An Algorithm to Find the Facets of the Convex Hull in Higher Dimensions

Wm. Mott Stewart
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick

Abstract

Given a set S of n points in $\Re^d$ we describe the convex hull of S by a linked set of its facets. Starting with an initial simplex, the points of S are added one by one with a simple update step. The algorithm is $O(d^3 \times n \times \Phi(n, d-1))$, where $\Phi(n,d)$ is the worst case number of facets of a convex hull of n points in $\Re^d$. This is asymptotically optimal for even d.

1 Introduction

The convex hull problem is a central one in computational geometry, with application to pattern recognition, image processing, stock cutting and allocation, generation of voronoi diagrams, and a number of other problems, where the generation of the convex hull is often a preprocessing step for a main algorithm.

Definition 1 *The convex hull* $\text{conv}(S)$ *of a finite set* $S = \{p_1, \ldots, p_n\}$ *in* $\Re^d$ *is the set of all convex combinations of the points of* S*, i.e.*

$$\text{conv}(S) =$$

$$\{\alpha_1 p_1 + \cdots + \alpha_n p_n, \mid \alpha_1 + \cdots + \alpha_n = 1,\ \alpha_i \geq 0\}$$

The convex hull is a convex polytope, or bounded convex polyhedron, that can be represented by its set of faces.

In $\Re^1$ the convex hull is the closed segment $[\min(S), \max(S)]$, and the practical solution is the set of faces $\{\min(S), \max(S)\}$. The problem is $\Omega(n)$ and an optimal solution is straightfoward.

In $\Re^2$ the convex hull is the smallest convex polygon containing S, and the practical solution is the set of faces or edges of the polygon. The problem is transformable from sorting, and therefore $\Omega(n \log n)$. The Graham Scan [3] was the first optimal algorithm for $\Re^2$, using a sorting preprocessing step and amortized constant time update step. A number of subsequent algorithms have been developed.

In $\Re^3$ the convex hull of S is the smallest 3-dimensional polytope containing S, with vertices that coincide with the extreme points of S. The number of faces is $O(n)$, but the lower bound complexity $\Omega(n \log n)$ is inherited from $\Re^2$. Preparata and Hong's 'divide and conquer' algorithm [4] is the only published optimal solution for $\Re^3$.

In $\Re^d$, $d \geq 4$, the number of faces of a convex hull can rise rapidly, and be as much as $\Phi(n,d)$, where $\Phi(n,d) = O(n^{\lfloor d/2 \rfloor})$. There are four known algorithms for higher dimensions.

Chand and Kapur's 'giftwrapping' algorithm [2] was the first solution with a complexity of $O(n \times \Phi(n,d))$. This algorithm only maintains the facets or highest dimensional faces of the hull.

Kallay's 'beneath-beyond' algorithm [5] has the same complexity as [2], $O(n \times \Phi(n,d))$. This algorithm maintains the complete facial graph, or faces of all dimensions, and is on-line.

Seidel's algorithm [6] operates in the dual space, updating the (dual) facial graph with the addition of each new point by a similar method to Kallay. Seidel's algorithm, however, has complexity $\Theta(\Phi(n,d))$ for even d and $O(n \times \Phi(n,d))$ for odd d, which is the same as the algorithm presented here. However, the algorithm presented here has a much simpler update step, and does not need to maintain the entire facial graph.

Seidel also discovered another algorithm [7] which constructs the convex hull at logarithmic cost per facet, and is therefore $O(F \log n)$, where F is the actual number of facets of the convex hull.

2 Convex Polytopes

We consider a set S of n points in $\Re^d$, for any $d \geq 2$ and $n \geq d+1$. For simplicity we assume that S is in general postion.

Definition 2 *An extreme point* $p \in S$ *is a point that is not the convex combination of any other two points of* S*. The set of all extreme points is denoted* $\text{ext}(S)$.

Theorem 1 (Brondsted, theorem 7.2) *The convex hull of* S *is equivalent to the polytope* $\mathcal{P}$ *of the extreme points of* S*, i.e.,* $\mathcal{P} = \text{conv}(\text{ext}(S))$.

Definition 3 *A face $\mathcal{F}$ of a polytope $\mathcal{P}$ is a convex subset such that, for any two distinct points $y, z \in \mathcal{P}$ with $]y,z[\cap \mathcal{F} \neq \emptyset$, then $[y,z] \subseteq \mathcal{F}$.*

We call a face $\mathcal{F}$ a k-face if $\dim(\mathcal{F}) = k$. The *proper* faces of $\mathcal{P}$ are of dimension 0 through $d-1$. All proper faces lie on the boundary of $\mathcal{P}$.

