# Overview of Parallel Convex Hull Algorithms

Brendan Benshoof

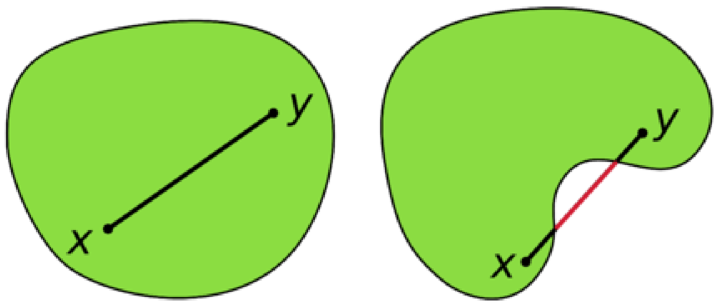
April 29, 2015

## 0.1 Introduction

The Convex Hull problem can be considered one of the most well studied problems in computational geometry. There are numerous centralized algorithms and an achievable lower bound of O(nlog(n)) time (the sorting bound). It is surprising that there are a comparatively smaller number of parallel algorithms that have been researched. Two major approaches (Divide and Conquer vs. Sampling) and three major algorithms (Atallah's algorithm[7], qHull, and High Confidence sampling) dominate literature on the topic.

### 0.1.1 Problem Formulation

A convex hull, is a graph of a subset of a set of points in space such that the convex polytope described by that graph contains all points in the set. In a geometric sense, this means describing a minimal space such that given any two points in a set, the line segment between them falls wholely within the set.



(a) A convex set (b) A non-convex (concave) set
Credit: *User A1 at en.wikipedia*

There are a variety of sequential algorithims to solve the 2d planar convex hull problem. Grahm's scan[6] and QuickHull[1] are worthy of note as solutions seeing common use. QuickHull is of specific interest as an examined line of parallel algorithims are based on it[15][14].

The convex hull problem is involved in many common computations including collision detection and delunay-triangulation/voronoi tessellation.

## 0.2 Overview of Literature

Literature can be divided into two strong groupings. Those that derive from a high-probability of accuracy sampling method and those papers who's algorithm derives from divide and conquer approach.

The papers concerning divide and conquer approaches can be divided further into the papers that implement the parallel convex hull algorithm for EREW PRAM presented by Atallah et al versus the more modern algorithm which derive from the quick hull algorithm presented by Barber et al.

## 0.3 Divide and Conquer methods

Divide and Conquer is one of the basic computer science problem solving techniques. It's basis is in find a method to reduce a problem to simpler sub-problems then merge those sub-problems into a global solution. Breaking the global problem into sub-problems makes it well adapted to application in parallel algorithms, where each sub-problem can be solved in parallel, with most time being consumed by merging the sub-problems.

### 0.3.1 Properties and applications

Most papers discussed in this section derive from a divide and conquer method presented by Atallah et al[7]. This algorithm is only defined for planar convex hull and does not extend readily into higher dimensions.

### 0.3.2 Overview of literature

Here I overview the papers I have categorized as Divide and Conquer approaches to parallel convex hull. They are presented in approximate order of development and publication. I present a summary of the contribution of the paper and a discussion of how it effected later work.

#### Atallah et al

Atallah et al[7] present a parallel variation of traditional divide and conquer algorithms for $O(n)$ processors. They present an $O(\log(n))$ time algorithm given $O(n)$ processors. Rather than splitting the problem space in an iterative binary fashion, they subdivide the problem space into $O(\sqrt{n})$ subdivisions. This allows the sub-problems each to be processed in parallel by $O(\sqrt{n})$ processors. This allows each sub-problem to be viewed independently as a problem with $O(n)$ input and $O(n)$ processors and this method can be applied recursively until trivial problems are established.

This elegant dividing of the problem space hinges on even subdivision of work. In the case of parallel convex hull, points are sorted along any convent axis using $O(\log(n))$ time (which dominates run-time). Then $O(\sqrt{n})$ contiguous portions of $O(\sqrt{n})$ points are recursively solved and merged.

When recursively subdividing the space, while the ratio between bits of input and number of processors is $O(1)$ it is realistic to assume that data points outnumber the processors by more than three to one. Thus, is subdivision reaches three or less points in a group, it can cease further subdivision as sets of points three or less have a trivial convex hull. More realistically, once the subdivision cannot continue due to lack of processors, the convex hull of the $O(1)$ remaining points can be solved by a single processor in constant time using any one of the established sequential methods. If all data has been provided initially to all nodes, the subdivision step can be preformed in $O(\log(n))$ time.

The bulk of computation is contained in the merging step. The recursive division step has created a merge tree, requiring $O(\log(n))$ iterations of the merging algorithm. Once the points have been separated into trivial sub-problems they are merged in parallel in constant time. $O(\sqrt{(n)})$ subsets are merged using

$O(n)$ processors, therefore we can assign one processor to each pair of potential merges.

Atallah et al's algorithm is applied to more specific parallel computing architectures in later papers.

