Survey of Algorithms for the Convex Hull Problem

Valentina Bayer
Department of Computer Science
Oregon State University
bayer@cs.orst.edu

March 19, 1999

Abstract

This paper presents a survey of deterministic algorithms, randomized algorithms and approximation algorithms for the convex hull problem. The algorithms range from almost three decade old ones, such as Graham's and Jarvis's, to modern randomized algorithms, overviewing output-sensitive algorithms and their worst case running times in higher dimensions.

1 Introduction

The convex hull of a set of points is the smallest convex set that contains all the points. In the plane, we can visualize the convex hull as a stretched rubber band surrounding the points that, when released, takes a polygonal shape. The extreme points in the set are the vertices of the convex polygon.

The convex hull has been extensively studied in computational geometry and its applications spread over an impressive number of fields: analysis of spectrometry data, file searching, cluster analysis, collision detection, crystallography, pattern recognition, image processing, numerical integration, statistics, metallurgy, cartography, etc. Other problems can be reduced to the convex hull: half-space intersections, Delaunay triangulation and Voronoi diagrams.

For example, the size of a distant moon is to be determined from a faint image. Some points on the moon may not be visible on the image. The convex hull of the image can be used to obtain an approximate shape of the moon, which is unaffected by missing points.

A similar case is the measurement of brain size in a standardized way. Brains are too convoluted to be measured exactly; the convex hull is a way to overcome this problem.

Another example of analyzing data is in physics, where in a particle scattering problem, the data is expected to lie in a bounded region.

In a machine vision application consisting in navigating through a field of obstacles, the objects can be simplified by reducing them to their convex hulls.

Many computer graphics applications require a graph to be printed or displayed on a fixed area. By knowing the convex hull of the graph, the program will compute how much to scale it to fit in the given area.

This paper presents an overview of deterministic algorithms, randomized algorithms and approximation algorithms, for two dimensions and for higher dimensions. An analysis of the algorithms and problems related to the convex hull is also discussed. The selection of the algorithms was made from the historical and the ground-breaking ones, as the bibliography on convex hull problem is huge.

2 Definitions

A set S in E^d is *convex* if for any two points p, q in the set, the segment $\overline{pq}$ is entirely contained in the set. By E^d we denote the d-dimensional Euclidian space. We will assume the dimension d is a fixed constant.

Given a set of n points, the *convex hull* is the smallest convex set that contains all n points (see Figure 1).

Convex hull problem: "given a set of n points in E^d, construct its convex hull (that is, the complete description of its boundary)"([8]).

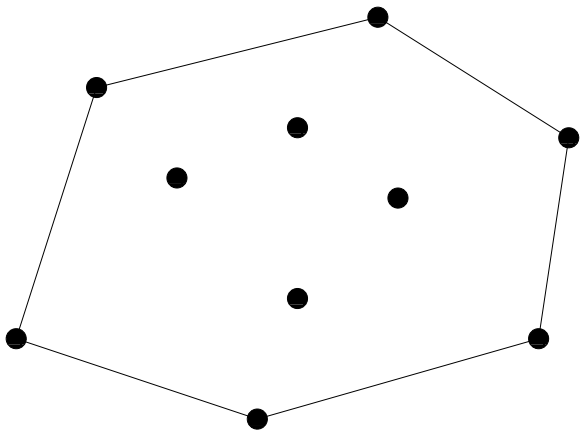

Figure 1: The convex hull of 10 points in the plane

3 Lower bound

SORTING $\leq_{LIN}$ (Planar) CONVEX HULL

Given n real numbers $x_1, x_2, \ldots, x_n$ we must show how an algorithm for CONVEX HULL can be used to sort the points, with only linear time overhead. The idea is to associate to each number x_i the point (x_i, x_i^2) lying on the parabola $y = x^2$. The convex hull of the points gives the vertices in anticlockwise order, starting from an arbitrary vertex (say the lowest rightmost one). In linear time we can find the vertex with the minimum x value and then copy, in order, all the other vertices, thus producing the list of the points sorted by abscissa. Notice that this transformation involves only arithmetic operations and comparisons.

Because sorting has a lower bound of $\Omega(n \log n)$, the planar convex hull also has a lower bound of $\Omega(n \log n)$. In one dimension, the convex hull of a set of points is the smallest interval containing them, which can be found in linear time (by finding the points with the smallest and the biggest value). For higher dimensions, the result is also valid, since any set of points in two dimensions can be embedded in E^d, with $d > 2$.

