Quantum Rendering

Marco Lanzagorta\textsuperscript{a}, Jeffrey K. Uhlmann\textsuperscript{b} and Richard B. Gomez\textsuperscript{c}

\textsuperscript{a}Center for Computational Science, Naval Research Laboratory
\textsuperscript{b} Dept. of Computer Engineering and Computer Science, University of Missouri-Columbia
\textsuperscript{c} School of Computational Sciences, George Mason University

ABSTRACT

In recent years, computer graphics has emerged as a critical component of the scientific and engineering process, and it is recognized as an important computer science research area. Computer graphics are extensively used for a variety of aerospace and defense training systems and by Hollywood’s special effects companies. All these applications require the computer graphics systems to produce high quality renderings of extremely large data sets in short periods of time. Much research has been done in “classical computing” toward the development of efficient methods and techniques to reduce the rendering time required for large datasets. Quantum Computing’s unique algorithmic features offer the possibility of speeding up some of the known rendering algorithms currently used in computer graphics. In this paper we discuss possible implementations of quantum rendering algorithms. In particular, we concentrate on the implementation of Grover’s quantum search algorithm for Z-buffering, ray-tracing, radiosity, and scene management techniques. We also compare the theoretical performance between the classical and quantum versions of the algorithms.

Keywords: Quantum Computing, Quantum Algorithms, Computer Graphics, Range Searching, Multi-dimensional Data Structures.

1. INTRODUCTION

Quantum computing (QC) is becoming an increasingly important new model of computation because it offers algorithmic solutions that are more efficient than what is provably possible for some problems under the classical computing (CC) model. Unfortunately, the advantages of many high-profile QC algorithms over CC algorithms have in the past been significantly misrepresented, especially in the popular media. For example, Grover’s algorithm for searching for a specified item in an “unsorted database” of \(N\) items in \(O(N^{1/2})\) time is often compared to the most naive CC algorithm, which involves performing a brute force \(O(N)\) examination of the stored items. This is misleading because regardless of what kind of search algorithm is to be used, \(O(N)\) time must be expended to initially store the database items. Thus, \(O(N)\) time is required to store the items in a quantum register (QR) to support the subsequent application of Grover’s \(O(N^{1/2})\) search algorithm. Analogously for the CC case, \(O(N)\) time is sufficient to store the database items in a hash table, but in contrast to the expected \(O(N^{1/2})\) complexity of Grover’s QC search algorithm, hashing supports searches in expected \(O(1)\) time. Thus, the best CC algorithm for this particular problem appears to be vastly superior to the best known QC algorithm in terms of asymptotic query-time complexity.

Although a fair and objective comparison of QC and CC algorithms often yields inconclusive results, e.g., there are time/space tradeoffs and questions about running time coefficients, there are some extremely important problems for which there is solid reason to believe that QC methods will prove to be superior to the best possible CC alternatives. In this paper we consider a class of problems that arise in computer graphics, simulation, and visualization applications for which CC algorithms have believed or established lower bounds that make them impractical for most dataset sizes of interest. In this paper we provide a brief overview of Grover’s algorithm and then show how it applies to general multidimensional search problems of relevance to large-scale graphics,
simulation, and visualization applications. We then consider four well-known computer graphics algorithms and discuss how Grover’s algorithm can be used to implement their quantum counterparts. The algorithms examined are Z-Buffering, Ray Tracing, Radiosity and Level of Detail management—all of which are well known and widely used by the computer graphics community. We conclude with a summary of the relative computational complexities of the classical algorithms and their quantum counterparts for these problems.

2. GROVER’S ALGORITHM

There are two ways to design a quantum algorithm for replacing a given classical algorithm. The simplest approach is to look for segments of the classical algorithm that are similar to any of the well-known algorithms that have an optimal quantum counterpart. A more sophisticated and potentially more powerful approach, however, is to look for hidden structures in the functional space of the classical algorithm that can fully exploit the quantum nature of quantum computers*. Grover’s algorithm is a general quantum search algorithm developed to satisfy queries from an unsorted and unstructured list of $N$ records in $O(N^{1/2})$ time. As has been mentioned, this complexity does not compare well to the best CC algorithm for the case of searching for a specified element, which has $O(1)$ complexity. And if the CC algorithm is permitted to expend $O(N\log(N))$ time to structure the database to support binary search, then a variety of queries can be supported in $O(\log(N))$ time, which is significantly better than $O(N^{1/2})$.

