Quantum Ray Marching: Reformulating Light Transport for Quantum Computers

by

Logan Mosier

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2023

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The use of quantum computers in computer graphics has gained some interest in recent years, especially for the application of rendering. The current state of the art in quantum rendering relies on Grover's search for finding ray intersections in $O(\sqrt{M})$ for M primitives, which is faster than the naive approach of $O(M)$ but slower than $O(\log M)$ of modern ray tracing with an acceleration data structure. Furthermore, this quantum ray tracing method is fundamentally limited to casting one ray at a time, making it less attractive even when quantum computers become much more mature in the future. We present a new quantum rendering method, quantum ray marching, based on the reformulation of ray marching as a quantum random walk. Our work is the first to provide a complete quantum rendering pipeline capable of supporting light transport simulation and remains fundamentally faster than non-quantum counterparts. Our quantum ray marching can trace an exponential number of rays in polynomial cost and leverage quantum numerical integration to converge in $O(1/N)$ for N estimates. These unique properties make our method asymptotically faster than Monte Carlo ray tracing on non-quantum computers for the first time. We numerically verify the proposed quantum algorithm by rendering both $2D$ and $3D$ scenes.

Acknowledgements

Firstly, I am extremely grateful to my supervisor, Toshiya Hachisuka for all his help, advice, and patience over the last 2 years. I would also like to thank Morgan McGuire for his help on the project that forms the basis of this thesis. Without their insights this work would not be what it is today.

I am also grateful to my girlfriend for all her proofreading, listening to me ramble about quantum computers and always being there for me.

Finally, thanks to my parents for their love and support.

Table of Contents

List of Figures

Chapter 1

Introduction

Though quantum computers are still very much in their infancy, work has already started looking for potential practical applications they may one day serve. Due to the unique nature of quantum computers, new algorithms are available to solve problems that were not possible on classical computers. These new algorithms rely on the properties like superposition to gain an advantage and run algorithmically faster than the current state of the art on classical computers.

One such problem that can benefit from the quantum properties of quantum computers is numerical integration. Quantum computers have been shown to gain an improvement over their classical counterparts [14, 32]. Numerical integration on a quantum computer or, quantum numerical integration, is an attractive alternative to Monte Carlo (MC) integration. This is because the answer converges to the correct answer (the root-mean-square error decreases) in $O(1/N)$ compared to the classical rate of $O(1/\sqrt{N})$ for N samples.

Another unique property of quantum numerical integration is that its error is *independent* from the variance of the integrand. This difference makes quantum numerical integration fundamentally different from MC integration, where the error is proportional to the square root of the integrand.

A popular application of numerical integration is realistic image synthesis. This method of image synthesis, typically called physically based rendering, is done by simulating the paths that light takes through a $3D$-scene. [27] Calculating how these light paths bounce around the scene and ultimately lead to the color that is seen in the final image involves solving a very high dimensional integral. As there is no analytical solution to this problem, we must use numerical methods. Unfortunately, due to the dimensionality of the problem it does not scale well with numerical methods that would typically be applied, like quadrature

methods, and thus we are left with using Monet Carlo integration which is not dependent on the number of dimensions of the integrand.

Another aspect of rendering that makes evaluating the integral needed for the final image difficult, is the effect of objects blocking light paths in the scene. This can cause high variance in the integrand which will slow the already relatively slow convergence of Monte Carlo integration. For these reasons, physically based rendering is a prime example of a problem that could be sped up using quantum numerical integration

However, to fully utilize quantum numerical integration for light transport simulation, you would need a method of sampling the light paths in the scene. This is typically done via a method called ray tracing, which involves casting rays into the scene to simulate how light propagates. Currently, ray tracing on quantum computers, which we will refer to as *quantum ray tracing*, is said to be realizable based on Grover's search algorithm [20, 3, 30]. Recently, Lu and Lin [24] pointed out a potential strength of light transport simulation on quantum computers, which we call *quantum light transport simulation*: it can branch a light transport path *exponentially* with a *polynomial cost* by utilizing the exponential nature of qubits versus bits. This property has also been utilized in quantum numerical integration [14, 32] where it assumes that the integrand can be evaluated against *all the possible inputs* (e.g., an exponential number of branching paths) with polynomial cost. It is in contrast to classical MC light transport simulation where it *stochastically* selects a single path to avoid this exponential branching.

While all those properties sound attractive, we identified two fundamental issues in this current formalism of quantum light transport simulation via quantum ray tracing. The first issue is that each quantum ray tracing operation based on Grover's search costs $O(\sqrt{M})$ for M primitives. While this is faster than a naive classical approach of $O(M)$, it is still asymptotically slower than the $O(\log(M))$ of ray tracing with a tree data structure on classical (i.e., non-quantum) computers [38]. With this asymptotic performance gap, even when quantum computers become as stable and fast as non-quantum computers in the future, using quantum ray tracing over classical ray tracing will never be attractive.

The second issue is that the assumption that an exponential number of light transport paths can be computed at a polynomial cost, with this particular quantum ray tracing algorithm with Grover's search, is in fact *incorrect*. The main theoretical issue is that no quantum algorithm so far can run an exponential number of Grover's searches in a polynomial time [15]. While qubits might be able to store an exponential number of search results at a polynomial *storage cost*, finding an exponential number of search results with Grover's search will still take an exponential *computation time*. Therefore, quantum light transport simulation with Grover's search will not bring any benefit over non-quantum MC

light transport simulation.

We propose a new formulation of quantum light transport simulation with a new *quantum ray marching* algorithm to address these issues. Unlike the existing quantum ray tracing with Grover's search, our quantum ray marching employs ray marching. On the first issue of scaling against classical algorithms, our quantum ray marching scales equivalently to classical ray marching for the number of voxels. While this property alone is neither good nor bad, quantum ray marching simultaneously solves the second issue and is capable of handling an exponential number of light transport paths in a polynomial time. In other words, our quantum ray marching is *exponentially faster* than classical ray marching or quantum ray tracing with Grover's search for tracing an exponential number of light transport paths. Last but not least, being a fully quantum approach, our approach can benefit from the faster convergence $O(1/N)$ of quantum numerical integration. Our work is thus the first to provide a full picture to implement quantum light transport simulation with its fundamental advantages maintained over classical approaches. We demonstrate the properties of our approach via both theoretical and numerical results.

1.1 Contributions

Our contributions are:

- Formulation of ray marching on quantum computers.
- Quantum light transport simulation algorithm based on quantum ray marching with exponential branching.
- First full pipeline of quantum light transport simulation that is asymptotically faster than MC light transport simulation.

1.2 Organization

Chapter 2 presents the necessary background from the field of computer graphics. The chapter starts with the basics of optics and light transport under geometric optics. Then the rendering equation is presented and the basics of Monte Carlo integration are introduced as a means of numerically solving said equation to generate photorealistic images. Finally, the method of ray marching for finding the intersection point between a ray and a surface as an alternative to ray tracing is covered.

Chapter 3 covers the basics of quantum computing needed for this work. First, the quantum circuit model will be presented covering the basics of qubits and quantum gates. Building upon the basics of quantum computing, the needed quantum algorithms that are of interest for quantum graphics are covered. Finally, the chapter will look at how these quantum algorithms are being applied in the current state of the art in the field of computer graphics.

Chapter 4 introduces our new framework for quantum light transport. Starting from ray marching it shows how to produce an oracle circuit using a framework of quantum walks to sample exponential light paths. This oracle circuit can then be used in conjunction with quantum numerical integration to render an image.

Chapter 5 shows the results of our method applied in several problem settings to demonstrate our method. We test the method in both $2D$ and $3D$ scenes. We also test the convergence of our method to verify that our result benefits from the improved convergence of quantum numerical integration.

Chapter 6 extends the result presented in the previous chapters to more arbitrary materials. This indicates that the method is not limited to simple materials and scenes but can instead be used on a much wider range of scenes.

Parts of the work present in this thesis are based on:

Logan Mosier, Morgan McGuire, Toshiya Hachisuka. "Quantum Ray Marching for Reformulating Light Transport Simulation", SIGGRAPH Asia, 2023

Chapter 2

Physically Based Rendering

2.1 Models of Optics

There are three models in the field of optics seeking to explain the nature of light and how it interacts with the rest of the world. The difference between the models is how exact they seek to model the nature of light. The most complete model of optics is known as quantum optics. As the name would imply, this level of optics seeks to capture the quantum behavior of light. In quantum optics, light is modeled as individual quantized photons [26]. Though it is the most accurate model of light, most behaviors of light that are of interest for the field of computer graphics are captured by more simplistic models. Simulating light at this level of accuracy is impractical. Modeling scenes at the required scale would be difficult, and the render times would be far too long, compared to other methods. Thus, while quantum optics remains a fascinating theoretical framework for studying light, it is currently beyond the scope of practical rendering applications

Physical optics is a more classical model of light compared to quantum optics, modeling light as waves. When simulating physical optics, each wave length must be treated differently as they interact with the materials in a scene. Not only that, but the waves of light can also interfere with each other in constructive and destructive ways [33]. Simulating this behavior is important in accurately simulating light transport and some effects like the shimmer of a CD surface, light dispersion through prisms or polarization require this level of detail to be accurately simulated. Much like quantum optics, the added complexity, and longer render times that come with simulating light as a wave often out weigh the ability to properly render these physical effects. There has been recent work on wave optic

rendering that brings it into the realm of practicality but most renderers do not use wave optics [33].

The most prevalent and widely adopted model for rendering is geometric optics. Geometric optics offers a balance between simplicity and realism, modeling light as individual rays. The two main assumptions that allow for geometric optics to be simulated more easily than the previous models are that rays travel straight through the scene, until they are scattered by a solid surface or participating medium, and that light travels instantly through the scene. Fortunately, these assumptions do not prevent us from achieving highly accurate renderings as the assumptions hold for scenes of typical interest and scale. As a result, geometric optics remains the go-to choice for rendering realistic images due to its ability to capture a significant portion of the visual effects required without excessive computational overhead.

2.2 Light Propagation

Even though light is modeled as rays under geometric optics, for the purpose of simulating the propagation of light it is useful to think of it as a single photon traveling through the scene along the ray. Light starts from an emissive surface. The amount of light emitted from a surface is defined by the function $L_e(x, \omega_o)$ where x is the position in the scene and ω_o is the direction that light is emitted in. Once light has been emitted into the scene it travels in a straight line from x in the direction of ω_o until it interacts with a medium or surface in the scene. Typically, it is assumed that the scene is a vacuum as the atmosphere has no noticeable effects on the scale that we are rendering. In this case, the surfaces are the only thing that light interacts with.

Once the light hits a surface it will be either absorbed or scattered in a different direction. Both the material and geometric properties of the surface at the given point define the behavior of the light. There are many models of materials but the standard properties that are of interset are the albedo, and bidirectional scattering distribution function (BSDF), ρ_x. The albedo of a material is the probability that light will be scattered instead of reflected. This is what defines the base color of the material as the reflected light is what is viewed by an observer. Though it is not based on physical reality, this is often defined using a vector of three values defining the amount of red, green, and blue (or RGB) light that is reflected by the material. In reality, this is defined by the wave lengths that are absorbed or reflected, but doing so would require simulating the different wave lengths of light and thus moving towards physical optics. The albedo of the material defines if the light is scattered but the BSDF defines the scattering behavior of the material. The

BSDF represents the nanoscale details of the surface that affect how the light is scattered as modeling the scene at this scale would be impractical. A BSDF may be defined as:

$$\rho(x, \omega_i, \omega_o) = \frac{dL_r(x, \omega_o)}{L_i(x, \omega_i} |n_x \cdot \omega_i| d\omega_i \tag{2.1}$$

and is the ratio of light that came from direction ω_i and is reflected out in ω_o. This ratio defines a probability distribution function, PDF, that we can then use to sample a new direction from and continue propagating the light in that direction.

To be physically accurate, the BSDF must follow two properties. The first is that the BSDF must be reciprocal,

$$\rho(x, \omega_i, \omega_o) = \rho(x, \omega_o, \omega_i)\,, \tag{2.2}$$

the important thing to note is that this means that the BSDF is invariant to the direction that the light is moving. This is a critical property allowing images to be efficiently rendered [33], as we will see in the next section.

The other property is that the BSDF must conserve energy,

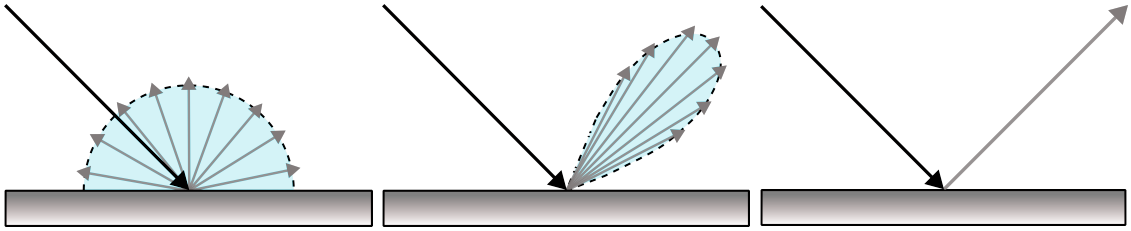
$$\int_\Omega \rho(x, \omega_i, \omega_o) |n_x \cdot \omega_o| d\omega_o \leq 1\,, \tag{2.3}$$

this insures that the BSDF does not add energy to the scene.