"Yao showed that $\Omega(N \log N)$ is a lower bound just by determining the points belonging to the convex hull, not necessarily producing them in cyclic order. This lower bound was proved for a decision tree model with quadratic tests, which accommodates all the known planar convex hull algorithms"([10]).

4 Deterministic algorithms for the planar convex hull problem

There is a strong analogy between the convex hull problem and the sorting problem (for the convex hull, we will have to think of all given points as points on the hull). In this section we will present the ideas for some of the fundamental algorithms that solve the convex hull problem in the plane. The algorithms will output the vertices of the convex hull in counterclockwise order.

"Graham's algorithm uses sorting explicitly. Jarvis's algorithm is similar to selection sort. The divide-and-conquer algorithms are the geometric equivalent of merge sort. The on-line algorithm is an insertion sort. Quickhull is

analogous to quicksort"([8]) .

For the 2-D problem, the following theorem is very important: "A point p fails to be an extreme point of a plane convex set S only if it lies in some triangle whose vertices are in S but is not itself a vertex of the triangle."

4.1 Graham's scan

Before discussing the algorithm, another theorem will be stated: "Consecutive vertices of a convex polygon occur in sorted angular order about any interior point."

Graham's algorithm starts with an internal point p_{origin} (for example, the centroid of the triangle formed by 3 non-collinear points). It then transforms the coordinates of the other points so p_{origin} is at the origin. Of course, if the origin was inside the set S, we can skip the above work. Afterwards, we sort the n points lexicographically by polar angle and distance from the origin. This takes time $O(n \log n)$. In linear time, a scan (see Figure 2) can be done around the sorted points to eliminate the internal ones, and therefore the result will be the points on the convex hull.

The algorithm uses $O(n)$ space (the sorted points are arranged into a doubly-linked circular list; the scan will only leave the points to appear on the convex hull).

The scan begins at the lowest rightmost point p_{start} (this is on the convex hull). When traversing a convex polygon, only "left turns" are made. Triples of consecutive points $p_1p_2p_3$ are examined, and if p_3 lies to the left of the oriented segment $p_1\vec{p}_2$, then the scan is advanced, otherwise point p_2 is eliminated, and $p_0p_1p_3$ is examined (p_0 is the point before p_1). This "back" elimination continues until a left turn angle is found. The scan terminates when it advances all the way around to p_{start}. Since there are only n points in the set, we cannot advance/(delete points) more than n times, therefore the scan takes $O(n)$ time.

Graham's algorithms takes $O(n \log n)$ time and it is a worst case optimal algorithm. It cannot be generalized to higher dimensions, because the theorem it is based on (see above) holds only in the plane.

It is interesting to notice that only arithmetic operations and comparisons are required in this algorithm, and an explicit conversion to polar coordinates (for sorting) is not necessary. Indeed, in sorting, the comparison between the polar angles of two points p_1 and p_2 can be done by computing the signed

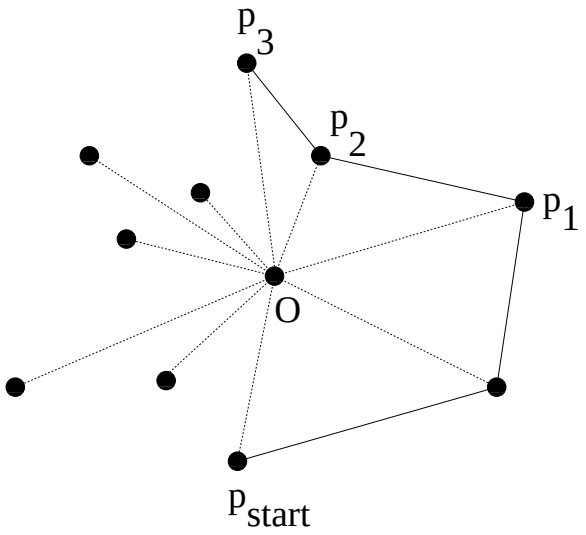

Figure 2: Graham's scan (in counterclockwise direction); point $p2$ is eliminated because $p_1p_2p_3$ makes a right turn

area of the $triangle(Op_1p_2)$. In general, let $p_i = (x_i, y_i)$, for $i = 1, 2, 3$. The determinant

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$$

gives twice the area of the signed $triangle(p_1p_2p_3)$, where the sign is + if and only if $p_1p_2p_3$ form a counterclockwise cycle (equivalent $p_1p_2p_3$ forms a left turn angle, or p_3 has a larger polar angle than p_2, with respect to p_1 as origin).

4.2 Jarvis's march

Theorem: "The line segment defined by two points of a set is an edge of the convex hull if and only if all the other points of the set lie on the same side of the edge (or on the edge)".

