MICT [Hor+11], and IRCCyN/IVC [AB09], which contain clean reference images (photographs) and their synthetically distorted test images, coupled with subjective quality scores given by and pooled over a set of human observers. We find that in many cases synthetic distortions do not present a representative target for training and evaluating metrics that will be applied to naturally distorted images. In our experiments we show that we can train models directly on images containing naturally occurring distortions from un-converged Monte Carlo rendering processes.

In our work, we propose a deep learning approach to NR-IQA trained specifically on noise from Monte Carlo rendering processes, which significantly outperforms existing NR-IQA methods, and produces performance close to the approximated FR-IQA measure.

In our first experiment we develop a convolutional neural network which accepts a noisy fixed sized 32×32 RGB image patch from a Monte Carlo rendered image, and predicts the quality of the patch to approximate the FR-IQA score that would be computed for the patch if its associated ground truth patch was available. We incorporate the Pearson's Correlation Coefficient which is commonly used to evaluate the performance of IQA methods into the loss function. This acts as a regularization term, allowing the model to train stably and consistently. To reduce over-fitting on the training data caused by memorization of colour palettes, we augment the training data by developing a jitter which operates in the HSV colour space. By randomly shifting the hue of input patches and modulating their contrast and saturation we discourage the model from using colour information as a feature and encourage it to look for structural information which remains invariant under these distortions.

A shortcoming of this approach is that individual patches of the images must be extracted and processed separately by the network, causing many duplicated convolutions to be performed between image patches which overlap in the source image. For our second experiment we move to a fully convolutional model which is able to perform a regression for all pixels in the source image simultaneously. In the fully convolutional architecture an initial set of feature extraction layers using 3×3 convolutions is used to increase the model's receptive field up to 11×11 pixels. We chose a receptive field of 11×11 pixels to match the default neighbourhood size used in the SSIM [Wan+04a] FR-IQA metric which was used to compute the ground truth values for the experiment. The feature channels are then put through several layers of 1×1 convolutions which are the equivalent of branching the network independently for every pixel and applying fully connected layers on the branches, where the fully connected layers all have shared weights. This allows for the network to have a deep structure for regression while still only having a 11×11 pixel receptive field.

We justify our method by comparing the accuracy on validation sets of Monte Carlo rendered images to values computed using existing state of the art NR-IQA methods, namely: BLIINDS [Saa+10], BIQI [MB10], BRISQUE [Mit+12a], HIGRADE 1 [Kun+16a], HIGRADE 2 [Kun+16a], JP2K-NR [She+05b], NIQE [Mit+13a], and OG-IQA [Liu+16]. In experiment I we compared our results to the MAE FR-IQA method and in experiment II we used SSIM as the FR-IQA method, computed on full images from the validation sets. In both experiments our method was able to accurately capture the distribution of image quality to a much higher degree than existing methods which often had low correlation or inconsistently showed both positive and negative correlation for different validation scenes. This implies that the synthetic distortions used in the publicly available datasets are not representative of the distribution of naturally occurring distortions we observe in Monte Carlo rendered images.

The code for the two experiments discussed in this chapter are released open source, implemented in Tensorflow [Aba+15] and Keras [Cho+15], and are available on Github under the MIT license at:
`https://github.com/CS-Swansea/MC-NR-IQA`

6.1 Summary of Contributions

In summary, with this thesis we address the problem of image quality assessment in the domain of images generated with physically based rendering algorithms. To this end we have provided contributions in the areas of high performance computing, image quality assessment, and machine learning.

We have presented MEL, a modern C++ framework built on top of MPI designed with goals of increasing type-safety and compile-time error handling while adding higher-level functionality such as deep-copy semantics. Using template meta programming to automatically deduce boilerplate parameters MEL decreases the potential for programmer error and significantly reduces code complexity compared to native MPI. We make this framework open source for anyone to use.

With MEL we implemented Monte Carlo rendering algorithms on HPC platforms and used this distributed rendering software to compute a dataset of images containing varying material, lighting, and scene compositions. This dataset was designed for the purpose of training and evaluating IQA measures directly on the domain specific distortions that occur naturally as a result of under-sampling in stochastic rendering processes.

Using the Monte Carlo image dataset we provided an ensemble study on the robustness of modern and classical IQA measures when we consider the scenario where the reference image used in FR-IQA or RR-IQA is not a perfect representation of the ground truth image, and contains some magnitude of distortion. Out of the IQA measures we evaluated, we found that the most robust were MS-SSIM [Wan+03] and SC-QI [BK16] when evaluating images produced by Monte Carlo rendering algorithms. We recommend that when evaluating such images, reference images should ideally be rendered with uniform sampling methods such as PT or BDPT to avoid the introduction of structural artefacts in IQA. We show that it is crucial that the reference image used is not only visually noise free, but also that it is of sufficiently higher numerical quality than images tested against it. We therefore recommend that reference images should be rendered to at least an order of magnitude higher sample count than images being evaluated to minimize the possibility of noise in the reference causing a significant deviation in reported error.

Finally, we investigated the NR-IQA problem with a specific focus on evaluating images produced by Monte Carlo rendering algorithms. As a result of this work we presented two CNN architectures, one which computes a scalar valued quality score when presented with a fixed sized 32×32 image patch, and a second more advanced model which densely predicts a quality score for every pixel in an arbitrary sized input image in parallel by using an FCNN architecture, yielding significant speed improvements for inferring image quality. These models are trained end-to-end to regress the quality scores given by FR-IQA measures calculated at each pixel between un-converged images and their clean ground truth images.

In order to improve model accuracy and training stability we develop HSV jitter as an input regularization technique to discourage models from memorizing colour palette information as part of their NR-IQA strategy. We also develop a joint loss function which combines the Charbonnier loss and the PCC computed batch-wise during training. Jointly minimizing the difference between the regression target and prediction while maximizing the correlation between the prediction and target value has the effect of greatly improving final accuracy and in several configurations avoid the exploding gradient problem when initiating training from random weights. From the raw un-converged image data, our models learn a distribution of what natural images from the distortion domain look like. Our proposed models are able, to within a high degree of accuracy, predict pixel level quality scores when only shown un-converged test images. We make the code for these models, along with their training and evaluation scripts, open source for anyone to use.

6.2 Further Work

It is important to reflect on the limitations of the current work that has been proposed, and to identify where improved or alternative strategies can be applied to address these problems going forward.

Our plans for further research from this work are to continue the development and expansion of MEL to further improve the libraries capacity for compile-time error handling and boilerplate code deduction. With this expansion we can address a limitation of MEL deep-copy which currently only supports a blocking communication function interface. We plan to investigate the implementation of a non-blocking interface for performing deep-copy by using hybrid MPI where deep-copy is spawned onto an auxiliary thread within the same address spaces as the initiating MPI processes. Synchronization of the non-blocking deep-copy can be implemented in this scenario by using a combination of thread and MPI process level synchronization functions to implement proper asynchronous semantics for wait, test, and barrier style functions.

We plan to address the limitations of the current Monte Carlo image dataset by adding additional scenes, including scenes which aim to model photo-realistic image compositions, and those which aim to model more synthetic based images of single material models under controlled lighting and viewing conditions. The expanded dataset will be computed in high-dynamic range so that both HDR and LDR quality measures can be trained and evaluated. With a larger dataset and one containing a wider range of material, lighting, and scene compositions models trained on the dataset will be able to better generalize and achieve more robust inference results.

In order to train and evaluate images using the Monte Carlo dataset experiments must currently be designed to predict the values reported by FR-IQA algorithms when they are shown both the un-converged image and the clean ground truth image. To address this, we plan to conduct a user study to collect MOS and DMOS for images in the expanded dataset. New FR-IQA, RR-IQA, and NR-IQA models will be able to be fit and evaluated against the perceptually aware quality scores provided by human observers. These scores represent a more meaningful target for IQA regression and have a broader range of applications in real-world inference.

Finally, we plan to continue investigating the use of machine learning models in IQA measures with specific application to the assessment of images rendered with Monte Carlo rendering algorithms. New models should be trained on the expanded Monte Carlo image dataset and should operate on raw HDR intensities and scene data that are available directly within the rendering software. They should also be trained to predict the MOS or DMOS given by human observers.

Bibliography

[AB09] Florent Autrusseau and Marie Babel. *Subjective quality assessment of LAR coded art images*. http://www.irccyn.ec-nantes.fr/ autrusse/Databases/. 2009 (cit. on pp. 6, 117, 132, 138, 162, 172).

[Aba+15] Martín Abadi, Ashish Agarwal, Paul Barham, et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015 (cit. on pp. 66, 165, 173).

[AC86] James Arvo and MA Chelmsford. "Backward ray tracing". In: *Developments in Ray Tracing, Computer Graphics, Proc. of ACM SIGGRAPH 86 Course Notes*. 1986, pp. 259–263 (cit. on p. 40).

[AES93] Arthur W. Springsteen Albert E. Stiegman Carol J. Bruegge. "Ultraviolet stability and contamination analysis of Spectralon diffuse reflectance material". In: *Optical Engineering* 32 (1993), pp. 32 –32 –6 (cit. on p. 32).

[And+96] Matthew Anderson, Ricardo Motta, Srinivasan Chandrasekar, and Michael Stokes. "Proposal for a Standard Default Color Space for the InternetsRGB". In: *Color and Imaging Conference* 1996.1 (1996), pp. 238–245 (cit. on p. 117).

[Avc+02] Ismail Avcibas, Bulent Sankur, and Khalid Sayood. "Statistical evaluation of image quality measures". In: *Journal of Electronic Imaging* 11.2 (2002), pp. 206–223 (cit. on p. 114).

[BA83] Peter Burt and Edward Adelson. "The Laplacian pyramid as a compact image code". In: *IEEE Transactions on communications* 31.4 (1983), pp. 532–540 (cit. on p. 115).

[Bab+07] R Venkatesh Babu, Sundaram Suresh, and Andrew Perkis. "No-reference JPEG-image quality assessment using GAP-RBF". In: *Signal Processing* 87.6 (2007), pp. 1493–1503 (cit. on p. 137).

[Bar17] Jonathan T. Barron. "A More General Robust Loss Function". In: *CoRR* abs/1701.03077 (2017) (cit. on pp. 145, 146).

[Bat05] Christopher Batty. "Implementing energy redistribution path tracing". In: *Department of Computer Science, The University of British Columbia* (2005) (cit. on pp. 52, 57, 58, 71).

[Bau+15] Pablo Bauszat, Martin Eisemann, Elmar Eisemann, and Marcus Magnor. "General and Robust Error Estimation and Reconstruction for Monte Carlo Rendering". In: *Computer Graphics Forum (Proc. of Eurographics EG)* 34.2 (2015), pp. 597–608 (cit. on p. 114).

[BC15b] Boost-Community. *BOOST C++ Libraries*. 2015 (cit. on pp. 77, 167).

[Bey+14] James Beyer, David Oehmke, and Jeff Sandoval. “Transferring user defined types in OpenACC”. In: *Proc. Cray User Group (CUG’14)* (2014) (cit. on p. 78).

[BK16] S. H. Bae and M. Kim. “A Novel Image Quality Assessment With Globally and Locally Consilient Visual Quality Perception”. In: *IEEE Transactions on Image Processing* 25.5 (2016), pp. 2392–2406 (cit. on pp. 69, 116, 119, 136, 138, 171, 174).

[BM98] Mark R Bolin and Gary W Meyer. “A perceptually based adaptive sampling algorithm”. In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM. 1998, pp. 299–309 (cit. on pp. 56, 58, 164).

[Bos+16a] S. Bosse, D. Maniry, T. Wiegand, and W. Samek. “A deep neural network for image quality assessment”. In: *2016 IEEE International Conference on Image Processing (ICIP)*. 2016, pp. 3773–3777 (cit. on pp. 142, 143).

[Bos+16b] Sebastian Bosse, Dominique Maniry, Klaus-Robert Múller, Thomas Wiegand, and Wojciech Samek. “Deep Neural Networks for No-Reference and Full-Reference Image Quality Assessment”. In: *CoRR* abs/1612.01697 (2016) (cit. on pp. 142, 143).

[Bot10] Léon Bottou. “Large-scale machine learning with stochastic gradient descent”. In: *Proceedings of COMPSTAT’2010*. Springer, 2010, pp. 177–186 (cit. on pp. 66, 68).

[Bou15] Aurélien Bouteiller. “Fault-Tolerant MPI”. In: *Fault-Tolerance Techniques for High-Performance Computing*. Ed. by Thomas Herault and Yves Robert. Springer Publishing Company, Incorporated, 2015. Chap. 3, pp. 145–228 (cit. on p. 79).

[Bou+10] Y-Lan Boureau, Jean Ponce, and Yann Lecun. “A Theoretical Analysis of Feature Pooling in Visual Recognition”. In: *icml*. 2010 (cit. on pp. 142, 143).

[Bov09] Alan C Bovik. “The essential guide to image processing”. In: *Academic Press* (2009), p. 583 (cit. on p. 116).

[BQ08] Tomás Brandão and Maria Paula Queluz. “No-reference image quality assessment based on DCT domain statistics”. In: *Signal Processing* 88.4 (2008), pp. 822–833 (cit. on p. 137).

[Bru+12] D. Brunet, E. R. Vrscay, and Z. Wang. “On the Mathematical Properties of the Structural Similarity Index”. In: *IEEE Transactions on Image Processing* 21.4 (2012), pp. 1488–1499 (cit. on p. 116).

[Cas+04] George Casella, Christian P. Robert, and Martin T. Wells. “Generalized Accept-Reject sampling schemes”. In: *A Festschrift for Herman Rubin*. Ed. by Anirban DasGupta. Vol. Volume 45. Lecture Notes–Monograph Series. Beachwood, Ohio, USA: Institute of Mathematical Statistics, 2004, pp. 342–347 (cit. on p. 15).

[CD14] T. Chai and R. R. Draxler. “Root mean square error (RMSE) or mean absolute error (MAE)?” In: *Geoscientific Model Development Discussions* 7 (2014), pp. 1525–1534 (cit. on p. 115).

[CH07] D. M. Chandler and S. S. Hemami. “VSNR: A Wavelet-Based Visual Signal-to-Noise Ratio for Natural Images”. In: *IEEE Transactions on Image Processing* 16.9 (2007), pp. 2284–2298 (cit. on pp. 69, 116, 119).

[Cho+15] François Chollet et al. *Keras*. `https://github.com/fchollet/keras`. 2015 (cit. on pp. 165, 173).

[Chu92] Charles K. Chui. *An Introduction to Wavelets*. San Diego, CA, USA: Academic Press Professional, Inc., 1992 (cit. on p. 115).

[Cia+11] Alexandre Ciancio, André Luiz N Targino da Costa, Eduardo AB da Silva, et al. “No-reference blur assessment of digital pictures based on multifeature classifiers”. In: *IEEE Transactions on image processing* 20.1 (2011), pp. 64–75 (cit. on p. 137).

[CK11] Ming Chuang and Michael Kazhdan. “Interactive and anisotropic geometry processing using the screened Poisson equation”. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 57 (cit. on p. 54).

[Cli+05] David Cline, Justin Talbot, and Parris Egbert. “Energy Redistribution Path Tracing”. In: *ACM Trans. Graph.* 24.3 (2005), pp. 1186–1195 (cit. on pp. 52, 57, 58, 71, 114).

[Cog05] Jeff Cogswell. “Adding an Easy File Save and File Load Mechanism to Your C++ Program”. In: *InformIT* (2005) (cit. on pp. 78, 105, 170).

[Coo+84] Robert L. Cook, Thomas Porter, and Loren Carpenter. “Distributed Ray Tracing”. In: *SIGGRAPH Comput. Graph.* 18.3 (Jan. 1984), pp. 137–145 (cit. on p. 30).

[CY10] Erez Cohen and Yitzhak Yitzhaky. “No-reference assessment of blur and noise impacts on image quality”. In: *Signal, image and video processing* 4.3 (2010), pp. 289–302 (cit. on p. 137).

