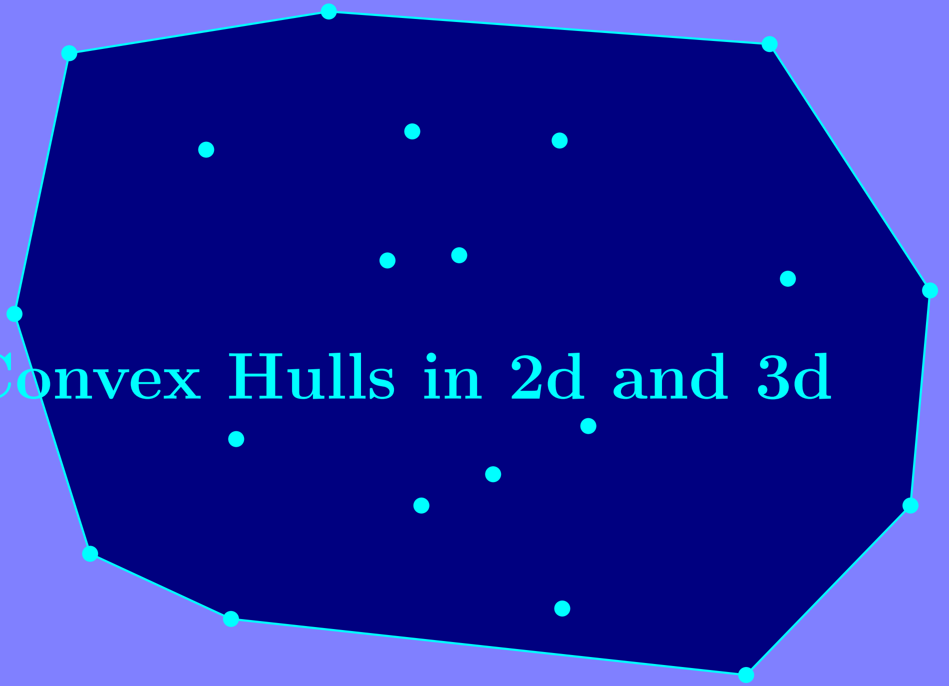
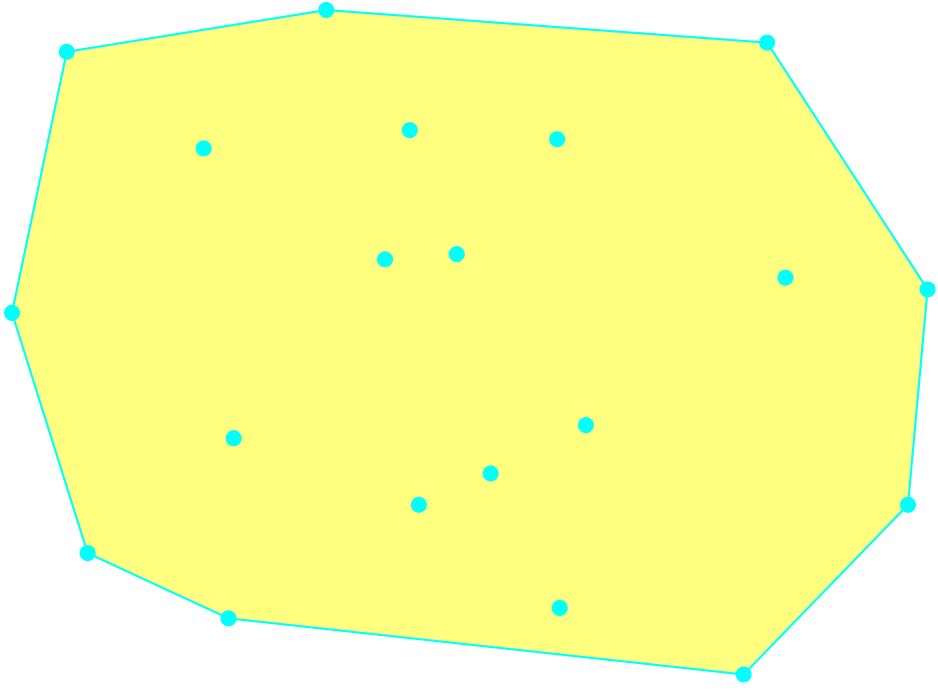
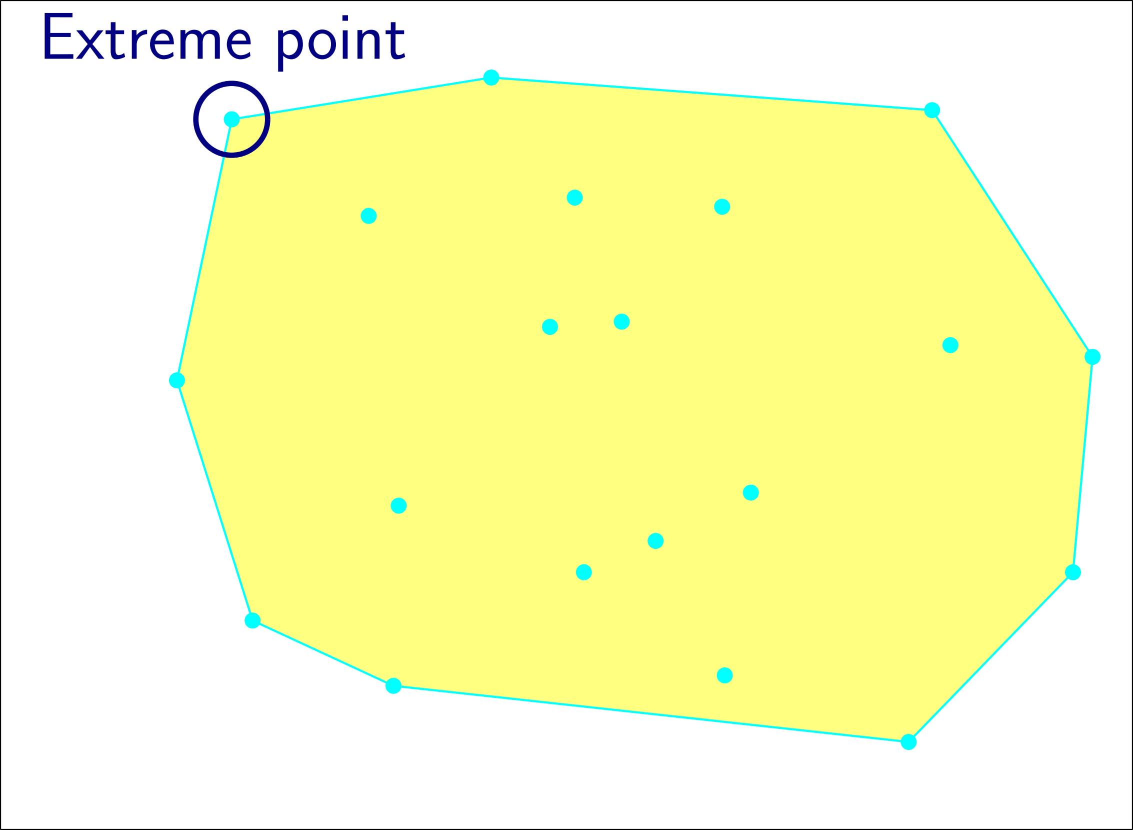
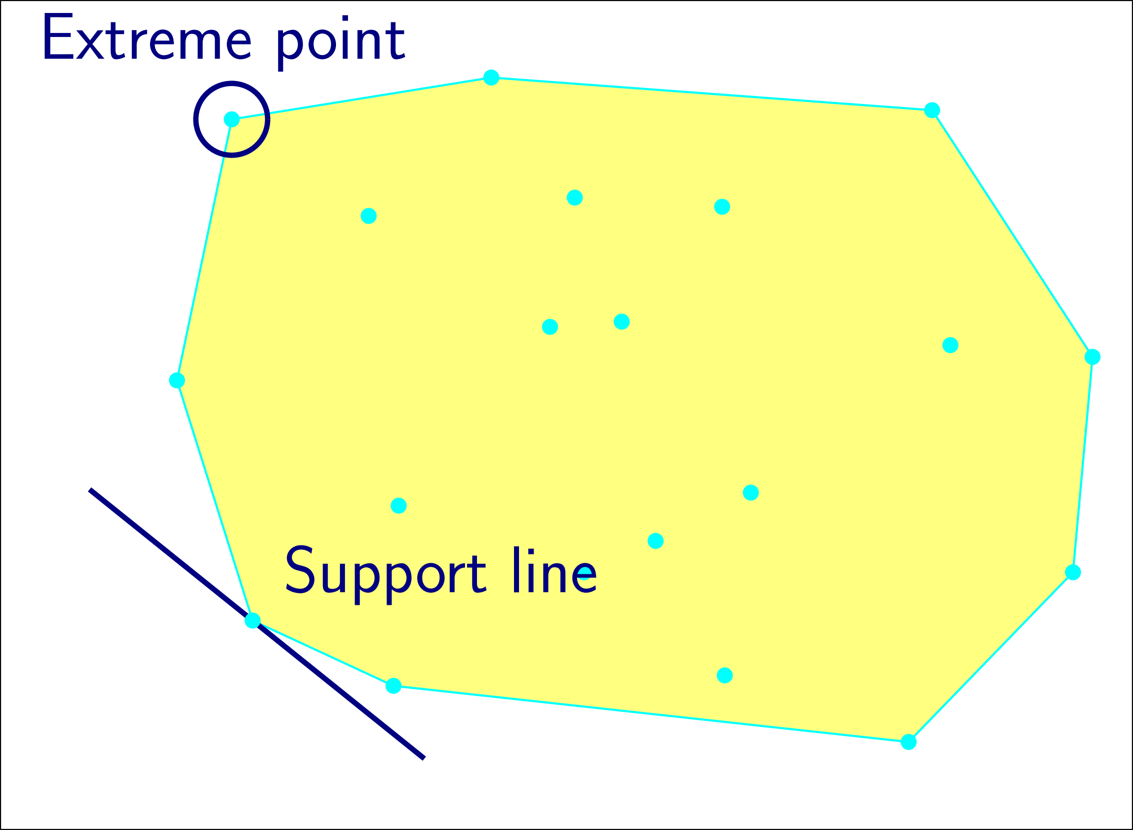
Convex Hulls in 2d and 3d

Convex Hulls in 2d and 3d

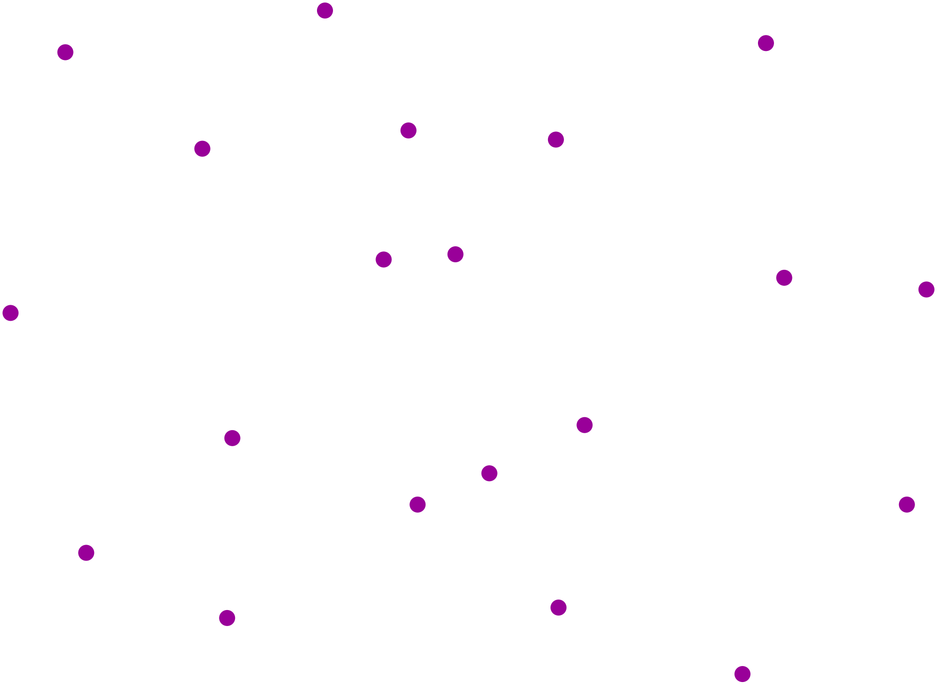





Extreme point

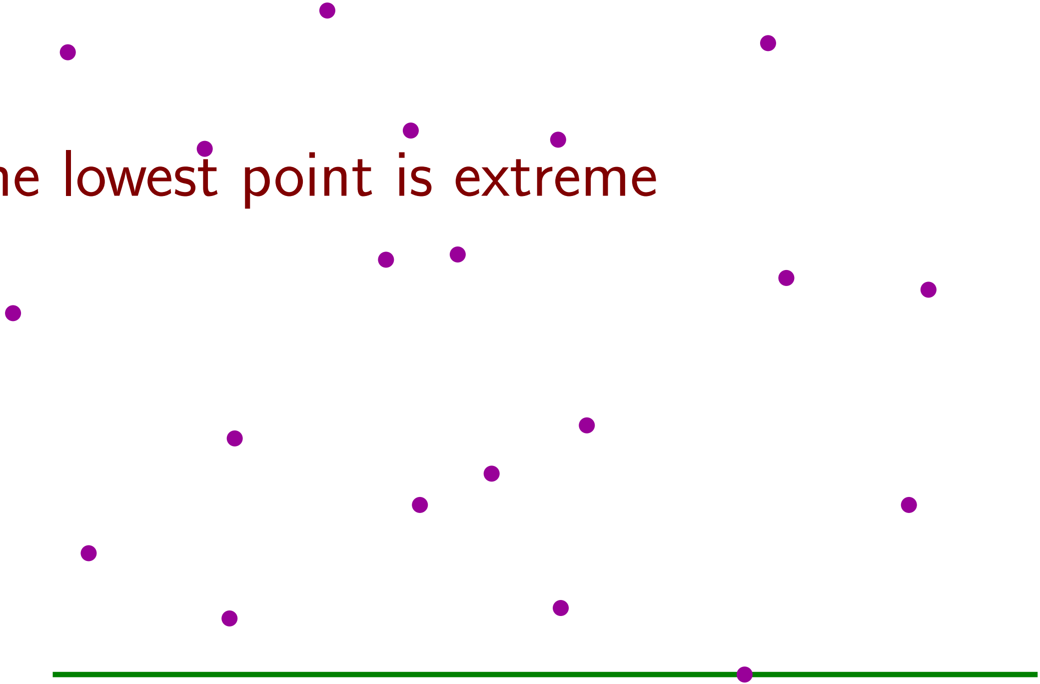

Extreme point
Support line

Jarvis march (Gift wrapping)

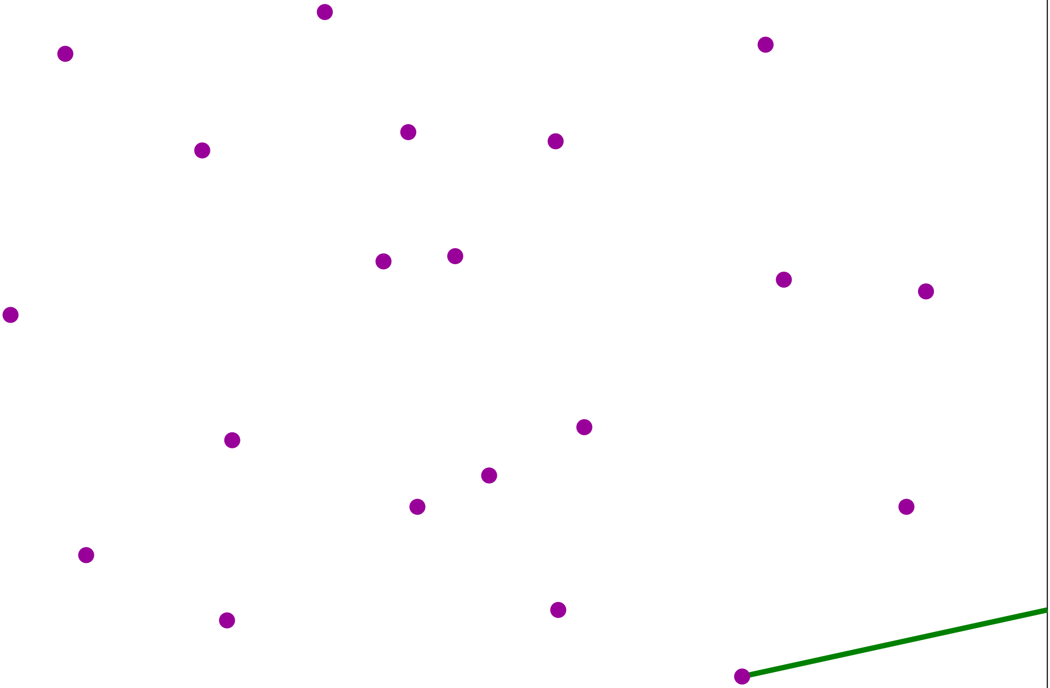


Jarvis march (Gift wrapping)

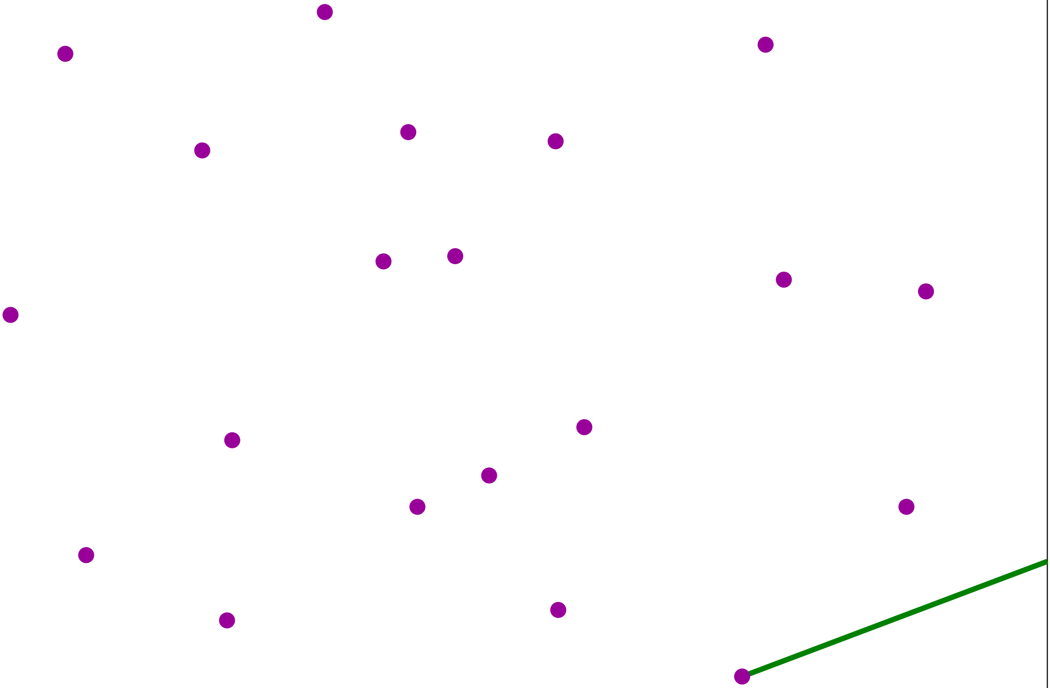
The lowest point is extreme



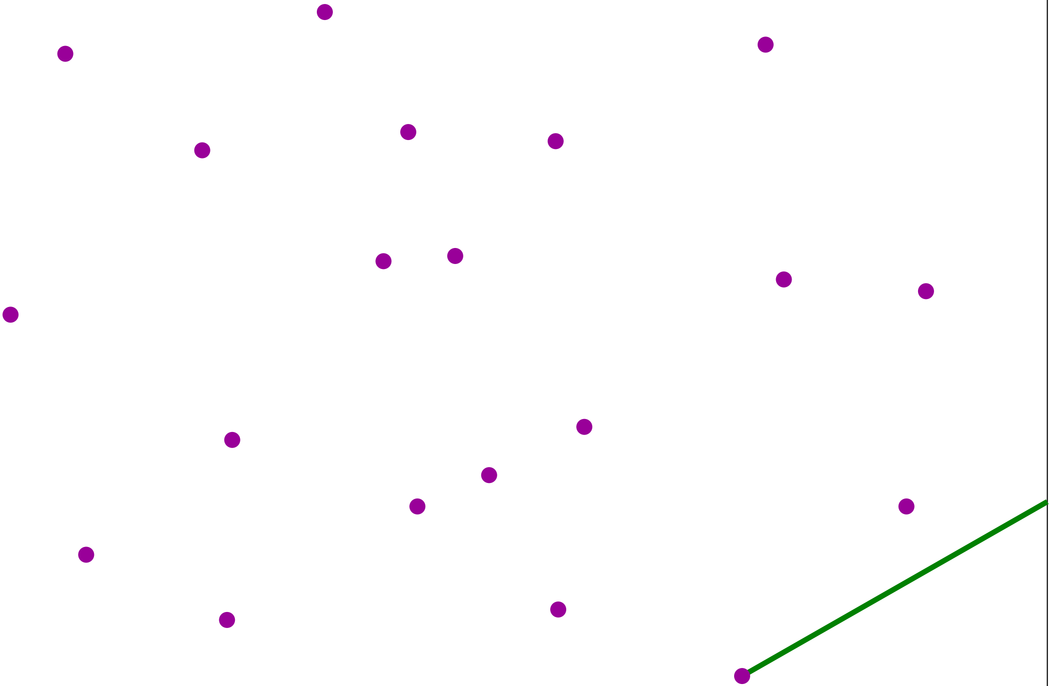
Jarvis march (Gift wrapping)



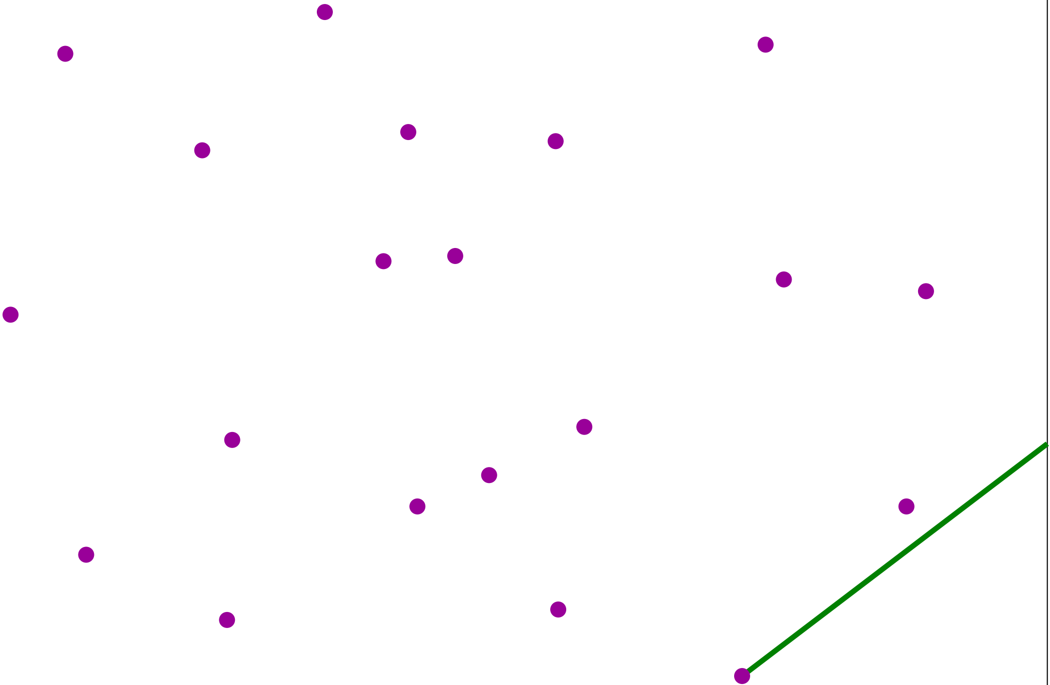
Jarvis march (Gift wrapping)



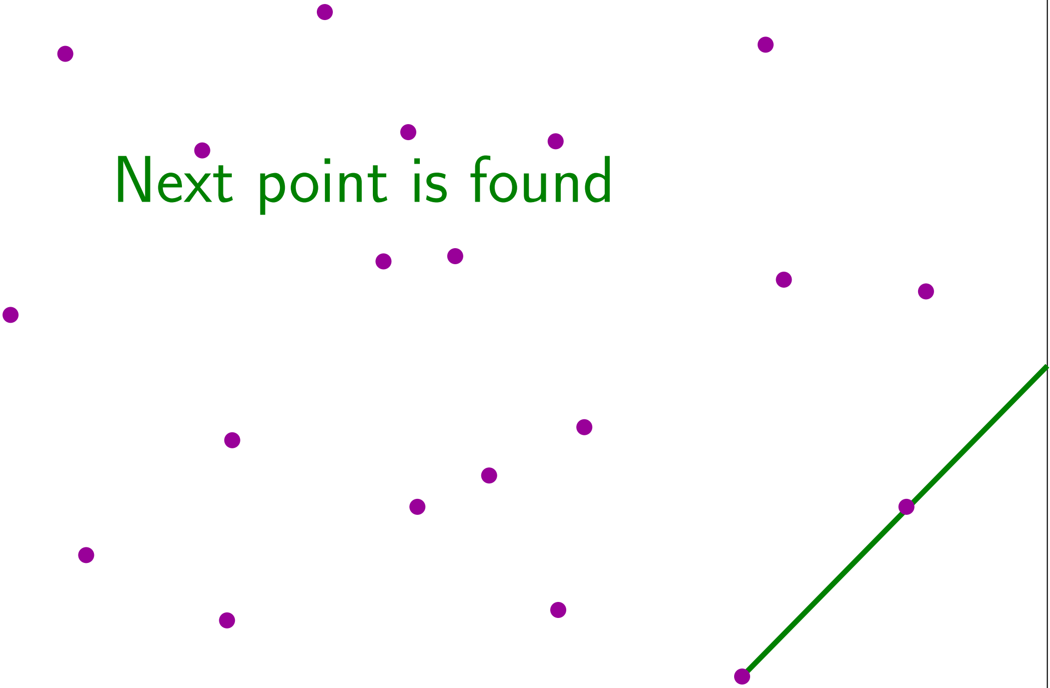
Jarvis march (Gift wrapping)



Jarvis march (Gift wrapping)



