CPS234 Computational Geometry **06 September 2005**

Lecture 3: Course Introduction

Lecturer: Pankaj K. Agarwal *Scribe: Monika Schaeffer*

3.1 Convex Hulls in 2D

3.1.1 Where we left off...

At the end of the previous lecture, we looked at two algorithms for computing the convex hull of a set of points in 2D. The first was the Graham's Scan, which runs in $O(n \log n)$ time. The second was the gift-wrapping algorithm, which runs in $O(nh)$ for output size h. The gift-wrapping algorithm is better if $h < \log n$, intuitively if most of the points lie within the confines of the convex hull.

We want to improve the gift-wrapping algorithm to $O(n \log h)$, which is provably optimal.

3.1.2 The (original) gift-wrapping algorithm

Given a set of points,

> Starting with the bottom-most point and a horizontal line...
> While the convex hull isn't closed off
>> Rotate the line anchored at the point counter-clockwise until you hit another point
>> Add the segment between the current anchor and the new point to the hull
>> Make the new point the anchor.
>
> end while

Note that, given the anchor point and another point p from S, iff p is the correct next point, all remaining points of S are on the same side of the line from the anchor to p.

3.1.3 How we really "rotate the line"

Given q the old anchor, p the new anchor, and $z \in S - \{p, q\}$ arbitrarily picked:

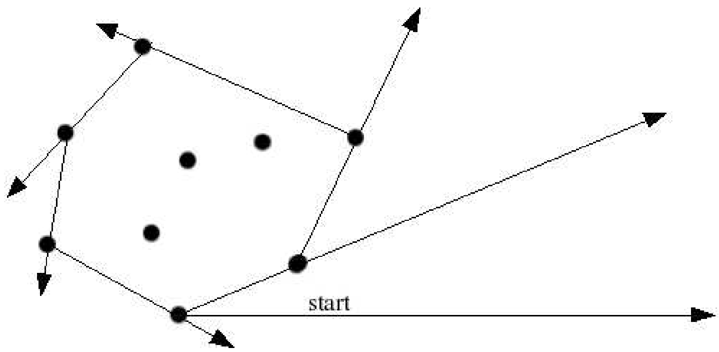

Figure 3.1: giftwrapping in 2D

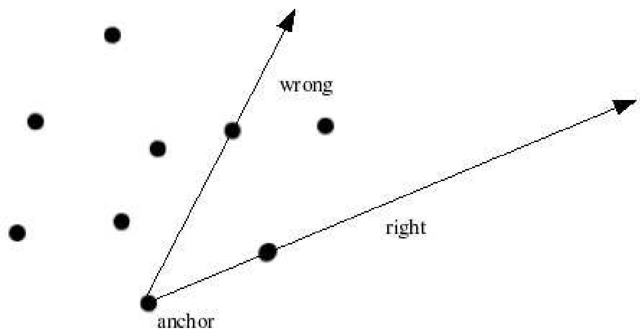

Figure 3.2: all the points in S will be on the same side of the line through the anchor and the correct next point.

$\forall w \in S - \{p, q, z\}$
    if w and q lie on the opposite side of line pz
        z=w
    end if
end $\forall$
return z

That takes $O(n)$ time. We'd like it to take $O(\frac{n}{h} \log n)$ time instead.

3.1.4 O/P-sensitive algorithm

Assume you know h.

Partition S into $\lceil n/h \rceil$ subsets $s_1, s_2 ... s_{\lceil n/h \rceil}$ where $|s_i| \leq h$.

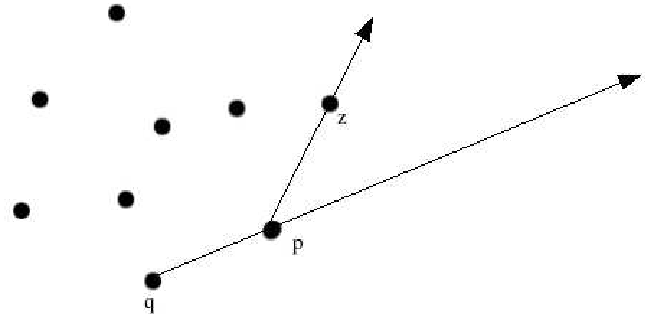

Figure 3.3: p, q, and z

For all i, compute $P_i = Conv(s_i)$ using Graham's Scan.

