Computational Geometry
2D Convex Hulls

Joseph S. B. Mitchell
Stony Brook University

Chapter 2: Devadoss-O’Rourke
Convexity

- Set $X$ is **convex** if $p, q \in X \Rightarrow pq \subseteq X$
- Point $p \in X$ is an **extreme point** if there exists a line (hyperplane) through $p$ such that all other points of $X$ lie strictly to one side of the line.
Convex Hull

**Definition.** The *convex hull* of $S$, denoted by $\text{conv}(S)$, is the intersection of all convex regions that contain $S$.

**Exercise 2.1.** Show that the use of the word “convex” in convex hull is justified; that is, show that $\text{conv}(S)$ is indeed a convex region.

Figure 2.1: A point set $S$ along with (a) a nonconvex region enclosing $S$, (b) a convex region enclosing $S$, and (c) the convex hull of $S$. 
Convex Hull, Extreme Points

Figure 2.6: A point set along with hull points and extreme points.
Convex Hull: Equivalent Def

- **Convex combination of points:**
  \[ \lambda_1 p_1 + \cdots + \lambda_n p_n \]
  where \( \lambda_i \geq 0 \) and \( \sum \lambda_i = 1 \)

**Theorem 2.2.** For a point set \( S = \{p_1, \ldots, p_n\} \), the convex hull of \( S \) is the set of all convex combinations of \( S \).

**Proof.** Let \( M \) be the set of convex combinations of \( S \). Formally,

\[ M = \left\{ \lambda_1 p_1 + \cdots + \lambda_n p_n \mid \lambda_i \geq 0, \quad \sum \lambda_i = 1 \right\}. \]

In order to prove \( \text{conv}(S) = M \), we show \( \text{conv}(S) \subseteq M \) and \( M \subseteq \text{conv}(S) \).
Convex Hull: Equivalent Defs

Exercise 2.5. Show that \( \text{conv}(S) \) is the convex polygon with the smallest perimeter that contains \( S \).

Exercise 2.6. Show that \( \text{conv}(S) \) is the convex polygon with the smallest area containing \( S \).
Equivalent Definitions of Convex Hull, CH(\(X\))

- \{all convex combinations of \(d+1\) points of \(X\}\}  
  \[Caratheodory's \ Thm\]  
  \(\text{(in any dimension } d\text{)}\)

- \(\bigcap_{T \supseteq X, T \text{ convex}} T\)  
  \(\text{Devadoss-O'Rourke Def}\)

- \(\bigcap_{H \supseteq X, H \text{ halfspace}} H\)

- Set-theoretic “smallest” convex set containing \(X\).

- In 2D: min-area (or min-perimeter) enclosing convex body containing \(X\)

- In 2D: \(\bigcup_{\Delta \text{abc} \text{ with } a,b,c \in X} \Delta\abc\)
Convex Hull in 2D

- **Fact**: If $X=S$ is a finite set of points in 2D, then $CH(X)$ is a convex polygon whose vertices (extreme points) are points of $S$. 
Fundamental Algorithmic Problem: 2D Convex Hulls

- **Input**: $n$ points $S = (p_1, p_2, \ldots, p_n)$

- **Output**: A boundary representation, e.g., ordered list of vertices (extreme points), of the convex hull, $CH(S)$, of $S$ (convex polygon)

Output: $(9, 6, 4, 2, 7, 8, 5)$

More generally: $CH($polygons$)$
Convex Hull Problem in 2D

Figure 2.2: The left side lists 18 points, with the hull points colored red. The right side shows the plot of these points in the plane, again with the hull points marked.
Function $T(n)$ is $O(f(n))$ if there exists a constant $C$ such that, for sufficiently large $n$, $T(n) < C f(n)$

