The Quickhull Algorithm for Convex Hulls

C. Bradford Barber* David P. Dobkin† Hannu Huhdanpaa‡

January 9, 1995

Submitted to the ACM Transactions on Mathematical Software

Abstract

The convex hull of a set of points is the smallest convex set that contains the points. This paper presents a practical convex hull algorithm that combines the two-dimensional Quickhull Algorithm with the general dimension Beneath-Beyond Algorithm. It is similar to the randomized, incremental algorithms for convex hull and Delaunay triangulation. We provide empirical evidence that the algorithm runs faster when the input contains non-extreme points, and that it uses less memory.

Computational geometry algorithms have traditionally assumed that input sets are well behaved. When an algorithm is implemented with floating point arithmetic, this assumption can lead to serious errors. We briefly describe a solution to this problem when computing the convex hull in two, three, or four dimensions. The output is a set of "thick" facets that contain all possible exact convex hulls of the input. A variation is effective in five or more dimensions.

1. Introduction

The convex hull of a set of points is the smallest convex set that contains the points. The convex hull is a fundamental construction for mathematics and computational geometry. For example, Boardman uses the convex hull in his analysis of spectrometry data [6] and Weeks uses the convex hull to determine the canonical triangulation of cusped hyperbolic 3-manifolds [44]. Other problems can be reduced to the convex hull – for example, halfspace intersection, Delaunay

*116 Fayerweather Street, Cambridge, MA 02138, `bradb@geom.umn.edu`. This research was supported in part by the National Science Foundation under Grants NSF-CCR-91-15793 750-7504, NSF/DMS-8920161, and CCR90-02352.

†Department of Computer Science, Princeton University, Princeton, NJ 08544, `dpd@cs.princeton.edu`. This research was supported in part by the National Science Foundation under Grant CCR-93-01254

‡Configured Energy Systems, Inc. 3140 Harbor Lane North, Suite 202 Plymouth, MN 55447, `hth@ces.com`

triangulation, Voronoi diagrams, and power diagrams. In his review article, Aurenhammer describes applications of these structures in mesh generation, file searching, cluster analysis, collision detection, crystallography, metallurgy, urban planning, cartography, image processing, numerical integration, statistics, sphere packing, and point location [2].

We represent a convex hull with a set of facets and a set of adjacency lists giving the neighbors and vertices for each facet. The boundary elements of a facet are called *ridges*. Each ridge signifies the adjacency of two facets. In $\mathbf{R}^3$ and general position, facets are triangles and ridges are edges.

A Delaunay triangulation in $\mathbf{R}^d$ can be computed from a convex hull in $\mathbf{R}^{d+1}$. To determine the Delaunay triangulation of a set of points: lift the points to a paraboloid and compute their convex hull. The set of ridges of the lower convex hull is the Delaunay triangulation of the original points [9].

The intersection of halfspaces about the origin is equivalent to the convex hull of the points in dual space [39]. To determine the intersection of halfspaces: locate an interior point by linear programming [43], translate the interior point to the origin, transform halfspaces into points by dividing offsets into coefficients, and compute the convex hull. Via the duality transform, each facet of the convex hull defines a point of intersection of the halfspaces.

Recent work on convex hulls and Delaunay triangulations has focused on variations of a randomized, incremental algorithm that has optimal expected performance [12] [15] [21] [28] [30] [37]. Points are processed one at a time in a random order. In this paper, we propose and analyze a strategy for processing points in a more efficient order. The result is a faster algorithm for distributions with interior points.

An incremental algorithm for the convex hull repeatedly adds a point to the convex hull of the previously processed points. Of particular interest is the Beneath-Beyond Algorithm [27] [31] [39]. A new point is processed in three steps. First, locate the visible facets for the point The boundary of the visible facets is the set of *horizon ridges* for the point. A facet is *visible* if the point is above the facet. Second, construct a cone of new facets from the point to its horizon ridges. Third, delete the visible facets, thus forming the convex hull of the new point and the previously processed points.

The original randomized incremental algorithm upon which we build was proposed by Clarkson and Shor [16]. They work in the dual space of halfspace intersections. Their algorithm adds a halfspace by intersecting it with the polytope of the previous intersections. They randomly select a halfspace to add to the polytope. For each unprocessed halfspace, they maintain the list of polytope edges that intersect the halfspace. The *conflict graph* is the set of all such lists. When a halfspace is processed, the conflict graph identifies the modified edges. To reduce memory requirements to $O(n)$, they propose storing a single modified edge for each unprocessed halfspace. A simple search of adjacent edges identifies the remaining modified edges.