Although the complexity of Grover’s algorithm, which is provably optimal under a variety of general assumptions about QC computability, is rather disappointing when compared with the best CC algorithms for 1-dimensional database queries, it does offer two important advantages. The first is that the $O(N^{1/2})$ complexity can be achieved for almost any query class for which element satisfaction can be verified in $O(1)$ time. This includes many types of queries for which the best CC algorithm scales worse, e.g., $O(N)$. The second advantage of the QC solution is that the storage/space complexity is only $O(\log(N))$, rather than $O(N)$ or worse for the best possible CC solution. For example, a quantum register containing $m$ qubits would be able to store $2^m$ records, so only 30 qubits are necessary to store a billion records while the CC solution requires gigabytes of storage. Clearly the QC savings in storage are enough to make it very attractive for large-scale graphics applications, but it turns out that it offers query-time advantages for 3-dimensional queries that are important in many graphics and simulation applications.

CC algorithms for satisfying range queries, interval intersection queries, and many other types of 1-dimensional retrieval operations achieve $O(\log(N))$ complexity by using either implicitly or explicitly a binary search tree. Unfortunately, multidimensional generalizations that are constrained to using minimal $O(N)$ space, such as the k-d tree, provide only rootic query time that depends on the number of dimensions. Specifically, orthogonal range queries and box intersection queries in D dimensions require $O(N^{1-1/D})$ time in the worst case, which is $O(N^{2/3})$ in three dimensions. Obviously this represents a lower bound for more general types of queries involving non-orthogonal objects, and this bound is worse than the $O(N^{1/2})$ complexity of Grover’s algorithm. The value of Grover’s algorithm in 4D simulations in which time represents an additional dimension is even more substantial as the CC complexity for interaction detection has a lower bound of $O(N^{3/4})$, and no algorithm exists that achieves this lower bound. In fact, it is believed that $O(N^{1-1/D})$ cannot be achieved in $O(N)$ space except in the special case of orthogonal query objects.

Recent analysis* suggests that the best CC spatial query algorithms can achieve polylogarithmic query time, i.e., $O(\log^c(N))$ for some constant $c > 1$, only at the expense of nonlinear space that has a lower bound of $O(N\log(N))$. It is likely that CC algorithms for most general types of spatial queries will have much higher space complexities, but even the lower bound complexity suggests that space will become an obstacle for large values of $N$. Thus, the only CC algorithms that achieve query-time complexity superior to that of Grover’s algorithm do so at the expense of nonlinear space. This makes the $O(N^{1/2})$ query complexity and $O(\log(N))$ space complexity of Grover’s algorithm highly attractive. Unfortunately, achieving this complexity for many widely-used graphics rendering algorithms is nontrivial because they must be formulated in a special form in order for Grover’s algorithm to be applied. The following sections develop such formulations for some of the most important rendering operations.

*As an example, Shor’s algorithm uses the functional space structure to reduce a factorization algorithm into a periodicity problem.
3. QUANTUM RENDERING

In computer graphics, each object that needs to be rendered is typically decomposed into several hundred or thousands of smaller polygons or other surfaces. These polygons (or surfaces) are stored in a database that is often very large in practical applications, e.g., at least a few hundred thousand elements depending on the complexity of the scene. In this context, a scene is simply a collection of many objects. To render a scene, the visualization system applies a variety of rendering algorithms on each individual element of the database. These rendering algorithms have the common feature of performing several searches over the entire database of objects in a scene. Considering the large number of elements in the database, the search process tends to be the most significant bottleneck in the rendering pipeline. Therefore, these search operations are prime candidates to be modified and optimized using Grover’s quantum search algorithm.

The question of whether quantum rendering algorithms can be defined at all has been considered before; however, no QC algorithm has been previously developed that offers any complexity improvements over existing CC query algorithms, and in fact no QC algorithm has previously been developed that even matches the query-time complexity of the best CC algorithms. In the following subsections we remedy this deficiency by providing QC algorithms for several important rendering and scene management applications. The results of our analysis include QC algorithms with superior worst-case complexity compared to their CC counterparts.

