Convex Hulls in Two Dimensions
Definitions

FIGURE 3.1 Any dent implies nonconvexity.
Nonextreme points

1. Let \( S = \{p_0, p_1, \ldots, p_{n-1}\} \), with all points distinct.
2. The in-triangle test can be implemented with three LeftOns.

**Algorithm:** INTERIOR POINTS

for each \( i \) do
  for each \( j \neq i \) do
    for each \( k \neq i \neq j \) do
      for each \( l \neq i \neq j \neq k \) do
        if \( p_l \in \triangle(p_i, p_j, p_k) \) then \( p_l \) is nonextreme

**Algorithm 3.1** Interior points.
Algorithm: Extreme Edges
for each $i$ do
  for each $j \neq i$ do
    for each $k \neq i \neq j$ do
      if $p_k$ is not left or on $(p_i, p_j)$
      then $(p_i, p_j)$ is not extreme

Algorithm 3.2  Extreme edges.
FIGURE 3.2  The next edge $e$ makes the smallest angle $\theta$ with respect to the previous edge.
**Algorithm:** GIFT WRAPPING

Find the lowest point (smallest y coordinate).
Let $i_0$ be its index, and set $i \leftarrow i_0$.

repeat
    for each $j \neq i$ do
        Compute counterclockwise angle $\theta$ from previous hull edge.
        Let $k$ be the index of the point with the smallest $\theta$.
        Output $(p_i, p_k)$ as a hull edge.
        $i \leftarrow k$
    until $i = i_0$

**Algorithm 3.3** Gift wrapping.
FIGURE 3.3 QuickHull discards the points in $\triangle abc$ (shaded) and recurses on $A$ and $B$. Here $A = \emptyset$ and $|B| = 2$. 
Algorithm: QUICKHULL

function QuickHull(a, b, S)
    if S = ∅ then return ()
    else
        c ← index of point with max distance from ab.
        A ← points strictly right of (a, c).
        B ← points strictly right of (c, b).
        return QuickHull(a, c, A) + (c) + QuickHull(c, b, B)

Algorithm 3.4 QuickHull.
Details: Boundary Conditions

§ A number of details have been ignored in our presentation so far.

1. “boundary” conditions
   – if a & b are not on the hull?

2. Implementation issues
   – if points are collinear or coincident?
FIGURE 3.6  Sorting points with collinearities. Indices indicate sorting rank. Points to be deleted are shown as open circles.
Algorithm: Graham Scan, Version B
Find rightmost lowest point; label it $p_0$.
Sort all other points angularly about $p_0$.
   In case of tie, delete the point closer to $p_0$
   (or all but one copy for multiple points).
Stack $S = (p_1, p_0) = (p_t, p_{t-1})$; $t$ indexes top.
i = 2
while $i < n$ do
   if $p_i$ is strictly left of $p_{t-1} p_t$
      then Push($p_i, S$) and set $i \leftarrow i + 1$
   else Pop($S$).

Algorithm 3.6    Graham Scan, Version B.
Implementation of Graham’s Algorithm

Sorting

1. Atan2

- The obvious choice is to define \( p_i < p_j \) if \( \text{angle}(r_i) < \text{angle}(r_j) \), where \( \text{angle}(r) \) is the counterclockwise angle of \( r \) from the positive \( x \) axis (see Figure 3.7)
- C provides the desired function:
  \[
  \text{angle}(r) = \text{atan2}( r[Y], r[X] )
  \]
- Two reasons not to use this
  - No guarantee that the arctangent computation is itself accurate
  - Expensive function. Slopes are simpler
Implementation of Graham’s Algorithm

