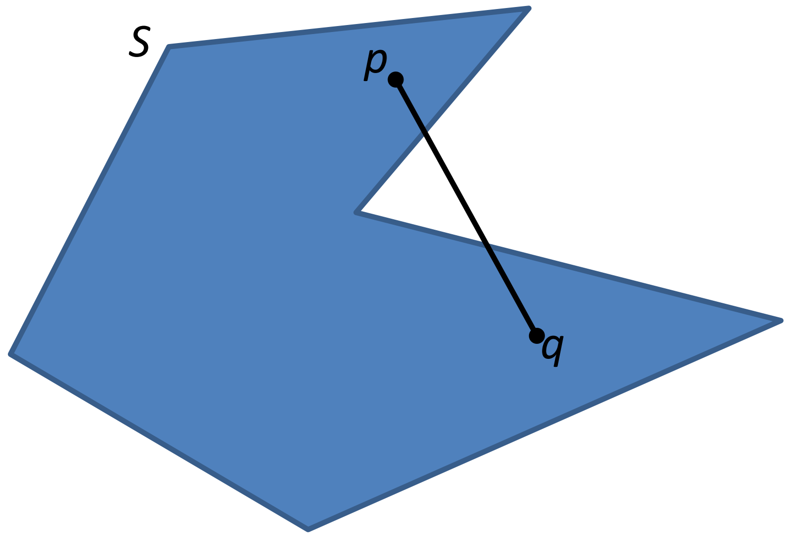
Computational Geometry

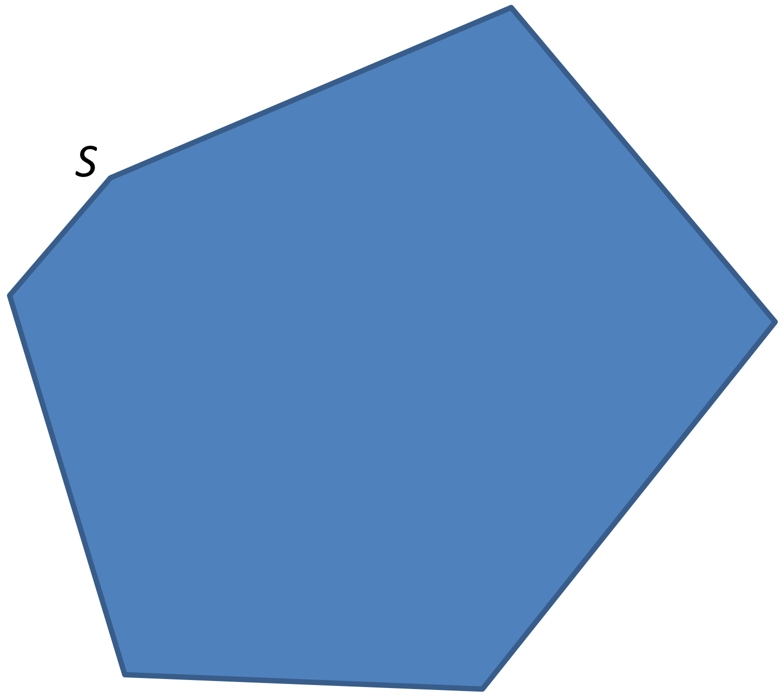
600.658

Convexity

A set S is *convex* if for any two points $p, q \in S$ the line segment $\overline{pq} \subset S$.