Our Quickhull Algorithm is a variation of Clarkson and Shor's algorithm. As with most of the variations, we work in the space of points and convex hulls instead of the dual space of halfspaces and polytopes. This turns the conflict graph into an *outside set* for each facet. A point is in a facet's outside set only if it is above the facet. Like Clarkson and Shor's algorithm, an unprocessed point is in exactly one outside set. Our variation is to process the furthest point of an outside set instead of a random point. In $\mathbf{R}^2$, this is the well-known Quickhull Algorithm [10] [20] [22] [26].

Other variations of the Clarkson and Shor algorithm do not maintain conflict graphs or outside sets. Instead, they retain old facets of the convex hull with links to the new facets that replaced them. This hierarchy begins with an initial simplex formed from $d+1$ of the input points. The algorithms find a visible facet for a point by searching for a chain of visible facets through the hierarchy. The chain either ends with a facet of the current hull, or all facets of the current hull are above the point and the point is discarded. These algorithms have been implemented. In practice, their running times are competitive with other algorithms [24].

We can compare Quickhull with the randomized incremental algorithms by changing the selection step of Quickhull. If Quickhull selects a random point instead of a furthest point, it is a randomized incremental algorithm. In our empirical tests, Quickhull runs faster than the randomized algorithms because it processes fewer interior points. Also, Quickhull reuses the memory occupied by old facets.

Our analysis is strictly empirical. We define two balance conditions that would guarantee behavior identical to the randomized algorithms when all points are extreme. If the balance conditions hold, our algorithm runs on an input of size n with r processed points in time $O(n \log r)$ for $d \leq 3$ and $O(n f_r / r)$ for $d \geq 4$ (f_r is the maximum number of facets for r vertices). We conjecture that this is always the case when the input precision is restricted to $O(\log n)$ bits.

Empirically, the number of points processed by Quickhull is proportional to the number of vertices in the output. Output-sensitivity is important for convex hull algorithms because the output size can be much smaller than the worst case size. In $\mathbf{R}^2$, Kirkpatrick and Seidel found an optimal output-sensitive algorithm for convex hull that runs in $O(n \log h)$ time, where h is the output size [32]. Clarkson & Shor give a 3-d convex hull algorithm with optimal output-sensitive expected time [16]; it was derandomized by Chazelle and Matoušek [12]. In higher dimensions, the best output-sensitive algorithm is Seidel's shelling algorithm at $O(n^2 + h \log n)$ when $h = \Omega(n)$ [40], and gift-wrapping at $O(nh)$ otherwise [11]. The Double-Description Method is the dual of the Beneath-Beyond Algorithm [36]. It is the earliest incremental method for computing the convex hull. It is an excellent choice in high dimensions when the number of facets is much smaller than the maximum number of facets for r vertices (f_r) [3] [25].

2. The Quickhull Algorithm

We assume that the input points are in general position (i.e., no set of $d+1$ points define a $(d-1)$-flat), so that their convex hull is a simplicial complex [39]. We represent a d-dimensional convex hull by its vertices and $(d-1)$-dimensional faces (the *facets*). A point is *extreme* if it is a vertex of the convex hull. Each facet includes a set of vertices, a set of neighboring facets, and a hyperplane equation. The $(d-2)$-dimensional faces are the *ridges* of the convex hull. Each ridge is the intersection of the vertices of two neighboring facets.

Quickhull use two geometric operations: *oriented hyperplane* and *signed distance to hyperplane*. We represent a hyperplane by its outward-pointing unit normal and its offset from the origin. The signed distance of a point to a hyperplane is the inner product of the point and normal plus the offset. The hyperplane defines a halfspace of points that have negative distances from the hyperplane. If the distance is positive, the point is *above* the hyperplane.

For processing a point we use a simplification of Grünbaum's Beneath-Beyond Theorem [Th. 5.2.1] [27]. The randomized incremental algorithms are based on this theorem.