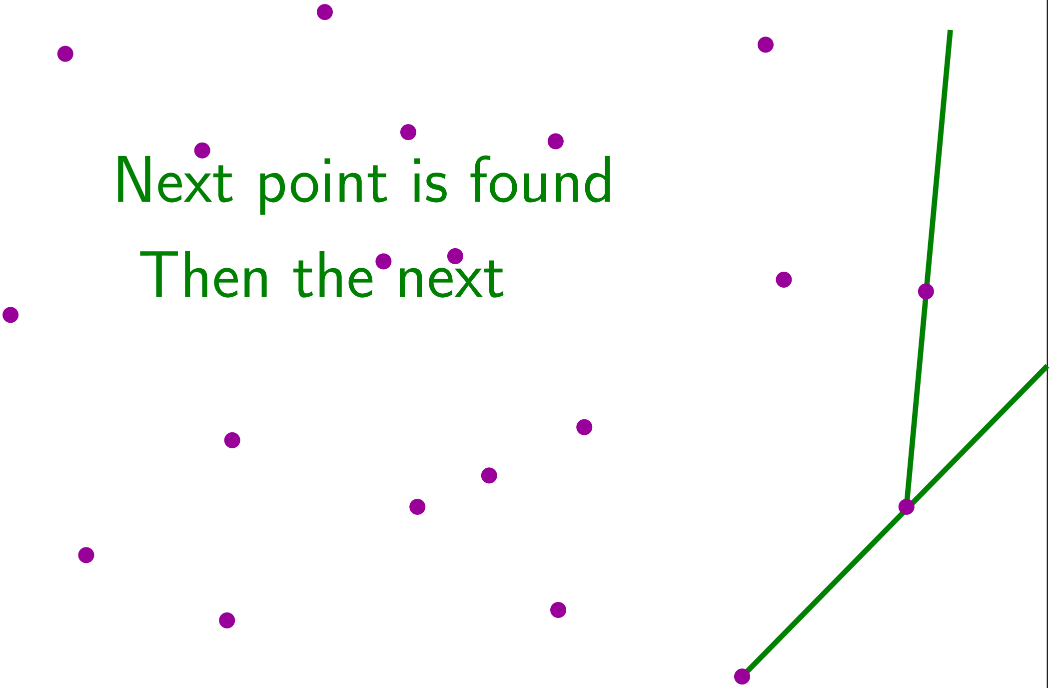
Jarvis march (Gift wrapping)



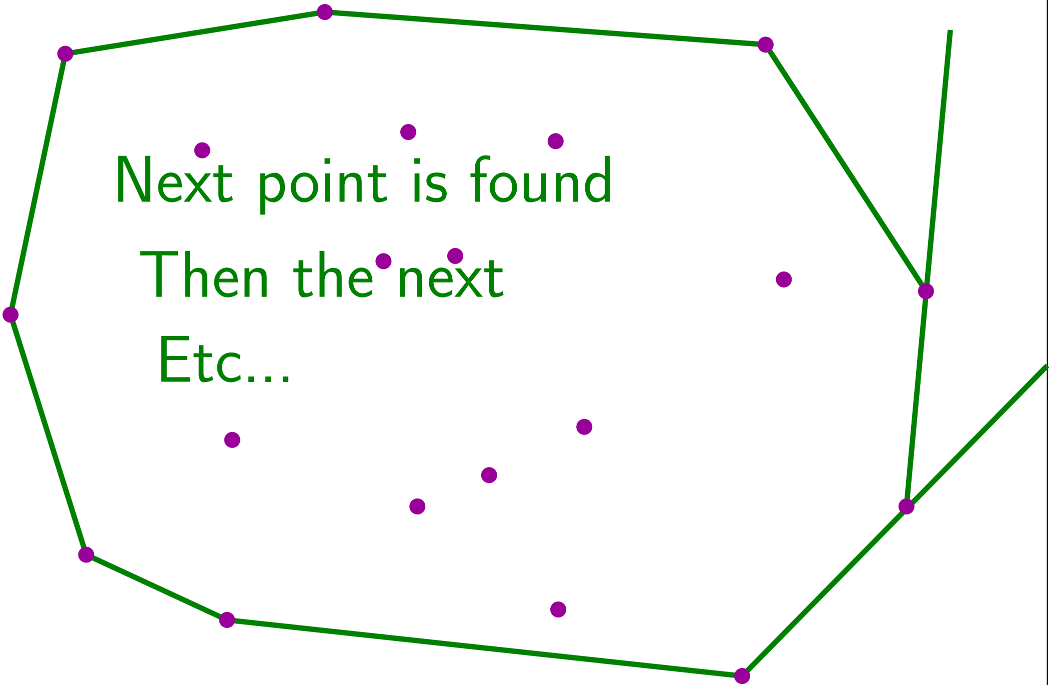
Jarvis march (Gift wrapping)

Next point is found

Then the next



Jarvis march (Gift wrapping)



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;
$min = \infty$
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;
$min = \infty$
For all $w \in S \setminus \{u\}$

$$O(n)$$

    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;
$min = \infty$
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 $v \neq u$

$$O(n)$$

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 $angle(ux, uw) < min$ 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;
$min = \infty$
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 $v \neq u$

$O(n)$

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 $angle(ux, uw) < min$ 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 $v \neq u$

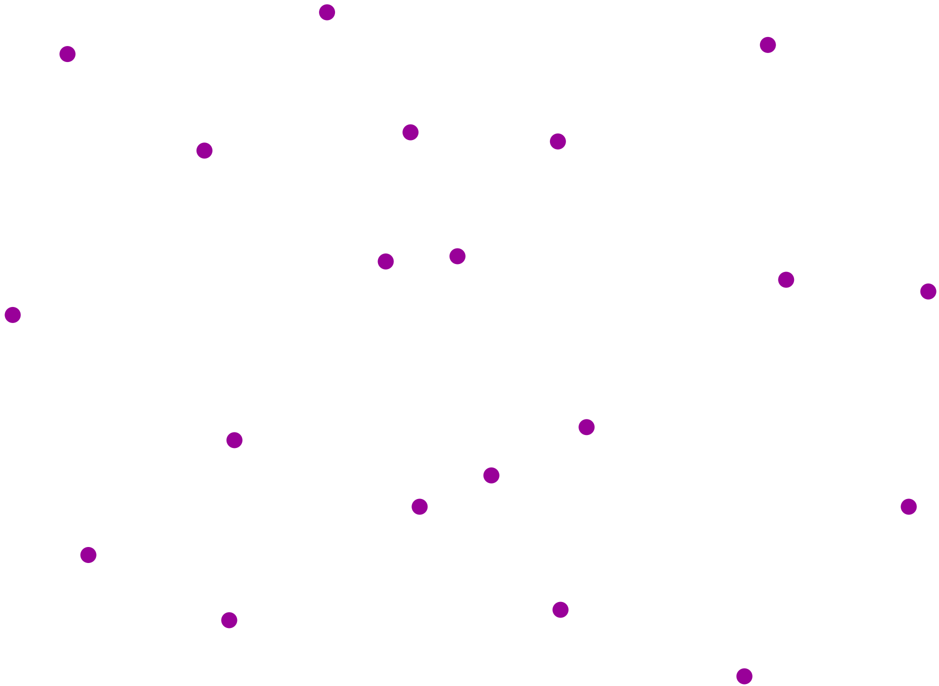
Jarvis march (Gift wrapping)

Complexity ?

$$O(nh)$$

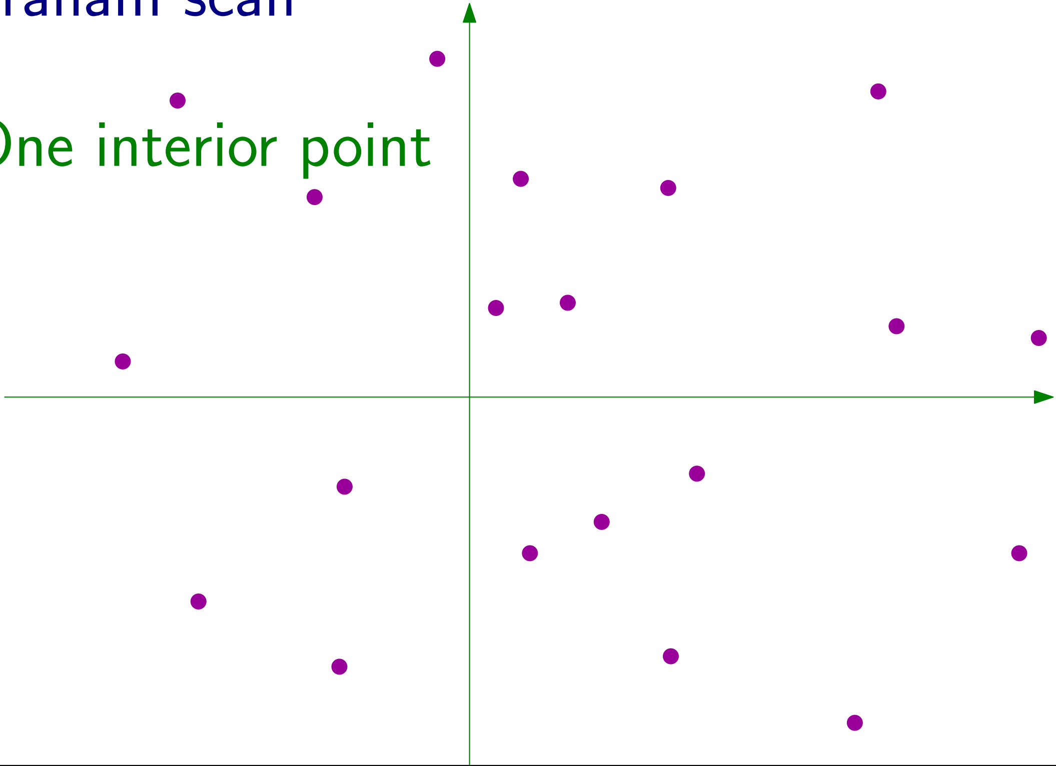
Input : S set of points.
$u =$ the lowest point of S;
$min = \infty$
For all $w \in S \setminus \{u\}$
    if $angle(ux, uw) < min$ then $min = angle(ux, uw)$; $v = w$;
$u.next = v$;
Do
    $S = S \setminus \{v\}$
    For all $w \in S$
        $min = \infty$
        If $angle(v.previous\ v, vw) < min$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