[Dal93] Scott Daly. “Digital Images and Human Vision”. In: ed. by Andrew B. Watson. Cambridge, MA, USA: MIT Press, 1993. Chap. The Visible Differences Predictor: An Algorithm for the Assessment of Image Fidelity, pp. 179–206 (cit. on p. 56).

[Deb06] Kurt Debattista. “Selective rendering for high-fidelity graphics”. PhD thesis. University of Bristol, 2006 (cit. on pp. 56, 58, 164).

[DJ13] Ian C. Doidge and Mark W. Jones. “Probabilistic illumination-aware filtering for Monte Carlo rendering”. English. In: *The Visual Computer* (June 2, 2013), pp. 1–10 (cit. on p. 114).

[DK03] Dariusz Dereniowski and Marek Kubale. “Cholesky factorization of matrices in parallel and ranking of graphs”. In: *International Conference on Parallel Processing and Applied Mathematics*. Springer. 2003, pp. 985–992 (cit. on p. 14).

[Doi+12] Ian Doidge, Mark W. Jones, and Benjamin Mora. “Mixing Monte Carlo and Progressive Rendering for Improved Global Illumination”. In: *The Visual Computer* 28.6–8 (June 12, 2012), pp. 603–612 (cit. on p. 114).

[Duc+11] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization”. In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159 (cit. on p. 66).

[Du+05] Juan Du, Yinglin Yu, and Shengli Xie. “A new image quality assessment based on HVS”. In: *Journal of Electronics (China)* 22.3 (2005), pp. 315–320 (cit. on p. 137).

[DV05a] M. N. Do and M. Vetterli. “The contourlet transform: an efficient directional multiresolution image representation”. In: *IEEE Transactions on Image Processing* 14.12 (2005), pp. 2091–2106 (cit. on p. 115).

[DV+00] Niranjan Damera-Venkata, Thomas D Kite, Wilson S Geisler, Brian L Evans, and Alan C Bovik. “Image quality assessment based on a degradation model”. In: *IEEE transactions on image processing* 9.4 (2000), pp. 636–650 (cit. on pp. 69, 116, 119).

[ER04] Ramin Eslami and Hayder Radha. “Wavelet-based contourlet coding using an SPIHT-like algorithm”. In: *Conference on Information Sciences and Systems (CISS)*. Department of Electrical Engineering, Princeton University, 2004, pp. 784–788 (cit. on pp. 70, 117, 119).

[Fag+01] Graham E. Fagg, Antonin Bukovsky, and Jack J. Dongarra. “HARNESS and Fault Tolerant MPI”. In: *Parallel Comput.* 27.11 (2001), pp. 1479–1495 (cit. on p. 80).

[FD04] Graham E. Fagg and Jack J. Dongarra. “Building and Using a Fault-Tolerant MPI Implementation”. In: *Int. J. High Perform. Comput. Appl.* 18.3 (2004), pp. 353–361 (cit. on p. 80).

[Fer+97] James A Ferwerda, Peter Shirley, Sumanta N Pattanaik, and Donald P Greenberg. “A model of visual masking for computer graphics”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 143–152 (cit. on p. 139).

[FK09] Rony Ferzli and Lina J Karam. “A no-reference objective image sharpness metric based on the notion of just noticeable blur (JNB)”. In: *IEEE transactions on image processing* 18.4 (2009), pp. 717–728 (cit. on p. 137).

[Fra99] Rich Franzen. “Kodak lossless true color image suite”. In: *source: http://r0k.us/graphics/kodak* 4 (1999) (cit. on pp. 6, 117, 132, 138, 162, 172).

[Fri+13] A. Friedley, T. Hoefler, G. Bronevetsky, and A. Lumsdaine. “Ownership Passing: Efficient Distributed Memory Programming on Multi-core Systems”. In: *Proceedings of the 18th ACM SIGPLAN symposium on Principles and practice of parallel programming*. Shenzen, China: ACM, 2013, pp. 177–186 (cit. on pp. 77, 79).

[Fu+17] X. Fu, J. Huang, X. Ding, Y. Liao, and J. Paisley. “Clearing the Skies: A Deep Network Architecture for Single-Image Rain Removal”. In: *IEEE Transactions on Image Processing* 26.6 (2017), pp. 2944–2956 (cit. on p. 140).

[Gao+09] Xinbo Gao, Wen Lu, Dacheng Tao, and Xuelong Li. “Image Quality Assessment Based on Multiscale Geometric Analysis”. In: *Trans. Img. Proc.* 18.7 (2009), pp. 1409–1423 (cit. on p. 115).

[GL04] William Gropp and Ewing Lusk. “Fault Tolerance in Message Passing Interface Programs”. In: *International Journal of High Performance Computing Applications* 18.3 (2004), pp. 363–372. eprint: `http://hpc.sagepub.com/content/18/3/363.full.pdf+html` (cit. on p. 79).

[GNA08] GNA. *Autoserial Library*. `http://home.gna.org/autoserial/mpi.html`. 2008 (cit. on p. 78).

[Gor+84] 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*. Vol. 18. 3. ACM. 1984, pp. 213–222 (cit. on p. 31).

[Gou+98] Delphine Stéphanie Goujon, Martial Michel, Jasper Peeters, and Judith Ellen Devaney. “AutoMap and AutoLink tools for communicating complex and dynamic data-structures using MPI”. In: *Network-Based Parallel Computing Communication, Architecture, and Applications: CANPC ’98*. Ed. by Dhabaleswar K. Panda and Craig B. Stunkel. Springer Berlin Heidelberg, 1998, pp. 98–109 (cit. on p. 77).

[Gru+00] T. Grundmann, M. Ritt, and W. Rosenstiel. “TPO++: an object-oriented message-passing library in C++”. In: *Parallel Processing, 2000. Proceedings. 2000 International Conference on*. 2000, pp. 43–50 (cit. on pp. 78, 167).

[Hab10] John Hable. “Everything has Fresnel. Filmic Games”. In: (2010) (cit. on p. 32).

[Hac+08a] Toshiya Hachisuka, Wojciech Jarosz, Richard Peter Weistroffer, et al. “Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing”. In: *ACM Trans. Graph.* 27.3 (2008), 33:1–33:10 (cit. on p. 114).

[Hac+08b] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. “Progressive Photon Mapping”. In: *ACM Trans. Graph.* 27.5 (2008), 130:1–130:8 (cit. on p. 114).

[Hac+14] Toshiya Hachisuka, Anton S. Kaplanyan, and Carsten Dachsbacher. “Multiplexed Metropolis Light Transport”. In: *ACM Trans. Graph.* 33.4 (2014), 100:1–100:10 (cit. on pp. 50, 71, 114).

[Har+17] Carlo Harvey, Kurt Debattista, Thomas Bashford-Rogers, and Alan Chalmers. “Multi-Modal Perception for Selective Rendering”. In: *Computer Graphics Forum* 36.1 (2017), pp. 172–183 (cit. on pp. 57, 58).

[Has70] W Keith Hastings. “Monte Carlo sampling methods using Markov chains and their applications”. In: *Biometrika* 57.1 (1970), pp. 97–109 (cit. on p. 17).

[Her+12a] Robert Herzog, Martin Čadík, Tunç O Ayd̆cin, et al. “NoRM: No-Reference Image Quality Metric for Realistic Image Synthesis”. In: *Computer Graphics Forum*. Vol. 31. 2pt3. Wiley Online Library. 2012, pp. 545–554 (cit. on p. 139).

[He+15a] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). arXiv: `1512.03385` (cit. on pp. 67, 68).

[He+15b] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”. In: *CoRR* abs/1502.01852 (2015) (cit. on pp. 63, 143).

[Hor+11] Yuukou Horita, Keiji Shibata, Yoshikazu Kawayoke, and ZM Parvez Sazzad. “MICT image quality evaluation database”. In: *[Online], http://mict.eng.u-toyama.ac.jp/mictdb.html* (2011) (cit. on pp. 6, 117, 132, 138, 162, 172).

[HR15] Thomas Herault and Yves Robert. *Fault-Tolerance Techniques for High-Performance Computing*. 1st. Springer, 2015 (cit. on p. 79).

[HS11] T. Hoefler and M. Snir. “Writing Parallel Libraries with MPI - Common Practice”. In: *Proceedings of the 18th MPI Users’ GroupMeeting*. Vol. 6960. 2011, pp. 345–355 (cit. on p. 77).

[Hua+06] Chao Huang, Gengbin Zheng, Laxmikant Kalé, and Sameer Kumar. “Performance Evaluation of Adaptive MPI”. In: *Proceedings of the Eleventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP ’06. New York, New York, USA: ACM, 2006, pp. 12–21 (cit. on p. 77).

[Hua+14] Xiaotong Huang, Li Chen, Jing Tian, Xiaolong Zhang, and Xiaowei Fu. “Blind noisy image quality assessment using block homogeneity”. In: *Computers and Electrical Engineering* 40.3 (2014), pp. 796–807 (cit. on p. 137).

[Hua+16] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. “Densely Connected Convolutional Networks”. In: *CoRR* abs/1608.06993 (2016). arXiv: `1608.06993` (cit. on pp. 67, 68).

[IS15] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *International conference on machine learning*. 2015, pp. 448–456 (cit. on pp. 140, 143).

[Jak10] Wenzel Jakob. *Mitsuba renderer*. http://www.mitsuba-renderer.org. 2010 (cit. on p. 120).

[JC95] Henrik Wann Jensen and Niels Jørgen Christensen. “Optimizing Path Tracing using Noise Reduction Filters”. In: *Proceedings of WSCG95*. 1995, pp. 134–142 (cit. on p. 114).

[Jen01] Henrik Wann Jensen. *Realistic Image Synthesis Using Photon Mapping*. Natick, MA, USA: A. K. Peters, Ltd., 2001 (cit. on p. 114).

[Jia+13] Shuhong Jiao, Huan Qi, Weisi Lin, and Weihe Shen. “Fast and efficient blind image quality index in spatial domain”. In: *Electronics Letters* 49.18 (2013), pp. 1137–1138 (cit. on p. 137).

[JM12] Wenzel Jakob and Steve Marschner. “Manifold exploration: a Markov chain Monte Carlo technique for rendering scenes with difficult specular transport”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 58 (cit. on pp. 50, 54, 58, 114).

[Kaj86] James T. Kajiya. “The Rendering Equation”. In: *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’86. New York, NY, USA: ACM, 1986, pp. 143–150 (cit. on pp. vii, 30, 36, 57, 59, 71, 114, 138).

[Kal+15] Nima Khademi Kalantari, Steve Bako, and Pradeep Sen. “A Machine Learning Approach for Filtering Monte Carlo Noise”. In: *ACM Transactions on Graphics (TOG) (Proceedings of SIGGRAPH 2015)* 34.4 (2015) (cit. on pp. 114, 140, 164).

[KB14] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014) (cit. on pp. 66, 140, 142, 144, 156, 157).

[KB15] Vipin Kamble and K.M. Bhurchandi. “No-reference image quality assessment algorithms: A survey”. In: *Optik - International Journal for Light and Electron Optics* 126.11–12 (2015), pp. 1090 –1097 (cit. on p. 137).

[Kel97] Alexander Keller. “Instant radiosity”. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 49–56 (cit. on pp. 139, 164).

[Kel+02] Csaba Kelemen, László Szirmay-Kalos, György Antal, and Ferenc Csonka. “A simple and robust mutation strategy for the metropolis light transport algorithm”. In: *Computer Graphics Forum* 21.3 (2002), pp. 531–540 (cit. on pp. 50, 57, 58, 71, 114).

[Ken16] Tom Kennedy. *Monte Carlo Methods - a special topics course - Chapter 6 - Importance sampling*. 2016. URL: `http://math.arizona.edu/~tgk/mc/book_chap6.pdf` (cit. on pp. 23, 24).

[Ket+15] Markus Kettunen, Marco Manzi, Miika Aittala, et al. “Gradient-Domain Path Tracing”. In: *ACM Trans. Graph.* 34.4 (2015) (cit. on pp. 55, 58).

[KK93] Laxmikant V. Kale and Sanjeev Krishnan. “CHARM++: A Portable Concurrent Object Oriented System Based on C++”. In: *Proceedings of the Eighth Annual Conference on Object-oriented Programming Systems, Languages, and Applications*. OOPSLA ’93. Washington, D.C., USA: ACM, 1993, pp. 91–108 (cit. on p. 78).

[Kon+04] Janne Kontkanen, Jussi Rosonen, and Alexander Keller. “Irradiance Filtering for Monte Carlo Ray Tracing”. In: *Monte Carlo and Quasi-Monte Carlo Methods 2004*. Springer, 2004, pp. 259–272 (cit. on p. 114).

[Kri+12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105 (cit. on pp. 67, 68, 154).

[KS00] Sing Bing Kang and Heung-Yeung Shum. “A Review of Image-based Rendering Techniques”. In: Institute of Electrical and Electronics Engineers, Inc., 2000 (cit. on p. 56).

[KS13] Nima Khademi Kalantari and Pradeep Sen. “Removing the Noise in Monte Carlo Rendering with General Image Denoising Algorithms”. In: *Computer Graphics Forum* 32.2pt1 (2013), pp. 93–102 (cit. on p. 114).

[Kun+16a] D. Kundu, D. Ghadiyaram, A. C. Bovik, and B. L. Evans. “No-reference image quality assessment for high dynamic range images”. In: *2016 50th Asilomar Conference on Signals, Systems and Computers*. 2016, pp. 1847–1852 (cit. on pp. 70, 137, 153, 162, 164, 173).

[Kun+16b] Debarati Kundu, Deepti Ghadiyaram, Alan C Bovik, and Brian L Evans. “No-reference Image Quality Assessment for High Dynamic Range Images”. In: *Proc. Asilomar Conf. on Signals, Systems, and Computers*. 2016 (cit. on p. 137).

[Lag+14] Ignacio Laguna, David F. Richards, Todd Gamblin, Martin Schulz, and Bronis R. de Supinski. “Evaluating User-Level Fault Tolerance for MPI Applications”. In: *Proceedings of the 21st European MPI Users’ Group Meeting*. EuroMPI/ASIA ’14. Kyoto, Japan: ACM, 2014, 57:57–57:62 (cit. on p. 79).

[Lam92] Johann Heinrich Lambert. *Photometrie: Photometria, sive De mensura et gradibus luminis, colorum et umbrae (1760)*. 31-33. W. Engelmann, 1892 (cit. on p. 32).

[LeC+98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 66).

[Lee06] Edward A. Lee. “The Problem with Threads”. In: *Computer* 39.5 (2006), pp. 33–42 (cit. on p. 77).

[Leh+13] Jaakko Lehtinen, Tero Karras, Samuli Laine, et al. “Gradient-Domain Metropolis Light Transport”. In: *ACM Trans. Graph.* 32.4 (2013) (cit. on pp. 54, 58).

[Lia+10] Luhong Liang, Shiqi Wang, Jianhua Chen, et al. “No-reference perceptual image quality metric using gradient profiles for JPEG2000”. In: *Signal Processing: Image Communication* 25.7 (2010), pp. 502–516 (cit. on p. 137).

[Liu+10] Hantao Liu, Nick Klomp, and Ingrid Heynderickx. “A no-reference metric for perceived ringing artifacts in images”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 20.4 (2010), pp. 529–539 (cit. on p. 137).

[Liu+16] Lixiong Liu, Yi Hua, Qingjie Zhao, Hua Huang, and Alan Conrad Bovik. “Blind image quality assessment by relative gradient statistics and adaboosting neural network”. In: *Signal Processing: Image Communication* 40 (2016), pp. 1 –15 (cit. on pp. 70, 142, 153, 162, 164, 173).

[Li+11] Chaofeng Li, Alan Conrad Bovik, and Xiaojun Wu. “Blind image quality assessment using a general regression neural network”. In: *IEEE Transactions on Neural Networks* 22.5 (2011), pp. 793–799 (cit. on pp. 142, 155).