BSDFs are often grouped into different categories based on how they scatter the light. These categories are diffuse, glossy, and specular. Diffuse BSDFs are perfectly matte surfaces that scatter light in many directions over the hemisphere. Ideal diffuse surfaces are Lambertian and appear the same no matter the viewing direction. Specular BSDFs are the opposite of diffuse in that they are mirror-like and reflect light back at the same angle over the normal. In between diffuse and specular there are glossy BSDFs which reflect light in a cone. Ideal versions of these materials are not often found in the real world, but they make good building blocks for making more complicated materials.

2.3 Physically Based Rendering

The problem of physically based rendering seeks to produce realistic looking images of virtual scenes. This is done in a similar idea to how a camera makes a realistic image of a real scene. Cameras function by capturing and recording the radiance that hits their film or sensor. In much the same way, a virtual camera is defined in the scene and the



(a) Diffuse (b) Glossy (c) Specular

Figure 2.1: Three models of BSDFs. **a)** is a diffuse surface and scatters light uniformly over the hemisphere. **b)** is a glossy BSDF and scatters the incoming light in a cone out from the surface. **c)** is a specular BSDF and reflects the light with no scattering

radiance at each pixel of the sensor must be captured. The standard approach to solve for the radiance hitting a point on the sensor is to use geometric optics and the rendering equation presented by Kajiya [18]. This equation defines the radiance that is leaving a point x in the direction of ω_o.

$$L_o(x, \omega_o) = L_e(x, \omega_o) + \int_\Omega \rho_x(x, \omega_i, \omega_o) L_i(x, \omega_i)(\omega_i \cdot n) d\omega_i \tag{2.4}$$

Equation (2.4) is a simplified version of the rendering equation that assumes a static scene and non-spectral rendering. L_o is the light coming from point x in the direction of ω_o. L_o is equal to the light emitted from x, L_e, plus the incoming light reflected in the direction of ω_o. The reflected light is calculated via an integral over the hemisphere above x with normal n and is equal to the product of the ρ_x, the incoming light L_i, and the cosine term calculated by taking the dot product of the incoming light direction, ω_i and n.

Equation (2.4) needs to be solved at all points seen from our virtual camera. As well, this equation is recursive in nature as the radiance leaving each point depends on the radiance coming in from all directions over the hemisphere. But the radiance coming in depends on the light leaving the points visible in the direction of $-\omega_i$. To solve this, we need to evaluate Equation (2.4) recursively at each point. This recursive nature makes Equation (2.4) a very high dimension integral.

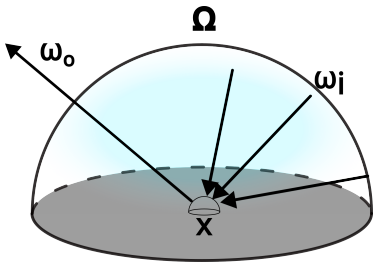


Figure 2.2: Diagram for Equation (2.4) over point X for direction ω_o

2.4 Monte Carlo Integration

Monte Carlo (MC) integration is a stochastic method for solving integration problems in the form of

$$F = \int_{\mathcal{D}} f(x)dx, \tag{2.5}$$

where $\mathcal{D}$ is the domain of integration and $f(x)$ is a scalar function that does not have an analytical expression for its integral F. To accomplish this, MC integration uses random samples from the domain to estimate the integral. For N uniform samples $x_i, \ldots x_N$ the estimation of F is:

$$\bar{F} = \frac{1}{N} \sum_{i}^{N} f(x_i)\,. \tag{2.6}$$

The error of the estimation $\bar{F}$ will converge to F in $O(1/\sqrt{N})$. This means to decrease the error of our estimate by 2 times we will need to take 4 times the number of samples.

There are two properties that make MC integration appealing in many applications. The first is that it is a relatively simple approach to implement compared to alternative methods. All that is needed is the ability to sample the integrand f. The other, arguably more important property is that the error does not scale with the number of dimensions of $\mathcal{D}$. Other deterministic methods, scale exponentially with the number of dimensions in the domain of integration. Because of this, for complex domains with many dimensions, MC integration is often the more appealing choice.

2.5 Monte Carlo Rendering

Rendering a photorealistic image involves solving Equation (2.4). Even in the simplified domain of geometric optics, this is still a very complicated problem. An analytical solution to the equation is impractical for a scene of even mild complexity, especially once the visibility term is considered. Instead, we must use numerical methods to evaluate the integral. This poses an issue as Equation (2.4) is a very high dimensional integral and most numerical methods scale with the number of dimensions. For this reason, MC integration has become the primary method to evaluate the rendering equation.

The most common method for generating samples is known as backwards ray tracing. This involves casting rays from the camera position through a pixel and into the scene. The rays are then propagated through the scene as described in Section 2.2 until they reach a light source to generate a sample path. This saves effort compared to forward ray tracing, where rays are cast from the light sources as it means we only sample paths that will reach the camera. There are methods like bidirectional ray tracing [35] and photon mapping [13] that make use of forward ray tracing as they allow for certain effects like caustics to be captured more easily. We can then use MC integration to evaluate the sampled light paths to generate an estimate for the pixel. This process is continued for each pixel until the image is complete.

2.6 Variance Reduction

As well as being a very high dimensional integral, Equation (2.4) can have very high variance due to the visibility term. The error of MC integration will converge in $O(1/\sqrt{N})$ samples, but this is in the limit as N goes to infinity. As sampling light paths is computationally expensive, we would like to ideally take as few samples as possible while still keeping the error in our estimation to a reasonable level. The mean squared error for an estimator is equal to its standard deviation [36]. For the MC estimator, Equation (2.6), the standard deviation can be shown to be:

$$\sigma[F_N] = \frac{1}{\sqrt{N}}\sigma f(x)\,. \tag{2.7}$$

For $N < \infty$ the standard deviation of the integrand, and by extension the variance, effects the error of our estimator. It can be shown that MC integration will eventually converge to the correct value with enough samples regardless of the variance, but this does motivate the idea behind variance reduction.

Since the error is dependent on the variance, if we can lower the variance, then we can achieve lower error with a similar number of samples, or alternatively, depending on our goal, the same error with fewer samples. A common technique for variance reduction is known as importance sampling. The idea behind importance sampling is that not every sample from the domain will be as *important* as other samples. With this in mind, we can try to distribute our samples in the areas with more importance, that will contribute more to the final value of the integral. This requires the modification of the MC estimator, defined in Equation (2.6), to still maintain an unbiased estimator. If instead of uniformly sampling we draw our samples proportionally to some function $p(x)$ we get the estimator

$$\bar{F} = \frac{1}{N} \sum_{i}^{N} \frac{f(x_i)}{p(x_i)} \,. \tag{2.8}$$

By choosing $p(x)$ well, we can lower the variance of our estimate and thus reduce the error. As an arguably hand-wavy proof of how this works, let us assume that we choose $p(x)$ such that $p(x) \propto f(x)$. This would be the ideal case but would require us to already know f over the domain of interest and thus would defeat the purpose of using importance sampling. Nonetheless, we have $p(x) = cf(x) = \frac{1}{\int f(x)dx}$. Then

$$\frac{f(x_i)}{p(x_i)} = \frac{1}{c} = \int f(x)dx \,, \tag{2.9}$$

would create an estimator that has zero variance. Unfortunately, this is not possible but using our knowledge of the domain we can instead choose a $p(x)$ that is close to the ideal choice in a hope that the variance is decreased.

For rendering there are many ways to do this, but generally the approaches are based on sampling proportionally to the easily calculable parts of Equation (2.4). A common form of this is sampling proportionally to the BSDF. Another method is to sample in the direction of the light sources. Nether of these methods are better than the other in every case, and it may not be obvious which of these methods will be better suited based on the scene. Instead of choosing one method or the other, the work of [36] introduced multiple importance sampling (MIS), which enables gaining the benefits of both methods.

MIS requires that we yet again change the MC estimator. If we have two sampling strategies $p_a(x)$ and $p_b(x)$ then our estimator becomes

$$\bar{F} = \frac{1}{N_a} \sum_{i}^{N_a} \frac{f(x_i) w_a(x_i)}{p_a(x_i)} + \frac{1}{N_b} \sum_{i}^{N_b} \frac{f(x_i) w_b(x_i)}{p_b(x_i)} \,. \tag{2.10}$$

The new MC estimator is thus the sum of the two MC estimators with the addition of the weighting function $w_s(x_i)$. This term properly weights the samples so that they may be combined without the variances adding together. $w_s(x_i)$ weights the function by all the ways that the sample could have been generated using all the sampling strategies. A common choice for w_s is the balance heuristic.

$$w_s(x) = \frac{N_s p_s(x)}{\sum_i N_i p_i(x)} \tag{2.11}$$

2.7 Ray Marching

Ray marching is a method for determining the intersection point of a ray against some scene. Unlike ray tracing, which solves for the intersection point of a ray with the object in the scene analytically, ray marching takes an iterative approach. To enable ray marching we need to be able to define a boolean function, $S(x)$, that will define if a given point x is inside an object. This function can take many forms, but as long as we can define such a function we will be able to carry out ray marching. The goal of ray marching is to find the first x in the direction of the ray such that $S(x)$ is true.

$$S(x) = \begin{cases} 1 & \text{if } x \text{ is on the boundary of an object} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

Ray marching functions by taking many steps through the scene to *march* the ray forward. After a step is taken $S(x)$ can be evaluated to check if a surface has been found. If a surface has not been found then another step may be taken. This process continues until the surface is found or another halting criteria, like a max number of steps, has been reached.

The benefit of ray marching is that it allows for intersections to be found between rays and otherwise difficult objects where an analytical solution would be infeasible.

2.7.1 Sphere Marching

The main issue that comes from using ray marching is setting the step size. Setting the step size too low causes the ray to waste many steps in empty spaces far away from any intersection points. Setting the step size too large means that there is a high chance that

you many miss finer details in the scene by stepping over them. Ray marching is rarely used in this naive form because of this trade off.

Sphere marching is an extension of ray marching that seeks to address this issue [12]. Instead of only taking fixed sized steps, the size of the step is varied based on the distance to the surface. As it relies on being able to easily evaluate the distance from any point in the scene to the surface, sphere marching is most commonly used in combination with a signed distance field (SDF). An SDF represents an implicit surface by either storing or in some way representing the the distance from each point to the closest point on the scene. More formally, an SDF is:

$$s(x) = \begin{cases} d(x, \partial\Omega) & \text{if } x \in \Omega \\ -d(x, \partial\Omega) & \text{if } x \in \Omega^c \end{cases} . \tag{2.13}$$

The SDF, $s(x)$, is a function that maps points x in the domain to the distance to the boundary of Ω, $\partial\Omega$, using some distance metric d. As well, if x is inside of Ω then the negative of the distance is returned. This is the reason that it is referred to as a *signed* distance field. Using an SDF, sphere marching is able to take larger steps in empty spaces while not having to worry about stepping over a surface intersection when close to the surfaces in the scene, even when very fine details are present. Though it solves many of the issues with simple ray marching, sphere marching still can take many small steps when a ray passes close to a surface without intersecting it.

Chapter 3

Quantum Computing

This chapter provides a brief overview of quantum computing to introduce basic concepts and symbols before covering some of the more advanced topics that are of interest for this thesis and the wider field of computer graphics. Though we hope to cover the topics in enough detail so that any reader should be able to understand the work presented, this will only be a high-level introduction to the topics of quantum computing. Readers who are interested in gaining a deeper and more comprehensive understanding of quantum computing may refer to Johnston et al. [15].

3.1 Basics of Quantum Computing

3.1.1 Qubits

The qubit is the quantum equivalent of the bit in classical computing [31]. Unlike a bit, which can only be a 0 or 1, a qubit is represented by a complex vector, known as a state vector. Each entry in this vector represents the amplitude of the corresponding state. The equivalent of 0 and 1 for a qubit are

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \tag{3.1}$$

for $|0\rangle$ and $|1\rangle$. This is *bra-ket* notation and is common in quantum computing literature. A ket, $|\rangle$, represents a complex column vector and is used to represent the state vector of a quantum system. The bra,$\langle|$, is the conjugate transpose of the equivalent ket.

Since the state of a qubit is a vector, we can represent a mixture of states known as a *superposition*. A superposition of the $|0\rangle$ and $|1\rangle$ states can be written in a general form as

$$|\psi\rangle = \sqrt{1-\alpha^2}\,|0\rangle + \alpha\,|1\rangle \quad (\alpha \in [0,1]) \tag{3.2}$$

where α is the amplitude of the $|1\rangle$ state. This ability to have a superposition of states is one of the unique properties that enables the new algorithms for quantum computers. The superposition will last until it collapses into one of the possible states. This is done when the qubit is measured. The amplitude α is a complex number, but for our purposes, α can be simply thought of as the square root of the probability that a given state will be measured. As such, the norm of a state vector must be 1. When working with n qubits the state vector will have 2^n entries.

A single qubit is not sufficient for doing much useful work. When working with a register of multiple qubits, the state of the whole system can be represented as the tensor product of the individual qubit states, i.e. $|0\rangle \otimes |1\rangle$. To simplify notation, the operator is often dropped and the state written as $|0\rangle\,|1\rangle$. This means that the state vector of a system with n qubits will have 2^n values. However, it is still required that the norm of this vector be one.