Graham scan



Graham scan

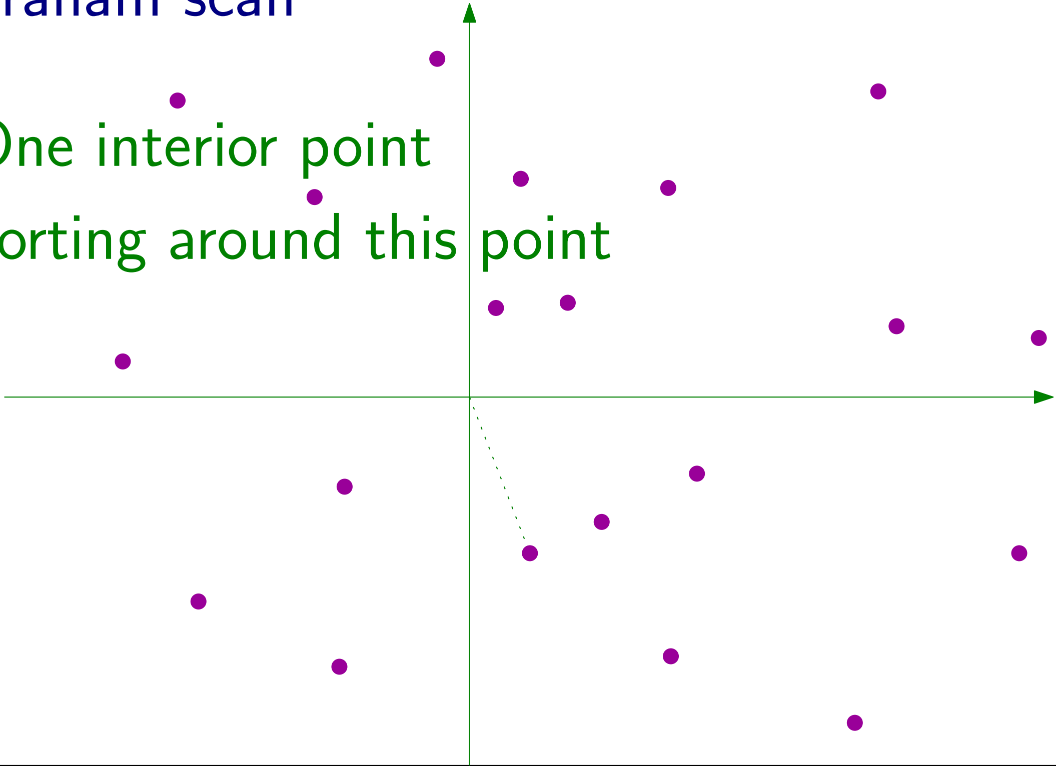
One interior point



Graham scan

One interior point

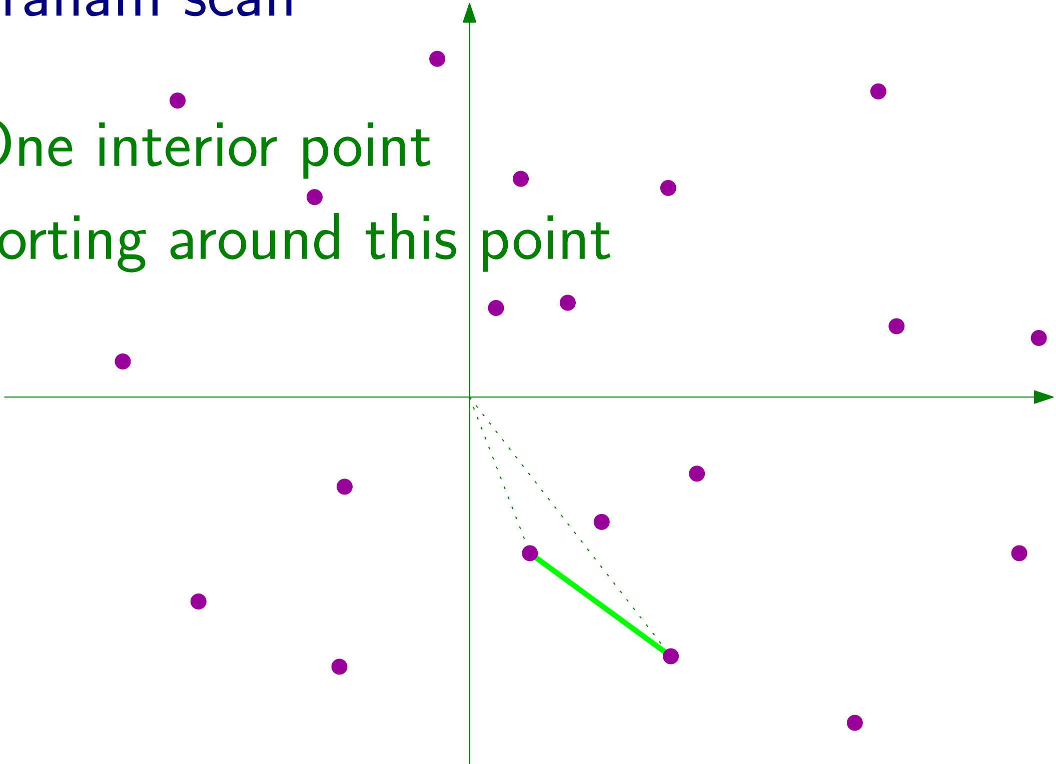
Sorting around this point



Graham scan

One interior point

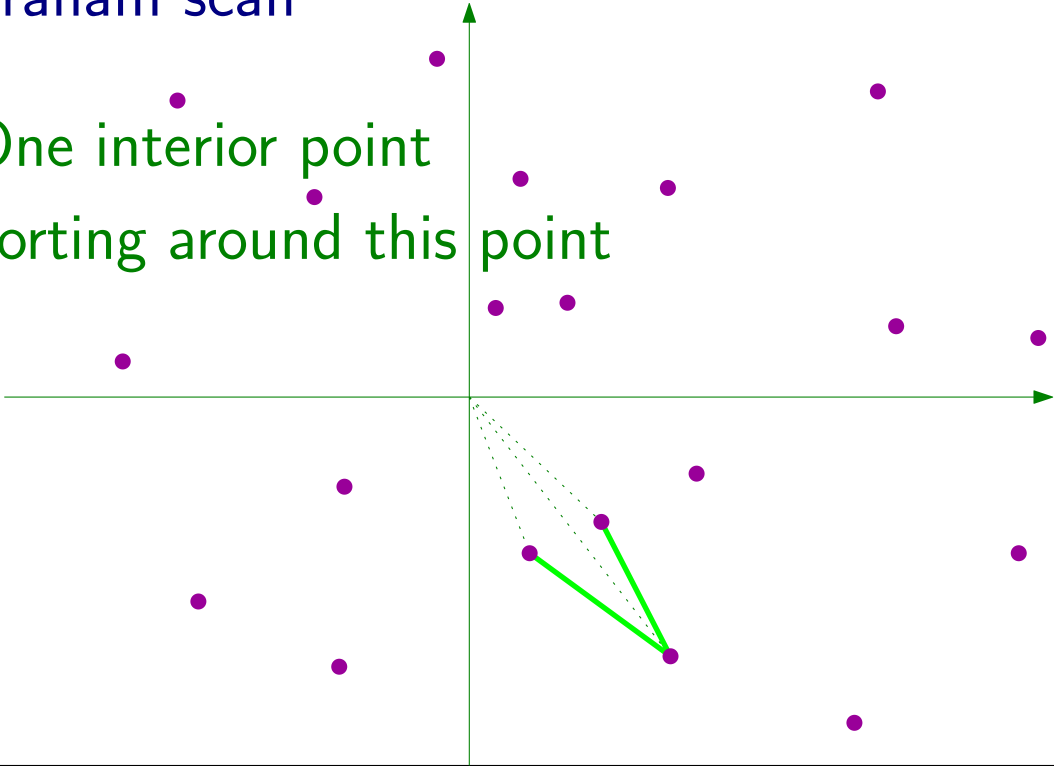
Sorting around this point



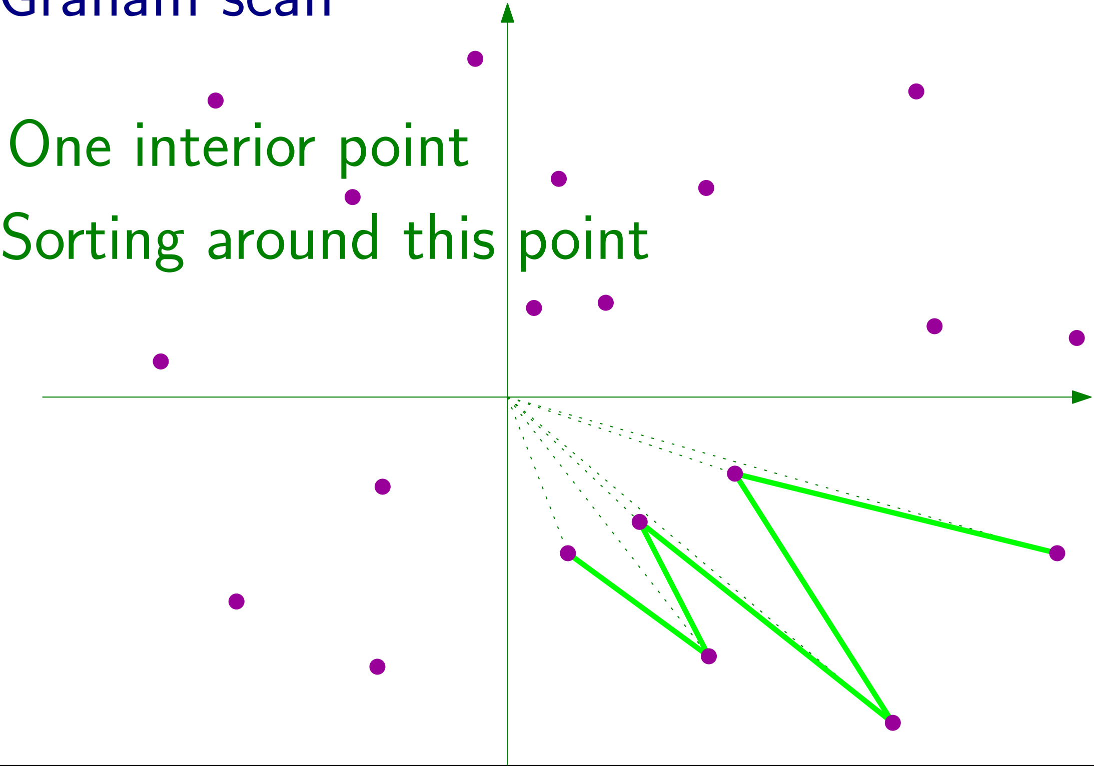
Graham scan

One interior point

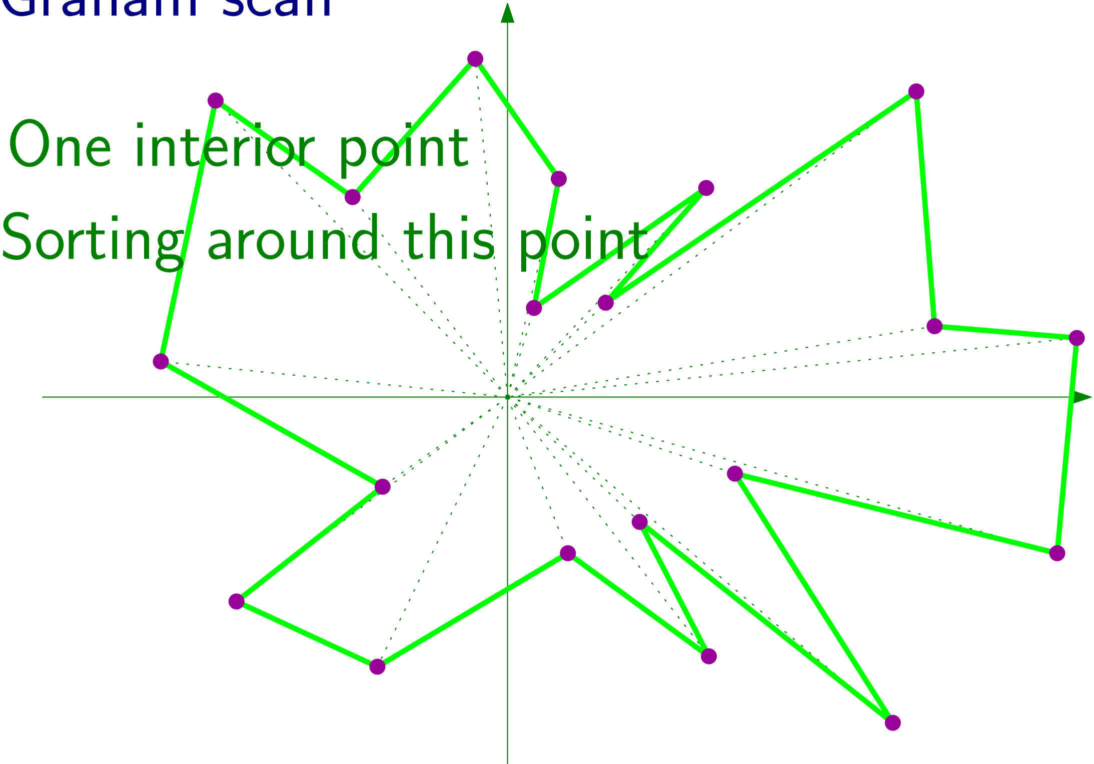
Sorting around this point



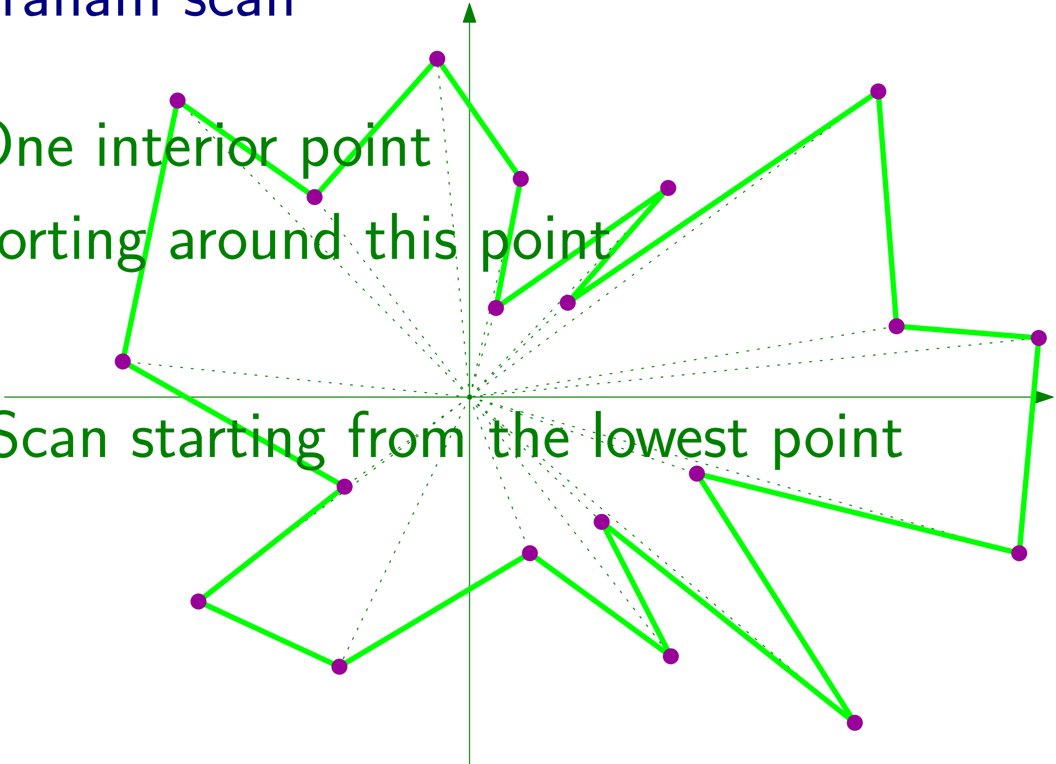
Graham scan



Graham scan

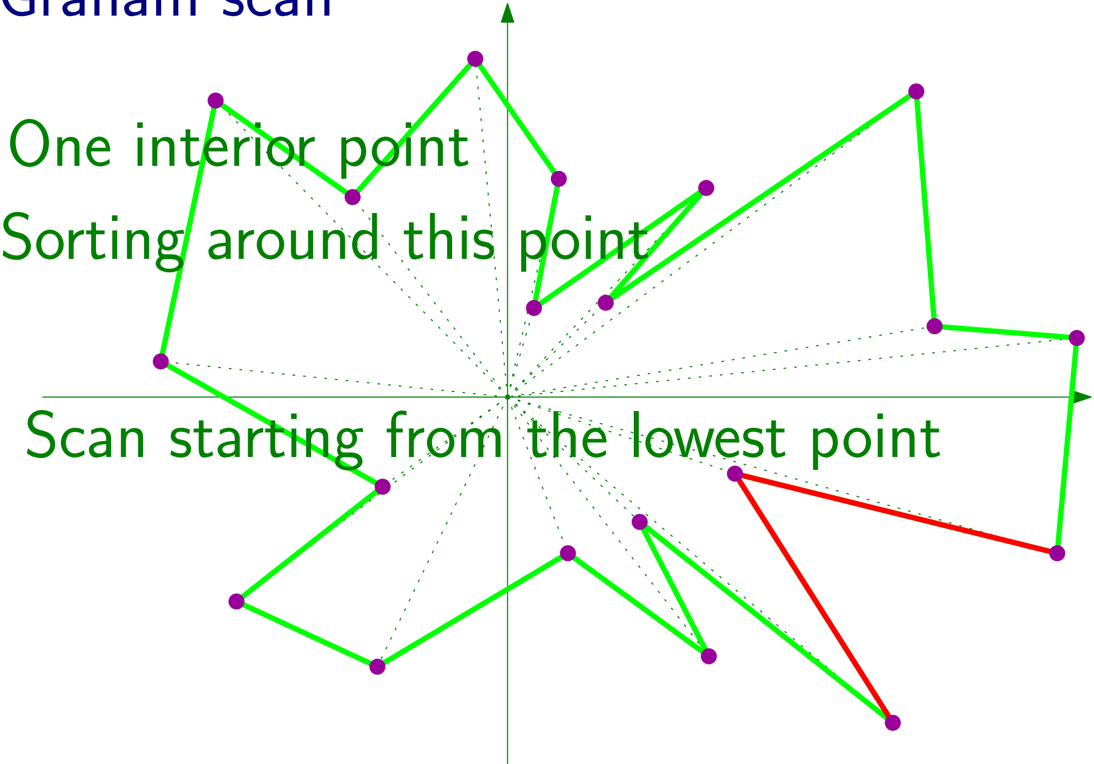


Graham scan

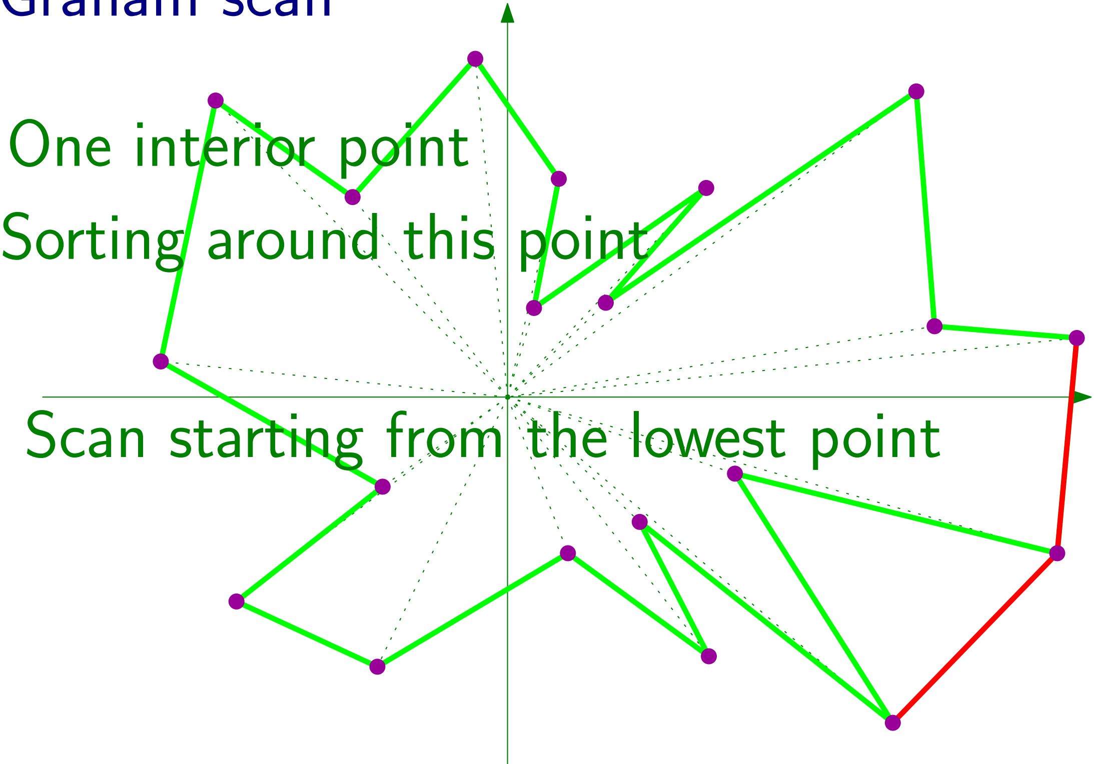


One interior point
Sorting around this point
Scan starting from the lowest point

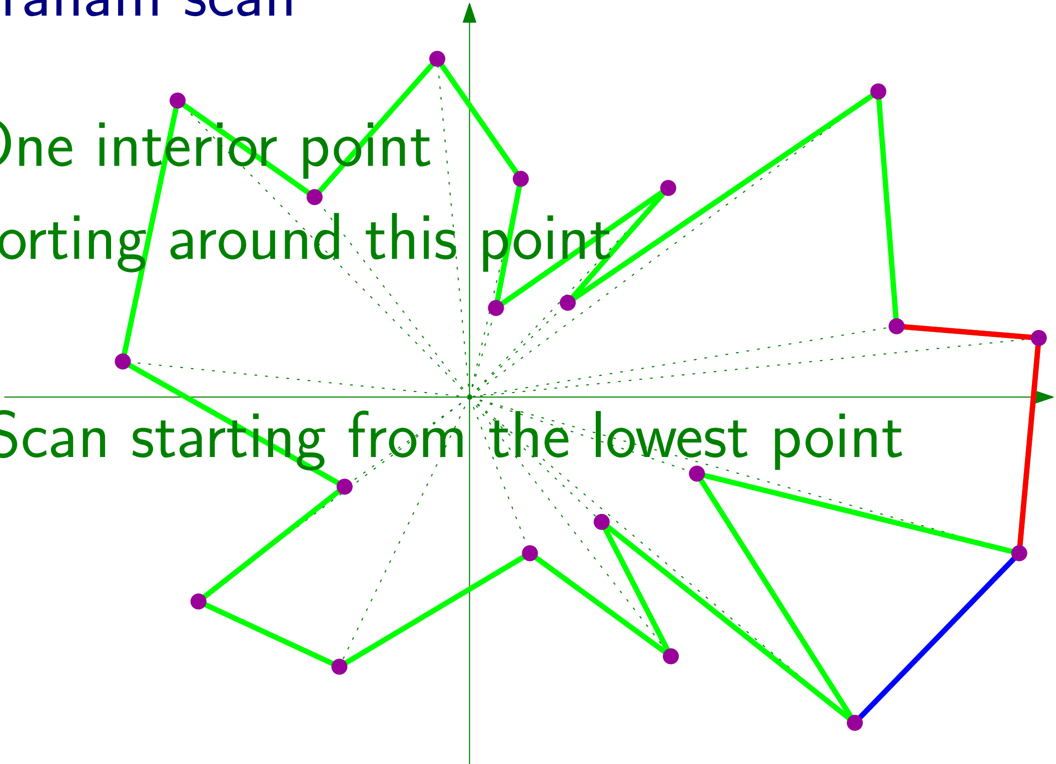
Graham scan


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

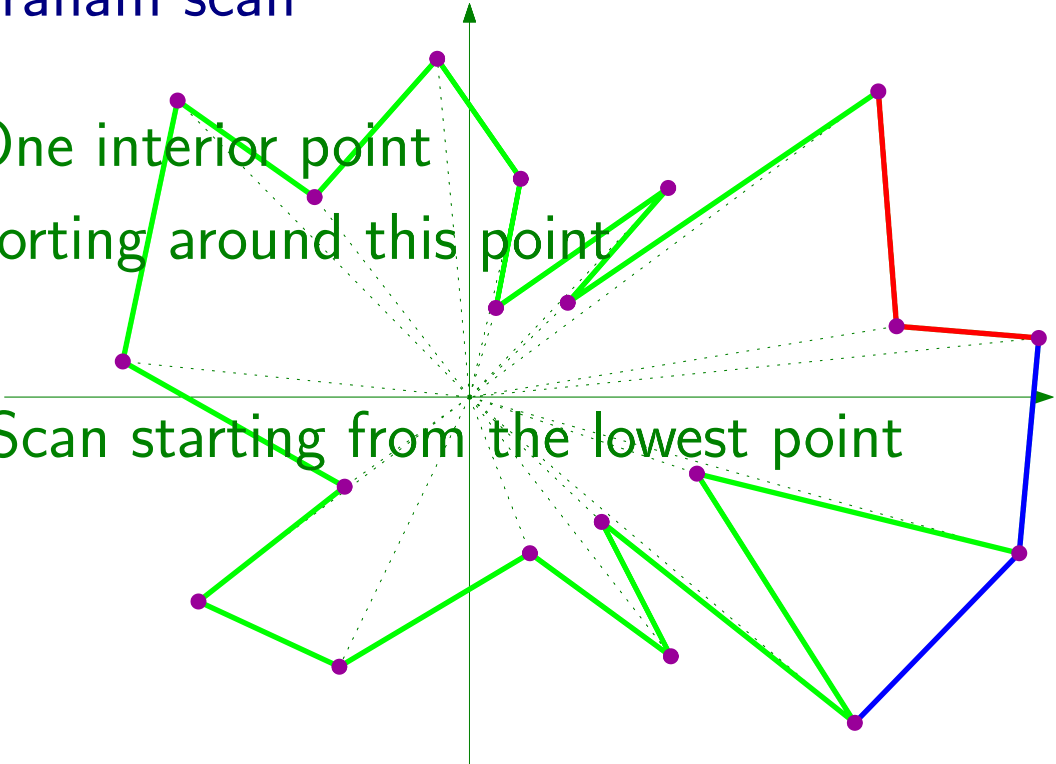


One interior point

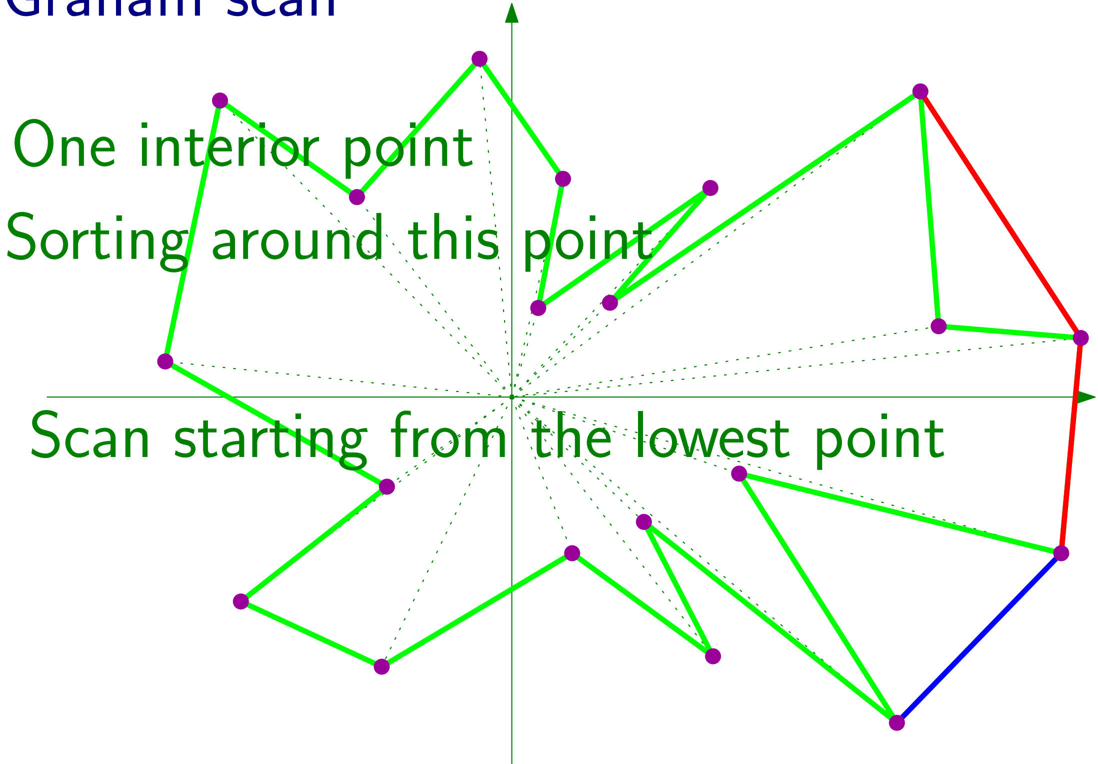
Sorting around this point

Scan starting from the lowest point

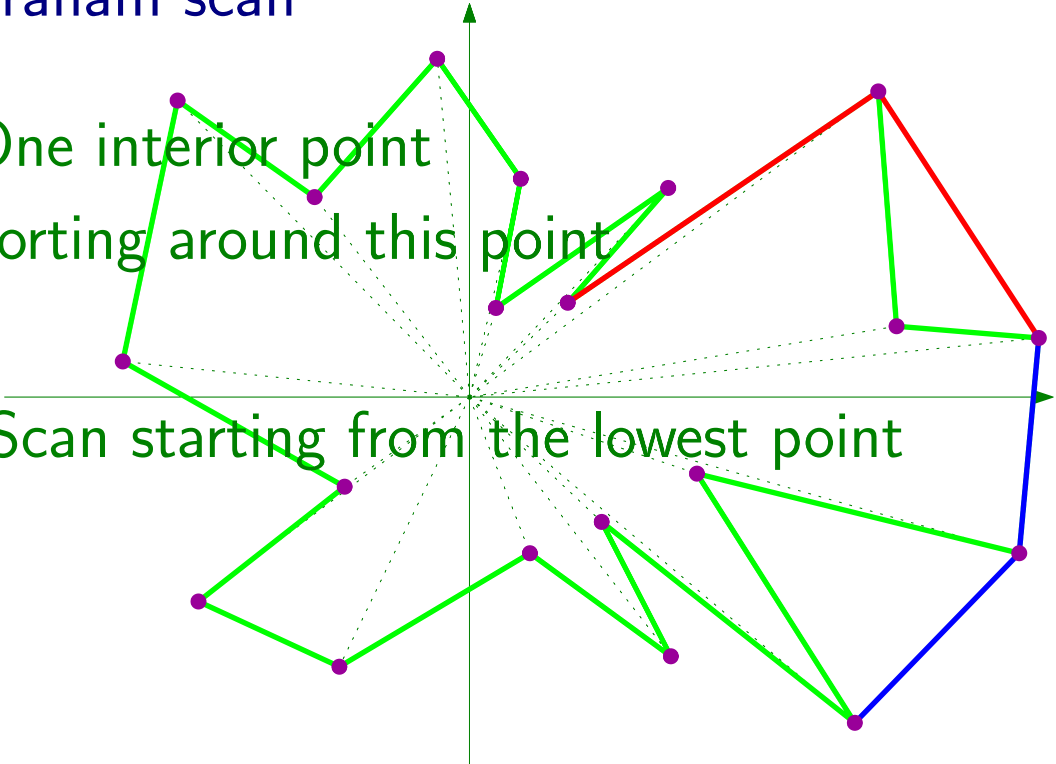
Graham scan


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

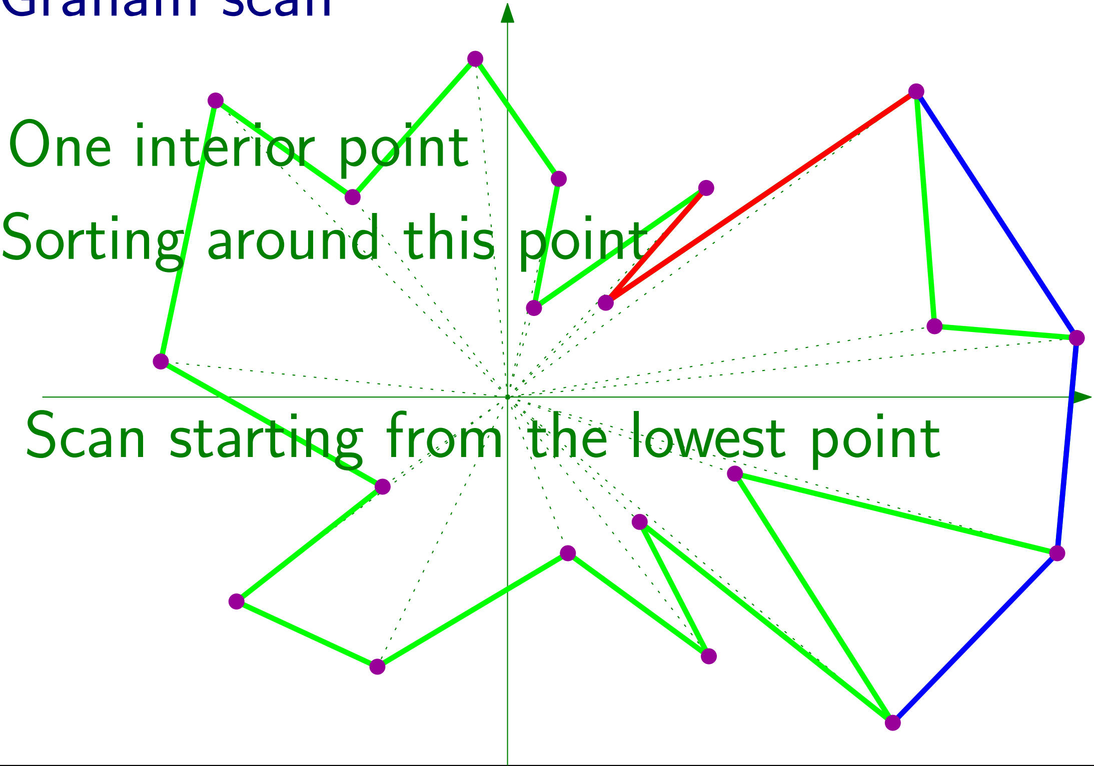


One interior point

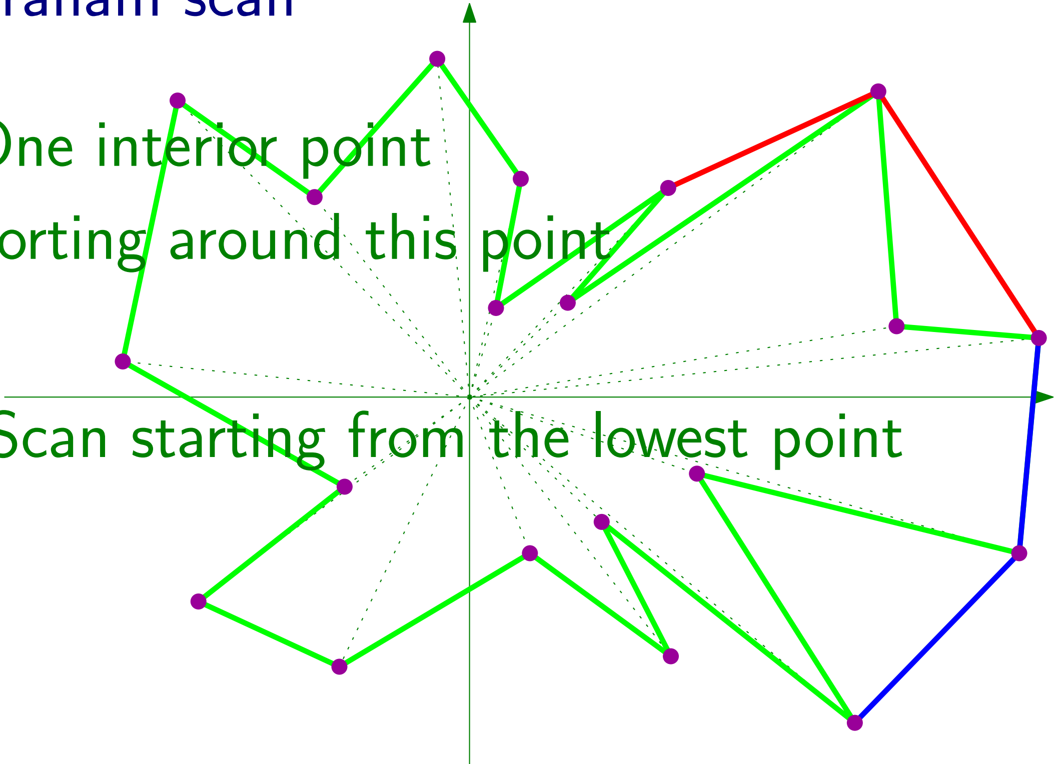
Sorting around this point

Scan starting from the lowest point

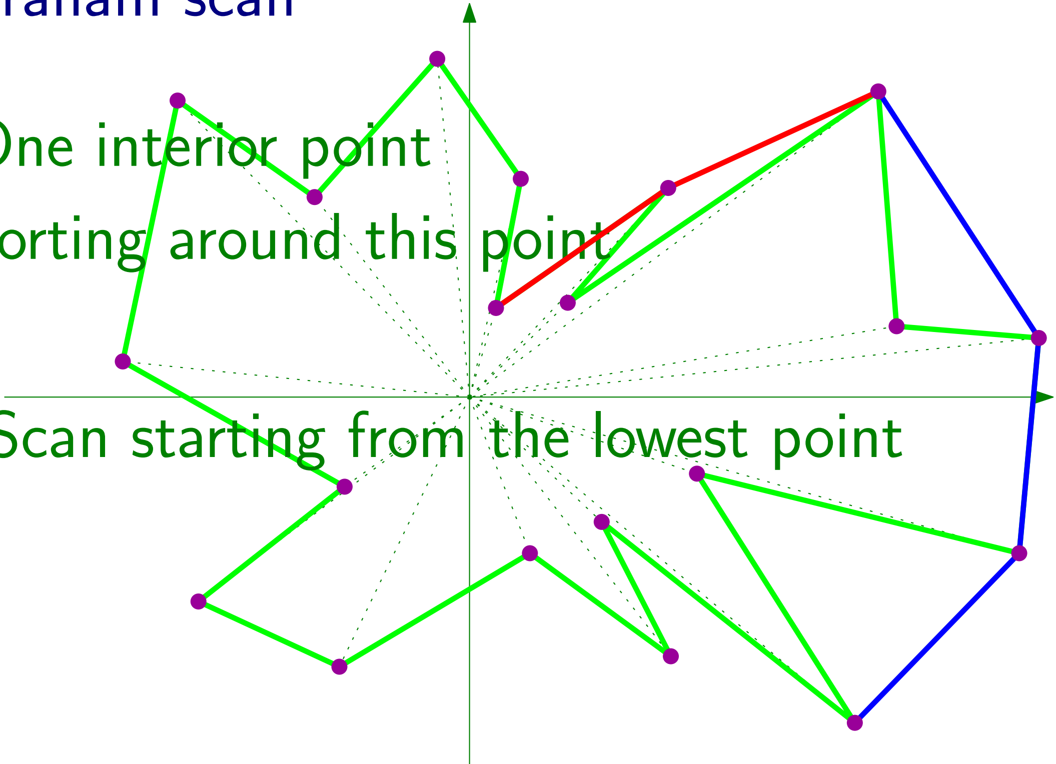
Graham scan


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

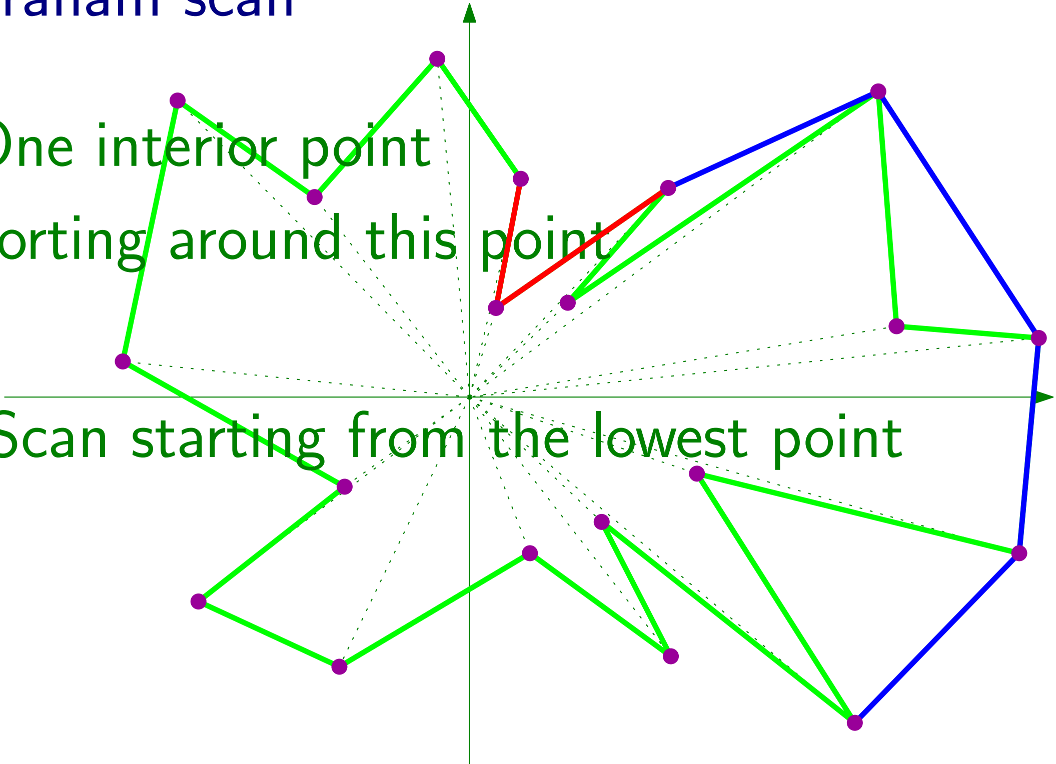
Scan starting from the lowest point

Graham scan

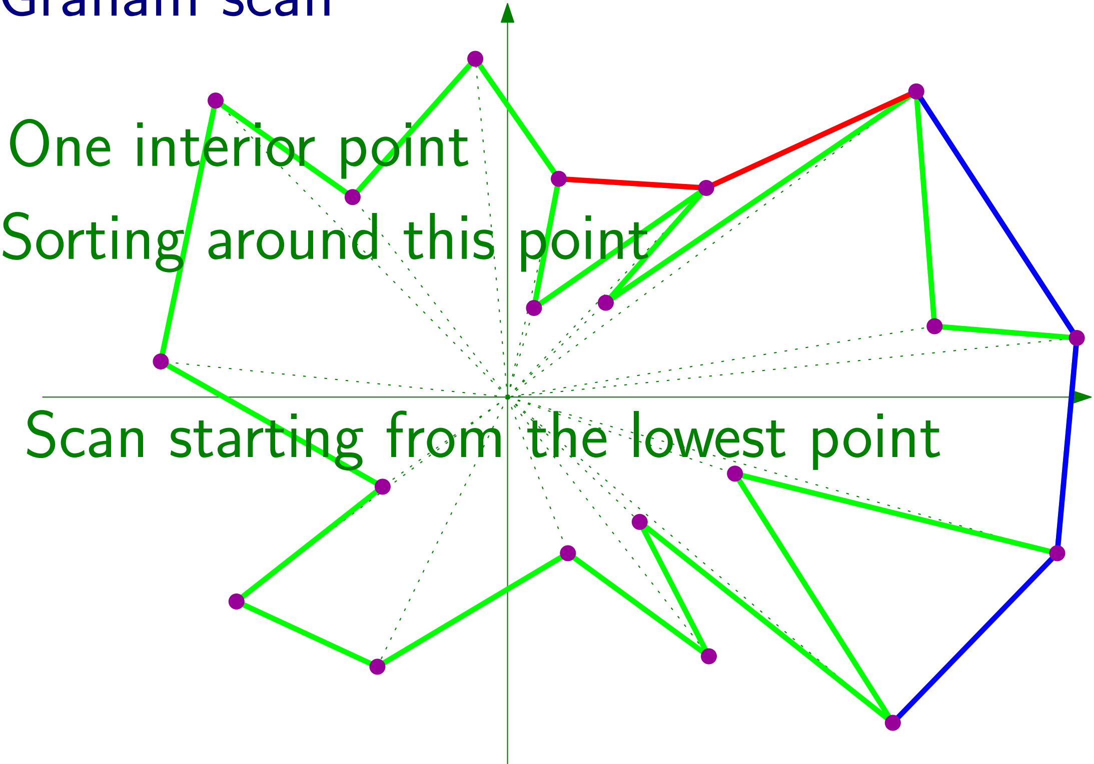
One interior point

Sorting around this point

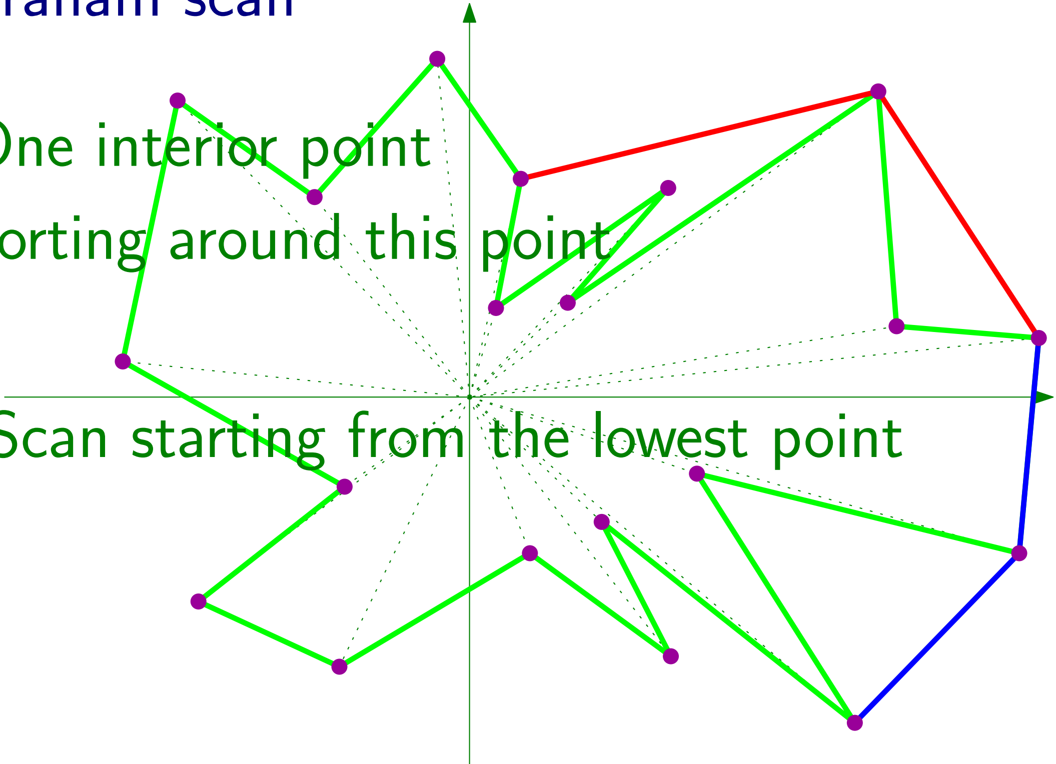
Scan starting from the lowest point



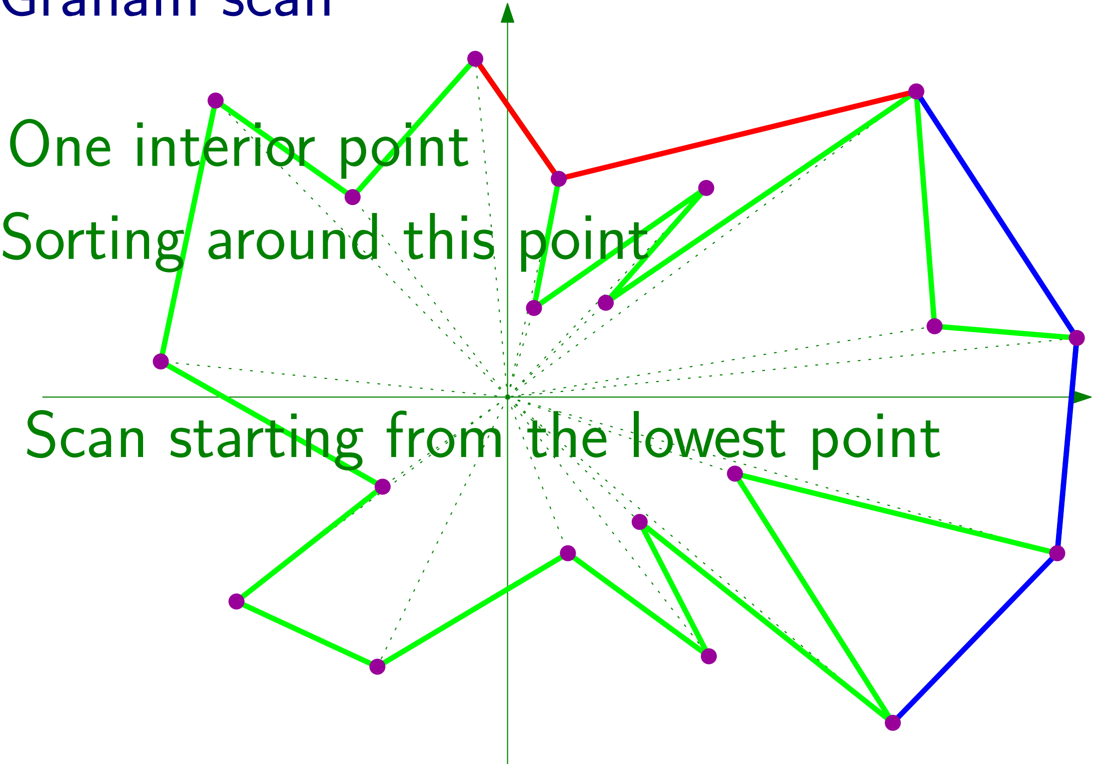
Graham scan


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

