Math. Struct. in Comp. Science (2010), *vol.* 20, *pp.* 1117–1125.
doi:10.1017/S0960129510000411

Quantum algorithmic methods for computational geometry

MARCO LANZAGORTA[†§] and JEFFREY UHLMANN[‡]

[†]*Advanced Information Systems, ITT Corporation,*
Alexandria, Virginia 22303, U.S.A.
Email: marco.lanzagorta@itt.com
[‡]*Department of Computer Science, University of Missouri-Columbia,*
Columbia, Missouri 65211, U.S.A.
Email: uhlmannj@missouri.edu

Received 30 June 2009; revised 14 July 2010

In this paper we develop novel quantum algorithms based on Quantum Multi-Object Search (QMOS) for convex hulls and general object intersection reporting, with applications to computer graphics. These algorithms are developed and described using standard concepts from computer science by encapsulating the physics of quantum computation within black-box subroutines.

1. Introduction

Assuming that its theoretical foundations are correct, it is known that the quantum computation (QC) model properly subsumes the classical Turing model of computation. The increased power of QC stems from its ability to manipulate a set of N entangled states with sublinear complexity. In other words, the theoretical QC model is able to exploit a kind of parallelism that is unavailable within the classical computation (CC) model.

Although there are many open questions about how best to construct a practical quantum computer, there are no obstacles to the theoretical derivation of quantum-based algorithms. For the computer scientist the details of quantum physics are less important than knowing what fundamental operations can be performed more efficiently with QC than is possible with CC. These operations can be exploited to develop new algorithms that are more efficient than the best possible classical alternatives.

In this paper we describe a general Quantum Multi-Object Search (QMOS) algorithm for retrieving the k elements from a dataset of size N that satisfy a given query using fundamental quantum algorithms for searching and counting. Although our QMOS algorithm has optimal $O((kN)^{1/2})$ complexity[¶], it is not necessarily the fastest or most practical to realise on actual quantum hardware. Rather, it is presented as a pedagogical example of

[§] This work was partly supported by the Office of Naval Research under the Basic Research Challenge grant N00014-08-1-1248.

[¶] Note that the complexities given for quantum algorithms in this paper are expected-case and should be regarded as soft in the sense that polylogarithmic factors that depend on register size are ignored because there is no consensus in the literature regarding what constitutes unit-cost operations in the QC model, for example, in the same way as arithmetic and other fixed-sized register operations are for CC analysis.

how efficient quantum algorithms can be developed from building blocks in a manner comparable to classical algorithm development. The sublinear query complexity of QMOS for general queries is clearly impossible for any classical algorithm and demonstrates the exciting potential of the QC model. We compare the complexity of new quantum algorithms to the best possible CC alternatives for a variety of computational geometry and spatial search problems arising in database and computer graphics applications. Of significant practical interest is a quantum-based online intersection-detection algorithm and a new quantum convex hull algorithm.

2. Grover's Algorithm

Grover's algorithm takes as inputs a dataset S of size N, an indicator function f (referred to as an *oracle*) with assumed $O(1)$ complexity and a quantum register r of $\lceil \log_2(N) \rceil$ qubits. After an application of Grover's algorithm, the quantum register r will contain the index of an element of S that is a solution according to the oracle. If it is known that there is a unique solution within S, the complexity of Grover's algorithm is $O(N^{1/2})$ (Nielsen and Chuang 2000).

The $O(N^{1/2})$ complexity of Grover's algorithm for general f is better than the $O(N)$ complexity of a classical search and clearly exploits some kind of parallelism to avoid having to apply f to each of the N elements of S. Grover's algorithm is provably optimal within QC when there is no *a priori* knowledge about the oracle. For particular classes of queries, however, it is possible to achieve better complexity. This should not be surprising since classical binary search can satisfy a variety of inequality-defined search queries in $O(\log(N))$ time, and hashing can satisfy exact-match queries in expected $O(1)$ time.

The key to Grover's algorithm is that the quantum register is able to hold the N indices in a quantum superposition. This superposition of states can be manipulated in parallel so that after $O(N^{1/2})$ references to the oracle, a read operation applied to the quantum register will give the index of the desired solution with high probability[†]. An essential characteristic of quantum computation is that a read (measurement) operation applied to a superposition causes the superposition to reduce to a single state. In other words, after a read operation, the quantum register is equivalent to a classical register containing the solution state.