We will utilize two approaches for representing values using qubits [40]. A binary value may be encoded in a register of qubits in the same way as on a classical computer, this is referred to as basis encoding [40]. When data is encoded in this way, the quantum state used will be denoted as $|v\rangle$. The nature of a qubit means that data can be encoded in more than just the traditional way. One of the alternative methods of encoding data is amplitude encoding. This method uses the amplitude of a state to store a value. The benefit of this method is that it allows storing 2^n values using only n qubits. Since the values are stored in the amplitudes of the different states, they cannot be arbitrarily chosen. They must be normalized before they can be used.

3.1.2 Hilbert Space and Bloch Sphere

The state of a quantum system is defined regarding a Hilbert space. A Hilbert space, by definition, is an infinite dimension complex space with an inner product. Nowadays, that definition is often loosened to include finite dimensional spaces. This is particularly useful for quantum computing, as we will often work within Hilbert spaces of 2^n dimension, where n is the number of qubits in our quantum system. To visualize the state of the system, a common method is the Bloch sphere. The Bloch sphere, as seen in Figure 3.1, is a geometric way to represent the current state of a quantum system. All pure states in the

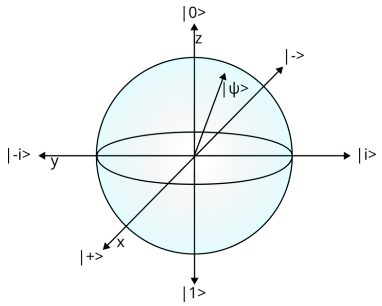


Figure 3.1: Bloch sphere. The $|0\rangle$ and $|1\rangle$ define the poles of the z axis. The $|+\rangle = [1, 1]^T$ and $|-\rangle = [1, -1]^T$ states on the x axis are the result of applying

quantum system lie on the surface of a Bloch sphere. Many of the operations that can be applied to a quantum system can be intuitively thought of as rotations around the Bloch sphere.

3.1.3 Gates

There are several models of quantum computing and the most prevalent one for our work is the quantum circuit model [8]. In the quantum circuit model, the state of the system is evolved by the application of quantum gates, similarly to how logic gates are applied to bits in non-quantum computers. Quantum gates are defined as matrices. Three gates that will be relevant to our work are the X, $R_y(\theta)$ and H gates. The X or Pauli-X gate is the quantum version of a bit flip, changing $|0\rangle$ to $|1\rangle$ and vice versa. This can also be thought of as a reflection over the X axis of the Bloch sphere. The $R_y(\theta)$ gate, on the other hand, does not have a classical counterpart. This gate applies a rotation around the Y-axis of the Bloch sphere to a qubit. This is a more general version of the Pauli-Y gate, which is equal to $R_y(\pi)$. The H, or Hadamard, gate puts a qubit into an equal superposition between the 1 and 0 states.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \tag{3.3}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.4}$$

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.5}$$

As quantum gates are limited to being unitary operations, we can define any possible single qubit gates in the form:

$$U(\psi, \theta, \alpha, \beta) = e^{\frac{i\psi}{2}i} \begin{bmatrix} e^{i\alpha}\cos(\theta) & e^{i\beta}\sin(\theta) \\ -e^{i\beta}\sin(\theta) & e^{-i\alpha}\cos(\theta) \end{bmatrix} . \tag{3.6}$$

Gates are not limited to acting on a single qubit and can be applied to multiple qubits at once. Controlled gates use such operations. These gates can be thought of as a version of the non-controlled gate but only act on the target qubit when the control qubit is in a desired state. If we have a quantum state that contains two qubits, $|t\rangle$ and $|c\rangle$ and a unitary gate U,

$$U = \begin{bmatrix} u_{00} & u_{01} \\ u_{10} & u_{11} \end{bmatrix} . \tag{3.7}$$

We wish to apply U to $|t\rangle$ but only if $|c\rangle$ is in the one state. Measuring $|c\rangle$ would collapse any potential superposition that the system may be in. Instead, we would need to apply a controlled version of U, U_c:

$$U_c |c\rangle |t\rangle = \begin{cases} |c\rangle \otimes |t\rangle & \text{if } |c\rangle = 0 \\ |c\rangle \otimes U |c\rangle & \text{if } |c\rangle = 1 \\ . \end{cases} \tag{3.8}$$

U_c is constructed as follows:

$$U_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & u_{10} & u_{11} \end{bmatrix} . \tag{3.9}$$

The construction of U_c depends on the order of qubits in the system. If instead $|c\rangle$ was the second qubit in the system, U_c would become

$$U_c = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & u_{00} & 0 & u_{01} \\ 0 & 0 & 1 & 0 \\ 0 & u_{10} & 0 & u_{11} \end{bmatrix} . \tag{3.10}$$

Controlled gates allow for more complex logic to be implemented on a quantum computer without having to measure the state of the quantum system.

There are physical restrictions on the types of gates that can be constructed in practice. In particular, all gates must be physically realizable and must be consistent with the laws of

Figure 3.2: (a) The first controlled gate U_0 only applies when the first qubit is 0. U_1 is similarly only applied when the first qubit is 1. Similarly, U_2 is only applied when the control qubits are 10. (b) A simple circuit diagram the equivalent mathematical representation of the gates applied in the circuit is $U_1(U_0 \otimes I)\,|\phi\rangle$

quantum mechanics, meaning that all gates take the form of unitary matrices [4]. While it may seem like a large restriction on the usefulness of quantum computers, it can be shown that the Toffoli gate, a X gate with two control qubits, is universal [1] (i.e., anything we can do with non-quantum computers can be done on quantum computers).

3.1.4 Quantum Circuit Diagrams

There area many ways to define and share quantum algorithms. A common way to represent quantum algorithms is using circuit diagrams. These are drawn very similarly to regular circuit models and can map directly to the mathematical definitions of the circuit. Each wire in the diagram represents a qubit or register of qubits. The x-axis represents the execution time of the circuit. The gates applied to the qubits are shown as blocks along the wires that they act on. Controlled gates are represented by black dots for the 1 state and white dots for the 0 state with a wire running to the gate they control.

The mapping between the mathematical definition of an algorithm and the circuit diagram is fairly straightforward. Gates that operate in parallel are equivalent between the tensor product of both gates. The identity gate is implicitly applied to any qubit that does not have a gate applied to it at each step. On the other hand, serial gates are the multiplication of the two gates.

Figure 3.3: Diagram for Grover's search / amplitude amplification

3.2 Grover's Search

Grover's search [11] is one of the most well-known quantum algorithms that was shown to solve a problem algorithmically faster than would be possible on classical computers. Grover's search solves the unstructured search problem in $O(\sqrt{N})$ instead of $O(N)$ for a database of N elements. The unstructured search problem is the problem of finding an element in a set of N elements X for which $f(x) = 1$ for some function $f : X \to [0, 1]$. Grover's search, like many quantum algorithms, is a probabilistic algorithm. It functions by creating a quantum state which when measured has a high probability of being in the desired state. This is done using *amplitude amplification* and an *oracle gate* to evolve some initial state into this desired state.

$$Q = (-\mathcal{I}O_0\mathcal{I}^\dagger O_f)^{N_{optimal}}\mathcal{I} \tag{3.11}$$

Equation (3.11) formally defines the full process of Grover's search. $\mathcal{I}$ is the initialization circuit to prepare the quantum state that will be searched over. In the standard example, this is simply a Hadamard gate applied to each qubit to create a uniform superposition but can be more complicated in practice. O_0 is an oracle gate that applies a phase shift of -1 to the 0 base state. $-O_0$ is used in this case to apply the phase shift to all but the 0 state. Finally, O_f is the oracle gate that applies phase shifts to the states that we are searching for:

$$O_f \left|\psi\right\rangle = (-1)^{f(x)} \left|\psi\right\rangle \ . \tag{3.12}$$

Amplitude amplification can be viewed as a rotation operator on a 2D subspace spanned by $|g\rangle$, the good state that we are searching for and $|b\rangle$, all the other states. So initially, the state of the system is $|\psi\rangle = \sqrt{\frac{M}{N}}\,|g\rangle + \sqrt{\frac{N-M}{N}}\,|b\rangle$.

This rotation *amplifies* the amplitude of $|g\rangle$ from states by increasing the amplitude of those states from $\sin(\theta) = \sqrt{\frac{M}{N}}$ to $\sin((2n+1)\theta)$, for n applications of our circuit. This is done without knowing the value of θ.

As the max value of sine is at $\sin(\frac{\pi}{2})$ we get that:

$$\begin{aligned} \frac{\pi}{2} &= (2n+1)\theta \qquad (3.13)\\ \frac{\pi}{2} &= (2n+1)\sin^{-1}(\sqrt{\frac{M}{N}}) \\ \frac{\pi}{4\sin^{-1}(\sqrt{\frac{M}{N}})} - \frac{1}{2} &= n \qquad (M \ll N \implies \sin^{-1}(\sqrt{\frac{M}{N}}) \simeq \sqrt{\frac{M}{N}}) \\ \frac{\pi}{4}\sqrt{\frac{N}{M}} - \frac{1}{2} &= n\,. \end{aligned}$$

Two interesting behaviors that the keen eyed reader may have noticed are that the number of applications of amplitude amplification is dependent on the number of solutions that are present and that the amplitude is a sinusoidal function. These two facts mean that some problems will not be well suited for standard Grover's search. Since in many search problems, like those we will be interested in the next section, the number of good elements is not known.

This means that we would only be able to guess at the number of iterations that would be needed to most optimally be able to read out one of the desired states. This could work, but if we select the wrong number of iterations we may apply amplitude amplification too many times and actually reduce the amplitude of the desired states. Furthermore, in the rare case that the number of desired elements is greater than half, applying $\sqrt{N}$ iterations will actually lower the chance of reading a desired element. To mitigate the issue of using Grover's search with an unknown number of elements, Boyer et al. [5] proposed an algorithm that increases the number of iterations exponentially at each step until a desired element is found. This algorithm can be shown to still scale $O(\sqrt{N})$ and allows for the application of Grover's search for an unknown number of target elements.

3.3 Amplitude Estimation

The problem of amplitude estimation is given a gate A, such that

$$A\,|0\rangle\,|0\rangle = |\psi\rangle = \sqrt{1-\alpha^2}\,|\psi_b\rangle\,|0\rangle + \alpha\,|\psi_g\rangle\,|1\rangle \qquad (3.14)$$

try and estimate the value of α. The naive approach for doing this would be to simply create many versions of $|\psi\rangle$, measure the state and make an MC estimate of α. As covered in Chapter 2 this would give the standard MC error of $O(1/\sqrt{N})$. Unlike classical computing though we can do better than this and can achieve a convergence of $O(1/N)$. The original method presented by Brassard et al. [6] was based on phase estimation and the quantum Fourier transform (QFT). The version that we will be using in the latter chapters is the more recent methods based on amplitude amplification. The reason that the amplitude amplification based versions have become more popular than the phase estimation approaches is that the phase estimation approach requires significantly more qubits and controlled gates than the amplitude amplification approaches.

Amplitude estimation by amplitude amplification functions by first noting that

$$\sqrt{1-\alpha^2}\,|\psi_b\rangle\,|0\rangle + \alpha\,|\psi_g\rangle\,|1\rangle = cos(\theta_\alpha)\,|\psi_b\rangle\,|0\rangle + sin(\theta_\alpha)\,|\psi_g\rangle\,|1\rangle \tag{3.15}$$

and that if we apply amplitude amplification to $|1\rangle$, n times we can scale the amplitude from $sin(\theta_\alpha)$ to $sin((2n+1)\theta_\alpha)$.

Though there are several approaches [25, 10, 32], the base idea is that after creating an initial estimate of θ_α we can establish a confidence interval of $[\theta_{alpha} - \frac{\delta}{2}, \theta_{alpha} + \frac{\delta}{2}]$ for some error δ. We then run the same circuit again, but this time amplitude amplification is used to scale the error range such that the error range is scaled. Next, the new value is estimated by creating and measuring the state many times. This new value will be an estimation of $sin((2n+1)\theta_\alpha)$. As n is known, this new estimation may be used to create an estimation for θ_α by scaling the value with an error of δ^2. How the samples are post-processed to determine the error ranges and the scaling that must be applied differs between the different approaches.

3.4 Quantum Numerical Integration

An application of amplitude estimation is quantum numerical integration. Quantum numerical integration is a new set of algorithms that seeks to leverage the unique properties to gain an algorithmic speed-up over classical techniques. Quantum numerical integration is solving the same problem as MC integration. The added restriction is that $F \in [0, 1]$. If this is not the case $f(x)$ may be scaled so that this condition is met without loss of generality.

In quantum numerical integration, a quantum circuit computes $f(x)$ for input qubits, x, representing all possible values of x given the representation of numbers (e.g., 32-bit

floating-point numbers). Given such a quantum circuit, it is easy to construct another qubit

$$|F\rangle = \sqrt{1 - F^2}\,|0\rangle + F\,|1\rangle\,. \tag{3.16}$$

where it encodes the *correct* integral F up to the limit of the precision of x, with the computation cost equivalent to evaluating $f(x)$ for a single sample in MC integration.

Quantum numerical integration then uses multiple samples to estimate the amplitude F since it is impossible to read F directly from $|F\rangle$ (i.e., instances of $|F\rangle$ must be made and read out once for each to estimate F). Once this state has been created, we can apply amplitude estimation to retrieve the value of F from the state vector.

