

Universidade do Minho

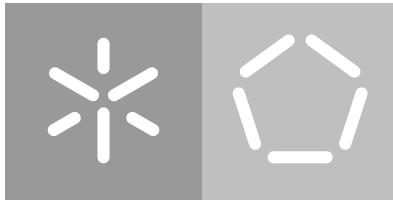
Escola de Engenharia

Departamento de Informática

Carolina Alves

A Quantum Algorithm for Ray Casting using an Orthographic Camera

December 2019



Universidade do Minho

Escola de Engenharia

Departamento de Informática

Carolina Alves

A Quantum Algorithm for Ray Casting using an Orthographic Camera

Master dissertation

Master Degree in Physics Engineering

Dissertation supervised by

Luís Paulo Santos

December 2019

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada. Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.



Atribuição-NãoComercial-Compartilhaigual
CC BY-NC-SA

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

ACKNOWLEDGEMENTS

This thesis would not have been possible without the help and guidance of Professor Luís Paulo Santos, my supervisor, to whom I want to give a big thank you.

I also want to thank my parents and my brother for all of their support throughout these course's years.

A kind thank you to my colleagues and friends who accompanied me across these years at university. Wishing you all the best.

Lastly, I want to thank all the professors in this course for all of their work that, in one way or another, helped me and led me to this.

This work is a result of project “SmartEGOV/NORTE-01-0145-FEDER-000037”, supported by the Norte Portugal Regional Operational Programme (NORTE 2020), under the PORTUGAL 2020 Partnership Agreement, through the European Regional Development Fund (ERDF).

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

ABSTRACT

Quantum computing has the potential to provide better time complexities (than those achieved with classical computers) for challenging problems solved by classical computers or even provide a solution for problems out of reach of classical computers in terms of time complexity (where the time consumed for the resolution of the problem is not practical, e.g. thousands of years). Here we're not solving an unsolvable problem but trying to improve its time complexity. There are several problems in rendering that are good candidates to being solved in a quantum fashion. Although previous research has proposed of theoretical ways of doing this, here we present a practical solution. This work takes a first step in applying quantum computing to one of the most fundamental operations in rendering: ray casting. This technique allows computing visibility between two points in a 3D model of the world which is described by a collection of geometric primitives. The algorithm returns, for a given ray, which primitive it intersects closest to its origin. Without a spatial acceleration structure, the complexity for this operation is $O(N)$. The main goal of this work is to use the Grover's Algorithm, a quantum search algorithm based on amplitude amplification, to improve the complexity of this problem. This algorithm provides a quadratic speed up allowing for visibility evaluation for unstructured primitives in $O(\sqrt{N})$ steps. Due to technological limitations associated with current quantum computers we had to simplify our problem's structure and in this work the geometrical setup is limited to rectangles and parallel rays (orthographic projection).

Keywords: quantum computing, Grover's algorithm, complexity, ray casting.

RESUMO

A computação quântica tem o potencial de proporcionar melhores complexidades temporais, do que aquelas alcançadas por computadores clássicos, para problemas exigentes resolvidos por estes ou até proporcionar uma solução para problemas fora do alcance dos mesmos em termos de tempo consumido (onde o tempo necessário para a resolução do problema não é prático, como por exemplo milhares de anos). Não estamos a tentar resolver um problema sem solução mas sim a tentar melhorar a sua complexidade. Existem vários problemas de renderização que são bons candidatos para serem resolvidos de forma quântica. Apesar de existirem algumas propostas teóricas para alcançar isso mesmo, aqui é apresentada uma solução prática. Este trabalho dá um primeiro passo em aplicar a computação quântica a uma das operações mais fundamentais da renderização: ray casting. Esta técnica permite computar visibilidade entre 2 pontos num modelo 3D do mundo que é descrito por um conjunto de primitivas. O algoritmo retorna, para um dado raio, qual a primitiva que este intersecta mais próxima da sua origem. Sem uma estrutura espacial de aceleração, a complexidade desta operação é $O(N)$. O principal objetivo desta dissertação é usar o algoritmo de Grover, um algoritmo quântico de procura, para melhorar a complexidade deste problema. O algoritmo permite uma aceleração quadrática possibilitando uma avaliação de visibilidade para primitivas não estruturadas em $O(\sqrt{N})$ passos. Devido a limitações tecnológicas associadas com os atuais computadores quânticos, tivemos a necessidade de simplificar a estrutura do nosso problema e, neste trabalho, a configuração geométrica é limitada a retângulos e raios paralelos (projeção ortográfica).

Palavras-chave: computação quântica, algoritmo de Grover, complexidade, ray casting.

CONTENTS

1	INTRODUCTION	1
1.1	Contributions	3
1.2	Document Structure	3
2	STATE OF THE ART / RELATED WORK	5
3	QUANTUM COMPUTING OVERVIEW	7
3.1	Fundamentals	7
3.1.1	Qubits and Superposition	7
3.1.2	Gates and Circuits	8
3.1.3	Reversibility	11
3.1.4	Measurements	11
3.2	Grover's Algorithm	12
4	RAY CASTING	17
4.1	Geometric setup and quantum circuits	17
4.1.1	Geometric Setup	18
4.1.2	Generating the bounds	19
4.1.3	The oracle	21
4.1.4	Inversion about the average	23
4.1.5	Overall circuit	24
4.2	Single Solution	25
4.2.1	Algorithm	25
4.2.2	Results	28
4.3	Multiple Solutions	33
4.3.1	Occlusion	34
4.3.2	Visibility	40
4.4	On a Real Quantum Machine	44
5	CONCLUSION	48
5.1	Prospect for future work	48

LIST OF FIGURES

Figure 1	Single qubit gates. Representation, as a quantum circuit, of the (a) X or NOT gate, (b) alternate representation of the X or NOT gate, (c) Z gate and (d) Hadamard gate.	9
Figure 2	Two possible representations of the CNOT gate in a quantum circuit.	9
Figure 3	Representation of the CZ gate in a quantum circuit. (a) Because both control and target qubits can be exchanged, both these representations can be used. (b) Since both qubits are exchangeable, we can also adopt this simpler representation.	10
Figure 4	CNOT equivalent with 2 Hadamards and 1 CZ.	10
Figure 5	Representation of the Toffoli gate in a quantum circuit.	11
Figure 6	Initial state.	14
Figure 7	Application of the oracle.	14
Figure 8	Application of the diffusion operator.	15
Figure 9		15
Figure 10		15
Figure 11	Diagram of the cases of ray casting studied.	18
Figure 12	Geometric setup for the 2D or non-overlapping case.	19
Figure 13	(a) Quantum circuit for "copying" a state ($ 0\rangle$ or $ 1\rangle$), (b) quantum circuit representing an AND gate and (c) quantum circuit representing an OR gate.	20
Figure 14	Quantum circuit encoding the max_x coordinates in the qubits b_i .	21
Figure 15	Quantum circuit for the oracle.	21
Figure 16	Quantum circuit for the Grover's diffusion operator for the case of 3-qubit primitive indeces.	24
Figure 17	Quantum circuit of the algorithm. (i is not necessarily the same for p , b and aux , it just represents that the quantum registers have multiple qubits)	24
Figure 18	Reference images obtained with the Qiskit simulator for the single solution case. The geometry in (a) is $[(0, 1, 0, 1), (2, 2, 0, 3), (3, 3, 1, 1), (3, 3, 3, 3)]$ and in (b) is $[(1, 3, 1, 2), (6, 6, 1, 4), (0, 3, 7, 7), (7, 7, 0, 0), (1, 2, 4, 5), (4, 4, 0, 2), (4, 4, 4, 7), (7, 7, 7, 7)]$	
Figure 19	Results retrieved with the Qiskit simulator for each pixel in the geometry $[(0, 1, 0, 1), (2, 2, 0, 3), (3, 3, 1, 1), (3, 3, 3, 3)]$ with 2048 shots.	30

Figure 20	Results obtained with the Qiskit simulator for pixel $(1,4)$ in geometry $[(1,3,1,2), (6,6,1,4), (0,3,7,7), (7,7,0,0), (1,2,4,5), (4,4,0,2), (4,4,4,7), (7,7,5,7)]$ with 2048 shots.	31
Figure 21	Scene for multiple solution occlusion case. (a) Depth $z = 1$, (b) depth $z = 2$ and (c) depth $z = 3$.	35
Figure 22	Results for the multiple solution occlusion case with $c = 2$.	35
Figure 23	Results for the multiple solution occlusion case with $c = 3$.	36
Figure 24	Results for the multiple solution occlusion case with $c = 4$.	37
Figure 25	Results for the multiple solution occlusion case, with an 8×8 scene.	38
Figure 26	Scene with geometry $[(1,2,0,0,0), (0,1,2,2,0), (1,1,0,3,1), (3,3,1,3,1), (1,3,2,2,2)]$.	39
Figure 27	Results for pixel $(2,0)$ with $r = 1$ and $r = 2$.	40
Figure 28	Results for pixel $(3,2)$ with $r = 1$ and $r = 2$.	40
Figure 29	Results for pixel $(1,2)$ with $r = 1$ and $r = 2$.	41
Figure 30	Results for the multiple solution visibility case.	44
Figure 31	Connectivity of the qubits in the IBM Q Poughkeepsie real quantum machine.	45
Figure 32	Geometry of the scene to execute on the real quantum machine.	45
Figure 33	Results retrieve from IBM Q's real quantum computer for pixels with an intersection.	46

LIST OF TABLES

Table 1	Table depicting the binary representation of the coordinates min_x and the primitives P_i ; p_i are the qubits representing the indices of the primitives and b_i are the qubits representing the min_x coordinates of the primitives.	20
Table 2	Growth of number of qubits of the circuits for different types of geometries.	31
Table 3	Growth of the depth of the circuits for different types of FULL geometries.	32
Table 4	Growth of the depth of the circuits for different types of NOT FULL geometries.	32
Table 5	Growth of number of gates of the circuits for different types of FULL geometries.	33
Table 6	Growth of number of gates of the circuits for different types of NOT FULL geometries.	33
Table 7	Growth of depth and number of gates depending on number of iterations of Grover's algorithm r in a 4x4 scene.	37
Table 8	Growth of depth and number of gates depending on number of iterations of Grover's algorithm r in a 8x8 scene.	39
Table 9	Comparison between the 4x4 and 8x8 scenes.	39
Table 10	Depth and number of gates for the visibility case.	44
Table 11	Depth and number of gates of circuit executed on the real quantum machine.	46

INTRODUCTION

The field of quantum computing has registered huge developments over the last few years, raising the perspective for this alternative computing paradigm to become practical and advantageous in the long run.

Richard Feynman is believed to have been the first to propose using quantum mechanics as a model for computation in 1982 [1]. Theoretical results soon followed, with, for example, the proposal of the quantum Turing machine by Deutsch in 1985 [2] and algorithms that lie in the heart of most quantum programs, including Grover's for unstructured searching [3], Shor's for prime factorization [4] and the Quantum Fourier Transform.

In recent years, technological advances have made it possible for some companies to make quantum computers available to the community. These machines are in their beginning stages and consequentially have many limitations still. Aspects like the size of the circuits or execution time of the problems they can solve are yet to be improved in order for quantum computing to become a mainstream technology. These limitations are due to limited coherence times, reduced number of qubits and significant error rates at the gate level. Despite all these constraints in producing and running quantum programs, the number of researchers, practitioners and companies experimenting with prototype applications to solve an ever growing variety of problems has been increasing exponentially. This rising interest on quantum computing is largely supported by a huge investment, both from private companies and research funding institutions all around the world [5,6].

It has been claimed that quantum computing has the potential to seriously impact many scientific fields, facilitating the solving of problems so complex that traditional computers, even giant supercomputers, could never handle. But how is this really possible? An n -qubit quantum computation evolves on an exponential state space, representing at each instant a linear superposition of 2^n states. Each computing step leads the system to another superposition, acting on the 2^n states simultaneously without any resources replication. On the other hand, a classical n -bit system represents, at each instant, a single state out of the 2^n possible states and a computing step leads to a single subsequent state. It is this exponential quantum parallelism that holds the promise of the quantum advantage over classical computing. It is the ability of a quantum register to exist in a superposition of states that allows for this quantum parallelism. A function performed once on a register

in a superposition of states is performed on each of the components of the superposition. Since the number of possible states is 2^n , where n is the number of qubits in the quantum register, you can perform, in one operation on a quantum computer, what would take an exponential number of operations on a classical computer. However, the results of such quantum operation are not accessible, meaning that even though a quantum computation evolves on a state space exponentially larger than classical computing, it is not accessible; quantum algorithms are required to maximize the probability of reading results of interest. For some problems, quantum algorithms have been found which exhibit better complexity than classical alternatives, allowing for the efficient computation of otherwise intractable sized problems.

One problem that is a good candidate to be adapted to a quantum algorithm because of its time consumption is the ray casting problem. This can be a very computationally demanding task and can potentially benefit from the time advantage in computation that a quantum algorithm may provide. Ray casting is a well known rendering technique, consisting of shooting a ray from a point along a given direction to determine i) whether that direction is occluded (i.e., if the ray intersects, at least, one geometric primitive - occlusion problem), or ii) what is the visible geometric primitive along the ray's direction (i.e., among all primitives intersected by the ray, which one is nearer to the ray's origin - visibility problem). If there are N geometric primitives and if these are not ordered in any manner, then classical ray casting requires testing intersection against all primitives, with complexity $\mathcal{O}(N)$. With an ordered structure the classical algorithm achieves a complexity of $\mathcal{O}(\log N)$ but here we're trying to improve the simplest classical algorithm's complexity using a quantum algorithm using a non-structured scene as well.

Grover's algorithm for searching on an N elements unstructured domain exhibits $\mathcal{O}(\sqrt{N})$ time complexity, as opposed to $\mathcal{O}(N)$ in the classical setting. In this work we demonstrate the application of Grover's algorithm to ray casting, addressing both the occlusion and the visibility problems, with worst-case complexity $\mathcal{O}(\sqrt{N})$. The idea of using Grover's quantum search algorithm to solve this problem has been suggested before (see Section 2 for more details), however no practical implementation or results have been ever presented (to the best of the author's knowledge). So we took that leap and we propose concrete implementations and present results obtained both from a simulator and from a real quantum computer. Advantages and limitations of the proposed approach are discussed, as well as improvements that can be performed and the next steps for future work in this area. All experiments, both simulations and real quantum computer executions, were performed using IBM's quantum computing framework, Qiskit¹ and the IBM Q Network 20 qubit Poughkeepsie machine.

¹ <https://qiskit.org/>

Current circuit-based quantum computers provide no functional units to perform even the most fundamental arithmetic operations to mathematical data types, such as integers or floating point numbers. In fact, the program consists essentially on a list of basic transformations, referred to gates, to apply to the qubits. Supporting complex operations, such as non-integer number representations or trigonometric functions, requires implementing them at the gate level. Besides the obvious programming overhead, these additional gates could increase dramatically the circuit depth and the number of required qubits, pushing the limits well beyond what is possible with today's noisy and limited scale quantum computers. In order to avoid such overheads, in this work we only work with integers, the view plane is contained within the $z = 0$ plane, all geometric primitives are axis-aligned rectangles parallel to the view plane and the rays are perpendicular to the view plane and parallel to the z -axis (thus the orthographic projection); all coordinates ((x, y) for pixels and rays, (x, y, z) for points in space) are integers. These restrictions facilitate the development of the quantum program and make it possible to use current simulators and real machines. There are, however, no theoretical reasons why floating point numbers and trigonometric functions cannot be used. They just wouldn't be feasible on current hardware given the limitations imposed by the technology's state of the art.

1.1 CONTRIBUTIONS

The main contribution of this work is the fact that we present a practical example of a quantum ray tracer. The idea of a quantum ray tracer using Grover's Algorithm is not new and theoretical proposals have been presented in the past (more on Section 2). In this work, we're trying to determine the possibility of a quantum ray tracer to be more efficient than a classical ray tracer but obviously quantum computation is still in its early stages and classical computer algorithms for ray tracing are very effective nowadays, thus we won't be able to present and accurately affirm we have found a better method for ray tracing. The goal is to present the possibility for a new method to be perfected and used in the future. The other objective of this dissertation is to produce real results in today's available quantum machines. Because quantum technology is still developing, our results are mainly retrieved from quantum simulators and afterwards analyzed to determine the possibility and effectiveness of running those programs on a real quantum computer.