Theorem 2 (Brondsted, theorem 7.3)
Every proper k-face $\mathcal{F}$ of a polytope $\mathcal{P}$ is itself a k-polytope, and $\text{ext}(\mathcal{F}) \subset \text{ext}(S)$.

We represent a k-face $\mathcal{F}$ by its $k+1$ extreme points $\text{ext}(\mathcal{F}) \subset S$.

The algorithm presented here maintains only $(d-1)$-faces or *facets*, and the $(d-2)$-faces or *subfacets*. We shall also consider (but not store) the $(d-3)$-faces or *subsubfacets*.

The well-known fundamental theorem concerning the facial structure of polytopes follows.

Theorem 3 *A subfacet of a polytope $\mathcal{P}$ is contained in exactly two distinct facets of $\mathcal{P}$.*

Each simplicial facet $\mathcal{F}$ then contains $\binom{d}{d-1} = d$ distinct subfacets, and each subfacet is shared with one *neighboring* facet $\mathcal{G}$.

3 Data Structure

We represent a d-dimensional convex hull by its set of facets $\mathcal{P}$, where each facet is linked by pointers to its d neighbors. In particular, each facet $\mathcal{F} \in \mathcal{P}$ has associated with it the following information.

1. $\text{Vert}^{\mathcal{F}}$; the d extreme points of S that define $\mathcal{F}$, i.e. $\text{Vert}^{\mathcal{F}} = \text{ext}(\mathcal{F})$.
2. $\text{Subf}_i^{\mathcal{F}}$, $i = 1, \ldots, d$; each subfacet defined by a distinct combination of $d-1$ points of $\text{Vert}^{\mathcal{F}}$.
3. $\text{Neig}_i^{\mathcal{F}}$, $i = 1, \ldots, d$; pointers to neighboring facets, such that $\text{Vert}^{\mathcal{F}} \cap \text{Vert}^{\text{Neig}_i^{\mathcal{F}}} = \text{Subf}_i^{\mathcal{F}}$.
4. $\text{Half}^{\mathcal{F}}$; the equation of the halfspace bounded by the hyperplane containing $\mathcal{F}$, and oriented so that $S \subset \text{Half}^{\mathcal{F}}$.

Given $\text{Vert}^{\mathcal{F}}$ we assume that $\text{Half}^{\mathcal{F}}$ costs $O(d^3)$ to create using any of the standard methods. Given a facet $\mathcal{F}$ and a point p, the test $(p \in \text{Half}^{\mathcal{F}})$? then takes $O(d)$ time.

4 Algorithm

The set S is first sorted lexicographically. An initial d-simplex $\mathcal{P}$ is created from $\{p_1, \ldots, p_{d+1}\}$ using the data structure given above.

The algorithm then consists of a simple update step, where each point $p_i, i = d+2, \ldots, n$ is successively added to $\mathcal{P}$. Note that each p_i is external to $\text{conv}(\{p_1, \ldots, p_{i-1}\})$ due to the lexicographic sort. The terminology for the update is motivated by imagining a camera at p_i.

Algorithm UPDATE:

1. Identify the set of *visible facets*,
$$V = \{\mathcal{F} \mid \mathcal{F} \in \mathcal{P},\ p_i \notin \text{Half}^{\mathcal{F}}\}$$
which must be later deleted from $\mathcal{P}$.
2. The set of subfacets shared by V and the set of non-visible facets is called the *visible boundary* B. Create and add to $\mathcal{P}$ the *cap* C of new facets containing p_i,
$$C = \{\mathcal{F} \mid \text{Vert}^{\mathcal{F}} = f \cup \{p_i\},\ f \in B\}$$
Link C to the set of non-visible facets of $\mathcal{P}$ on B.
3. Interlink neighboring facets in C across their common subfacets.
4. Delete V from $\mathcal{P}$.

First, note that at least one visible facet $\mathcal{F}$ can always be provided by the previous update.

Lemma 1 *After the update of $\mathcal{P}$ with p_{i-1} at least one facet $\mathcal{F} \in C$ is visible with respect to p_i.*

Proof: S is lexicographically sorted. Therefore $[p_{i-1}, p_i] \cap \text{conv}(\{p_1, \ldots, p_{i-1}\}) = p_{i-1}$, which implies that p_{i-1} is visible from p_i. Therefore at least one facet $\mathcal{F} \in C$ containing p_{i-1} must also be visible from p_i. □

Let us now examine the four steps of the update in more detail.