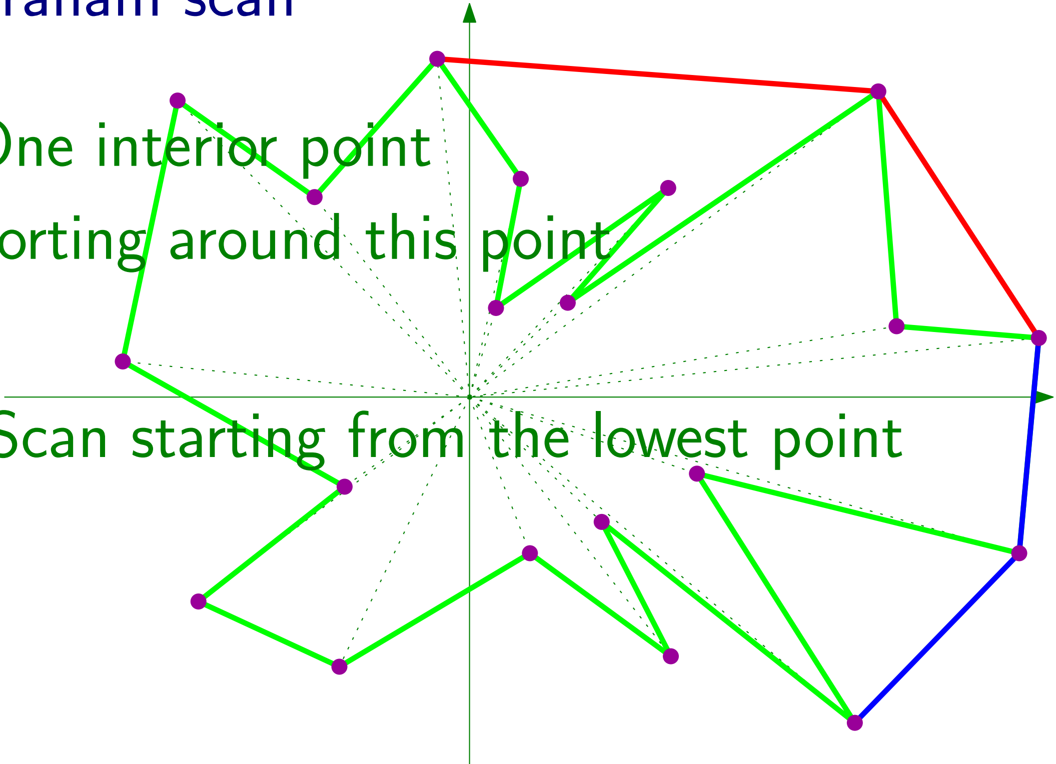

One interior point
Sorting around this point
Scan starting from the lowest point

Graham scan

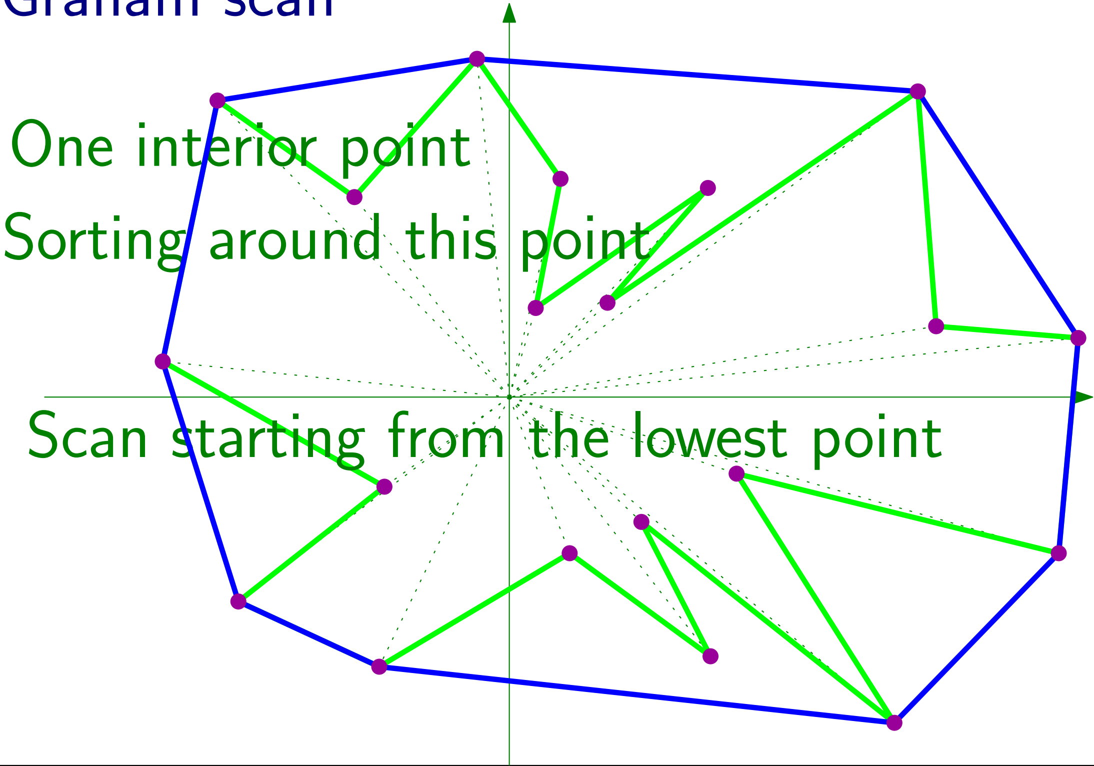
One interior point

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.

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

$$O(1)$$

$$O(n \log n)$$

$$O(n)$$

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

$$O(1)$$

$$O(n \log n)$$

$$O(n)$$

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

$$O(1)$$

$$O(n \log n)$$

$$O(n)$$

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

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

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; $O(1)$

sort S around the origin; $O(n \log n)$

$u =$ the lowest point of S; $O(n)$

$v = u$;

while $v.next \neq u$

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

        $v = v.next$;

    else

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

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

$O(n)$

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

        $v = v.next$;

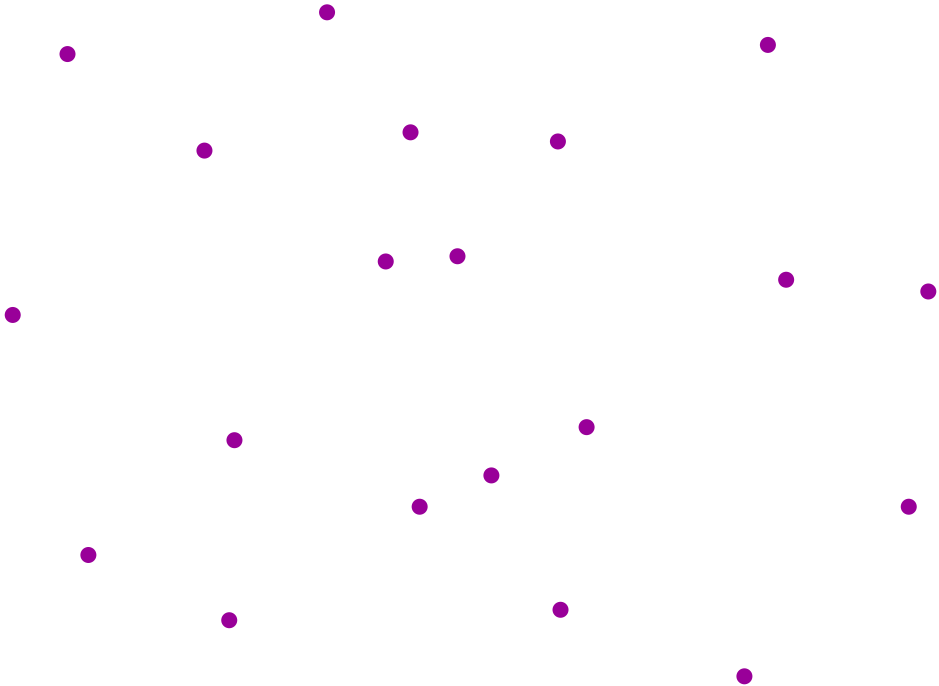
    else

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

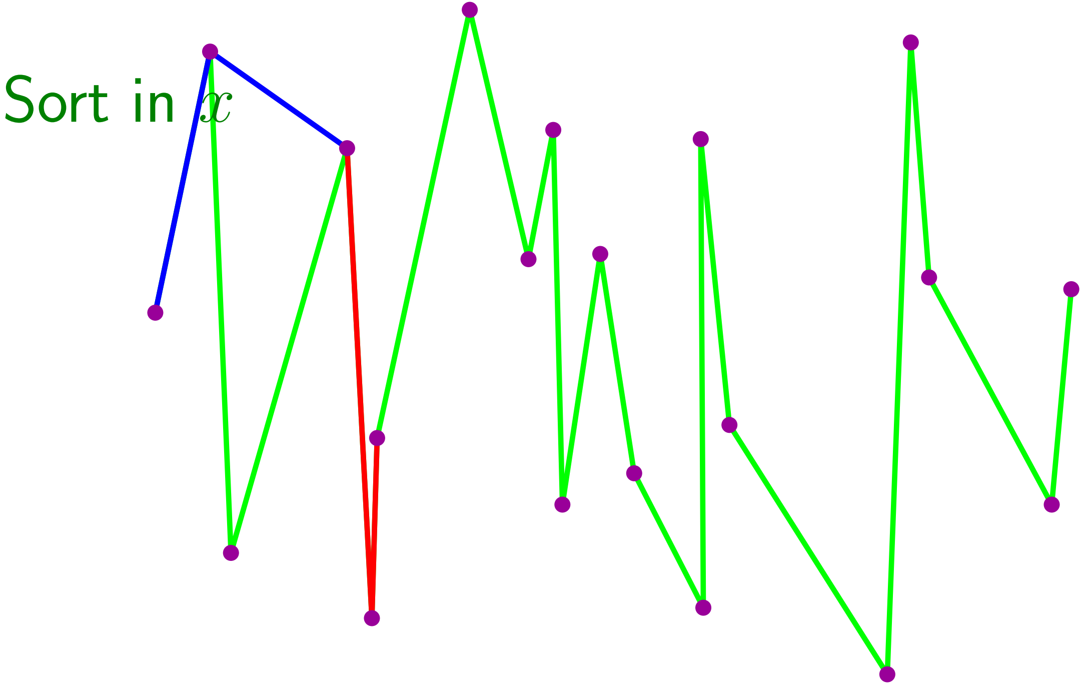
        if $v \neq u$ $v = v.previous$;

$$O(n \log n)$$

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

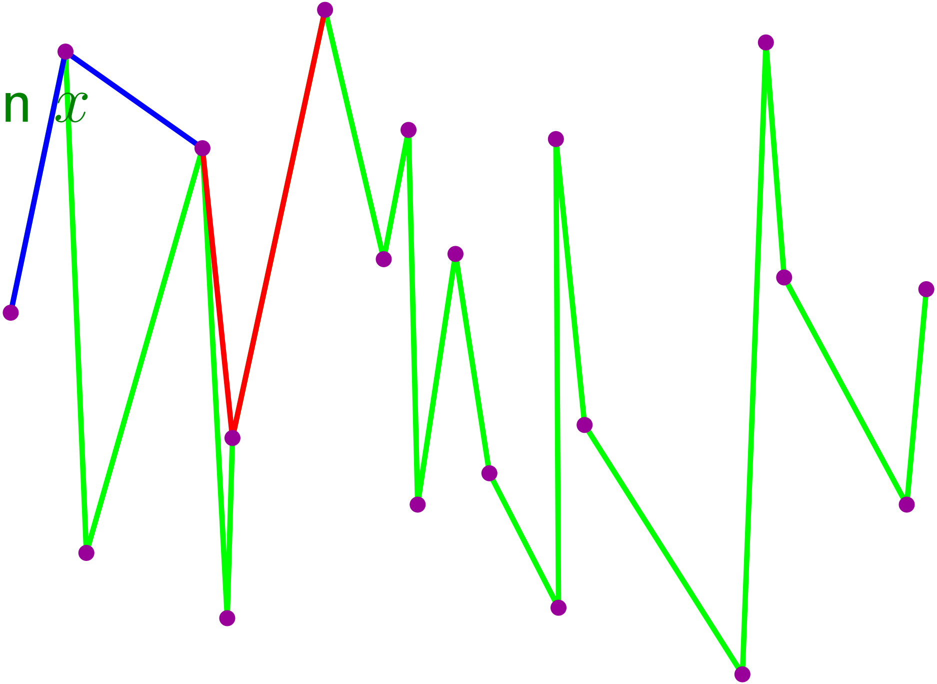


Graham alternative: origin at $y = -\infty$

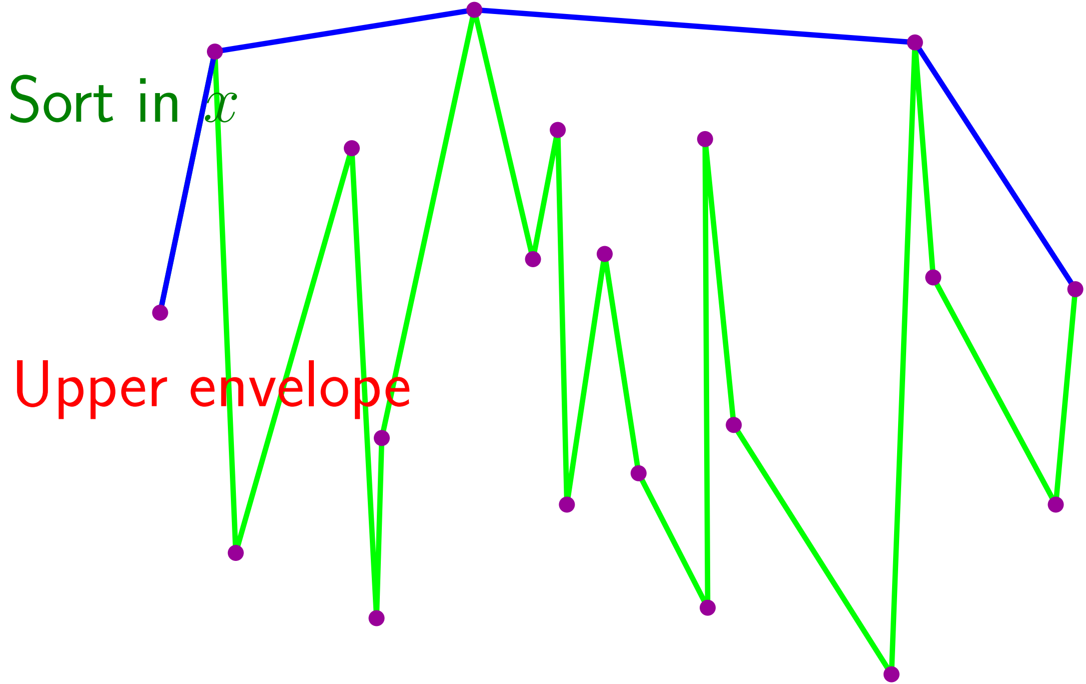


Graham alternative: origin at $y = -\infty$

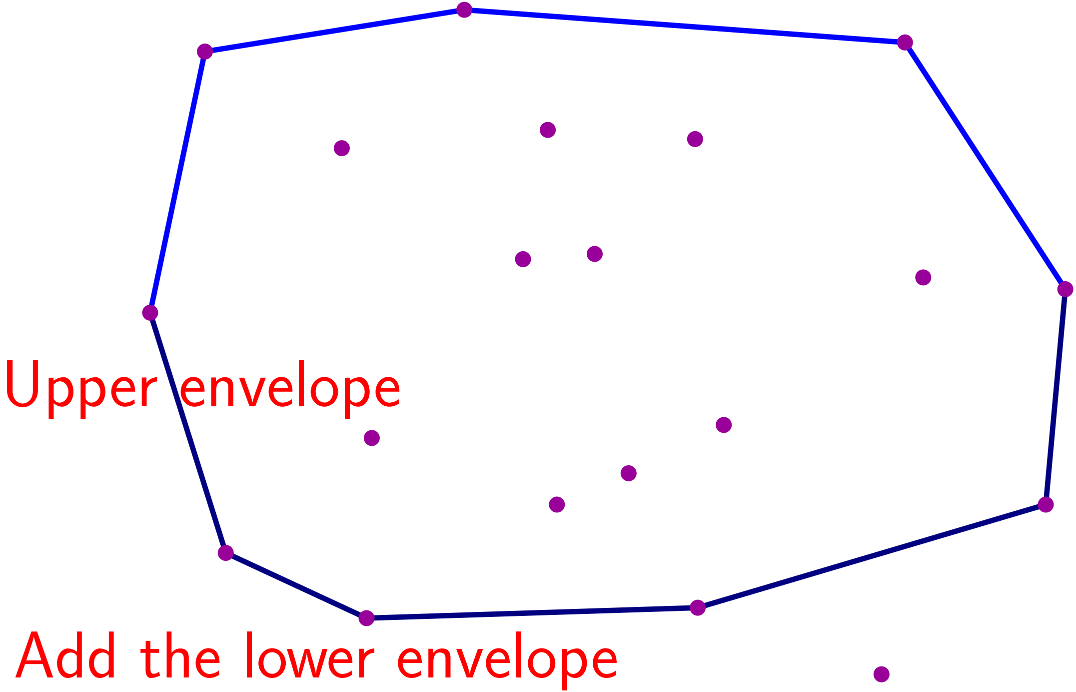
Sort in x



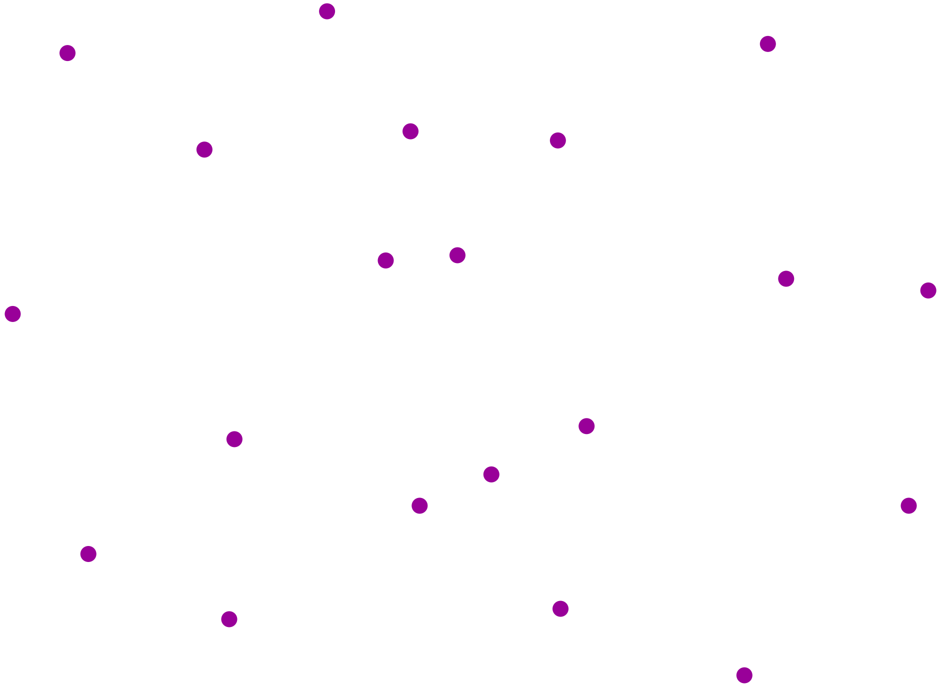
Graham alternative: origin at $y = -\infty$