There are two unique properties of quantum numerical integration. Firstly, it internally encodes F and its estimation of F is *independent* of the variance of $f(x)$. Its accuracy thus depends solely on amplitude estimation of F as demonstrated by prior work [14, 32, 23]. MC integration requires more samples to reach the same error when the integrand $f(x)$ has a larger variance, and variance reduction techniques need to be employed. Such variance reduction is fundamentally unnecessary in quantum numerical integration.

Secondly, quantum numerical integration achieves $O(1/N)$ convergence for N evaluations of F, in contrast to the $O(1/\sqrt{N})$ convergence of MC integration for N evaluations of $f(x)$. Note that they have the same computational complexity despite the differences between F and $f(x)$. Even with quasi-Monte Carlo integration, the best possible convergence is $O((\log N)^d/N)$ for a d dimensional integrand [21]. Quantum numerical integration is thus asymptotically faster than non-quantum approaches.

3.5 Quantum Walks

Random walks are a common algorithmic framework that has been used As the name implies, quantum walks are the quantum analogue to classical random walks. Much like its classical counterpart, the quantum walk is a useful framework for designing all sorts of algorithms.

3.5.1 Types of Quantum Walks

Quantum walks may also be split into two main groups, continuous and discrete walks. The difference conceptually between the two is based on how the walker is thought to be moving. In the case of a continuous walk, the walker is always on the move, at any given

moment the walk may move to any other position that is reachable from its current state with some probability defined by the problem. On the other hand, the discrete walker only moves at fixed intervals. Though the two types of walk are similar conceptually, they are typically implemented in fairly different ways. For this work, we will only be working with discrete time quantum walks, so we will not look at how continuous time walks function. For those curious, Kempe [19] and Venegas-Andraca [37] provide more background on discrete and continuous quantum walks.

3.5.2 Discrete Quantum Walks

The most basic discrete quantum walk is a 1D walk on a line. In this example, our walker starts at the origin and can move either left or right. In a classical walk, this decision would be made using some form of randomness, like a coin flip, to determine which direction to take a step in. Then, once the decision has been made, the walker moves to the new location and makes the same left or right decision again. This process continues until stopping criteria is met. For the quantum walk this will be a predefined number. A quantum walk does not have to make a choice, instead a *Coin* gate is used that creates a superposition of both directions. Then when moving to the next step we end up in a matching superposition.

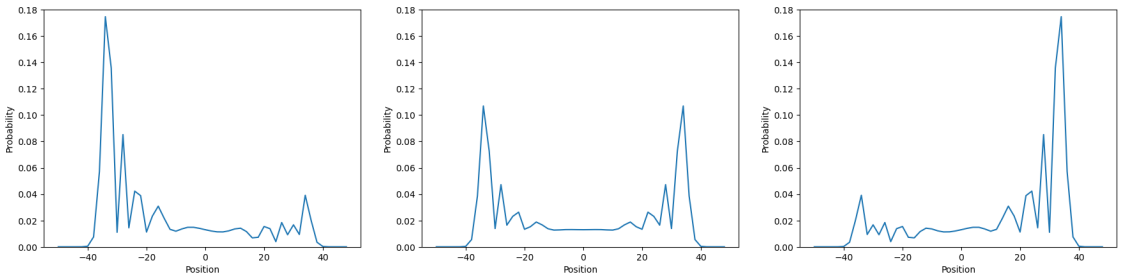
A discrete quantum walk starts from an initialized quantum state with two registers, $|p\rangle$ for the current position and $|d\rangle$ for the direction of the walker and a walk operator U constructed as

$$U = S(C \otimes I)\,. \tag{3.17}$$

The two unitary operators, C and S, are encoded the dynamics of the walk in the circuit. C serves as the coin gate, creating a superposition of the directions which we want to travel in. There are many ways to implement this gate but a common and simple one is the Hadamard coin. The Hadamard coin is simply a Hadamard gate.

$$C = H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3.18}$$

Using a Hadamard gate gives us an equal probability of stepping to the left or the right. If we then measured the state of the system, we would find that, as expected, the system is one step to either the left or the right. Continuing in this way, coin flip, step, measure, repeat would lead to the same behavior as we see in classical walks. If instead we don't measure the system, preserving the superposition, we can see the unique properties of the quantum walk.



(a) (b) (c)

Figure 3.4: The probabilities of 1D random walk on the line after 50 steps with the Hadamard coin. a) and c) are the result of stating with an initial direction of $|0\rangle$ or $|1\rangle$ respectively. This leads to asymmetric walk due to the destructive interface that occurs during the walk. b), on the other hand, is the result of starting the walk with the state of $|0\rangle + i\,|1\rangle$. This prevents destructive interference from occurring during the walk, leading to a symmetric walk in both the left and right directions

As can be seen in Figure 3.4 the walk does not follow the standard probability distribution that we would expect to find with a classical random walk. The walker seems to spread out much faster than is seen classically. This trajectory is referred to as *ballistic* [43]. As well, the initial state of the walker effects the direction of the walk, which is another difference between the classical and quantum walks. This due to the nature of how the Hadamard gate treats the $|0\rangle$ and $|1\rangle$ states differently, this leads to interference between the different states of the walker. In some cases, the states destructively interfere, canceling each other out, leading to this unequal probability.

To achieve a more classical walk behavior where the walker that is just as likely to be found to the left or right from the origin, we need to avoid the destructive interference between the states. There are two options to accomplish this task. This first is that we can initialize the coin to some other initial condition besides $|0\rangle$. For the Hadamard gate, this is the state $|0\rangle + i\,|1\rangle$. The alternative is that we could use a different gate with similar scattering behavior, equal chance of going left or right, but does not lead to the destructive interference. The Y-gate provides us with a symmetric walk without changing the initial coin state.

$$Y = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & i \\ i & 1 \end{bmatrix} \tag{3.19}$$

The unique properties of quantum walks have led to them being used to solve various problems and, as such, is a quite well-studied area of quantum computing [17].

3.6 Quantum Computing in Computer Graphics

3.6.1 Grover's Ray Tracing

Grover's search was introduced to computer graphics by Lanzagorta and Uhlmann [20] as a technique to efficiently solve various problems in computer graphics, including ray tracing. Their ray tracing approach involves formulating the computation of the first intersection between a ray and M primitives as a search problem to identify a single primitive with the first intersection. Recent work by Alves et al. [3] has presented a practical implementation of this approach for simple scenes with orthographic rays. The latest approach along this line by Santos et al. [30] now supports arbitrary rays and Whitted ray tracing [41].

Though the work of Santos et al. [30] is quite impressive, there are several limitations inherent in this method of quantum ray tracing due to the use of Grover's search. The first limitation is the run time of $O(\sqrt{M})$. Though this is faster than the naive approach on classical computers, $O(M)$ it is still slower than other common approaches using acceleration data structure. For example, using a bounding volume hierarchy (BVH) [27] allows ray intersections to be found in only $O(log(M))$. Santos et al. [30] make the argument that their approach is thus better suited to dynamic scenes where the construction of the BVH, which is $O(M)$ would need to be done often. It may be possible that this is improved in the future with a *binary Grover's search*, but no such technique currently exists.

Another major problem of this approach is that it does not accelerate ray tracing of *multiple* rays. Due to the probabilistic nature of Grover's search, many measurements must be made to find the primitive that intersects the ray, and even then, it is not guaranteed that the correct primitive will be found. These measurements collapse the quantum state each time, meaning that there is no hope of leveraging the inherent parallelism that could be gained by using a quantum computer. Though it is slightly unfair to call this a limitation of the method as the goal of the approach is not to handle multiple rays, it still prevents the use of using this ray intersection method as part of any other quantum algorithm.

3.6.2 Quantum Super Sampling

The other quantum algorithm of interest in computer graphics has been quantum numerical integration. This was first introduced by Johnston [14], in the form of quantum supper sampling, as a way to super sample images. An oracle was created that would create a superposition of the subpixels in the image. Then quantum amplitude estimation, based on phase estimation was used to find the average value for the pixel. Later Shimada and

Hachisuka [32] introduced a method that was better suited to the near term quantum computers due to its simpler circuits. They were also the first to point out that rendering would benefit from the use of quantum numerical integration due to its increased convergence rate and the fact that the method is invariant to the variance of the integrand. This is especially promising as the variance can be large due to the discontinuities of the integrand because of the visibility term [27]. Unfortunately, creating an oracle circuit that may be used to evaluate Equation (2.4) is far from trivial.

Chapter 4

Quantum Light Transport

This chapter covers the main contribution of this work, *quantum ray marching*. This is the first work to present a fully working method to evaluate light paths on a quantum computer.

4.1 Overview

We use *ray marching* as a basic building block in quantum light transport simulation. Our algorithm takes a ray and the first intersection along it as input and outputs the resulting radiance that is emitted from the intersection point in the inverse direction of the ray. Figure 4.1 illustrates our quantum ray marching.

Unlike the classical counterpart or even quantum ray tracing with Grover's search, our quantum light transport can process an exponential number of paths in a polynomial storage *and* computation cost because the states of ray marching are all superpositioned as qubits. Our quantum ray marching scales competitively with non-quantum ray marching for M voxels, which makes it a more feasible future option than quantum ray tracing with Grover's search.

Our quantum ray marching works as a quantum evaluation of the integrand $f(x)$ which allows us to take a superposition of all the possible paths x and construct the qubit $|F\rangle$. As a result, our method can handle an exponential number of rays in a polynomial cost (both storage and time) and simultaneously achieves faster $O(1/N)$ convergence for N estimates, both of which fundamentally outperform algorithms on non-quantum computers.

The above properties cannot be achieved by any of the combinations of classical and quantum algorithms at the moment. For example, quantum ray tracing with Grover's search scales *worse* than non-quantum ray tracing, and it scales linearly to the number of rays. One might consider running ray tracing classically, loading the results into qubits, and performing quantum numerical integration to combine both. This hybrid of non-quantum ray tracing and quantum numerical integration achieves better scaling for the number of primitives, though the construction of the qubit $|F\rangle$ for quantum numerical integration takes a prohibitive amount of computation (i.e., need to generate all the possible paths x as an input to a quantum circuit to construct $|F\rangle$). Our approach is the first to provide a full pipeline of quantum light transport simulation that would be fundamentally faster than non-quantum counterparts.

4.2 Ray Marching as a Random Walk

Given a camera configuration and a pixel, we assume that we have already found the first hit point of a ray through the pixel. This initialization step can be done on a non-quantum or quantum computer using the existing methods. Though we could start the walk from the camera, we choose to cast the ray on a classical computer for two reasons. The first is that, since the air in the scene is not a participating media, it will have no effect on the actual value that we hope to sample but will allow the walk to be much shorter as we do not have to take all the initial steps that will not impact the sample. The second reason that we do this is that it separates the initial rays from the effects of the rounding to some extent. If we started the walk from the camera, then we would need to be able to represent all the different rays that are sent through the camera. As the number of pixels in the image increases, the difference between these rays becomes finer, thus many qubits would be needed to differentiate between the rays that go through each pixel. By first casting the ray classically we can side step this problem. The rounding process at the hit point will affect the image, as close together pixels may still map to the same point and direction when they should not. Casting the initial ray classically seeks to minimize the impact of this to some extent as the rays may hit different points in the scene that they would not have if the walk had started at the camera. Let us denote the position of this first intersection point as x_0 and the direction of the ray as ω_0 and we start a random walk process.

We define the state of our random walk as a position x_i in the 3D space, and it moves along the ray direction ω_i at each step. A scattering event will change the direction to a new sampled direction ω_s if it hits a surface. In classical ray marching, each step i will be

formulated as

$$\omega_{i+1} = \begin{cases} \omega_s & (p_i \text{ is on surface}) \\ \omega_i & (\text{otherwise}) \end{cases} \tag{4.1}$$

$$p_{i+1} = p_i + \Delta V \omega_{i+1} \tag{4.2}$$

where ΔV is a distance along ω_i to the next voxel. To implement this process on a quantum computer, we naturally choose the equivalent of a random walk for quantum computing, a *quantum walk* [2].

Both quantum and random walks transition from their current states to the new one step by step. The key difference is that, instead of deciding on a single state to visit at each step, a quantum walk creates a superposition of states at each step as if we were to step in *every possible direction* at each x_i. For instance, if we have n possible (discretized) scattering directions, a quantum walk can take all the directions into account by constructing a superposition of all the n directions and the next states can be similarly defined as a superposition of all the n possible next positions. Each step of a quantum walk thus becomes

$$|p_i\rangle |\omega_{i+1}\rangle = C |p_i\rangle |\omega_i\rangle \tag{4.3}$$

$$|p_{i+1}\rangle |\omega_{i+1}\rangle = S |p_i\rangle |\omega_{i+1}\rangle \ . \tag{4.4}$$

The unitary matrix C changes $|\omega_i\rangle$ based on the position of the walker. Similarly, S is another unitary matrix that moves the walker by $\Delta V \omega_i$. We evolve the quantum walk until a desired number of steps are taken, and then measure the resulting quantum state.

Once measured, this qubit collapses the superposition into a single state, giving us a random state among all the possible paths visited by the walker, proportional to the probability that the walker was in that state after our chosen number of steps. This effectively allows for an exponential number of states to be visited by a walker with a polynomial storage and computation cost.