[Li+14] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. “Efficient mini-batch training for stochastic optimization”. In: *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2014, pp. 661–670 (cit. on pp. 144, 156).

[Li+15] Tzu-Mao Li, Jaakko Lehtinen, Ravi Ramamoorthi, Wenzel Jakob, and Frédo Durand. “Anisotropic Gaussian Mutations for Metropolis Light Transport Through Hessian-Hamiltonian Dynamics”. In: *ACM Trans. Graph.* 34.6 (2015), 209:1–209:13 (cit. on pp. 52, 53, 58, 71, 114).

[LL03] Lie-Quan Lee and A. Lumsdaine. “The Generic Message Passing framework”. In: *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*. 2003 (cit. on p. 77).

[LM15a] Guillaume Lavoué and Rafał Mantiuk. “Quality assessment in computer graphics”. In: *Visual Signal Quality Assessment*. Springer, 2015, pp. 243–286 (cit. on p. 139).

[Lon+15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. “Fully convolutional networks for semantic segmentation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3431–3440 (cit. on p. 154).

[LP08] Feng Li and Fatih Porikli. “Harmonic variance: A novel measure for in-focus segmentation”. In: *Trans. PAMI* 30.10 (2008), pp. 1699–1712 (cit. on p. 26).

[LP12] Sangwoo Lee and Sang Ju Park. “A new image quality assessment method to detect and measure strength of blocking artifacts”. In: *Signal Processing: Image Communication* 27.1 (2012), pp. 31–38 (cit. on p. 137).

[Lub95] Jeffrey Lubin. “A visual discrimination model for imaging system design and evaluation”. In: *Vision Models for Target Detection and Recognition: In Memory of Arthur Menendez*. World Scientific, 1995, pp. 245–283 (cit. on p. 56).

[Lu+10] Wen Lu, Kai Zeng, Dacheng Tao, Yuan Yuan, and Xinbo Gao. “No-reference image quality assessment in contourlet domain”. In: *Neurocomputing* 73.4 (2010), pp. 784–794 (cit. on p. 137).

[LW93] Eric P Lafortune and Yves D Willems. “Bi-directional path tracing”. In: *Proceedings of CompuGraphics*. Vol. 93. 1993, pp. 145–153 (cit. on pp. 44, 49, 57, 71, 114).

[Maa+13a] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *Proc. icml*. Vol. 30. 1. 2013, p. 3 (cit. on p. 63).

[Man+11] Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. “HDR-VDP-2: a calibrated visual metric for visibility and quality predictions in all luminance conditions”. In: *ACM Transactions on Graphics (TOG)* 30.4 (2011), p. 40 (cit. on pp. 69, 117, 119, 136).

[Man+15] Marco Manzi, Markus Kettunen, Miika Aittala, et al. “Gradient-domain bidirectional path tracing”. In: (2015) (cit. on pp. 55, 58).

[Mar+04] Pina Marziliano, Frederic Dufaux, Stefan Winkler, and Touradj Ebrahimi. “Perceptual blur and ringing metrics: application to JPEG2000”. In: *Signal processing: Image communication* 19.2 (2004), pp. 163–172 (cit. on p. 137).

[MB10] Anush Krishna Moorthy and Alan Conrad Bovik. “A two-step framework for constructing blind image quality indices”. In: *IEEE Signal processing letters* 17.5 (2010), pp. 513–516 (cit. on pp. 70, 137, 141, 153, 162, 164, 173).

[MB11] Anush Krishna Moorthy and Alan Conrad Bovik. “Blind image quality assessment: From natural scene statistics to perceptual quality”. In: *IEEE transactions on Image Processing* 20.12 (2011), pp. 3350–3364 (cit. on p. 137).

[McC+96] B. C. McCandless, J. M. Squyres, and A. Lumsdaine. “Object Oriented MPI (OOMPI): a class library for the Message Passing Interface”. In: *MPI Developer’s Conference, 1996.* 1996, pp. 87–94 (cit. on pp. 77, 167).

[Mil15] Phil Miller. “Productive Parallel Programming with CHARM++”. In: *Proceedings of the Symposium on High Performance Computing*. HPC ’15. Alexandria, Virginia: Society for Computer Simulation International, 2015, pp. 241–242 (cit. on p. 78).

[Mit+12a] A. Mittal, A. K. Moorthy, and A. C. Bovik. “No-Reference Image Quality Assessment in the Spatial Domain”. In: *IEEE Transactions on Image Processing* 21.12 (2012), pp. 4695–4708 (cit. on pp. 70, 137, 153, 162, 164, 173).

[Mit+12b] Anish Mittal, Gautam S Muralidhar, Joydeep Ghosh, and Alan C Bovik. “Blind image quality assessment without human training using latent quality factors”. In: *IEEE Signal Processing Letters* 19.2 (2012), pp. 75–78 (cit. on p. 137).

[Mit+12c] Anish Mittal, Anush Krishna Moorthy, and Alan Conrad Bovik. “No-reference image quality assessment in the spatial domain”. In: *IEEE Transactions on Image Processing* 21.12 (2012), pp. 4695–4708 (cit. on p. 137).

[Mit+13a] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. “Making a “completely blind” image quality analyzer”. In: *IEEE Signal Processing Letters* 20.3 (2013), pp. 209–212 (cit. on pp. 70, 137, 153, 162, 164, 173).

[Mit+13b] Anish Mittal, Rajiv Soundararajan, and Alan C Bovik. “Making a “completely blind” image quality analyzer”. In: *IEEE Signal Processing Letters* 20.3 (2013), pp. 209–212 (cit. on p. 137).

[MP43] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cit. on p. 59).

[MV93] Theophano Mitsa and Krishna Lata Varkur. “Evaluation of contrast sensitivity functions for the formulation of quality measures incorporated in halftoning algorithms”. In: *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*. Vol. 5. IEEE. 1993, pp. 301–304 (cit. on p. 115).

[Mys98] Karol Myszkowski. “The visible differences predictor: Applications to global illumination problems”. In: *Rendering Techniques’ 98*. Springer, 1998, pp. 223–236 (cit. on pp. 56, 58, 164).

[Mys+00] Karol Myszkowski, Przemyslaw Rokita, and Takehiro Tawara. “Perception-based fast rendering and antialiasing of walkthrough sequences”. In: *IEEE Transactions on Visualization and Computer Graphics* 6.4 (2000), pp. 360–379 (cit. on pp. 56, 58, 164).

[Nea11] R Neal. “MCMC using Hamiltonian dynamics”. In: *Handbook of Markov Chain Monte Carlo* (2011), pp. 113–162 (cit. on p. 20).

[Nei10] Richard D. Neidinger. “Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming”. In: *SIAM Review* 52.3 (2010), pp. 545–563 (cit. on pp. 53, 66).

[NK11] Niranjan D Narvekar and Lina J Karam. “A no-reference image blur metric based on the cumulative probability of blur detection (CPBD)”. In: *IEEE Transactions on Image Processing* 20.9 (2011), pp. 2678–2683 (cit. on p. 137).

[Pap85] Athanasios Papoulis. *Random Variables and Stochastic Processes*. 1985 (cit. on p. 14).

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010 (cit. on p. 114).

[Pon+09] Nikolay Ponomarenko, Vladimir Lukin, Alexander Zelensky, et al. “TID2008-a database for evaluation of full-reference visual quality assessment metrics”. In: *Advances of Modern Radioelectronics* 10.4 (2009), pp. 30–45 (cit. on pp. 6, 117, 132, 137, 138, 162, 172).

[Pon+13] N. Ponomarenko, O. Ieremeiev, V. Lukin, et al. “Color image database TID2013: Peculiarities and preliminary results”. In: *European Workshop on Visual Information Processing (EUVIP)*. 2013, pp. 106–111 (cit. on pp. 6, 117, 132, 138, 162, 172).

[Ram+99] Mahesh Ramasubramanian, Sumanta N Pattanaik, and Donald P Greenberg. “A perceptually based physical error metric for realistic image synthesis”. In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1999, pp. 73–82 (cit. on pp. 56, 58, 164).

[RB93] Martin Riedmiller and Heinrich Braun. “A direct adaptive method for faster backpropagation learning: The RPROP algorithm”. In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 586–591 (cit. on p. 140).

[Red+17] K. S. Reddy, U. Singh, and P. K. Uttam. “Effect of image colourspace on performance of convolution neural networks”. In: *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2017, pp. 2001–2005 (cit. on p. 142).

[Rei+02] Erik Reinhard, Michael Stark, Peter Shirley, and James Ferwerda. “Photographic tone reproduction for digital images”. In: *ACM transactions on graphics (TOG)* 21.3 (2002), pp. 267–276 (cit. on p. 120).

[Ren07] Éric Renault. “Extended MPICC to Generate MPI Derived Datatypes from C Datatypes Automatically”. In: *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 14th European PVM/MPI User’s Group Meeting*. Ed. by Franck Cappello, Thomas Herault, and Jack Dongarra. Springer Berlin Heidelberg, 2007, pp. 307–314 (cit. on p. 77).

[RM51] Herbert Robbins and Sutton Monro. “A stochastic approximation method”. In: *The annals of mathematical statistics* (1951), pp. 400–407 (cit. on p. 141).

[Ros57] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957 (cit. on pp. 59, 68).

[Rou+12a] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. “Adaptive Rendering with Non-local Means Filtering”. In: *ACM Trans. Graph.* 31.6 (2012), 195:1–195:11 (cit. on p. 114).

[Rou+12b] Fabrice Rousselle, Claude Knaus, and Matthias Zwicker. “Adaptive rendering with non-local means filtering”. In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 195 (cit. on p. 140).

[Rum+85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning internal representations by error propagation*. Tech. rep. California University San Diego La Jolla Inst for Cognitive Science, 1985 (cit. on pp. 59, 65, 68).

[Rum+95] David E. Rumelhart, Richard Durbin, Richard Golden, and Yves Chauvin. “Backpropagation”. In: ed. by Yves Chauvin and David E. Rumelhart. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995. Chap. Backpropagation: The Basic Theory, pp. 1–34 (cit. on pp. 65, 68).

[RW94] Holly E. Rushmeier and Gregory J. Ward. “Energy Preserving Non-linear Filters”. In: *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’94. New York, NY, USA: ACM, 1994, pp. 131–138 (cit. on p. 114).

[Saa+10] Michele A Saad, Alan C Bovik, and Christophe Charrier. “A DCT statistics-based blind image quality index”. In: *IEEE Signal Processing Letters* 17.6 (2010), pp. 583–586 (cit. on pp. 70, 137, 153, 162, 164, 173).

[Saa+12] Michele A Saad, Alan C Bovik, and Christophe Charrier. “Blind image quality assessment: A natural scene statistics approach in the DCT domain”. In: *IEEE Transactions on Image Processing* 21.8 (2012), pp. 3339–3352 (cit. on p. 137).

[Saz+08] ZM Parvez Sazzad, Yoshikazu Kawayoke, and Yuukou Horita. “No reference image quality assessment for JPEG2000 based on spatial features”. In: *Signal Processing: Image Communication* 23.4 (2008), pp. 257–268 (cit. on p. 137).

[SB06] Hamid R Sheikh and Alan C Bovik. “Image information and visual quality”. In: *IEEE Transactions on image processing* 15.2 (2006), pp. 430–444 (cit. on pp. 69, 116, 119, 141).

[Ser+13] Amina Serir, Azeddine Beghdadi, and Fatma Kerouh. “No-reference blur image quality measure based on multiplicative multiresolution decomposition”. In: *Journal of Visual Communication and Image Representation* 24.7 (2013), pp. 911–925 (cit. on p. 137).

[SH17] Mateu Sbert and Vlastimil Havran. “Adaptive Multiple Importance Sampling for General Functions”. In: *Vis. Comput.* 33.6-8 (2017), pp. 845–855 (cit. on p. 26).

[She+05a] H. R. Sheikh, A. C. Bovik, and G. de Veciana. “An Information Fidelity Criterion for Image Quality Assessment Using Natural Scene Statistics”. In: *Trans. Img. Proc.* 14.12 (2005), pp. 2117–2128 (cit. on pp. 69, 116, 119).

[She+05b] H. R. Sheikh, A. C. Bovik, and L. Cormack. “No-reference quality assessment using natural scene statistics: JPEG2000”. In: *IEEE Transactions on Image Processing* 14.11 (2005), pp. 1918–1927 (cit. on pp. 70, 137, 153, 162, 164, 173).

[She+05c] Hamid R Sheikh, Alan C Bovik, and Lawrence Cormack. “No-reference quality assessment using natural scene statistics: JPEG2000”. In: *IEEE Transactions on Image Processing* 14.11 (2005), pp. 1918–1927 (cit. on p. 137).

[She+06] H.R. Sheikh, M.F. Sabir, and A.C. Bovik. “A Statistical Evaluation of Recent Full Reference Image Quality Assessment Algorithms”. In: *Image Processing, IEEE Transactions on* 15.11 (2006), pp. 3440–3451 (cit. on pp. 115, 117).

[She+14] H. R. Sheikh, Z. Wang, L. Cormack, and A. C. Bovik. *LIVE Image Quality Assessment Database Release 2*. 2014 (cit. on pp. 6, 117, 132, 137, 138, 142, 162, 172).

[Shi+96] Peter Shirley, Changyaw Wang, and Kurt Zimmerman. “Monte Carlo techniques for direct lighting calculations”. In: *ACM Transactions on Graphics (TOG)* 15.1 (1996), pp. 1–36 (cit. on p. 38).

[SP94] Douglas Shy and Pietro Perona. “XY separable pyramid steerable scalable kernels”. In: *CVPR*. 1994, pp. 237–244 (cit. on p. 115).

[Sri+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. “Dropout: A simple way to prevent neural networks from overfitting”. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958 (cit. on pp. 142, 143).

[Sur+09] Sundaram Suresh, R Venkatesh Babu, and HJ Kim. “No-reference image quality assessment using modified extreme learning machine classifier”. In: *Applied Soft Computing* 9.2 (2009), pp. 541–552 (cit. on p. 137).

[Sut09] Shan Suthaharan. “No-reference visually significant blocking artifact metric for natural scene images”. In: *Signal Processing* 89.8 (2009), pp. 1647–1652 (cit. on p. 137).

[SW00] Frank Suykens and Yves D. Willems. “Adaptive filtering for progressive Monte Carlo image rendering”. In: *The 8-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media 2000 (WSCG’ 2000), February 2000. Held in Plzen, Czech Republic*. 2000 (cit. on p. 114).

[SZ14] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014) (cit. on pp. 67, 68, 154).

[Sze+14] C Szegedy, W Liu, Y Jia, et al. “Going deeper with convolutions, CoRR abs/1409.4842”. In: *arXiv preprint arXiv:1409.4842* (2014) (cit. on pp. 67, 68, 154).

[Tao+09] D. Tao, X. Li, W. Lu, and X. Gao. “Reduced-Reference IQA in Contourlet Domain”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 39.6 (2009), pp. 1623–1627 (cit. on pp. 70, 117, 119).

[TH12] T Tieleman and G Hinton. “RMSprop Gradient Optimization”. In: *Lecture Notes* (2012) (cit. on p. 66).

[TJ97] Rasmus Tamstorf and Henrik Wann Jensen. “Adaptive Sampling and Bias Estimation in Path Tracing”. In: *Proceedings of the Eurographics Workshop on Rendering Techniques ’97*. London, UK, UK: Springer-Verlag, 1997, pp. 285–296 (cit. on p. 114).

[Toe+89] Alexander Toet, Lodewik J Van Ruyven, and J Mathee Valeton. “Merging thermal and visual images by a contrast pyramid”. In: *Optical engineering* 28.7 (1989), pp. 287789–287789 (cit. on p. 115).