An important issue that cannot be overlooked is that n qubits will store a superposition of 2^n indices, so if N is not a power of 2, the indexed array will have to be padded with 'dummy' keys that the oracle will recognise as non-solutions. This can be achieved using special dummy keys or by having a bit vector to mark which elements of the array are dummies. In either case, the oracle will expend only $O(1)$ time to perform the test.

In the case of an exact-match query, the oracle f simply checks whether a given index references an element of S equal to the query value (which requires the indexed set S to be part of the implementation of the oracle). Grover's algorithm applies f in such a way as to ensure that a subsequent read of r will give the desired index with high probability.

[†] Quantum algorithms are inherently probabilistic, but the probability of error can be made aribitrarily small by iterating the algorithm.

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

What is surprising from a CC perspective is that the oracle is not applied sequentially to each of the N indices, nor is it even explicitly applied to any particular index. Although f is defined to test a single index, it is applied in Grover's algorithm to the superposition of all indices in r. This is where quantum parallelism is exploited.

Grover's algorithm for identifying the unique solution in S according to the oracle f can be summarised as follows:

(1) A quantum register r is initialised to a superposition of all N indices corresponding to elements of a data set S. This step has complexity $O(1)$.
(2) The superposition in r is manipulated to amplify the probability of measuring a solution state according to the oracle function f. This step has complexity $O(N^{1/2})$.
(3) A read of the value in r reveals the desired solution index with high probability. This step has complexity $O(1)$.

As already mentioned, a classical (or quantum) algorithm can achieve better complexity for certain classes of queries by storing the dataset S in an appropriate data structure. The key feature of Grover's algorithm is that it achieves its sublinear complexity for any type of query. In a sense, the superposition in the quantum register can be thought of as its data structure.

3. Grover's algorithm with multiple solutions

The use of Grover's algorithm in the previous section assumes that there is a single unique solution in S according to the oracle. Somewhat surprisingly, the algorithm can fail if there are multiple solutions because the optimal number of iterations will be different. If the number of solutions, say k, is known in advance, it is possible to determine the appropriate number of iterations to ensure that a solution will be found with high probability in $O((N/k)^{1/2})$ time (Nielsen and Chuang 2000). Thus, if there are many solutions, this generalisation of Grover's algorithm can more efficiently find one of them. Unfortunately the value of k is rarely known *a priori* in practical situations, so it must be computed using a quantum counting algorithm. Unfortunately, the best possible (provably optimal) quantum counting algorithm (Brassard et. al. 2000) has complexity $O([(k+1)(N-k+1)]^{1/2}) = O((kN)^{1/2})$.

An alternative to Grover's algorithm can search efficiently for a single solution without knowledge of k, but many practical applications require the retrieval of all the solutions in S, not just one of them (Boyer et. al. 1996). As we will discuss in this section, the lower bound for retrieving the k solutions is determined by the lower bound on counting them. Counting therefore can be performed as a first step in the retrieval algorithm without undermining the overall complexity. Grover's algorithm can then be applied with known k essentially to sample one of the solutions at random. This can be repeated $O(k\log(k))$ times to ensure with high probability that all k solutions are sampled. This algorithm can be summarised as follows.

Suboptimal quantum multi-object search algorithm

(1) Perform a quantum counting operation using the oracle function f to determine the number k of solutions in dataset S of size N. This step has complexity $O((kN)^{1/2})$

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

(2) While *counter* $<k$ do: (expected number of iterations: $O(k\log(k))$)

(a) Initialise quantum register r to a uniform superposition of N indices corresponding to elements in S. This step has complexity $O(1)$.

(b) Apply Grover's algorithm to sample one of the k solutions according to f in $O((N/k)^{1/2})$ time.

(c) Insert the sampled solution into a solution set (no duplicates) implemented with a hash table and increment *counter*. This step has complexity $O(1)$.

(3) End Repeat.
Complexity of loop is $O((kN)^{1/2}\log(k))$.

The overall complexity is dominated by the $O(k\log(k))$ applications of Grover's algorithm, each of which has $O((N/k)^{1/2})$ complexity.

The complexity to retrieve all the solutions is clearly $\Omega(N)$ when $k=N$. Classical exhaustive search therefore has optimal worst-case complexity. However, the above quantum algorithm has complexity $O(N\log(N))$ in the worst case and is therefore suboptimal. The problematic logarithmic factor results from the repeated sampling of already-known solutions. This can be avoided if each sampled solution is marked as being a dummy and swapped with the last non-dummy key. In other words, the indexed array containing N non-dummy keys followed by $2^n - N$ dummies will be modified after the first seach so that only the first $N-1$ keys are now non-dummies. Thus each search causes the size of the dataset to decrease by one[†].