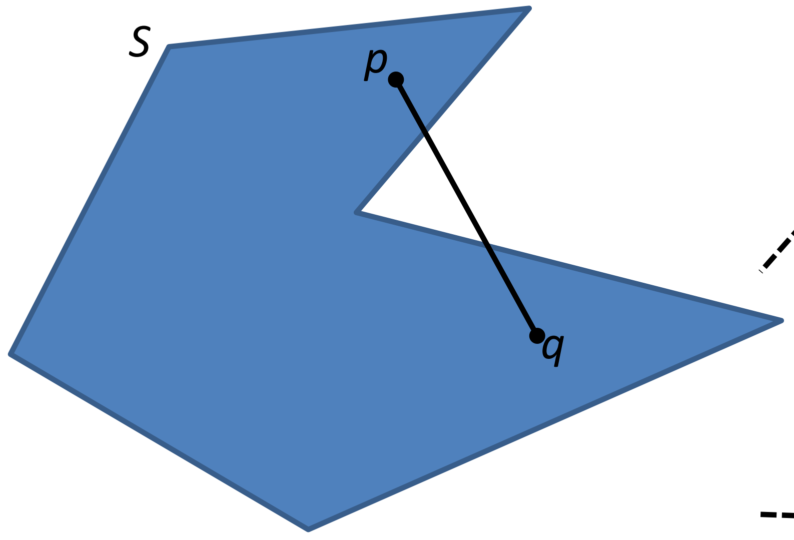
Not convex



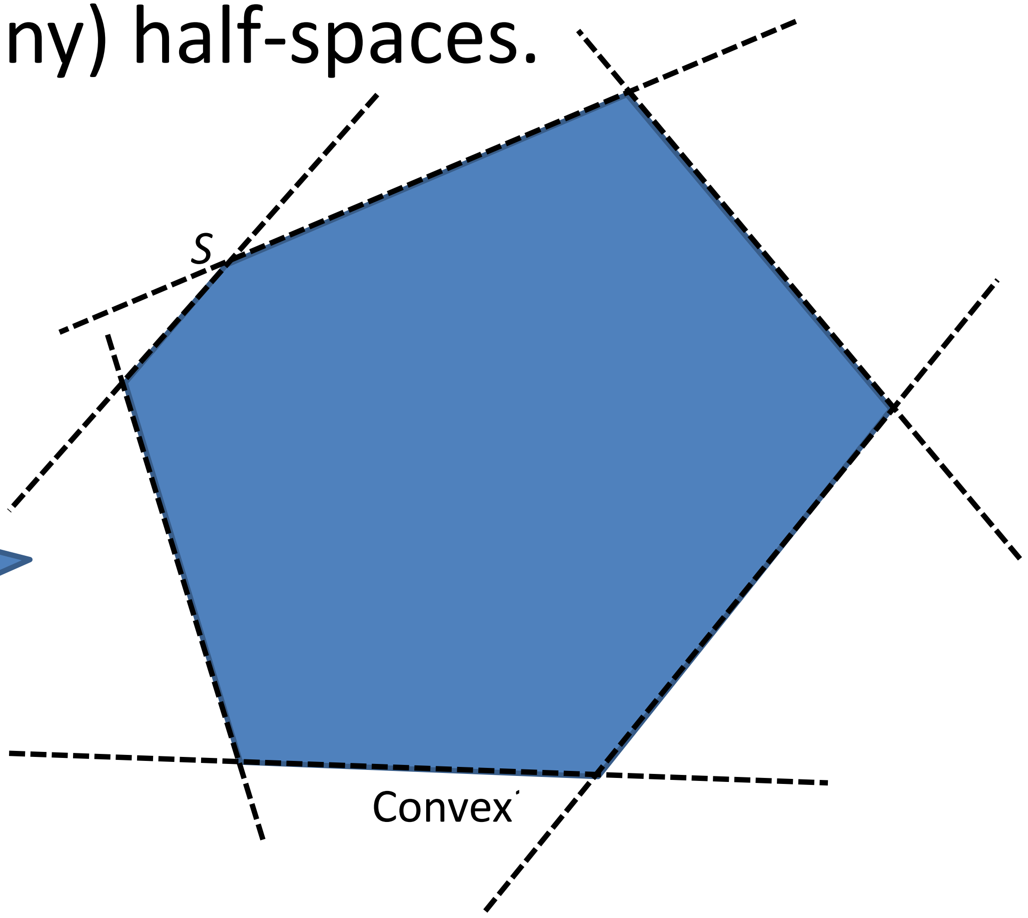
Convex?

Convexity

A set S is *convex* if it is the intersection of (possibly infinitely many) half-spaces.



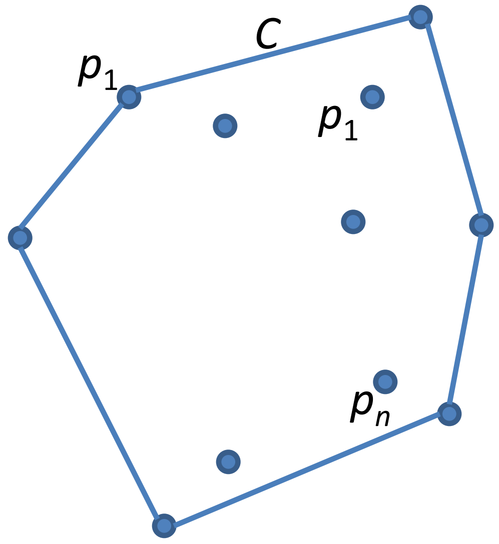
Not convex



Convex

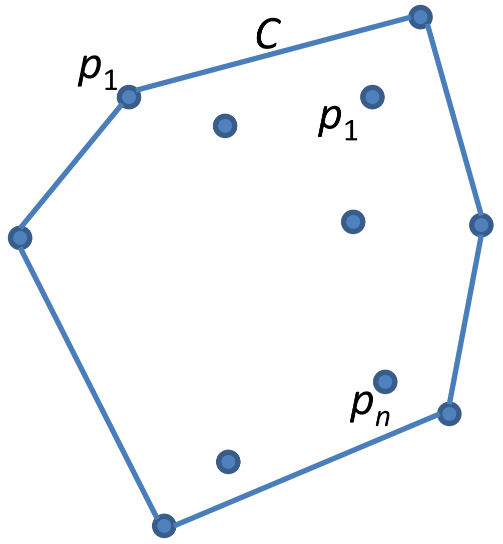
Convex Hull

Given a finite set of points $P = \{p_1, \dots, p_n\}$ the *convex hull* of P is the smallest convex set C such that $P \subset C$.