[Tof15] Chris Tofallis. “A better measure of relative prediction accuracy for model selection and model estimation”. In: *Journal of the Operational Research Society* 66.8 (2015), pp. 1352–1362 (cit. on p. 122).

[TT08] W. Tansey and E. Tilevich. “Efficient automated marshaling of C++ data structures for MPI applications”. In: *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*. 2008, pp. 1–12 (cit. on p. 78).

[Vea97] Eric Veach. “Robust Monte Carlo methods for light transport simulation”. PhD thesis. Stanford University, 1997 (cit. on pp. 25, 44, 49, 57, 71).

[VG95] Eric Veach and Leonidas J. Guibas. “Optimally Combining Sampling Techniques for Monte Carlo Rendering”. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’95. New York, NY, USA: ACM, 1995, pp. 419–428 (cit. on p. 25).

[VG97] Eric Veach and Leonidas J. Guibas. “Metropolis Light Transport”. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 65–76 (cit. on pp. 49, 57, 58, 71, 114).

[Vis+10] Abhinav Vishnu, Huub Van Dam, Wibe de Jong, Pavan Balaji, and Shuaiwen Song. “Fault-tolerant communication runtime support for data-centric programming models”. In: *2010 International Conference on High Performance Computing, HiPC 2010, Dona Paula, Goa, India, December 19-22, 2010*. 2010, pp. 1–9 (cit. on p. 79).

[VS05] P Vetrivelan and RR Subha. “Wavelet based contourlet transform for image compression”. In: *Proceeding of International conference of Cognition and Recognition. IEEE*. Citeseer. 2005, pp. 915–919 (cit. on p. 115).

[Wan+03] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. “Multiscale structural similarity for image quality assessment”. In: *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*. Vol. 2. Ieee. 2003, pp. 1398–1402 (cit. on pp. 69, 115, 119, 136, 138, 171, 174).

[Wan+04a] Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. “Image Quality Assessment: From Error Visibility to Structural Similarity”. In: *Trans. Img. Proc.* 13.4 (2004), pp. 600–612 (cit. on pp. 69, 114, 115, 119, 136, 141, 154, 163, 173).

[WB02] Zhou Wang and A. C. Bovik. “A universal image quality index”. In: *IEEE Signal Processing Letters* 9.3 (2002), pp. 81–84 (cit. on pp. 69, 115, 119).

[WB09] Z. Wang and A. C. Bovik. “Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures”. In: *IEEE Signal Processing Magazine* 26.1 (2009), pp. 98–117 (cit. on p. 115).

[Whi80] Turner Whitted. “An Improved Illumination Model for Shaded Display”. In: *Commun. ACM* 23.6 (June 1980), pp. 343–349 (cit. on p. 30).

[Whi+16] Joss Whittle, Rita Borgo, and Mark W. Jones. “Implementing generalized deep-copy in MPI”. In: *PeerJ Computer Science* 2 (2016), e95 (cit. on p. 74).

[Whi+17] Joss Whittle, Mark W. Jones, and Rafał Mantiuk. “Analysis of reported error in Monte Carlo rendered images”. In: *The Visual Computer* 33.6 (2017), pp. 705–713 (cit. on p. 113).

[WL11] Z. Wang and Q. Li. “Information Content Weighting for Perceptual Image Quality Assessment”. In: *IEEE Transactions on Image Processing* 20.5 (2011), pp. 1185–1198 (cit. on pp. 69, 116, 119).

[WM05] Cort J Willmott and Kenji Matsuura. “Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance”. In: *Climate research* 30.1 (2005), p. 79 (cit. on p. 115).

[Wu+09] Shiqian Wu, Weisi Lin, Shoulie Xie, et al. “Blind blur assessment for vision-based applications”. In: *Journal of Visual Communication and Image Representation* 20.4 (2009), pp. 231–241 (cit. on p. 137).

[Xu+15] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *CoRR* abs/1505.00853 (2015). arXiv: `1505.00853` (cit. on p. 63).

[YD12] Peng Ye and David Doermann. “No-reference image quality assessment using visual codebooks”. In: *IEEE Transactions on Image Processing* 21.7 (2012), pp. 3129–3138 (cit. on p. 137).

[Yee+01a] Hector Yee, Sumanita Pattanaik, and Donald P Greenberg. “Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments”. In: *ACM Transactions on Graphics (TOG)* 20.1 (2001), pp. 39–65 (cit. on p. 139).

[YK03] Christopher C Yang and Sai Ho Kwok. “Efficient gamut clipping for color image processing using LHS and YIQ”. In: *Optical Engineering* 42.3 (2003), pp. 701–711 (cit. on p. 116).

[Zha+08] Guangtao Zhai, Wenjun Zhang, Xiaokang Yang, Weisi Lin, and Yi Xu. “No-reference noticeable blockiness estimation in images”. In: *Signal Processing: Image Communication* 23.6 (2008), pp. 417–432 (cit. on p. 137).

[Zha+11c] L. Zhang, L. Zhang, X. Mou, and D. Zhang. “FSIM: A Feature Similarity Index for Image Quality Assessment”. In: *IEEE Transactions on Image Processing* 20.8 (2011), pp. 2378–2386 (cit. on pp. 69, 116, 119).

[Zha+17a] He Zhang, Vishwanath Sindagi, and Vishal M. Patel. “Image De-raining Using a Conditional Generative Adversarial Network”. In: *CoRR* abs/1701.05957 (2017) (cit. on pp. 140, 141).

[Zha+17b] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang. “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising”. In: *IEEE Transactions on Image Processing* 26.7 (2017), pp. 3142–3155 (cit. on pp. 140, 155).

[ZL10] Jing Zhang and Thinh M Le. “A new no-reference quality metric for JPEG2000 images”. In: *IEEE Transactions on Consumer Electronics* 56.2 (2010) (cit. on p. 137).

[ZMS07] Izak van Zyl Marais and Willem Herman Steyn. “Robust defocus blur identification in the context of blind image quality assessment”. In: *Signal Processing: Image Communication* 22.10 (2007), pp. 833–844 (cit. on p. 137).

[Zwi02] D. Zwillinger. *CRC Standard Mathematical Tables and Formulae, 31st Edition*. Advances in Applied Mathematics. CRC Press, 2002 (cit. on p. 14).

[Mes14] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard Version 3.1*. Tech. rep. Stuttgart, DE: High Performance Computing Center Stuttgart (HLRS), 2014 (cit. on p. 76).

Appendix A

A.1 MEL Experiment 1: Broadcasting a large tree structure

Full code for this example is available at `https://github.com/CS-Swansea/MEL/` under `example-code/RayTracingDeepCopy.cpp`

```cpp
// Example Usage:
// mpirun -n [number of processes] ./RayTracingDeepCopy [mesh path] [method]
// mpirun -n 8 ./RayTracingDeepCopy "Teapot.obj" 0

int main(int argc, char *argv[]) {
    MEL::Init(argc, argv); // Setup

    // Who are we?
    MEL::Comm comm = MEL::Comm::WORLD;
    const int rank = MEL::CommRank(comm),
              size = MEL::CommSize(comm);

    // Check param count
    if (argc != 3) {
        if (rank == 0)
            std::cout << "Wrong number of parameters..." << std::endl;
        MEL::Exit(-1);
        // ^^ Equivalent of calling MPI_Finalize() followed by std::exit(-1)
    }

    // Which model should we load and which algorithm should we use?
    const std::string meshPath = std::string(argv[1]);
    const int         method   = std::stoi(argv[2]);

    // Load the scene on the root process
    Scene *scene = nullptr;
    if (rank == 0) {
        std::cout << "Loading scene..." << std::endl;
        scene = loadScene(meshPath);
    }

    MEL::Barrier(comm);
    auto startTime = MEL::Wtime(); // Start the clock!

    // Broadcast the Scene structure with the selected method
    switch (method) {
    case 0:
        // Call MEL::Deep method.
        MEL::Deep::Bcast(scene, 0, comm);
        break;
    case 1:
        // Call MEL::Deep method.
        MEL::Deep::BufferedBcast(scene, 0, comm);
        break;
    case 2:
        // Hand written below
        MPI_NonBufferedBcast_Scene(scene, 0, (MPI_Comm) comm);
        break;
    case 3:
        // Hand written below
        MPI_BufferedBcast_Scene(scene, 0, (MPI_Comm) comm);
        break;
    default:
        if (rank == 0) std::cout << "Invalid method index..." << std::endl;
        MEL::Exit(-1);
    }

    MEL::Barrier(comm);
    auto endTime = MEL::Wtime(); // Stop the clock!

    if (rank == 0) {
        std::cout << "Broadcast Scene in " << (endTime - startTime)
                  << " seconds..." << std::endl;
    }

    // All processes now have a Scene pointer that points to an equivalent
    // data-structure

    // Now we can do some ray-tracing using the scene object!

    // Clean up
    MEL::MemDestruct(scene);
    // ^^ Equivalent to explicitly calling the destructor
    //    followed by MPI_Free_mem.
    // scene->~Scene();
    // MPI_Free_mem(scene);

    MEL::Finalize(); // Tear down
    return 0;
}
```

Listing A.1: Deep Copy of Ray Tracing Scene Object

A.1.1 Scene object containing MEL Deep Copy methods

```
// Structure representing a node in the BVH Tree
struct TreeNode {
    int startElem, endElem; // Start and End indices into vector of triangles
    Vec v0, v1; // Vec is non-deep struct
    TreeNode *leftChild, *rightChild; // TreeNode is deep struct

    TreeNode() : TreeNode(0, 0) {}
    TreeNode(const int _s, const int _e)
        : startElem(_s), endElem(_e),
          leftChild(nullptr), rightChild(nullptr),
          v0{ INF, INF, INF }, v1{ -INF, -INF, -INF } {}

    // Ensure TreeNode can't be used incorrectly
    TreeNode(const TreeNode &old)                     = delete;
    inline TreeNode& operator=(const TreeNode &old)   = delete;
    TreeNode(TreeNode &&old)                          = delete;
    inline TreeNode& operator=(TreeNode &&old)        = delete;

    ~TreeNode() {
        MEL::MemDestruct( leftChild);
        MEL::MemDestruct(rightChild);
    }

    // Implementation of Ray-TreeNode (Ray-AABB) intersection
    // omitted for this example
    bool intersect(const Ray &rayInv, double &tmin, const double dist) const;

    template<typename MSG>
    inline void DeepCopy(MSG &msg) {
        msg.packPtr( leftChild);
        msg.packPtr(rightChild);
    }
};

// Structure representing a scene object to be rendered
struct Scene {
    Camera                  camera;     // Camera is non-deep struct
    std::vector<Material> materials; // Material is non-deep struct
    std::vector<Triangle> mesh;      // Triangle is non-deep struct
    TreeNode               *rootNode;  // TreeNode is deep struct

    Scene() : rootNode(nullptr) {}

    // Ensure Scene can't be used incorrectly
    Scene(const Scene &old)                     = delete;
    inline Scene& operator=(const Scene &old) = delete;

    // Move Constructor
    Scene(Scene &&old)
        : mesh(std::move(old.mesh)), materials(std::move(old.materials)),
          camera(old.camera), rootNode(old.rootNode) {
        old.mesh.clear();
        old.materials.clear();
        old.rootNode = nullptr;
    }

    // Move Assignment Operator
    inline Scene& operator=(Scene &&old) {
        mesh      = std::move(old.mesh);
        materials = std::move(old.materials);
        rootNode  = old.rootNode;
        camera    = old.camera;
        old.mesh.clear();
        old.materials.clear();
        old.rootNode = nullptr;
        return *this;
    }

    ~Scene() {
        MEL::MemDestruct(rootNode);
    }

    // Implementation of Ray-Scene intersection omitted for this example
    bool intersect(const Ray &ray, Intersection &isect) const;

    template<typename MSG>
    inline void DeepCopy(MSG &msg) {
        msg & mesh & materials;
        msg.packPtr(rootNode);
    }
};
```

Listing A.2: Ray Tracing Scene Object

A.1.2 Hand coded Non-Buffered Bcast of Scene object

```
inline void MPI_NonBufferedBcast_Scene(Scene *&scene,
                                       const int root,
                                       const MPI_Comm comm) {
    int rank;
    MPI_Comm_rank(comm, &rank);

    // Receiving nodes allocate space for scene
    if (rank != root) {
        MPI_Alloc_mem(sizeof(Scene), MPI_INFO_NULL, &scene);
        new (scene) Scene();
    }

    // Bcast the camera struct
    MPI_Bcast(&(scene->camera), sizeof(Camera), MPI_CHAR, root, comm);

    // Bcast the vector sizes
    int sizes[2];
    if (rank == root) {
        sizes[0] = (int) scene->mesh.size();
        sizes[1] = (int) scene->materials.size();
    }
    MPI_Bcast(sizes, 2, MPI_INT, root, comm);

    // 'Allocate' space for vectors
    if (rank != root) {
        scene->mesh.resize(sizes[0]);
        scene->materials.resize(sizes[1]);
    }

    // Bcast the vectors
    MPI_Bcast(&(scene->mesh[0]), sizeof(Triangle) * sizes[0],
              MPI_CHAR, root, comm);
    MPI_Bcast(&(scene->materials[0]), sizeof(Material) * sizes[1],
              MPI_CHAR, root, comm);

    // Receiving nodes allocate space for rootNode
    if (rank != root) {
        MPI_Alloc_mem(sizeof(TreeNode), MPI_INFO_NULL, &(scene->rootNode));
        new (scene->rootNode) TreeNode();
    }

    // While the stack is not empty there is work to be done
    std::stack<TreeNode*> treeStack;
    treeStack.push(scene->rootNode);
    while (!treeStack.empty()) {
        // Get the current node to traverse
        TreeNode *currentNode = treeStack.top();
        treeStack.pop();

        // Bcast the current node's values
        MPI_Bcast((currentNode), sizeof(TreeNode), MPI_CHAR, root, comm);

        // Do we need to send/receive children?
        bool hasChildren = (currentNode->leftChild != nullptr);

        if (hasChildren) {
            // Allocate space for child nodes on receiving process
            if (rank != root) {
                MPI_Alloc_mem(sizeof(TreeNode),
                              MPI_INFO_NULL, &(currentNode->leftChild));
                MPI_Alloc_mem(sizeof(TreeNode),
                              MPI_INFO_NULL, &(currentNode->rightChild));

                // Default construct new nodes into allocated memory
                new (currentNode->leftChild)  TreeNode();
                new (currentNode->rightChild) TreeNode();
            }

            // Push children onto the stack so they get processed
            treeStack.push(currentNode->leftChild);
            treeStack.push(currentNode->rightChild);
        }
    }
}
```