4.1 Steps 1 and 2

Steps 1 and 2 are performed together. We identify the visible set V by a simple recursive search of the neighboring facets of the first visible facet $\mathcal{F}$ provided by the previous update. At the same time it will be convenient to create the cap of new facets C and link C to the set of non-visible facets of $\mathcal{P}$ on the visible boundary B.

We start with $V = \mathcal{F}$ and $C = \emptyset$.

Algorithm TRAVERSE ($\mathcal{F}$) recursive:

1. Examine each neighboring facet $\mathcal{G}$ of $\mathcal{F}$. If $\mathcal{G} \notin V$ (that is, $\mathcal{G}$ has not already been visited) then:
   (a) If $p_i \notin \text{Half}^{\mathcal{G}}$, then $\mathcal{G}$ is visible. Put $\mathcal{G}$ in V, and then recursively TRAVERSE ($\mathcal{G}$).

(b) Otherwise, $\mathcal{G}$ is non-visible. The subfacet f shared by $\mathcal{F}$ and $\mathcal{G}$ is in the visible boundary B. Create a new facet $\mathsf{Vert}^{\mathcal{T}} = f \cup \{p_i\}$. Link $\mathcal{T}$ and $\mathcal{G}$ across f. Put $\mathcal{T}$ in C.

At termination of TRAVERSE, V will be identified, C will be created, and C will be linked to $\mathcal{P}$ across B.

4.2 Step 3

The major step remains—interlinking the facets of C with each other. Interestingly, most neighboring facets of the facets of C are themselves in C.

Lemma 2 *Each facet of C has one neighboring facet in $\mathcal{P}$ and $d-1$ neighboring facets in C.*

Proof: Consider any facet $\mathcal{T}$ of C. Only one subfacet of $\mathcal{T}$ does not contain p_i, i.e. subfacet $f = \mathsf{Vert}^{\mathcal{T}} \setminus \{p_i\}$. Now, f cannot be shared with some other facet $\mathcal{W} \in C$, because then $\mathcal{W} = f \cup \{p_i\} = \mathcal{T}$, so f must be shared with a neighboring facet in $\mathcal{P}$.

The other $d-1$ subfacets of $\mathcal{T}$ contain p_i. But only facets of C contain p_i, so the remaining $d-1$ neighboring facets of $\mathcal{T}$ must be in C. □

We interlink the facets of C by walking around a one-dimensional, closed, and unique path of neighboring facets in $\mathcal{P}$ that contain a very particular subsubfacet. First, note that such a path exists.

Theorem 4 *The path through neighboring facets of a d-polytope, all of which contain the same subsubfacet, is one-dimensional, closed, and unique.*

Proof: Consider any facet $\mathcal{F}$ and subsubfacet $z \subset \mathsf{Vert}^{\mathcal{F}}$. Consider $\mathcal{F}$ as a $(d-1)$-polytope. Then z is a *subfacet* of $\mathcal{F}$, and therefore contained in exactly two distinct *facets* f and g of $\mathcal{F}$.

But f and g are really *subfacets* in the d-polytope $\mathcal{P}$, so z is contained in exactly two subfacets of $\mathcal{F}$. Therefore exactly two neighboring facets of $\mathcal{F}$ contain z, and the path is one-dimensional. The path is closed because the number of facets of $\mathcal{P}$ is finite. The path is unique because it is unique from $\mathcal{F}$. □

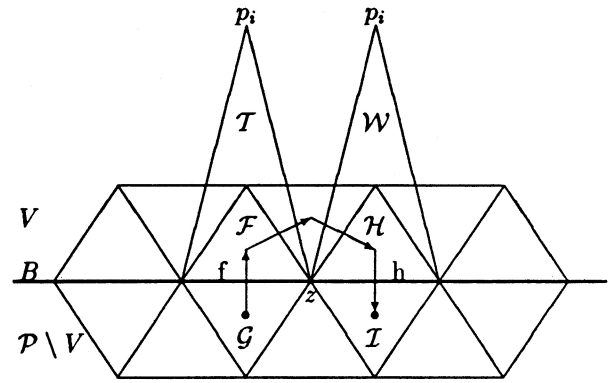
For each $\mathcal{T} \in C$ and null $\mathsf{Neig}_j^{\mathcal{T}}$ we find the neighboring facet $\mathcal{W} \in C$ sharing $\mathsf{Subf}_j^{\mathcal{T}}$ as follows.