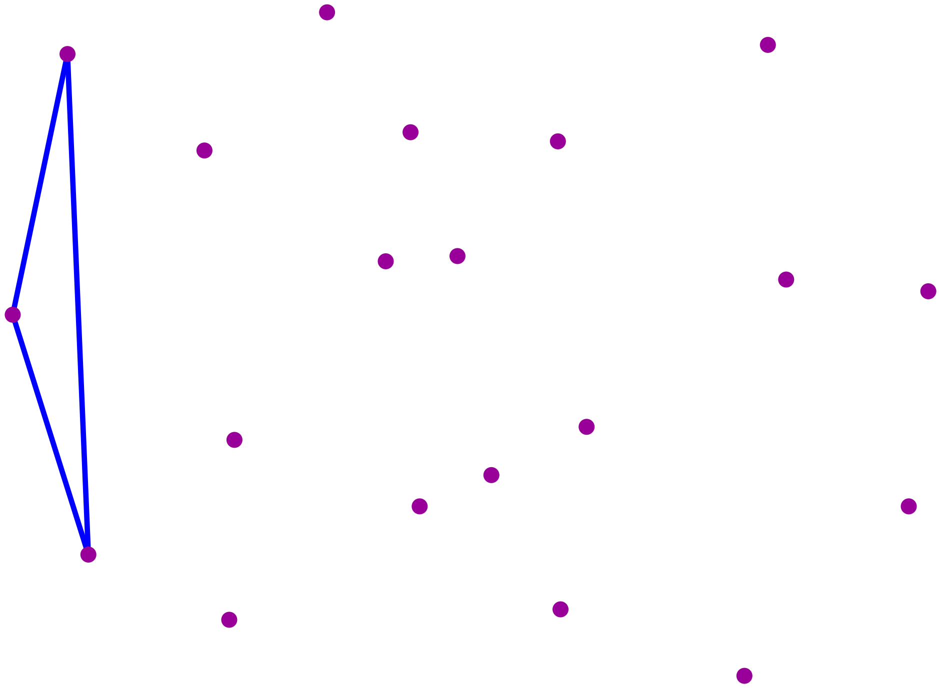
Graham alternative: origin at $y = -\infty$



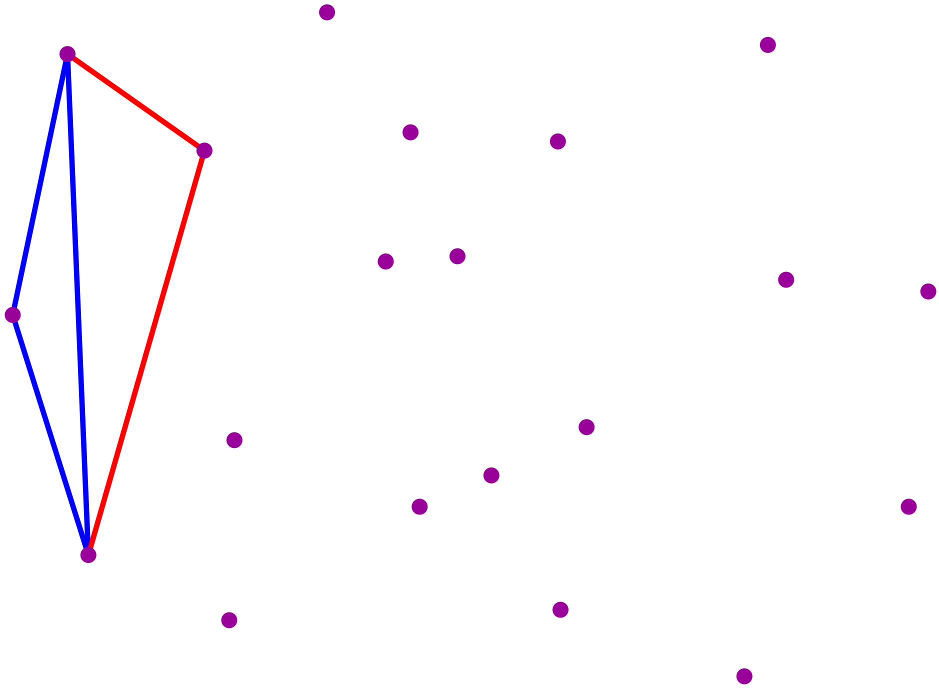
Deterministic incremental algorithm



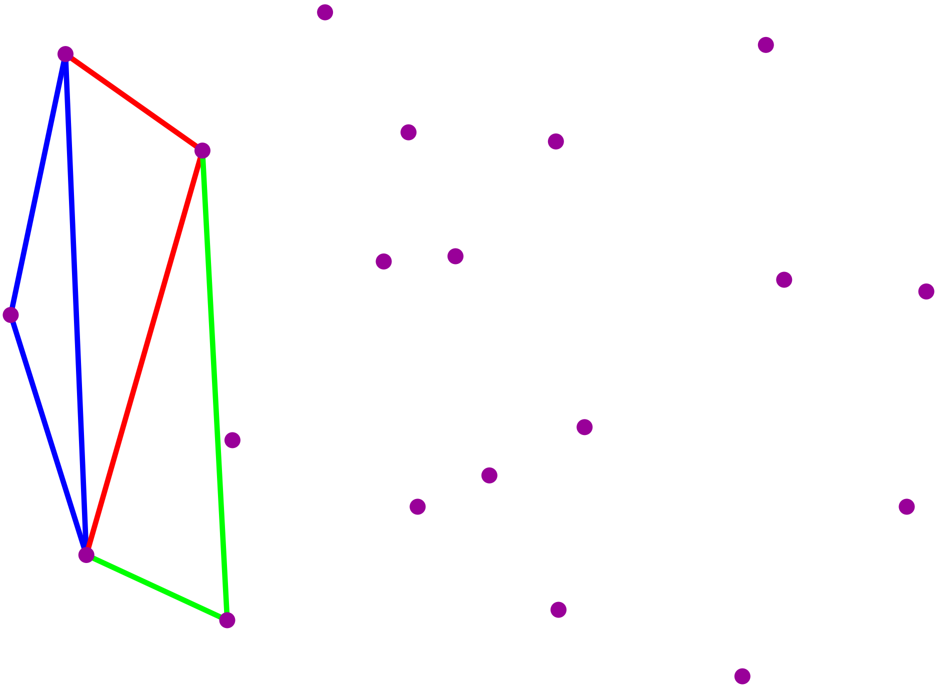
Deterministic incremental algorithm



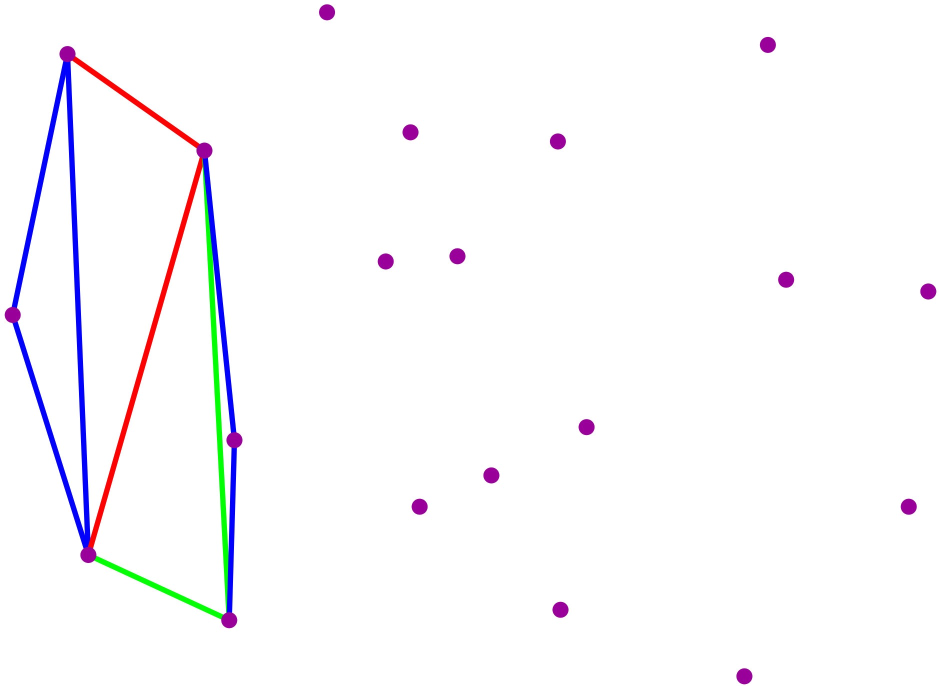
Deterministic incremental algorithm



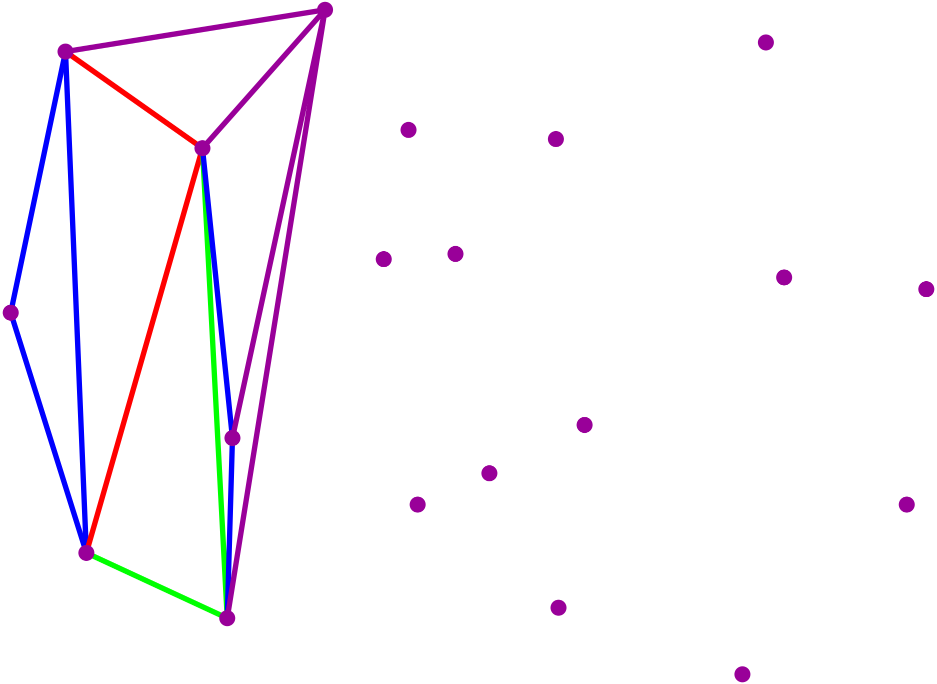
Deterministic incremental algorithm



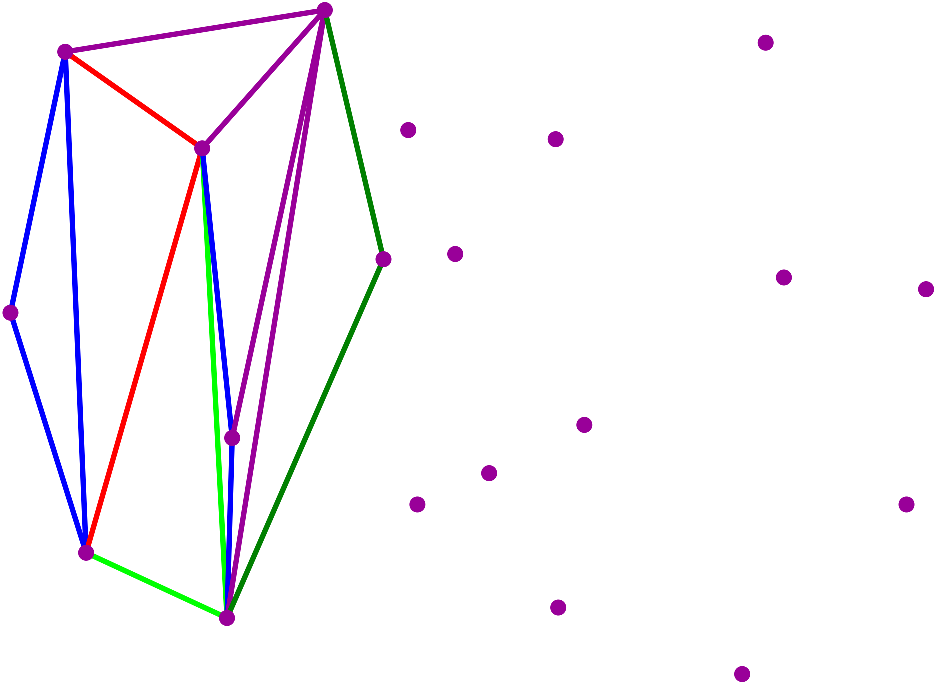
Deterministic incremental algorithm



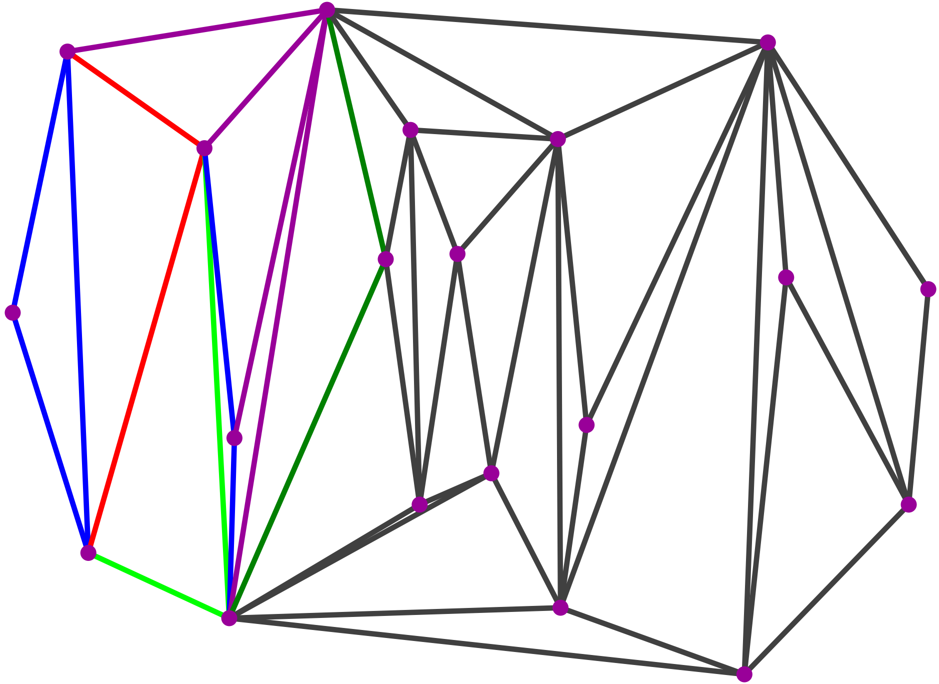
Deterministic incremental algorithm



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

        $u = u.next$;

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

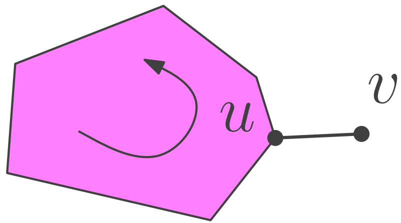
    while $(v, w, w.previous)$ turn left

        $w = w.previous$;

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

    $u = v$;

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

$u = u.next$;

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

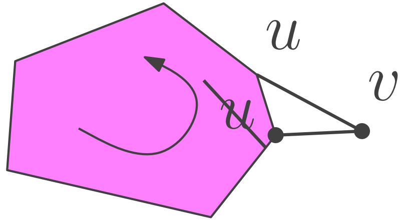
while $(v, w, w.previous)$ turn left

$w = w.previous$;

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

$u = v$;

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

    $u = u.next$;

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

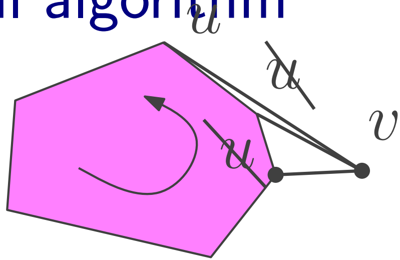
  while $(v, w, w.previous)$ turn left

    $w = w.previous$;

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

  $u = v$;

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

        $u = u.next;$

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

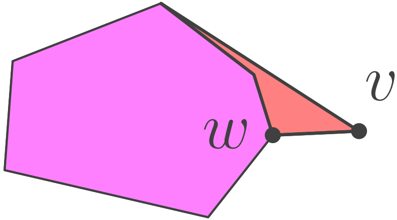
    while $(v, w, w.previous)$ turn left

        $w = w.previous;$

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

    $u = v;$

Deterministic incremental algorithm



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

        $u = u.next;$

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

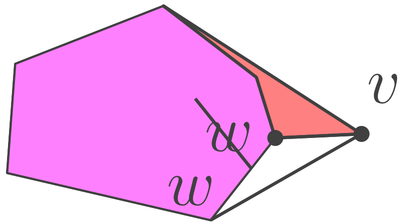
    while $(v, w, w.previous)$ turn left

        $w = w.previous;$

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

    $u = v;$

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

        $u = u.next$;

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

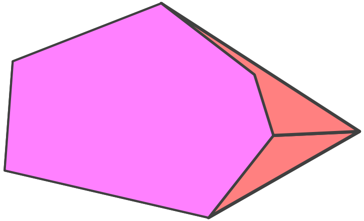
    while $(v, w, w.previous)$ turn left

        $w = w.previous$;

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

    $u = v$;

Deterministic incremental algorithm



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;

$$O(n \log n)$$

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

$u = u.next$;

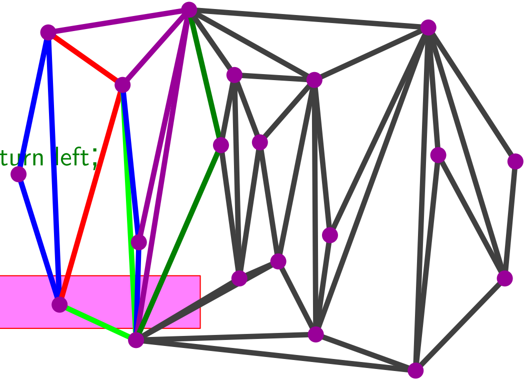
$v.next = u$; $u.previous = v$;

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

$w = w.previous$;

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

$w = v$;



Draw an edge in the triangulation

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

$u = u.next$;

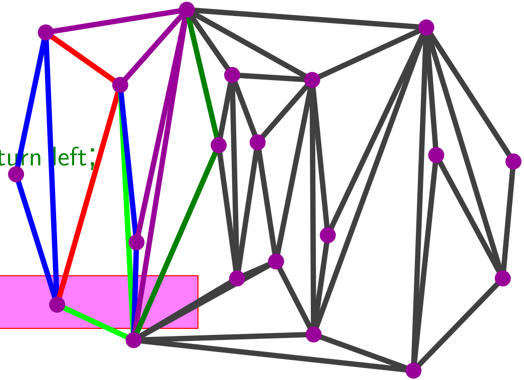
$v.next = u$; $u.previous = v$;

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

$w = w.previous$;

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

$u = v$;



nb of edges $\simeq 3n$

Draw an edge in the triangulation

Deterministic incremental algorithm

Complexity

$O(n \log n)$

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

$u = u.next$;

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

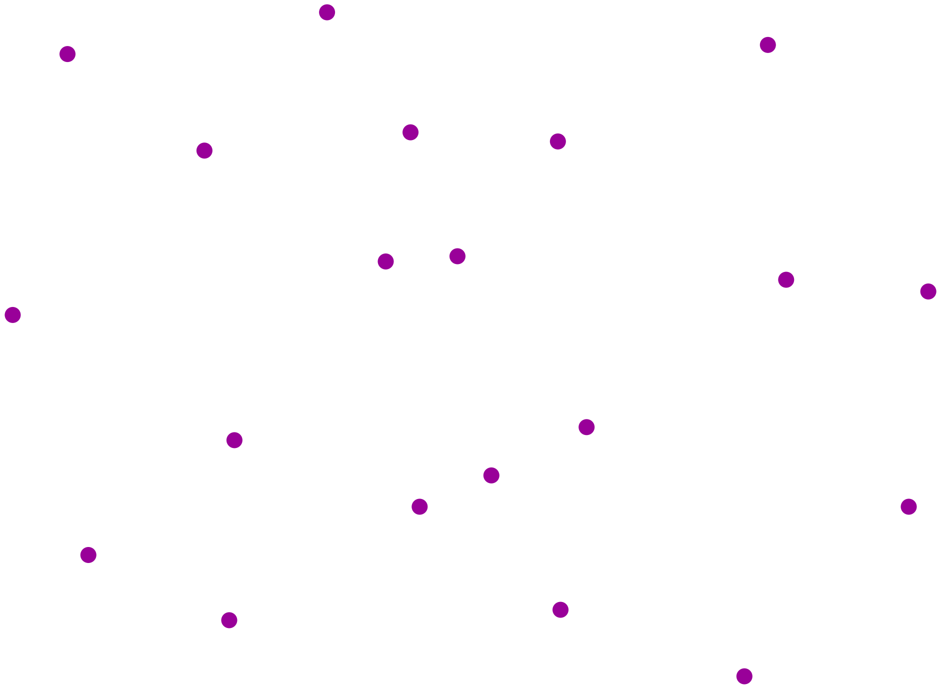
while $(v, w, w.previous)$ turn left

$w = w.previous$;

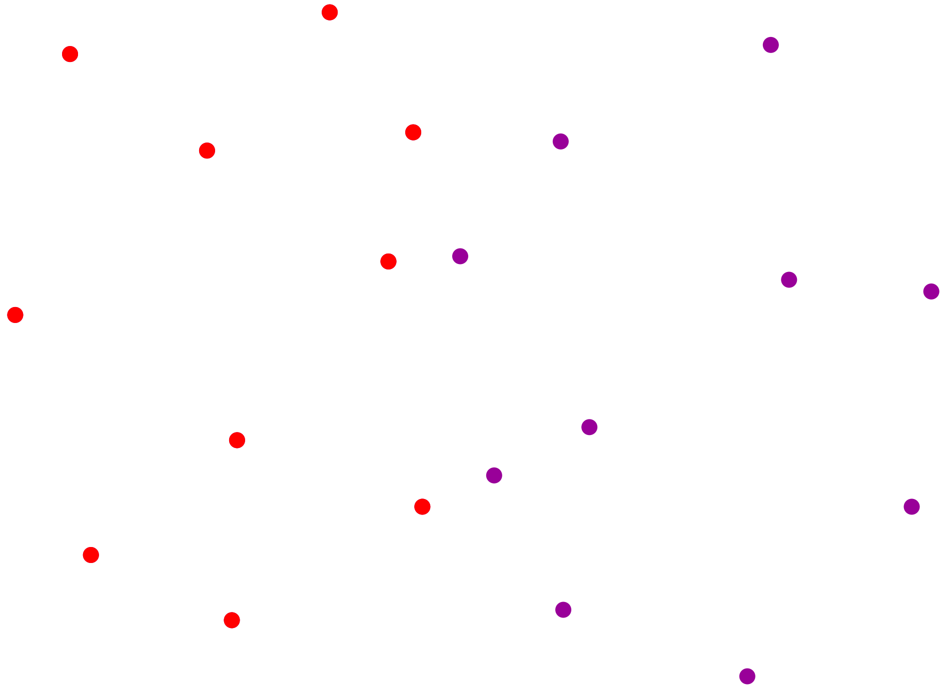
$v.previous = w$; $w.next = v$;

$u = v$;