Listing A.3: Hand coded Non-Buffered Bcast of Ray Tracing Scene Object

A.1.3 Hand coded Buffered Bcast of Scene object

```
inline void MPI_BufferedBcast_Scene(Scene *&scene, const int root, const
    MPI_Comm comm) {
    int rank;
    MPI_Comm_rank(comm, &rank);

    // Receiving nodes allocate space for scene
    if (rank != root) {
        MPI_Alloc_mem(sizeof(Scene), MPI_INFO_NULL, &scene);
        new (scene) Scene();
    }

    // Calculate the byte size of the tree on root process
    int packed_size = 0;
    if (rank == root) {
        packed_size += sizeof(Camera);
        packed_size += sizeof(int)
                    + ((int) scene->mesh.size() * sizeof(Triangle));
        packed_size += sizeof(int)
                    + ((int) scene->materials.size() * sizeof(Material));

        // While the stack is not empty there is work to be done
        std::stack<TreeNode*> treeStack;
        treeStack.push(scene->rootNode);
        while (!treeStack.empty()) {
            // Get the current node to traverse
            TreeNode *currentNode = treeStack.top();
            treeStack.pop();

            packed_size += sizeof(TreeNode);

            // Do we need to send children?
            bool hasChildren = (currentNode->leftChild != nullptr);

            if (hasChildren) {
                // Push children onto the stack so they get processed
                treeStack.push(currentNode->leftChild);
                treeStack.push(currentNode->rightChild);
            }
        }
    }

    // Share the buffer size to all processes
    MPI_Bcast(&packed_size, 1, MPI_INT, root, comm);

    // Allocate the buffer
    int position = 0;
    char *buffer;
    MPI_Alloc_mem(packed_size, MPI_INFO_NULL, &(buffer));

    // If root then we pack the structure into the buffer
    if (rank == root) {
        // Pack the camera struct
        MPI_Pack(&(scene->camera), sizeof(Camera),
                 MPI_CHAR, buffer, packed_size, &position, comm);

        int mesh_size      = scene->mesh.size(),
            materials_size = scene->materials.size();

        // Pack the mesh vector
        MPI_Pack(&(mesh_size), 1,
                 MPI_INT, buffer, packed_size, &position, comm);
        MPI_Pack(&(scene->mesh[0]), mesh_size * sizeof(Triangle),
                 MPI_CHAR, buffer, packed_size, &position, comm);

        // Pack the materials vector
        MPI_Pack(&(materials_size), 1,
                 MPI_INT, buffer, packed_size, &position, comm);
        MPI_Pack(&(scene->materials[0]), materials_size * sizeof(Material),
                 MPI_CHAR, buffer, packed_size, &position, comm);

        // While the stack is not empty there is work to be done
        std::stack<TreeNode*> treeStack;
        treeStack.push(scene->rootNode);
        while (!treeStack.empty()) {
            // Get the current node to traverse
            TreeNode *currentNode = treeStack.top();
            treeStack.pop();

            // Pack the current node
            MPI_Pack(currentNode, sizeof(TreeNode),
                     MPI_CHAR, buffer, packed_size, &position, comm);

            // Do we need to send children?
            bool hasChildren = (currentNode->leftChild != nullptr);

            if (hasChildren) {
                // Push children onto the stack so they get processed
                treeStack.push(currentNode->leftChild);
                treeStack.push(currentNode->rightChild);
            }
        }

        // Send the buffer
        MPI_Bcast(buffer, packed_size, MPI_CHAR, root, comm);
    }

    // If not root then we unpack the structure from the buffer
    else {
        // Receive the packed buffer
        MPI_Bcast(buffer, packed_size, MPI_CHAR, root, comm);

        MPI_Alloc_mem(sizeof(TreeNode), MPI_INFO_NULL, &(scene->rootNode));
        new (scene->rootNode) TreeNode();

        // Unpack the camera struct
        int mesh_size, materials_size;
        MPI_Unpack(buffer, packed_size, &position, &(scene->camera),
                   sizeof(Camera), MPI_CHAR, comm);

        // Unpack mesh vector
        MPI_Unpack(buffer, packed_size, &position, &(mesh_size),
                   1, MPI_INT, comm);

        scene->mesh.resize(mesh_size);

        MPI_Unpack(buffer, packed_size, &position, &(scene->mesh[0]),
                   mesh_size * sizeof(Triangle), MPI_CHAR, comm);

        // Unpack materials vector
        MPI_Unpack(buffer, packed_size, &position, &(materials_size),
                   1, MPI_INT, comm);

        scene->materials.resize(materials_size);

        MPI_Unpack(buffer, packed_size, &position, &(scene->materials[0]),
                   materials_size * sizeof(Material), MPI_CHAR, comm);

        // While the stack is not empty there is work to be done
        std::stack<TreeNode*> treeStack;
        treeStack.push(scene->rootNode);
        while (!treeStack.empty()) {
            // Get the current node to traverse
            TreeNode *currentNode = treeStack.top();
            treeStack.pop();

            // Unpack the current node
            MPI_Unpack(buffer, packed_size, &position, currentNode,
                       sizeof(TreeNode), MPI_CHAR, comm);
```

```cpp

            // Do we need to receive children?
            bool hasChildren = (currentNode->leftChild != nullptr);

            if (hasChildren) {
                // Allocate space for child nodes on receiving process
                MPI_Alloc_mem(sizeof(TreeNode), MPI_INFO_NULL,
                              &(currentNode->leftChild));
                MPI_Alloc_mem(sizeof(TreeNode), MPI_INFO_NULL,
                              &(currentNode->rightChild));

                new (currentNode->leftChild)  TreeNode();
                new (currentNode->rightChild) TreeNode();

                // Push children onto the stack so they get processed
                treeStack.push(currentNode->leftChild);
                treeStack.push(currentNode->rightChild);
            }
        }
    }

    // Clean up
    MPI_Free_mem(buffer);
}
```

Listing A.4: Hand coded Buffered Bcast of Ray Tracing Scene Object

A.2 MEL Experiment 2: Communicating Generic Directed Graph structures

```cpp
// Example Usage:
// mpirun -n [num of procs] ./GraphCycles [graph nodes] [graph type]
// mpirun -n 8 ./GraphCycles 11 0

int main(int argc, char *argv[]) {
    MEL::Init(argc, argv);

    MEL::Comm comm = MEL::Comm::WORLD;
    const int rank = MEL::CommRank(comm),
              size = MEL::CommSize(comm);

    if (argc != 3) {
        if (rank == 0)
            std::cout << "Wrong number of parameters..." << std::endl;
        MEL::Exit(-1);
    }

    const int numNodes  = 1 << std::stoi(argv[1]), // 2^n nodes
              graphType =      std::stoi(argv[2]);

    DiGraphNode<int> *graph = nullptr;
    if (rank == 0) {
        switch (graphType) {
        case 0:
            graph = MakeBTreeGraph(numNodes);
            break;
        case 1:
            graph = MakeRingGraph(numNodes);
            break;
        case 2:
            graph = MakeRandomGraph(numNodes);
            break;
        case 3:
            graph = MakeFullyConnectedGraph(numNodes);
            break;
        }
    }

    MEL::Barrier(comm);
    auto startTime = MEL::Wtime(); // Start the clock!

    // Deep copy the graph to all nodes
    MEL::Deep::Bcast(graph, 0, comm);

    MEL::Barrier(comm);
    auto endTime = MEL::Wtime(); // Stop the clock!

    if (rank == 0) {
        std::cout << "Broadcast Graph in " << (endTime - startTime)
                  << " seconds..." << std::endl;
    }

    // File name for output
    std::stringstream sstr;
    sstr <<   "rank=" << rank     << " type=" << graphType
         << " nodes=" << numNodes << ".graph";

    // Save the output to disk from each node
    std::ofstream graphFile(sstr.str(), std::ios::out | std::ios::binary);
    if (graphFile.is_open()) {
        MEL::Deep::FileWrite(graph, graphFile);
        graphFile.close();
    }

    DestructGraph(graph);

    MEL::Finalize();
    return 0;
}
```

Listing A.5: Functions for constructing Directed Graphs in different shapes

A.2.1 Factory functions for building Directed Graphs in different shaped structures

```
inline DiGraphNode<int>* MakeBTreeGraph(const int numNodes) {
    // BTree Graph
    std::vector<DiGraphNode<int>*> nodes(numNodes);
    for (int i = 0; i < numNodes; ++i) {
        nodes[i] = MEL::MemConstruct<DiGraphNode<int>>(i);
    }

    if (numNodes > 1) nodes[0]->edges.push_back(nodes[1]);

    for (int i = 1; i < numNodes; ++i) {
        const int j = ((i - 1) * 2) + 2;
        nodes[i]->edges.reserve(2);
        if (j < numNodes)       nodes[i]->edges.push_back(nodes[j]);
        if ((j + 1) < numNodes) nodes[i]->edges.push_back(nodes[j + 1]);
    }
    return nodes[0];
}

inline DiGraphNode<int>* MakeRingGraph(const int numNodes) {
    // Ring Graph
    std::vector<DiGraphNode<int>*> nodes(numNodes);
    for (int i = 0; i < numNodes; ++i) {
        nodes[i] = MEL::MemConstruct<DiGraphNode<int>>(i);
    }

    for (int i = 0; i < numNodes; ++i) {
        nodes[i]->edges.reserve(2);
        nodes[i]->edges.push_back(nodes[(i + 1) % numNodes]);
        nodes[i]->edges.push_back(nodes[(i == 0) ? (numNodes - 1) : (i - 1)]);
    }
    return nodes[0];
}

inline DiGraphNode<int>* MakeRandomGraph(const int numNodes) {
    srand(1234567);

    // Random Graph
    std::vector<DiGraphNode<int>*> nodes(numNodes);
    for (int i = 0; i < numNodes; ++i) {
        nodes[i] = MEL::MemConstruct<DiGraphNode<int>>(i);
    }

    for (int i = 0; i < numNodes; ++i) {
        const int numEdges = rand() % numNodes;
        nodes[i]->edges.reserve(numEdges);
        nodes[i]->edges.push_back(nodes[(i + 1) % numNodes]);
        for (int j = 1; j < numEdges; ++j) {
            nodes[i]->edges.push_back(nodes[rand() % numNodes]);
        }
    }
    return nodes[0];
}

inline DiGraphNode<int>* MakeFullyConnectedGraph(const int numNodes) {
    // Fully Connected Graph
    std::vector<DiGraphNode<int>*> nodes(numNodes);
    for (int i = 0; i < numNodes; ++i) {
        nodes[i] = MEL::MemConstruct<DiGraphNode<int>>(i);
    }

    for (int i = 0; i < numNodes; ++i) {
        nodes[i]->edges.reserve(numNodes);
        for (int j = 0; j < numNodes; ++j) {
            nodes[i]->edges.push_back(nodes[j]);
        }
    }
    return nodes[0];
}
```

Listing A.6: Functions for constructing Directed Graphs in different shapes

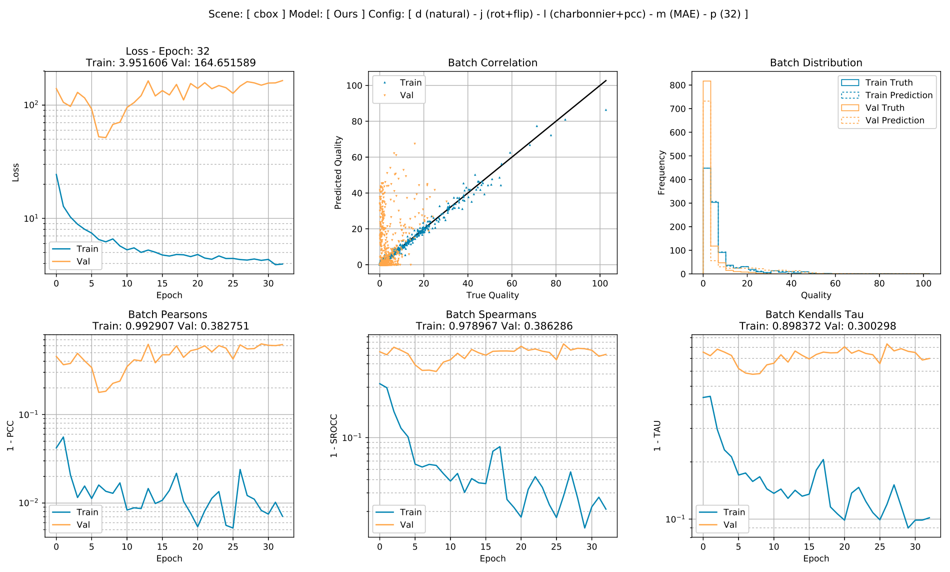
A.2.2 Generic implementation of Directed Graph container

```
template<typename T>
struct DiGraphNode {
    T value;
    std::vector<DiGraphNode<T>*> edges;

    DiGraphNode() {};
    explicit DiGraphNode(const T &_value) : value(_value) {};

    template<typename MSG>
    inline void DeepCopy(MSG &msg) {
        msg & edges;
        for (auto &e : edges) msg.packSharedPtr(e);
    }
};

inline void VisitGraph(DiGraphNode<int> *&root,
                       std::function<void(DiGraphNode<int> *&node)> func) {

    std::unordered_set<DiGraphNode<int>*> pointerMap;
    std::stack<DiGraphNode<int>*> stack;

    stack.push(root);
    while (!stack.empty()) {
        DiGraphNode<int> *node = stack.top();
        stack.pop();

        // If node has not been visited
        if (pointerMap.find(node) == pointerMap.end()) {
            pointerMap.insert(node);
            for (auto e : node->edges) stack.push(e);
            func(node);
        }
    }
}

inline void DestructGraph(DiGraphNode<int> *&root) {
    VisitGraph(root, [](DiGraphNode<int> *&node) -> void {
        MEL::MemDestruct(node);
    });
}
```

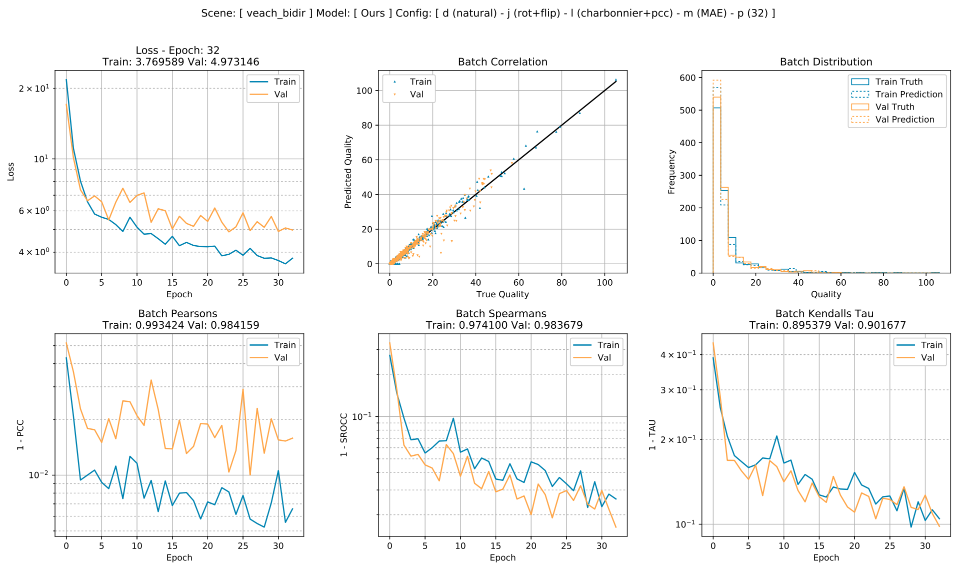
Listing A.7: Generic implementation of Directed Graph container for Deep Copy