From a point on the hull, the intuition is to find the next point on the hull by "sweeping" a horizontal ray in counterclockwise direction until it hits

another point(see Figure 3).

The algorithm starts with the lowest rightmost point p_{start} (this is on the convex hull). The next point on the hull is the one that makes the smallest polar angle with respect to p_{start} as origin (consider the farthest one, for ties). Once determined, we apply the same idea for the new found point. We continue like this until we find the highest rightmost point p_{high}, from where we continue the search, but now the least polar angle is with respect to the negative x axis. Again, the algorithm uses only arithmetic operations and comparisons (the smallest polar angle can be found without using polar coordinates, as explained in the section on Graham's algorithm).

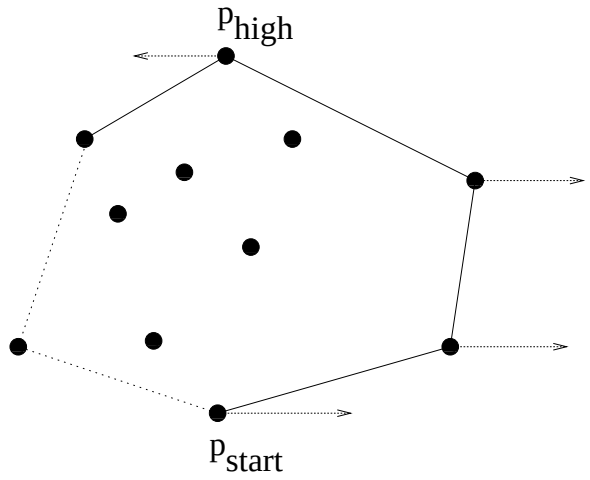

Figure 3: Jarvis' s march

Let f be the number of points on the hull. For each of them it takes $O(n)$ to find the (next) point with the minimum polar angle, therefore the algorithm's running time is $O(nf)$. This is an example of an output-sensitive algorithm (running time is dependent on the size of the output as well as the input) and we will talk more about this in a future section.

In the best case, when f is known in advance to be small, the algorithm runs in linear time $O(n)$. In the worst case, when all points are on the hull, the algorithm has complexity $O(n^2)$.

For the average case analysis, we must compute the expected value of

f, $E(f)$. We enumerate some known results for the average time of Jarvis's march, for different distributions of the n points in the set (the results are asymptotic, for $n \to \infty$):

- n points chosen uniformly inside a circle $\to O(n^{4/3})$
- n points chosen uniformly inside a sphere $\to O(n^{3/2})$
- n points chosen from a normal distribution (in the plane) $\to O(n(\log n)^{1/2})$
- n points chosen uniformly inside a convex polygon $\to O(n \log n)$

Jarvis's algorithm is a special case (in two dimensions) of the "gift-wrapping" technique.

4.3 Quickhull

Quickhull starts by finding the line passing through the points l and r with smallest and largest absicissae. This line divides the points into two subsets of points, S_1 above the line and S_2 below the line. The concatenation of the polygonal chains of S_1 and S_2 gives the convex hull. Here we will focus on S_1.

The algorithm finds the point $h \in S_1$ that has the maximum area among all triangles (formed by a point $p \in S_1$ and l, r). In case of ties, the left-most point is chosen. h belongs to the convex hull. The points inside the $triangle(hlr)$ can be eliminated. S_1 is partitioned (in linear time) into two sets S_1^1 and S_1^2, where S_1^1 contains the points to the left of the ray $l\vec{h}$ and to the left of the ray $r\vec{h}$, and S_1^2 contains the points to the right of the ray $l\vec{h}$ and to the right of the ray $r\vec{h}$. The Quickhull algorithm is recursively applied to S_1^1 and S_1^2. The base case of the recursion is for a set containing two points, which will be returned as output.

If the partition is approximately balanced, Quickhull runs in $O(n \log n)$, but it has $O(n^2)$ worst case running time.

4.4 Divide-and-conquer

To avoid the worst case running time of $O(n^2)$, the partitions must be (almost) equally balanced. This suggests dividing the set of points S into two

subsets S_1 and S_2, each containing half of the points. Recursively we find the convex hulls $CH(S_1)$ and $CH(S_2)$, which are two convex polygons.

$$CH(S_1 \cup S_2) = CH(CH(S_1) \cup CH(S_2))$$

Theorem: "The convex hull of the union of two convex polygons can be found in time proportional with their total number of vertices".