1.2 DOCUMENT STRUCTURE

This document is divided into 5 chapters. The first chapter introduces this dissertation and presents this work's contributions. Chapter 2 showcases some works, mostly theoretical, about the topic of this dissertation. Chapter 3 details the basic notions about quantum

computing necessary to understand this work, like qubits, gates, quantum circuits, and a section describing Grover's algorithm. Chapter 4 is where we present the work we've done; a first section entails the more theoretical aspects our work and the next sections detail the results obtained. Finally, Chapter 5 concludes this dissertation and provides some possibilities for future work.

STATE OF THE ART / RELATED WORK

This chapter reviews some important work done regarding the theme of this dissertation.

This work is based on the use of Grover's Algorithm, a quantum search algorithm created in 1996 by Lov Grover [3].

In 2001, Andrew Glassner discussed quantum computing in his notebook [7–9] stating that anything involving searching for a minimum or maximum is a good candidate for quantum algorithms because Grover's algorithm can speed this up significantly. He presents two possible contenders for this kind of technique, the first being the Z-buffer problem. To solve it, a superposition over all the polygons and respective depth would be created and Grover used to identify the minimum depth. The other contender is ray tracing and for this a superposition over all spheres and respective parameters (a, b, c) would be created; equation $at^2 + bt + c = 0$ would then be solved simultaneously for all spheres and Grover used to locate the minimum positive t . For both problems a quadratic advantage is obtained over an unordered classical search.

In their 2009 book [10], Marco Lanzagorta and Jeffrey Uhlmann describe the use of Grover's algorithm for a number of computational geometry problems, including nearest neighbor queries, object-object intersection, Z-buffering, ray tracing, radiosity and level of detail, among others.

In a more recent related work, Simona Caraiman [11] also proposes to apply Grover's algorithm to the Z-buffer and ray tracing problems, including algorithms for both.

Two important works for this dissertation are the ones used to solve the multiple solution occlusion and visibility problems. Michel Boyer et al. [12] present a way to determine the number of iterations of Grover's algorithm necessary to achieve almost certainty of finding the answer when the number of solutions is unknown. They also present a way to count the number of solutions beforehand so that the number of times Grover's algorithm needs to be applied is easily calculated. Durr and Hoyer [13] demonstrate a quantum algorithm for finding the minimum, which we'll need to find the primitive with the minimum depth.

A more practical and experimental work is the problem is addressed by Johnston [14]. The author performs pixel level supersampling by resorting to the quantum amplitude estimation algorithm to numerically integrate sub pixel samples through a combination of a quantum algorithm and a classical lookup table. This improved results compared to those

obtained with Monte Carlo integration for the same number of samples. These results are obtained using a simulator, an IBM five qubit machine, and a photonic quantum computer. This paper focused on supersampling pixels based on a known underlying signal, whereas our work computes this signal from a scene description.

All the above cited works, except the last, maintain the discussion at the algorithmic and complexity analysis levels. No simulation, implementation or execution of the proposed algorithms is performed and no real experimental results are presented. In this dissertation we present an actual implementation and practical results for the ray casting problem.

Our results are retrieved from the IBM Q Experience by IBM where they provide simulators for quantum machines and also real quantum machines where we can submit our circuits.¹ Other existing platforms of cloud-based quantum computing are Forest by Rigetti Computing, which includes a programming language, development tools and example algorithms.², Azure by Microsoft which is a diverse set of quantum services, ranging from pre-built solutions to software and quantum hardware³, Quantum in the Cloud by The University of Bristol, which consists of a quantum simulator and a 4-qubit optical quantum system⁴, Quantum Playground by Google, which features a simulator with a simple interface, its own scripting language with debugging and 3D quantum state visualization features⁵, Quantum Inspire by Qutech⁶, Forge by QC Ware⁷, among others.

¹ <https://www.ibm.com/quantum-computing/>

² <https://www.rigetti.com/systems>

³ <https://azure.microsoft.com/en-gb/services/quantum/>

⁴ <http://www.bristol.ac.uk/physics/research/quantum/engagement/qcloud/>

⁵ <https://opensource.google/projects/quantum-computing-playground>

⁶ <https://www.quantum-inspire.com>

⁷ <https://forge.qcware.com>

QUANTUM COMPUTING OVERVIEW

This chapter will present the notions in quantum computing necessary to understand this work. In a first section, the notions and properties of quantum computing will be addressed, alongside quantum gates and their representation. A second section will be dedicated to Grover's algorithm.

3.1 FUNDAMENTALS

3.1.1 Qubits and Superposition

Classical computers are made up of bits that, at any given moment, can represent a zero or a one. But a quantum computer works with qubits (quantum bits). The qubit is the basic unit of quantum information – the quantum version of the classical binary bit. A qubit can represent a zero or a one as well, but also a superposition of these two basis states.

Thus, 1 qubit can be in a superposition of 2 states, 2 qubits can be in a superposition of 4 states, 3 qubits in a superposition of 8 states, etc. Each of these states has an associated amplitude. The probability of a state being measured is its amplitude squared, thus the sum of all the states' amplitudes squared must equal to 1.

$$\begin{aligned} |q\rangle &= \alpha |0\rangle + \beta |1\rangle \\ \alpha, \beta &\in \mathbb{C} \\ |\alpha|^2 + |\beta|^2 &= 1 \end{aligned}$$

Generally speaking, if we have a quantum computer with n qubits, we have a superposition of $N = 2^n$ basis states (a classic computer can only be in one of these states at a single time):

$$\begin{aligned} |\Psi\rangle &= \sum_{i=0}^{N-1} \alpha_i |i\rangle \\ \alpha_i &\in \mathbb{C} \\ \sum_{i=0}^{N-1} |\alpha_i|^2 &= 1 \end{aligned}$$

Any computational operation over this superposition will simultaneously act over the 2^n states, in what is referred to as exponential quantum parallelism. This means that applying a function $f(\cdot)$ once to superposition $|\Psi\rangle$ effectively results in a new superposition $|\Psi'\rangle$ containing all $N = 2^n$ values of $f(|\Psi\rangle)$.

3.1.2 Gates and Circuits

Whereas with classical circuits signals flow along the circuit, with quantum circuits, time flows along the circuit, from left to right – qubits do not flow, rather gate operations are applied onto the qubits.

The building blocks of quantum circuits are quantum gates, like classical logic gates are for conventional digital circuits. In a quantum gate the number of input qubits is always the same as the output qubits because the operations have to be reversible (See section 3.1.3).

A gate acting on n qubits is represented by a $2^n \times 2^n$ unitary matrix and the general quantum state of a qubit can be represented by a linear superposition of its two orthonormal basis states:

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

These two orthonormal basis states span the two-dimensional linear vector (Hilbert) space of the qubit and are denominated the computational basis.

A very simple quantum gate is the Pauli-X gate. This gate acts on a single qubit and is the equivalent of the NOT gate for classical computers. It maps $|0\rangle$ to $|1\rangle$ and $|1\rangle$ to $|0\rangle$. It is also commonly designated as bit-flip and it's represented by the Pauli-X matrix:

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

To obtain the result of a gate applied to a qubit we multiply the vector and the matrix.

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle$$

As we can see, the NOT gate changed the state $|0\rangle$ to $|1\rangle$.

Another Pauli gate is the Pauli-Z gate. This gate leaves the basis state $|0\rangle$ unchanged and maps $|1\rangle$ to $-|1\rangle$.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

A very important quantum gate is the Hadamard gate. This gate also acts on a single qubit and it's used to create superposition. Its matrix is the following:

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

It maps $|0\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, which means that a measurement of this qubit has the same probability of yielding a 0 or a 1.

The next figure shows the representation of the single qubit gates mentioned above:

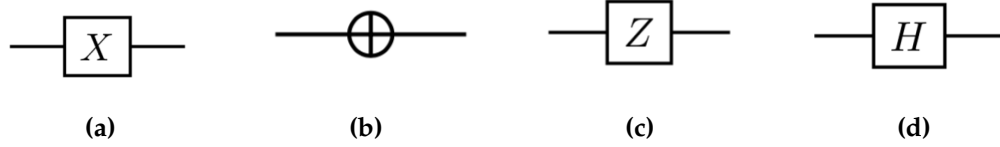


Figure 1: Single qubit gates. Representation, as a quantum circuit, of the (a) X or NOT gate, (b) alternate representation of the X or NOT gate, (c) Z gate and (d) Hadamard gate.

Transitioning to 2-qubit gates, a crucial quantum gate is the Controlled-NOT (CNOT) gate (also known as Controlled-X gate). This gate acts by flipping the second qubit (target qubit) if and only if the first qubit (control qubit) is $|1\rangle$. It is represented in Figure 2 and its matrix is defined as:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

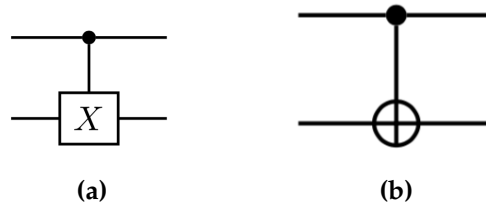


Figure 2: Two possible representations of the CNOT gate in a quantum circuit.

There is also the Controlled-Z gate, presented in Figure 3, and, in this case, the control and target roles can be interchanged as this gate is symmetrical. It induces a sign shift for states where both qubits are $|1\rangle$, thus any qubit can be said to be the control. Its matrix is given by:

$$\text{Controlled-Z} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

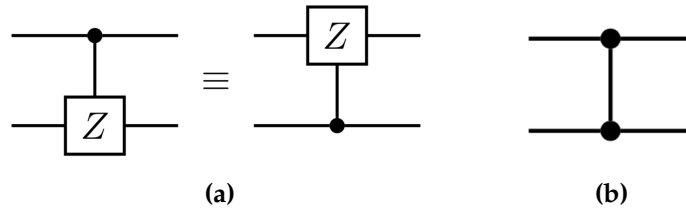


Figure 3: Representation of the CZ gate in a quantum circuit. (a) Because both control and target qubits can be exchanged, both these representations can be used. (b) Since both qubits are exchangeable, we can also adopt this simpler representation.

An important fact about these gates is that the CX (CNOT) operation can be described in terms of a CZ gate and 2 Hadamard gates (Figure 4). The X gate can be decomposed as a sequence of three single qubit gates (2 Hadamards and 1 CZ) in the following way: $X = HZH$. Thus, the CX gate can be decomposed as: $CX = (I \otimes H)CZ(I \otimes H)$. When the control is $|0\rangle$, the two Hadamard gates cancel each other and, when it is $|1\rangle$, the combination of gates acts as a NOT [15].

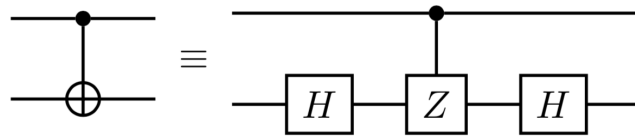


Figure 4: CNOT equivalent with 2 Hadamards and 1 CZ.

Another very useful gate is the Toffoli gate. This gate acts on 3 qubits and it's also known as the CCNOT gate. It works like the CNOT but with 2 control qubits; if both these qubits are set to $|1\rangle$, the third qubit (target qubit) is inverted, otherwise it stays unchanged.

This gate, presented in Figure 5, has to be constructed using the single and 2-qubit gates presented above and it can be used with n control qubits.

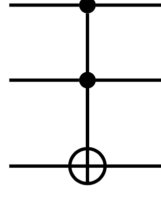


Figure 5: Representation of the Toffoli gate in a quantum circuit.

3.1.3 Reversibility

Quantum mechanics' laws require that all operations performed on qubits be unitary and reversible. A unitary operation ensures that the norm of the vector of basis states coefficients α_i is maintained and equal to 1 after the transformation; this is required since the vector of α_i 's is in fact a probability distribution thus the sum of probabilities of all possible outcomes of any event always equals 1. No information gets lost on the transition from one configuration to another. Reversibility is a consequence of the unitary requirement and means that, given the outputs of the transformation, its inputs can be known. From the ending point we can reach the starting point: the computation is reversible.

We can check if an operation is reversible by multiplying a matrix representing an operation by its conjugate transpose. If it equals the Identity matrix then the operation is unitary, and therefore, reversible. Summing up, a matrix U is unitary if $U \times U^\dagger = U^\dagger \times U = I$. Checking with the Pauli-X matrix we can see that the operation is indeed reversible:

$$X \times X^\dagger = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I$$

Reversibility has several consequences at the circuit level: all quantum gates, which operate over qubits, must have the same number of inputs and outputs; this often results in circuits with a larger number of gates and more qubits than would be required of their classical counterpart. Information is never destroyed, since the computation can be reversed.

3.1.4 Measurements

Even though the quantum computation evolves on an exponentially large state space computing all solutions simultaneously, this space is not accessible. The superposition state holds until the moment of a measurement. When a measurement on the computational basis is made the result is always a zero or a one for each qubit. Measurements can be made using other bases using quantum gates to change the basis of the qubits. Upon measurement, the superposition $|\Psi\rangle$ collapses into one basis state $|i\rangle$, among all basis states

included on $|\Psi\rangle$, with probability $|\alpha_i|^2$. Contrary to quantum operations, a measurement is irreversible - all the information on the superposition is lost with a measurement and the register then behaves as classical data. The role of quantum algorithms is to maximize the probability of measuring the desirable states within a superposition and to postpone measurements (i.e., reading quantum data) until the last step of the algorithm.

3.2 GROVER'S ALGORITHM

Created in 1996 by Lov Grover [3], Grover's Algorithm is a quantum algorithm that performs a search through an unstructured set of $N = 2^n$ elements to find a unique item that satisfies a certain condition. It does this using just $O(\sqrt{N})$ operations which represents a quadratic gain over the $O(N)$ complexity of a classical algorithm. Unlike other quantum algorithms, which may provide exponential speedup over their classical counterparts, Grover's algorithm provides "only" a quadratic speedup. However, even quadratic speedup is considerable when N is large.

The search is called "unstructured" because we are given no guarantees as to how the database is ordered. If we were given a sorted database, for instance, then we could perform binary search to find an element in logarithmic time. Instead, we have no prior knowledge about the contents of the database. With classical circuits, we cannot do better than performing a linear number of queries to find the target element.

Like many quantum algorithms, Grover's algorithm is probabilistic. This means that it gives the correct answer with a probability of less than 1. Though there is technically no upper bound on the number of repetitions that might be needed before the correct answer is obtained, the expected number of repetitions is a constant factor that does not grow with N .

The setup of the problem is as follows:

For n qubits and $N = 2^n$ elements, let $f : \{0, 1, \dots, 2^n - 1\} \rightarrow \{0, 1\}$ such that

$$f = \begin{cases} 0 & \text{if } x \neq x^* \\ 1 & \text{if } x = x^* \end{cases}$$

The algorithm returns, with high probability, the unique state $x^* \in \{0, \dots, N - 1\} : f(x^*) = 1$.

The first step of the Grover's algorithm is to create a uniform superposition over all basis states by applying Hadamard gates on our qubits:

$$|s\rangle = \hat{H}|0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$$

The second step of the algorithm is to apply the oracle. The oracle is made up of unitary operations and is going to mark the state we want to find. It marks the state by flipping its amplitude. The oracle can be defined as a unitary matrix $\hat{O}$ that acts on the states in the following way:

$$\hat{O}|x\rangle = (-1)^{f(x)}|x\rangle$$

We see that if $|x\rangle$ is an unmarked item, the oracle does nothing to the state. However, when we apply the oracle to the basis state $|x^*\rangle$, it maps $\hat{O}|x^*\rangle = -|x^*\rangle$. Thus, after applying the oracle, our superposition state looks like this:

$$\hat{O}|s\rangle = \frac{1}{\sqrt{N}} \hat{O} \sum_{x=0}^{N-1} |x\rangle = \frac{1}{\sqrt{N}} \left[\left(\sum_{x \neq x^*} |x\rangle \right) - |x^*\rangle \right]$$

The superposition state stays the same except the state we want to find that has its amplitude negated. The next step is to increase this state's amplitude. The procedure is called amplitude amplification and it raises the amplitude of the marked item, which shrinks the other items' amplitude, so that measuring the final state will return the right item with near-certainty. This is performed by an operator $\hat{D}$ called Grover's diffusion operator. It is defined as:

$$\hat{D} = 2|s\rangle\langle s| - \hat{I}$$

This algorithm has a nice geometrical interpretation in terms of two reflections, which generate a rotation in a two-dimensional plane. The only two special states we need to consider are the state we're looking for, $|x^*\rangle$, and the uniform superposition $|s\rangle$. These two vectors span a two-dimensional plane in the vector space $\mathbb{C}^N$.