A.3 NR-IQA Experiment 1: Patch Based

A.3.1 Training and Validation Curves without HSV Jitter

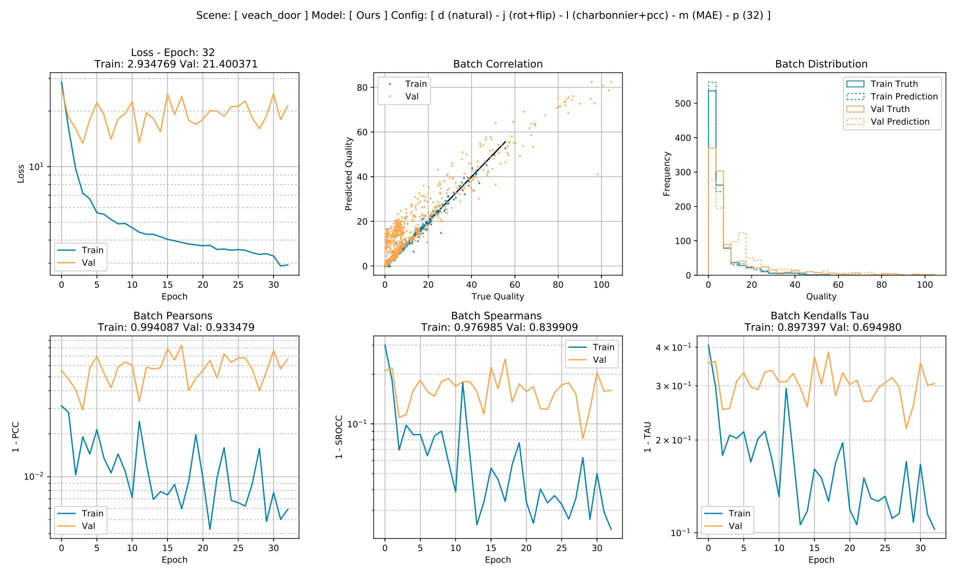


(a) Validation Scene: Cornell Box

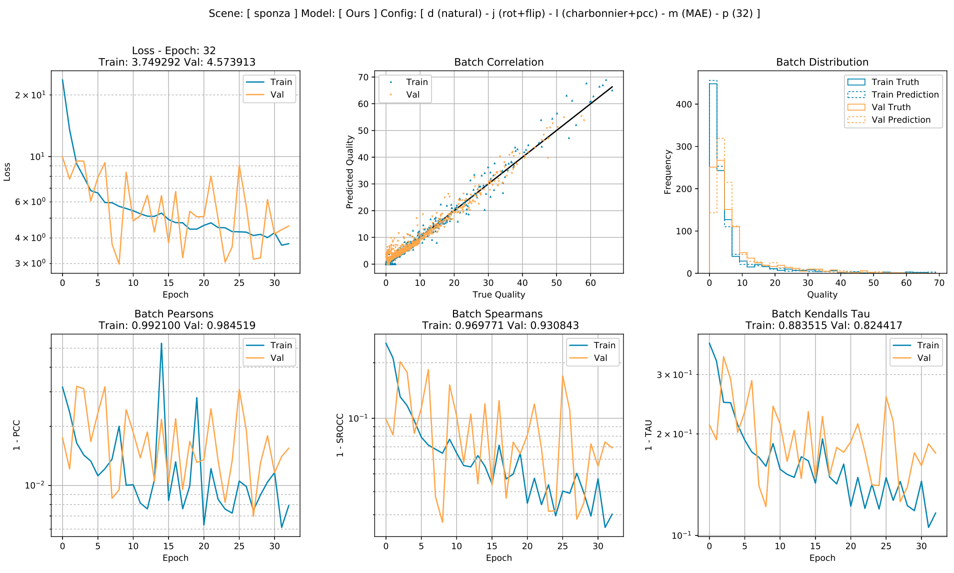


(b) Validation Scene: Veach Bidir

Fig. A.1.: Experiment I: Training and Validation set accuracies curves over 32 **epochs** for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, and flip jitter** and $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 32 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 32 epochs (top-right); and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training.



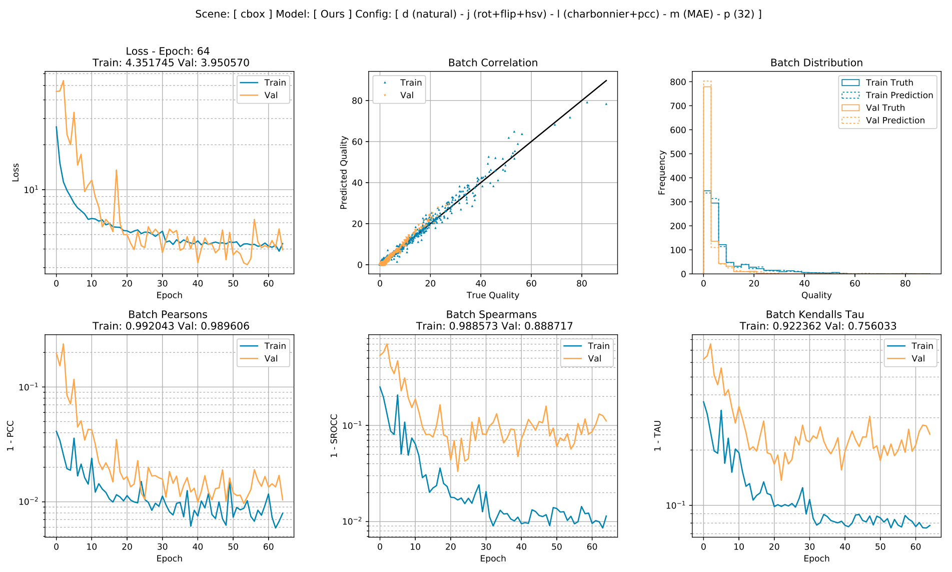
(a) Validation Scene: Veach Door



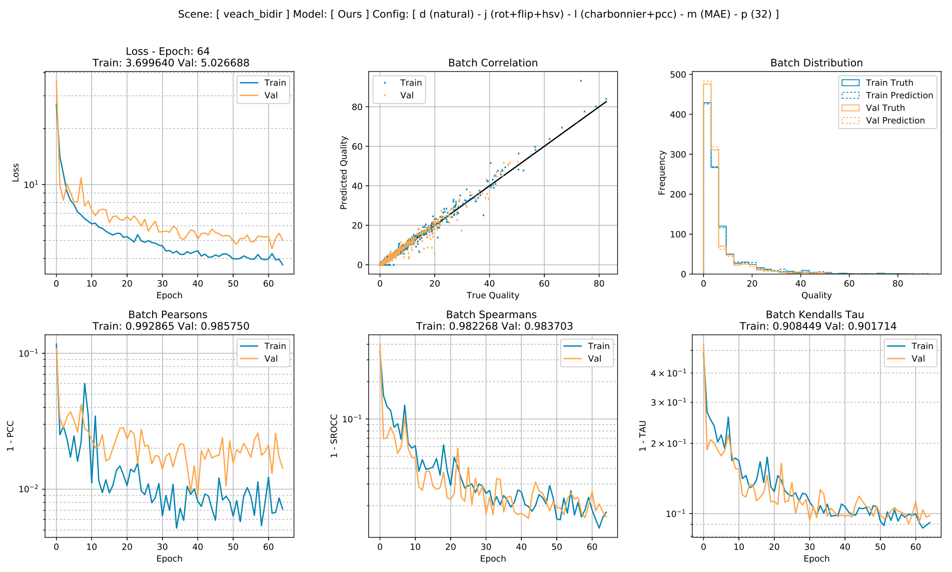
(b) Validation Scene: Sponza

Fig. A.2.: Experiment I: Training and Validation set accuracies curves over 32 **epochs** for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, and flip jitter** and the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 32 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 32 epochs (top-right); and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training.

A.3.2 Training and Validation Curves with HSV Jitter

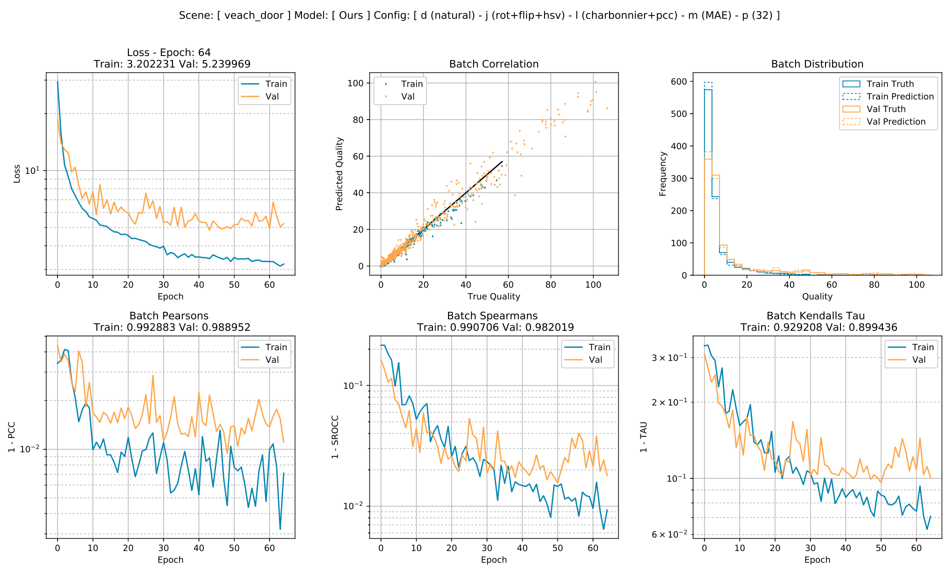


(a) Validation Scene: Cornell Box

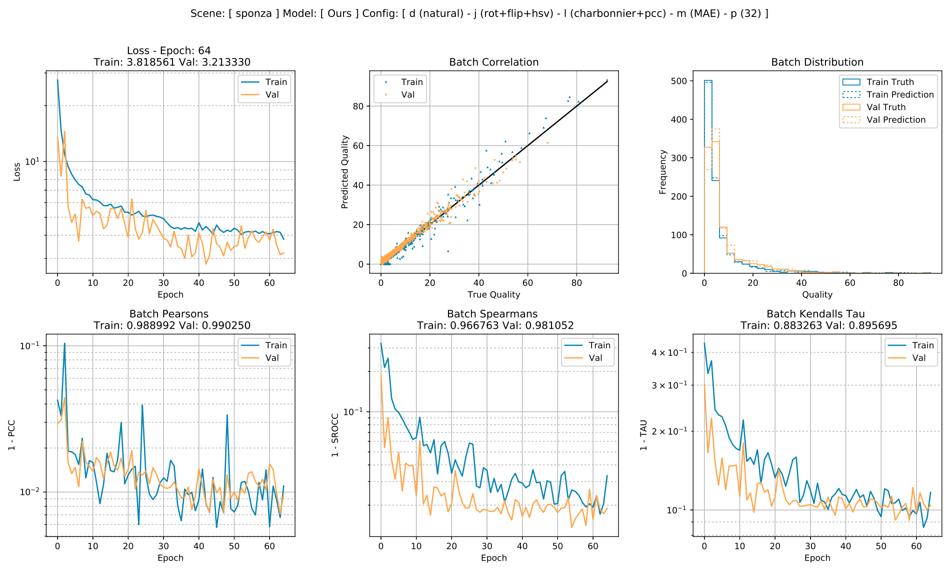


(b) Validation Scene: Veach Bidir

Fig. A.3.: Experiment I: Training and Validation set accuracies curves over 64 **epochs** for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** and the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 64 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 64 epochs (top-right); and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training.



(a) Validation Scene: Veach Door

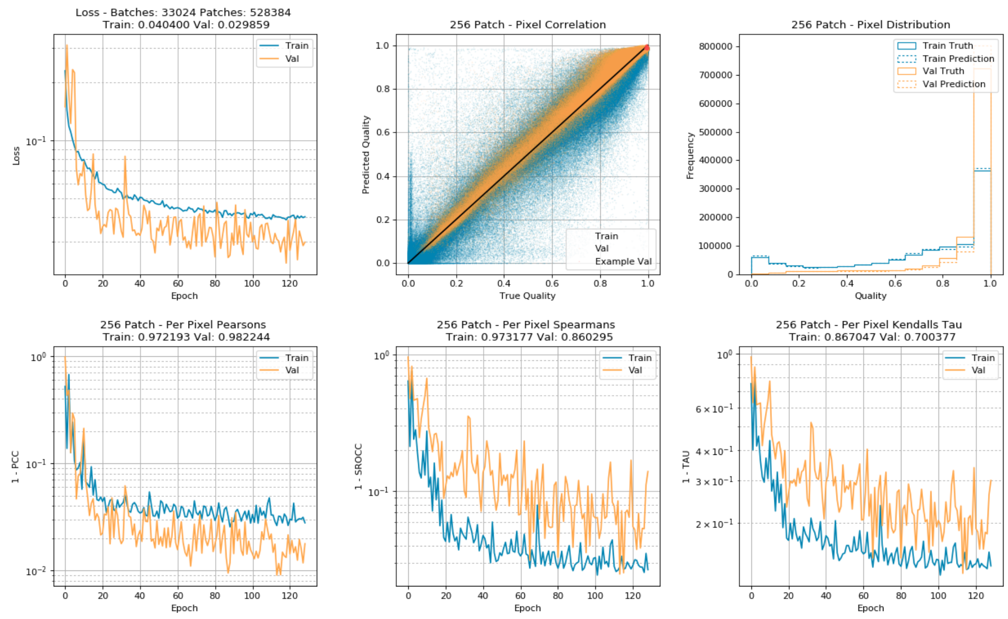


(b) Validation Scene: Sponza

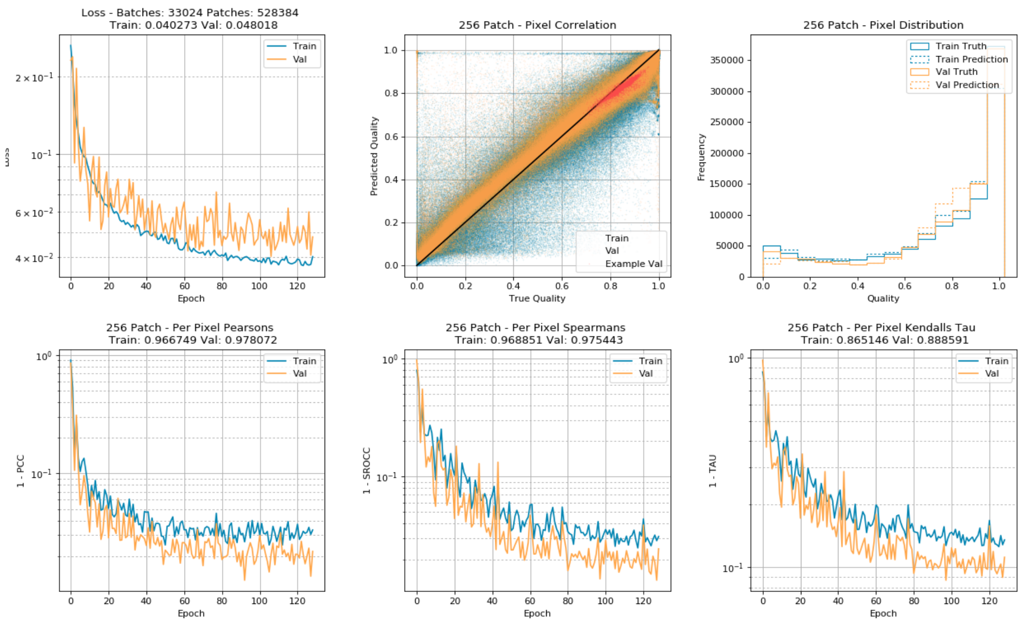
Fig. A.4.: Experiment I: Training and Validation set accuracies curves over 64 **epochs** for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, flip, and HSV jitter** and the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each patch in the most recent training and validation batch at the end of 64 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each patch in the most recent batches at the end of 64 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the last batch in each epoch during training.