Algorithm WALK:

1. First, note that $\mathsf{Vert}^{\mathcal{T}} = f \cup \{p_i\}$, where f is a subfacet of B. Call the visible and non-visible facets sharing f as $\mathcal{F}$ and $\mathcal{G}$ respectively.

   Second, consider the subsubfacet $z = \mathsf{Subf}_j^{\mathcal{T}} \setminus \{p_i\}$. Note that $p_i \notin z$, so $z \subset f$, $z \subset \mathsf{Vert}^{\mathcal{F}}$, and $z \subset \mathsf{Vert}^{\mathcal{G}}$. Therefore, $\mathcal{F}$ and $\mathcal{G}$ are on a one-dimensional closed path around subsubfacet z.

Figure 1: Walking Around a Subsubfacet



   Furthermore, since this path crosses the visible boundary at f it must therefore must cross it again.

2. Walk around z through the visible set V, from visible $\mathcal{F}$ away from non-visible $\mathcal{G}$, and stop when the first non-visible facet $\mathcal{I}$ is encountered. Call the previous (visible) facet on the walk $\mathcal{H}$. Call the subfacet shared by $\mathcal{H}$ and $\mathcal{I}$ as h.

3. Note that h is in B. There is therefore a facet $\mathcal{W} \in C$, such that $\mathsf{Vert}^{\mathcal{W}} = h \cup \{p_i\}$, linked to $\mathcal{I}$ across h. By lemma 3 below, $\mathsf{Subf}_j^{\mathcal{T}} \subset \mathsf{Vert}^{\mathcal{W}}$. Interlink $\mathcal{T}$ and $\mathcal{W}$ across $\mathsf{Subf}_j^{\mathcal{T}}$.

We now show that $\mathcal{W}$ does indeed contain $\mathsf{Subf}_j^{\mathcal{T}}$.

Lemma 3 $\mathsf{Subf}_j^{\mathcal{T}} \subset \mathsf{Vert}^{\mathcal{W}}$.

Proof: $z \subset \mathsf{Vert}^{\mathcal{H}}$ and $z \subset \mathsf{Vert}^{\mathcal{I}}$, so $z \subset h$. Since $\mathcal{W} = h \cup \{p_i\}$, therefore $z \subset \mathsf{Vert}^{\mathcal{W}}$, and since $p_i \in \mathsf{Vert}^{\mathcal{W}}$, therefore $z \cup \{p_i\} \subset \mathsf{Vert}^{\mathcal{W}}$.

But recall that $z = \mathsf{Subf}_j^{\mathcal{T}} \setminus \{p_i\}$, so by re-arrangment $\mathsf{Subf}_j^{\mathcal{T}} = z \cup \{p_i\}$. Therefore $\mathsf{Subf}_j^{\mathcal{T}} \subset \mathsf{Vert}^{\mathcal{W}}$. □

An example of this walk in $\Re^3$ is shown in Figure 1 around subsubfacet (vertex) z, from facets $\mathcal{G}$ and $\mathcal{F}$ to facets $\mathcal{H}$ and $\mathcal{I}$, to find facet $\mathcal{W} \in C$ sharing subfacet $z \cup \{p_i\}$ with facet $\mathcal{T} \in C$.

After completing the interlinking of C, we delete V from $\mathcal{P}$ (step 4), thus completing the update.

5 Complexity

The following analysis shows that the complexity of each update is strictly a function of the size of C.

Each update consists of a traversal step to identify V and create C (steps 1 and 2), and an interlink step to interlink the facets of C (step 3).

There are at most d visits to each facet $\mathcal{F} \in V$ during the traversal step—once from each neighbor. The first visit costs $O(d)$ time to determine $p_i \in \mathsf{Half}^{\mathcal{F}}$, and each subsequent visit is $O(1)$. This one-time $O(d)$ cost per facet of V may be absorbed in the original $O(d^3)$ cost to create each facet of V, because V is deleted at the end of the update.

The traversal step also creates a facet of C during each encounter with the visible boundary B. We require a bound on $|C|$.

Lemma 4 $|C| = O(\Phi(i-1, d-1))$.