<table>
<thead>
<tr>
<th>Order</th>
<th>Name</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O(1)$</td>
<td>Constant</td>
<td>Adding or multiplying two numbers (of constant size)</td>
</tr>
<tr>
<td>$O(\log n)$</td>
<td>Logarithmic</td>
<td>Finding an item in a sorted list by binary search</td>
</tr>
<tr>
<td>$O(n)$</td>
<td>Linear</td>
<td>Finding an item in an unordered list</td>
</tr>
<tr>
<td>$O(n \log n)$</td>
<td>“$n \log n$”</td>
<td>Sorting a list</td>
</tr>
<tr>
<td>$O(n^2)$</td>
<td>Quadratic</td>
<td>Incremental convex hull algorithm</td>
</tr>
<tr>
<td>$O(n^k)$</td>
<td>Polynomial</td>
<td>Motion planning for a $(k-1)$-DOF robot arm</td>
</tr>
<tr>
<td>$O(e^n)$</td>
<td>Exponential</td>
<td>SET PARTITION via the brute-force algorithm</td>
</tr>
</tbody>
</table>
Comparing \( O(n), O(n \log n), O(n^2) \)

Function \( T(n) \) is \( O(f(n)) \) if there exists a constant \( C \) such that, for sufficiently large \( n \), \( T(n) < C f(n) \)

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n \log n )</th>
<th>( n^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( 2^{10} \approx 10^3 )</td>
<td>( 10 \cdot 2^{10} \approx 10^4 )</td>
<td>( 2^{20} \approx 10^6 )</td>
</tr>
<tr>
<td>( 2^{20} \approx 10^6 )</td>
<td>( 20 \cdot 2^{20} \approx 2 \cdot 10^7 )</td>
<td>( 2^{40} \approx 10^{12} )</td>
</tr>
</tbody>
</table>

Interactive Processing

<table>
<thead>
<tr>
<th>( n )</th>
<th>( n = 1000 )</th>
<th>( n = 1000000 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n \log n ) algorithms</td>
<td>yes</td>
<td>?</td>
</tr>
<tr>
<td>( n^2 ) algorithms</td>
<td>?</td>
<td>no</td>
</tr>
</tbody>
</table>
2D Convex Hull Algorithms

- $O(n^4)$ simple, brute force (but finite!)
- $O(n^3)$ still simple, brute force
- $O(n^2)$ incremental algorithm
- $O(nh)$ simple, “output-sensitive”
  - $h$ = output size (# vertices)
- $O(n \log n)$ worst-case optimal (as fcn of $n$)
- $O(n \log h)$ “ultimate” time bound (as fcn of $n,h$)
- Randomized, expected $O(n \log n)$

[DO] Section 2.2
Lower Bound

- Lower bound: $\Omega(n \log n)$

From SORTING:

As function of $n$ and $h$: $\Omega(n \log h)$ holds

Note: Even if the output of CH is not required to be an ordered list of vertices (e.g., just the # of vertices), $\Omega(n \log n)$ holds
Figure 2.9: Giving height $x_i^2$ to points $x_i$ in a list, landing on a parabola $y = x^2$.

**Theorem 2.26.** Let $S$ be a point set in the plane. An algorithm that finds the hull points of $S$ in the order of walking around the convex hull cannot be faster than $O(n \log n)$ time. That is, $\Omega(n \log n)$ is a lower bound on computing the ordered convex hull of $n$ points in the plane.
Theorem 2.26. Let $S$ be a point set in the plane. An algorithm that finds the hull points of $S$ in the order of walking around the convex hull cannot be faster than $O(n \log n)$ time. That is, $\Omega(n \log n)$ is a lower bound on computing the ordered convex hull of $n$ points in the plane.

An even stronger statement is true:

Theorem 2.27. A lower bound for any algorithm that identifies the hull points of a point set in the plane is $\Omega(n \log n)$. 
Primitive Computation

- “Left” tests: sign of a cross product (determinant), which determines the orientation of 3 points

- Time $O(1)$ (“constant”)

$$Left(a, b, c) = \text{TRUE} \iff ab \times ac > 0$$

$c$ is left of $ab$
SuperStupidCH: $O(n^4)$

- **Fact**: If $s \in \Delta pqr$, then $s$ is not a vertex of $\text{CH}(S)$
  - $\forall p$
    - $\forall q \neq p$
      - $\forall r \neq p,q$
        - $\forall s \neq p,q,r$: If $s \in \Delta pqr$ then mark $s$ as NON-vertex
  - $O(n^3)$

- **Output**: Vertices of $\text{CH}(S)$
- Can sort ($O(n \log n)$) to get ordered
StupidCH: $O(n^3)$

- **Fact**: If all points of $S$ lie strictly to one side of the line $pq$ or lie in between $p$ and $q$, then $pq$ is an edge of $\text{CH}(S)$.