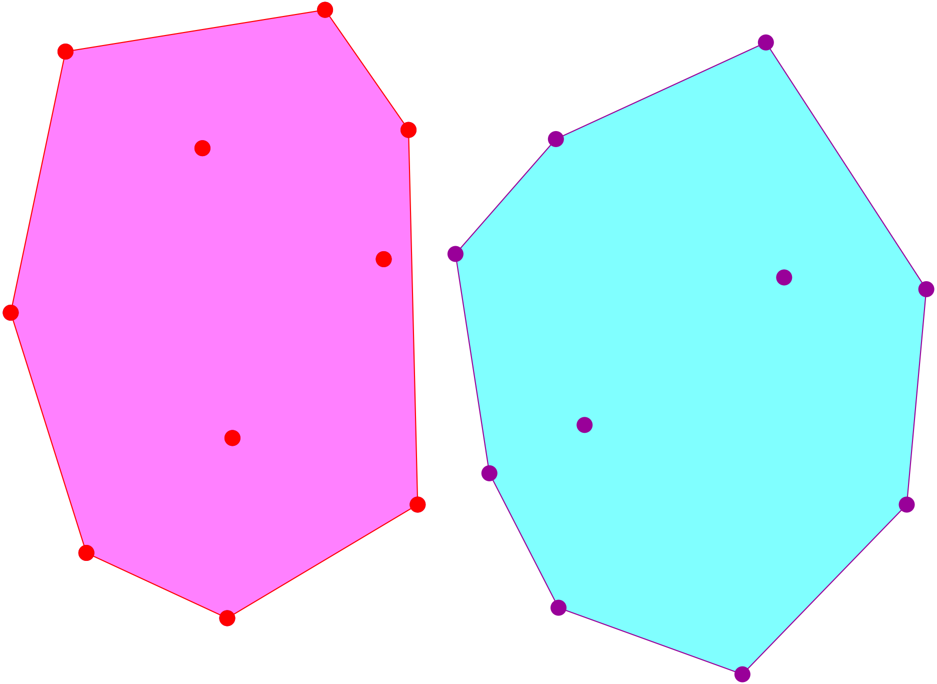
Divide & conquer algorithm



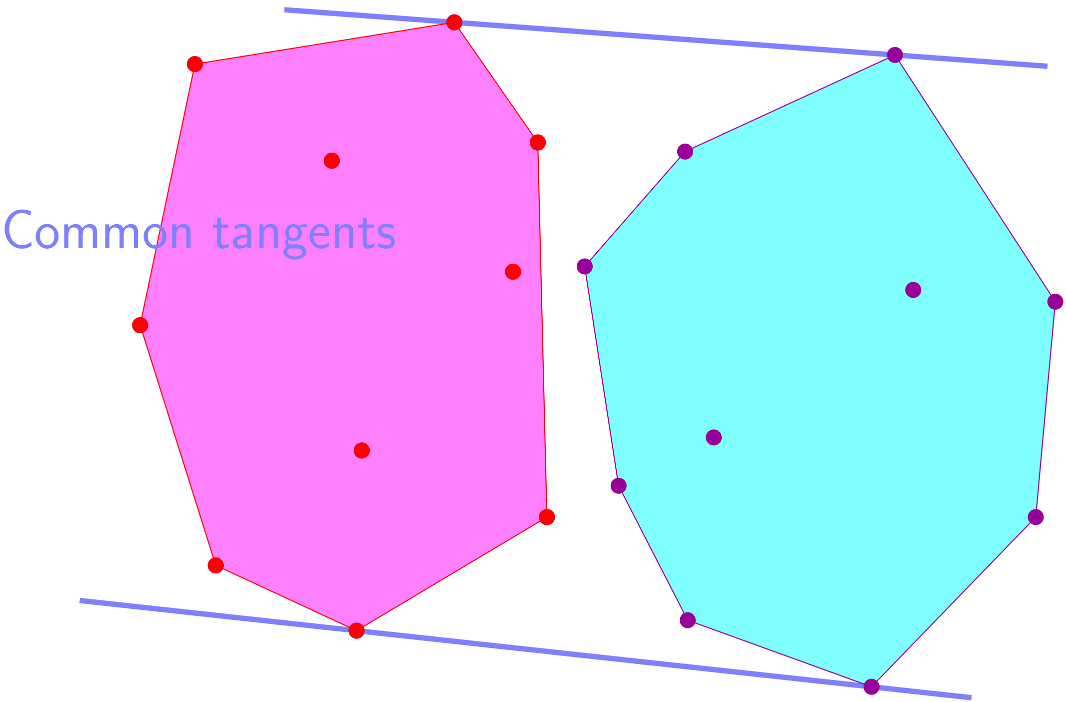
Divide & conquer algorithm



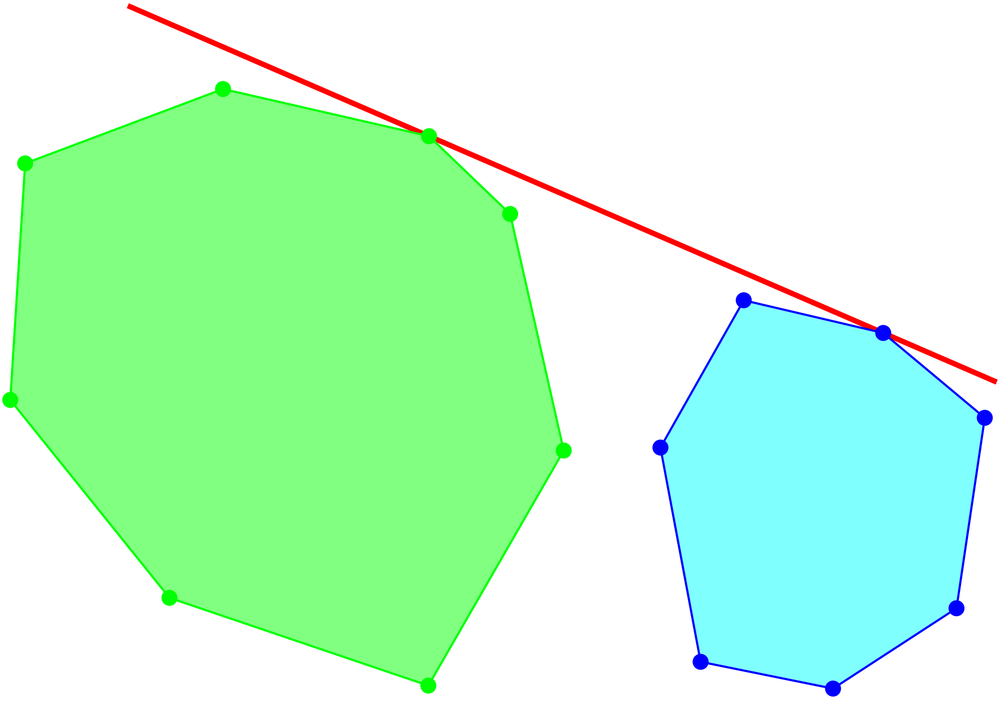
Divide & conquer algorithm



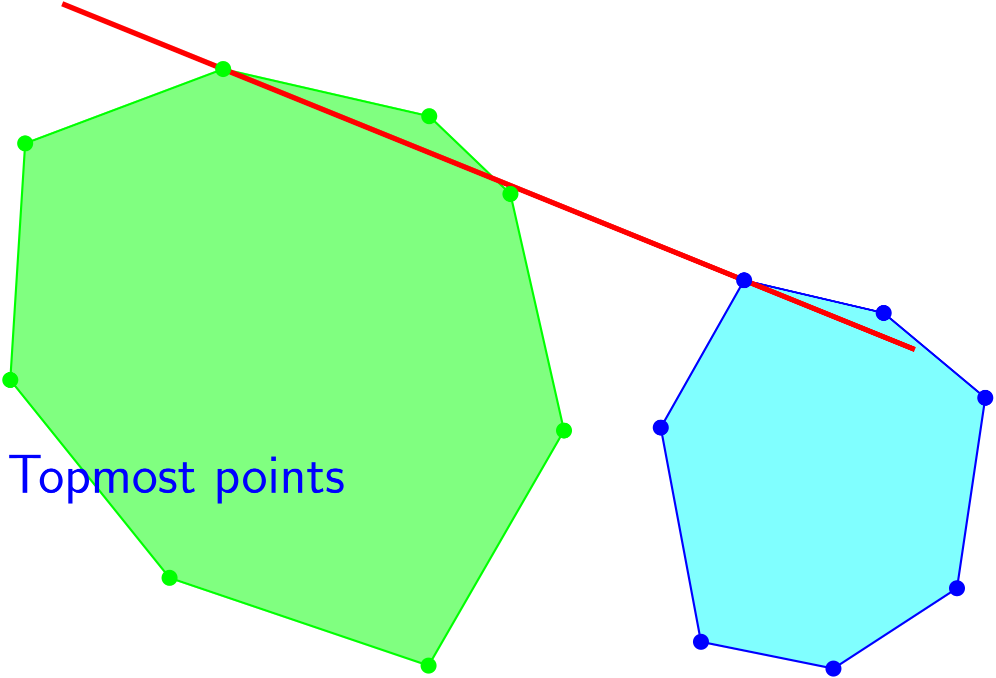
Divide & conquer algorithm



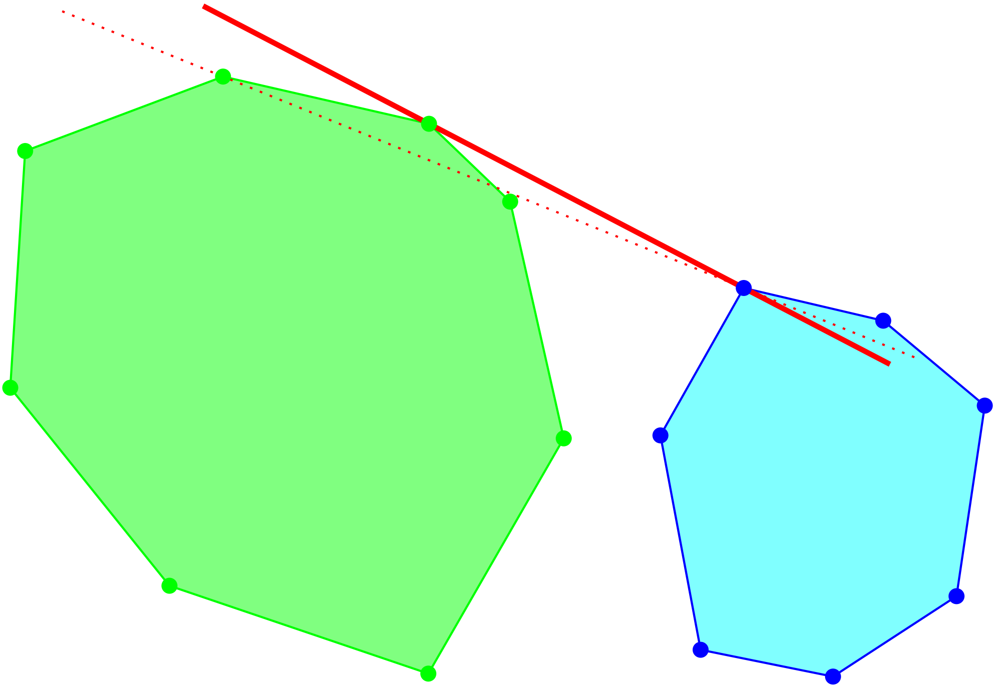
Upper tangent



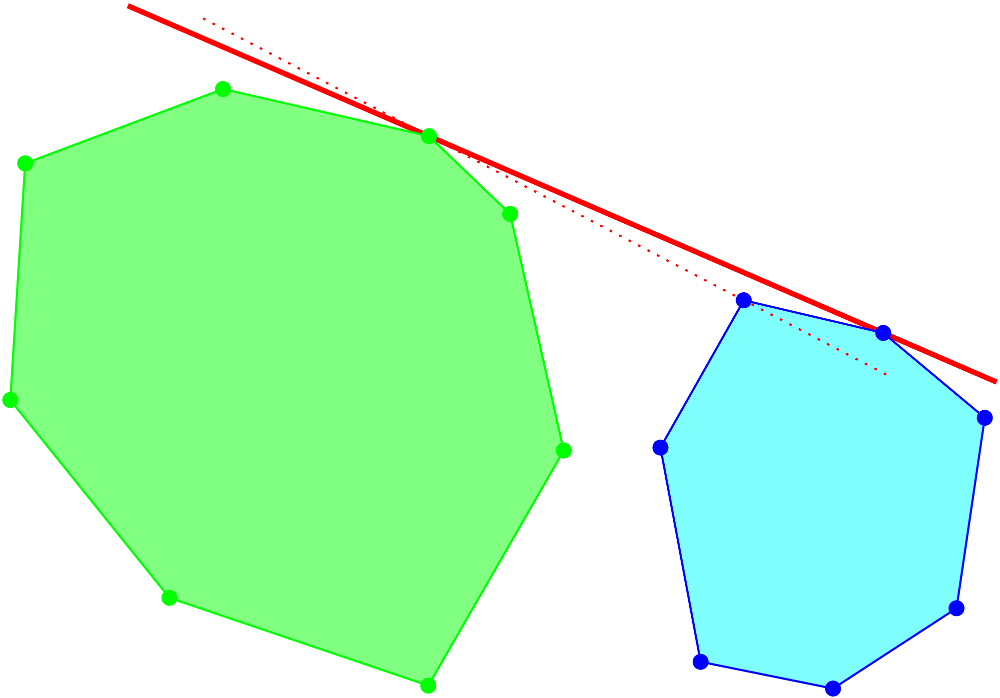
Upper tangent



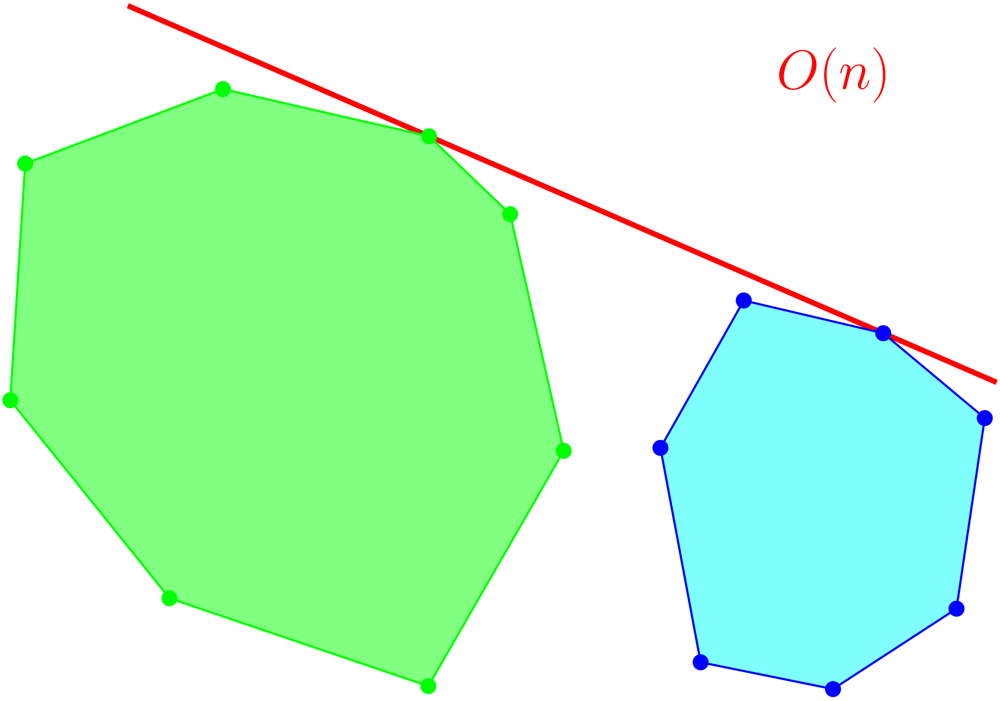
Upper tangent



Upper tangent



Upper tangent



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 $O(n)$

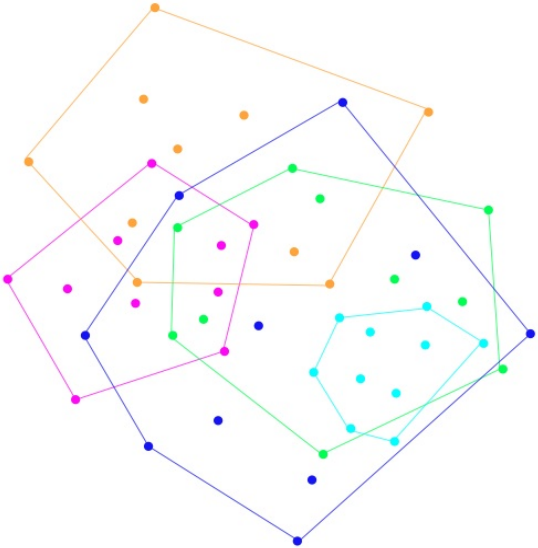
Balanced partition

(preprocessing in $O(n \log n)$

CHAN's ALGORITHM

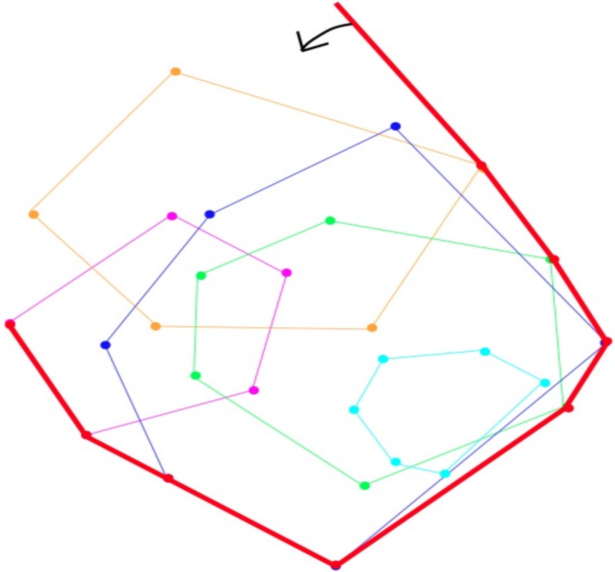
CHAN's ALGORITHM

Compute the CH of k subsets of P



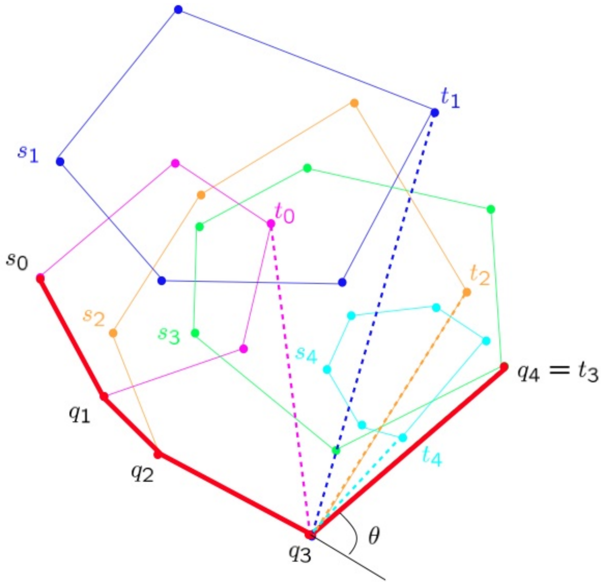
CHAN's ALGORITHM

Wrap around the CH of the k subsets



CHAN's ALGORITHM

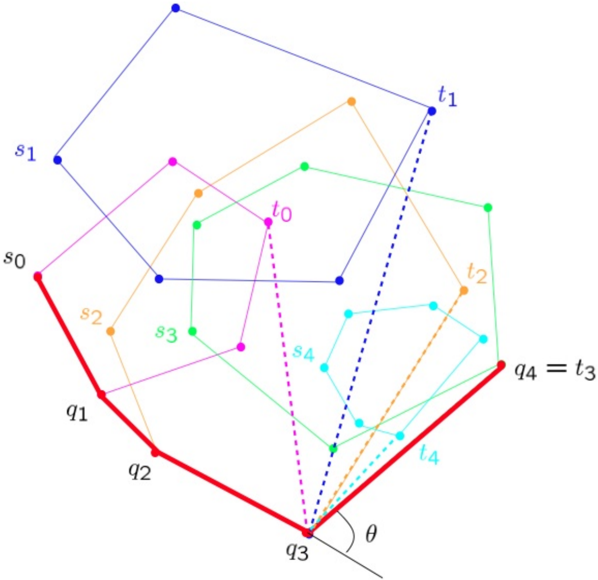
Compute the CH of $k = n/m$ subsets of size m:
each in $O(m \log m)$, all in $k/m \times O(m \log m) = O(n \log m)$



CHAN's ALGORITHM

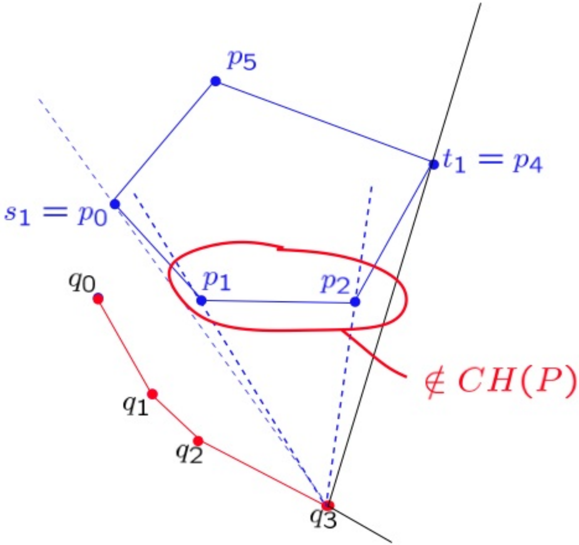
Compute the CH of $k = n/m$ subsets of size m:
each in $O(m \log m)$, all in $k/m \times O(m \log m) = O(n \log m)$

Find the s_i: $O(n)$



CHAN's ALGORITHM

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



CHAN's ALGORITHM

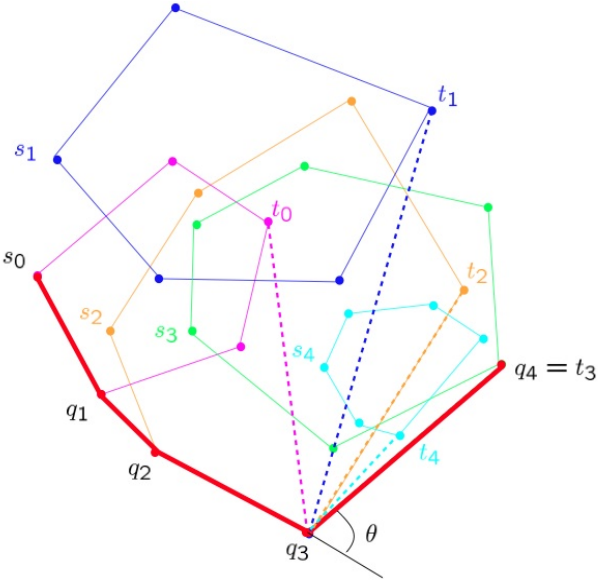
Compute the CH of $k = n/m$ subsets of size m:
each in $O(m \log m)$, all in $k/m \times O(m \log m) = O(n \log m)$

Find the s_i: $O(n)$

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

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

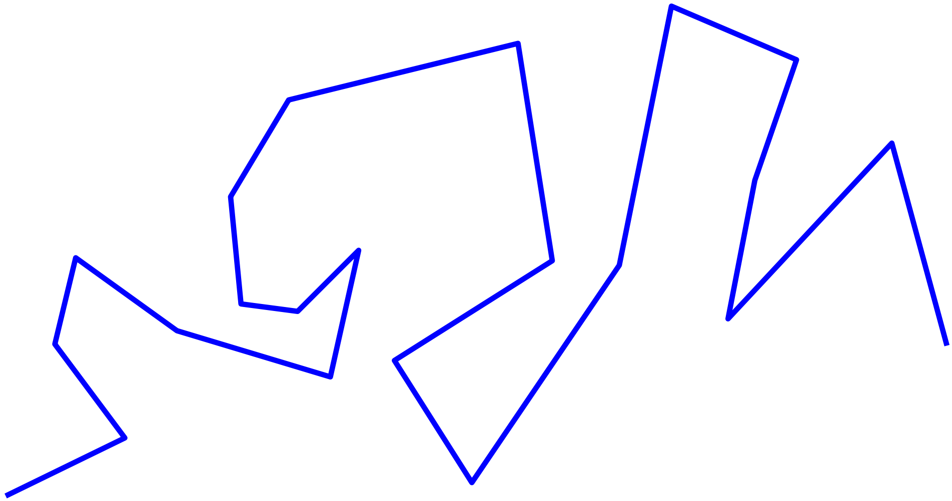
Hull(P,m,H)
  Do H wrapping steps
    If the last wrap comes back to the first point then return success
    Else return incomplete

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

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

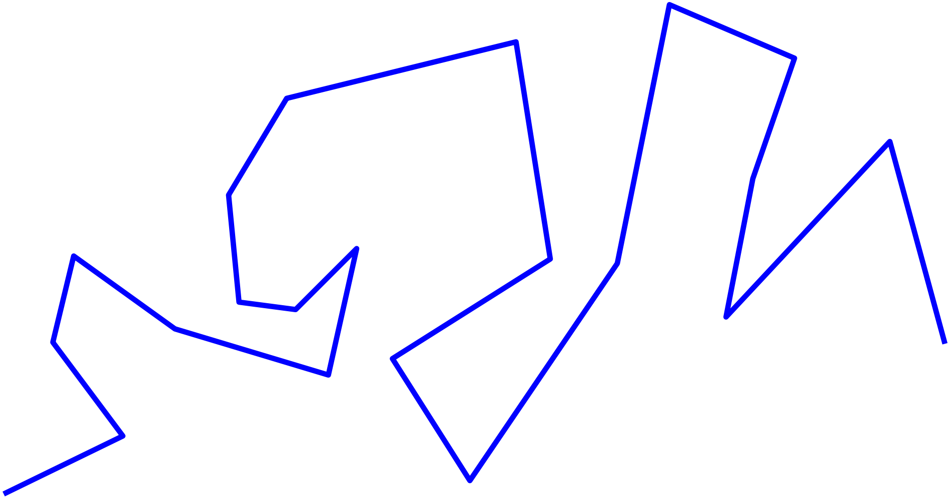
Complexity:
  Nb of iterations $= O(\log \log h)$
  Cost of i^{th} iterations $= O(n \log H) = O(n2^i)$
  Total: $O(\sum_{i=1}^{\log \log n} n2^i) = O(n2^{\log \log n+1}) = O(n \log h)$