The states $|s\rangle$ and $|x^*\rangle$ form a basis of this subspace but they're not orthogonal, their inner product is $\frac{1}{\sqrt{2^n}}$. We can define a state $|s''\rangle$ that is orthogonal with $|x^*\rangle$ by subtracting off the $|x^*\rangle$ component of $|s\rangle$. This way their inner product is zero. However, $|s''\rangle$ isn't normalized, its inner product with itself isn't equal to 1. This is how we get to $|s'\rangle$, which is parallel to $|s''\rangle$, and perpendicular to $|x^*\rangle$ but has been normalized.

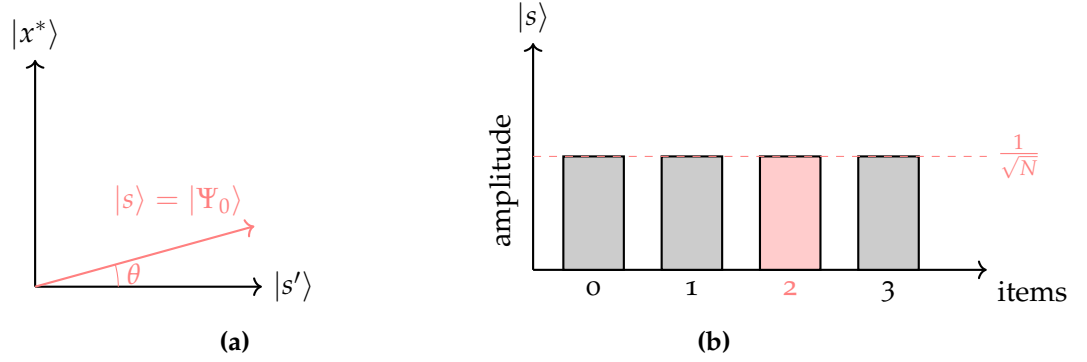


Figure 6: Initial state.

Figure 6 represents the initial state of uniform superposition $|s\rangle = \Psi_0$. In (a) we have represented the vectors $|x^*\rangle$, the state we want to find, $|s\rangle$, the superposition state and $|s'\rangle$, the superposition state without the $|x^*\rangle$ state; (b) is a bar graph of the amplitudes of the state $|s\rangle$ for the case $N = 4$. In the beginning all states have the same amplitude and the dashed red line represents the average amplitude.

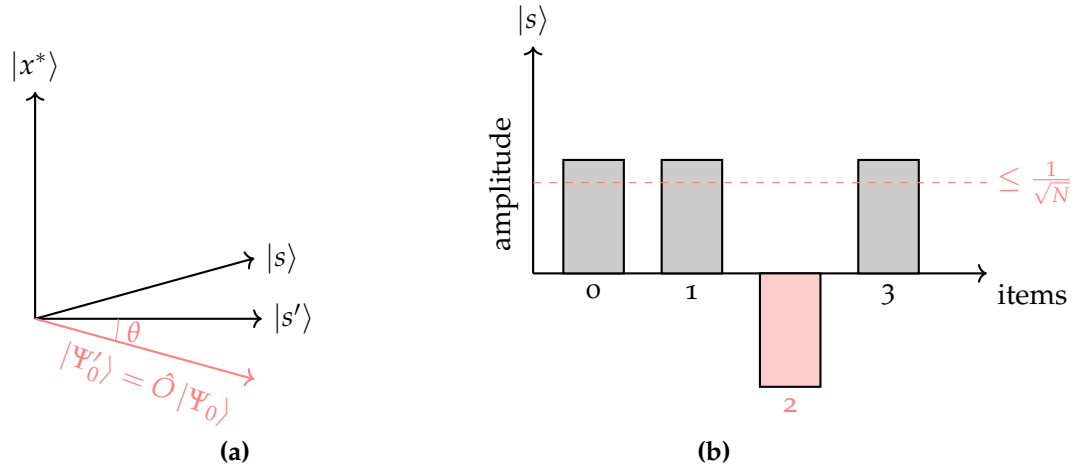


Figure 7: Application of the oracle.

Figure 7 represents the application of the oracle $\hat{O}$ to the initial state. This works by performing a reflection around $|s'\rangle$ ((a)). As mentioned before, the oracle negates the amplitude of the targeted state as we can see in (b); consequently the average amplitude is decreased. But marking the target state does not change its amplitude, this is achieved in the next step.

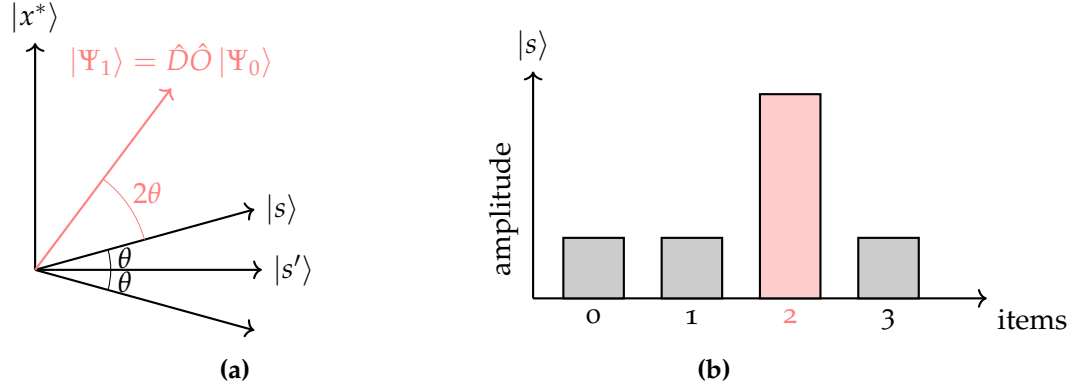


Figure 8: Application of the diffusion operator.

The last step of the algorithm before measurement is to perform the inversion about the average. This is shown in Figure 8. In (a) we can see the application of the diffusion operator $\hat{D} = 2|s\rangle\langle s| - \hat{I}$ and that it corresponds to a reflection around $|s\rangle$. We can observe that these 2 reflections rotated the initial state $|s\rangle$ closer to the right item $|x^*\rangle$. In (b) we see that the amplitudes have been reflected around the average which resulted in an increase of the amplitude of the right state $|x^*\rangle$ and a decrease of all other states.

One important thing to know is how many times we need to perform this rotation $\hat{D}\hat{O}$. For this we'll start by noting that $|s\rangle = \sin\theta|x^*\rangle + \cos\theta|s'\rangle$ (Figure 9). Since the probability is the amplitude squared, we can see that the probability of the right state $|x^*\rangle$ is $\sin^2\theta$. To maximize this probability we need to maximize $\sin^2\theta$ which is the same as making $\theta = \frac{\pi}{2}$.

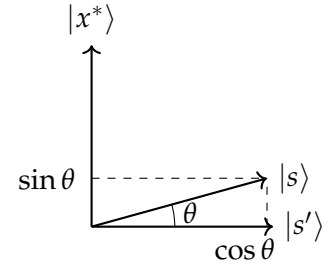


Figure 9

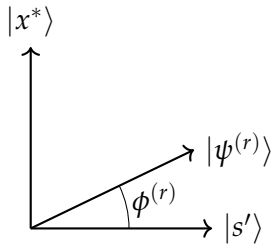


Figure 10

Let r be the number of times a rotation is made and $\phi^{(r)}$ be the angle between $|s'\rangle$ and $|\psi^{(r)}\rangle$ (Figure 10). We know, from Figure 6, that $\phi^{(0)} = \theta$ and from Figure 8 that $\phi^{(1)} = \theta + 2\theta$. In the next repetition we would have $\phi^{(2)} = \theta + 4\theta$, thus $\phi^{(r)} = (1 + 2r)\theta$. We know that in order to maximize the probability of the target state this angle must be equal to $\frac{\pi}{2}$ and after some calculations we reach $r = \frac{\pi}{4\theta} - \frac{1}{2}$.

Now we need to find an expression of r in function of N to see how the number of repetitions varies with the number of items in the set. From Figure 9 we see that:

$$\cos \theta = \langle s' | s \rangle = \underbrace{(N-1)}_{\substack{\text{elements} \\ \text{in common}}} \times \underbrace{\frac{1}{\sqrt{N-1}}}_{\substack{\text{coefficient of} \\ \text{states in } s'}} \times \underbrace{\frac{1}{\sqrt{N}}}_{\substack{\text{coefficient of} \\ \text{states in } s}} = \sqrt{\frac{N-1}{N}}$$

Using the equation $\sin^2 \theta + \cos^2 \theta = 1$ we arrive to the expression $\sin \theta = \frac{1}{\sqrt{N}} \Leftrightarrow \theta = \sin^{-1} \left(\frac{1}{\sqrt{N}} \right)$. Using the small angle approximation we can say that $\sin^{-1} \left(\frac{1}{\sqrt{N}} \right) \approx \frac{1}{\sqrt{N}}$. Now we can finally replace θ by $\frac{1}{\sqrt{N}}$ in the expression of r obtaining:

$$r = \frac{\pi}{4 \frac{1}{\sqrt{N}}} - \frac{1}{2} = \frac{\pi \sqrt{N}}{4} - \frac{1}{2} \approx \frac{\pi}{4} \sqrt{N} \propto \sqrt{N}$$

Finally we can see the number of repetitions of Grover's algorithm is proportional to $\sqrt{N}$ resulting in a quadratic speedup from the classical case. This result applies to large N and when there is only one solution. For multiple solutions, the number of repetitions is $\propto \sqrt{\frac{N}{t}}$, t being the number of solutions. Performing more iterations than the ideal r actually reduces the probability of measuring a correct state; successful application of Grover's algorithm depends on the ability to compute r and, consequently, the previous knowledge of t . When t is unknown it can be estimated or we can use an algorithm of exponential search like we utilize in this dissertation (Q_Search).

RAY CASTING

The first ray casting algorithm used for rendering was presented by Arthur Appel in 1968 [17]. Ray casting is a rendering technique used in computer graphics and computational geometry. It is capable of creating a three-dimensional perspective in a two-dimensional map. In this work we make use of an orthographic camera. This uses orthographic projection which is a means of representing three-dimensional objects in two dimensions. It is a form of parallel projection, in which all the projection lines are orthogonal to the projection plane. Ray casting is considered one of the most basic graphics-rendering algorithms. The idea behind it is to trace rays from the eye, one per pixel, and find the closest object blocking the path of that ray to form an image.

We divided the ray casting problem in 2 cases: the single solution case and the multiple solution case. In the single solution case, one only primitive can be intersected by a ray, there is no overlapping of primitives. In this case, the possible number of solutions, for each ray, is 0 (ray is not occluded) or 1 (ray is occluded). In the multiple solution case, overlapping of primitives is possible and, therefore, one ray can intersect more than one primitive, resulting in a scenario of multiple possible solutions.

Within the context of the multiple solutions case, we will address two different types of problem in the ray tracing context: to determine if a point is illuminated or not – occlusion problem, and what point is visible along a given direction – visibility problem. For the occlusion case, we just want to determine whether the ray intersects any primitive (no matter which), whereas for the visibility case, we want to determine which primitive, amongst all those intersected by the ray, is closer to the ray's origin. We can see a diagram of these cases in Figure 11.

4.1 GEOMETRIC SETUP AND QUANTUM CIRCUITS

This section explains the geometric setup of the problem and the circuits used to generate the bounds (coordinates of the primitives), for the oracle and for the Grover's diffusion operator. These circuits will be used for the 3 different cases of this work (single solution,

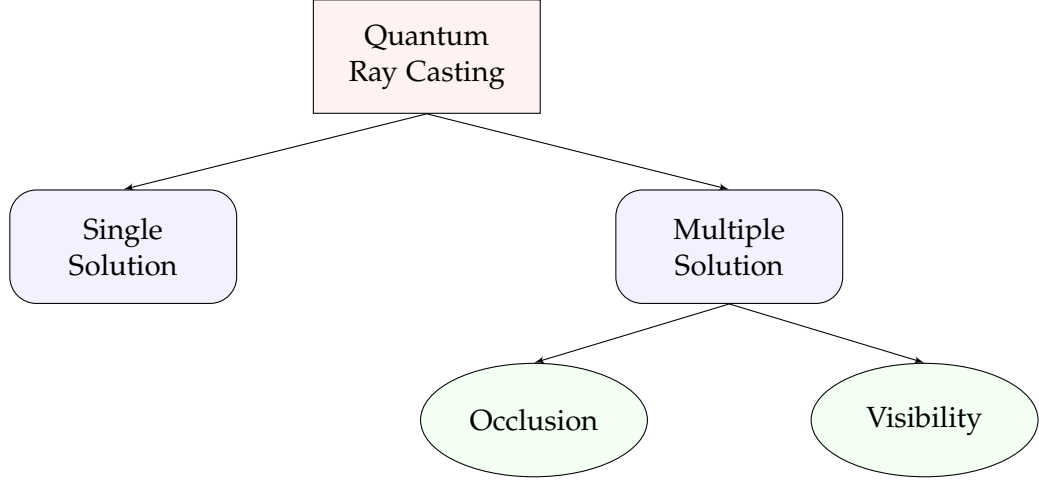


Figure 11: Diagram of the cases of ray casting studied.

multiple solution occlusion and multiple solution visibility), with exception of the oracle which will have to be extended in the last case.

4.1.1 Geometric Setup

In the geometric setup of this problem, as described in the introduction, the view plane is contained in the $z = 0$ plane, the primitives are rectangles parallel to the view plane and the primary rays are parallel among themselves and the z axis; there is a one to one correspondence between pixels in the image plane and primary rays. All coordinates are positive integers. The primitives are rectangles characterized by five parameters: $(min_x, max_x, min_y, max_y, z)$.

There are two different types of geometric setup: the setup for the single solution case and the setup for the multiple solution case. In the single solution case, at most, one primitive projects onto one pixel (x, y) since there is no overlap with the primitives; there exists only one primitive in each ray's direction. Here, the z coordinate, representing the depth, is irrelevant and this can be seen as a 2D case. An example of this setup is shown in Figure 12.

For the multiple solution case, a third dimension is added to the primitives; z is no longer irrelevant here. This makes possible the overlap of primitives in different depths since one ray can encounter more than one primitive along its direction. This represents a 3D case.

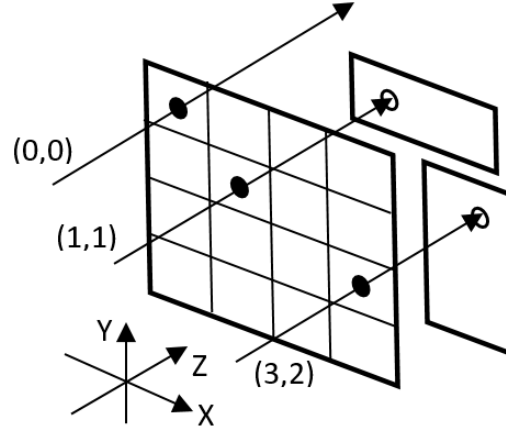


Figure 12: Geometric setup for the 2D or non-overlapping case.

4.1.2 Generating the bounds

It's necessary to generate the minimum and maximum X and Y coordinates and Z coordinate of a primitive from its index. The initial state of all qubits on a quantum computer is $|0\rangle$ and there's, currently, no mechanism to load data into the machine, i.e., to set the qubits on an initial state different from $|0\rangle$. The only way this can be achieved is by setting a circuit that transforms the qubits into the desired state. This has the disadvantage of increasing circuit depth and, therefore, compromise both running time and noise levels.

The primitives' parameters are generated using such an initializing circuit, which takes as input the primitive index. On a quantum machine, the index will be a superposition of all possible indices and this initializing circuit can, eventually, process all primitives in parallel. This is done by the transforms mX, MX, mY, MY and also Z in the case where there's depth. These are functions made by us that receive the IDs of the primitives defined for each scene and transform them into a quantum circuit that is calculated using basic Boolean simplification techniques (seen in the table below). Here's an example with $N = 4$ primitives and the coordinates going from 0 to 15, thus $n = \log_2 4 = 2$ (number of qubits necessary to represent the primitives) and $b = \log_2 16 = 4$ (number of qubits necessary to represent the bounds). Imagine the following geometry where we're only addressing the min_x coordinates leaving all others with the symbol '-' to indicate their value here is not relevant: $[(1, -, -, -, -), (5, -, -, -, -), (9, -, -, -, -), (14, -, -, -, -)]$.

