Convex Hulls in 2d and 3d
Convex Hulls in 2d and 3d
Extreme point
Extreme point

Support line
Jarvis march (Gift wrapping)
Jarvis march (Gift wrapping)

The lowest point is extreme
Jarvis march (Gift wrapping)
Jarvis march (Gift wrapping)
Jarvis march (Gift wrapping)
Jarvis march (Gift wrapping)
Jarvis march (Gift wrapping)

Next point is found
Jarvis march (Gift wrapping)

Next point is found
Then the next
Jarvis march (Gift wrapping)

Next point is found
Then the next
Etc...
Jarvis march (Gift wrapping)

Input: $S$ set of points.
$u$ = the lowest point of $S$;
$\min = \infty$
For all $w \in S \setminus \{u\}$
    if $\angle(ux, uw) < \min$ then $\min = \angle(ux, uw)$; $v = w$;
$u.next = v$;
Do
    $S = S \setminus \{v\}$
    For all $w \in S$
        $\min = \infty$
        If $\angle(v.previous v, vw) < \min$ then
            $\min = \angle(v.previous v, vw)$; $v.next = w$;
$v = v.next$;
While $v \neq u$
Jarvis march (Gift wrapping)

Complexity?

Input: $S$ set of points.
$u =$ the lowest point of $S$;
$\text{min} = \infty$
For all $w \in S \setminus \{u\}$
  if $\text{angle}(ux, uw) < \text{min}$ then $\text{min} = \text{angle}(ux, uw)$; $v = w$;
$u.\text{next} = v$;
Do
  $S = S \setminus \{v\}$
  For all $w \in S$
    $\text{min} = \infty$
    If $\text{angle}(v.\text{previous} v, vw) < \text{min}$ then
      $\text{min} = \text{angle}(v.\text{previous} v, vw)$; $v.\text{next} = w$;
    $v = v.\text{next}$;
While $v \neq u$
Jarvis march (Gift wrapping)

Complexity $O(n)$

Input: $S$ set of points.
$u =$ the lowest point of $S$;
$\min = \infty$

For all $w \in S \setminus \{u\}$

if $\text{angle}(ux, uw) < \min$ then $\min = \text{angle}(ux, uw); \ v = w;$
\hspace{1cm} u.next $= v;$
\hspace{1cm} Do
\hspace{1.5cm} $S = S \setminus \{v\}$
\hspace{1.5cm} For all $w \in S$
\hspace{2cm} $\min = \infty$
\hspace{2cm} If $\text{angle}(v.\text{previous } v, vw) < \min$ then
\hspace{3cm} $\min = \text{angle}(v.\text{previous } v, vw); \ v.\text{next } = w;$
\hspace{1.5cm} $v = v.\text{next }$;
\hspace{1cm} While $v \neq u$
Jarvis march (Gift wrapping)

Complexity?

Input: $S$ set of points.
$u =$ the lowest point of $S$;
$\min = \infty$
For all $w \in S \setminus \{u\}$
if $\text{angle}(ux, uw) < \min$ then $\min = \text{angle}(ux, uw)$; $v = w$;
$u.\text{next} = v$;
Do
$S = S \setminus \{v\}$
For all $w \in S$
$\min = \infty$
If $\text{angle}(v.\text{previous} v, vw) < \min$ then
$\min = \text{angle}(v.\text{previous} v, vw)$; $v.\text{next} = w$;
$v = v.\text{next}$;
While $v \neq u$

$O(n)$
Jarvis march (Gift wrapping)

Complexity ?

Input : $S$ set of points.
$u =$ the lowest point of $S$;
$\min = \infty$
For all $w \in S \setminus \{u\}$
    if $\text{angle}(ux, uw) < \min$ then $\min = \text{angle}(ux, uw)$; $v = w$;
$u.$next $= v$;
Do
    $S = S \setminus \{v\}$
    For all $w \in S$
        $\min = \infty$
        If $\text{angle}(v.$previous $v, vw) < \min$ then
            $\min = \text{angle}(v.$previous $v, vw)$; $v.$next $= w$;
    $v = v.$next;
While $v \neq u$
Jarvis march (Gift wrapping)

Complexity?