3.1. Z-Buffering

One of the simplest methods used to determine what polygons are visible in a scene that needs to be rendered is the Z-Buffering algorithm. This is a very simple, but effective, algorithm. The first step is to scan-convert each polygon in the scene. This means that for each polygon, we calculate its projection onto the screen, and determine what pixels in the screen need to be shaded according to the color of the polygon. Then, for each shaded pixel, we record the distance to the polygon. The algorithm iterates over each polygon in the database, and if for a given pixel there is a new polygon that is closer, we update the color of the pixel to represent the new polygon. In other words, for each polygon we draw its image on the screen and keep a buffer of values that represent its distance to the screen. If the polygon is the closest to the screen, then the respective pixels are shaded with the color of such a polygon (Figure 1). Therefore, the entire operation requires $O(pN)$ time, where $p$ is average number of pixels in the projection area per scan-converted polygon. In other words, the problem is solved in $O(p)$ time per polygon.

This is not a search problem per-se, but it can be modified in such a way that we can solve it using Grover’s algorithm. In a sense, this problem is rather similar to some graph problems that have been studied within the context of quantum computing. Let us suppose that for a pixel, for all the polygons in the scene, we calculate the values “$Zi”$, which represents the distance between the pixel and polygon “$i”$. Then, for this pixel, we need to determine which $Z$ is the smallest one. To this end, let us create a $|\Psi\rangle$ quantum state in the following way:

$$|\Psi\rangle = \alpha|Z1, Z2, Z3,..., ZN > + |Z2, Z3, Z4,..., Z1 > + \ldots + |ZN, Z1, Z2,... > \quad (1)$$

So, $|\Psi\rangle$ is a uniform superposition of $N$ states, where each state is characterized for having a different $Z$ value. Now we define a function $f$ such that:

$$f(|Za, Zb, Zc >) = 1 \quad \text{if } Za > Zj \text{ for all } j \quad (2)$$

$$f(|Za, Zb, Zc >) = 0 \quad \text{any other case} \quad (3)$$

Thus, it is straightforward to apply Grover’s algorithm to find the state where the first element represents the closest distance to the screen, which gives the polygon that needs to be used to shade the pixel. This process has to be repeated once for each polygon. Therefore, the complexity of the quantum Z-Buffering algorithm is $O(N^{1/2})$ per pixel for an overall complexity of $O(PN^{1/2})$, where $P$ is the total number of pixels. Although the factor of $P$ is very large in the QC complexity, the QC algorithm scales much better with the number of polygons $N$ than does the corresponding CC algorithm. Thus, as $N$ grows into the billions, the QC algorithm achieves better query time complexity using $O(\log(N))$ space than the best CC algorithm using $O(N)$ space. The appeal of the QC algorithm with 40+ qubits can store and efficiently render scenes containing trillions of objects. This is the scale of problem that is relevant to many virtual reality applications of interest, e.g., involving a complete virtual representation of an urban environment.
3.2. Ray Tracing

Ray tracing is one of the most commonly used rendering techniques.\(^8\) A ray tracer determines the visibility of surfaces by tracing imaginary rays from the viewer’s eye to the objects in the scene. The intersection between a ray and an object determines the shade of the pixel, as seen in Figure 2. This process has to be performed for each pixel on the screen. Therefore, for each pixel on the screen, the ray tracer determines if there is an intersection between the ray and any object in the scene. If the ray intersects many objects, only the one closest to the screen is rendered. Thus, the ray tracer requires \(O(N)\) operations per ray, as the intersections need to be determined for each of the \(N\) objects in the scene.

Although a variety of CC data structures exhibit much better performance than what the worst-case complexity might indicate, most of the data structures have been primarily analyzed empirically on special types of datasets of spatially proximate objects with small level-of-detail variance, e.g., objects in a single room. There is reason to believe that their performance for scenes involving views of large areas, e.g., through the trees and foliage of a virtual forest, will be substantially worse.

A ray tracer can be implemented in such a way that the objects in the scene reflect the original ray several times (Figure 3). This method allows the ray tracer to realistically model reflected and refracted light. However, each new ray in the iteration requires \(O(N)\) further steps. So, if we consider a ray tracer with two secondary rays, as the one shown in Figure 3, the rendering time triples.

Because of the ray-tracing algorithm’s intrinsic searching nature, which requires a search for intersections between a ray and a collection of \(N\) objects, ray tracing is a prime candidate to be optimized by means of Grover’s quantum search algorithm. Before we implement a quantum ray tracer, let us introduce the concept of bounding boxes. A bounding box is the smallest sphere or box that completely surrounds each object in the scene (let us recall that these objects are small polygons or surfaces). Therefore, the ray tracer can be simply implemented to calculate the intersection between rays and bounding boxes. And once we determine the object being intersected, we can perform further refinements to properly shade the corresponding pixel. The intersection between a ray and a bounding box is a very simple procedure, and it involves basic analytical geometry to calculate the intersection (if any) between a line and a cube (or a sphere).
Figure 2. The ray-tracing algorithm calculates the intersection between imaginary rays emitted by the eye and the objects in the scene.