Theorem 1. (Simplified beneath-beyond) *Let H be a convex hull in $\mathbf{R}^d$ and let p be a point in $\mathbf{R}^d - H$. Then F is a facet of* $\operatorname{conv}(p \cup H)$ *if and only if*

1. *F is a facet of H and p is below F, or*
2. *F is not a facet of H and its vertices are p and the vertices of a ridge of H with one incident facet below p and the other incident facet above p.*

The central problem of Beneath-Beyond is determining the visible facets efficiently. Since a facet is linked to its neighbors, locating one visible facet allows the rest to be located quickly. Most of the randomized algorithms use the previously constructed facets to locate the first visible facet for a point. Our solution is simpler. After initialization, Quickhull assigns each unprocessed point to an outside set. By definition, the corresponding facet is visible from the point.

When Quickhull creates a cone of new facets, it builds new outside sets from the outside sets of the visible facets. This process, called *partitioning*, locates a visible new facet for each point. If a point is above multiple new facets, one of the new facets is selected. If it is below all of the new facets, the point is inside the convex hull and can be discarded. Partitioning also records the furthest point of each outside set.

```
create a simplex of d + 1 points
for each facet F
    for each unassigned point p
        if p is above F
            assign p to F's outside set
for each facet F with a non-empty outside set
    select the furthest point p of F's outside set
    initialize the visible set V to F
    for all unvisited neighbors N of facets in V
        if p is above N
            add N to V
    the set of horizon ridges H is the boundary of V
    for each ridge R in H
        create a new facet from R and p
        link the new facet to its neighbors
    for each new facet F'
        for each unassigned point q in an outside set of a facet in V
            if q is above F'
                assign q to F''s outside set
    delete the facets in V
```

Figure 1: Quickhull Algorithm for the convex hull in $\mathbf{R}^d$.

Quickhull selects a non-degenerate set of points for the initial simplex. If possible, it selects points with either a maximum or minimum coordinate. An outline of the algorithm is given in Figure 1.

To prove the correctness of Quickhull, we first prove that a point can be partitioned into any legal outside set. If so, extreme points of the input will always be vertices of the output.

LEMMA 1. *If an extreme point of the input is above two or more facets at a partition step in Quickhull, it will be added to the hull irrespective of which outside set it is assigned to.*

Proof: Assume the contrary and consider an extreme point p that is not assigned to an outside set and hence never added to the convex hull. Since p is an extreme point, it must have been outside at least one facet of the initial simplex. By assumption, there is a point q with p in its visible outside sets but not in its new outside sets. So p is above a visible facet and below all new facets for q. This implies that p is inside the convex hull and hence, not an extreme point. ∎

THEOREM 2. *The Quickhull Algorithm produces the convex hull of a set of points in* $\mathbf{R}^d$.

Proof: Quickhull starts with the convex hull of $d+1$ points. Quickhull partitions points into outside sets and picks furthest points for processing. By Lemma 1, partitioning can not prevent an extreme point from being processed. Quickhull processes a point according to Theorem 1. It retains the facets in Part 1 of the theorem, and constructs new facets as specified in Part 2. It produces the convex hull of the processed points. Processing the last point produces the convex hull of the input set. ∎

Most of the randomized incremental algorithms perform the same steps as Quickhull but in a different order. They use Beneath-Beyond on a random permutation of the points. They locate a visible facet by a depth-first search of the previous convex hulls. Consider the sequence of facets tested for a point. Quickhull may test the same sequence during successive partitions of the point into outside sets.

Edelsbrunner and Shah [21], Joe [30], and Boissonnat and Devillers-Teillaud [7] use a similar method for Delaunay triangulations. They express their algorithm in terms of triangulations and the in-sphere test. By the correspondence between Delaunay triangulation and convex hull, each triangle is a facet of the convex hull and the in-sphere test determines the visible facets for the lifted point [9].

3. Comparison of Quickhull with the randomized algorithms

If Quickhull and the randomized algorithms perform essentially the same steps, why do we prefer Quickhull? Quickhull uses less space than most of the randomized incremental algorithms and runs faster for distributions with non-extreme points. The main costs for these algorithms are creating facets and computing distances. To isolate the effect of randomization on time efficiency, we can change the processing order of Quickhull. Instead of selecting the furthest point of an outside set, we can select a random point from all outside sets.

This section reports the results of experiments comparing two versions of Quickhull, with and without randomization. Each test is an average and range for ten trials. Quickhull partitions points to the first visible facet. The randomized version starts with a random initial simplex. In a previous report, we compared actual time and space for Quickhull and a randomized incremental program [4]. Those figures support our results here.