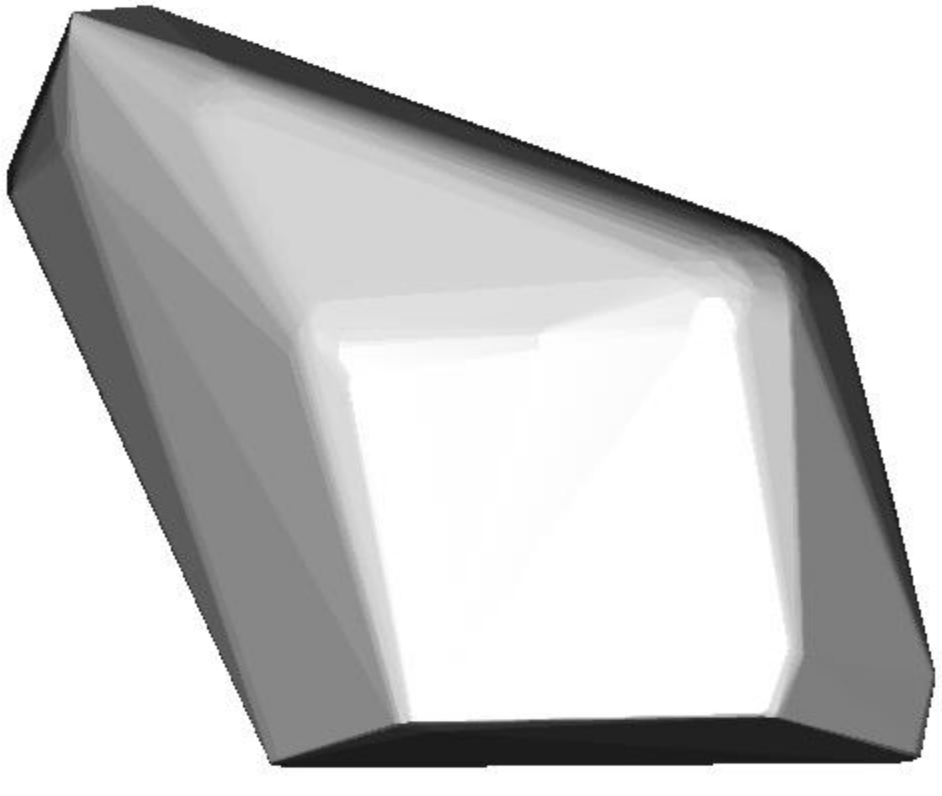
Convex Hull

In 2D, the convex hull of a set of points is made up of line segments with endpoints in P.



Convex Hull

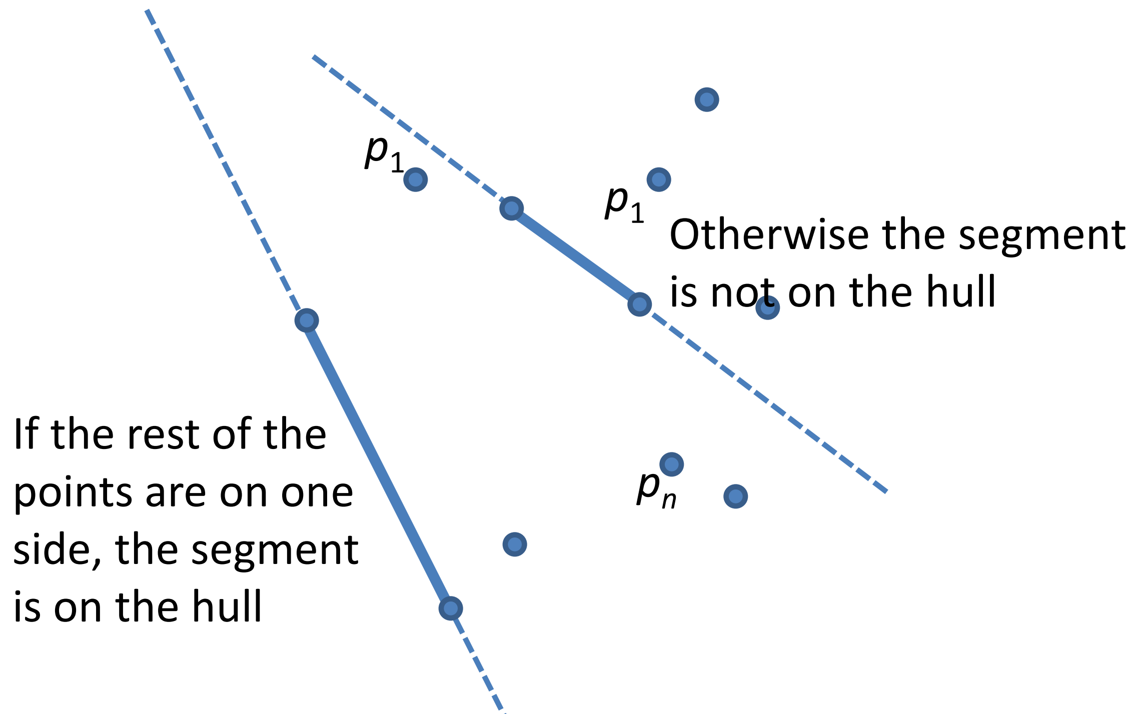
In 3D, the convex hull is a (triangle) mesh with vertices in P.