From this table, and using Boolean simplification techniques, we can define the min_x coordinates in terms of the qubits representing the primitives:

	p_1	p_0	b_3	b_2	b_1	b_0
P_0	0	0	0	0	0	1
P_1	0	1	0	1	0	1
P_2	1	0	1	0	0	1
P_3	1	1	1	1	1	0

Table 1: Table depicting the binary representation of the coordinates min_x and the primitives P_i ; p_i are the qubits representing the indices of the primitives and b_i are the qubits representing the min_x coordinates of the primitives.

$$b_3 = P_2 + P_3 = p_1\bar{p}_0 + p_1p_0 = p_1$$

$$b_2 = P_1 + P_3 = \bar{p}_1p_0 + p_1p_0 = p_0$$

$$b_1 = P_3 = p_1p_0$$

$$b_0 = P_0 + P_1 + P_2 = \bar{p}_1\bar{p}_0 + \bar{p}_1p_0 + \bar{p}_1p_0 = \bar{p}_1p_0 + \bar{p}_0 = \bar{p}_1\bar{p}_0 + \bar{p}_1p_0 + \bar{p}_0 = \bar{p}_1 + \bar{p}_0$$

The complexity of these operations was not studied in depth in this work although, in the future, its calculation is important to understand if it doesn't lose the complexity advantage of Grover's algorithm over classical search algorithms.

After having this representation of min_x encoded in the quantum register $|b\rangle$, which we designate the bounds, it's needed to represent this as a quantum circuit, and for that we need to build the quantum circuits equivalent to the above simplified Boolean expressions.

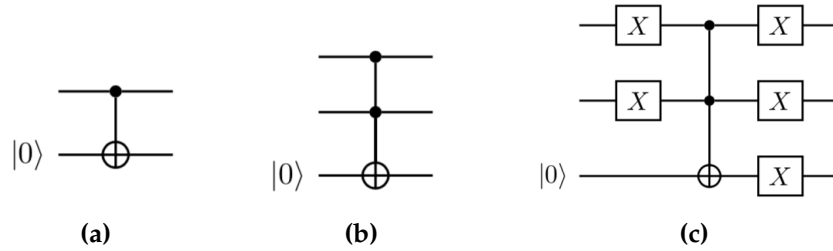


Figure 13: (a) Quantum circuit for "copying" a state ($|0\rangle$ or $|1\rangle$), (b) quantum circuit representing an AND gate and (c) quantum circuit representing an OR gate.

In Figure 13 we see how we can build the quantum circuits to apply the Boolean operations necessary to represent the coordinates as Boolean expressions. The circuit in (a) is a CNOT gate and works by changing the target qubit to $|1\rangle$ if the control qubit is $|1\rangle$ and keeps the target qubit unchanged, meaning $|0\rangle$, if the control qubit is $|0\rangle$; note that both qubits become entangled, meaning that a transformation on one state will change the other state as well. In (b) we have a Toffoli gate which works as an AND gate; if and only if both

control qubits are $|1\rangle$ the target qubit changes to $|1\rangle$, which is exactly what an AND gate does. And finally we have the representation of an OR gate; an OR gate can be defined as an AND gate by negating the inputs and the output of the AND gate, which is what is represented in (c).

With this we can now build a circuit that encodes the max_x coordinates in the qubits b_i (Figure 14). This is done for every coordinate, $(min_x, max_x, min_y, max_y, z)$; circuits to generate bounds with more complex Boolean expressions may require additional, auxiliary, qubits.

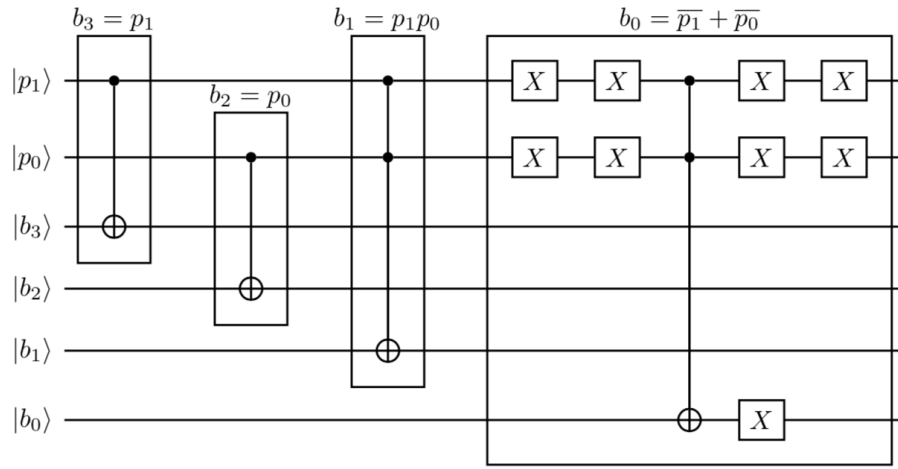


Figure 14: Quantum circuit encoding the max_x coordinates in the qubits b_i .

4.1.3 The oracle

Now we'll take a look at the oracle. We know that a ray (x, y) intersects a primitive if $min_x \leq x \leq max_x \wedge min_y \leq y \leq max_y$ and this is the oracle's job.

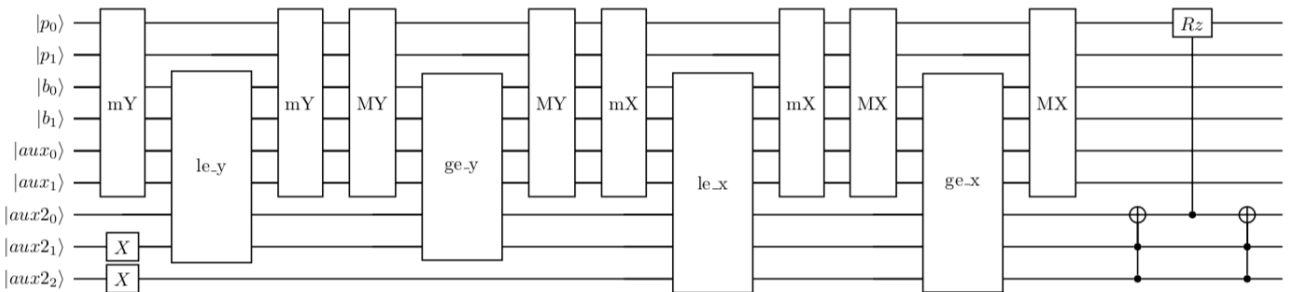


Figure 15: Quantum circuit for the oracle.

If we follow Figure 15 we see the oracle encoded in the ge and le operators that compare the bounds with the ray's coordinates (x, y) , which are hardwired into the circuit. The process described in the previous section to generate the bounds is encoded in the $\{m|M\}\{X|Y\}$ operators. Here, we have 2 options: using different qubits for each of the primitives' coordinates or reusing the same qubits for all primitives. The former would require more qubits (4 times more, 5 times more if the z coordinate is also taken into account), the latter allows for using much less qubits, but increases the circuit depth, since each of the coordinates has to be generated, compared to the ray's x and y values and then reverted. For this work we selected the latter, since the number of qubits that can be used, on both the simulator and the real machines, is rather limited.

Comparison results for x and y are stored onto $aux2_2$ and $aux2_1$ respectively (which have been negated, this will be addressed later) after applying the le_x, ge_x, le_y, ge_y operators which change the $aux2$ qubits' state if, for example, the min_x coordinate of a primitive is lower or equal than the ray's x coordinate (le_x). These 2 resulting qubits are then anded together onto $aux2_0$, using a Toffoli gate. The latter will thus be $|1\rangle$ for the primitives intersected by the ray and $|0\rangle$ for the remaining (the whole circuit is on a superposition over all possible states of $|p\rangle$: 4 basis states in this example). After each of the comparison operators, the bounds generator operator appears again: this is required to reverse the state computed onto $|b\rangle$, resetting its value to $|00\rangle$ before it is used again for the next computation. Such is the nature of quantum reversible computing: all computations have to be undone before the qubits can be used again by some other operator. Finally, a rotation over the Z axis is applied to $|p\rangle$ conditioned to $|aux2_0\rangle$ being $|1\rangle$, thus flipping the sign of the coefficients of those primitives which intersect ray (x, y) .

After the above described circuit, the state of $|aux2\rangle$ hasn't been reversed to $|000\rangle$. However, all ancillary qubits have to be reversed into the original state. This implies repeating the whole circuit in reversed order. The last Toffoli gate in Figure 15 reverses the state of $|aux2_0\rangle$; this is followed by a complete repetition of the remaining of the circuit, in order to reverse $|aux2_1\rangle$ and $|aux2_2\rangle$ – these are not depicted here due to space constraints.

This oracle is valid for the single solution and multiple solution occlusion problems, but in the case of the multiple solution visibility problem the z coordinate enters the oracle's calculations for the intersection because, additionally to the conditions already mentioned, the depth of the ray must be lower than a current depth. This adds more blocks (similar to the ones already shown but regarding the z coordinate) to the oracle circuit in Figure 15.

4.1.4 Inversion about the average

After the solution has been marked by the oracle, the inversion about the average is performed, using Grover's diffusion operator, to increase the amplitude of the marked state, thus increasing its probability of being measured.

Let's take a closer look to the Grover's diffusion operator $\hat{D}$. As we've seen before, this operator is defined as $\hat{D} = 2|s\rangle\langle s| - \hat{I}$ but this doesn't really give us any clues about how to implement $\hat{D}$ using common quantum gates. It turns out to be much simpler to figure this out if we slightly rewrite $\hat{D}$. Instead of writing $\hat{D}$ as a reflection about $|s\rangle$, we can write it as reflection about $|0\rangle$ but preceded and followed by a Hadamard gate applied to each of the qubits.

This equivalence is not immediately obvious, so we can start by dividing this operator into 2 simpler operators as shown in the next equation:

$$\hat{D} = 2|s\rangle\langle s| - \hat{I} = \hat{H}(2|0\rangle\langle 0| - \hat{I})\hat{H} = 2\underbrace{\hat{H}|0\rangle\langle 0|\hat{H}}_{=|s\rangle\langle s|} - \underbrace{\hat{H}\hat{I}\hat{H}}_{\hat{I}}$$

From the equation above, we clearly see that $\hat{H}\hat{I}\hat{H} = \hat{I}$ because $\hat{H}$ is its own inverse. This is looking closer to the original expression for $\hat{D}$ but we still need to prove the equivalence of the other part of the operator. The first piece, $\hat{H}|0\rangle$, is clearly equal to the vector $|s\rangle$, so what we really need to show is $\langle 0|\hat{H} = \langle s|$.

To prove this equivalence, we'll take an arbitrary vector $|v\rangle$. Applying $\hat{H}$ followed by taking the inner product with $\langle 0|$ to the vector $|v\rangle$ yields a scalar, $\langle 0|\hat{H}|v\rangle = \langle 0|Hv\rangle$, and we want to show that this scalar is equivalent to the inner product of the uniform superposition $|s\rangle$ with $|v\rangle$, $\langle 0|Hv\rangle = \langle s|v\rangle$.

We'll show this by using a property of $\hat{H}$. $\hat{H}$ is a unitary operation, thus its conjugate transpose is itself and from the properties of the conjugate transpose we have that, for a matrix $\hat{A}$ and vectors $|x\rangle$ and $|y\rangle$, $\langle x|\hat{A}y\rangle = \langle \hat{A}^\dagger x|y\rangle$.

Putting all this together we get:

$$\langle 0|Hv\rangle = \langle H^\dagger 0|v\rangle = \langle H0|v\rangle = \langle s|v\rangle$$

proving that the Diffusion operator is indeed a reflection about $|0\rangle$ preceded and followed by a Hadamard gate: $\hat{D} = 2|s\rangle\langle s| - \hat{I} = \hat{H}(2|0\rangle\langle 0| - \hat{I})\hat{H}$.

This is useful because it's much more obvious how to implement this reflection about $|0\rangle$ than it is to implement the reflection about $|s\rangle$. Let's consider what the reflection about $|0\rangle$ does to different inputs:

$$\begin{aligned}
2|0\rangle\langle 0|0\rangle - \hat{I}|0\rangle &= 2|0\rangle 1 - |0\rangle = |0\rangle \\
2|0\rangle\langle 0|x\rangle - \hat{I}|x\rangle &= 2|0\rangle 0 - |x\rangle = -|x\rangle
\end{aligned}$$

We see from the above equation that if we give the input $|0\rangle$, the reflection about $|0\rangle$ is just going to give $|0\rangle$ and if we give some input $|x\rangle$ different than $|0\rangle$ we put a negative sign in front of that vector. Given this information, we need to negate the amplitude of every state except the zero state, or the negative, which is to negate the amplitude of the zero state. The quantum circuit for the Grover's diffusion operator is presented in Figure 16 [18].

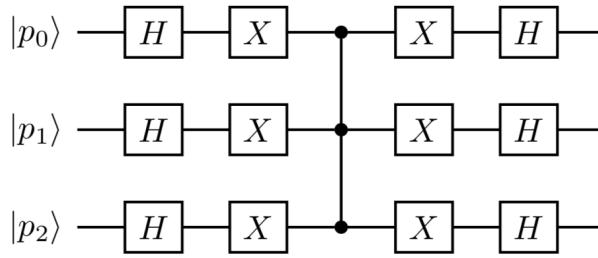


Figure 16: Quantum circuit for the Grover's diffusion operator for the case of 3-qubit primitive indices.

4.1.5 Overall circuit

The overall and most simplified circuit of this quantum ray tracer is represented in Figure 17. Here we see the steps of the algorithm clearly: (1) application of Hadamard gates; (2) application of oracle; (3) application of Grover's diffusion operator (Gdo in the figure) (last 2 steps are applied r times); (4) measurement.

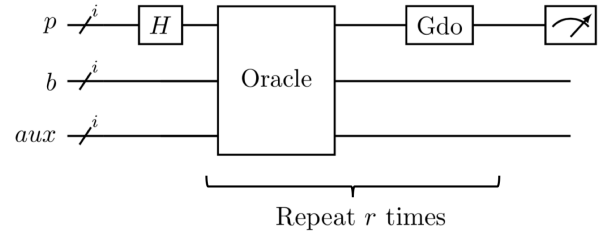


Figure 17: Quantum circuit of the algorithm. (i is not necessarily the same for p , b and aux , it just represents that the quantum registers have multiple qubits)

4.2 SINGLE SOLUTION

This section addresses the case where no more than one primitive projects to each pixel, therefore, for each primary ray, and thus for each quantum query, there are zero solutions or only one. The important points to consider are:

- (i) the number of solutions is known ($t = 1$), which is fundamental for calculating the number of Grover's iterations ($t = 0$ will be handled as a failure to find a solution);
- (ii) the occlusion ("is the ray intersected?") and visibility ("which primitive does the ray intersect?") problems are the same, since the ray intersects only one primitive.

4.2.1 Algorithm

Grover's algorithm is used to identify intersections between a given ray (x, y) and the geometric primitives. These algorithms assume there are $N = 2^n$ primitives indexed from $0, \dots, N - 1$ using an n -qubit quantum register labelled $|p\rangle$; the quantum register $|b\rangle$ is prepared with the coordinates of the primitives, thus there are $n + b$ qubits plus auxiliary qubits, with $b = \lceil \log_2 (\max(\max_x, \max_y)) \rceil$ (the closest upper integer of the $\log_2$ of the maximum between $\max_x$ and $\max_y$).

The first step of the algorithm is to do a superposition over the qubits representing the primitives $|p\rangle$. The next step is to apply the oracle and the diffusion operator r times. Lastly, a measurement of the quantum register representing the primitives is done. These steps are described in Algorithm 1.

Algorithm 1 Q-RayCast: quantum algorithm for ray (x, y) , N primitives and r iterations

```

Superposition over the primitives IDs:  $|p\rangle = H^{\otimes n} |0\rangle^{\otimes n}$ 
 $r = \lfloor \frac{\pi}{4} \sqrt{N} \rfloor$ 
for  $r$  iterations do
  Grover's oracle {Algorithm 2}
  Grover's diffusion operator
end for
ID  $\leftarrow$  Measure  $|p\rangle$ 
return ID

```

This algorithm will run for each ray and, in each execution, new quantum and classical registers are created alongside a new quantum circuit.

As for the oracle, the algorithm is presented below, regarding the diffusion operator, its algorithm is simply to add Hadamard, CX and CZ gates as discussed in Section 4.1.4 and shown in Figure 16.