- $\forall p$
  - $\forall q \neq p$
    - $\forall r \neq p, q$: If $r \in \text{red}$ then mark $pq$ as NON-edge (ccw)

- **Output**: Edges of $\text{CH}(S)$

- Can sort ($O(n \log n)$) to get ordered

Caution!! Numerical errors require care to avoid crash/infinite loop!
Incremental Algorithm

Convex Hull Algorithm $O(n^2)$

Sort the points of $S$ according to their $x$-coordinate. The first three of these points determine a triangle, our starting hull. Consider the next point in the ordered set $S$, add it to the hull, and remove the enclosed non-hull points. Continue this process of adding one point of $S$ at a time until all of $S$ has been processed.

Exercise 2.15. In the incremental algorithm, find a method to search for the tangent lines that leads, overall, to a time complexity of $O(n)$ rather than $O(n^2)$. Notice that this improves the speed of the algorithm to $O(n \log n)$. 
Incremental Algorithm

- Sort points by x-coordinate $O(n \log n)$
- Build $CH(X)$, adding pts left to right

Figure 2.3: The incremental algorithm in action.
**Incremental Algorithm**

**Definition.** Let $P$ be a convex polygon and $x$ a point on the boundary of $P$. Then a line $L$ containing $x$ supports $P$ at $x$ if all of $P$ lies on one side of $L$. Line $L$ is then called a tangent to $P$ at $x$.

Figure 2.4: Convex hull of $k$ points and the incremental addition of another point.
Figure 2.5: tangent line lines transitioning from grouping and separating the convex hull from the new point $p$. 
$O(nh)$: Gift-Wrapping

- **Idea**: Use one edge to help find the next edge.

- **Output**: Vertices of $\text{CH}(S)$

- **Demo applet** of Jarvis march

Key observation: Output-sensitive!
Gift-Wrapping

[DO] Section 2.4

Figure 2.7: The gift-wrapping algorithm in action.
**Gift-Wrapping**

**GiftWrapping** \( Convex \) Hull Algorithm \( O(nh) \)

Start with a known point on the hull as an anchor, such as the bottommost point. Comparing angles to all other points from this anchor, choose the point with the largest angle. Repeat this process, moving around the hull, analogous to the process of winding a string around the point set.

**Exercise 2.19.** Describe a point set with \( n \) points that serves as the worst-case for the gift-wrapping algorithm.

**Exercise 2.20.** Describe a point set with \( n \) points that constitutes the best-case for the gift-wrapping algorithm. What is its time complexity in this case?
**O(n log n): Graham Scan**

- **Idea:** Sorting helps!
- Start with $v_{\text{lowest}}$ (min-$y$), a known vertex \(O(n)\)
- Sort $S$ by angle about $v_{\text{lowest}}$ \(O(n \log n)\)
- **Graham scan:**
  - Maintain a stack representing (left-turning) CH so far \(O(n)\)
  - If $p_i$ is left of last edge of stack, then PUSH
  - Else, POP stack until it is left, charging work to popped points

**Demo applet**
Graham Scan

Red points form right turn, so are discarded.
Graham Scan

Convex Hull Algorithm $O(n \log n)$

Choosing the bottommost point as an anchor, order all other points based on the angles they form about this anchor. Construct the hull by following this ordering, adding points for left hull turns and deleting for right turns.
Graham Scan

Sorted points:

<table>
<thead>
<tr>
<th>vnum</th>
<th>(x, y)</th>
<th>delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>(3, -2)</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>(5, 1)</td>
<td>x</td>
</tr>
<tr>
<td>18</td>
<td>(7, 4)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>(6, 5)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>(4, 2)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>(3, 3)</td>
<td>x</td>
</tr>
<tr>
<td>1</td>
<td>(3, 5)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>(2, 5)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>(0, 5)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>(0, 1)</td>
<td>x</td>
</tr>
<tr>
<td>7</td>
<td>(-3, 4)</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>(-2, 2)</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>(0, 0)</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>(-3, 2)</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>(-5, 2)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>(-5, 1)</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>(-5, -1)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>(1, -2)</td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>(-3, -2)</td>
<td></td>
</tr>
</tbody>
</table>

[O'Rourke, Chapter 3]
tStack Graham()
{
    tStack top;
    int i;
    tPoint p1, p2; /* Top two points on stack. */