4.3 Quantum Ray Marching

The ray marching oracle O is defined as

$$O = P(SCM)^s R \, . \tag{4.5}$$

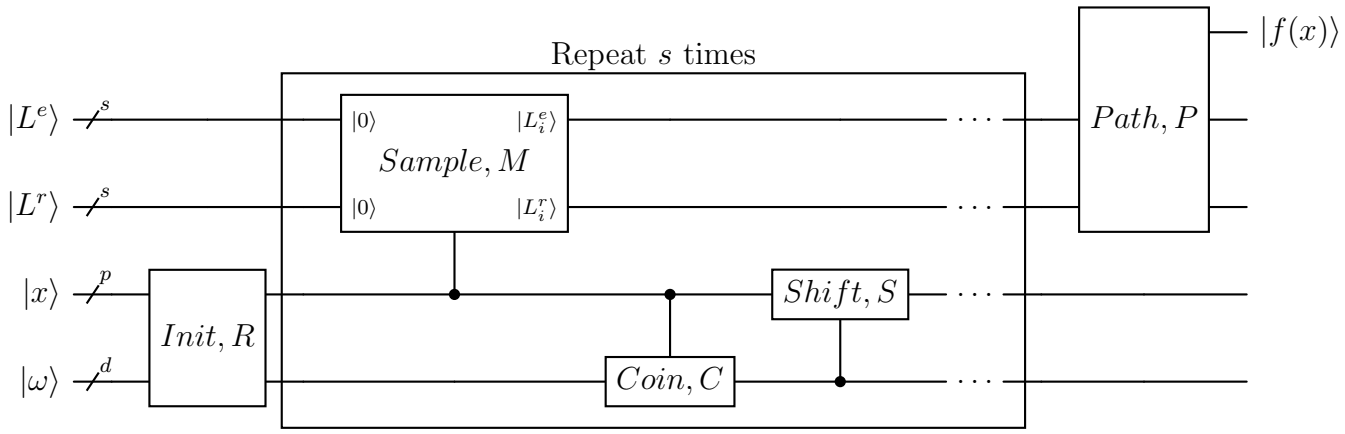


Figure 4.1: Overview of our ray marching circuit. The *Init* circuit initializes the position and direction of the walk. Next, the *Sample* gate encodes the emitted radiance and reflectance of the surface at the given position into a qubit from the $|L^e\rangle$ and $|L^r\rangle$ registers. After, the *Coin* gate scatters the ray in superposition over the outgoing directions. Finally, the *Shift* gate steps one unit along the ray. This process is repeated for each step of the walk. After all the samples are taken, *Path*, expanded in figure 4.2, evaluates the samples stored in $|L^e\rangle$ and $|L^r\rangle$ to evaluate $f(x)$ for all the paths in superposition.

The first gate, R, initializes the state of the walker at the position and direction of the first intersection of the ray that was cast classically. Once the initial state has been created, the sample gate, M is applied. Unlike a regular walk, we are interested in sampling the paths along the walk and not simply the final state. This means that we must store the information at each position along the walk, $\tilde{x}$. Trivially storing each position along the path in a separate register using controlled X,-gates would allow us to store the vertices visited at each step but would mean that as the scene grew in size the number of qubits needed would rapidly grow past what is currently possible to simulate. Instead, we need to store the samples in a more qubit efficient way. As the values that we wish to store are all normalized in the range of $[0, 1]$, amplitude encoding is a natural fit for the storing the samples. This motivates the construction of M. M encodes the light emitted, L^e, and reflected L^r, into two qubits via amplitude encoding the value in the 1 state of the qubit. This is done using a series of controlled RY gates. Each gate is controlled by the current position of the walker and applies a rotation of $\sin^{-1}(\sqrt{L^x})$.

$$M\left|x\right\rangle\left|0\right\rangle\left|0\right\rangle = \left|x\right\rangle(\sqrt{1+L^e}\left|0\right\rangle + \sqrt{L^e}\left|1\right\rangle)(\sqrt{1+L^r}\left|0\right\rangle + \sqrt{L^r}\left|1\right\rangle) \tag{4.6}$$

The next two gates are the shift and walk gates and are defined similarly to that of the coined quantum walks defined in Chapter 3. The coin gate governs the movement of the rays through the scene. As the surface material, or lack thereof in the case of the air in the scene, changes based on the position, the coins will be controlled by the position of the walker.

$$C\left|x\right\rangle\left|\omega\right\rangle = \left|x\right\rangle\left|\rho(x,\omega)\right\rangle \tag{4.7}$$

The coin must scatter the rays proportionally to the BSDF that is defined for the material at the given position. This is done by transferring the amplitude from the incoming direction to the outgoing directions. The coin operator may be defined as:

$$C = \sum_{x\in X}\left|x\right\rangle\left\langle x\right| \otimes C_x \tag{4.8}$$

where C_x is the coin that is defined for the given point in the scene. We present several of these coins in the following section.

Finally, the shift gate, S moves the walker from its current position, represented in $\left|x\right\rangle$ based on the direction in $\left|\omega\right\rangle$.

$$S\left|x\right\rangle\left|\omega\right\rangle = \left|x+\omega\right\rangle\left|\omega\right\rangle \tag{4.9}$$

Implementing a quantum adder is not quite a solved problem and the representation of the direction and position vectors will govern the implementation of the gate.

4.3.1 Coin Gates

Coins represent ray-surface (non)interactions in our quantum ray marching. We have three different coins, one models the air, the other two coins model different surfaces. The air coin models the non-participating medium that fills the space in the scene and is thus simply the identity gate:

$$C_{\text{air}} = I\,. \tag{4.10}$$

The first surface coin is the perfectly specular surface coin. As a perfectly specular surface simply reflects incoming light at an angle depending on the normal of the surface, this coin is fairly trivial. The coin will be dependent on the surface normal at the point of the scene, but that is known at the construction time of the circuit, and thus we are able to encode that in the gate directly. For a given surface normal, the specular coin is then a permutation matrix that maps incoming directions to outgoing directions. Since Permutation matrices are guaranteed to be unitary this gate can be implemented in the circuit.

$$C_{\text{specular}}(N) = P \tag{4.11}$$

$$P_{ij} = \begin{cases} 1 & \text{if direction } i \text{ is relected out in direction } j \\ 0 & \text{other wise} \end{cases} \tag{4.12}$$

The behavior of the gate is such that it maps the incoming vector to the appropriate matching reflected direction over the normal of the surface. This mapping can be precalculated and a circuit constructed that carries out this mapping.

The final surface coin that we present is the Lambertian surface coin. Lambertian surfaces are perfectly diffuse surfaces. As such, they evenly scatter light in every direction, and appear the same regardless which direction they are viewed from. Since these surfaces scatter light uniformly, the resulting superposition of directions should be an even superposition over the outgoing directions. The difficulty in this is that many ray directions may map to the same resulting distribution of scattered rays. Since all quantum gates must be reversible, it may seem like this scattering behavior should not be possible. One potential solution would be to use a new register for the direction at each step. This would make our circuits much wider than we would like as we would need an additional d qubits for each step. Instead, we make use of the fact that the amplitude of a state may be negative. This allows us to map the input ray directions to different quantum states by changing the signs of the amplitudes. By mapping each incoming direction to a unique quantum state in this fashion, we can construct a unitary matrix that enables the desired scattering behavior while still being reversible.

4.3.2 Quantum Evaluation of $f(x)$

Just being able to sample a path x is not sufficient for a full pipeline of quantum light transport simulation. We should evaluate the contribution function $f(x)$ in a quantum circuit where x is potentially a superposition of many paths represented as qubits.

The result of the previous step is a quantum state encoding the samples along the light paths that have been traced.

$$\bigotimes_{i=0}^{s} |L_i^e\rangle |L_i^r\rangle \tag{4.13}$$

Where $|L_i^e\rangle$ and $|L_i^r\rangle$ are the qubits storing the emitted and reflected light at the ith step of walk. Before we can apply any form of quantum amplitude estimation to this state, we need to calculate $f(x)$ using the values encoded in these qubits. To achieve this we construct the circuit P,

$$P|0\rangle \bigotimes_{i=0}^{s} |L_i^e\rangle |L_i^r\rangle = (\sqrt{1 - f(x)^2}\,|0\rangle + f(x)\,|1\rangle) \bigotimes_{i=0}^{s} |L_i^e\rangle |L_i^r\rangle \tag{4.14}$$

$$f(x) = \sum_{i=0}^{s} L_i^e \prod_{k=0}^{i-1} L_k^r \,. \tag{4.15}$$

Evaluating $f(x)$ requires the ability to add and multiply the values encoded in the amplitude of the sample qubits. Our method for doing this can be seen in Figure 4.2. Our approach takes inspiration from the work of Wang et al. [39]. Enough ancillary qubits are used to basis encode the number of steps that the walk takes. These are used to create a superposition over the number of steps that were taken in the walk, allowing us to control which multiplication we are adding to the output qubit. We perform these multiplications using multi-controlled X gates. Each gate will be controlled by the step number and all the samples that are needed to compute that term of the sum. The resulting amplitude, $f'(x)$, is

$$f'(x) = \sum_{i=0}^{s} \frac{1}{s} L_i^e \prod_{k=0}^{i-1} L_k^r \tag{4.16}$$

where s is the number of steps that were taken. This means the resulting mean estimation value, F', will be scaled by a factor of $1/s$ but F may be recovered simply by multiplying by s.

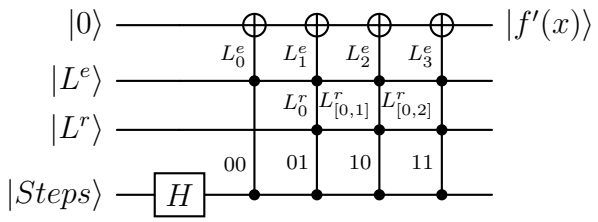


Figure 4.2: The internals of the *Path* gate for a three step walk. Each multi-controlled X gate calculates one of the terms of the sum in equation 4.16. The ancillary register, $|Steps\rangle$ is used to select the term of the sum.

4.4 Implementation

We list several implementation details that we chose for our experiments. Other choices are potentially feasible, but we explain the reasoning behind our choices below.

4.4.1 Encoding of Numbers

We encode the origin and direction via *basis encoding*, which can be thought of as fixed-point binary encoding of values as states. Basis encoding allows us to perform essentially the same class of computations on binary numbers as non-quantum computers can do. Another option is *amplitude encoding*, which encodes a value as a magnitude of a specific state, allowing us to represent a real number with an amplitude value. Arbitrary arithmetic operations on amplitudes are considered more challenging. For origins, the integer part represents a 3D voxel index, and the fractional part represents a sub-voxel location within the corresponding voxel. For directions, rather than representing a direction vector as three values, we tabulate a predefined set of directions as a 1D table of 3D vectors and index it via one integer value. It is still considered a form of basis encoding since one state corresponds to one direction. We found that this representation dramatically simplifies stepping along the ray and scattering of rays.

4.4.2 Scene Representation

We choose to represent a scene as a voxel grid for simple lookup and indexing of the needed scene information. At each voxel, we store the following material properties: emitted radiance, reflectance, (averaged) surface normal, and type of that voxel. We make

two assumptions to simplify the circuit construction, the emitted radiance is uniform in all directions and all surfaces are Lambertian. The voxel type defines how the rays are scattered as described already. The scene description will need to be built as a quantum circuit (conceptually the same as procedural modeling) since current quantum computers have no model of memory or storage. It is, however, not a fundamental requirement of our quantum ray marching, and we expect that a quantum storage will allow us to store a scene description, similar to how a scene is stored in non-quantum computers.

4.4.3 Direction Look-up Table Construction

As quantum computers are still at the level of circuits, implementing complex algorithms is difficult as operations that would normally be trivial, such as adding, require the construction of specialized circuits. To help facilitate the construction of the circuits presented above, some approximations were made to the standard ray marching that would be typically implemented on a classical computer. The main approximation that is made is that a look-up table of directions must be defined before circuit construction. This lookup table greatly simplifies the implementation of both the surface coins and the shift operator.

Our method does not rely on any particular implementation for defining this lookup table, uniformly sampling of the unit sphere will be chosen for the results presented in the following chapters, though empirically random sampling can provide better results when the number of directions used is small. It is important to note that the resolution of the lookup table effects the scaling of the circuit. The position of the walker must be represented with a high enough resolution such that at any step the addition of any two vectors will result in the walker arriving at a unique final location when starting from the same initial location. Failing to do this, will result in the direction of the lookup table not being accurately applied to the position of the walker. Also, the look-up of the appropriate vector to apply at each step is linear in the number of directions since there are no efficient memory lookups on quantum computers. This means that the depth of the circuit grows linearly as the number of directions increases.

4.4.4 Coin Gate Construction

Once the lookup table is defined we can then construct the circuit based on the scene and set of directions. The shift operator is implemented by controlled increment and decrement gates that add the constant values to the position register, controlled by the direction register. The coin operator is built out of the many controlled coin gates. Each

coin gate is controlled from the position of the walker based on what voxels the walker is currently in. The coins function as a mapping from the incoming direction ID to a set of outgoing IDs. For the air, each ID is mapped to itself. Since this operation has no effect, these gates are skipped in the circuit construction, but it can be helpful to think of them conceptually being the identity gate as then it may be treated the same as any other coin present in the scene. The specular coin is a 1-to-1 mapping of directions whereas the Lambertian coin is a 1 to many mapping. The exact directions that are elected for the mappings may differ depending on how the set of directions that was used, but the directions and weights must be selected such that they respect the properties of BSDF that the coin is emulating.