The new algorithm can be summarised as follows.

Optimal quantum multi-object search (QMOS) algorithm

(1) Perform a quantum counting operation using the oracle function f to determine the number k of solutions in dataset S of size N. This step has complexity $O((kN)^{1/2})$.

(2) While $k>0$:

(a) Initialise quantum register r to a uniform superposition of N indices corresponding to elements in S. This step has complexity $O(1)$.

(b) Apply Grover's algorithm to sample one of the k solutions according to f in $O((N/k)^{1/2})$ time and add it to the output dataset in $O(1)$ time.

(c) Mark the sampled solution as a non-solution (and swap with the last non-dummy element). This step has complexity $O(1)$.

(d) $k \longleftarrow k-1$.

(3) End While.
Complexity of loop is $O((kN)^{1/2})$.

(4) Use the solution set to unmark all the elements of S in the oracle so that the complexity of future queries is not compromised by an $O(N)$ initialisation step. This step has complexity $O(k)$.

[†] However, the actual searched set does not actually decrease in size until the number of non-dummies reaches a smaller power of two. At that point the oracle can be applied using one fewer qubits. Like classical dynamic arrays, the amortised complexity is the same as if the searchable dataset decreased continuously rather than by powers of two.

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

The overall complexity is determined by the claimed complexity of $O((kN)^{1/2})$ for the loop in Step 2*b*. This iteration can be expressed as the sum

$$\sum_{i=1}^{k}(N/i)^{1/2} \;=\; N^{1/2}\sum_{i=1}^{k} i^{-1/2}\ . \tag{1}$$

Using the well-known result

$$\sum_{i=1}^{n} i^{c} \;\in\; O(n^{c+1}) \;\text{ for real } c \text{ greater than } -1, \tag{2}$$

we obtain the complexity $N^{1/2}O(k^{1/2}) = O((kN)^{1/2})$ as claimed. Furthermore, this overall complexity is optimal by virtue of the optimality of the same complexity for quantum counting[†]. In other words, retrieving the k solutions yields the value of k and therefore cannot be accomplished with complexity better than that of the optimal counting algorithm.

The general quantum search algorithm, QMOS, is potentially applicable to the general-purpose database problem in which it is desirable to support arbitrary queries with sublinear complexity. Traditional databases can only provide sublinear complexity for predefined classes of queries and thus cannot efficiently support many types of data mining and analysis applications. In this case quantum searching provides a complexity improvement over classical search only by virtue of its generality. There are also important special classes of queries, such as multidimensional range searching, for which quantum search is more efficient than the best possible classical search algorithm. These and other applications are discussed in the following sections.

4. Applications of QMOS

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 some variant of binary search. Unfortunately, multidimensional generalisations that are constrained to using minimal $O(N)$ space, such as the multidimensional binary search tree (also known as the k-d tree), provide only rootic query time that depends on the number of dimensions (Preparata and Shamos 1985). Specifically, orthogonal range queries in d dimensions require $O(N^{1-1/d})$ time ($O(N^{2/3})$ in three dimensions) in the worst case using a multidimensional Binary Search Tree (BST). Obviously, this represents a lower bound for more general types of queries involving non-orthogonal objects.

In most practical applications involving the retrieval of geometrically proximate objects (sometimes referred to as 'near neighbours') the value of k is $O(1)$. In such cases the complexity of QMOS is superior to that of a multidimensional BST for orthogonal range queries in all dimensions greater than two. The advantage of the QMOS algorithm in 4D

[†] However, the use of quantum counting as a subroutine of the QMOS algorithm could make it less practical to implement and apply than alternative algorithms that are specially derived from the ground up using QM to simultaneously search and count. We present our QMOS algorithm as an example of how optimal complexity can also be achieved using only a limited number of QM building blocks.

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

Search Type	Preprocessing Time	Query Time	Space Resources
CC for general search queries	$O(N)$	$O(N)$	$O(N)$
CC for box queries (minimal storage)	$O(N \log N)$	$O(N^{1-1/d} + k)$	$O(N)$
CC for box queries (minimal query time)	$O(N \log^{d-1} N)$	$O(\log^{d}(N) + k)$	$O(N \log^{d-1} N)$
QMOS for general search queries	$O(N)$	$O((kN)^{1/2})$	$O(N)$

Table 1. *Complexity comparison of classical (Preparata and Shamos 1985) and quantum algorithms for satisfying multidimensional spatial search and intersection queries with solution size k (box queries refer to the identification of points or orthogonal boxes that are in or intersect a given orthogonal box; general queries refer to objects other than boxes for which the intersection between a pair of objects can be identified in $O(1)$ time).*

simulations, in which time represents an additional dimension, is even more substantial as the CC complexity for general point location has a lower bound of $\Omega(N^{3/4})$, and no CC algorithm has been found that achieves this lower bound. In fact, it may be the case that $O(N^{1-1/d})$ cannot be achieved in minimal $O(N)$ space except in the special case of orthogonal query objects.