If $T(n)$ is the time to find the convex hull of a set of n points, then the differential equation for the divide-and-conquer algorithms is

$$T(n) = 2T(n/2) + cn$$

where c is a constant. The solution to the equation is $\Theta(n \log n)$. The base case (appears when the number of points is small) constructs the convex hull by some direct method.

By generalizing the divide-and-conquer algorithm in 2-D, Preparata and Hong devised an algorithm for the three dimensional case that runs in time $O(n \log n)$.

The algorithm starts by sorting the points by their x coordinate, then it divides them in two halves S_1 and S_2. So far, $O(n \log n)$ time is spent. After recursively computing $CH(S_1)$ and $CH(S_2)$, the result will be two nonintersecting 3-dimensional polytopes (they do not intersect because of the initial sorting of the points). In the merging step, the hull of the union of $CH(S_1)$ and $CH(S_2)$ is formed by "gift-wrapping" the two objects, in time $O(n)$. The overall complexity is $O(n \log n)$.

4.5 On-line algorithms

All the algorithms discussed above are off-line algorithms, that is, all the points are present before the computation begins. In many practical applications the points arrive one at a time, and to compute their convex hull we need on-line algorithms, that recompute the convex hull with each new point received (after $p_1, p_2, \ldots p_i$ points are received, the on-line algorithms have computed CH($\{p_1, p_2, \ldots p_i\}$)). There is a theorem stating that "any on-line convex hull algorithm must spend $\Omega(\log n)$ processing time between successive inputs, in the worst case".

Preparata ([8]) developed an optimal on-line algorithm that finds the convex hull of a set of n points in the plane in $\Theta(n \log n)$ time with $\Theta(\log n)$

update time. His algorithm is based on the construction of the support lines from a point to a convex polygon.

5 Randomized algorithms

The randomized algorithms for convex hulls are all Las-Vegas kind: they always give the correct solution. It is also interesting to remark that, while in many computational problems randomized algorithms achieve faster times than the deterministic ones, in computational geometry they often only match the running times of known deterministic algorithms ([7]). Their advantage lies in the fact that they are easy to implement, easy to understand and easy to extend to higher dimensions.

We will present the idea for a 2-D and 3-D randomized algorithm. The algorithms are incremental, that is, elements are considered one at a time, in random order.

5.1 In the plane

The points are first randomly permuted. At step i, point p_i is added to $conv(S_{i-1})$ to form $conv(S_i)$. Let p_0 be a point internal to $conv(S)$ (for example, the centroid of $conv(S_3)$). For every point $p \in S \setminus S_i$, a bidirectional pointer is maintained to the edge of $conv(S_i)$ cut by the ray p_0p. If point p_i is inside $conv(S_{i-1})$ (this is decided in constant time), the pointer to the edge is deleted. Otherwise, starting from the vertices of the edge cut by p_0p_i, we find the vertices of $conv(S_{i-1})$ that have to be eliminated, and the neighbours of p_i in $conv(S_i)$, v_1 and v_2 (see Figure 4). For the points that cut the edges being deleted, we update their pointers (in constant time) to either p_iv_1 or p_iv_2 (these two lines are called the supporting lines from p_i to the convex polygon $conv(S_{i-1})$).

The algorithm uses $O(n)$ space.

At each step, at most two edges are created, so over n steps, at most $2n$ edges are created/deleted. We also have to take into consideration the cost of updating the pointers for the points outside the current convex hull. For this, we will use *backward analysis*, where we imagine running the algorithm backwards and deleting a random point from $conv(S_i \setminus S_3)$ to form $conv(S_{i-1})$. The point can be from inside the $conv(S_i)$. The number of pointers that have

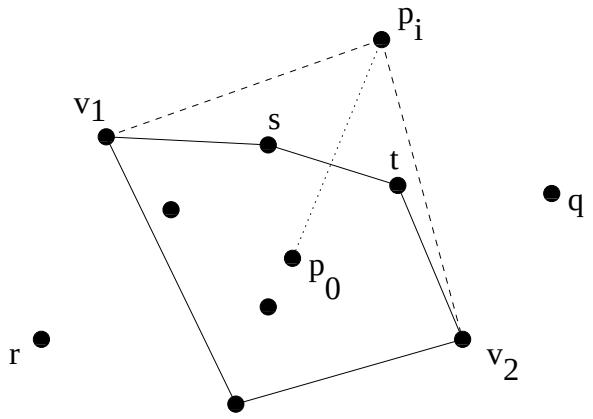

Figure 4: Adding p_i to $conv(S_{i-1})$: points s and t are eliminated, the pointer for q needs to be updated, the one for r does not

to be updated in the forward run of the algorithm is the same as the number of pointers being updated in the backward algorithm. At step i, there are $n-i$ points outside $conv(S_i)$. For one of these outside points, the probability that its pointer will be updated is equal to the probability that the edge it cuts will be deleted. An edge is deleted if one of its end points are deleted. But a point is deleted at random from the $i-3$ points in $conv(S_i \setminus S_3)$, that is, with probability $O(1/i)$. Therefore the expected number of pointers updated is $O((n-i)/i)$, or $O(n/i)$. After n steps, the expected running time of this randomized incremental algorithm is $O(n \log n)$.