First consider uniform random distributions projected to a sphere. Each coordinate is selected randomly from the interval [-0.5, 0.5]. The sphere is radius 0.5 centered at the origin. All points are extreme. In $\mathbf{R}^3$ to $\mathbf{R}^7$, there is little difference in operation counts with and without randomization.

For example, consider the convex hull of 300 uniform random points in $\mathbf{R}^5$ projected to a sphere. Quickhull created 28,200 (27,600-28,700) facets and computed 51,600 (50,200-53,100) distance tests. For the same distributions, randomized Quickhull created 29,000 (28,600-29,900) facets and computed 53,400 (51,900-55,600) distance tests. Delaunay triangulations show similar results because all points are extreme.

Quickhull uses less memory than most of the randomized algorithms. For the $\mathbf{R}^5$ example, Quickhull uses memory proportional to the 7124 output facets, while an algorithm that retains old facets uses memory proportional to the 29,000 (28,600-29,900) created facets.

Figures 2a-2c show the results for uniform random distributions of points in the unit cube. In ten trials, the convex hull of 10,000 random points had 245 (226-288) facets and 125 (115-134) vertices. Quickhull created 730 (662-831) facets while the randomized version created 2210 (1960-2440). The randomized version performed twice as many distance tests. These differences are due to the number of processed points. Quickhull processed 153 (141-171) points while the randomized version processed 387 (345-421) points.

Consider the case of finding the convex hull of a set of uniform random points in $\mathbf{R}^3$ projected to a sphere of radius 0.5 centered at the origin. To these points add the vertices of a cube, also centered at the origin. As the edge length of the cube increases, the number of extreme, cospherical points decrease. We consider the situation when the cube has edge length 0.96. In this case, the sphere extends slightly beyond each facet of the cube.

The convex hull of 10,000 cospherical points and a 0.96 cube contains 76 (68-84) facets and 40 (36-44) vertices. Quickhull created 315 (155-807) facets while the randomized version created 26,000 (11,900-37,600) facets. The difference is due to the number of processed points. Quickhull processed 74 (42-174) points while the randomized version processed 4360 (2080-6253) points. Because Quickhull processes the furthest point of each outside set, it processes the cube's vertices before processing most of the cospherical points.

The randomized incremental algorithms can perform significantly worse than expected. There are two bad cases. Each point could create many new facets, or each search for a visible facet could visit many old facets. The first case occurs when the points of a spiral happen to be added in order. This can not occur for Quickhull because the end of the spiral would be added before most of the intermediate points.

For Quickhull, the second case corresponds to assigning all points to one outside set for most partitions. We conjecture that this can not occur if we restrict the input precision to $O(\log n)$ bits [35]. In $\mathbf{R}^d$, a point creates at least d new facets. Since Quickhull processes the furthest point of each outside set, a sequence of partitions to a single facet would produce a facet that is too small for the input precision. A similar restriction holds for Quicksort using a mean pivot. For Quicksort to be unbalanced, most of the points must consistently fall in one of two partitions. With a mean pivot, this would require $O(n)$ bits of precision.

We conjecture that Quickhull iterations are average (i.e., each iteration creates an average number of new facets whose outside sets are average size). This defines two balance conditions. Let d be the dimension, n be the number of input points, r be the number of processed points, and f_r be the maximum number of facets of r vertices ($f_r = O(r^{\lfloor d/2 \rfloor}/\lfloor d/2 \rfloor!)$ [33]).

DEFINITION . *An execution of Quickhull is* balanced *if*

- *the average number of new facets for the j-th processed point is* df_j/j
- *the average number of partitioned points for the j-th processed point is* $d(n-j)/j$

To test the balance conditions statistically, we can compute the expected number of new facets from the actual number of facets. Similarly we can compute the expected number of partitioned points. For example, each iteration of Quickhull on ten trials of 6000 random cospherical points in $\mathbf{R}^3$ created 3.7 fewer new facets than expected (1.3 standard deviation), and partitioned 5.4 fewer points than expected (50 s.d.). On the same distributions, randomized Quickhull created 0.1 fewer facets than expected (1.3 s.d.), and partitioned 1.4 fewer points than expected (46 s.d.).