Figure 3. Ray tracing with multiple reflections.
To implement a quantum ray tracer, we create a state $|\Psi\rangle$ that encodes all the bounding boxes in the database. We can use Hadamard gates to create a uniform superposition of states:

$$|\Psi\rangle = \alpha(|00..01\rangle + |00..10\rangle + ...).$$

(4)

where each element in the superposition marks or encodes one of the different bounding boxes in the scene. In other words, the state $|\Psi\rangle$ is made of $N$ “segments,” where each segment encodes a bounding box (the coordinates of its center and radius, for example). This superposition of states is used for each ray in the process. We also need to define a function $f$ such that:

$$f(x) = 1 \text{ if } x \text{ intersects the ray}$$

(5)

$$f(x) = 0 \text{ any other case}$$

(6)

After adding some ancillary qubits, we can use quantum parallelism to evaluate $f(x)$ for each element of the superposition (each object in the scene) in a single step. From here it is straightforward to use Grover’s algorithm to determine what bounding boxes intersect a ray in $O(N^{1/2})$. Therefore, as we would have expected, we have obtained a square root improvement over the classic algorithm.

Even more, we recall that ray tracing of a ray with $S - 1$ reflections is accomplished with a classical computer after $O(SN)$ steps. However, with a quantum computer such a process can be performed with only $O(SN^{1/2})$ operations.

### 3.3. Radiosity

As we mentioned before, ray tracing is an excellent method to simulate reflected and refracted light. However, a ray tracer can only approximate in a very crude and expensive way diffuse light (the light scattered by opaque materials). Radiosity is a method that models extremely well diffuse light but offers a poor representation of reflected light.² In other words, ray tracing is a good method to render images of shiny and semi-transparent materials, while radiosity is optimal to render opaque surfaces.

The radiosity method is based on the principle of energy conservation. Radiosity embodies the idea that the total energy radiated by an opaque material is equal to the energy naturally emitted by the material plus the reflected energy:

$$\text{Radiosity} = \text{emitted energy} + \text{reflected energy}$$

where:

$$\text{Reflected energy} = \text{reflection coefficient} \times \text{total energy incident on the object from all other objects in the scene.}$$

This can be written in the form of the so-called radiosity equation, given by:

$$B_i dA_i = E_i dA_i + \zeta_i \int B_j F_{ji} dA_j$$

(7)

If we consider the scene to comprise a discrete number of individual objects (as we have already stated), the radiosity equation takes the form:

$$B_i A_i = E_i A_i + \zeta_i \sum_j B_j F_{ji} A_j$$

(8)

In these equations $B$ is the radiosity of each element, $A$ is the area of the element, $E$ is the emitted energy and $F$ is the form factors matrix that determines the incident energy which was emitted by the other elements in the scene. The radiosity equation can also be rewritten in matrix form as:

$$M \cdot B = E$$

(9)
where $\mathbf{M}$ is a form factors matrix. If we know $\mathbf{E}$ and $\mathbf{M}$, we can use the Gauss method to solve the system for the radiosity vector $\mathbf{B}$ in $O(N)$ steps. $\mathbf{E}$ is an intrinsic property of the material and therefore it is a known quantity. The form factors $\mathbf{M}$ are unknown but can be calculated by performing ray tracing between all the elements in the scene, which requires $O(N^2)$ steps in the classical case.

Now, once we know $\mathbf{B}$, we still need to calculate the shading of each pixel on the screen, which is accomplished by performing a technique very similar to ray tracing. So, the radiosity of a pixel $P$ (which can be translated into a shading value) on the screen is given by:

$$\mathbf{B}(P) = \mathbf{B}\mathbf{N}(xp)$$

where $\mathbf{N}(xp)$ is a vector that depends on the pixel $P$ and the point of the scene that it intersects $xp$ (Figure 4). Each element of $\mathbf{N}(xp)$ determines how much the radiosity of the element $i$ contributes to the shading of the pixel $P$. So, for example, if an object is not visible from $P$, the value of $\mathbf{N}$ for that element is zero (it does not contribute to the shading of the pixel). To calculate the values of $\mathbf{N}(xp)$ we need to apply a ray tracer that determines the visibility of each object in the scene from pixel $P$, so it takes $O(N)$ operations to complete.