Recent analysis (Andersson and Swanson 1997) suggests that the best CC spatial query algorithms in 2D can only achieve polylogarithmic query time, that is, $O(\log^{c}(N))$ for some constant $c > 1$, at the expense of $\Omega(N \log N)$ space. It is likely that CC data structures for most types of spatial queries will have much higher space complexities (with logarithmic factors that are exponential in d), but even this lower bound complexity suggests that space will become an obstacle for large values of N. Because the CC algorithms that achieve query-time complexity superior to that of QMOS do so at the expense of nonlinear space, the $O((kN)^{1/2})$ query complexity and $O(N)$ space complexity of QMOS are very attractive. Table 1 summarises these results and offers a comparison of the resulting complexities when using different multidimensional spatial search methods.

The above complexities for QMOS hold for both point retrieval and for finding all objects that intersect a given query object as long as the identification of whether two objects intersect can be done in $O(1)$ time. If this is not the case, the complexity for finding the k intersections must be multiplied by the complexity for determining whether a given pair of objects intersect. Both aspects of the problem are considered in the following two subsections.

4.1. *QMOS for object–object intersection identification*

When simulating the motion of objects in a multi-object simulation or VR system, it is necessary to identify the interaction of any pair of objects so that their respective states can be updated, for example, to reflect the kinematic changes resulting from a collision. Collision detection introduces two computational challenges. The first involves the determination of whether two given objects are interacting as indicated by an intersection of their surfaces. This can be computationally expensive for objects whose surfaces are non-convex. The second challenge is to identify which pairs have interacted

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

out of a set of N moving objects. This is often the most computationally expensive part of a simulation or VR system because $O(N^2)$ time is required to check every pair, and, moreover, the most commonly used algorithms for reducing this complexity tend to be data-dependent with scaling between $O(N\log(N))$ and $O(N^2)$. Grid-based algorithms (possibly using hash indexing) can make time/space tradeoffs in which storage is proportional to the volume of the search space, or the query time is proportional to the volume of the objects. None of these approaches appear to be practical for very large problems.

The most basic capability necessary for collision detection is to determine whether two given objects intersect, for example, after their states have been motion-updated during a timestep. If the objects are convex, intersections can be performed very efficiently. However, if they are very complex dynamic objects, for example, streamers blowing in the wind, determining the intersection can be computationally expensive.

A standard mechanism for reducing the computational overhead of intersection detection is to apply coarse gating criteria that can efficiently identify pairs that do *not* intersect. One such mechanism is the use of bounding volumes such as spheres, coordinate-aligned boxes and convex hulls. If the bounding volumes for two given objects do not intersect, then the objects cannot possibly intersect, so there is no need to perform more complex calculations involving the actual objects themselves.

One approach for gating is to compute the 2D convex hulls of each 3D object obtained by projecting (implicitly) its surface points onto each coordinate plane. With these hulls it is possible to determine that two given objects do not intersect simply by showing that their 2D convex hulls do not interesect in one of the coordinate planes. No projection operation needs to be explicitly performed because the projection of a point onto one of a coordinate plane is found directly from two of the point's three coordinates, so the computation of the convex hulls represents the main computational expense.

There are many efficient classical algorithms for computing 2D convex hulls, but the most efficient of these algorithms requires $\Omega(N\log(h))$ time for N objects with h points forming the convex hull (Preparata and Shamos 1985). The relevant question is whether a better QC algorithm exists. One approach for finding such an algorithm is to examine each of the known CC algorithms to determine whether Grover's Algorithm can be productively applied. It turns out that one classical algorithm, the Jarvis March (Preparata and Shamos 1985), can exploit Grover's Algorithm.

The Jarvis March begins by identifying one point on the convex hull. This can be achieved by finding the point with the minimum x-coordinate value. The next point in clockwise order on the convex hull can be found by computing the angles between the line $y = 0$ through the first point and the lines determined by the first point and every other point in the dataset. The line having the smallest angle (measured clockwise) goes through the next point on the convex hull. The same procedure is repeated using the line through the second point. Thus each point on the hull is found successively in $O(N)$ time for an overall complexity of $O(Nh)$.