THEOREM 3. *Let n be the number of input points in $\mathbf{R}^d$ and r be the number of processed points. If the balance conditions hold, the worst case complexity of Quickhull is $O(n \log r)$ for $d \leq 3$ and $O(nf_r/r)$ for $d \geq 4$.*

Proof:

There are two costs to Quickhull, adding a point to the hull and partitioning. The dominant cost for adding a point is $O(d^3)$ work to create hyperplanes. The dominant cost for partitioning is $O(d)$ work to compute distances. The total cost for adding points is proportional to the total number of new facets created, $O(d^3 \sum_{j=1}^{r} df_j/j)$. This simplifies to $O(f_r)$ (each term is less than df_r/r and the $\lfloor d/2 \rfloor!$ denominator of f_r subsumes d^4).

The cost of partitioning one point in one iteration is proportional to the number of new facets. The total cost is $O(d \sum_{j=1}^{r} d^2(n-j)f_j/j^2)$. Expanding f_j yields $O(d^3 n \sum_{j=1}^{r} j^{\lfloor d/2 \rfloor - 2}/ \lfloor d/2 \rfloor!)$. If $d \leq 3$, the sum is $O(n \log r)$. Otherwise, each term is less than f_r/r^2 and the sum is $O(nf_r/r)$. ∎

If $r = n$ and the balance conditions hold, the cost of Quickhull is $O(n \log n)$ for $d \leq 3$ and $O(f_n)$ otherwise. This is the same as the expected cost of the randomized incremental algorithms [15].

For Quickhull, the furthest point of an outside set is not always the furthest point from a facet, but in many cases, it is the furthest overall and hence an extreme point. If it is not an extreme point, then a later point will delete it. An extreme point that deletes multiple vertices appears to occur infrequently. For these reasons, we make the following conjecture.

CONJECTURE . *Let n be the number of input points in $\mathbf{R}^d$, and v be the number of output vertices. If the precision of the input points is $O(\log n)$, the worst case complexity of Quickhull is $O(n \log v)$ for $d \leq 3$ and $O(n f_v / v)$ for $d \geq 4$.*

4. Coping with imprecision

So far, we have followed the real-RAM model and assumed that the input set is non-degenerate. If we implement Quickhull with floating point arithmetic, round-off and representation errors may introduce degeneracies. We briefly describe heuristic extensions of Quickhull that work well in practise.

Quickhull partitions a point and determines its horizon facets by computing whether the point is above or below a hyperplane. We have assumed that computations return consistent results. The existence of roundoff errors calls this assumption into question. For example in Figure 3, the point may be computed to be above facets F1 and F3 and below F2. The Beneath-Beyond step replaces F1 and F3 with five new facets. Since it does not replace F2, the new facets contain serious geometric and topological errors. Two of the facets are flipped up-side-down and four of the facets meet at a ridge.

With floating point arithmetic, we can not prevent such errors from occurring but we can repair the damage after processing a point. We use brute force: if adjacent facets are non-convex, if more than two facets facets meet at a ridge, or if other topological faults occur, one of the facets is merged into a neighbor. Quickhull merges the facet that minimizes the maximum distance of a vertex to the neighbor.

For example in Figure 3, label the new facets by ridges $a - e$. Quickhull first handles ridges with more than two facets. In this case, facets a, b, d, and e meet at the same ridge. Quickhull repairs the fault by merging the two closest facets, say b and d. The vertices of facet F2 and facet c are contained in the merged facet bd, so Quickhull merges them together. The merged facet $bcd2$ has only two neighbors. It is merged into the closer of facets a and e. Say the later occurs. If facet $bcde2$ and facet a are clearly convex, Quickhull is finished with this pair of facets.

We can not use the counter-clockwise test to test the convexity of non-simplicial facets: The test depends on vertices coinciding with the intersection of hyperplanes. Instead, non-simplicial facets have a point (the *centrum*) that must be clearly below the hyperplanes of neighboring facets. Except for large facets, the centrum is the average of the vertices projected to the hyperplane. Centrums are fixed for large facets.

The result of merging is a "thick" facet defined by a positive and negative offset from the facet's hyperplane. The polytope defined by the negative offsets clearly excludes the vertices; the polytope defined by the positive offsets clearly includes the input points. The space between the polytopes contains the boundaries of all possible, exact convex hulls through the points. We determine the offsets – after constructing the hull – by testing all points near the boundary of the convex hull.