5.2 In three dimensions

Computing the convex hulls in three dimensions can be reduced to the problem of computing half-space intersections in 3-D. To understand this, we have to introduce the notion of *geometric duality*.

In 3-D the dual of the point $p = (a, b, c)$ is the plane defined by the equation $ax + by + cz + 1 = 0$. If the point p is at distance d from the origin, it can be shown that its dual plane is perpendicular to the line joining p with the origin, and the distance from the origin to this plane is $1/d$. Furthermore, the origin is between the plane and the point.

Assume the convex hull of a set of points contains (inside) the origin. Then for every point, we compute its dual plane and for this plane we define the half-space bounded by the plane that contains the origin. We can see the half-space intersection problem as the dual of the convex hull problem.

The intersection of n half-spaces in three dimensions is a (possibly unbounded) convex polyhedral set in space (if non-empty). We will call this $inter(S)$. The polyhedron $inter(S)$ can be represented as a planar graph with at most n vertices and edges. The randomized algorithm computing the half-spaces intersections is similar to the randomized algorithm that computes the convex hull in 2-D. The details can be found in [7].

The expected running time of the randomized incremental algorithm for computing the intersection of n half-spaces in 3-D is $O(n \log n)$. Therefore we get a randomized algorithm for computing the convex hull of n points in three dimensions with expected running time of $O(n \log n)$.

6 Higher dimensions

"A polyhedral set in E^d is the intersection of a finite set of closed half-spaces (a half-space is the portion of E^d lying on one side of a hyperplane). A polyhedral set is convex. A bounded d-dimensional polyhedral set will be referred to as a *d-polytope*. A $(d-1)$-face is called a *facet*" ([8]).

In the worst case, it is known that a d-polytope with n vertices can have as many as $\Theta(n^{\lfloor d/2 \rfloor})$ facets ([4]). However, for randomly generated point sets and point sets used in practice, convex hulls often have fewer faces than the worst case bound. Consequently, many algorithms have been analyzed in terms of both n, the size of the input, and in terms of the output size. These algorithms are called *output sensitive*. For the convex hull problems, there are two types of such algorithms:

- the ones that enumerate the facets of the convex hull
- the ones that produce the complete *facial lattice*, that is, a description of all faces and incidence relationships of the convex polytope

Jarvis's march has running time $O(nf)$, where f is the number of vertices on the convex hull, therefore is an output sensitive algorithm. For small f the algorithm runs in linear time, while its performace is $O(n^2)$ in the worst case (all points on the convex hull).

Let f denote the number of facets (in the plane, the number of facets, or edges, is equal to the number of vertices). If f is very large (that is, exponential in the number of vertices), a trivial lower bound for the convex hull problem is $\Omega(f)$. If f is small, the lower bound drops to $\Omega(n \log f)$. Therefore, when taking the output size f into account, the lower bound for the convex hull problem is $\Omega(n \log f + f)$.

There are two general methods that solve the convex hull in higher dimensions: the *"gift-wrapping"* algorithm and the "beneath-beyond" method.

The idea of the gift-wrapping method is to proceed from one facet towards an adjacent facet, in the same way we would wrap a sheet around a plane-bounded object. A special case of this technique is Jarvis's march (in 2-D).

The worst case complexity of the gift-wrapping method is $O(nf)$ for enumerating the facets, or $O(\min(nL^2, n^{\lfloor d/2 \rfloor + 1}))$ for producing a lattice of size L.

The beneath-beyond method is an incremental approach that constructs the convex hull by adding one point at a time. If $conv(S_{i-1})$ is the current hull and we want to add a point p_i external to the hull, the 3-D analogy is to construct a "cone" from p_i to $conv(S_{i-1})$ and remove the points from $conv(S_{i-1})$ that are in the "shadow" of the cone. The algorithm has worst case complexity $O(n^{\lfloor (d+1)/2 \rfloor})$ and is optimal for even dimensions ([4]).