Input: $S$ set of points.
$u =$ the lowest point of $S$;
$\min = \infty$
For all $w \in S \setminus \{u\}$
if $\angle(ux, uw) < \min$ then $\min = \angle(ux, uw)$; $v = w$;
$u.next = v$;

Do
$S = S \setminus \{v\}$
For all $w \in S$
if $\angle(v.previous v, vw) < \min$ then
$\min = \angle(v.previous v, vw)$; $v.next = w$;
$v = v.next$;

While $v \neq u$ \hspace{1cm} $O(n)$
Jarvis march (Gift wrapping)

Complexity \( O(n^2) \)

Input : \( S \) set of points.
\( u = \) the lowest point of \( S \);
\( \min = \infty \)
For all \( w \in S \setminus \{u\} \)
  if \( \text{angle}(ux, uw) < \min \) then \( \min = \text{angle}(ux, uw) \); \( v = w \);
\( u.next = v \);
Do
  \( S = S \setminus \{v\} \)
  For all \( w \in S \)
    \( \min = \infty \)
    If \( \text{angle}(v(previous v), vw) < \min \) then
      \( \min = \text{angle}(v(previous v), vw) \); \( v.next = w \);
  \( v = v.next \);
While \( v \neq u \)
Jarvis march (Gift wrapping)

Complexity \(O(nh)\)

Input: \(S\) set of points.  
\(u = \) the lowest point of \(S\);  
\(\text{min} = \infty\)  
For all \(w \in S \setminus \{u\}\)  
\[\text{if} \ \angle(ux, uw) < \text{min} \ \text{then} \ \text{min} = \angle(ux, uw); \ v = w;\]  
\(u.\text{next} = v;\)  
Do  
\[S = S \setminus \{v\}\]  
For all \(w \in S\)  
\[\text{min} = \infty\]  
\[\text{If} \ \angle(v.\text{previous} v, vw) < \text{min} \ \text{then} \]
\[\text{min} = \angle(v.\text{previous} v, vw); \ v.\text{next} = w;\]
\(v = v.\text{next};\)  
While \(v \neq u\)
Graham scan
Graham scan

One interior point
Graham scan

One interior point
Sorting around this point
Graham scan

One interior point

Sorting around this point
Graham scan

One interior point

Sorting around this point
Graham scan

One interior point
Sorting around this point
Graham scan

One interior point
Sorting around this point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point

Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point

Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point

Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point
Scan starting from the lowest point
Graham scan

One interior point

Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point

Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point
Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point

Scan starting from the lowest point
Graham scan

One interior point
Sorting around this point
Scan starting from the lowest point
Graham Scan

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
u = the lowest point of $S$;
v = u;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;
Graham Scan

Complexity

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
$u =$ the lowest point of $S$;
v = u;
while $v.next \neq u$
  if $(v, v.next, v.next.next)$ turn left
    $v = v.next$;
  else
    $v.next = v.next.next$;
if $v \neq u$ $v = v.previous$;
Graham Scan

Complexity

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
$u =$ the lowest point of $S$;
$v = u$;
while $v.next \neq u$
  if $(v, v.next, v.next.next)$ turn left
    $v = v.next$;
  else
    $v.next = v.next.next$;
  if $v \neq u$ $v = v.previous$;
Graham Scan

Complexity

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
$u$ = the lowest point of $S$;
$v = u$;
while $v.$next $\neq u$
    if $(v, v.$next, v.$next.$next)$ turn left
        $v = v.$next;
    else
        $v.$next = v.$next.$next;
        if $v \neq u$ $v = v.$previous;

$O(n \log n)$
Graham Scan

Complexity

Input : $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u =$ the lowest point of $S$;
- $v = u$;
- while $v.next \neq u$
  - if $(v, v.next, v.next.next)$ turn left
    - $v = v.next$;
  - else
    - $v.next = v.next.next$;
    - if $v \neq u$ $v = v.previous$;

$O(1)$
$O(n \log n)$
Graham Scan

Complexity

Input: $S$ a set of $n$ points.