When done, Quickhull reports the maximum facet width and guarantees that neighboring facets are clearly convex. In practice, Quickhull produces a convex hull whose thickest facet is less than six times thicker than merging two non-convex simplicial facets. In $\mathbf{R}^2$ to $\mathbf{R}^4$, a ratio greater than ten probably indicates an error.

For $\mathbf{R}^d$, Quickhull repairs the following faults in this order: more than two facets meeting at a ridge, a facet contained in another facet, a facet with fewer than d neighbors, a facet with flipped orientation, a newly processed point that is coplanar with an horizon facet, concave facets, coplanar facets, and redundant vertices. It removes coplanar facets in independent sets of merges sorted by angle. In $\mathbf{R}^5$ and higher, it removes coplanar facets after constructing the hull. This may cause a poor approximation and the hull may be wider than reported. Both events appear to be unlikely.

Quickhull does not handle all faults from degenerate distributions. For example, faults occur for the furthest-site Delaunay triangulation of 2000 points within 10^{-12} of the unit circle. During construction of the corresponding convex hull, two neighboring facets may span the point set with opposite orientations. A point may be just above both facets yet remain distant from the precise convex hull. Later, the coplanar point may be far above a new facet. If this occurs, Quickhull generates a warning and reports a wide facet.

In $\mathbf{R}^2$, there are several robust convex hull and Delaunay triangulation algorithms [23] [29] [34]. In $\mathbf{R}^3$, Sugihara and Dey et. al. produce a topologically robust convex hull and Delaunay triangulation [18] [42]. Their algorithms are a variation of Beneath-Beyond with steps to prevent topological anomalies such as in Figure 3. The output may contain unbounded geometric faults.

There are several implementations for computing the convex hull with precise arithmetic. The output is a triangulation. If the input is degenerate, the output may contain simplices with zero volume. Clarkson's `hull` implementation of the randomized incremental algorithm restricts the input precision to about fifteen decimal digits. The implementation computes the exact sign of determinants [13]. It is a practical solution for precise convex hulls and Delaunay triangulations [14].

We timed `hull`, `hullio`[19] (a precursor of `hull` without exact arithmetic), `triangle`[41] (a two dimensional Delaunay triangulation program with exact arithmetic), and our implementation of Quickhull (`qhull 2.2`) on a Silicon Graphics 100 MHz R4000. These are fastest implementations known to the authors. We used a Sun SPARCstation for performance tuning of `qhull`. The times are the average and range of user CPU seconds for ten different trials.

`hull` computed the Delaunay triangulations of 10,000 uniform random points in a square in 10.23 (10.0-10.5) seconds, while `qhull` with merging computed them in 6.61 (6.5-6.7) seconds. If we compare `hullio` and `qhull` without merging, the corresponding figures are 6.2 (6.0-6.4) and 5.93 (5.8-6.1) respectively. The difference between `hull` and `qhull` with merging is largely due to exact arithmetic. Specialized code for two dimensions is much faster. The `triangle` program computes the same Delaunay triangulations in 1.3 (1.3-1.4) seconds.

`hull` computed the convex hulls of 20,000 uniform random points in a cube in 9.9 (8.2-10.9) seconds, `hullio` computed them in 4.5 (4.0-5.4) seconds, `qhull` with merging computed them in 1.75 (1.6-2.1) seconds, and `qhull` without merging computed them in 1.67 (1.6-1.8) seconds. These figures support the previous comparison for uniform random points.

`hull` computed the convex hulls of 5,000 random points on the surface of a hypercube in 28 (26-32) seconds. They had 6700 facets on average. For the same inputs, `qhull` with merging produced 1700 facets on average in 5.6 (5.2-6.0) seconds. It merged 2300 (2000-2500) facets. There was one facet of the convex hull for each facet of the hypercube. These facets contained 4500 coplanar points on average. `qhull` without merging produced 4300 facets on average in 1.9 (1.7-2.0) seconds. The larger time differences for `hull` and `hullio` and for `qhull` with and without merging reflect the increased use of exact arithmetic and facet merging.

5. Summary

Our goal is a practical algorithm for general dimension convex hulls. We have shown empirical evidence that the algorithm satisfies its balance conditions and that it performs like a randomized incremental algorithm that is output-sensitive to the number of vertices. The Quickhull Algorithm uses less space than most of the randomized incremental algorithms and executes faster for inputs with non-extreme points.