Algorithm 2 Oracle for ray (x, y) intersection with N primitives

```

 $|H_{int}\rangle \leftarrow |1\rangle; |V_{int}\rangle \leftarrow |1\rangle; |intersect\rangle \leftarrow |0\rangle$ 
 $min_x: |p\rangle |b\rangle = mX |p\rangle |0\rangle^{\otimes nb}$ 
Negate  $H_{int}$  if  $x \geq b$ ; revert  $|b\rangle$ 
 $max_x: |p\rangle |b\rangle = MX |p\rangle |0\rangle^{\otimes nb}$ 
Negate  $H_{int}$  if  $x \leq b$ ; revert  $|b\rangle$ 
 $min_y: |p\rangle |b\rangle = mY |p\rangle |0\rangle^{\otimes nb}$ 
Negate  $V_{int}$  if  $y \geq b$ ; revert  $|b\rangle$ 
 $max_y: |p\rangle |b\rangle = MY |p\rangle |0\rangle^{\otimes nb}$ 
Negate  $V_{int}$  if  $y \leq b$ ; revert  $|b\rangle$ 
 $|intersect\rangle = |H_{int}\rangle \wedge |V_{int}\rangle$ 
Flip the primitive's coefficient sign if  $|intersect\rangle = |1\rangle: |p\rangle = (-1)^{|intersect\rangle} |p\rangle$ 
Revert  $|H_{int}\rangle, |V_{int}\rangle$  and  $|intersect\rangle$ 

```

Algorithm 2 presents the oracle. We saw its circuit representation in Figure 15. We see we start by initializing the $|H_{int}\rangle$ and $|V_{int}\rangle$ qubits with the value $|1\rangle$ (these qubits are the $aux2_2$ and $aux2_1$ qubits respectively, presented in Figure 15). The $|intersect\rangle$ qubit is initialize with $|0\rangle$ and corresponds to the $aux2_0$ qubit in Figure 15. The negation of $|H_{int}\rangle$ and $|V_{int}\rangle$ is done because, if a ray intersects the primitive, it has to check 2 conditions whose result is stored in the $|H_{int}\rangle$ and $|V_{int}\rangle$ qubits. If it checks both, it will result in negating the qubits twice, setting them back to their original value $|1\rangle$. If a primitive doesn't check the vertical or horizontal condition (or both), meaning $|H_{int}\rangle$ or $|V_{int}\rangle$ is set to $|0\rangle$ (or both), the $|intersect\rangle$ qubit will remain $|0\rangle$. In the case where the number of primitives is not a power of 2 and the geometry has to be filled with "false" primitives, meaning, primitives with numbers that make an intersection impossible to occur, the first number is bigger than the second, both $|H_{int}\rangle$ and $|V_{int}\rangle$ qubits will have the value $|0\rangle$ and the $|intersect\rangle$ qubit will remain $|0\rangle$ as expected. The qubits never stay unchanged from $|1\rangle$ because they always check one condition of the vertical/horizontal dimension (min or max). The last line of the algorithm reverts these qubits to their original state so they can be used again.

After the execution of Algorithm 1, the probability of reading the ID of the primitive that intersects the ray, the success probability (p_s), is $\sin^2((1 + 2r)\theta)$ with $\theta = \sin^{-1} \frac{1}{\sqrt{N}}$ (Page 15). For $N > 4$ this probability is close to, but less than, 1. Upon measurement three different cases can occur:

1. the intersecting primitive is measured;
2. a non-intersecting primitive is measured, but in fact the ray intersects another primitive;
3. a non-intersecting primitive is measured and in fact there is no intersection (all primitives have a uniform probability of $\frac{1}{N}$ of being measured).

To solve this issue we have Algorithm 3. Algorithm 3 calls quantum Algorithm 1 and then verifies whether the ray intersects the measured primitive using the function *verify_intersection* which compares, in a classical and very straightforward way, the ray's coordinates with the coordinates of the primitive obtained as the result. This is a hybrid algorithm, since the classical code calls a quantum program. If the measured primitive is not intersected, the quantum algorithm is executed again. If after c such iterations no measured primitive intersects the ray, then that pixel is treated as not occluded.

Algorithm 3 Hybrid algorithm for the non overlapping case (single solution)

```

for all pixels  $(x, y)$  on the image plane do
  intersected = false
  iteration = 0
   $r = \lfloor \frac{\pi}{4} \sqrt{N} \rfloor$ 
  while not intersected and (iteration < c) do
    ID = Q_RayCast( $x, y$ , primitives,  $r$ ) {Algorithm 1}
    intersected = verify_intersection( $x, y$ , ID)
    iteration ++
  end while
end for

```

To discover the probability, or certainty, of the pixel not being occluded, let's start by defining the probability of a measurement being a success or a failure. A success is when the result yields an intersection and a failure is when the results yield no intersection (we can see a success as a 1 and a failure as a 0). We have already defined that the probability of a successful measurement is p_s , thus the probability of a failed measurement is $1 - p_s$. With an accurate number of Grover's iterations, the probability of success is really high, making the probability of a failure very low. The probability of obtaining a successful measurement after a failed one is $(1 - p_s)p_s$ and after 2 failed ones is $(1 - p_s)(1 - p_s)p_s$, thus the probability of obtaining a successful result after c failed measurements is $(1 - p_s)^c p_s$. This probability is really low and decreases with each measurement because if there hasn't been a success, it's really unlikely that there will be one. We can now conclude that, after

$c - 1$ measurements returning non-occlusion, the probability of measuring an occlusion at the c -th measurement is $p_{s,c} = 1 - (1 - p_s)^{c-1}p_s$ according to the geometric distribution and this probability is really high, because with each measurement, if there hasn't been a success, it's more and more likely to not have one after each measurement. With this, we can define the probability, after c failed measurements, of the ray not being occluded. This will be equal to the sum of the probability of the pixel not being occluded in each measurement:

$$p_{s,c} = p_s + \sum_{j=1}^{c-1} p_s (1 - p_s)^j$$

This probability increases with the number of measurements; the more measurements we do, the more certain we are that there isn't indeed an intersection. Because this is a probabilistic algorithm, we are never 100% certain of the result, but with a good number of Grover's algorithm iterations we can be close to that value.

Note that within this work's context, verifying whether a given ray intersects a specific primitive is assumed to be $\mathcal{O}(1)$: the goal of the quantum approach is to reduce the number of evaluations of the *verify_intersection* function, not the cost of each such evaluation. Repeating the algorithm a few times, does not have any material impact on quadratic speedup for large N .

4.2.2 Results

Each scene has a given geometry. This geometry is defined as a list in which each element represents a primitive. As mentioned in page 26, if the number of primitives is not a power of 2, the list is filled with "false" primitives until the number of primitives is a power of 2. The length of this list is the number of primitives in the scene. A primitive is also represented as a list. As mentioned in Section 4.1.1, this list has 5 elements: $(min_x, max_x, min_y, max_y, z)$, representing the limits of the primitive, as well as its depth (in this case that's not relevant).

Figure 18 presents the images obtained with the Qiskit simulator for 2 geometric configurations (scenes) with 4 and 8 primitives, respectively. The numbers identify the primitives, whereas the dots depicted in each pixel represent the number of iterations required to evaluate the pixel, c , due to evaluating non intersecting primitives. When there is no intersection, the algorithm will run the maximum allowed number of iterations. From Figure 18 we see that the intersecting primitive was always measured on the first iteration of Algorithm 3,

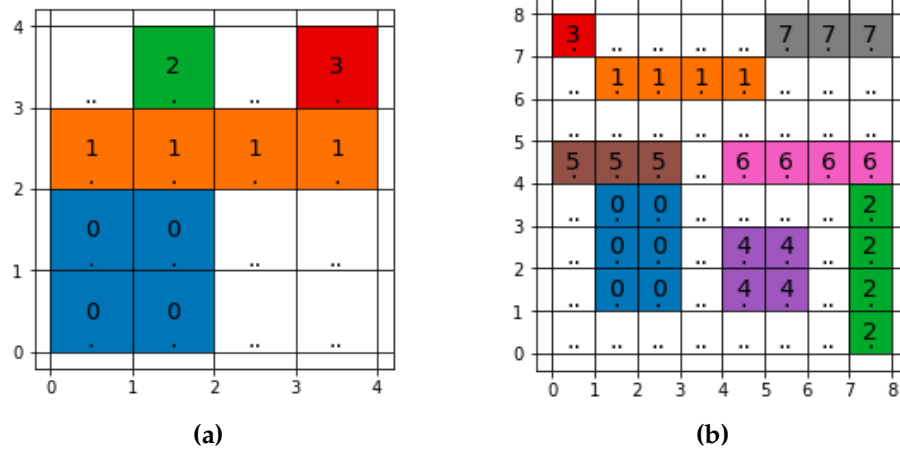


Figure 18: Reference images obtained with the Qiskit simulator for the single solution case. The geometry in **(a)** is $[(0,1,0,1), (2,2,0,3), (3,3,1,1), (3,3,3,3)]$ and in **(b)** is $[(1,3,1,2), (6,6,1,4), (0,3,7,7), (7,7,0,0), (1,2,4,5), (4,4,0,2), (4,4,4,7), (7,7,5,7)]$

although in a second run with geometry **(b)**, 2 measurements were needed to find some primitives. For the geometry in **(a)**, since $N = 4$ and $t = 1$ (number of solutions), the probability of success (on a non noisy environment such as the simulator) is 1, therefore having $r = 1$ would be enough as is shown below.

Figure 19 presents the results with 2048 shots in the simulator for each pixel with the 4×4 geometry of Figure 18. We see that, for this case, where $t = \frac{N}{4}$, we have $\sin^2 \theta = \frac{t}{N} \Leftrightarrow \sin^2 \theta = \frac{1}{4} \Leftrightarrow \theta = \sin^{-1} \left(\sqrt{\frac{1}{4}} \right) \Leftrightarrow \theta = \frac{\pi}{6}$. With Grover's iterations $r = 1$, the exact probability of measuring the correct answer is $\sin^2 \left(\left(r + \frac{1}{2} \right) \theta \right) = \sin^2 \left(\left(1 + \frac{1}{2} \right) \frac{\pi}{6} \right) = \sin^2 \frac{\pi}{2} = 1$ (Page 26), meaning a solution is found with certainty after a single iteration.

In Figure 20, we have the histogram for pixel (4,1) in the 8×8 geometry of Figure 18 for 2048 shots. Because here we have $t = \frac{N}{8}$, the probability of measuring the right primitive is not 100%, although it is really close. In this case r was set to 2, and, following the same calculations done for the previous case, the probability of measuring the correct answer is 0.945, which is extremely close to the probability obtained.

The tables below present some characteristics of the circuits, for different geometries, that are important to understand the viability of the execution of this algorithm on a real quantum machine with today's technology, such as, the total number of qubits, the depth of the circuit and the total number of gates. These statistics were found using qiskit's built-in functions when the circuits were run in the simulator.

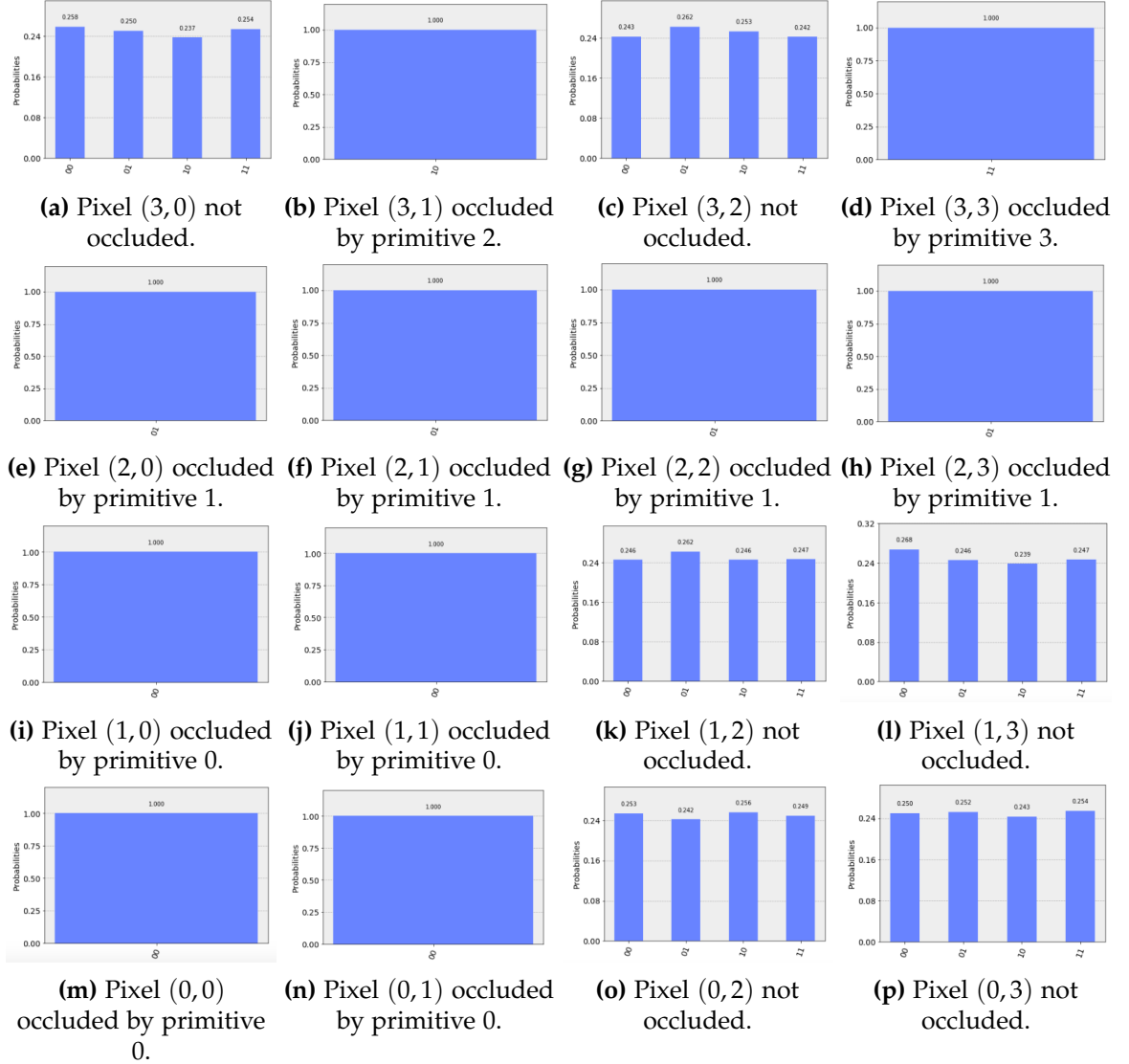


Figure 19: Results retrieved with the Qiskit simulator for each pixel in the geometry $[(0,1,0,1), (2,2,0,3), (3,3,1,1), (3,3,3,3)]$ with 2048 shots.

During the acquirement of the results, we noted some differences between full and not full geometries. A full geometry happens when the numbers of primitives is a power of 2; otherwise, the remaining primitives have to be generated creating "false" primitives. This first table presents the results for full geometries. In some cases, having less primitives but using the same amount of qubits to represent them, results in a lower number of total qubits used; for example, when having 3 qubits for the primitives and for the coordinates, with only 6 "real" primitives, 13 qubits are used instead of 15.

A note for the case with 3 qubits for the primitives and 2 for the coordinates. A geometry with 8 "real" primitives in this case implies primitives that only occupy 1 square and, in

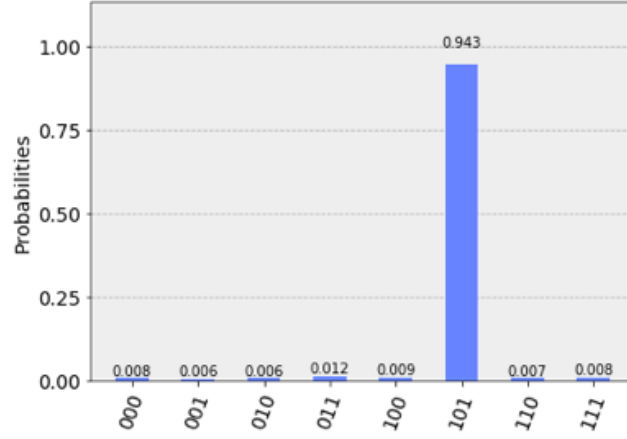


Figure 20: Results obtained with the Qiskit simulator for pixel (1,4) in geometry $[(1,3,1,2), (6,6,1,4), (0,3,7,7), (7,7,0,0), (1,2,4,5), (4,4,0,2), (4,4,4,7), (7,7,5,7)]$ with 2048 shots.

that case, the number of qubits needed is lower. The results showed for that case, present the worst case for that specific geometry, which is a case with less than 8 “real” primitives but where not all primitives occupy only 1 square. The number of qubits varies according to the shape and size of the primitives. In this geometry, with the 8 1-squared primitives, only 10 qubits were necessary.