1. origin = barycenter of 3 points of $S$;
2. sort $S$ around the origin;
3. $u =$ the lowest point of $S$;
4. $v =$ $u$;
5. while $v$.next $\neq u$
   
   if $(v, v$.next, v$.next$.next) =$ turn left
      
      $v =$ $v$.next;
   
   else
      
      $v$.next = $v$.next$.next;
      
      if $v \neq u$ $v =$ $v$.previous;

$O(1)$

$O(n \log n)$

$O(n)$
Graham Scan
Complexity

Input: $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u =$ the lowest point of $S$;
- $v = u$;

while $v.next \neq u$

  if $(v, v.next, v.next.next)$ turn left
    $v = v.next$;
  else
    $v.next = v.next.next$;
  if $v \neq u$ $v = v.previous$;

$O(1)$
$O(n \log n)$
$O(n)$
Input: $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u =$ the lowest point of $S$;
- $v = u$;

while $v.next \neq u$

  if $(v, v.next, v.next.next)$ turn left
    $v = v.next$;
  else
    $v.next = v.next.next$;

if $v \neq u$ $v = v.previous$;

Graham Scan
Complexity

$O(1)$
$O(n \log n)$
$O(n)$
Graham Scan
Complexity

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
$u =$ the lowest point of $S$;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

$O(1)$
$O(n \log n)$
$O(n)$

at most $n$ deletions
Input: $S$ a set of $n$ points.

origin = barycenter of 3 points of $S$;

sort $S$ around the origin;

$u =$ the lowest point of $S$;

$v = u$;

while $v$ next $\neq u$

if $(v, v$ next $, v.next.next)$ turn left

$v = v$ next;

else

$v$ next $= v.next.next$;

if $v \neq u$ $v = v.previous$;

at most $n$ deletions

\begin{align*}
\text{Graham Scan} & \\
\text{Complexity} & \\
O(1) & \\
O(n \log n) & \\
O(n) & \text{at most } n \text{ deletions}
\end{align*}
Input: $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u$ = the lowest point of $S$;
- $v = u$;

while $v$.next $\neq u$
  if $(v, v$.next, v$.next$.next) turn left
    $v = v$.next;
  else
    $v$.next = v$.next$.next;
    if $v \neq u$ $v = v$.previous;

Graham Scan Complexity

Input: $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u$ = the lowest point of $S$;
- $v = u$;

while $v$.next $\neq u$
  if $(v, v$.next, v$.next$.next) turn left
    $v = v$.next;
  else
    $v$.next = v$.next$.next;
    if $v \neq u$ $v = v$.previous;

- $O(1)$
- $O(n \log n)$
- $O(n)$

at most $n$ times

at most $n$ deletions
Graham Scan

Complexity

Input: $S$ a set of $n$ points.

- origin = barycenter of 3 points of $S$;
- sort $S$ around the origin;
- $u =$ the lowest point of $S$;
- $v = u$;

while $v$.next $\neq u$
  if $(v, v$.next, v.next.next) turn left
    $v = v$.next;
  else
    $v$.next = v.next.next;
    if $v \neq u$ $v = v$.previous;

$O(1)$
$O(n \log n)$
$O(n)$
$O(n)$
Graham Scan

Complexity

Input: $S$ a set of $n$ points.
origin = barycenter of 3 points of $S$;
sort $S$ around the origin;
$u =$ the lowest point of $S$;
$v = u$;
while $v.next \neq u$
    if $(v, v.next, v.next.next)$ turn left
        $v = v.next$;
    else
        $v.next = v.next.next$;
        if $v \neq u$ $v = v.previous$;