In addition, Quickhull uses merged facets to guarantee that the output is clearly convex. The algorithm is implemented with floating point arithmetic. It reports the maximum width of a merged facet. In $\mathbf{R}^5$ and higher, the maximum width may be wider than reported.

We have implemented Quickhull for general dimension. The program, `qhull`, computes convex hulls, Delaunay triangulations, Voronoi vertices, furthest-site Voronoi vertices, and halfspace intersections. It is available from <`http://www.geom.umn.edu/locate/qhull`> and <`ftp://geom.umn.edu/pub/software/ qhull.tar.Z`>. The program includes options for imprecise data and arithmetic, facet area and hull volume, partial hulls, input transformations, randomization, tracing, multiple output formats, graphical output, and execution statistics. The program can be called from within an application.

Over the last two years, 3000 copies of `qhull` were retrieved via `ftp`. It has been used for support structures in layered manufacturing [1], classification of molecules by their biological activity, vibration control, geographic information systems, neighbors of the origin in the $\mathbf{R}^8$ lattice, stress analysis, stability of robot grasps [5], spectrometry [6], constrained control allocation [8], robot navigation [17], micromagnetic modeling [38], and invariant sets of delta-sigma modulators [45].

Acknowledgments: A special thanks to Albert Marden and Victor Milenkovic for providing excellent environments for completing this work. The referees' comments greatly improved the presentation and content of this paper.

References

[1] S. Allen and D. Dutta. Determination & evaluation of support structures in layered manufacturing. *Journal of Design & Manufacturing*, 5:153–162, 1995.

[2] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23:345–405, 1991.

[3] D. Avis and D. Bremner. How good are convex hull algorithms? In *Proc. 11th Annual ACM Symposium on Computational Geometry*, pages 20–28, 1995.

[4] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The Quickhull algorithm for convex hull. Technical Report GCG53, The Geometry Center, 1993.

[5] L. Belsis, S. Grundmann, T. Rasanen, J. Sullivan, J. B. Walker, and M. Wright. Personal communications regarding `qhull`, 1995.

[6] J.W. Boardman. Automating spectral unmixing of AVIRIS data using convex geometry concepts. Fourth Airborne Geoscience Workshop, Washington, D.C., October 1993.

[7] J.-D. Boissonnat and M. Devillers-Teillaud. On the randomized construction of the Delaunay tree. *Theoretical Computer Science*, 1990.

[8] K. A. Bordignon and W. C. Durham. Closed-form solutions to constrained control allocation problem. *Journal of Guidance, Control, and Dynamics*, 18(5):1000–1007, 1995.

[9] D.F. Brown. Voronoi diagrams from convex hulls. *Information Processing Letters*, 9:223–228, 1979.

[10] A. Bykat. Convex hull of a finite set of points in two dimensions. *Information Processing Letters*, 7:296–298, 1978.

[11] D.R. Chand and S.S. Kapur. An algorithm for convex polytopes. *Journal of the ACM*, 7:78–86, 1970.

[12] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry: Theory and Applications*, 1991.

[13] K. L. Clarkson. Safe and effective determinant evaluation. In *Proc. 31st IEEE Symposium on Foundations of Computer Science*, pages 387–395, 1992.

[14] K. L. Clarkson. A program for convex hulls. http://netlib.att.com/netlib/voronoi/hull.html, 1995.

[15] K. L. Clarkson, K. Mehlhorn, and R. Seidel. Four results on randomized incremental constructions. In *Symposium on Theoretical Aspects of Computer Science*, 1992.

[16] K.L. Clarkson and P.W. Shor. Applications of random sampling in computational geometry, ii. *Discrete Computational Geometry*, 4:387–421, 1989.

[17] P. Cucka, N.S. Netanyahu, and A. Rosenfeld. Learning in navigation: Goal finding in graphs. Technical Report CAR-TR-759, Center for Automation Research, University of Maryland, 1995.

[18] T. K. Dey, K. Sugihara, and C. L. Bajaj. Delaunay triangulations in three dimensions with finite precision arithmetic. *Computer Aided Geometric Design*, 9:457–470, 1992.

[19] S. Dorward. Personal communication, 1992.