4.4.5 Mean Estimation

The result of running our proposed circuit is a qubit that stores F' at the desired point in the scene in the amplitude of the 1 state. To retrieve this value, we choose to use the approach from Nakaji [25]. Though we are not limited to this specific approach, it offers several features that complement our goals. Unlike the work by Johnston [14], the method by Nakaji [25] is similar to the method by Shimada and Hachisuka [32] in that it does not use phase estimation to accomplish this task. As identified by the prior work [32], we found that the use of amplitude amplification instead of phase estimation allows us to cut down on the number of qubits and multi qubit gates that are needed.

Chapter 5

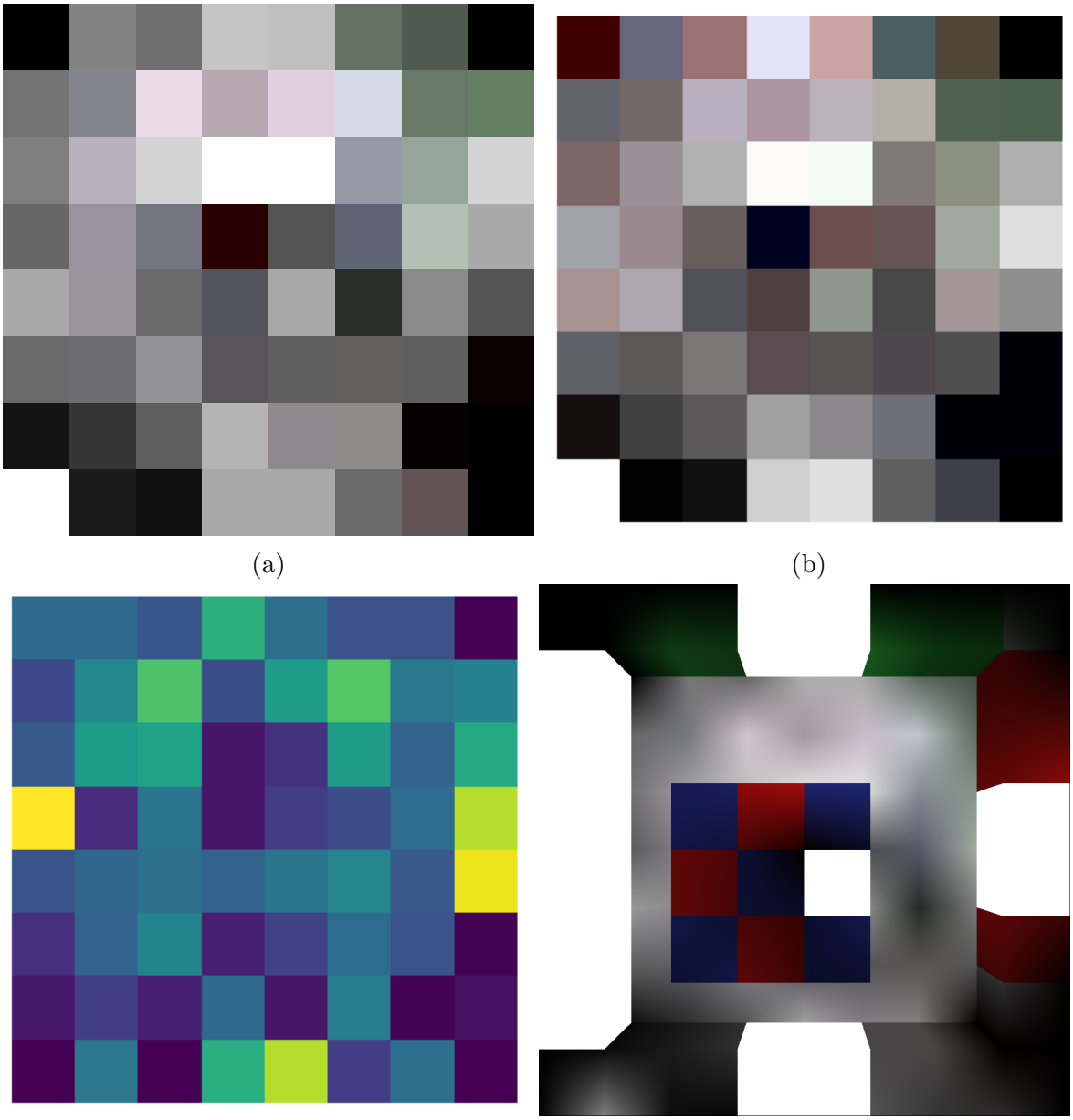
Results

We conducted three numerical experiments to verify our method. First we verify that our method is capable of simulating quantum light transport by rendering a $2D$ light map. We then modified our method to simulate $3D$ scenes in a given limited framework for quantum computing at the moment. Finally, we verified that our method can achieve the claimed convergence rate of quantum numerical integration.

5.1 Fully Quantum 2D Light Transport

Unfortunately, even modern quantum computers are not yet robust enough to run our complete circuit for general 3D scenes. We thus tested our method by simulating the quantum circuit for $2D$ ray marching on a classical computer. We choose to test our method on a $2D$ scene as simulating circuits of this complexity is very computationally expensive and stepping down to $2D$ allows for modelling of more complicated scenes with the same computational budget. We model the scene as a 8×8 grid. At each cell, we store the same information that we would store in a $3D$ scene. The only change is that the directions are now defined on a unit circle instead of a sphere. We use our method to create a light map by sampling the incoming light at the center of every cell in the scene. We then use the resulting light map to render a higher resolution image, using bilinear interpolation to sample from the light map. Even in this simplified problem setting, this approach takes roughly 5 days to finish simulating all the quantum circuits for the full image.

Figure 5.1 shows the resulting images of our method. We can see that quantum ray marching is capable of capturing both direct and indirect lighting in the scene. This is



(a) (b) (c) (d)

Figure 5.1: (a) The 8×8 light map of the $2D$ scene sampled at each point using a quantum walk of three steps. (b) The same light map generated by tracing exponential rays classically. (c) The mean square error of the classical and quantum light maps. Besides some noise, the results match overall. (d) The final higher-resolution image using the quantum light map

best shown by the light being reflected off the colored walls causing some color to spill onto the floor. Though this is a simple scene compared to modern standards, it shows that our method can accurately simulate the complicated process of light transport.

5.2 Emulated 3D Light Transport

To evaluate our method on more complicated $3D$ scenes, we choose to take a different approach. We developed a classical ray-marcher that samples the scene with the same discretization and scattering behavior as the quantum method. This yields the same samples that would be used for the quantum walk but takes exponential time with the number of steps instead of the linear time that would be taken on a quantum computer. These values are then used directly in the quantum circuit simulation in place of the quantum walk. We call this approach *emulated* to distinguish it from the fact that our 2D results are from simulation of quantum computers. It allows us to validate the properties of our method, and nothing has been fundamentally compromised, except that the actual running time is now exponential.

To further simplify the quantum circuit, we limit the total number of paths that are used. These paths are selected at random from the set of total paths and are then used to initialize $|L^e\rangle$ and $|L^r\rangle$. At this point, we can apply our path processing circuit to generate the final path contribution. We then use the amplitude estimation method [25] to estimate the final value.

Figures 5.2 and 5.3 show the results. The images still do contain noise due to the limited number of paths used to keep runtimes at a reasonable length. Unlike previous quantum ray tracing methods [24, 30], our approach easily handles paths with multiple bounces as well as other effects such as soft shadows with no changes to the base implementation.

5.3 Convergence Rates

To compare convergence rates, we need a common metric for computation cost, which is difficult as how the methods function are inherently different. We choose to use the "number of samples" in MC integration or quantum amplitude estimation as the common metric. This means, for the classical method, we will be using the number of paths traced, and, for our method, we will use the number of times our ray marching circuit is executed. We argue that this is a fair metric as it is the way in which both methods sample from

Figure 5.2: A modified Cornell box rendered with our method using 32 paths per pixel (structural noise is due to limitations of simulation of quantum computing)

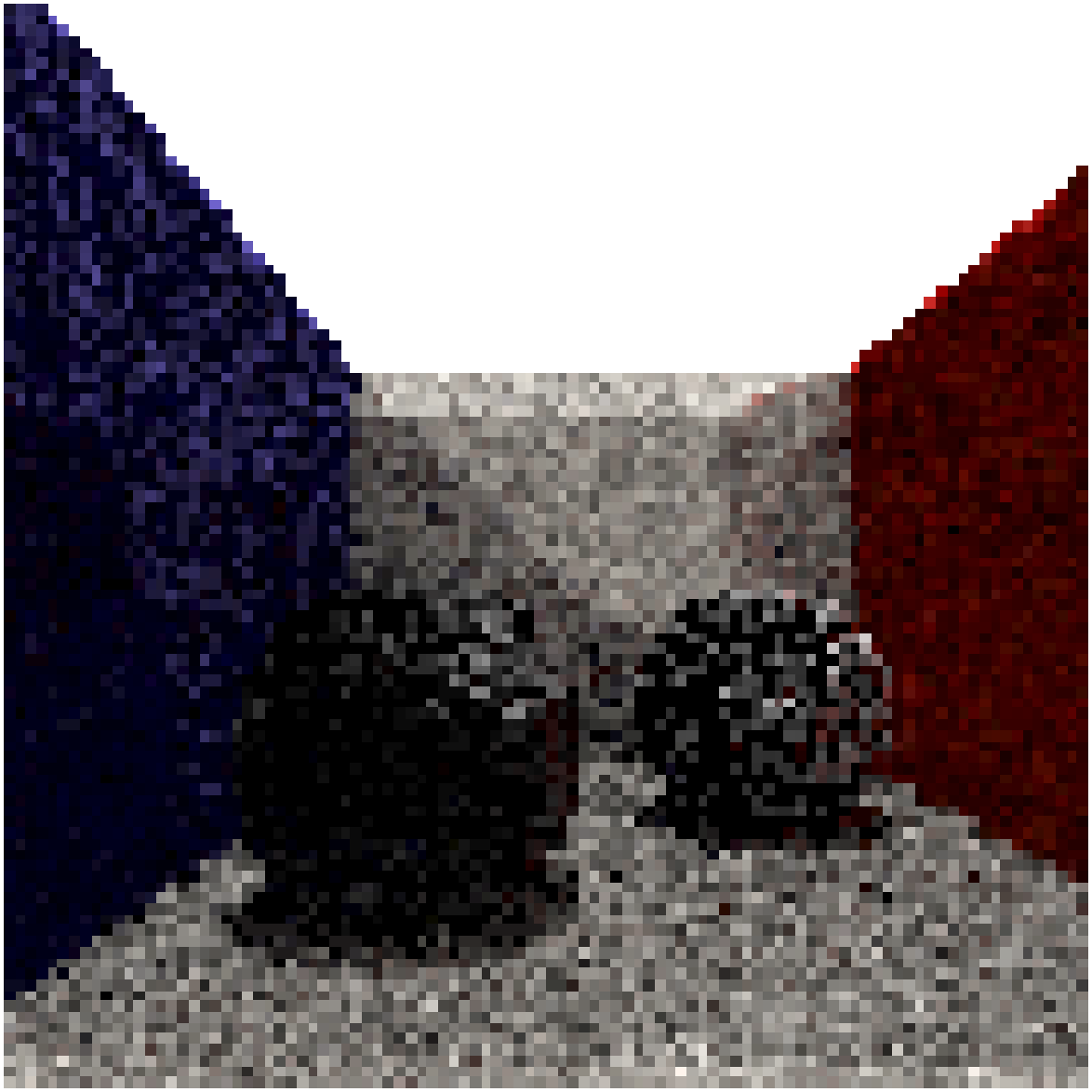


Figure 5.3: Different test scene of two spheres in a box with an overhead light rendered by the same algorithm. While the result is very noisy due to the current limitations of the quantum computing environment we used, the image shows how shadows and some interreflections can be captured via a fully quantum algorithm for the first time.

the scene. We could have used the "number of light transport paths" per evaluation where our method will have exponentially many more paths for one evaluation, but we believe it is unfair for MC integration since our quantum method cannot read out such a result F' directly.

Figure 5.4 shows that our method does converge with the expected behavior. The classical MC method has the expected convergence with a slope of about -0.471. Our method actually has a better than expected result with a slope of about -1.407. While this convergence is better than the classical approach, the actual error of our method does initially start higher than classical MC. The reason for this behavior is likely because the first few samples are to simply generate an MC error bounds on the value. We do not benefit from the improved convergence during this step of quantum mean estimation.