$O(n \log n)$
Graham alternative: origin at $y = -\infty$
Graham alternative: origin at $y = -\infty$

Sort in $x$
Graham alternative: origin at $y = -\infty$

Sort in $x$
Graham alternative: origin at $y = -\infty$

Sort in $x$

Upper envelope
Graham alternative: origin at $y = -\infty$

Upper envelope

Add the lower envelope
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm
Deterministic incremental algorithm

Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
  $w = u$
  while $(v, u, u.next)$ turn right
    $u = u.next$;
    $v.next = u; u.previous = v$;
  while $(v, w, w.previous)$ turn left
    $w = w.previous$;
    $v.previous = w; w.next = v$;
  $u = v$;
Deterministic incremental algorithm

Input: $S$ set of $n$ points. 
sort $S$ in $x$; 
initialize a circular list with the 3 leftmost points such that $u$ is on the right $u, u.next, u.next.next$ turn left; 
For $v$ the next point in $x$ 
  $w = u$
  while $(v, u, u.next)$ turn right 
    $u = u.next$;
    $v.next = u$; $u.previous = v$;
  while $(v, w, w.previous)$ turn left 
    $w = w.previous$;
    $v.previous = w$; $w.next = v$;
  $u = v$;
Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right, $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
  $w = u$
  while $(v, u, u.next)$ turn right
    $u = u.next$;
    $v.next = u; u.previous = v$;
  end while
  $(v, w, w.previous)$ turn left
  $w = w.previous$;
  $v.previous = w; w.next = v$;
  $u = v$;
Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
  $w = u$
  while $(v, u, u.next)$ turn right
    $u = u.next$;
    $v.next = u$; $u.previous = v$;
  while $(v, w, w.previous)$ turn left
    $w = w.previous$;
    $v.previous = w$; $w.next = v$;
  $u = v$;
Deterministic incremental algorithm

Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
\[
w = u\]
\[
\text{while } (v, u, u.next) \text{ turn right } \\
u = u.next; \quad v.next = u; \quad u.previous = v; \\
\text{while } (v, w, w.previous) \text{ turn left } \\
w = w.previous; \quad v.previous = w; \quad w.next = v; \quad u = v;
\]
Deterministic incremental algorithm

Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
  $w = u$
  while $(v, u, u.next)$ turn right
    $u = u.next$;
    $v.next = u; u.previous = v$;
  while $(v, w, w.previous)$ turn left
    $w = w.previous$;
    $v.previous = w; w.next = v$;
  $u = v$;
Deterministic incremental algorithm

Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$
  $w=u$
  while $(v, u, u.next)$ turn right
    $u = u.next$;
  $v.next = u; u.previous = v$;
  while $(v, w, w.previous)$ turn left
    $w = w.previous$;
  $v.previous = w; w.next = v$;
  $u = v$;
Deterministic incremental algorithm

**Complexity**

**Input**: $S$ set of $n$ points.

sort $S$ in $x$;

initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;

For $v$ the next point in $x$

$w = u$

while $(v, u, u.next)$ turn right

$u = u.next$;
$v.next = u; u.previous = v$;

while $(v, w, w.previous)$ turn left

$w = w.previous$;
$v.previous = w; w.next = v$;
$u = v$;
Deterministic incremental algorithm

Complexity

Input: $S$ set of $n$ points.

sort $S$ in $x$;

initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;

For $v$ the next point in $x$

$w = u$

while $(v, u, u.next)$ turn right

$u = u.next$;
$v.next = u; u.previous = v$;

while $(v, w, w.previous)$ turn left

$w = w.previous$;
$v.previous = w; w.next = v$;