[20] W. Eddy. A new convex hull algorithm for planar sets. *ACM Transactions on Mathematical Software*, 1977.

[21] H. Edelsbrunner and N.R. Shah. Incremental topological flipping works for regular triangulations. In *Proceedings of the Symposium on Computational Geometry*, pages 43–52. ACM, 1992.

[22] R.W. Floyd. Personal communication to Preparata & Shamos on Quickhull, 1976.

[23] S. Fortune. Stable maintenance of point-set triangulation in two dimensions. In *30th Annual Symposium on the Foundations of Computer Science*. IEEE, 1989.

[24] S. Fortune. Computational geometry. In R. Martin, editor, *Directions in Geometric Computing*. Winchester UK: Information Geometers, 1993.

[25] K. Fukuda. `cdd` Reference Manual. ftp://ifor13.ethz.ch/pub/fukuda/cdd, 1995.

[26] P.J. Green and B.W. Silverman. Constructing the convex hull of a set of points in the plane. *Computer Journal*, 22(262-266), 1979.

[27] B. Grünbaum. Measures of symmetry for convex sets. In *Proceedings of the Seventh Symposium in Pure Mathematics of the American Mathematical Society, Symposium on Convexity*, pages 233–270, 1961.

[28] L. Guibas, D.E. Knuth, and M. Sharir. Randomized incremental construction of Delaunay and Voronoi diagrams. *Algorithmica*, pages 381–413, 1992.

[29] L. Guibas, D. Salesin, and J. Stolfi. Constructing strongly convex approximate hulls with inaccurate primitives. *Algorithmica*, 9:534–560, 1993.

[30] B. Joe. Construction of three-dimensional Delaunay triangulations using local transformations. *Computer-Aided Geometric Design*, 8:123–142, 1991.

[31] M. Kallay. Convex hull algorithms in higher dimensions. Unpublished manuscript, Dept. Mathematics, Univ. of Oklahoma, Norman, Oklahoma (see Preparata & Shamos 1985), 1981.

[32] D.G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. Computing*, 15:287–299, 1986.

[33] V. Klee. Convex polytopes and linear programming. In *Proc. IBM Sci. Comput. Symp.: Combinatorial Problems*, pages 123–158, 1966.

[34] Z. Li and V. Milenkovic. Constructing strongly convex hulls using exact or rounded arithmetic. In *Proceedings of the Symposium on Computational Geometry*, pages 197–207. ACM, 1990.

[35] V. Milenkovic. Personal communication, 1994.

[36] T. S. Motzkin, H. Raiffa, G. L. Thompson, and R. M. Thrall. The double description method. In H. W. Kuhn and A. W. Tucker, editors, *Contributions to the Theory of Games II*, volume 8 of *Annals of Mathematical Studies*, pages 51–73. Princeton University Press, 1953.

[37] K. Mulmuley. *Computational Geometry, An Introduction Through Randomized Algorithms.* Prentice-Hall, 1994.

[38] D.G. Porter, E. Glavinas, P. Dhagat, J. A. O'Sullivan, R. S. Indeck, and M. W. Muller. Irregular grain structure in micromagnetic simulation. *Journal of Applied Physics*, to appear April 1996.

[39] D F.P. Preparata and M.I. Shamos. *Computational Geometry. An Introduction.* Springer-Verlag, 1985.

[40] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proc. 18th ACM Symposium on the Theory of Computation*, pages 404–413, 1986.

[41] J. R. Shewchuk. Triangle: A two-dimensional quality mesh generator and Delaunay triangulator. Technical report, Carnegie-Mellon Institute, 1995.

[42] K. Sugihara. Topologically consistent algorithms related to convex polyhedra. In *Algorithms and Computation, Third International Symposium, ISAAC '92*, volume 650 of *Lecture Notes in Computer Science*, pages 209–218. Springer-Verlag, 1992.

[43] S. Teller. Personal communication, 1994.

[44] J.R. Weeks. Convex hulls and isometries of cusped hyperbolic 3-manifolds. Technical Report TR GCG32, The Geometry Center, Univ. of Minnesota, August 1991.

[45] B. Zhang, M. Goodson, and R. Schreier. Invariant sets for general second-order low-pass delta-sigma modulators with dc inputs. In *IEEE Inter. Symposium on Circuits and Systems*, volume 6, pages 1–4, 1994.