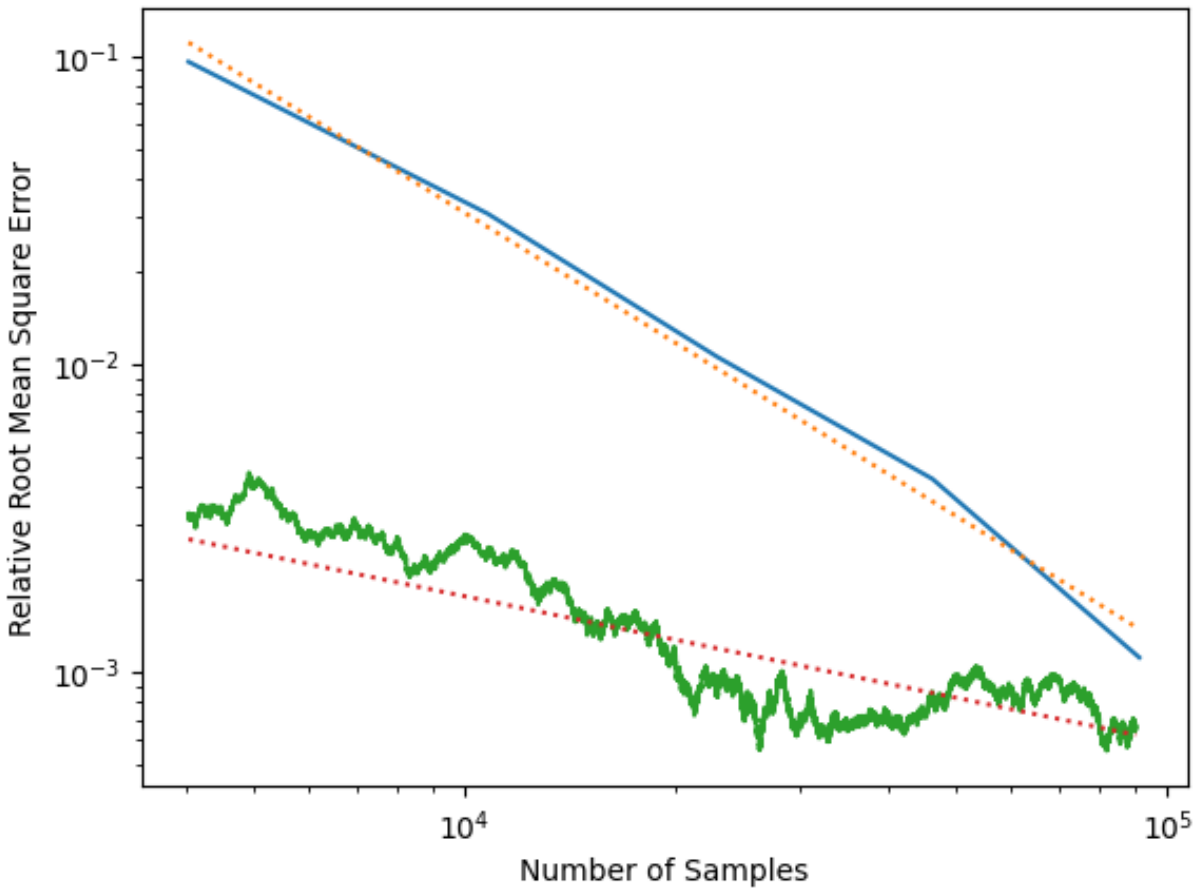


Figure 5.4: The convergence plot. Our quantum light transport simulation with quantum ray marching (blue) converges asymptotically faster than classical MC rendering (green).

Chapter 6

Arbitrary Materials

6.1 Motivation

The work presented in previous chapters can be extended to allow the rendering of scenes containing more arbitrary materials than purely specular or Lambertian materials. This would make the method more useful and closer to a viable alternative to rendering on classical computers. Being able to render arbitrary materials would mean that any scene that can be represented as a voxel grid could be rendered with the method that we present

As presented in Chapter 2 the BRDF defines the properties of how light interacts with a surface and scatters back into the scene. To render materials properly, the scattering behavior of the BRDF must be represented in a method that can be used in the simulation. These functions define a PDF over the hemisphere that governs the probability that an incoming ray is scattered in any given direction. Thus, to be able to handle a broader array of materials in this quantum ray marching framework, the corresponding PDF of the material must be able to be represented in the framework. Since the operation will need to run on a quantum computer, it must be a unitary operation. This poses a challenge as it is not immediately obvious how to do this or if the PDF can even be converted to a unitary operator.

6.2 BRDF to Transition Matrix

To create a method to that is capable of representing more arbitrary materials, it will be helpful to reformat the BRDF and its PDF to a representation that more closely resembles

those presented in quantum computing literature. The first step in doing this is to discretize the BRDF over the set of directions that will be used. This is done so that it may be represented in the quantum walk. When doing this, it is important that the reciprocity of the BRDF is preserved. Doing so may require modifications to the original BRDF as not all possible directions that the BRDF was originally defined over will be represented. Unlike on a classical computer, where failing to have a BRDF that is reciprocal would just lead to images that are not physically accurate, for a quantum BRDF this is one of the properties that allows us to efficiently implement the BRDF. As well, unlike in the classical method, the BRDF should be defined over the hemisphere at a given point in the scene. This impacts the directions that would typically be below the surface. This is not of concern as long as the scene construction does not allow a surface to be intersected from a direction that it should not be hit by. The scattering behavior of the directions that are below the surface of the object can be defined in any way so long as it does not prevent the unitary construction of the total BRDF coin.

Once the BRDF is discretized over the set of directions, obeying reciprocity and conservation of energy, it can then be converted into a quantum BRDF to be used in the quantum ray marching process. Classically, the PDF is then sampled from using one of a few techniques to choose the direction that light will be scattered in. In the quantum version of light transport we instead wish to shift the amplitudes proportionally to this PDF, representing the probability of light scattering in that direction directly.This repeated scattering of light can be viewed as a Markov process. In this Markov process, the current state of the system can be defined as the position and direction of the vector. Starting in this initial state, we can then construct a transition matrix that defines the transitions between any state in the scene.

6.3 Transition Matrix to Circuit

Using the transition matrix we can finally construct the circuit for the quantum walk. Unfortunately, the transition matrix in general is not a unitary matrix. There is a large amount of work on quantum walks for arbitrary graphs. The most relevant technique is the work by Szegedy [34]. Szegedy's quantum walks are the quantization of a classical Markov process. Szegedy's walks function similarly to coin-based walks but differ in the way that the state space is extended from just the set of vertices. In the coin-based walk, the state space is extended using the addition of the coin register. This register is used to represent the direction that the walk will move in. In the Szegedy walk the state space is instead extended by duplicating the set of vertices, creating two sets; the original set x

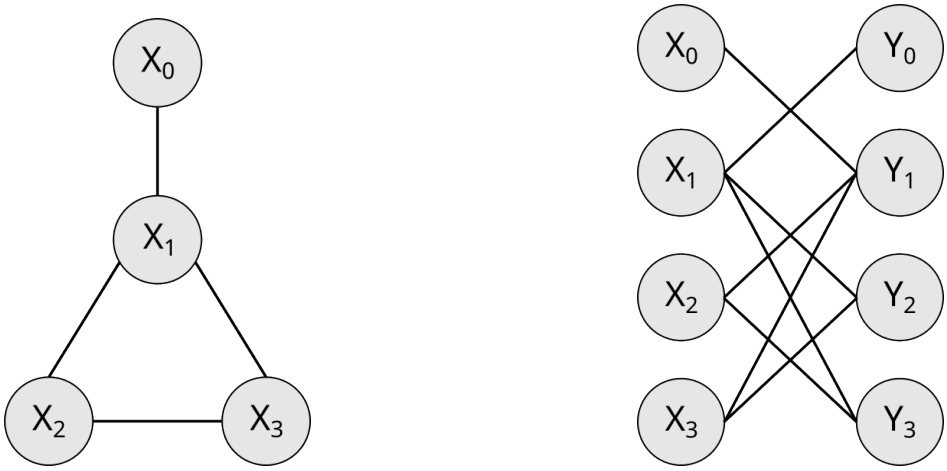


Figure 6.1: Left: the original graph that we want to run a quantum walk over. Right: the corresponding bipartite graph that is created to allow for the construction of a Szegedy's walk

and the copy y. This second set of vertices are connected to the original such that they create a bipartite graph between the two sets. The edges of the graph are replaced from the original and instead the vertices in x are connected to the vertices in y that are the copy of the vertex in x that would have been connected to. If there was an edge connecting x_i and x_{i+1} there would instead be an edge connecting x_i with y_{i+1}.

With this in mind, the state of the walker is represented by the two registers where $|x\rangle$ is the current position of the walker and $|y\rangle$ is the vertex that it will be at after the next step of the walk. Much like the coin-based walks, the Szegedy's walk is also constructed using a combination of two gates.

$$U = R_2 R_1 \tag{6.1}$$

These two operations are reflections around the different vectors defining the basis sates of the quantum walk. The operators are defined as

$$R_1 = 2 \sum_{x \in V} |\phi_x\rangle \langle \phi_x| - I \tag{6.2}$$

$$R_2 = 2 \sum_{y \in V} |\psi_y\rangle \langle \psi_y| - I \tag{6.3}$$

where

$$\phi_x = |x\rangle \otimes \sum_{y \in V} \sqrt{P_{xy}} \, |y\rangle \tag{6.4}$$

$$\psi_y = \sum_{y \in V} \sqrt{P_{yx}} \, |x\rangle \otimes |y\rangle \ . \tag{6.5}$$

Though they are different ways of framing the problem, it can be shown that the coined walk and the Szegedy's walk are equivalent [28]. Portugal [28] show that every step of the Szegedy's walk is equal to two steps of the coined walk. In fact, the operator R_1 can be shown to be equal to the coin operator in the coin-based walk. This is helpful as it provides insight into how walks that would be difficult to design in the coin based quantum walk can be implemented. Specifically, the work by Wong [44] presents a framework for building coin-based walks over weighted graphs based on this relation between coined walks and the Szegedy's walks. The type of walk that is needed for quantum ray marching is similar to those presented by Wong [44] and as such, the next section of work is heavily based on that work.

6.4 Implementation

The circuit presented in Chapter 4 must be modified to allow for the new extension. Instead of having a coin register that represents the direction of the positions in the scene, we must instead represent the state of the walker as both its current position and direction. Then the position and direction registers need to be duplicated to create the bipartite graph. Thus, the state of the system is then

$$|x_0, \omega_0\rangle \, |x_1, \omega_1\rangle \ , \tag{6.6}$$

where $|x_0, \omega_0\rangle$ is a position direction pair that represents the current state of the system and $|x_1, \omega_1\rangle$ is the following state. From there our coin can be constructed as follows:

$$C = 2 \sum_{x_i, \omega_i \in \Omega} |\phi_{x_i,\omega_i}\rangle \langle \phi_{x_i,\omega_i}| - I \tag{6.7}$$

$$\phi_{x_i,\omega_i} = |x_i, \omega_i\rangle \otimes \sum_{x_j, \omega_j \in \Omega} \sqrt{P_{x_i,\omega_i \to x_j,\omega_j}} \, |x_j, \omega_j\rangle \tag{6.8}$$

where the shift circuit then simply becomes a swap gate between the two sets of registers. This provides the benefit of being comparably simpler than the arithmetic circuits that were previously used.

The rest of the circuit may be kept the same as in Chapter 4 for the sampling as the register representing the current position and direction have not changed in their representation.

Unfortunately, this method is significantly more computationally expensive to simulate as it nearly doubles the number of required qubits. As simulating even the simple scene in Figure 5.1 took a week, and the doubling of the number of qubits leads to an exponential increase in the state space, simulating this method is currently impractical for even simple scenes. For this reason, we choose not to run a simulation of this method. Though in the future it would be worthwhile to try out this method.

Chapter 7

Discussion

7.1 Novelty in Quantum Computing

Though there has been limited work in the field of computer graphics, quantum numerical integration been an active area of research in other fields. One of these fields is finance, where the problem of pricing financial derivatives is classically done via MC integration. Those prior works [22, 29, 42] have looked at the application of quantum mean estimation methods for this problem. For these methods, the challenge of state preparation for quantum numerical integration was side-stepped by assuming that the solution to the *integral* is readily computable. While this assumption is impractical since this solution is what we wanted to compute, it allows for such quantum methods to be tested and analyzed. Our work, in contrast, shows how to exactly compute the *integrand* for the path integral for light transport on quantum computers, and we numerically tested our method without making this impractical assumption. The most similar work to our method is by Chakrabarti et al. [7]. They proposed to use quantum random walks to estimate the volume of a given shape. Their method uses continuous quantum walks and simulated annealing in contrast to our approach that relies on discrete walks and mean estimation.

7.2 Complexity Scaling of Quantum Circuit

There are two metrics in which a quantum circuit can scale and both of them are important for the usability of an algorithm. The first is the width of the circuit, or how many qubits are needed. With current quantum computers being very limited in the number of qubits,

this is a large restriction in being able to run on physical devices. It is also an issue for simulating the circuit on a classical computer as the size of the quantum state grows exponentially in the number of qubits. Our approach scales in $O(s + log(p) + log(d))$ in the number of qubits needed. That is, it scales logarithmically in the number of positions p along an axis and directions d that can be represented in the scene, but linear in the number of steps s needed.

This scaling could be improved to also be logarithmic, but it would require that the light samples be stored via basis encoding. This is fine, but it means that the simple controlled R_y gates used to encode the material properties of the scene would need to be replaced with more complicated arithmetic circuits. This would greatly affect the other important scaling dimension of circuit depth.