$u = v$;
Input: \( S \) set of \( n \) points.
sort \( S \) in \( x \);
initialize a circular list with the 3 leftmost points
such that \( u \) is on the right \( u, u.next, u.next.next \) turn left;
For \( v \) the next point in \( x \)
\[w = u\]
\[
\text{while} \ (v, u, u.next) \ 	ext{turn right}
\]
\[
u = u.next;
\]
\[
v.next = u; \ u.previous = v;
\]
\[
\text{while} \ (v, w, w.previous) \ 	ext{turn left}
\]
\[
w = w.previous;
\]
\[
v.previous = w; \ w.next = v;
\]
\[u = v;\]
Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points
such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$

\[
\begin{align*}
\text{w} & = u \\
\text{while (v, u, u.next) turn right} & \\
\text{u} & = u.next; \\
\text{v.next} & = u; u.previous = v; \\
\text{while (v, w, w.previous) turn left} & \\
\text{w} & = w.previous; \\
\text{v.previous} & = w; w.next = v; \\
\end{align*}
\]

"Draw an edge in the triangulation"
Deterministic incremental algorithm

Input: $S$ set of $n$ points.
sort $S$ in $x$;
initialize a circular list with the 3 leftmost points such that $u$ is on the right $u, u.next, u.next.next$ turn left;
For $v$ the next point in $x$

$$\begin{align*}
&\texttt{w:=u} \\
&\texttt{while (v, u, u.next) turn right} \\
&\hspace{1em} u = u.next; \\
&\hspace{1em} v.next = u; u.previous = v; \\
&\texttt{while (v, w, w.previous) turn left} \\
&\hspace{1em} w = w.previous; \\
&\hspace{1em} v.previous = w; w.next = v; \\
&\texttt{v:=v} \\
\end{align*}$$

nb of edges $\simeq 3n$

Complexity

Draw an edge in the triangulation
Deterministic incremental algorithm

**Complexity**

\[ O(n \log n) \]

**Input:** A set of \( n \) points.
Sort \( S \) in \( x \);
initialize a circular list with the 3 leftmost points such that \( u \) is on the right, \( u, u.next, u.next.next \) turn left;

For \( v \) the next point in \( x \)

\[\begin{align*}
  &w = u \\
  &\text{while } (v, u, u.next) \text{ turn right} \\
  &\quad u = u.next; \\
  &\quad v.next = u; u.previous = v; \\
  &\text{while } (v, w, w.previous) \text{ turn left} \\
  &\quad w = w.previous; \\
  &\quad v.previous = w; w.next = v; \\
  &u = v;
\end{align*}\]
Divide & conquer algorithm
Divide & conquer algorithm
Divide & conquer algorithm
Divide & conquer algorithm

Common tangents
Upper tangent
Upper tangent

Topmost points
Upper tangent
Upper tangent
Upper tangent

$O(n)$
Divide & conquer algorithm

Complexity

\[ f(n) = \]
Divide & conquer algorithm

Complexity

\[ f(n) = A \cdot n + f\left(\frac{n}{2}\right) + f\left(\frac{n}{2}\right) \]
Divide & conquer algorithm

Complexity

\[ f(n) = A \cdot n + f\left(\frac{n}{2}\right) + f\left(\frac{n}{2}\right) \]

\[ = O(n \log n) \]
Divide & conquer algorithm

Complexity

\[ f(n) = A \cdot n + f\left(\frac{n}{2}\right) + f\left(\frac{n}{2}\right) \]

\[ = O(n \log n) \]

Divide and merge in \( O(n) \)

Balanced partition

(preprocessing in \( O(n \log n) \))
CHAN’s ALGORITHM
CHAN’s ALGORITHM

Compute the CH of $k$ subsets of $P$
CHAN’s ALGORITHM

Wrap around the $CH$ of the $k$ subsets
CHAN’s ALGORITHM

Compute the CH of \( k = n/m \) subsets of size \( m \):
each in \( O(m \log m) \), all in \( k/m \times O(m \log m) = O(n \log m) \)
CHAN’s ALGORITHM

Compute the CH of $k = n/m$ subsets of size $m$:
   each in $O(m \log m)$, all in $k/m \times O(m \log m) = O(n \log m)$