FIGURE 3.7 Notation for sorting angle.
Implementation of Graham’s Algorithm

10 Sorting

1. Slopes
   - In the first quadrant, the slope $r[Y]/r[X]$ can substitute for the arctangent
   - In the second quadrant, $-r[X]/r[Y]$,
   - Several weaknesses
   - If $r_j = cr_j$, where $c$ is some positive number, no guarantee that $\text{angle}(r_i) = \text{angle}(r_j)$
   - Problem of Floating-point division in C
   - Machine-dependent
   - We opt for integer computation
Implementation of Graham’s Algorithm

**Sorting**

1. **Left**
   - Integer computations to compare $r_i$ and $r_j$
   - Recall that Left was itself a simple test on the value of Area2, which computes the signed area of the triangle determined by three points
   - Use this area function to distinguish ties
Implementation of Graham’s Algorithm

```c
int Compare( const void *tpi, const void *tpj )
{
    int a;            /* area */
    int x, y;         /* projs. of ri & rj in 1st quadrant */
    tPoint pi, pj;
    pi = (tPoint)tpi;
    pj = (tPoint)tpj;

    a = Area2( P[0].v, pi->v, pj->v );
    if (a > 0)
        return -1;
    else if (a < 0)
        return 1;
    else { /* Collinear with P[0] */
        x = abs( pi->v[X] - P[0].v[X] ) - abs( pj->v[X] - P[0].v[X] );
        y = abs( pi->v[Y] - P[0].v[Y] ) - abs( pj->v[Y] - P[0].v[Y] );
    }
}
```
if ( (x < 0) || (y < 0) ) {
    pi->delete = TRUE;
    return -1;
}
else if ( (x > 0) || (y > 0) ) {
    pj->delete = TRUE;
    return 1;
}
else { /* points are coincident */
    if (pi->vnum > pj->vnum)
        pj->delete = TRUE;
    else
        pi->delete = TRUE;
    return 0;
}
ndelete++;
main()
{
    tStack    top;

    n = ReadPoints();
    FindLowest();
    qsort(
        &P[1],    /* pointer to 1st elem */
        n-1,      /* number of elems */
        sizeof( tsPoint ),  /* size of each elem */
        Compare    /* -1,0,+1 compare function */
    );
    Squash();

    top = Graham();
    PrintStack( top );
}

Code 3.6  main.
Implementation of Graham’s Algorithm

FIGURE 3.8 Graham Scan for Figure 3.6. Indices correspond to the coordinates in Table 3.1.
Incremental Algorithm

§ Issue: adding a single point to an existing hull
§ Let our set of points be \( P = \{p_0, p_1, \ldots, p_{n-1}\} \)
§ Assume the points are in general position
§ See Algorithm 3.7

\[
\begin{align*}
\text{Algorithm: Incremental Algorithm} \\
&\text{Let } H_2 \leftarrow \text{conv} \{p_0, p_1, p_2\}. \\
&\text{for } k = 3 \text{ to } n - 1 \text{ do} \\
&\quad H_k \leftarrow \text{conv} \{H_{k-1} \cup p_k\}
\end{align*}
\]
Incremental Algorithm

FIGURE 3.10  Tangent lines from $p$ to $Q$; “left” means that $p$ is left of the indicated directed line, and “!left” means “not left.”
FIGURE 3.11 Finding the lower tangent: from (4, 7) to (0, 12).
Algorithm: LOWER TANGENT

\[ a \leftarrow \text{rightmost point of } A. \]
\[ b \leftarrow \text{leftmost point of } B. \]

while \( T = ab \) not lower tangent to both \( A \) and \( B \) do
  while \( T \) not lower tangent to \( A \) do
    \[ a \leftarrow a - 1 \]
    \[ a \leftarrow a - 1 \]
  while \( T \) not lower tangent to \( B \) do
    \[ b \leftarrow b + 1 \]

Algorithm 3.9  Lower tangent.
Lower Bound

FIGURE 3.9 Parabola construction for sorting (2, 3, 5, 8, 9, 10).
Assignments

§ Ex: 3.2.3-4, 3.4.1-2, 3.5.7-1, 3.7.1-3 due 10/01 in the beginning of the class