Chazelle ([3]) improved the running time to $O(n \log n + n^{\lfloor d/2 \rfloor})$ by derandomizing a randomized incremental algorithm by Clarkson and Shor. Because $\Omega(n^{\lfloor d/2 \rfloor})$ is a trivial worst case lower bound (the size of the output), Chazelle's algorithm is optimal in all dimensions, in the worst case.

Seidel's "shelling" algorithm, related to the gift-wrapping method, runs in time $O(n^2 + f \log(n))$ for facet enumeration and $O(n^2 + L \log(n))$ for computing the facial lattice.

There are a couple of divide-and-conquer algorithms that construct the convex hull in:

- 4 dimensions in $O((n + f) \log^2 f)$ time and $O(n + f)$ space ([9])
- 5 dimensions in $O((n + f) \log^3 f)$ time ([1])

Erickson showed that, in the worst case, $\Omega(n^{\lceil d/2 \rceil - 1} + n \log n)$ time is needed to determine the number of convex hull facets ([5]). For odd dimensions, this matches Chazelle's upper bound.

7 Two dimensions revisited

This section discusses output-sensitive algorithms for the planar convex problem.

Kirkpatrick and Seidel proposed an $O(n \log f)$ time algorithm, that is both output-sensitive and worst case optimal ([6]). It constructs separately the upper and the lower hulls. For the upper hull, the points in S are split into two equally-size subsets (separated by a vertical line). The innovative idea is to compute the merge of the two subsets (in this case, the "upper bridge"), before recursively solving the subproblems; this is an application of the principle of "marriage before conquest".

The points of S are paired into couples (p_a, p_b) such that $x_a < x_b$. These ordered pairs define $\lfloor n/2 \rfloor$ straight lines. The median m of the set of lines' slopes is found, then a straight line of slope m that contains at least a point of S but has no point of S above it is determined. This straight line either contains the "upper bridge", or can be used to remove some internal points in S, until the "upper bridge" is found. Actually, the bridge can be defined as the solution to a linear programming problem that can be solved in linear time.

Other algorithms were developed that run in $O(n \log f)$ time. We mention here three of them and refer to the original papers for implementation and analysis details.

Chan, Snoeyink and Yap proposed a deterministic variation of Quickhull which runs in time $O(n \log f)$ and can be generalized to higher dimensions ([9]). This algorithm also finds the median slope, and is faster than Kirckpatrick-Seidel's algorithm by a constant factor. The median-finding is the most costly operation in both algorithms. Let cn be the time to find the median of n numbers. Then Kirckpatrick and Seidel's algorithm spends, in the worst case, $3bn \log f$ time and Chan et al's algorithm spends approximatively $1.2n \log f$.

Wenger ([11]) designed a randomized version of Quickhull (see section on Quickhull), where instead of finding the extreme point h that maximizes the area of the triangle(hlr), two points q_1 and q_2 are chosen at random such that l, r, q_1, q_2 are the vertices of a convex quadrilateral. Then h is chosen (from the points of S_1 situated on the opposite side of the line(q_1q_2) from l and r) to maximize the area of the triangle(hq_1q_2). Another difference from Quickhull is the attempt to balance the partitions S_1^1 and S_1^2 by eliminating

interior points. It is important to note, though, that the hidden constants in the analysis of the $O(n \log f)$ running time are very high (at least 300).

Bhattacharya and Sen proposed in [2] an algorithm similar to Chan et al's, but the median-finding step is replaced by an approximate median. The algorithm is a randomized one and runs in expected time $O(n \log f)$. Unlike Wenger's analysis, the constant associated with the expected running time is quite small. In one strategy, the median element of a given set is replaced by a random element of the set. The algorithm terminates with probability $1 - O(h^{-1})$. In the other strategy, the median element is replaced by the exact median of some randomly selected elements. In this case, the algorithm terminates with probability $1 - O(2^{-\Omega(n^{1/4})})$.

Also, T.M. Chan reported an output-sensitive convex hull determinstic algorithm based on ray shooting, with optimal $O(n \log f)$ time in two and three dimensions.

8 Approximation algorithms

Approximation algorithms are useful for applications that require rapid solutions, even at the expense of accuracy. Examples include statistical applications with noisy observations within a well-defined range.

There are two types of approximate convex hulls: conservative and liberal. Conservative approximations compute a convex hull that is included (as a set of points) in the true convex hull. Liberal approximations output an approximate convex hull that is a superset of the true convex hull.

We present a conservative algorithm for the planar convex hull (see [8]). The idea is to split the points among k vertical equally-spaced strips (see Figure 5), for each strip keeping track of the points with the smallest and largest y coordinate. The bounding strips also contain the two points of the set with smallest and largest x coordinate. It then remains to construct the convex hull for these $2k + 4$ points.