#qubits for the primitives	#qubits for the coordinates	#auxiliary qubits	Total #qubits
2	2	5	9
2	3	6	11
2	4	7	13
3	2	7	12
3	3	9	15
3	4	7	14

Table 2: Growth of number of qubits of the circuits for different types of geometries.

From Table 2 we see that, with 2 qubits for the primitives, every time we increase the number of qubits for the coordinates by one, one more auxiliary qubit is needed. With 3 qubits for the primitives, the number of auxiliary qubits increased by 2.

The next tables present the variation of the depth and number of gates of the circuits. We have divided these results into full and not full geometries because, in some instances, only one of the cases isn’t enough to demonstrate the variation of the numbers accurately and

by doing this we get a wider perspective of the results.

Regarding the depth of the circuit, when going from 2 to 3 qubits for the coordinates with 2 qubits for the primitives, the depth increases approximately 3 times, and 1 additional time when adding a 4th qubit (Table 4). With the same happening for the number of gates of the circuit seen in Table 6. This happens for full geometries; in the other case, going from 3 to 4 qubits for the coordinates doesn't increase the number of gates or depth by a lot.

As for passing from 2 to 3 qubits in the coordinates having 3 qubits for the primitives, the depth and number of gates increases approximately 4 times. This is obtained with the table for geometries that are not a power of 2 because, in the other case, the growth is really high due to the shape of the primitives (only 1 square). It was also observed that, when going to 4 qubits, in full geometries, the depth and number of gates increases but not by much, and in not full geometries, those values decreased.

#qubits for the primitives	#qubits for the coordinates	Minimum depth	Maximum depth	Average depth
2	2	416	594	496
2	3	1234	1760	1463
2	4	1602	2482	1972
3	2	536	874	685
3	3	6776	7832	7234

Table 3: Growth of the depth of the circuits for different types of FULL geometries.

#qubits for the primitives	#qubits for the coordinates	Minimum depth	Maximum depth	Average depth
2	2	749	925	830
2	3	1420	1948	1652
2	4	1573	2285	1881
3	2	2348	2820	2572
3	3	8888	10036	9402

Table 4: Growth of the depth of the circuits for different types of NOT FULL geometries.

It's good to remind that the results vary, in each case (same number of qubits for primitives and coordinates), due to the size and shape of the primitives, but the cases presented intend to give a generalized view of the statistics of the circuits used. The depth and number of gates appear with a minimum, maximum and average value because these parameters vary for each pixel.

#qubits for the primitives	#qubits for the coordinates	Minimum #gates	Maximum #gates	Average #gates
2	2	701	981	827
2	3	2029	2869	2427
2	4	2645	4045	3324
3	2	788	1348	1040
3	3	13204	14884	14000

Table 5: Growth of number of gates of the circuits for different types of FULL geometries.

#qubits for the primitives	#qubits for the coordinates	Minimum #gates	Maximum #gates	Average #gates
2	2	1257	1537	1383
2	3	2341	3181	2739
2	4	2322	3405	2852
3	2	4084	4644	4336
3	3	16596	18996	17408

Table 6: Growth of number of gates of the circuits for different types of NOT FULL geometries.

4.3 MULTIPLE SOLUTIONS

Moving to the case where primitives can be in different depths is the same as saying we are now in a 3D scenario. By adding a third dimension to the primitives (z is now relevant and defines the depth of the primitives), it becomes possible that a ray intersects more than one primitive but at different depths. In this setup we can formulate two distinct questions: "is the ray occluded by any primitive?" – occlusion and "what's the closest primitive the ray intersects?" – visibility. This brings an issue that we didn't have in the previous case. The number of solutions to Grover's algorithm, t , can now range from 0 (no occlusion) to the total number of primitives, N . As long as t is known and less than $N/2$, the problem can still be solved using the algorithm proposed in the previous section by setting the number of Grover's iterations $r \leftarrow \lfloor \frac{\pi}{4} \sqrt{\frac{N}{t}} \rfloor$. But, in most real cases, the number of primitives projecting onto a given pixel is unknown, therefore r cannot be set *a priori* to the ideal number of iterations, meaning, we don't know how many times we should apply the Grover's algorithm. However, there's an algorithm mentioned in the next section, called Q-Search, that will help us calculate the number of Grover iterations.

4.3.1 Occlusion

To solve the problem of occlusion with multiple solutions, we use an algorithm called Q_Search, proposed by Boyer et al. [12], to address the issue of not knowing how many times to apply Grover's. The algorithm is based on exponential search and demonstrated to be expected to converge in a total of $\mathcal{O}\left(\sqrt{\frac{N}{t}}\right)$ Grover iterations (not to be confused with the number of the *while* loop iterations); the case $t = 0$ (no solution) is handled by terminating after a constant number of the *while* loop iterations, c .

Algorithm 4 Q_Search: Quantum exponential search algorithm for the overlapping case - pixel (x, y)

```

intersected = false
iteration = 0
l = 0
d = 1.3
while not intersected and (iteration < c) do
  ID = Q_SampleUniformDistribution(primitives)
  intersected = verify_intersection(x, y, ID)
  if not intersected then
    l = l + 1
    S = min( $d^l$ ,  $\sqrt{N}$ )
    r = rand(1, ..., S)
    ID = Q_RayCast(x, y, primitives, r) {Algorithm 1}
    intersected = verify_intersection(x, y, ID)
    iteration++
  end if
end while

```

At each iteration, Algorithm 4 exponentially increases the maximum number of possible Grover iterations S and then randomly selects r from $1, \dots, S$; S is equal to d^l , where d is a number between 0 and 1 and l starts by being 0 and is incremented by 1 in each iteration (S can't be bigger than $\sqrt{N}$). At the beginning of each iteration the primitives are also randomly sampled (*Q_SampleUniformDistribution*), which will succeed with probability $\frac{t}{N}$. The algorithm runs until an intersection is found or, if there isn't one found, it runs a maximum of c times. The algorithm runs for each pixel.

To demonstrate this algorithm, we selected a scene with 8 primitives at 3 different depths as shown in Figure 21 with following results obtained with the Qiskit simulator.

Figure 22 illustrates 6 independent executions of the proposed algorithm with the geometry presented in Figure 21 and with $c = 2$, i.e., a maximum of 2 iterations until the pixel

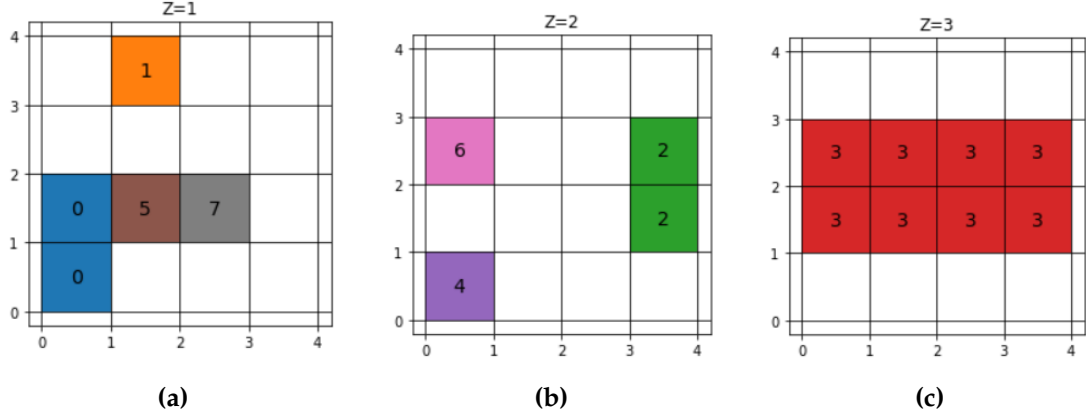


Figure 21: Scene for multiple solution occlusion case.
(a) Depth $z = 1$, **(b)** depth $z = 2$ and **(c)** depth $z = 3$.

is classified as non occluded. We can see that only in 2 out of 6 experiments is occlusion correctly evaluated (figures (c) and (d)); in 1 case (figure (a)) 2 out of 16 pixels are wrong, whereas on the remaining 3, one pixel is wrong.

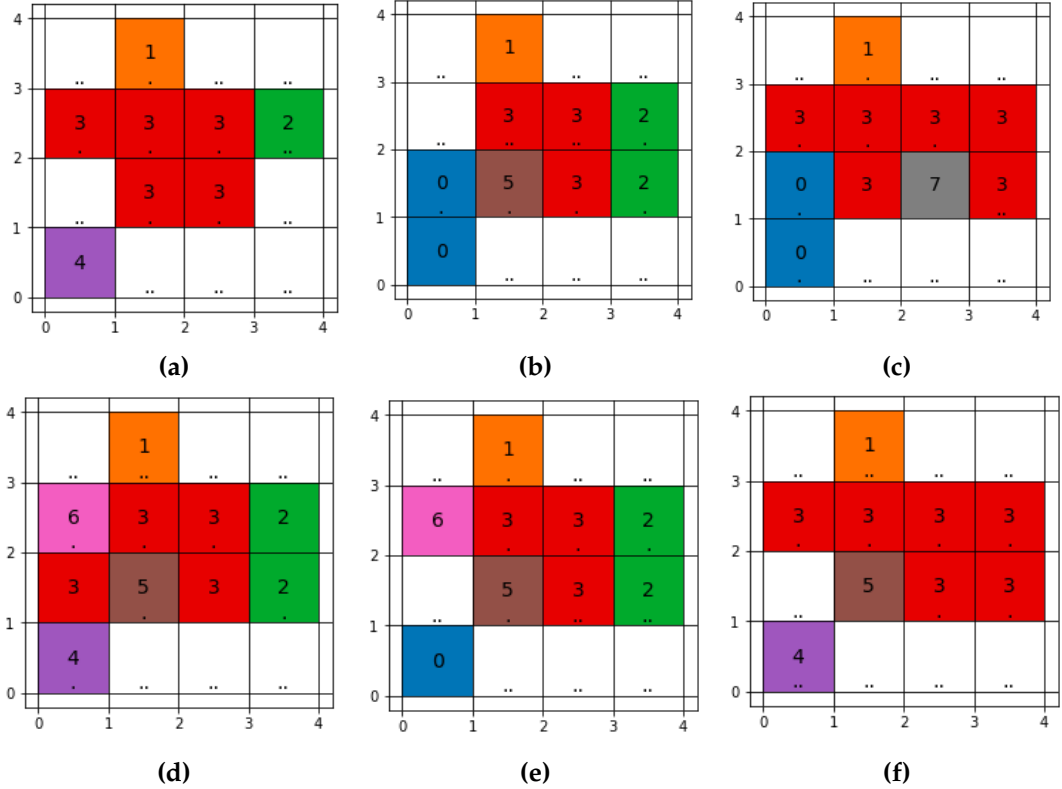


Figure 22: Results for the multiple solution occlusion case with $c = 2$.

The probabilistic nature of the algorithm and the importance of the random sampling step are clearly demonstrated by the number of iterations required to find an occluding primitive. Some pixel's results were found with 0 iterations by random sampling and other pixels require arbitrarily 1 or 2 iterations (with an unreported number of Grover iterations).

In the figure above we can also see that the pixels that didn't show an occlusion when they should have, are pixels with an overlap of primitives thus the number of Grover's iterations may be less accurate.

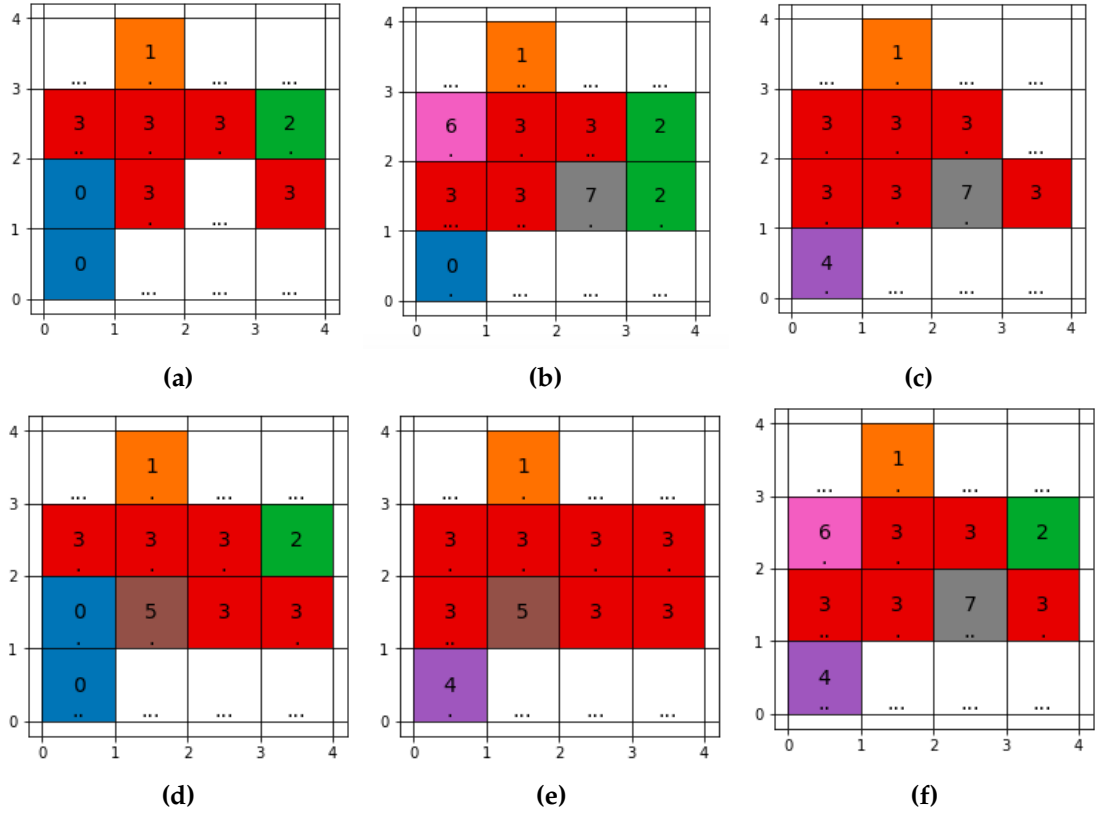


Figure 23: Results for the multiple solution occlusion case with $c = 3$.

Afterwards, c was increased to 3 and the results are shown in Figure 23. We can see that the results have improved and all the primitives occluded are shown almost every time, only 2 out of 6 runs have an error on a single pixel. In the cases a primitive is not measured and there is an intersection, there is an overlap, as in the previous case ($c = 2$).

We continued to increase c and with 4 iterations the results were 100% accurate and no primitive needed more than 3 repetitions to be found as an occluding primitive, as seen in Figure 24.

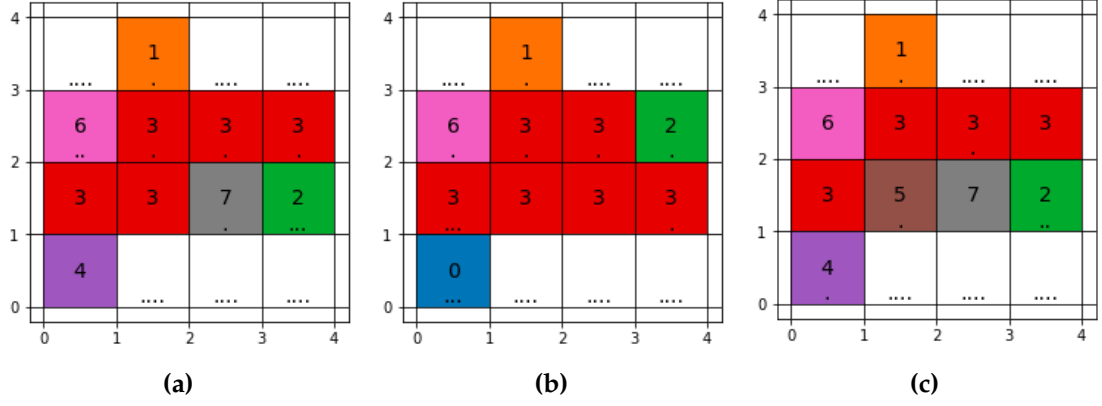


Figure 24: Results for the multiple solution occlusion case with $c = 4$.