Find the $s_i$: $O(n)$
CHAN’s ALGORITHM

One wrapping step:
find the bitangents $t_i$ on each subset:
$n/m + O(\text{nb points visited})$
$= n/m + O(\text{nb points removed})$
CHAN’s ALGORITHM

Compute the CH of $k = n/m$ subsets of size $m$:
   each in $O(m \log m)$, all in $k/m \times O(m \log m) = O(n \log m)$

Find the $s_i$: $O(n)$

One wrapping step:
   find the bitangents $t_i$ on each subset:
   $n/m + O($nb points visited$)$
   =$n/m + O($nb points removed$)$

For $h$ wrapping steps:
   $O(hn/m +$ total points removed$)$
   =$O(hn/m + n)$

Total:
   $O(n \log m + hn/m + n) =$
   $O(n(1 + h/m + \log m))$
CHAN’s ALGORITHM

Set a parameter $H$, call $h$ the actual size of the convex hull of $P$

Hull(P,m,H)
   Do $H$ wrapping steps
      If the last wrap comes back to the first point then return success
      Else return incomplete

• Success if $H > h$
• Complexity of Hull(P,H,H) is $O(n \log H)$
CHAN’s ALGORITHM

Set a parameter $H$, call $h$ the actual size of the convex hull of $P$

Hull(P,m,H)
   Do $H$ wrapping steps
      If the last wrap comes back to the first point then return success
      Else return incomplete

- Success if $H > h$
- Complexity of Hull(P,H,H) is $O(n \log H)$

Hull(P)
   For $i = 1, 2, ...$ do
      $L = \text{Hull}(P,H,H)$ with $m = H = \min(2^i, n)$
      If $L \neq \text{incomplete}$ then return $L$

Complexity:
   Nb of iterations = $O(\log \log h)$
   Cost of $i^{th}$ iterations = $O(n \log H) = O(n 2^i)$
   Total: $O(\sum_{i=1}^{\log \log n} n 2^i) = O(n 2^{\log \log n + 1}) = O(n \log h)$
Special case: simple polygonal line
Special case: simple polygonal line

already seen: monotone polyline in Graham’s scan
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

Graham does not work
Special case: simple polygonal line

\[ P = p_1, p_2, \ldots, p_{14} \]

\[ Q \text{ contains the subsequence } p_2, p_4, p_6, p_{11} \]
Special case: simple polygonal line

Incremental algorithm: order given by the polyline
\( A_M = \) highest rank point in \( CH(P_{i-1}) \)

Property: \( p_i \) interior to \( CH(P_{i-1}) \) iff \( p_i \in H^+_{\text{pred}} \cap H^+_{\text{succ}} \)
Special case: simple polygonal line

Incremental algorithm: order given by the polyline

$A_M =$ highest rank point in $CH(P_{i-1})$

Property: $p_i$ interior to $CH(P_{i-1})$ iff $p_i \in H_{pred}^{+} \cap H_{succ}^{+}$

Update $CH(P_i)$ as in the incremental algorithm

Complexity: $O(n)$
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping
3D: Gift wrapping

Complexity: $O(nh)$
3D Divide & conquer algorithm
3D Divide & conquer algorithm

Sort in $x$ and divide
3D Divide & conquer algorithm
3D Divide & conquer algorithm

Find new edge: construct the \( CH \) of the projections, use the 2d algo for bitangent
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges
3D Divide & conquer algorithm

Search only in the star of the new edge vertices
3D Divide & conquer algorithm
Search only in the star of the new edge vertices
3D Divide & conquer algorithm

Search only in the star of the new edge vertices
3D Divide & conquer algorithm

Search only in the star of the new edge vertices
3D Divide & conquer algorithm
Search only in the star of the new edge vertices
3D Divide & conquer algorithm

Search only in the star of the new edge vertices
3D Divide & conquer algorithm

Search only in the star of the new edge vertices  Merge in $O(n)$
3D Divide & conquer algorithm

Search only in the star of the new edge vertices    Merge in $O(n)$

Complexity: $O(n \log n)$