    /* Initialize stack. */
    top = NULL;
    top = Push ( &P[0], top );
    top = Push ( &P[1], top );

    /* Bottom two elements will never be removed. */
    i = 2;

    while ( i < n ) {
        p1 = top->next->p;
        p2 = top->p;
        if ( Left( p1->v, p2->v, P[i].v ) ) {
            top = Push ( &P[i], top );
            i++;
        } else
            top = Pop( top );
    }
    return top;
}
Stack History

i = 2: 18, 15
i = 3: 6, 18, 15
i = 4: 12, 6, 18, 15
i = 4: 6, 18, 15
i = 5: 1, 6, 18, 15
i = 5: 6, 18, 15
i = 6: 3, 6, 18, 15
i = 6: 6, 18, 15
i = 7: 16, 6, 18, 15
i = 8: 7, 16, 6, 18, 15
i = 9: 4, 7, 16, 6, 18, 15
i = 9: 7, 16, 6, 18, 15
i = 10: 5, 7, 16, 6, 18, 15
i = 10: 7, 16, 6, 18, 15
i = 11: 8, 7, 16, 6, 18, 15
i = 12: 14, 8, 7, 16, 6, 18, 15
i = 12: 8, 7, 16, 6, 18, 15
i = 13: 9, 8, 7, 16, 6, 18, 15
i = 14: 11, 9, 8, 7, 16, 6, 18, 15
$O(n \log n)$: Divide and Conquer

- Split $S$ into $S_{left}$ and $S_{right}$, sizes $n/2$
- Recursively compute CH($S_{left}$), CH($S_{right}$)
- Merge the two hulls by finding upper/lower bridges in $O(n)$, by “wobbly stick”

Time:

$T(n) \leq 2T(n/2) + O(n) \Rightarrow T(n) = O(n \log n)$
Divide and Conquer

[DO] Section 2.6

Figure 2.10: The divide-and-conquer algorithm in action.
Figure 2.11: (a) Finding two tangents and (b) constructing the convex hull.

Figure 2.12: Alternately walking between A and B, looking for the lower tangent.
DIVIDE-AND-CONQUER       Convex Hull Algorithm  $O(n \log n)$

Sort the points of $S$ by $x$-coordinate. Divide the points into two (nearly) equal groups. Compute the convex hull of each group (recursively using divide and conquer). Merge the two groups together with upper and lower supporting tangents to get the hull of $S$. 

[DO] Section 2.6
QuickHull

- QuickHull\((a,b,S)\)
  
  to compute upperhull\((S)\)
  
  - If \(S = \emptyset\), return ()
  
  - Else return (QuickHull\((a,c,A)\), \(c\), QuickHull\((c,b,B)\))

\(c = \text{point furthest from } ab\)

Discard points in \(\Delta abc\)

Worst-case: \(O(n^2)\)
Avg: \(O(n)\)

Works well in higher dimensions too!

Qhull website
QuickHull

- **Applet** (programmed by Jeff So)
- **Applet** (by Lambert)

- **When is it “bad”? (see hw2)**
  - If no points are discarded (in \( \triangle abc \)), at each iteration, and
  - The “balance” may be very lop-sided (between sets A and B; in fact, one set could be empty, at every iteration!)
QuickHull

Fig. 14: QuickHull’s initial quadrilateral.

Fig. 15: QuickHull elimination procedure.

\[ T(n) = \begin{cases} 
\frac{1}{T(n_1) + T(n_2)} & \text{if } n = 1 \\
T(n_1) + T(n_2) & \text{where } n_1 + n_2 \leq n.
\end{cases} \]

[David Mount lecture notes, Lecture 3]
$O(n \log n)$: Incremental Construction

- Add points in the order given: $v_1, v_2, \ldots, v_n$
- Maintain current $Q_i = CH(v_1, v_2, \ldots, v_i)$
  - If $v_{i+1} \in Q_i$, do nothing
  - Else insert $v_{i+1}$ by finding tangencies, updating the hull
- Worst-case cost of insertion: $O(\log n)$
- But, uses complex data structures
$O(n \log n)$: Randomized Incremental

- Add points in random order
- Keep current $Q_i = CH(v_1, v_2, \ldots, v_i)$
  - Else insert $v_{i+1}$ by finding tangencies, updating the hull

Expected cost of insertion: $O(\log n)$
Each uninserted \( v_j \not\in Q_i \) (\( j > i + 1 \)) points to the bucket (cone) containing it; each cone points to the list of uninserted \( v_j \not\in Q_i \) within it (with its conflict edge).