The attractive feature of the Jarvis March algorithm, from a QC perspective, is that each successive point can be determined after the application of a simple calculation for each of the points in the dataset. Specifically, the angles for the points in each step can be

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

Method	Computation Time	Space Resources
Classical (optimal)	$O(N\log(h))$	$O(N)$
Quantum (Jarvis w/Grover)	$O(N^{1/2}h)$	$O(N)$
Quantum (Akl-Toussaint w/QMOS)	$O((Nh)^{1/2})$	$O(N)$

Table 2. *Complexity comparison of the best classical 2D convex hull algorithm with that of two quantum alternatives. (N is the total number of points, and h is the number of points comprising the hull.)*

computed, and the minimum point retrieved, in $O(N^{1/2})$ time using Grover's Algorithm. This reduces the complexity of the overall convex hull determination to $O(N^{1/2}h)$, that is, sublinear time, if h is a constant, which is often the case in practice.

The above complexity can be further improved in practice through the application of Akl–Toussaint heuristics with QMOS. Specifically, a set of $O(1)$ extreme points (for example, ones with minimum or maximum x and/or y coordinates) can be found using Grover's algorithm in $O(N^{1/2})$ time. Under assumptions that hold in a variety of practical contexts, the number of points from the dataset that do not fall within the convex hull of the extreme points is expected to be $O(h)$. QMOS can be used to find this $O(h)$ superset of points (followed by an $O(h\log(h))$ step to extract the actual hull) in $O((hN)^{1/2})$ time, which is better than the above Jarvis-based algorithm by a factor of $h^{1/2}$.

Table 2 shows that the quantum 2D convex hull algorithm is considerably more efficient than the best classical algorithm as long as h is not large compared to N.

4.2. *QMOS for batch intersection identification*

In the previous section we were concerned with the efficient determination of whether two objects, comprised of $O(N)$ surface points, intersect. In this section we are concerned with the identification of all pairs of intersecting objects in a dataset of N objects. This type of operation is required for the identification of all interactions or collisions in multi-object simulations.

The identification of all intersections can be accomplished in an online setting by applying the QMOS algorithm sequentially for each of the N objects with an oracle that identifies all objects that intersect the object. The resulting complexity is:

$$N * O\left((k_{\text{avg}}N)^{1/2}\right) = O\left(N^{1.5}k_{\text{avg}}\right) \tag{3}$$

where k_{avg} is the average number of intersections per object.

A more direct batch algorithm for finding the m total intersections from a set of N objects is possible by further exploiting the power of QC by doubling the size of the quantum register and constructing the Cartesian product of the N objects. In QC this can be done in $O(1)$ time. QMOS can then be applied to the N^2 pairs to extract the m intersecting pairs in $O((N^2m)^{1/2}) = O(Nm^{1/2})$ time. For $m = N * k_{\text{avg}}$, this complexity is the same as the earlier result obtained by performing N sequential intersection queries; however, the overhead of iteration in the latter approach is completely avoided, that is,

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press

it may be speculated that the batch identification of m intersections should prove to be significantly faster in practice.

5. Summary

In this paper we have discussed efficient applications of Quantum Multi-Object Search (QMOS) to computational geometry, and specifically to problems of relevance to virtual reality and simulation systems. In particular, we have described a class of search algorithms for which quantum search asymptotically outperforms the best CC algorithms. We have shown that a quantum algorithm can be applied to construct convex hulls in sublinear time when the number of points on the hull is constant. We have also developed a quantum algorithm for the online detection of intersections in multi-object systems. And, finally, we have derived an algorithm for the batch identification of all intersections of objects with applications to discrete time simulation systems.

References

Andersson, A. and Swanson, K. (1997) On the difficulty of range searching. *Computational Geometry with Applications* **8** (3) 115122.

Boyer, M., Brassard, G., Hoyer, P. and Tapp, A. (1996) Tight bounds on quantum searching. *Proceedings of the Fourth Workshop on Physics and Computation.*

Brassard, G., Hoyer, P., Mosca, M. and Tapp, A. (2000) Quantum amplitude amplification and estimation. e-print quant-ph/0005055.

Nielsen, M. A. and Chuang, I. L. (2000) *Quantum Computation and Quantum Information*, Cambridge University Press.

Preparata, F. P. and Shamos, M. I. (1985) *Computational Geometry*, Springer-Verlag.

https://doi.org/10.1017/S0960129510000411 Published online by Cambridge University Press