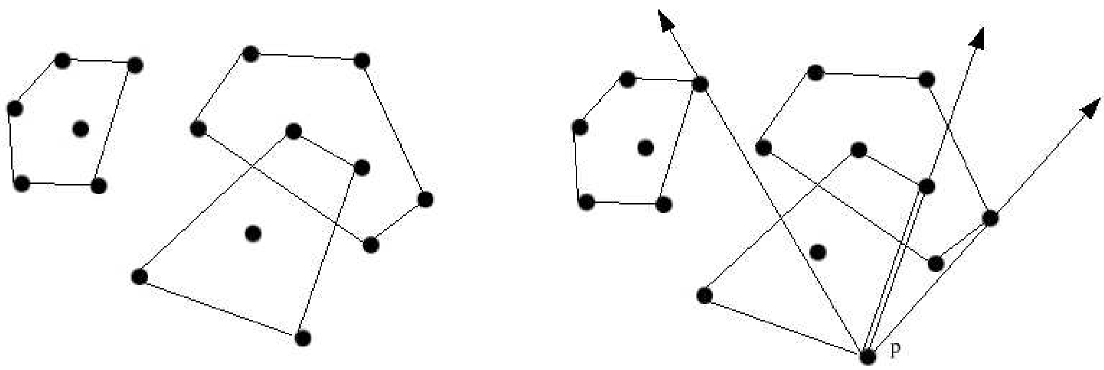

Figure 3.4: Step 1: Divide into subsets and get their convex hulls. Step 2: Calculate tangents to each hull from the anchor p.

Starting with the bottom-most point as your anchor, find the (first, counter-clockwise) tangent lines from that point to each P_i. The points on each hull that correspond to these will be $\{t_1, t_2 ...\}$ (For the hull that contains the anchor, use the next point around the hull.) The next point in $Conv(S)$ is one of these. These t_i can be calculated in $O(\log h)$ time, and there are $\lceil n/h \rceil$ of them.

This changes the algorithm to this (where l is the line from q to p, and q was the previous anchor):

FIRSTPOINT(p, l, S)
    Preprocessing: Find all the t_i
    $\forall w \in \{t_1, ...\}$
        if w and q lie on the opposite side of line pz
            z=w
        end if
    end $\forall$
    return z
end

MAIN
  start with the bottom-most point p_0 and the next one along the convex hull, p
  while $p \neq p_0$ again
    q =FIRSTPOINT(p, l, S)
    $CH = CH \circ q$
    $p = q$
  end while
  return CH
end

Time is $O(n \log h)$.

3.1.5 How do we know h?

guess a small h_1
run the algorithm
if $h_i <$ the real h
  $h_{i+1} = h_i^2$ and repeat

Note that the runtime doubles between $O(n \log h_i)$ and $O(n \log h_{i+1}) = O(n \log(h_i^2)) = O(2{*}n \log h_i)$. For k iterations, you have $n \log h_1 + n \log h_2 + \cdots + n \log h_k =$ a geometric series $= O(n \log h_k) = O(n \log h)$, and $h_i = 2^{2^i}$.

3.2 Convex hulls in higher dimensions

Given S: set of points in $\mathbb{R}^d$ (and using notation h^+ : halfspaces)

$$conv(s) = \bigcap_{h^+ \supseteq S} h^+$$

$$\Updownarrow$$

$$convspan(s) = \{x \in \mathbb{R}^d | x = \sum_{i=1}^{n} \lambda_i p_i, \sum_{i=1}^{n} \lambda_i = 1, 0 \leq \lambda_i \leq 1\}$$

P is the convex hull of S. It is a convex polytope. A hyperplane h supports P if $P \cap h \neq 0$ and $p \subseteq h^+$ (one of the two halfspaces defined by h)

$f = P \cap h$ is a face of P. f is the convex hull of $(S \cap h)$.

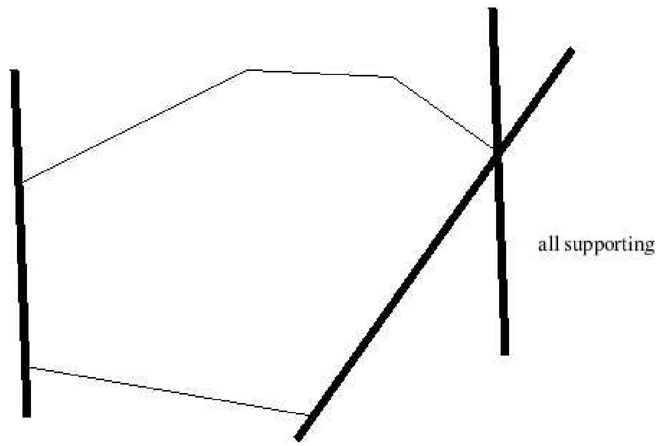

Figure 3.5: supporting hyperplanes in 2D

Faces have dimensions.

vertex: Face of dimension 0
edge: Face of dimension 1

A d-dimension convex hull has faces of dimensions 0 to d-1.

facet: Face of dimension d-1
ridge: Face of dimension d-2

Each ridge connects two facets.

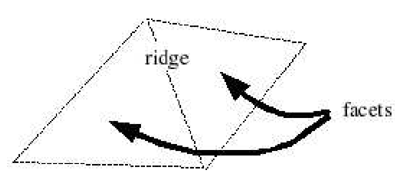

Figure 3.6: two facets and a ridge of a convex hull in 3D

3.2.1 properties of faces

$h, g \subseteq h$ are faces of $P \Rightarrow g$ is a face of h.

if g, h are faces of P, $g \cap h$ is also a face of P.