Special case: simple polygonal line



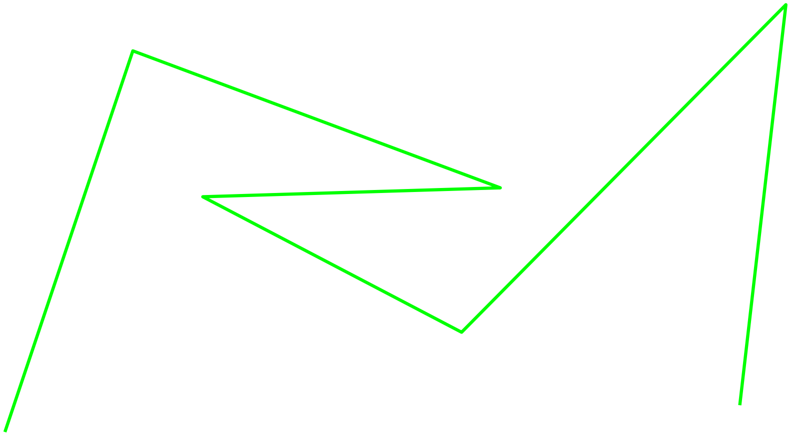
Special case: simple polygonal line

already seen: monotone polyline in Graham's scan



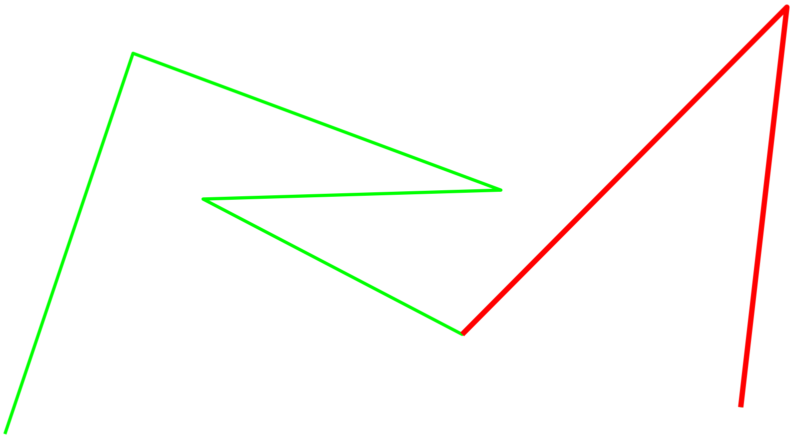
Special case: simple polygonal line

Graham does not work



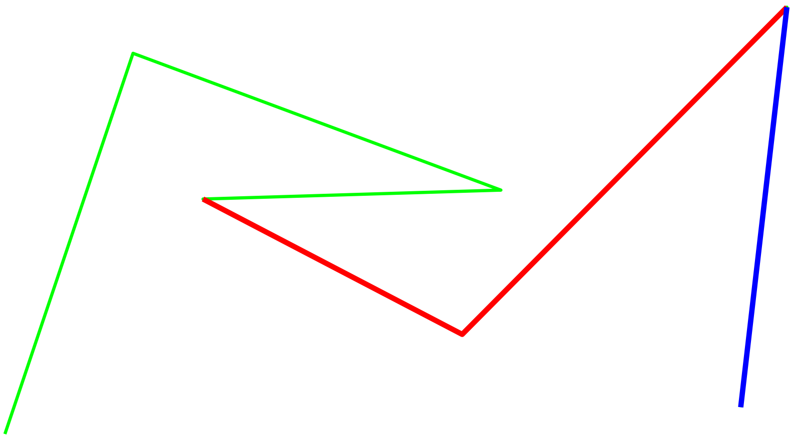
Special case: simple polygonal line

Graham does not work



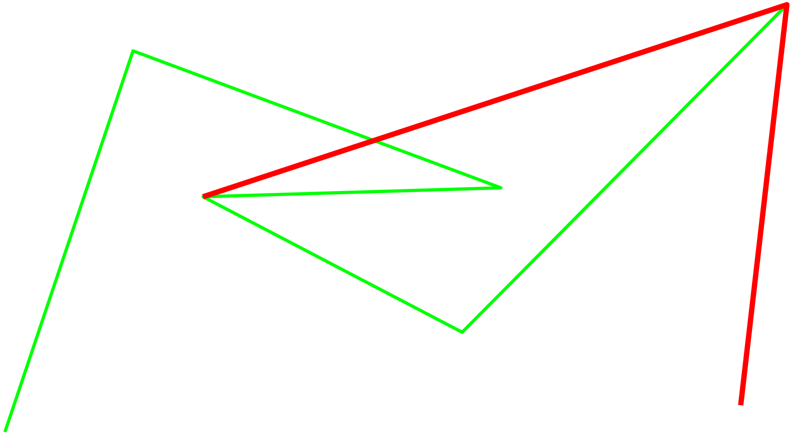
Special case: simple polygonal line

Graham does not work



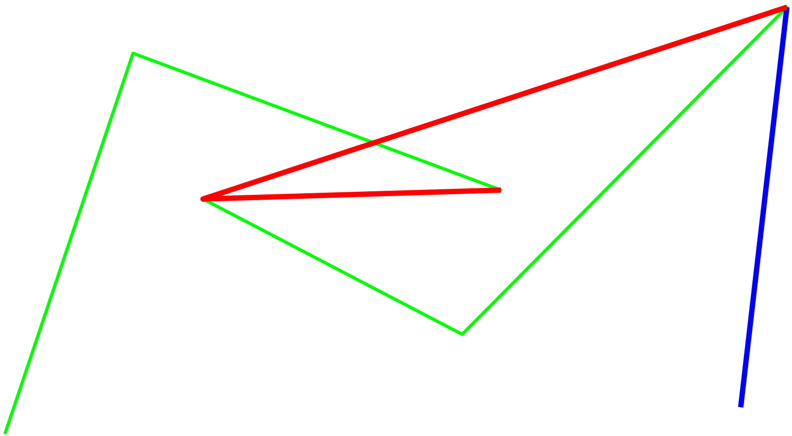
Special case: simple polygonal line

Graham does not work



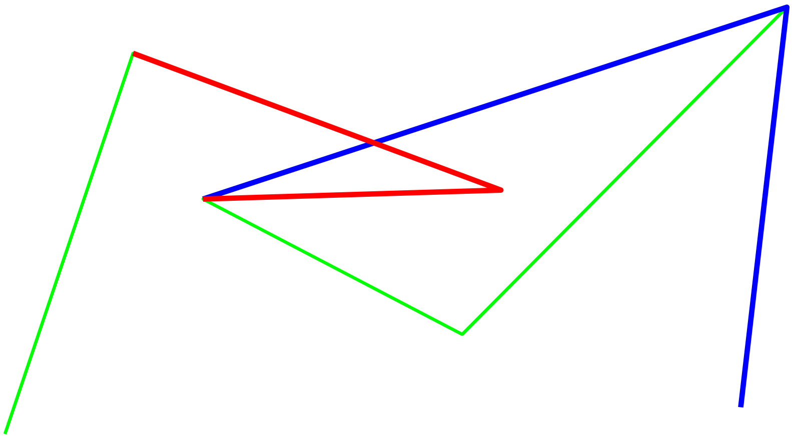
Special case: simple polygonal line

Graham does not work



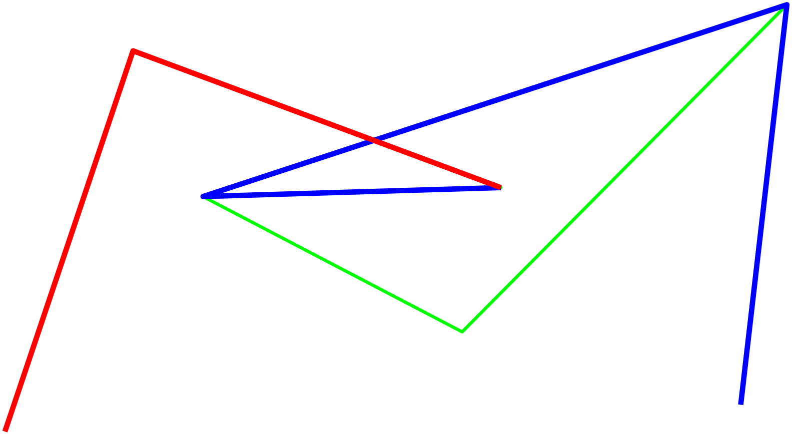
Special case: simple polygonal line

Graham does not work



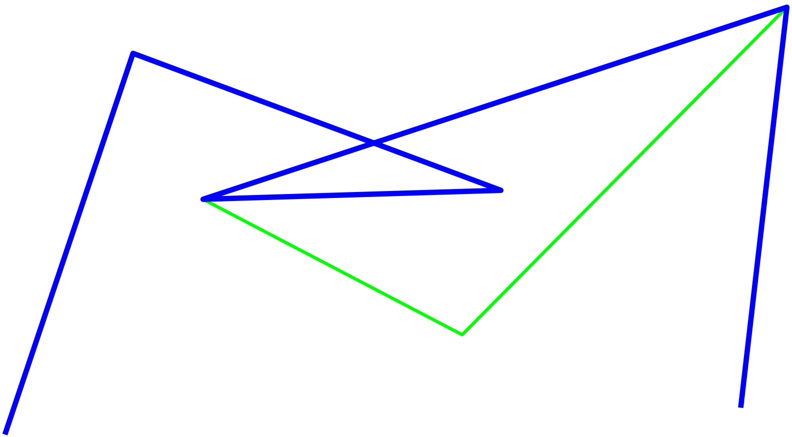
Special case: simple polygonal line

Graham does not work



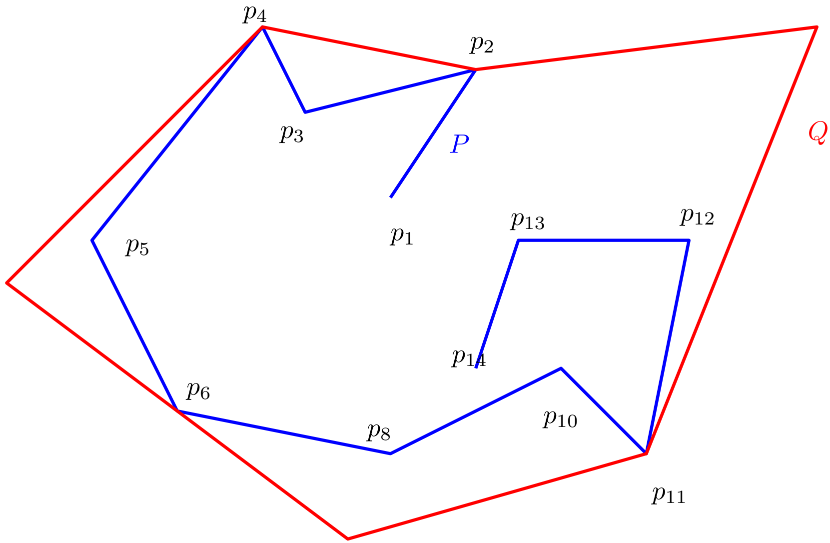
Special case: simple polygonal line

Graham does not work



Special case: simple polygonal line

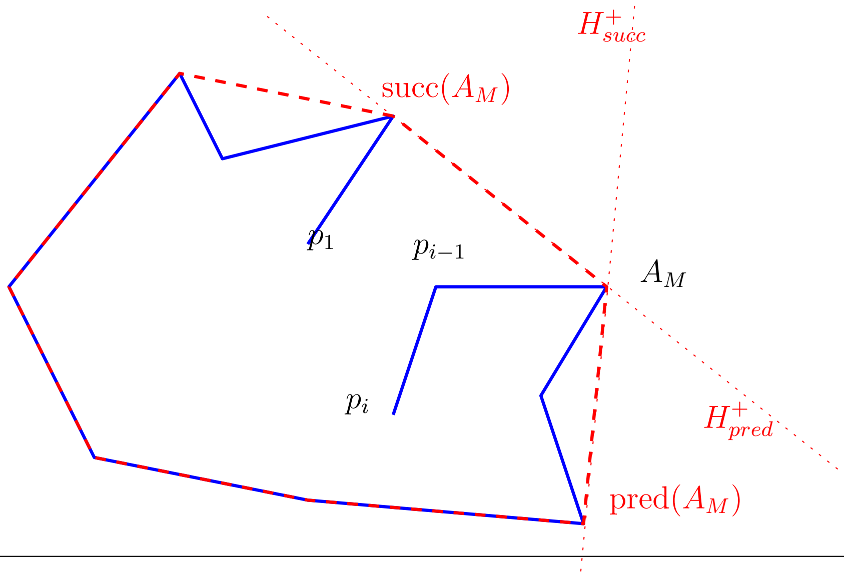
$P = p_1, p_2, ..., p_{14}$
Q contains the subsequence p_2, p_4, p_6, p_{11}



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 $p_i \in H^+_{pred} \cap H^+_{succ}$

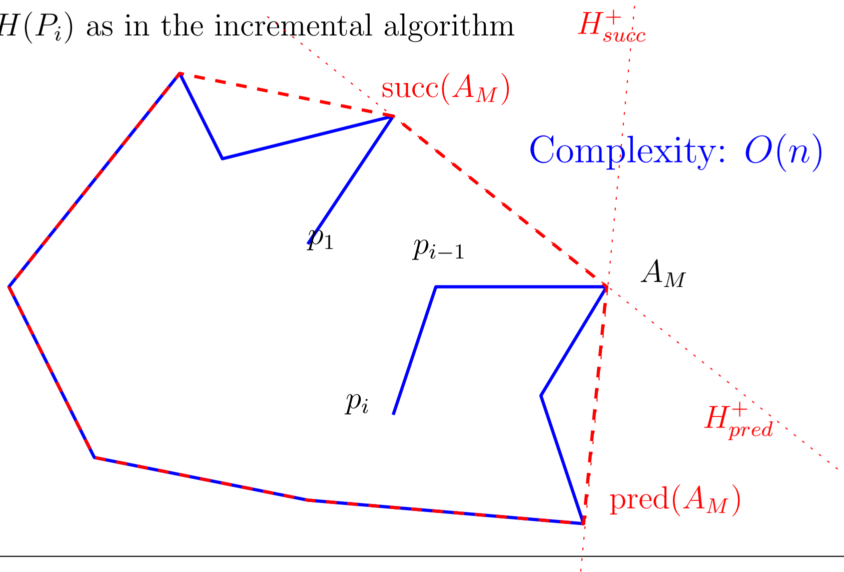


Special case: simple polygonal line

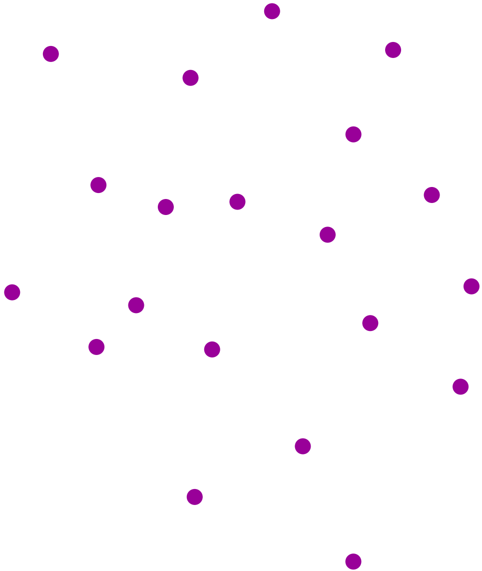
Incremental algorithm : order given by the polyline
A_M = highest rank point in $CH(P_{i-1})$

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

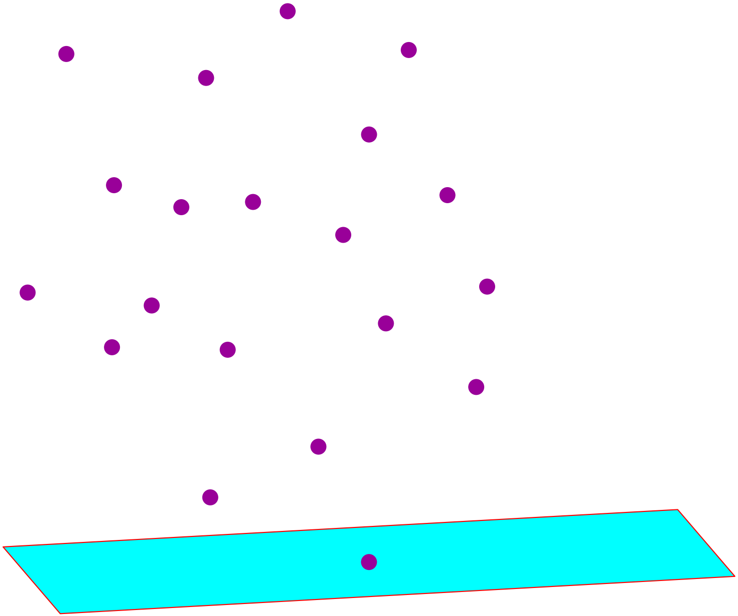
Update $CH(P_i)$ as in the incremental algorithm