To assign a point $p = (x, y)$ to a strip, we have to compute the floor of the ratio $(x - x_{min})/$ (width of a strip) $= (x - x_{min})/((x_{max} - x_{min})/k)$. That means we have to add the floor function to the primitive operations allowed in our computation model.

The algorithm requires $\Theta(k)$ space. Finding the minimum and maximum x value of n points requires $\Theta(n)$ time, and also finding the extremes for all

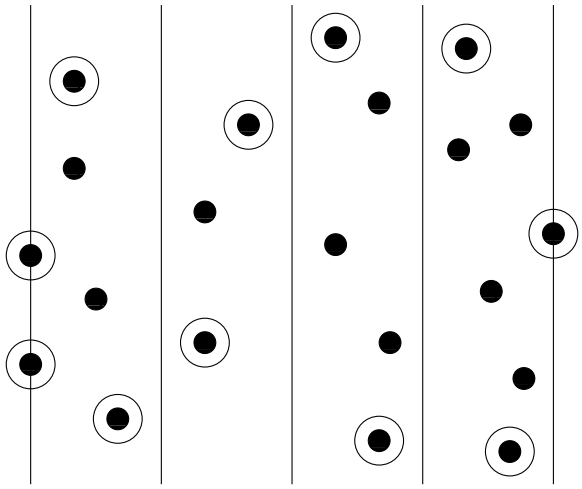

Figure 5: Partition into $k = 4$ strips; for each strip the vertical extremes are outlined; the set's horizontal extremes are also outlined

strips takes $\Theta(n)$ time (this is the time to distribute the points in the strips). The strips are already sorted in increasing order of the x value, it remains to sort in increasing order the points in each strip, so in time $\Theta(k)$ we will have all the $2k + 4$ points sorted by abscissa. Because the points are already sorted, we can apply the Graham scan to compute their convex hull and this takes linear time $\Theta(k)$. Therefore the time complexity of this approximation algorithm is $\Theta(n + k)$, that is linear time for constant k.

We also have to think of the accuracy of this algorithm: if the approximate convex hull leaves out some points of the original set, how far are they? Any point $p \in S$ that is not inside the approximate convex hull is within distance $(x_{max} - x_{min})/k$ of the hull. So if we want to increase the accuracy, we have to choose a higher constant k, but we will also do more work.

The 3-D case follows the same idea of assigning points to strips. We consider the rectangle in the (x, y) plane defined by x_{min}, x_{max} and y_{min}, y_{max}. We split this into k^2 squares and imagine its 3-D "block extension". In every "block" we keep the points with minimum and maximum z value. Again, it takes $O(n)$ time to distribute the points into the blocks. Unfortunately, we are not able to take advantage of the structure of the grid (like in the

2-D case, where the points were ordered by absicissa), so for the resulting $\Theta(k^2)$ points we will apply a general method to compute its convex hull, namely Preparata-Hong's general hull algorithm in 3-D, with running time $O(n \log n)$ for n points. For $n = k^2$ we get $O(k^2 \log k)$. So the 3-D approximation algorithm requires $O(k^2)$ space and $O(n + k^2 \log k)$ time. The distance from the approximate hull to any point external to it is bounded above by $\sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}/k$.

9 Related problems

9.1 Diameter of a set of points

Given n points in E^d, the largest distance between two points is called the *diameter* of the set. For $d \geq 2$ the problem has a lower bound of $\Omega(n \log n)$.

In the plane, the diameter of a set is equal to the diameter of its convex hull. The convex hull can be computed in $O(n \log n)$ time. Because the convex hull is a convex polygon in the plane, its diameter can be found in time linear in its number of vertices, that is $O(n)$, so the diameter of a set of points can be computed in $O(n \log n)$.

9.2 Depth of a set of points (convex layers)

"The depth of a point p in a set S is defined as the number of convex hulls (convex layers) that have to be stripped from S before p is removed". To find the set of convex layers, we compute and remove the convex hull of S, then compute and remove the convex hull of the remainder and so on until no points are left (the procedure is known as "shelling" or "peeling"). In the plane, the convex layers can be computed in $O(n \log n)$ time.

9.3 Convex hull of a simple polygon

This is a special case of the convex hull problem, where the points are known to be the vertices of a simple polygon (no intersecting edges). The lower bound for this problem is $\Omega(n)$. It can be shown that the convex hull of an n-vertex simple polygon can be constructed in optimal $\Theta(n)$ time and $\Theta(n)$ space.