We verify that the results weren't 100% accurate for a lower value of c . The fact that we're dealing with small numbers ($N = 8$) hinders Grover's algorithm accuracy because the calculated number of iterations r is meant for very large numbers. This contributes to those failed results in some attempts.

In Table 7 is presented the growth of the depth of the circuit and the number of gates according to the increase of the number of repetitions of Grover's algorithm, r , that this algorithm calculates in each iteration. This geometry used 11 qubits.

	Avg depth	Avg #gates
$r = 1$	2408	4210
$r = 2$	4806	8405
$r = 3$	7146	12475

Table 7: Growth of depth and number of gates depending on number of iterations of Grover's algorithm r in a 4x4 scene.

As we can observe, every time r is increased, the depth and number of gates increases in the same amount as expected, because with each increasing r we are repeating almost the whole circuit and those 2 quantities will increase in the same amount.

We also ran the algorithm for a scene with coordinates from 0 to 7. Figure 25 shows the results obtained and Table 8 shows the depth and number of gates of the circuits used to obtain the results. This geometry used 15 qubits.

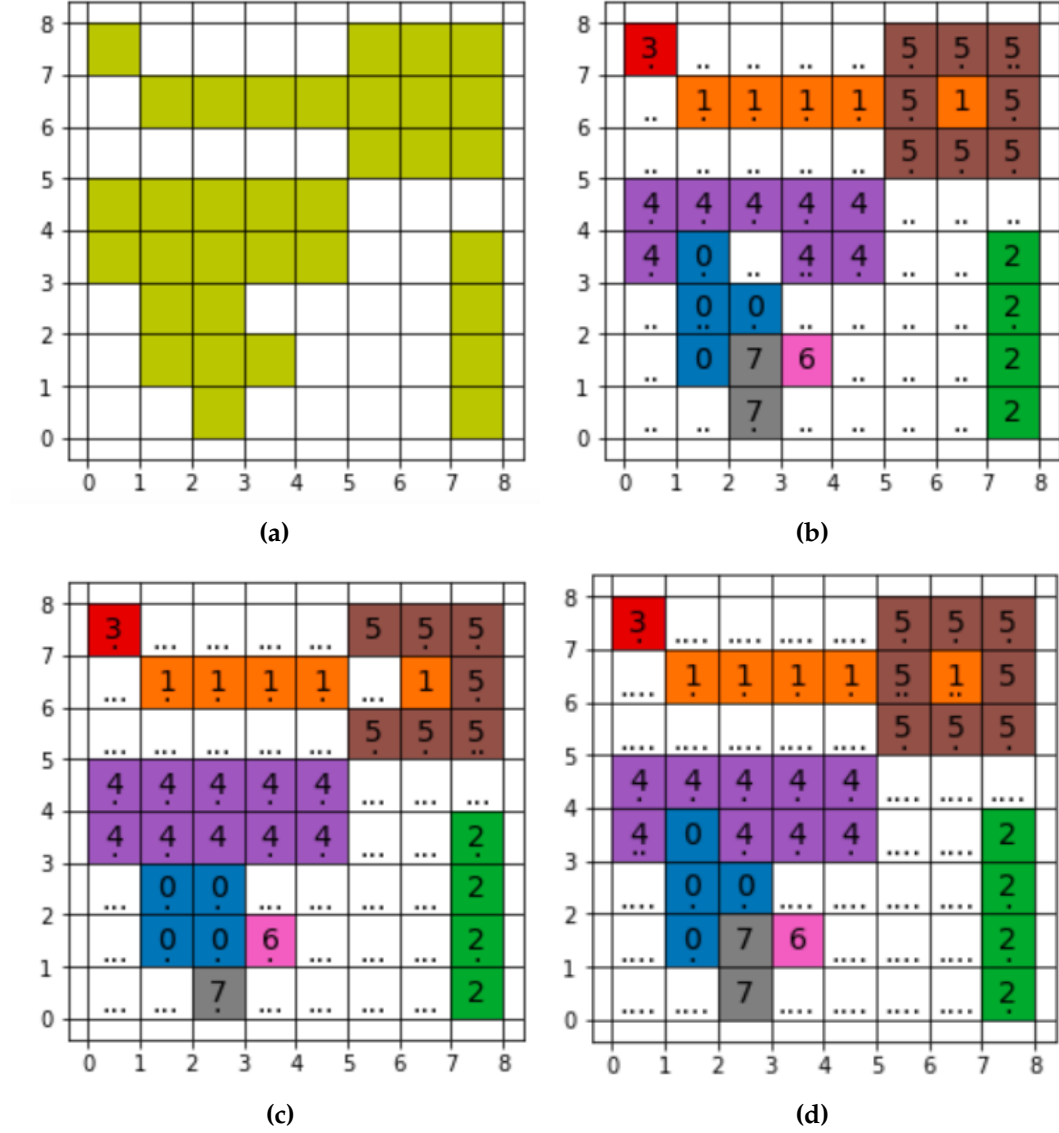


Figure 25: Results for the multiple solution occlusion case, with an 8x8 scene.

From Figure 25, we see that we only obtained 100% accurate results with $c = 4$. In the other 2 cases, $c = 2$ and $c = 3$, there was one pixel with an incorrect result. Both incorrect results occurred in a pixel with more than one intersection.

Regarding Table 8, as in the previous case, every time r increases, the depth and number of gates increase in the same amount.

In Table 9 is presented the comparison between the 4x4 and 8x8 geometries, in regards to the depth and number of gates of the circuits. There's an increase of approximately 1,7 times of the depth and 1,8 times of the number of gates from one case to the other.

	Avg depth	Avg #gates
$r = 1$	4006	7646
$r = 2$	8024	15325
$r = 3$	12060	23047

Table 8: Growth of depth and number of gates depending on number of iterations of Grover's algorithm r in a 8x8 scene.

	$r = 1$		$r = 2$		$r = 3$	
	Avg depth	Avg #gates	Avg depth	Avg #gates	Avg depth	Avg #gates
4x4	2408	4210	4806	8405	7146	12475
8x8	4006	7646	8024	15325	12060	23047

Table 9: Comparison between the 4x4 and 8x8 scenes.

To illustrate the importance of getting the number of Grover iterations right with the Q-Search algorithm, we've created a geometry, shown in Figure 26, where we have pixels with 1, 2 and 3 different primitives intersecting with the ray.

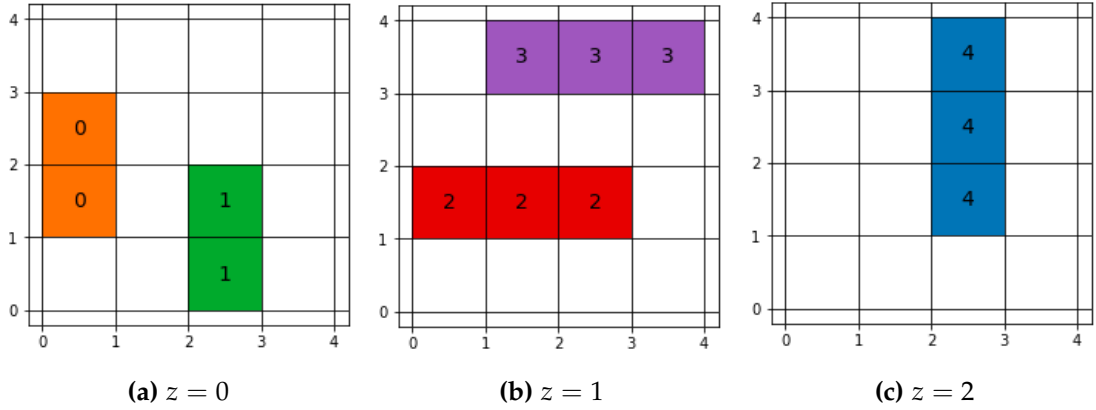


Figure 26: Scene with geometry $[(1, 2, 0, 0, 0), (0, 1, 2, 2, 0), (1, 1, 0, 3, 1), (3, 3, 1, 3, 1), (1, 3, 2, 2, 2)]$.

In this first case (Figure 27), pixel $(2, 0)$, there is only 1 solution, primitive 0. We observe that with 1 iteration of Grover's algorithm, $r = 1$, we can obtain the correct answer, but if we do a second run of the algorithm we obtain the same correct result with a higher probability.

In Figure 28, below, we have the case of a pixel having 2 intersecting primitives. Here we have the case where $t = \frac{N}{4}$ ($t = 2$ and $N = 8$) and we can see the probabilities are totally distributed by the 2 intersecting primitives. However, if we do 2 iterations of Grover's, we end up with a uniform distribution and can't get any good results.

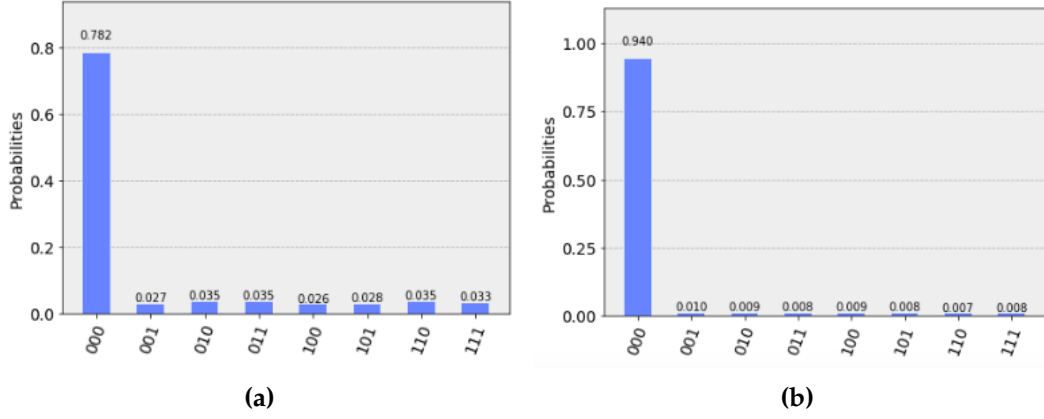


Figure 27: Results for pixel (2,0) with $r = 1$ and $r = 2$.

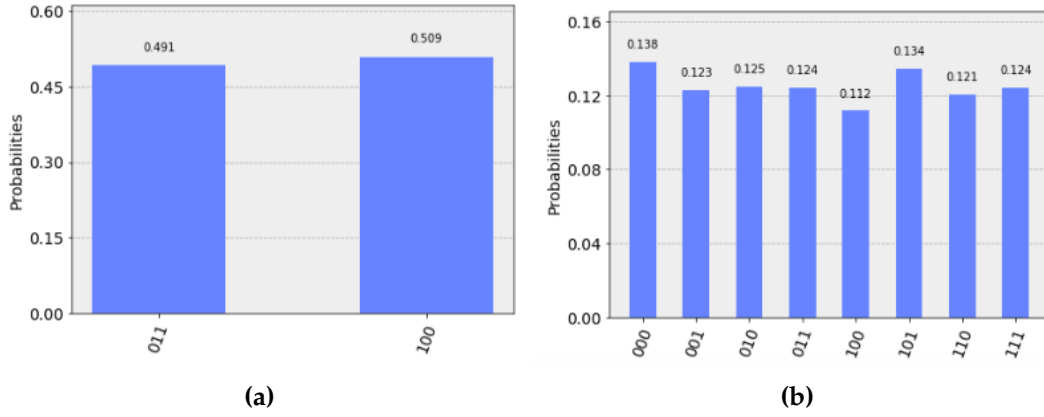


Figure 28: Results for pixel (3,2) with $r = 1$ and $r = 2$.

Finally, we present in Figure 29 the case with 3 intersecting primitives (pixel (2,0)). With $r = 1$, the 3 primitives show a greater probability than all the others, yet, with $r = 2$ the probabilities are all flipped and the primitives we wanted to find become the ones with the lowest probabilities.

With these 3 figures we are able to have a better understanding of how the correct or incorrect number of Grover's iterations influences the results obtained. Calculating and using the right amount of Grover's iterations is fundamental to gather correct results.

4.3.2 Visibility

Contrary to the occlusion problem, where we just wanted to know if a ray was occluded, in this case, we want to know, among all primitives intersected by the ray, which one is closer to its origin. In order to compute which primitive, if any, is visible along each pixel, the minimum depth (z coordinate) has to be evaluated. This means we'll need to update

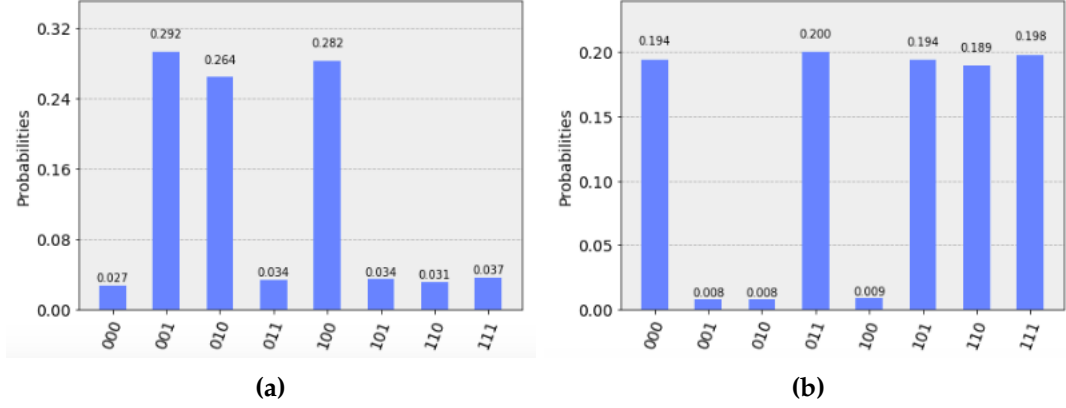


Figure 29: Results for pixel (1,2) with $r = 1$ and $r = 2$.

our oracle algorithm, given in Algorithm 2, with depth data to include this new evaluation. The new oracle is presented in Algorithm 5.

Algorithm 5 Oracle for ray (x, y) intersection with N primitives and depth $< \min Z$

$|H_{int}\rangle \leftarrow |1\rangle; |V_{int}\rangle \leftarrow |1\rangle; |D_{int}\rangle \leftarrow |0\rangle; |intersect\rangle \leftarrow |0\rangle$
 $\min_x: |p\rangle |b\rangle = mX |p\rangle |0\rangle^{\otimes nb}$
 Negate H_{int} if $x \geq b$; revert $|b\rangle$
 $\max_x: |p\rangle |b\rangle = MX |p\rangle |0\rangle^{\otimes nb}$
 Negate H_{int} if $x \leq b$; revert $|b\rangle$
 $\min_y: |p\rangle |b\rangle = mY |p\rangle |0\rangle^{\otimes nb}$
 Negate V_{int} if $y \geq b$; revert $|b\rangle$
 $\max_y: |p\rangle |b\rangle = MY |p\rangle |0\rangle^{\otimes nb}$
 Negate V_{int} if $y \leq b$; revert $|b\rangle$
 $z: |p\rangle |b\rangle = Z |p\rangle |0\rangle^{\otimes nb}$
 Negate D_{int} if $z \leq b$; revert $|b\rangle$
 $|intersect\rangle = |H_{int}\rangle \wedge |V_{int}\rangle \wedge |D_{int}\rangle$
 Flip the primitive's coefficient sign if $|intersect\rangle = |1\rangle: |p\rangle = (-1)^{|intersect\rangle} |p\rangle$
 Revert $|H_{int}\rangle, |V_{int}\rangle, |D_{int}\rangle$ and $|intersect\rangle$

Besides generating the primitives' bounds and comparing them with the ray's coordinates, now the primitives' depths are also generated and compared with the current minimum depth; the oracle will only flip a primitive's coefficient sign if the 5 conditions (x and y minimum and maximum values and depth - z - value) are satisfied. This extension to the oracle's algorithm translates in the addition of one more *aux2* qubit in the circuit of Figure 15 and one more set of boxes for the le_z and Z operations.

We have a function that verifies if a primitive has really been intersected but now we need to test if that primitives' depth is lower than the current minimum depth. For that, we have Algorithm 6. If the current primitive has been intersected, the algorithm will save that result if it is the first obtained intersected primitive, or, if other primitives have been intersected, it will save the result if its depth is lower than the current minimum.