#### Miller et al

"Efficient parallel convex hull algorithms"[12] was published in 1988, two years after Atallah et al's[7] foundational work. Miller et al[12] shows algorithms for preforming the convex hull operation given pre-sorted input on a variety of parallel architectures: a hypercube, pyramid, tree, mesh-of-trees, mesh with reconfigurable bus, EREW PRAM, and a modified AKS network. This paper allows the algorithim described by Atallah et al[7] to be applied to a variety of practical parallel computing architechtures while retaining the $O(log(n)$ time and $O(nlog(n)$ work properties. However becuase the algorithims are predicated on being pre-sorted (by order on any axis). This means that in cases of arbitrary input order, the algorithims are further bounded by the cost of sorting on that platform.

#### Ferreira et al

Ferreira et al[8] further extends Atallah et al onto a "Coarse Grained" computer model. This shows the algorithm can be preformed while preserving it's cost on a variety of limited communication systems. This genralization shows that further proof that runtime and work established by Atallah et al can be maintained on a given "coarse grained" parallel computer system.

#### Amato et al

In 1994 Amato et al published "Parallel algorithms for higher - dimensional convex hulls"[10] which provides a EREW PRAM model for solving high dimension convex hulls using probabilistic linear programming methods in $O(log(n))$ time with $O(nlog(n) + n^{\frac{d}{2}})$ work. Work in this paper is built on extensions of the parallel divide and conquer method into higher dimensions. (It is worth noting that this paper shares an author with Atallah et al in Michael Goodrich and it is clearly generalizing directly on his previous work.)

### 0.3.3 QuickHull derived methods

In practice, on GPU implementations, modern algorithms for convex hull calculation are based on the quick hull algorithm presented by Barber et al.

#### Barber et al

Barber et al[1] does not present a parallel algorithm. It is included in this survey as it presents the QuickHull algorithm which other papers present a parallel variation of for GPU computing.

QuickHull's approach is to grow a area/volume which aggressively discards points and divides the remaining points into smaller sub-problems along the faces of the area's growth. Barber et al presents QuickHull as a solution to 2d convex hull, however unlike many other sequential convex hull algorithms it is

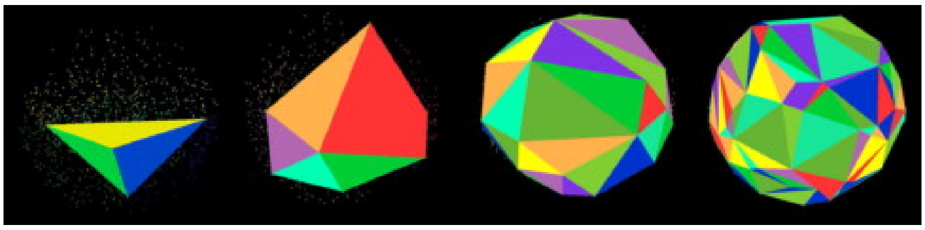


Figure 2: Visualization of QuickHull in 3D. This image is sourced from CUDAHull[15]

extended trivially into higher dimensions. Like many other sequential convex hull algorithms, it requires $O(n\log(n))$ time (the sorting lower bound).

Quickhull can be formally described as follows:

Given $O(n)$ $d$-dimensional points in set $P$. Find $d+1$ maximal points and use them to form a d-dimensional simplex. Discard all points in $P$ that are contained in this simplex. Each face of the simplex partitions the remainder of $P$, recurse onto each of these partitions, utilizing the bounding face to bootstrap finding a new simplex.

It is important to note, that this algorithm only produces the set of points in the convex hull. In 2D, how these points are connected is trivial (sorted by angle around any point contained in the hull). In higher dimensions, if a convex polytope is the desired output rather than a set of points, the partially correct linking information produced by the algorithm must be corrected to generate a correct convex polytope. This cleanup operation is at worst $O(h)$ cost, where $h$ is the number of points on the convex hull.

### Srungarapu et al

Srungarapu et al present “Fast two dimensional convex hull on the GPU”[14] in 2011, which describes an implementation of QuickHull in 2D for the CUDA platform. It presents a $O(n\frac{log(n)}{p})$ time algorithm with $O(p)$ processors. The point culling steps in QuickHull require minimal branching and is easily mapped to GPU parallel behavior. As the recursive are steps also similar, many recursions can be ran in parallel (each finding if a given point falls inside a given triangle).