9.4 Isotonic regression

"The problem of isotonic regression is to find a best isotone (that is, monotone nonincreasing or nondecreasing) approximation to a finite point set". The error norm usually used is L_2, or least squares.

The set of points $p_i = (x_i, y_i)$ is first sorted by the x coordinate (in $O(n \log n)$). We can define the cumulative sum diagram to be the set of points $q_j = (j, s_j)$, where $q_0 = (0, 0)$ and s_j is the sum of $y_1, \ldots, y_j$. It is known that the slope of the lower-hull of the cumulative sum diagram gives the slope of the isotonic regression. Therefore the least-square isotonic regression of a set of n points in the plane can be found in $O(n \log n)$ time.

If the points are already sorted by absicissa (for example, in many applications data is recorded in increasing order of time), then an isotonic regression can be found in linear time (Graham's algorithm computes the convex hull in linear time if the points are already sorted).

10 Conclusions or "Which algorithm to choose?"

After reviewing some of the fundamental algorithms for the convex hull problem, we are left with the hard task of suggesting what algorithm to choose for a given problem. The answer is not easy, and it depends on the knowledge of the problem.

Let's consider the planar case. If we know that the number of edges of the convex hull is small, then we can apply Jarvis's algorithm and we solve the problem in linear time. If the points belong to a simple polygon or are already sorted, again we obtain linear time (in the second case, Graham's algorithm can be applied directly, for the first case see [8]).

Or we can choose an output-sensitive algorithm, like Kirckpatrick and Seidel's, with time $O(n \log f)$. In the plane, the number of vertices f can be at most n, therefore their algorithm is worst case optimal. As mentioned before, there are a couple of output-sensitive algorithms that report the same time analysis $O(n \log f)$, but some require more space, some have higher hidden constants (like Wenger's algorithm), some are easier to understand and implement, some require linear programming techniques to be solved, etc.

If we were to choose among the classical deterministic algorithms (with

time complexity $O(n \log n)$), we will have to pay attention to the amount of space used, even if the reported space is $O(n)$. Divide-and-conquer algorithms, due to the merging part, will require more space than Graham's scan, for example, but they are well suited for parallel environments.

Randomized algorithms also require more space (still linear), and their performance only matches the running time of the deterministic algorithms. But they are easy to understand and implement and are scalable to higher dimensions (by using geometric duality, a powerful tool). An open problem is to design Monte Carlo algorithms for the convex hull problem.

If the nature of the problem provides the points "on-line", either deterministic or randomized algorithms can be used to compute the convex hull, with similar performances (total time $\Theta(n \log n)$, with update time $\Theta(\log n)$).

For three dimensions, either Preparata and Hong's or Chazelle's algorithms can be chosen, or the randomized 3-D algorithm, all with expected time $O(n \log n)$.

For higher dimensions, the worst case lower bound is $\Omega(n^{\lfloor d/2 \rfloor})$. Seidel showed that this lower bound is tight for even dimensions and Erickson showed that it is tight for odd dimensions.

Approximation algorithms shouldn't be neglected, especially when quick results are required; the more accurate the answers, the more work is to be done, though.

In conclusion, the convex hull is an important problem with many applications. It has been thoroughly worked by many researchers. It will be exciting to see if more progress can be made in the future.

References

[1] N.M. Amato and E.A. Ramos. On computing Voronoi diagrams by divide-prune-and-conquer. *Proceedings of the 12th Annual ACM Symposium on Computational Geometry, pages 166-175*, 1996.

[2] B.K. Bhattacharya and S. Sen. On a simple, practical, optimal, output-sensitive randomized planar convex hull algorithm. *Journal of Algorithms, Vol. 25, pages 177-193*, 1997.

[3] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Computational Geometry, Volume 10, pages 377-409*, 1993.

[4] H. Edelsbrunner. *Algorithms in combinatorial geometry.* Springer-Verlag, 1987.

[5] J. Erickson. New lower bounds for convex hull problems in odd dimensions. *Proceedings of the 12th Annual ACM Symposium on Computational Geometry, pages 1-9*, 1996.

[6] D. Kirckpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing, Vol 15, pages 287-299*, 1986.

[7] R. Motwani and P. Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.

[8] F.P. Preparata and M.I. Shamos. *Computational Geometry: An introduction.* Springer-Verlag, 1985.

[9] M.C. Timothy, J. Snoeyink, and C. Yap. Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams. *preliminary version of this paper in Proceedings 6th ACM-SIAM SODA, pages 282-291*, 1995.

[10] J. van Leeuwen (managing editor). *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity.* MIT Press, 1990.

[11] R. Wenger. Randomized quick hull. *to appear, Algorithmica*, 1995.