The other important metric is the depth of the circuit. Our proposed circuit scales in depth by $O(s(p^3 + d))$. Scaling cubically in scene size is not ideal, but it is still viable. The main issue is that quantum computers do not have any form of memory or storage that can be referenced. This restriction means that each voxel must be represented in the circuit. As there are p^3 voxels, we get p^3 gates at each step. This analysis is only in the worst case though, as in practice there is often overlap in the material properties between different voxels, allowing us to simplify the circuit to use fewer gates by combining gates. This issue would be entirely negated if quantum random memory were to be invented [9]. In that case, the material values could then be accessed in a similar method as on a classical computer, removing the material lookup from the circuit depth.

7.3 Quantum Primitive

Computing a 3D translation (3 additions) via homogeneous matrix multiplication (16 multiplications and 12 additions) appears to be a remarkably inefficient design choice at first. However, translations are pervasively computed this way in graphics. It offers two system-wide advantages even though the individual unit appears suboptimal. Mapping translation to a matrix product allows a single computational unit of abstraction to many different operations rather than requiring bespoke hardware and APIs for each. By mapping those operations to the common computational substrate of homogeneous matrix multiplication, it also allows end-to-end optimization and avoids conversion between domains (e.g., between matrices and individual vectors and quaternions for the operations).

Similarly, there is a value of having quantum primitives to combine into quantum circuits, even in cases where there is no efficiency gain for performing that primitive in the

quantum domain. It avoids a larger task having to leave the quantum realm, enables end-to-end optimization, and allows reuse of generic quantum elements instead of requiring bespoke ones. A foreseeable future would be to have a quantum co-processor, *quantum computing unit* (QPU) [15], that acts like a GPU but performs quantum computation in tandem with non-quantum processors. For the graphics community, the current state of quantum computers would be reminiscent of the early stages of programmable GPUs where the number of instructions were quite limited. This is a core system design issue. We make observations for quantum-classical hybrid computation. It has also been seen in (fiber-)optical-electrical switching in routers, GPU-CPU computation for rendering and high-performance computing, tensor core-ALU for machine learning, and digital-analog for signal processing. In each case, avoiding a domain switch for subunits is more critical for net efficiency than having every subunit implemented in the individually most efficient way. For example, when working on a machine learning pipeline, it is often more efficient to remain in the deep neural net structure even for units of the pipeline for which analytic solutions are known, because it allows end-to-end solutions entirely within one computational framework.

7.4 Limitations

While our framework is quite general, there are some limitations that mostly come from our current implementation. Foremost is its reliance on large many-qubit gates. Current generation quantum computers only support a limited set of one and two qubit gates, so these larger gates need to be decomposed into many gates, greatly increasing the depth of the circuit. The increase in the depth in turn increases the amount of time the system needs to stay coherent, and eventually leads to the chance of errors. The other limitation is the use of angle encoding to store the sample values limits the light values that can be defined. We thus need to have a separate scaling to represent a light value greater than 1. It likely would be possible to use basis encoding for the samples but doing so would lead to even more complicated circuits.

7.5 Future Work

Though we are only capable of modelling and rendering simple scenes currently, this is not a fundamental issue of our method but instead a limitation on the current capabilities of

quantum computers (and quantum simulation). Being able to numerically test and verify our method on more complex scenes with a broader set of materials would be interesting.

Ray marching was chosen as the method of finding the ray scene intersections for this work, but the approach of generating light paths with a quantum walk should not limited to ray marching. To extend the method to ray tracing, the shift operator in the discrete quantum walk would need to be modified to compute the first intersection along the ray direction. This change would save the method from having to make many steps before hitting a surface. This would in turn greatly mean less steps of the quantum walk would be needed. We are also interested in running our algorithm on actual quantum computers once they became capable of running our 3D approach.

References

[1] Dorit Aharonov. A simple proof that toffoli and hadamard are quantum universal, 2003.

[2] Yakir Aharonov, Luiz Davidovich, and Nicim Zagury. Quantum random walks. *Phys. Rev. A*, 48:1687–1690, Aug 1993. doi: 10.1103/PhysRevA.48.1687. URL https://link.aps.org/doi/10.1103/PhysRevA.48.1687.

[3] Carolina Alves, Luís Paulo Santos, and Thomas Bashford-Rogers. A Quantum Algorithm for Ray Casting using an Orthographic Camera. In *2019 International Conference on Graphics and Interaction (ICGI)*, pages 56–63, 2019. doi: 10.1109/ICGI47575.2019.8955061.

[4] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Phys. Rev. A*, 52:3457–3467, Nov 1995. doi: 10.1103/PhysRevA.52.3457. URL https://link.aps.org/doi/10.1103/PhysRevA.52.3457.

[5] Michel Boyer, Gilles Brassard, Peter Høyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik: Progress of Physics*, 46(4-5):493–505, 1998.

[6] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation. *Contemporary Mathematics*, 305:53–74, 2002.

[7] Shouvanik Chakrabarti, Andrew M. Childs, Shih-Han Hung, Tongyang Li, Chunhao Wang, and Xiaodi Wu. Quantum algorithm for estimating volumes of convex bodies. *ACM Transactions on Quantum Computing*, 4(3), may 2023. ISSN 2643-6809. doi: 10.1145/3588579. URL https://doi.org/10.1145/3588579.

[8] D. Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 425(1868):73–90, 1989. ISSN 00804630. URL http://www.jstor.org/stable/2398494.

[9] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008. doi: 10.1103/PhysRevLett.100.160501. URL https://link.aps.org/doi/10.1103/PhysRevLett.100.160501.

[10] Dmitry Grinko, Julien Gacon, Christa Zoufal, and Stefan Woerner. Iterative quantum amplitude estimation. *npj Quantum Information*, 7(1):52, 2021.

[11] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[12] John C Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[13] Henrik Wann Jensen. *Realistic image synthesis using photon mapping*, volume 364. Ak Peters Natick, 2001.

[14] Eric R. Johnston. Quantum supersampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, page 1. Association for Computing Machinery, 2016. ISBN 978-1-4503-4282-7. doi: 10.1145/2897839.2927422. URL https://doi.org/10.1145/2897839.2927422.

[15] Eric R. Johnston, Nic Harrigan, and Mercedes Gimeno-Segovia. *Programming Quantum Computers: Essential Algorithms and Code Samples*. O'Reilly Media, 2019. ISBN 9781492039631. URL https://books.google.ca/books?id=SKegDwAAQBAJ.

[16] Karthik S. Joshi, S. K. Srivatsa, and R. Srikanth. Path integral approach to one-dimensional discrete-time quantum walk, 2018.

[17] Karuna Kadian, Sunita Garhwal, and Ajay Kumar. Quantum walk and its application domains: A systematic review. *Computer Science Review*, 41:100419, 2021. ISSN 1574-0137. doi: https://doi.org/10.1016/j.cosrev.2021.100419. URL https://www.sciencedirect.com/science/article/pii/S1574013721000599.

[18] James T Kajiya. The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, 1986.

[19] J Kempe. Quantum random walks: An introductory overview. *Contemporary physics*, 44(4):307–327, 2003. ISSN 0010-7514.

[20] Marco Lanzagorta and Jeffrey K. Uhlmann. Hybrid quantum-classical computing with applications to computer graphics. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05, pages 2–es. Association for Computing Machinery, 2005. ISBN 978-1-4503-7833-8. doi: 10.1145/1198555.1198723. URL https://doi.org/10.1145/1198555.1198723.

[21] Christiane Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*. Springer Series in Statistics. Springer New York, 2009. ISBN 9780387781655. URL https://books.google.ca/books?id=wj5OyydZ5bkC.

[22] Yongming Li and Ariel Neufeld. Quantum monte carlo algorithm for solving black-scholes pdes for high-dimensional option pricing in finance and its proof of overcoming the curse of dimensionality, 2023.

[23] Xi Lu and Hongwei Lin. Improved quantum supersampling for quantum ray tracing, 2022.

[24] Xi Lu and Hongwei Lin. A framework for quantum ray tracing, 2022.

[25] Kouhei Nakaji. Faster Amplitude Estimation. 20:1109–1123, 2020. ISSN 15337146, 15337146. doi: 10.26421/QIC20.13-14-2. URL http://arxiv.org/abs/2003.02417.

[26] Carlos Navarrete-Benlloch. Introduction to quantum optics, 2022.

[27] Matt Pharr, Wenzel Jakob, and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2016. ISBN 0128006455.

[28] Renato Portugal. Establishing the equivalence between szegedy's and coined quantum walks using the staggered model. *Quantum Information Processing*, 15:1387–1409, 2015.

[29] Patrick Rebentrost, Brajesh Gupt, and Thomas R Bromley. Quantum computational finance: Monte carlo pricing of financial derivatives. *Physical Review A*, 98(2):022321, 2018.

[30] Luís Paulo Santos, Thomas Bashford-Rogers, João Barbosa, and Paul Navrátil. Towards Quantum Ray Tracing. 2022. URL http://arxiv.org/abs/2204.12797.

[31] Benjamin Schumacher. Quantum coding. *Phys. Rev. A*, 51:2738–2747, Apr 1995. doi: 10.1103/PhysRevA.51.2738. URL https://link.aps.org/doi/10.1103/PhysRevA.51.2738.

[32] Naoharu H. Shimada and Toshiya Hachisuka. Quantum coin method for numerical integration. *Computer graphics forum*, 39(6):243–257, 2020. ISSN 0167-7055.

[33] Shlomi Steinberg, Pradeep Sen, and Ling-Qi Yan. Towards practical physical-optics rendering. *ACM Transactions on Graphics*, 41(4):1–13, Jul 2022. doi: 10.1145/3528223.3530119.

[34] M. Szegedy. Quantum speed-up of markov chain based algorithms. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004. doi: 10.1109/FOCS.2004.53.

[35] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Photorealistic Rendering Techniques*, pages 145–167. Springer, 1995.

[36] 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*, pages 419–428, 1995.

[37] Salvador Elías Venegas-Andraca. Quantum walks: a comprehensive review. *Quantum information processing*, 11(5):1015–1106, 2012. ISSN 1570-0755.

[38] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in o (n log n). In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 61–69. IEEE, 2006.

[39] Shengbin Wang, Zhimin Wang, Guolong Cui, Lixin Fan, Shangshang Shi, Ruimin Shang, Wendong Li, Zhiqiang Wei, and Yongjian Gu. Quantum Amplitude Arithmetic. 2020. URL http://arxiv.org/abs/2012.11056.

[40] Manuela Weigold, Johanna Barzen, Frank Leymann, and Marie Salm. Data encoding patterns for quantum computing. In *Proceedings of the 27th Conference on Pattern Languages of Programs*, PLoP '20, pages 1–11. The Hillside Group, 2022. ISBN 978-1-941652-16-9.

[41] Turner Whitted. An improved illumination model for shaded display. In *ACM Siggraph 2005 Courses*, pages 4–es. 2005.

[42] Stefan Woerner and Daniel J Egger. Quantum risk analysis. *npj Quantum Information*, 5(1):15, 2019.

[43] Daniel K Wójcik and JR Dorfman. Diffusive-ballistic crossover in 1d quantum walks. *Physical review letters*, 90(23):230602, 2003.

[44] Thomas G. Wong. Coined quantum walks on weighted graphs. *Journal of Physics A: Mathematical and Theoretical*, 50, 2017.

Appendix

Proof of Exponential Paths

To illustrate how our method can evaluate an exponential number of paths relative to the number of steps in the walk, we apply our method to evaluate the integral along some function $f(x)$ on a $1D$ line. We choose to evaluate our method this way as this setting allows for analysis of the paths traced by the walk compared to a more complicated domain of light paths in a $3D$ scene.

To analyze the paths of the walk we will be following the work of Joshi et al. [16]. Beginning with an initial state

$$|\psi_0\rangle = |x_0\rangle \otimes (\alpha\,|\uparrow\rangle + e^{i\phi}\beta\,|\downarrow\rangle) \bigotimes^{s} |0\rangle\ , \tag{1}$$

where x_0 is the initial position of the walker. The direction register is set to $(\alpha\,|\uparrow\rangle + e^{i\phi}\beta\,|\downarrow\rangle)$, with α and β being real values, where $|\uparrow\rangle$ implies heading to the right and $|\downarrow\rangle$ is to the left. The remaining qubits will be used to store the samples along the walk. In the full rendering approach, $2s$ qubits would be needed as we sample 2 values at each step, but only 1 sample is needed for this task. Evolving the system using a coin of $C = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ results in

$$|\psi_1\rangle = SCM\,|\psi_0\rangle = (\alpha\,|x_1\rangle\,|\uparrow\rangle\,|s_0\rangle - e^{i\phi}\beta\,|x_{-1}\rangle\,|\downarrow\rangle\,|s_0\rangle) \bigotimes^{s-1} |0\rangle\ . \tag{2}$$

Joshi et al. [16] show that for a $1D$ walk, the number of potential paths doubles at each step. This means the total number of paths after n steps will be equal to 2^n giving us an exponential number of paths that could be traced. Unlike most quantum walk works, our method is interested in the result of the paths of the walk and not the vertices that are

reached at the end of the walk. As the walk progresses and the state evolves, we sample the $f(x)$ at each position visited by the walker using the M circuit described in the main paper. This captures the path that was taken by the walker. The circuit M does not depend on the walk, depending solely on the current position of the walker, so it is able to capture the paths in the more complicated domain of rendering. Similarly to the $1D$ walk, there will also be a similar exponential branching as the light is scattered at each diffuse surface.