Add \( v_{i+1} \not\in Q_i \):
- Start from “conflict” edge \( e \), and walk cw/ccw to establish new tangencies from \( v_{i+1} \), charging walk to deleted vertices
- Rebucket points in affected cones (update lists for each bucket)
- Total work: \( O(n) + \) rebucketing work
- \( E(\text{rebucket cost for } v_j \text{ at step } i) = O(1) \times P(\text{rebucket}) \leq O(1) \times (2/i) = O(1/i) \)
- \( E(\text{total rebucket cost for } v_j \text{ or } v_j) = \sum O(1/i) = O(\log n) \)
- Total expected work = \( O(n \log n) \)

Backwards Analysis:
\( v_j \) was just rebucketed iff the last point inserted was one of the 2 endpoints of the (current) conflict edge, \( e \), for \( v_j \)
Output-Sensitive: $O(n \log h)$

- The “ultimate” convex hull (in 2D)
  - “Marriage before Conquest” : $O(n \log h)$
  - Lower bound $\Omega(n \log h)$ [Kirkpatrick & Seidel’86]

- Simpler [Chan’95]

- 2 Ideas:
  - Break point set $S$ into groups of appropriate size $m$ (ideally, $m = h$)
  - Search for the “right” value of $m$ ($= h$, which we do not know in advance) by repeated squaring

AMS 545 / CSE 555
Chan's Algorithm

- **Break $S$ into $n/m$ groups, each of size $m$**
  - Find CH of each group (using, e.g., Graham scan):
    - $O(m \log m)$ per group, so total $O((n/m) m \log m) = O(n \log m) = O(n \log h)$
  - **Gift-wrap the $n/m$ hulls to get overall CH:**
    - At each gift-wrap step, when pivoting around vertex $v$
      - find the tangency point (binary search, $O(\log m)$) to each group CH
      - pick the smallest angle among the $n/m$ tangencies:
        - $O(h (n/m) \log m) = O(h (n/h) \log h) = O(n \log h)$
  - **Hope:** $m=h$
  - **Try** $m = 4, 16, 256, 65536,...$

$O(n ( \log (2^{2^1}) + \log (2^{2^2}) + \log (2^{2^3}) + \ldots \log (2^{2^{\log (\log h)}) )
= O(n (2^1 + 2^2 + 2^3 + \ldots + 2^{\log (\log h)}) )
= O(n (2 \log h)) = O(n \log h)$
CH of Simple Chains/Polygons

- **Input**: ordered list of points that form a simple chain/cycle
- Importance of **simplicity** (noncrossing, Jordan curve) as a means of "sorting" (vs. sorting in $x, y$)
- **Melkman's Algorithm**: $O(n)$ (vs. $O(n \log n)$ or $O(n \log h)$)
Melkman's Algorithm

Keep hull in DEQUE: \( < v_b, v_{b+1}, ..., v_{t-1}, v_t = v_b > \)

While \( \text{Left}(v_b, v_{b+1}, v_i) \) and \( \text{Left}(v_{t-1}, v_t, v_i) \)

\[ i \leftarrow i + 1 \]

Remove \( v_{b+1}, v_{b+2}, ..., v_{t-1}, v_t \)

and \( v_{t-1}, v_{t-2}, ..., v_i \)

until convexity restored

New \( v_b = v_t = v_i \)

**Claim:** Simplicity assures that the only way for the chain to exit the current hull is via pockets \( v_b v_{b+1} \) or \( v_{t-1} v_t \).

**Time:** \( O(n) \), since \( O(1) \) per insertion
Melkman’s Algorithm

Algorithm:

(0) Initialize: If \( \text{Left}(v_0, v_1, v_2) \), then \( D \leftarrow \langle v_2, v_0, v_1, v_2 \rangle \); else, \( D \leftarrow \langle v_2, v_1, v_0, v_2 \rangle \).
   \[ i \leftarrow 3 \]

(1) While (\( \text{Left}(d_{t-1}, d_t, v_i) \) and \( \text{Left}(d_b, d_{b+1}, v_i) \)) do
   \[ i \leftarrow i + 1 \]
   (We now have a point \( v_i \) that is not in the convex cone defined by the two hull edges \( d_b d_{b+1} \) and \( d_t d_{t-1} \) (recall that \( d_b = d_t \)).)