A.4 NR-IQA Experiment 2: Full Convolutional

A.4.1 Training and Validation Curves for $\mathcal{L}_1$ Loss

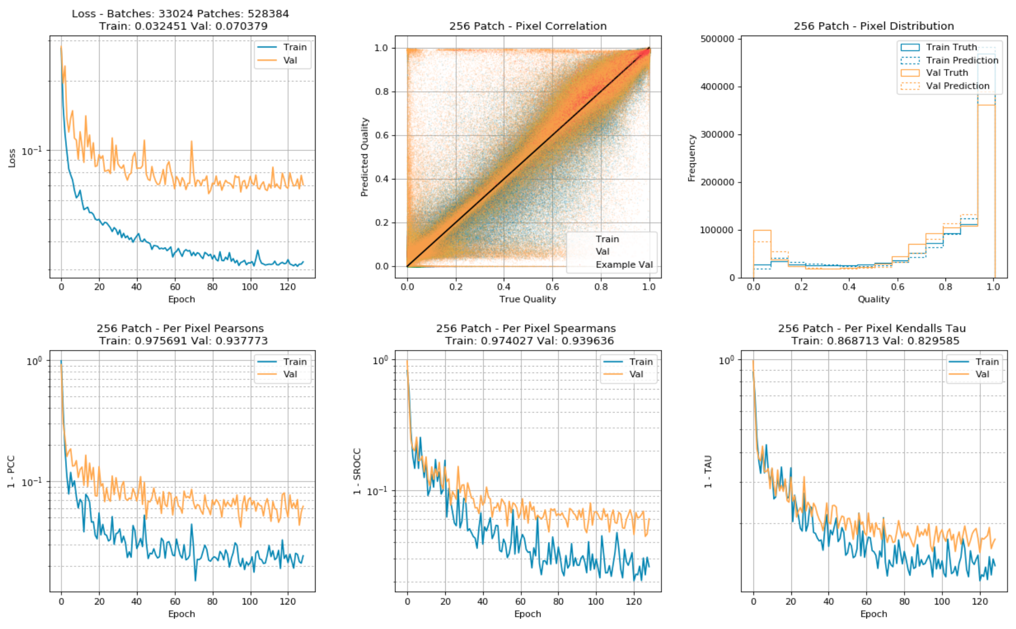


(a) Validation Scene: Cornell Box

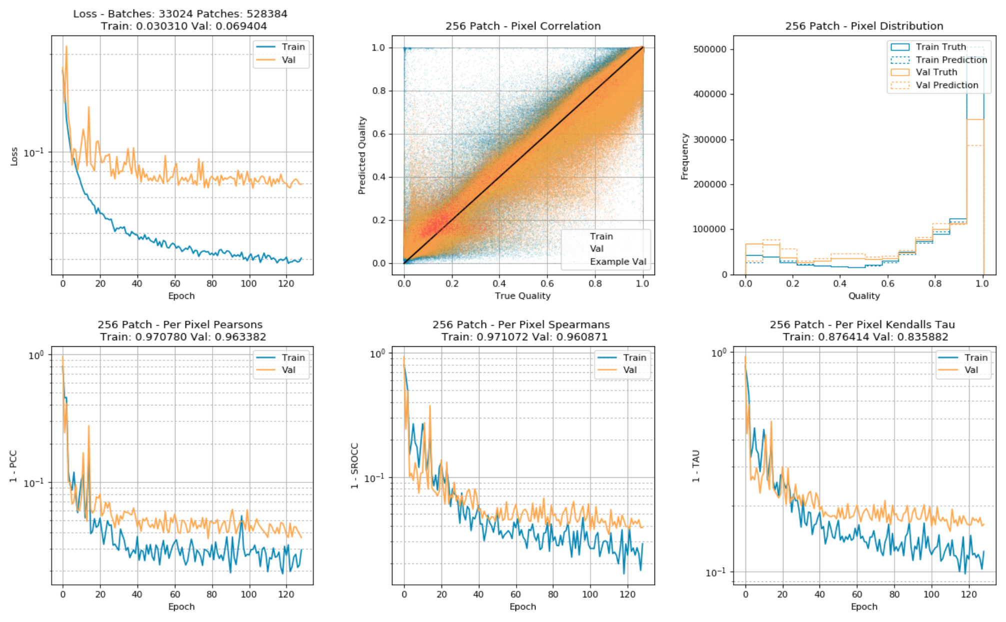


(b) Validation Scene: Veach Bidir

Fig. A.5.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_1$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.



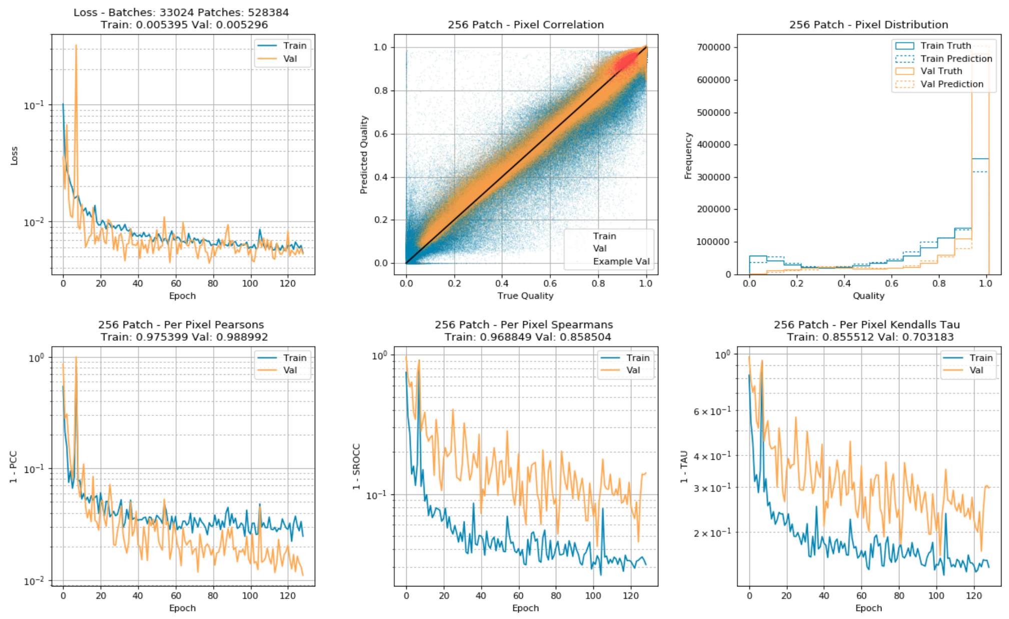
(a) Validation Scene: Veach Door



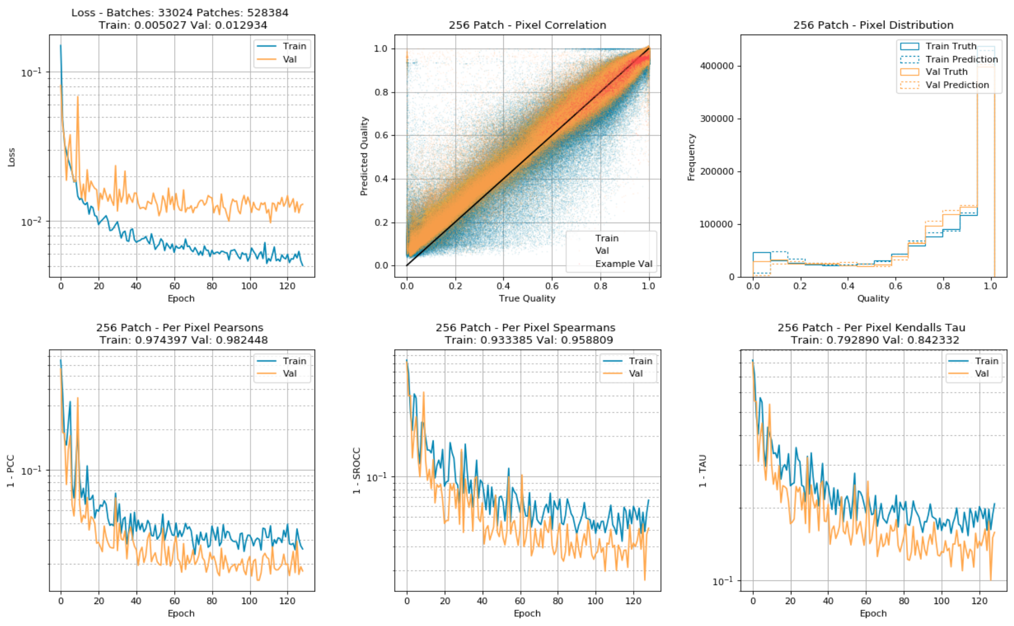
(b) Validation Scene: Sponza

Fig. A.6.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_1$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.

A.4.2 Training and Validation Curves for $\mathcal{L}_2$ Loss

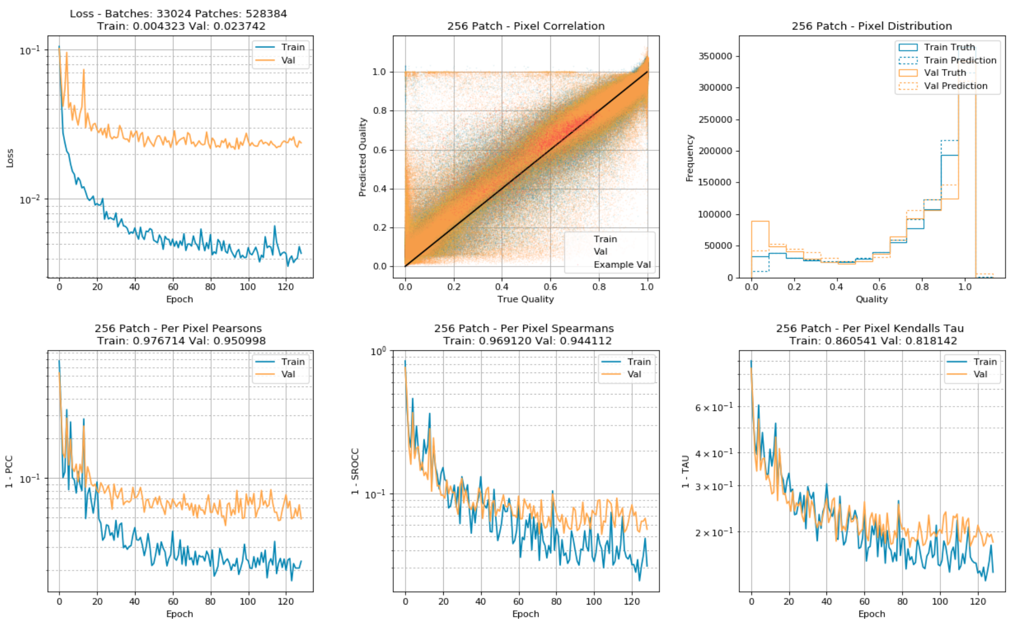


(a) Validation Scene: Cornell Box

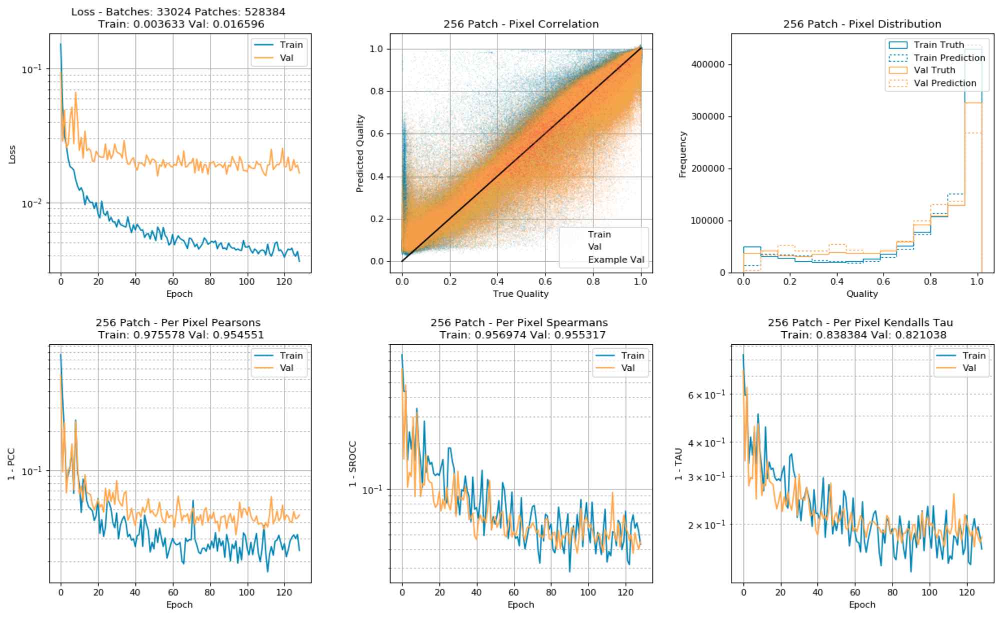


(b) Validation Scene: Veach Bidir

Fig. A.7.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_2$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.



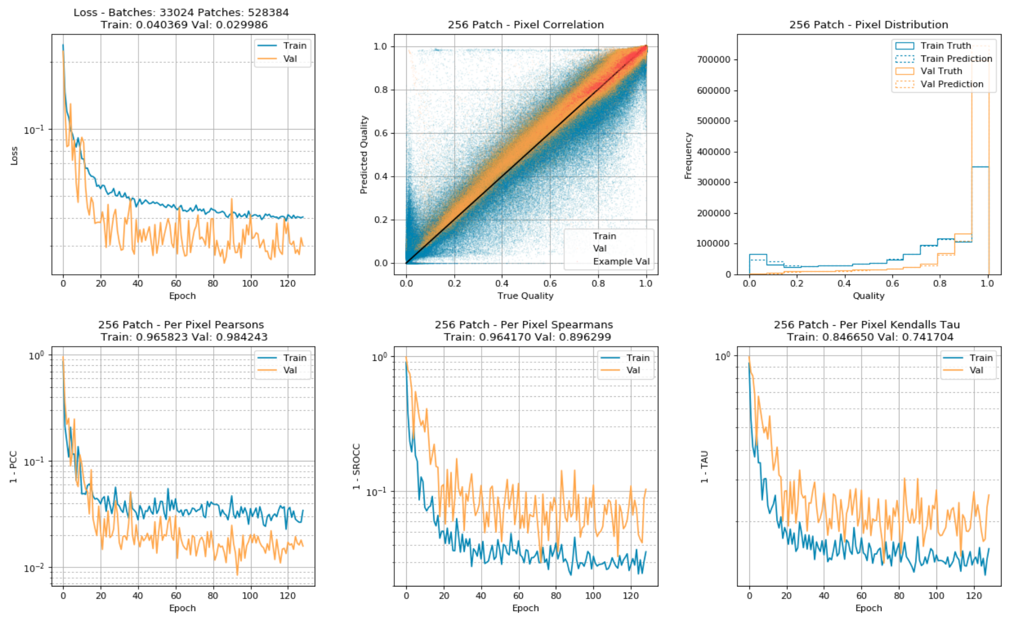
(a) Validation Scene: Veach Door



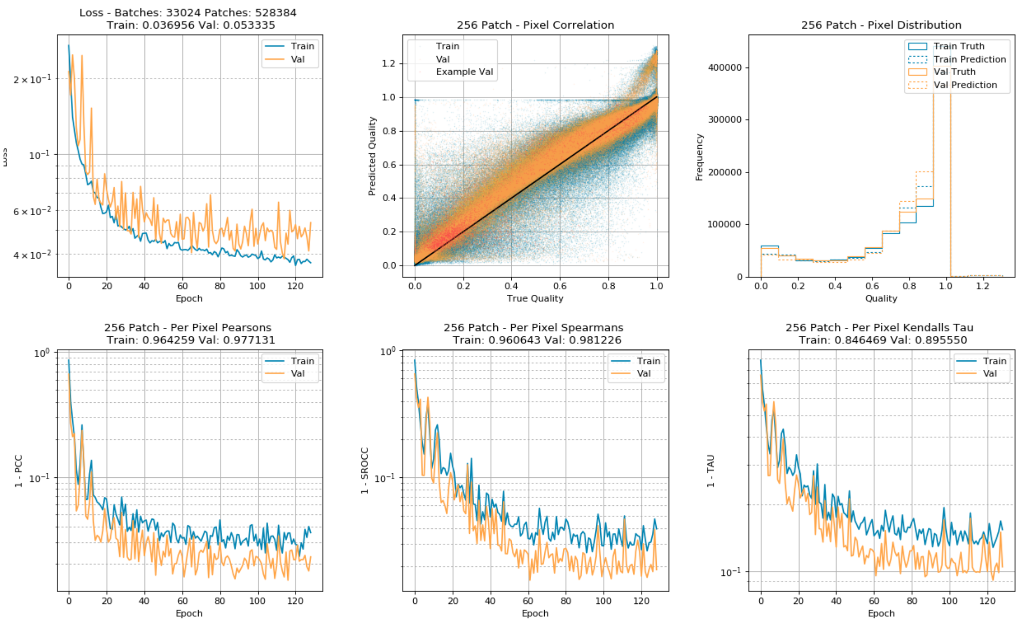
(b) Validation Scene: Sponza

Fig. A.8.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_2$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.