Proof: Consider a *vertex figure* $\mathcal{V}$ of p_i, defined by $\mathcal{V} = H \cap \mathcal{P}$, where H is a hyperplane that separates p_i from all other vertices of $\mathcal{P}$. V is a $(d-1)$-polytope, and there is a 1-1 correspondence between:

1. the edges of $\mathcal{P}$ containing p_i, and the vertices of $\mathcal{V}$;
2. the facets of $\mathcal{P}$ containing p_i, and the facets of $\mathcal{V}$.

There are at most $i-1$ edges containing p_i, and therefore at most $i-1$ vertices of $\mathcal{V}$. There are therefore at most $\Phi(i-1, d-1)$ facets of the $(d-1)$-polytope $\mathcal{V}$, and similarly at most $\Phi(i-1, d-1)$ facets of $\mathcal{P}$ containing p_i. □

In other words, the worst-case size of C is the same as the worst-case complexity of the problem in one lower dimension. The traversal step for each update of point p_i then costs $O(d^3 \times \Phi(i-1, d-1))$ to create C.

During the interlink step, facets of V are processed during each walk around a subsubfacet. Using an appropriate data structure, each visit can be done in $O(d)$ time to find the next facet in the path. Each facet in V may be visited at most once for each contained subsubfacet, and a facet of a simplicial d polytope contains $\binom{d}{d-2} = O(d^2)$ subsubfacets. The interlink step then costs $O(d^3 \times |V|)$. This one-time $O(d^3)$ cost per facet of V may be absorbed in the original $O(d^3)$ cost to create each facet of V, because V is deleted at the end of the update.

The complexity of each update of point p_i is therefore strictly a function of the size of C, $O(d^3 \times \Phi(i-1, d-1))$. We can now present the overall complexity.

Theorem 5 *The algorithm is* $O(d^3 \times n \times \Phi(n, d-1))$, *which is optimal for even* d.

Proof: Each update of point p_i costs $O(d^3 \times \Phi(i-1, d-1))$. The total cost is then

$$O(\sum_{i=d+2}^{n} d^3 \times \Phi(i-1, d-1))$$

or, recalling that $\Phi(n, d) = O(n^{\lfloor d/2 \rfloor})$,

$$O(d^3 \times n \times \Phi(n, d-1))$$

This is $d^3 \times n$ times the complexity of the problem in one lower dimension. But when d is even,

$$n \times n^{\lfloor (d-1)/2 \rfloor} = n^{\lfloor d/2 \rfloor}$$

Therefore, the algorithm is optimal when d is even, and exceeds optimality by a factor of n when d is odd. □

6 Conclusions

The algorithm has been implemented and performs well for sets from various distributions. It is expected that the assumption of general position may not be necessary. Similarly, an on-line version may be possible.

It is not yet known how much space is saved by storing only the facets as opposed to the entire facial graph, expressed as a function of d. However, the compact data structure seems to provide an advantage that becomes more pronounced in higher dimensions.

Lemma 4 indicates that any on-line algorithm which updates the convex hull point by point will exceed optimality by a factor of n in odd dimensions in the worst case, since the number of new facets that must be added during each update can be as much as the number of facets in the final hull. For example, this worst case is manifested by points arriving one by one in lexicographic order from a cyclic polytope.

References

[1] Brondsted, A. (1983) *An Introduction to Convex Polytopes*, Springer-Verlag, New York

[2] Chand, D. R.; Kapur, S. S. (1970) *An Algorithm for Convex Polytopes*, Journal of the Association for Computing Machinery, Vol. 17, No. 1, January, pp. 78-86

[3] Graham, R. L. (1972) *An Efficient Algorithm for Determining the Convex Hull of a Finite Planar Set*, Information Processing Letters 1, pp. 132-133

[4] Preparata, F. P.; Hong, S. J. (1977) *Convex Hulls of Finite Sets in Two and Three Dimensions*, Communications of the ACM, Vol. 20, No. 2, February, pp. 87-93

[5] Shamos, M. I.; Preparata, F. P. (1985) *Computational Geometry*, Springer-Verlag, New York, Section 3.4.2, pp. 131-134

[6] Seidel, R. (1981) *A Convex Hull Algorithm Optimal for Point Sets in Even Dimensions*, Tech. Rep. 81-14, Dept. of C.S., Univ. of B.C., British Columbia, Canada

[7] Seidel, R. (1986) *Constructing Higher-Dimensional Convex Hulls at Logarithmic Cost per Face*, Proc. 18th ACM Symp. on Theory of Computing, pp. 404-413, ACM, May