(2) Restore convexity:
   Until \( \text{Left}(d_{t-1}, d_t, v_i) \), do pop \( d_t \). Push \( v_i \).
   Until \( \text{Left}(v_i, d_b, d_{b+1}) \), do remove \( d_b \). Insert \( v_i \).
   \[ i \leftarrow i + 1 \]
   Go to (1).
Melkman's Algorithm
Melkman's Algorithm
Exercise 2.10. Prove that if $S$ is the set of $n$ points sampled from a uniform distribution in a unit square, then the expected number of points on the hull of $S$ is of order $O(\log n)$. 
Expected Size, h, of CH

- **Claim**: For n points uniform in a square, the expected number of maximal points is $O(\log n)$

- **Corollary**: $E(h) = O(\log n)$ (since CH vertices are subset of maximal points)

- Thus, Gift-Wrap/Jarvis runs in expected $O(n \log n)$, if points are uniform in a box, etc.
**Fact:** The number of $j$-dimensional faces on a $d$-simplex is equal to the number $(j + 1)$-element subsets of domain of size $d + 1$, that is,

\[
\binom{d + 1}{j + 1} = \frac{(d + 1)!}{(j + 1)!(d - j)!}.
\]
Representing the structure of a simplex or polytope: Incidence Graph

Later: for planar subdivisions (e.g., boundary of 3-dim polyhedra), we have various data structures: winged-edge, doubly-connected edge list (DCEL), quad-edge, etc.

[David Mount lecture notes, Lecture 5]
Convex Hull in 3D

Figure 2.13: The convex hull of 758 random points on the surface of a sphere.
Convex Hull in 3D

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>2D Complexity</th>
<th>3D Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Gift wrapping</td>
<td>$O(nh)$</td>
<td>$O(nf)$</td>
</tr>
<tr>
<td>Divide-and-conquer</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>Graham scan</td>
<td>$O(n \log n)$</td>
<td>?</td>
</tr>
</tbody>
</table>

**Unsolved Problem 10**

3D Graham Scan

Find a natural counterpart for the Graham scan algorithm in 3D.
CH in Higher Dimensions

- **3D: Divide and conquer:**
  - $T(n) \leq 2T(n/2) + O(n)$
  - $O(n \log n)$
  - Output-sensitive: $O(n \log h)$ [Chan]

- **Higher dimensions: ($d \geq 4$)**
  - $O(n \lfloor d/2 \rfloor)$, which is worst-case OPT, since point sets exist with $h=\Omega(n \lfloor d/2 \rfloor)$
  - Output-sensitive: $O((n+h) \log^{d-2} h)$, for $d=4,5$
Convex Hull in 3D

Figure 2.14: (a) The hull $Q$ of 100 points and (b) the hull of $Q \cup p$ along with the shadow boundary marked.
Convex Hull in 3D

Figure 2.15: Two hulls $A$ and $B$ along with the hull of $A \cup B$. The shadow boundaries are marked.

Figure 2.16: Possible shadow boundary edges of polyhedra.
More Demos

- Various 2D and 3D algorithms in an applet
- Applets of algorithms in O’Rourke’s book
- Applets from Jack Snoeyink
Review: Convex Hull Algorithms

- $O(n^4), O(n^3)$, naïve
- $O(n^2)$ incremental algorithm
- $O(nh)$ simple, “output-sensitive”
- $O(n \log n)$ Graham scan, divide-and-conquer, incremental (with fancy data structures); worst-case optimal (as fcn of $n$)
- $O(n \log h)$ “ultimate” time bound (as fcn of $n,h$)
- Randomized, expected $O(n \log n)$ (simple)
- 3D: Divide and conquer: $O(n \log n)$
  - Output-sensitive: $O(n \log h)$ [Chan]
- Higher dimensions: ($d \geq 4$)
  - $O(n^{\lfloor d/2 \rfloor})$, which is worst-case OPT,
  - Output-sensitive: $O((n+h) \log^{d-2} h)$, for $d=4,5$