### Stein et al

Stein et al presented CudaHull[15] in 2012 which provides implementation of 3D QuickHull for CUDA. They provide a $O(\frac{n}{p}log(n))$ time algorithm, with a $O(h)$ sequential cleanup setup (where $h$ is the number of points on the convex hull) to provide a proper convex polytope description of the Hull (nontrivial in 3+ dimensions). Their application is analogous to Srungarapu et al's[14] with the added complexity of utilizing 3D tetrahedrons rather than 2D triangles. The inital cost is identical because the cost of testing if a point is contained in a simplex (of any dimension) is $O(d)$ which is assumed to be constant for these applications. Because the final connections of points on the hull is not fully determined by QuickHull in higher dimensions,

## 0.4 Sampling Methods

Sampling methods are generally not searching for solutions in a geometrically intuitive way. They pose the convex hull problem as a linear programming problem and utilize conventional parallel search methods to generate a high confidence set of points as members of the convex hull. This presents the problem, that while finding the set of points in a convex hull is a more computationally simple task than finding the entire hull, most applications require a full description of bother the verticies and edges of the convex hull polytope.

### 0.4.1 Overview of literature

#### Dehne et al

Dehne et al “A Randomized Parallel 3D Convex Hull Algorithm for Coarse Grained Multicomputers”[4] supplies a high-probability of accuracy algorithm for 3d convex hull on EREW PRAM of O(log(n)) time by converting the convex hull problem into a linear program. The linear program solution only provides the set of points on the hull and a linear time method of building the polytope is utilized.

#### Ghouse et al

in 1997 Ghouse et al “Fast randomized parallel methods for planar convex hull construction” [9] proposes the O(1) and O(nlog(n)) work algorithm for CRCW PRAM. The convex hull problem is presented as a linear programming problem and solved using probabilistic parallel linear programming techniques. This work applies a previous published technique of “in-place”[5] algorithms operating on convex hulls.

It is worth noting that this work shares coauthor with Atallah et al in Michael Goodrich.

## 0.5 Conclusions and Areas for New Research

The field of works concerning Parallel Convex Hull algorithms is small but mature. It will be difficult for new works to be published in this area simply because existing works have presented practical and provably optimal solutions to the Convex Hull problem. Applications of these algorithms to solve practical problems and onto new computing platforms is the most clear space for future work. Here I present some possible application areas and summaries of them.

### 0.5.1 High performance convex hulls in higher-dimensional

The Convex Hull problem provides solutions to many other geometric problems via embedding of those problems into a convex hull of higher dimension. There are not currently practical CUDA GPU tools for solving these higher dimensional convex hull problems.

### 0.5.2 Parallel convex hulls to solve geometric problems in non-euclidean spaces

Convex Hull algorithms in higher dimension provide solutions to delunay triangulation and other geometric problems in non-euclidean metric spaces. Such problems are seeing application in machine learning and network optimization. There are no current techniques for solving these problems on commodity parallel hardware.

### 0.5.3 Parallelization of other Convex Hull Algorithms

This servery only covers 3 sequential algorithms adapted for use in parallel computation. A wide field of sequential algorithms have not had published adaptations.

## Bibliography

[1] C. Bradford Barber and David P. Dobkin. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. December*, 22, 1996 1996.

[2] A. Chow. Parallel algorithms for geometric problems.

[3] A. M. Day. Parallel implementation of 3d convex - hull. *algorithm Computer - Aided Design*, 1991.

[4] Frank Dehne, Xiaotie Deng, Patrick Dymond, Andreas Fabri, and Ashfaq A. Khokhar. A randomized parallel 3d convex hull algorithm for coarse grained multicomputers. In *Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures*, SPAA '95, pages 27–33, New York, NY, USA, 1995. ACM.

[5] Mujtaba R Ghouse and Michael T Goodrich. In-place techniques for parallel convex hull algorithms (preliminary version). In *Proceedings of the third annual ACM symposium on Parallel algorithms and architectures*, pages 192–203. ACM, 1991.

[6] Ronald L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

[7] Atallah Mikhail J. and Michael T. Goodrich. Efficient parallel solutions to some geometric problems. *Journal of Parallel and Distributed Computing*, 3, 1986.

[8] Mohamadou. Scalable 2d convex hull and triangulation algorithms for coarse grained multicomputers. *Journal of Parallel and Distributed Computing*, 56, 1999.

[9] R. Mujtaba and Michael T. Goodrich. Fast randomized parallel methods for planar convex hull construction. *Computational Geometry*, 7, 1997.

[10] M. Nancy, Michael T. Goodrich, and Edgar A. Ramos. Parallel algorithms for higher - dimensional convex hulls. *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on. IEEE,*, 1994.

[11] John H Reif and Sandeep Sen. Optimal parallel randomized algorithms for three-dimensional convex hulls and related problems. *SIAM Journal on Computing*, 21(3):466–485, 1992.

[12] Quentin F. Russ and Stout. Efficient parallel convex hull algorithms. *Computers, IEEE Transactions on*, 37, 1988.

[13] Raimund Seidel. *A convex hull algorithm optimal for point sets in even dimensions*. PhD thesis, University of British Columbia, 1981.

[14] Srikanth. Fast two dimensional convex hull on the gpu. *Advanced Information Networking and Applications ( WAINA)*, 2011.

[15] Eran Geva Ayal Stein and Jihad El-Sana. Cudahull: Fast parallel 3d convex hull on the gpu. *Computers and Graphics*, 36:265–271, 2012.