(For consistency and so we don't need lots of special cases, let's let $\emptyset$ be a face of everything, with dimension -1, and P be a d-dimension face of P)

3.2.2 How to represent

Define a face-graph of $P : F(P)$ with the nodes being all the faces.

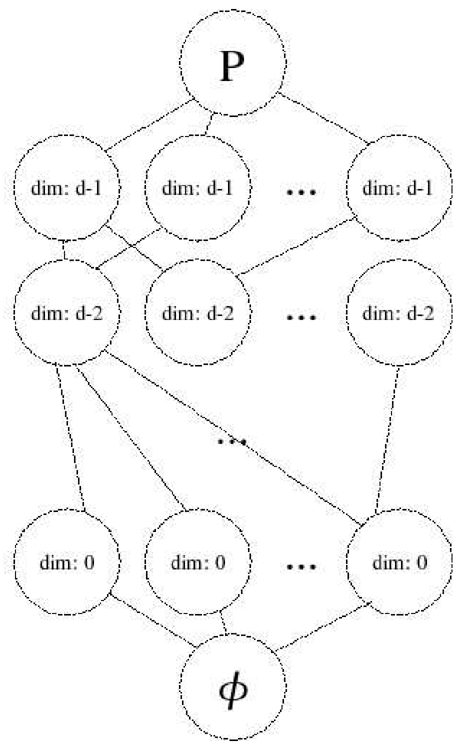

Figure 3.7: A face-graph F(P). Each level contains all the faces of that dimension.

A face is defined by its verticies.

3.2.3 two algorithms for computing a convex hull

but first! Upper bound Thm:

A convex polytope with n verticies has $O(n^{\lfloor d/2 \rfloor})$ faces. For $d = 2, 3$, linear. For $d = 4, 5$, quadratic! This is a tight bound, with examples (such as the cyclic polytope).

3.2.4 Higher dimension gift-wrapping

For a ridge, there are 2 facets.

Given a ridge and a facet attatched to it, find the other facet.

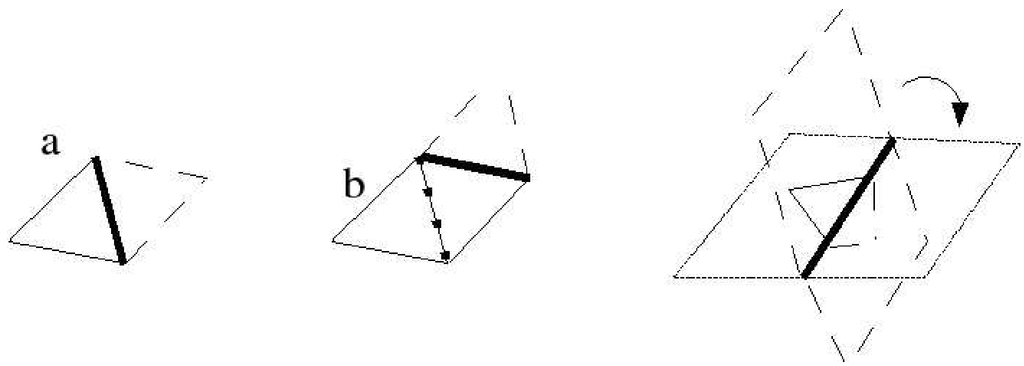

Figure 3.8: Gift-wrapping in 3D: In a, you have a found facet and a ridge to search from. In b, that ridge's other facet's been found, and now you search from a new ridge. New facets are found by rotating hyperplanes anchored at ridges.

Each ridge is a convex hull of d-1 points, and each facet is a convex hull of d points.

3.2.5 d-simplexes

A d-simplex is a convex hull of d+1 affinely independent points. A 0-simplex is a point. A 1-simplex is a line segment. A 2-simplex is a triangle. A 3-simplex is tetrahedron. Faces of simplexes are simplexes.

If the points in S are affinely independent, every ridge is a d-2-simplex, and every facet is a d-1-simplex.

3.2.6 (back to) Higher dimension gift-wrapping

Ridges can be represented as $(p_1, ..., p_{d-1}) = g$ and facets adjacent to them as (g, c).

To calculate the convex hull, then,

$Q = \emptyset$ =queue of open ridges
f: find some facet of conv(S)
$f = (p_1, ...p_d)$
for $i = 1$ to d
    insert $((f - \{p_i\}), p_i)$ into Q
while Q not empty
    (g, c) =delete(Q)
    $(g, b) = find - other - facet((g, c), S)$
    for all other ridges on the new facet
        if they're in Q, delete them
        else, insert into Q
end

This runs in $O(nh)$ time, and can be fiddled with to run in $O(h^2 + h \log n)$ time.