Algorithm 6 Min: Minimum depth algorithm

```

intersected = verify_intersection ( $x, y$ , ID, geom)
if intersected then
  if (depth[ID] <  $pZ$  and there_was_one_intersection = 1) or there_was_one_intersection
    = 0 then
    there_was_one_intersection = 1
     $pZ$  = depth[ID]
     $p$  = ID
    iteration ++
    continue
  end if
end if
if (there_was_one_intersection = 1) then
  oracle_circuit = Algorithm 5
else
  oracle_circuit = Algorithm 2
end if

```

The Min algorithm also changes the variable *there_was_one_intersection* which decides which oracle is going to be executed. If this variable is 0, it means that there is no primitive marked as intersected and the oracle that will run in the next iteration is the oracle without the comparison of the z coordinate because the next primitive to be marked as intersected (and its depth) will be automatically saved. If the variable is 1, it means that there has been 1 primitive marked as intersected and we have to run the oracle with the comparison of the depths because now we have a value for z .

To achieve a Q_SearchMin algorithm we use an approach based on Durr et al. [13] and it is given as Algorithm 7. The algorithm is based on the Q_Search algorithm (Algorithm 4) but incorporating the Min algorithm (Algorithm 6).

This algorithm will run c times and doesn't stop when *intersected* = *true* like the other algorithms, because here, even if we find one primitive (*intersected* = *true*), there may be more with a lower depth, so we need to keep searching. The algorithm starts by executing the first 2 steps of the Q_Search algorithm: measuring 1 primitive randomly and verifying if it is an intersection (this second step is hidden as Min's first step). Afterwards, the Min

Algorithm 7 Q_SearchMin: Quantum minimum algorithm

```

 $p = -1$ 
 $pZ = 2^{n_{\text{bounds\_qbs}}}$ 
iteration = 0
 $l = 0$ 
 $d = 1.3$ 
there_was_one_intersection = 0
while iteration <  $c$  do
    ID = Q_SampleUniformDistribution(primitives)
    Min( $x, y, pZ$ , primitives) {Algorithm 6}
     $l = l + 1$ 
     $S = \min(d^l, \sqrt{N})$ 
     $r = \text{rand}(1, \dots, S)$ 
    ID = Q_RayCast( $x, y$ , primitives,  $r$ ) {Algorithm 1}
    Min( $x, y, pZ$ , primitives) {Algorithm 6}
end while

```

algorithm is called to check if the primitives' depth is lower than the current. If the Min algorithm has found an intersecting primitive with a lower depth than the current, it will skip the rest of the code in the loop and start a new iteration (continue command) meaning we have found a correct primitive with random sampling and don't need to run the oracle in this iteration. Up until here we haven't run the oracle, we just chose a random primitive. After these steps, we carry out the next steps of the Q_Search algorithm: calculating the number of times the Grover's algorithm will run and applying Grover's algorithm (Q_RayCast). Next, we verify the intersection and see if the primitive's depth is lower than the current with the Min algorithm.

We've taken the same scene from the previous case (Figure 21) and ran Algorithm 7 in the simulator. The result are presented in Figure 30.

Analyzing the result, we see that they're 100% correct with the number of iterations, c , at 3; for every pixel with an intersection, the primitives with the lowest depth have been correctly measured.

Table 10 shows the depth and number of gates of the circuits used for this geometry. We still see the values increasing by 2 when r is doubled. We also see that these numbers show a big difference before and after the value of z is set to the value of the depth of a primitive. This is caused due to the choice of the oracle. When an intersection has not been found, the executed oracle is the one in Algorithm 2 which doesn't account for the depth data, thus needing less calculation, therefor fewer gates and circuit depth. When an intersection has

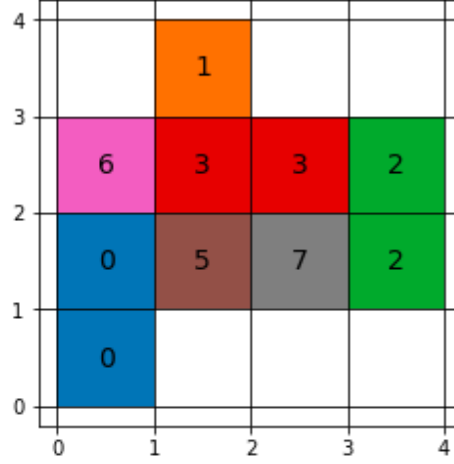


Figure 30: Results for the multiple solution visibility case.

already been found, the oracle to be executed in the one in Algorithm 5 which requires more calculations.

	Before z		After z	
	Avg depth	Avg #gates	Avg depth	Avg #gates
$r = 1$	2410	4211	2870	5010
$r = 2$	4748	8289	5685	9915

Table 10: Depth and number of gates for the visibility case.

4.4 ON A REAL QUANTUM MACHINE

To attempt to execute this algorithm on a real quantum machine, several things have to be considered. The machine used for the execution was the 20 qubit IBM Q Poughkeepsie quantum device. The circuits have to be mapped onto the real machine, which has limited connectivity among physical qubits, as shown in Figure 31. This restriction implies that more gates have to be added in order to swap logical qubits among physical qubits, such that non connected qubits can still be entangled. This, in addition to an already large executable circuit gate count and depth, results in a noisy circuit. Adding the fact that execution times must be kept short due to qubits dephasing and limited coherence times among qubits, a long circuit is going to make it difficult to attain good results.

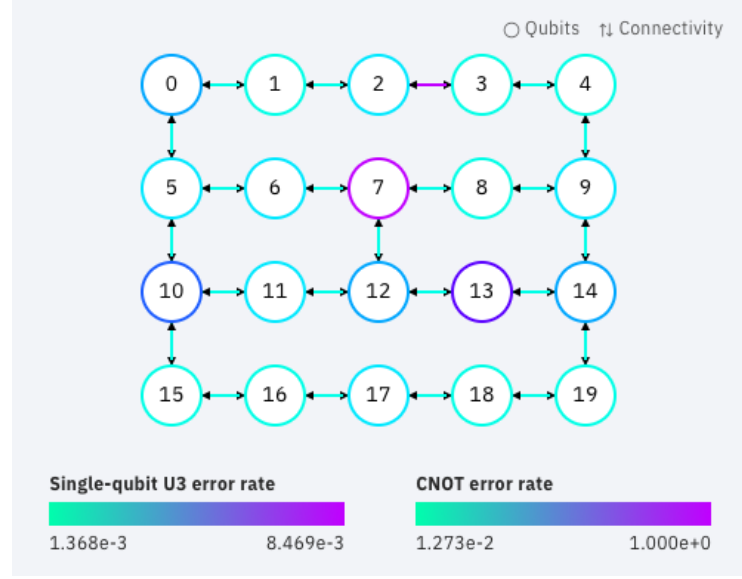


Figure 31: Connectivity of the qubits in the IBM Q Poughkeepsie real quantum machine.

To allow execution on a real machine, a simpler circuit was prepared for the scene depicted in Figure 32; each primitive has coordinates equal to its ID, therefore the operators mX, mY, MX, MY that generate the primitive bounds are not required. The ray coordinates can be compared directly with the primitive ID and the bounds representation $|b\rangle$ is not required (compare with Figure 15).

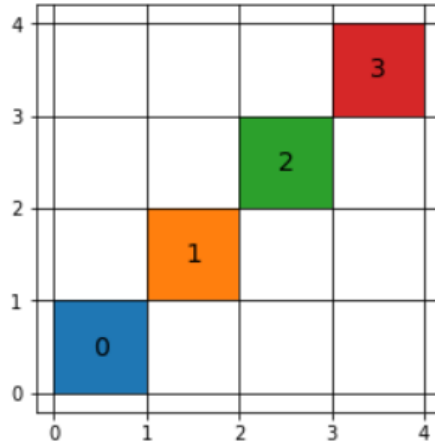


Figure 32: Geometry of the scene to execute on the real quantum machine.

The table above presents the depth and number of gates used to run the circuit on the real quantum computer. We can check that we've managed to reduce these numbers in

order to be able to run the program and also in a reasonable amount of time. We've also succeeded in decreasing the number of qubits by not needing qubits for the coordinates and this geometry used 7 qubits.

	Depth	#gates
Minimum	219	326
Maximum	472	717
Average	321	499

Table 11: Depth and number of gates of circuit executed on the real quantum machine.

Although we managed to reduce the number of qubits, gates and depth, we still didn't get good results when doing 2048 shots for each pixel with an intersection. These results are presented in Figure 33.

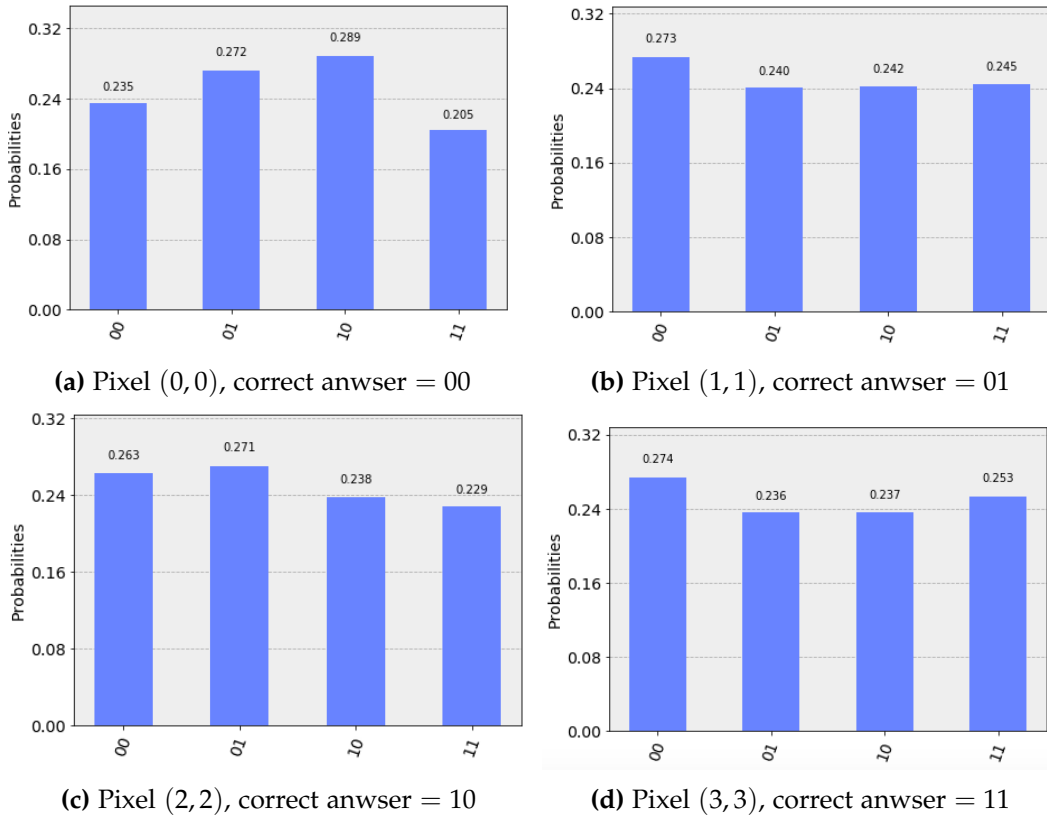


Figure 33: Results retrieve from IBM Q's real quantum computer for pixels with an intersection.

Given the almost uniform distribution of measurements across all the primitives, the circuits are not usable on their current form. Advanced techniques of quantum error correction or measurement error mitigation might help and will have to be investigated in the future.

CONCLUSION

In this dissertation we aimed to present practical algorithms and implementations of ray casting for visibility testing and occlusion from an orthographic camera based on quantum algorithms. Using Grover’s algorithm to solve this problem has been discussed and thought of in the past but never tested and demonstrated in practice. We were able to advance with those ideas in practice, building quantum circuits using the technology available at the moment. The algorithms proposed in this work have been evaluated both via simulation of a quantum machine, indicative of future quantum computing capabilities, and in a simplified form on a real quantum computer. We were able to gather some good results in the simulator, however, in the real quantum computer, the results still need a lot of improvement. While these implementations are not yet fully fledged and further work is required to increase the probability of selecting intersecting primitives, the results show the potential of quantum computation for computer graphics through a reduction of complexity for core operations in a rendering pipeline.

5.1 PROSPECT FOR FUTURE WORK

One of the greater obstacles in this work is running Grover’s algorithm without knowing the number of solutions (number of intersecting primitives). This is critical because, not knowing this, we can’t accurately calculate the number of Grover iterations needed to maximize the probability of a right primitive. One of the ways to improve this algorithm in the future is to investigate quantum counting schemes to calculate the number of iterations required for Grover’s Algorithm when having an unknown number of solutions. The complexity of such quantum counting algorithms will have to be carefully weighted against the current $\sqrt{N}$ complexity, to guarantee that there is a quantum advantage with respect to a classical approach.

Another improvement, to be done when quantum machine technology allows, is to extend this work to generalized geometric setups, such as triangles instead of axis aligned quads, and arbitrary ray directions. This involves some trigonometric calculations and will likely implicate significantly more qubits and a larger circuit depth, which additionally

indicates the need for more sophisticated noise tolerance in quantum algorithms for graphics. This may be achieved via development of domain specific quantum error correction schemes.

The oracle used throughout this work uses the same set of qubits for all the primitives coordinates; mX, mY, MX, MY, Z . This implies generating one of the coordinates, testing it, uncomputing that coordinate and proceed with the next one. The rationale here is to use the minimum number of qubits, but this results on a very deep circuit due to the repetitive need to compute and uncompute the coordinates. An alternative is to use different sets of qubits for each of the 5 coordinates; now these are computed only once, before the Grover's circuit, and remain untouched until the end of the quantum program. This will result on a much shallower circuit (less gates on the critical path) , but wider in terms of the number of qubits. Future work will include studying the impact of these two alternatives on current noisy quantum machines. A third possibility includes multiplexing all the coordinates on a superposition on the same set of qubits; additional ancillary selector qubits have to be used to indicate which coordinate is represented on each term of the superposition.

We believe that in the future, with these improvements and when quantum computing technology is more advanced, a quantum algorithm for ray tracing has a good chance of being adopted.

BIBLIOGRAPHY

- [1] Richard P. Feynman. Simulating Physics with Computers. *International Journal of Theoretical Physics*, 21(6):467–488, 1982.
- [2] David Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London*, 400:97–117, 1985.
- [3] 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, pages 212–219, New York, NY, USA, 1996. ACM.
- [4] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26:1484–1509, 1997.
- [5] Aymard de Touzalin, Charles Marcus, Freeke Heijman, Ignacio Cirac, Richard Murray, and Tommaso Calarco. Quantum manifesto: A new era of technology. Technical report, European Commission, 2016. <http://qurope.eu/manifesto>.
- [6] Emily Grumbling and Mark Horowitz. Quantum computing: Progress and prospects. Technical report, National Academies of Sciences, Engineering, and Medicine, 2019. <https://www.nap.edu/catalog/25196/quantum-computing-progress-and-prospects>.
- [7] Andrew Glassner. Andrew Glassner's Notebook: Quantum Computing - Part 1. *IEEE Computer Graphics and Applications*, 21(4):84–92, 2001.
- [8] Andrew Glassner. Andrew Glassner's Notebook: Quantum Computing - Part 2. *IEEE Computer Graphics and Applications*, 21(5):86–95, 2001.
- [9] Andrew Glassner. Andrew Glassner's Notebook: Quantum Computing - Part 3. *IEEE Computer Graphics and Applications*, 21(6):72–82, 2001.
- [10] Marco Lanzagorta and Jeffrey Uhlmann. *Quantum Computer Science*. Synthesis Lectures on Quantum Computing. Morgan & Claypool, 2009.
- [11] Simona Caraiman. Quantum computer graphics algorithms. *Buletinul Institutului Politehnic din Iasi, Sectia Automatica si Calculatoare*, 62(4):21–38, 2012.
- [12] Michel Boyer, Gilles Brassard, Peter Hoyer, and Alain Tapp. Tight bounds on quantum searching. *Fortschritte der Physik*, 46(5):493–505, 1998.

- [13] Christoph Durr and Peter Hoyer. A quantum algorithm for finding the minimum. In *in LANL e-print quantph/9607014*, <http://xxx.lanl.gov>, 1996.
- [14] Eric R. Johnston. Quantum supersampling. In *ACM SIGGRAPH 2016 Talks*, SIGGRAPH '16, pages 38:1–38:1, New York, NY, USA, 2016. ACM.
- [15] Juan Carlos García-Escartín and Pedro Chamorro-Posada. Equivalent quantum circuits. 2011.
- [16] R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, July 1961.
- [17] Arthur Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference, AFIPS '68 (Spring)*, pages 37–45, New York, NY, USA, 1968. ACM.
- [18] Ahmed Younes. Strength and weakness in grover’s quantum search algorithm. 2008.
- [19] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum amplitude amplification and estimation, 2000.