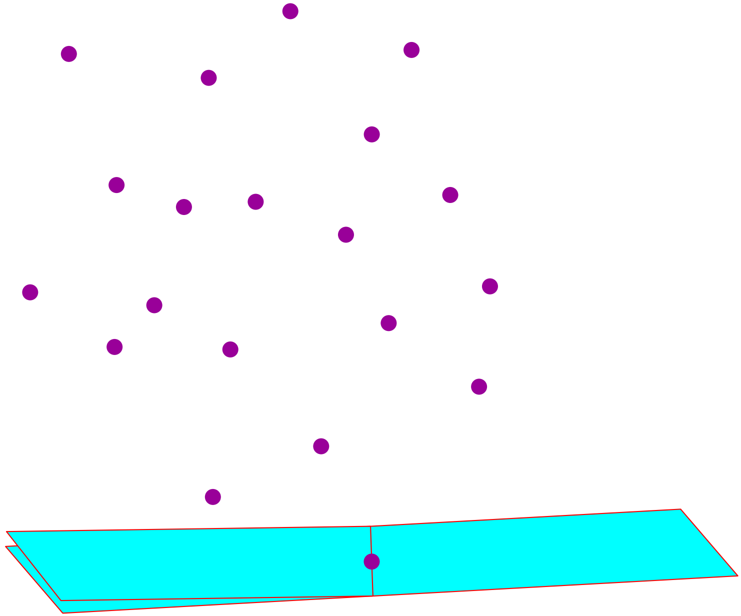
3D: Gift wrapping



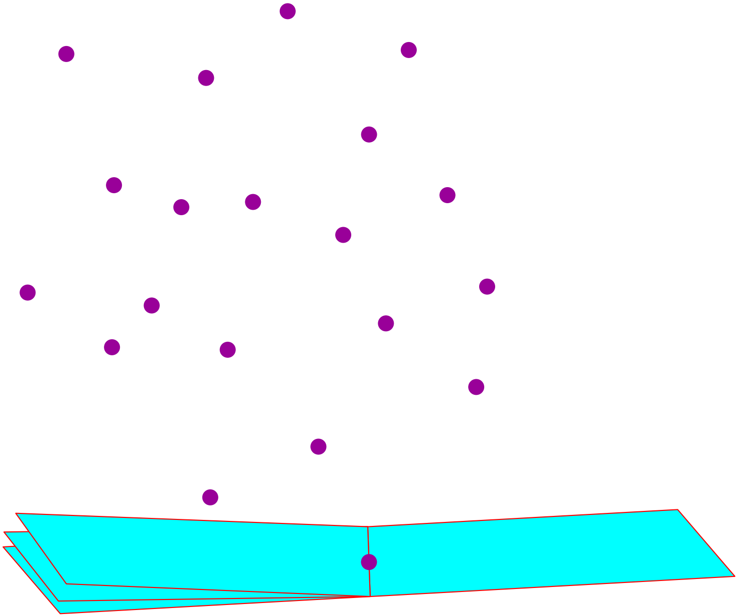
3D: Gift wrapping



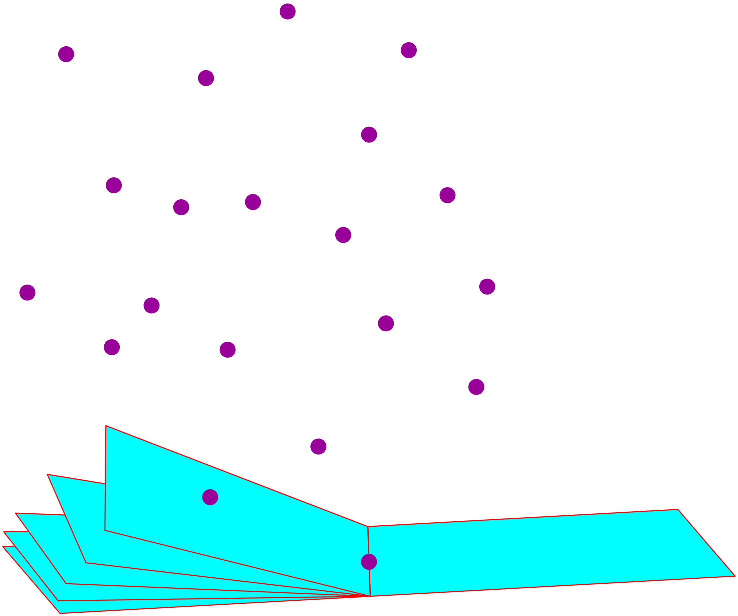
3D: Gift wrapping



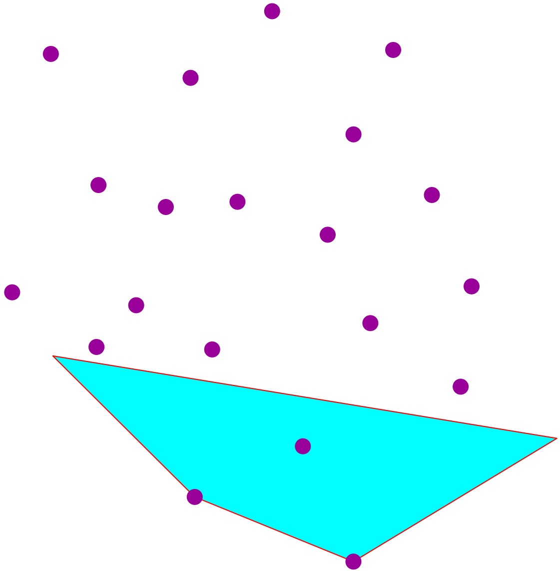
3D: Gift wrapping



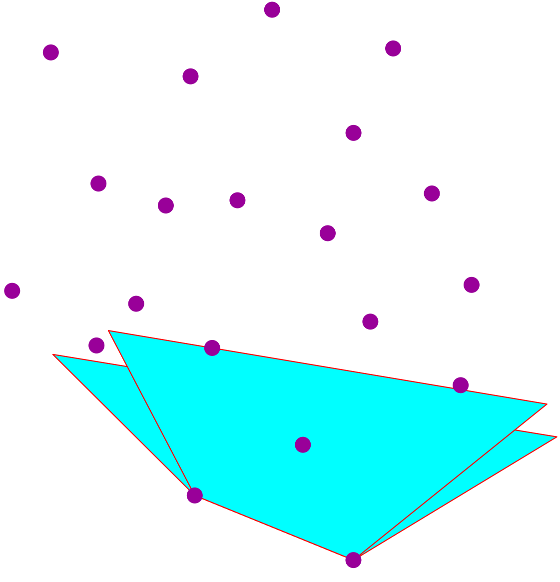
3D: Gift wrapping



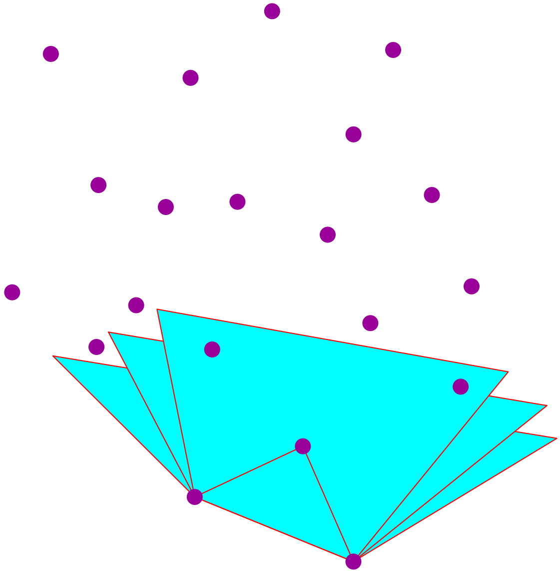
3D: Gift wrapping



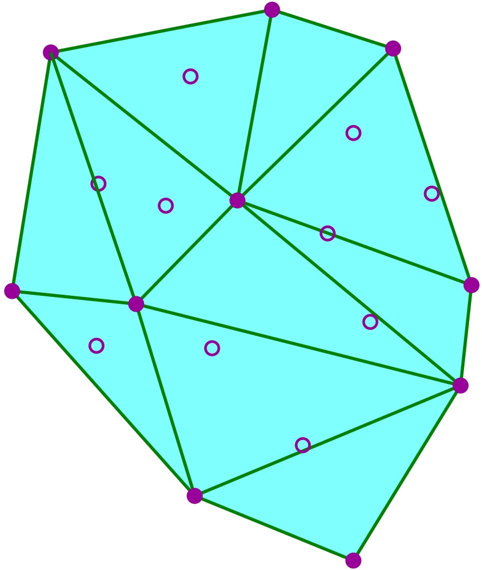
3D: Gift wrapping



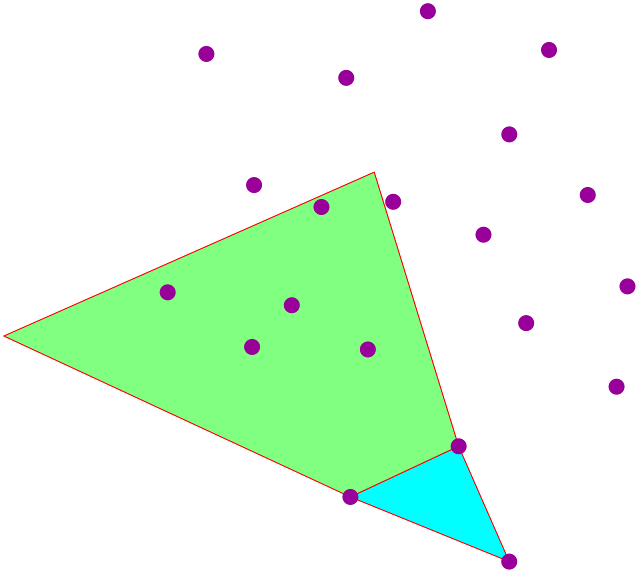
3D: Gift wrapping



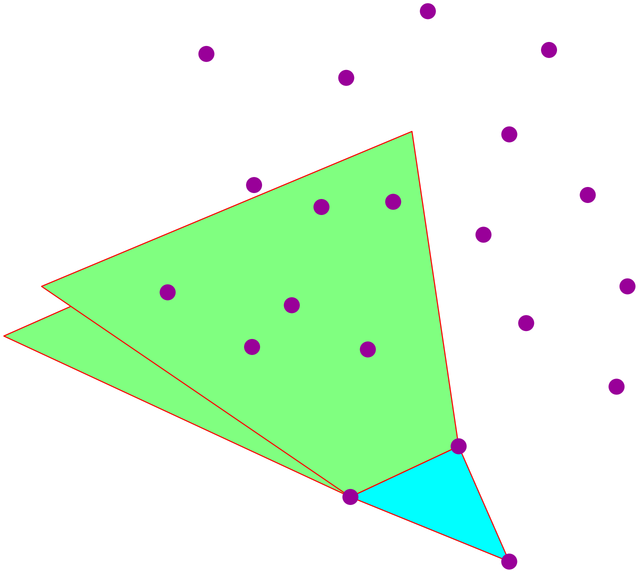
3D: Gift wrapping



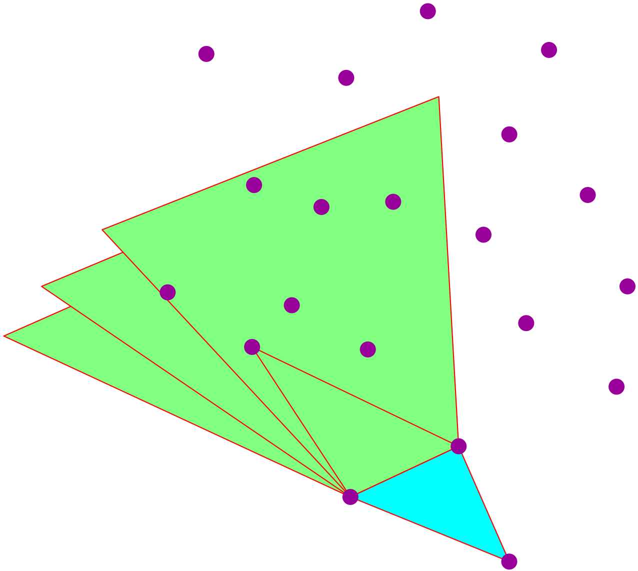
3D: Gift wrapping



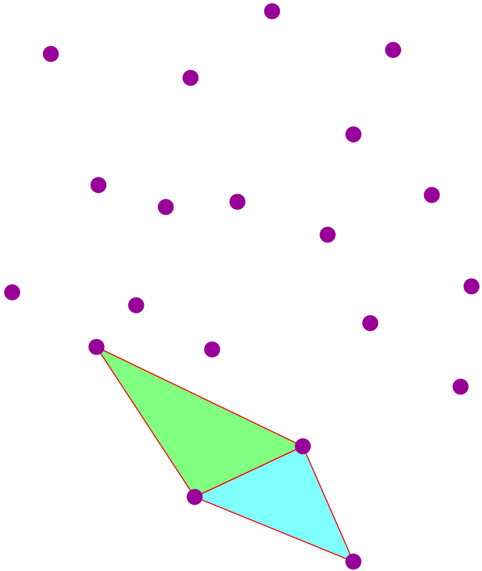
3D: Gift wrapping



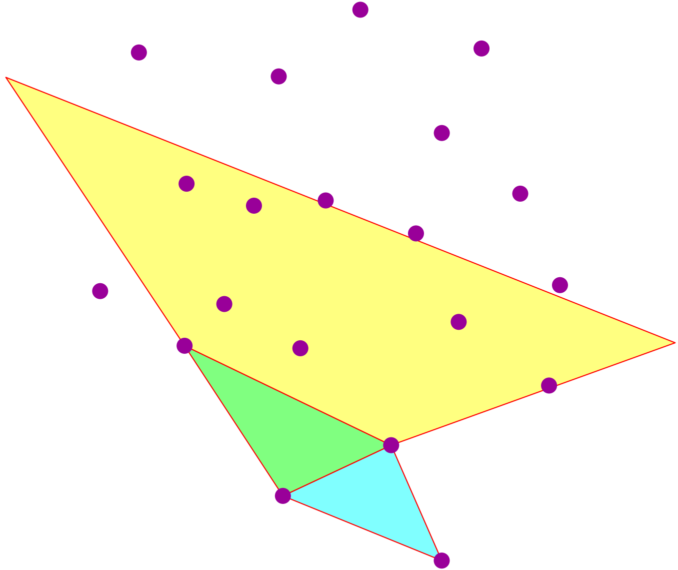
3D: Gift wrapping



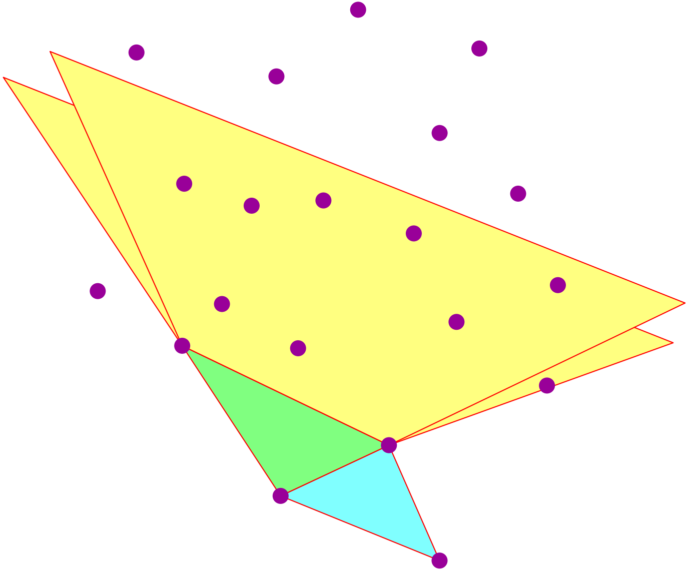
3D: Gift wrapping



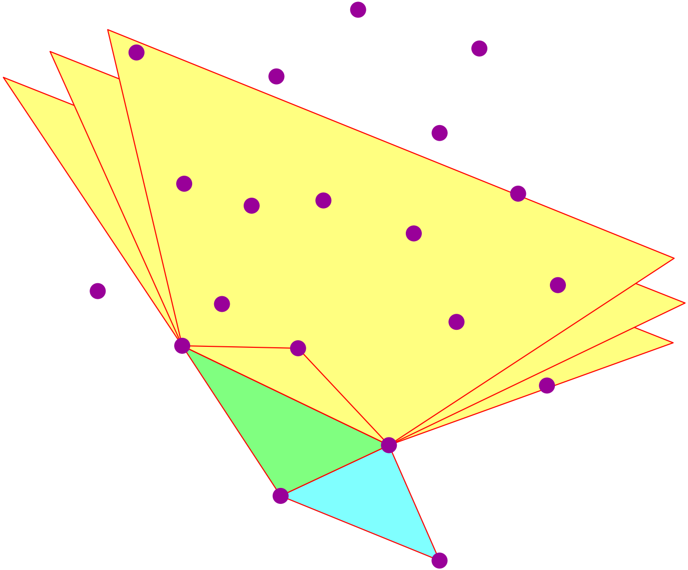
3D: Gift wrapping



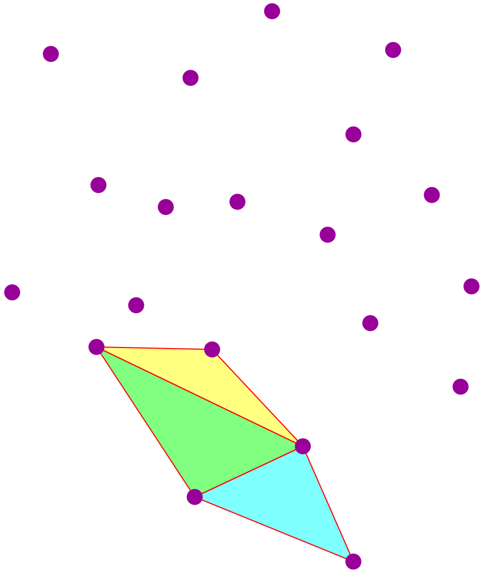
3D: Gift wrapping



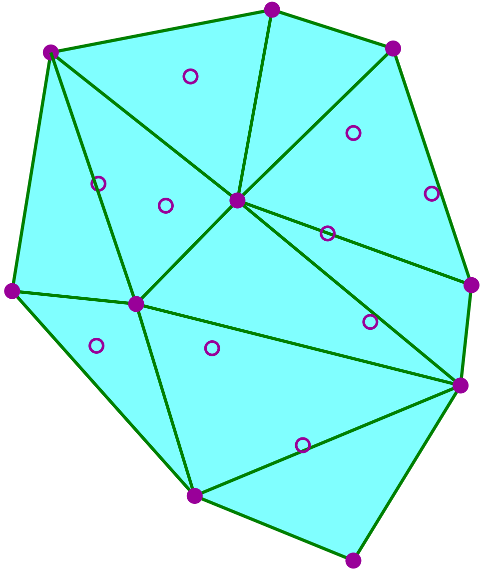
3D: Gift wrapping



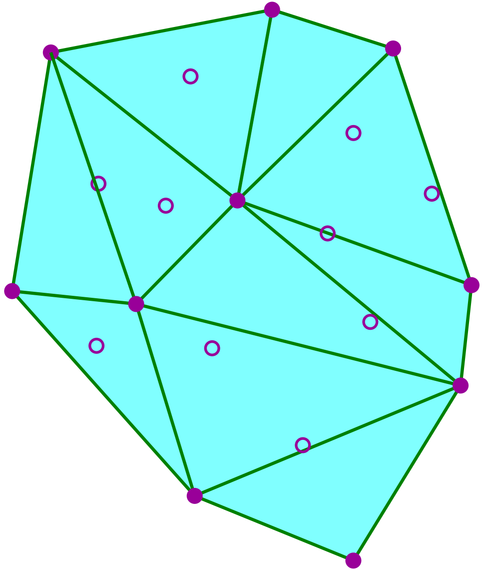
3D: Gift wrapping



3D: Gift wrapping

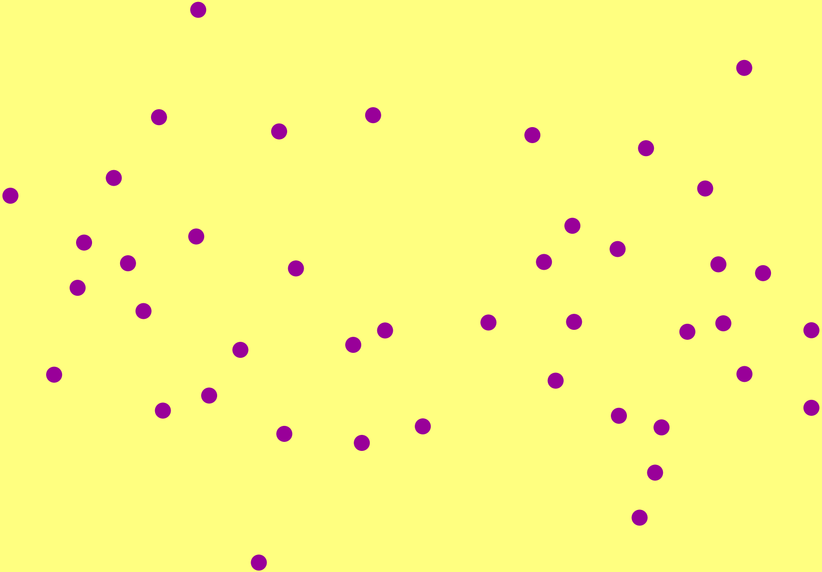


3D: Gift wrapping



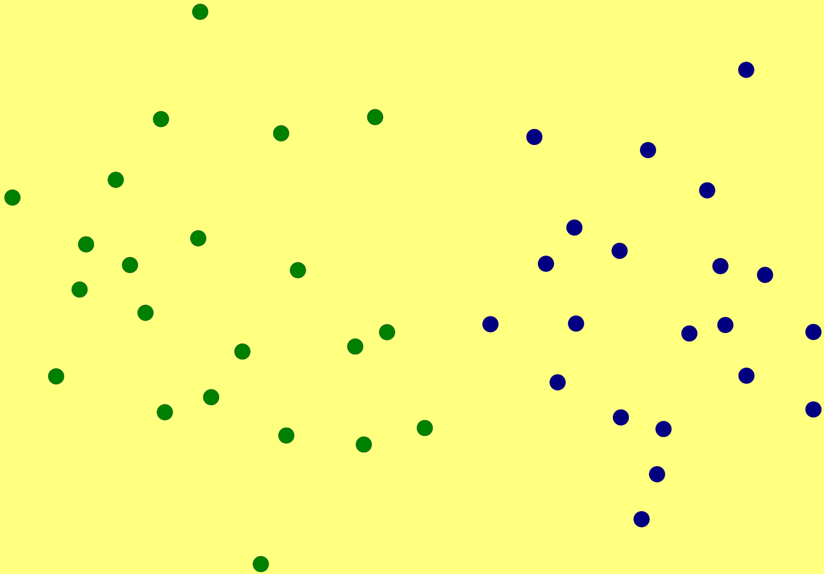
Complexity: $O(nh)$

3D Divide & conquer algorithm

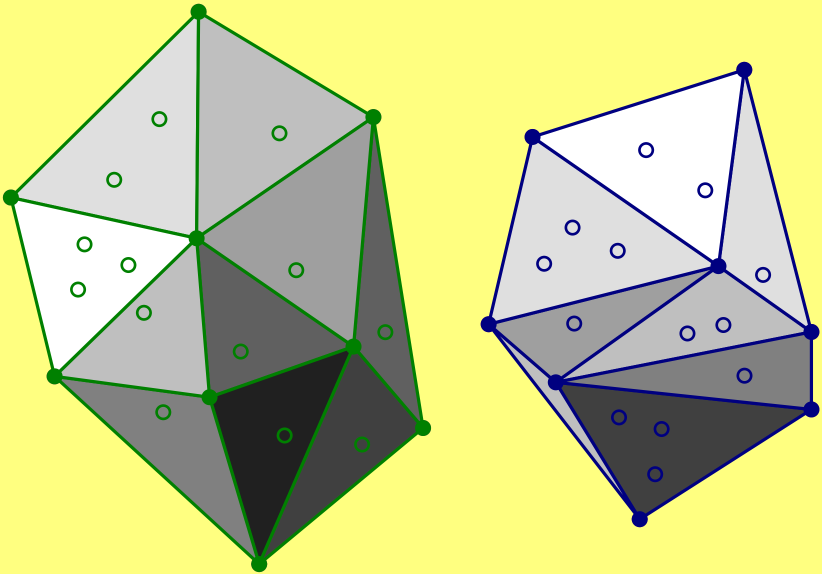


3D Divide & conquer algorithm

Sort in x and divide

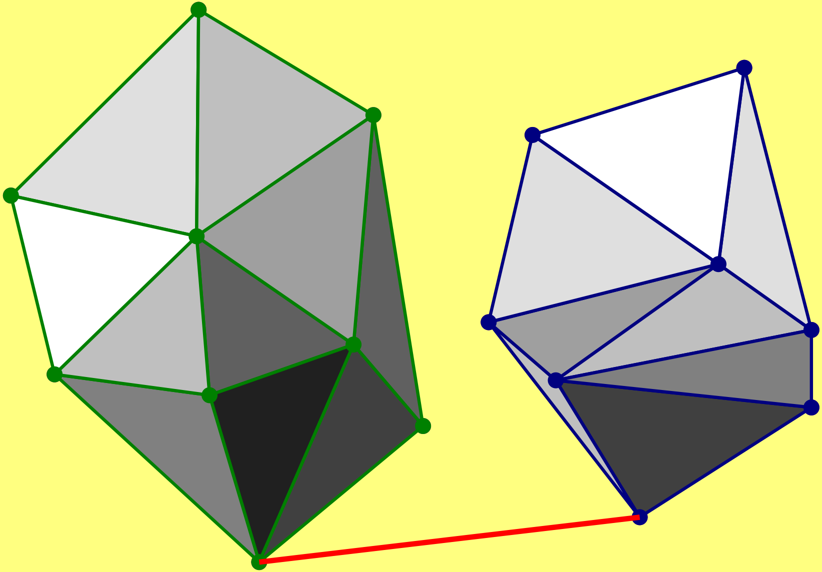


3D Divide & conquer algorithm



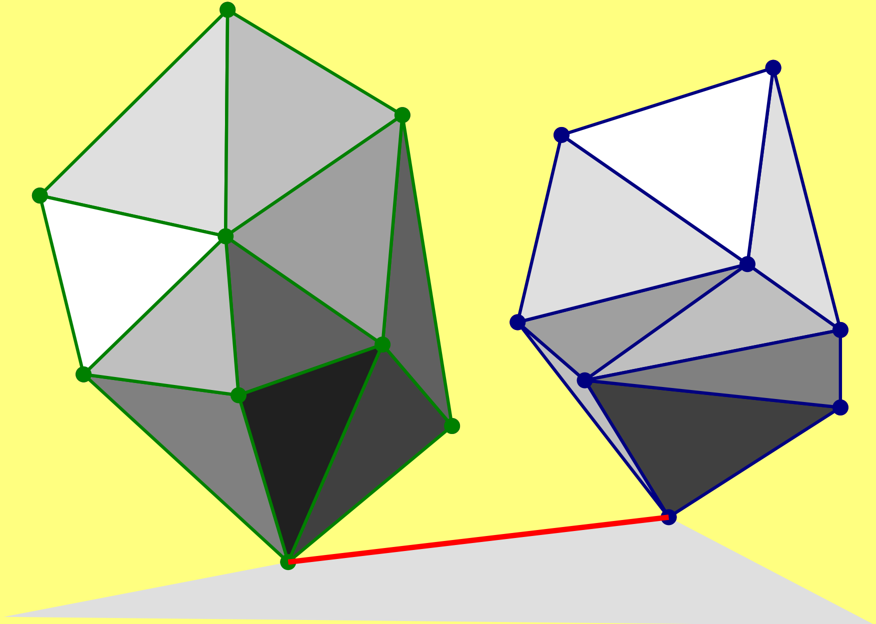
3D Divide & conquer algorithm

Find new edge: construct the CH of the projections,
use the 2d algo for bitangent



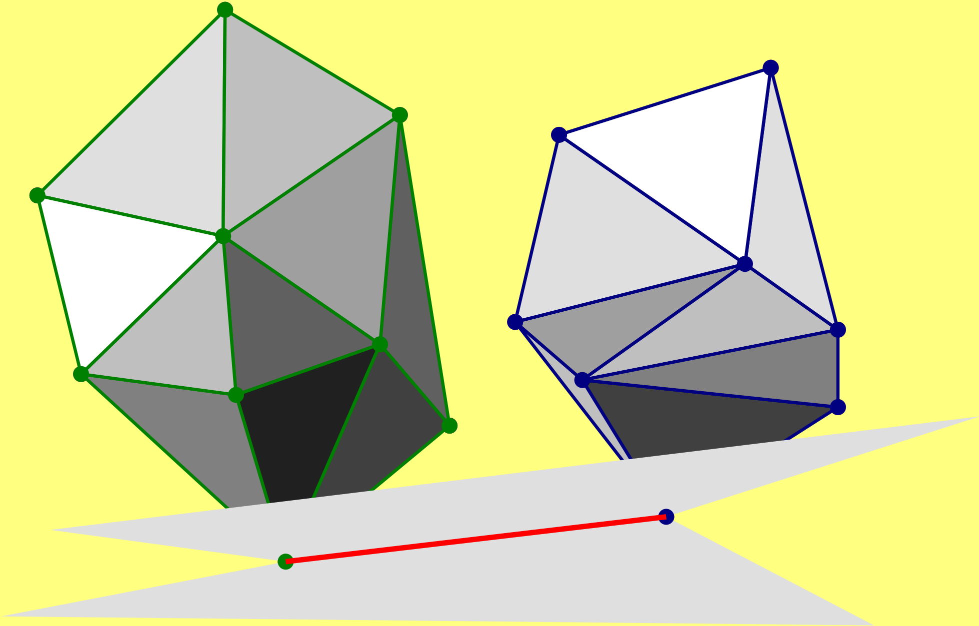
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges



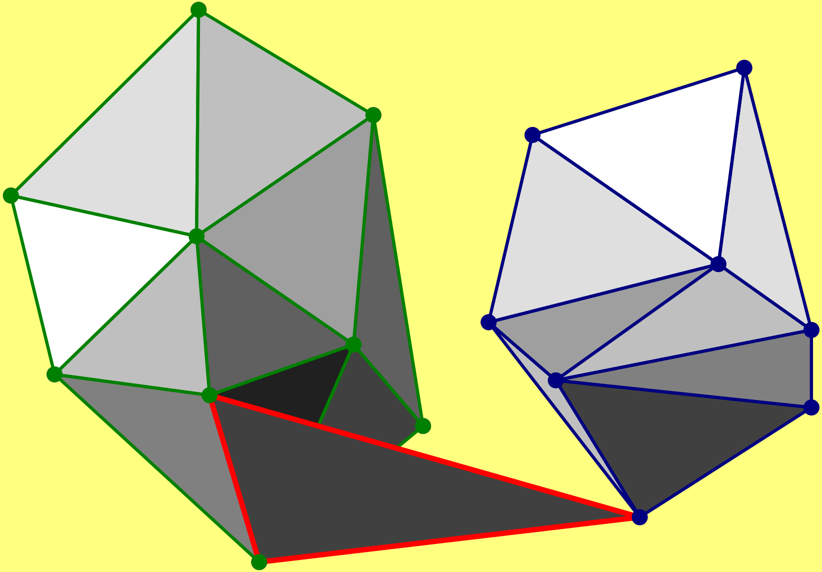
3D Divide & conquer algorithm

Use a wrapping algorithm around the new edges



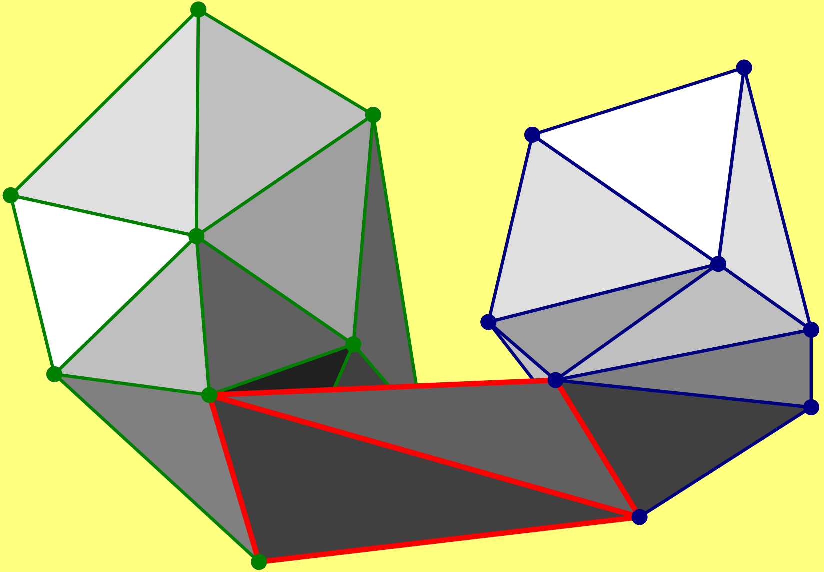
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



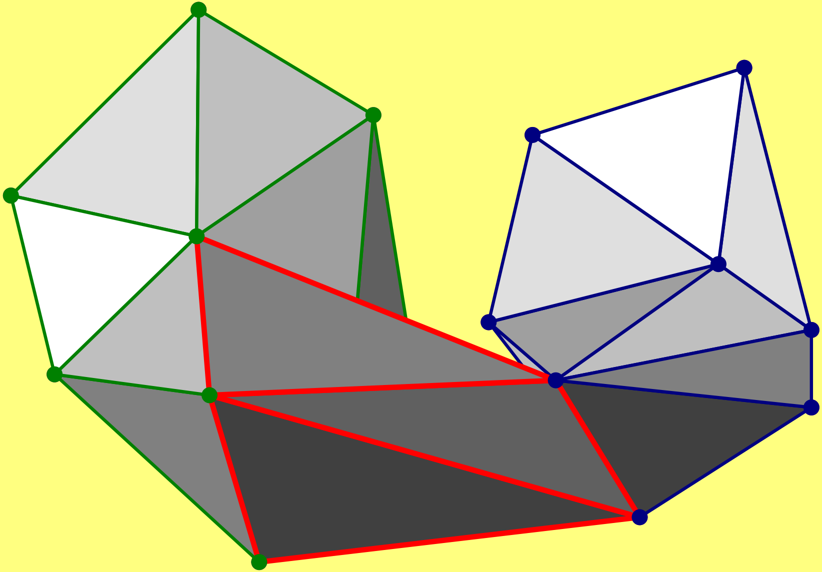
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



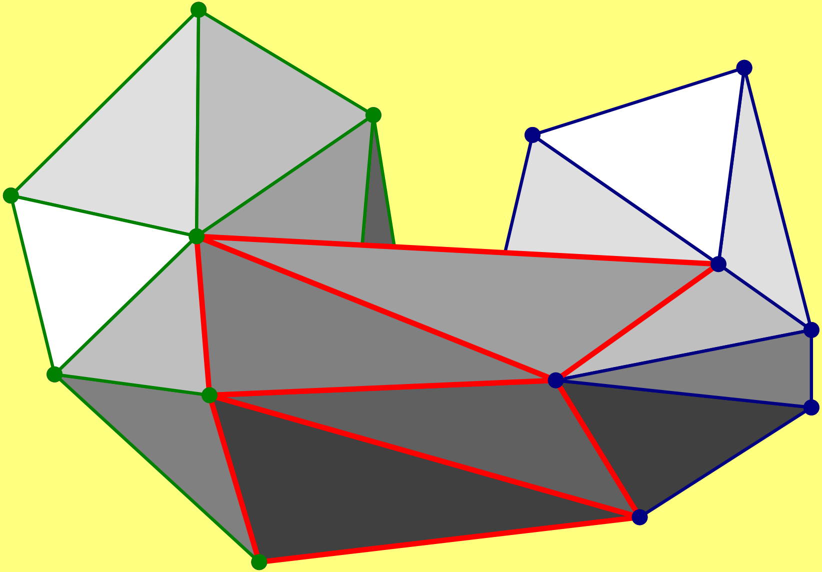
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



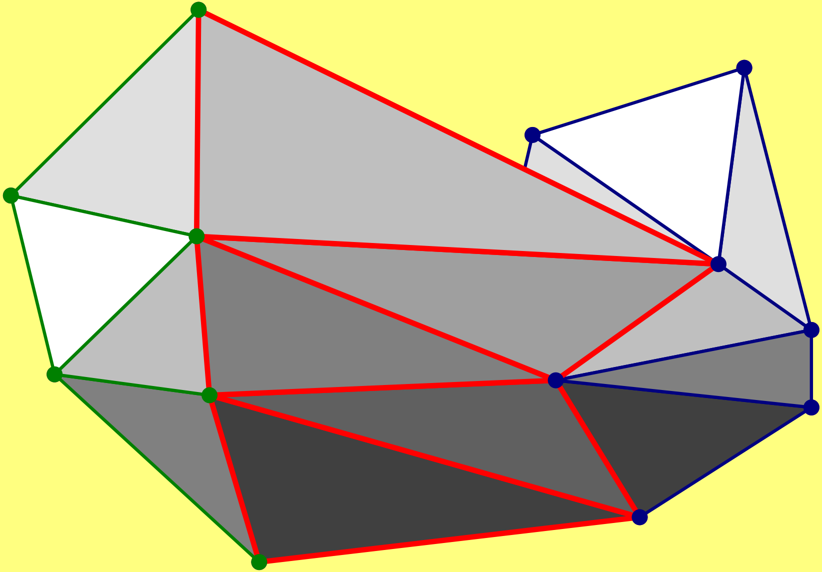
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



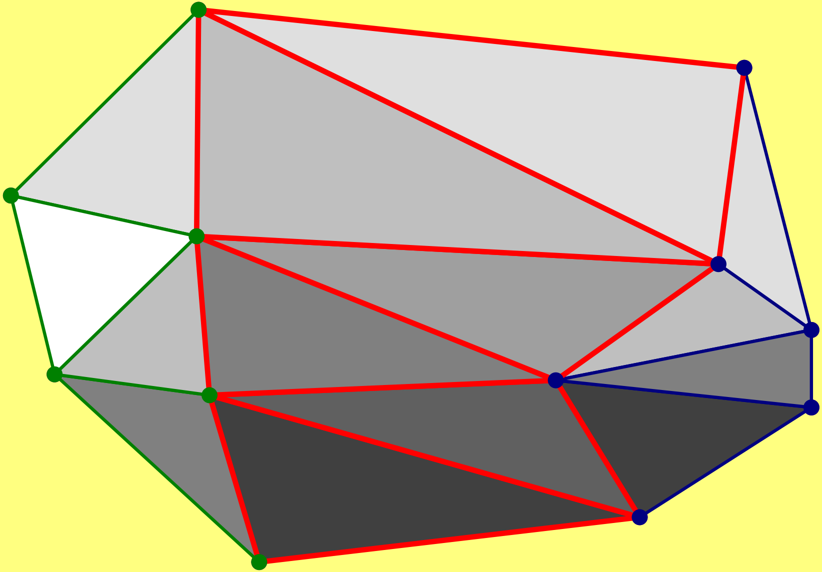
3D Divide & conquer algorithm

Search only in the star of the new edge vertices



3D Divide & conquer algorithm

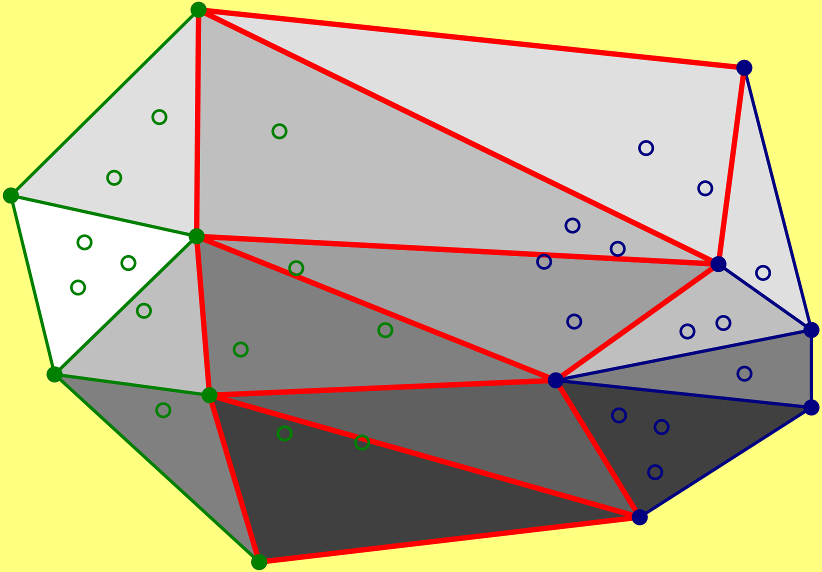
Search only in the star of the new edge vertices



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: $O(n \log n)$