Therefore, the shading of each pixel involves the following three operations:

1. An $O(N^2)$ operation to calculate the form factor matrix.
2. An $O(N)$ operation to determine $\mathbf{N}$.
3. An $O(N)$ operation to solve the radiosity matrix equation for $\mathbf{B}$.

As already discussed, (1) and (2) are operations similar to ray tracing, which we already know how to implement on a quantum computer. So, using a quantum ray tracer we can perform task (1) in $O(N)$ steps and task (2) in $O(N^{1/2})$ steps. However, we cannot further optimize task (3). Consequently, quantum radiosity can be performed in $O(N)$ steps, in contrast to the $O(N^2)$ time required in the classical case.

The speed up achieved by using a quantum radiosity algorithm can be used to increase the realism of the rendered images. As discussed before, radiosity does not model accurately reflected lights. However, there is an extension to the radiosity method, known as ray-traced radiosity that correctly simulates reflections and refractions. As the name indicates, ray-traced radiosity implements both techniques into a single, more complex method. Ray traced radiosity is a two pass method. On the first pass we perform radiosity as usual, and on the second pass we take reflections into account. On the second pass, for each pixel, we shoot several rays and obtain
the radiosity of the reflected rays (Figure 5). If we use \( M \) rays per pixel, the second pass requires \( O(M \times N \times S) \) operations if we consider S-1 reflections. Evidently, this is an extremely time consuming algorithm. However, a quantum ray traced radiosity algorithm is straightforward to implement, and we obtain a square root speed up as usual. The second pass is then performed after \( O(M \times S \times N^{1/2}) \) steps.

### 3.4. Level of Detail

Level of Detail (LOD) is not a rendering algorithm, but a scene management method. LOD is based on the idea that, in many applications, we can trade fidelity for speed. For example, some details of an object may not be visible from the viewer’s point of view, or these details may be so small that are imperceptible. At other times, we may purposely need a low-resolution image of the object to perform a quick visual exploration. In these cases, we can use a LOD algorithm that determines when and for how much we can simplify a given model in the scene.

If a polygonal mesh is used to describe the models, the LOD method determines what vertices in the mesh are necessary to represent the model according to the given circumstances. If some vertices are not required, the LOD eliminates them from the mesh. To determine if a vertex is necessary or not, the LOD function evaluates an error function \( \epsilon(\nu) \) for each vertex \( \nu \). If the error associated with a vertex is less than a certain threshold \( \delta < \epsilon \), then the vertex can be removed. This procedure has to be repeated for all the vertices in the mesh, for each frame being rendered. For increased performance, all the polygons are encoded in a hierarchical tree structure that determines which vertices are active at a given time. In this case, if the error is smaller than the threshold, the vertex may be expanded to include lower portions of the hierarchical tree. Nevertheless, the process has to be performed over the entire set of active vertices and requires \( O(N) \) steps, where \( N \) is the number of active vertices.

The quantum version of a LOD method is easy to implement. For each frame, we prepare a state \( |\Psi\rangle \) that encodes the entire mesh made of \( N \) vertices. We can use Hadamard gates to obtain a uniform superposition of the state. Each element of the superposition represents a vertex in the mesh. Now, we define a function \( f \) such that:

\[
  f(\nu) = \begin{cases} 
  1 & \text{if } \epsilon(\nu) < \delta \\
  0 & \text{any other case}
  \end{cases}
\]

We add some ancillary qubits and use quantum parallelism to evaluate \( f(\nu) \) for each element of the superposition in a single step. Then, it becomes straightforward to apply Grover’s algorithm to search for an unknown number of vertices with an error small enough that makes them removable. This operation can be performed with
$O(N^{1/2})$ steps, in contrast to the $O(N)$ steps required in the classical case. Of course this procedure is used for each frame being rendered and $N$ is the number of active vertices in the hierarchical tree structure.

4. CONCLUSIONS

This paper has provided quantum algorithms for rendering applications that are superior in one or more respects to their classical counterparts. The analysis presented in this paper has been limited to the use of Grover’s quantum search algorithm in computer graphics, but it is important to note that it is possible for the functional space of the computer graphics problems to have hidden structures that could be used to further increase the performance of the quantum rendering algorithm. However, the search for these hidden structures is a complex task, and indicates the need for further research to be performed.

Acknowledgements: The authors would like to thank Mil Mascaras for useful conversations.

REFERENCES