A.4.3 Training and Validation Curves for $\mathcal{L}_C$ Loss

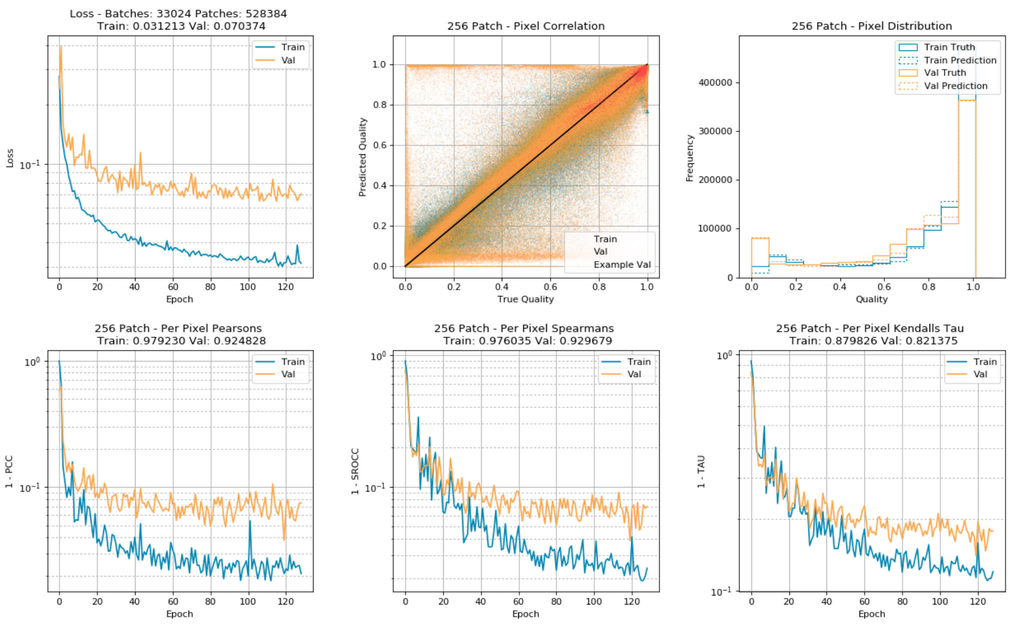


(a) Validation Scene: Cornell Box

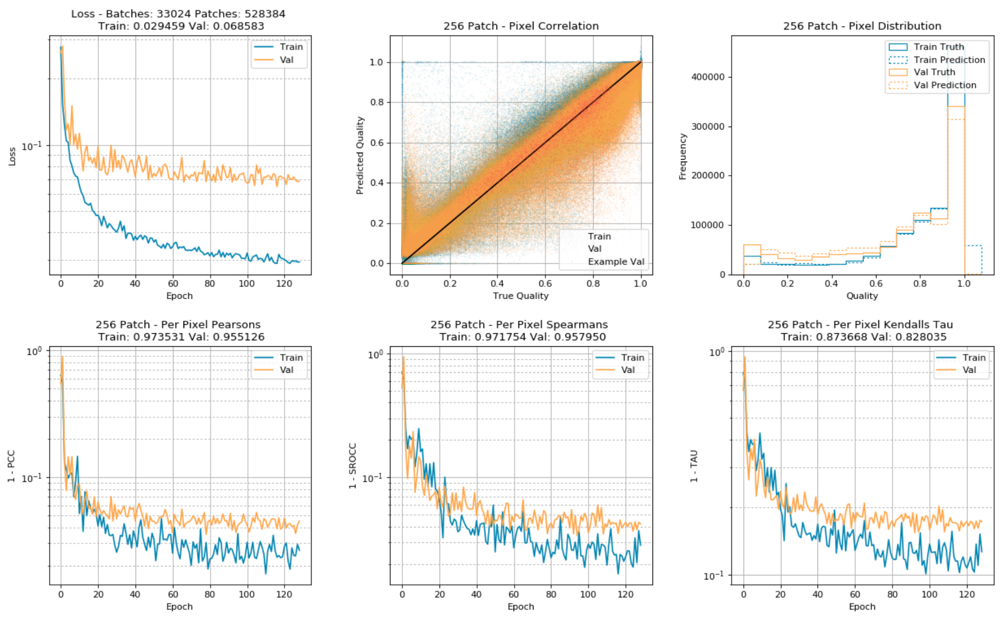


(b) Validation Scene: Veach Bidir

Fig. A.9.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_C$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.



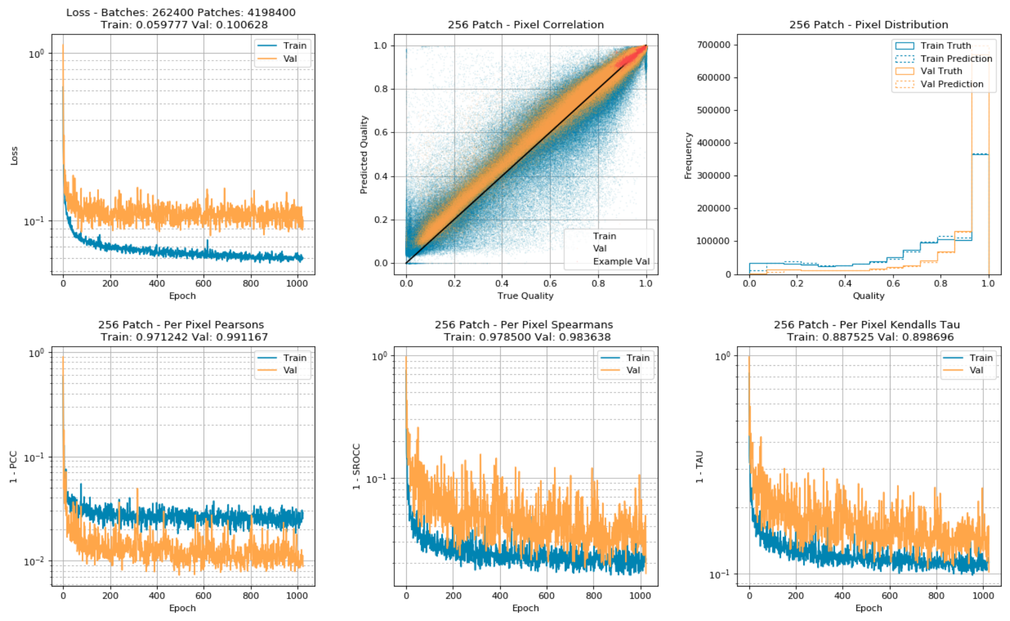
(a) Validation Scene: Veach Door



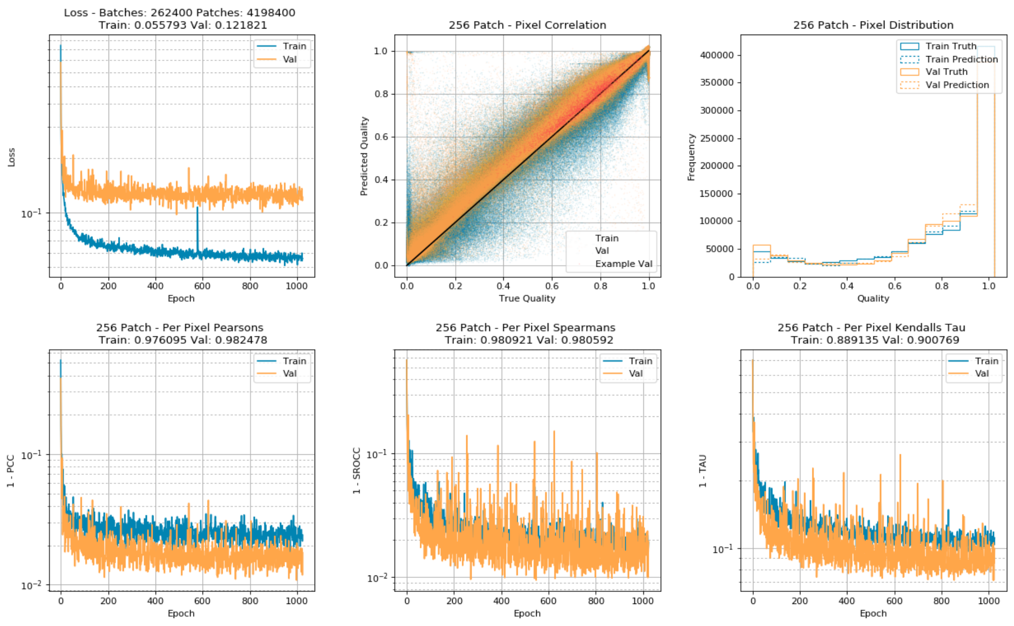
(b) Validation Scene: Sponza

Fig. A.10.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\mathcal{C}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 128 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 128 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.

A.4.4 Training and Validation Curves for $\mathcal{L}_{\text{Joint}}$ Loss

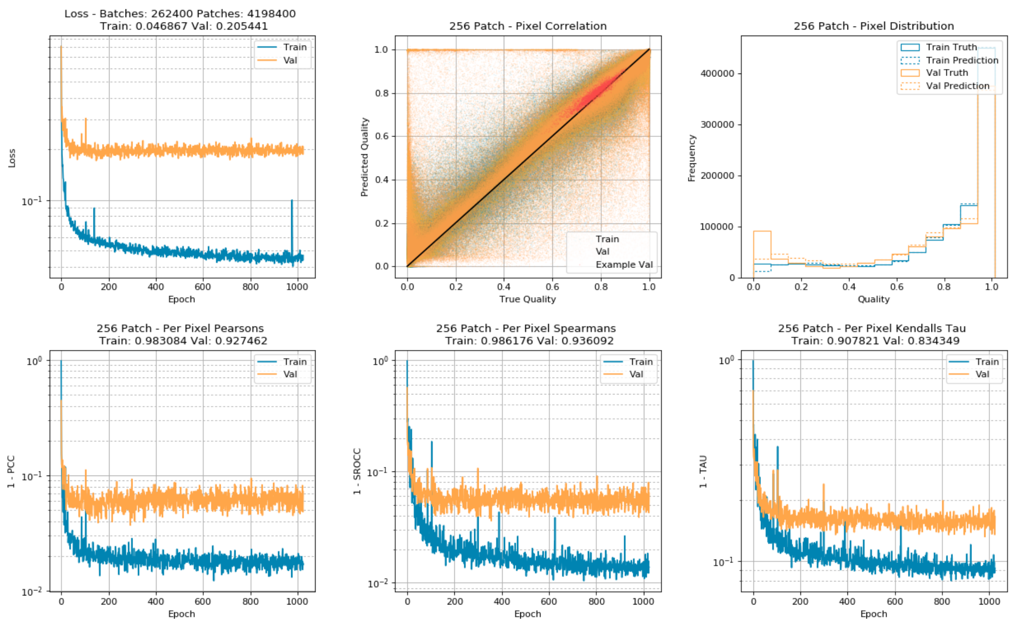


(a) Validation Scene: Cornell Box

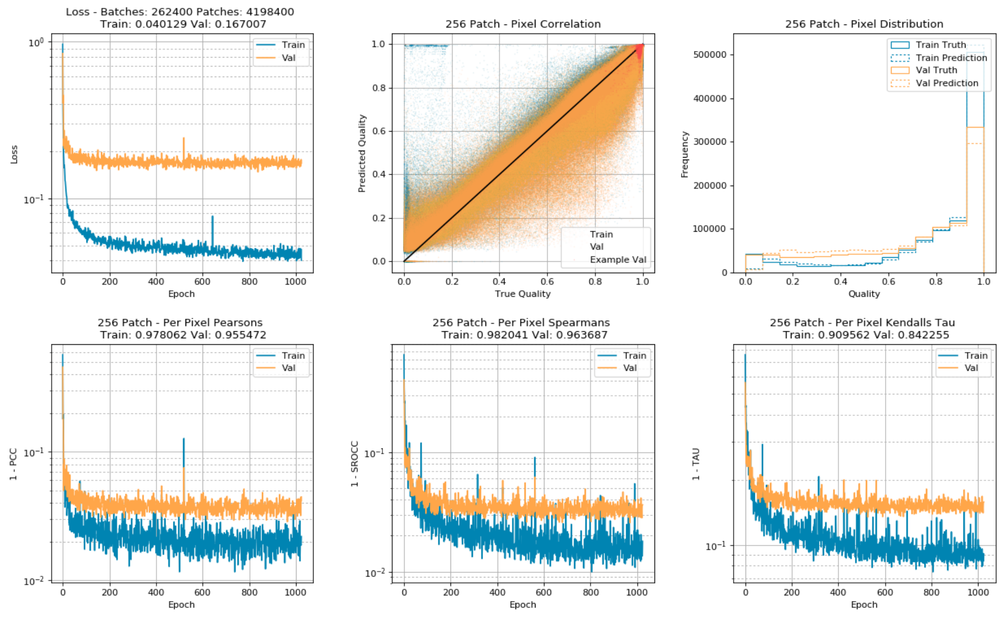


(b) Validation Scene: Veach Bidir

Fig. A.11.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Cornell Box and Veach Bidir scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 1024 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 1024 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.



(a) Validation Scene: Veach Door



(b) Validation Scene: Sponza

Fig. A.12.: Experiment II: Training and Validation set accuracies curves over each epoch for **leave one scene out** cross validation, for Veach Door and Sponza scenes, using **rotation, flip, and HSV jitter** with the $\mathcal{L}_{\text{Joint}}$ loss. The plots show: Loss value over training and validation batches within an epoch (top-left), a scatter plot of correlation between predicted and true quality for each pixel within each patch of the most recent training and validation batch at the end of 1024 epochs (top-centre), overlaid histograms of predicted and true quality distributions for each pixel in each patch of the most recent batches at the end of 1024 epochs (top-right), and the PCC (bottom-left), SROCC (bottom-centre), and Kendall's Tau (bottom-right) for the pixels in the last batch of each epoch during training.

B List of Acronyms

Monte Carlo Sampling

MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
IS	Importance Sampling
MIS	Multiple Importance Sampling
ARS	Accept Reject Sampling
MHMC	Metropolis-Hastings Monte Carlo
HMC	Hamiltonian Monte Carlo
HHMC	Hessian Hamiltonian Monte Carlo
i.i.d.	independent and identically distributed
s.p.p.	samples per pixel
PDF	Probability Density Function

Physically Based Rendering

PT	Path Tracing
BDPT	Bidirectional Path Tracing
MLT	Metropolis Light Transport
PSSMLT	Primary Sample Space Metropolis Light Transport
MMLT	Multiplexed Metropolis Light Transport
ERPT	Energy Redistribution Path Tracing
G-PT	Gradient Domain Path Tracing
G-BDPT	Gradient Domain Bidirectional Path Tracing
G-MLT	Gradient Domain Metropolis Light Transport
BSDF	Bidirectional Scattering Distribution Function
BRDF	Bidirectional Reflectance Distribution Function
BTDF	Bidirectional Transmittance Distribution Function
BSSRDF	Bidirectional Sub-Surface Scattering Reflectance Distribution Function

High Performance Computing

HPC	High Performance Computing

MPI	Message Passing Interface
MEL	MPI Extension Library
OO	Object Orientated
OpenMP	Open Multi-Processing
Pthreads	POSIX Threads
OpenCL	Open Computing Language
CUDA	Compute Unified Device Architecture

Image Quality Assessment

IQA	Image Quality Assessment
FR-IQA	Full Reference Image Quality Assessment
RR-IQA	Reduced Reference Image Quality Assessment
NR-IQA	No Reference Image Quality Assessment
HVS	Human Visual System
MGA	Multi-scale Geometric Analysis
JND	Just Noticeable Difference
GT	Ground Truth
MOS	Mean Opinion Score
DMOS	Differential Mean Opinion Score
LDR	Low Dynamic Range
HDR	High Dynamic Range
PCC	Pearson's Correlation Coefficient
SROCC	Spearman's Rank Order Correlation Coefficient
Kendall's Tau	Kendall's Tau Rank Order Correlation Coefficient

List of Figures

C

List of Tables