Computing the Convex Hull

Brute Force (2D):

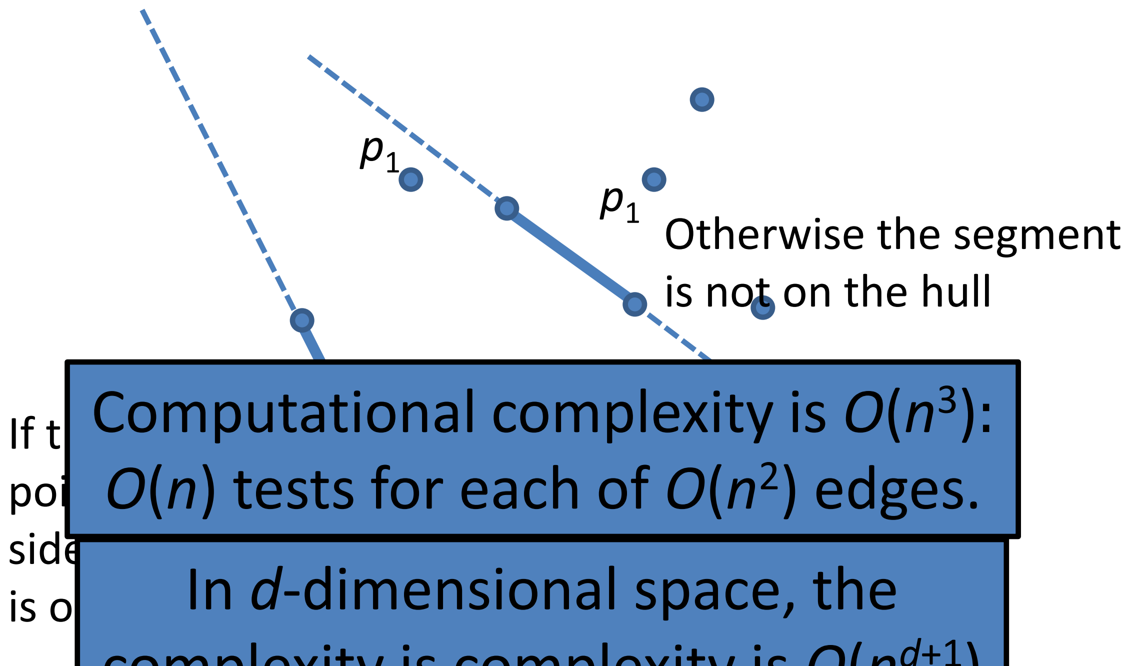
Given a set of points P, test each line segment to see if it is an edge on the convex hull.


p_1
p_1
Otherwise the segment is not on the hull
p_n
If the rest of the points are on one side, the segment is on the hull

Computing the Convex Hull

<u>Brute Force (2D)</u>:

Given a set of points P, test each line segment to see if it is an edge on the convex hull.



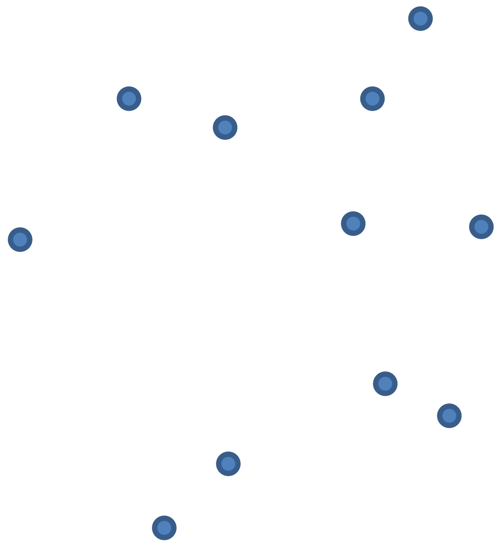
Computational complexity is $O(n^3)$: $O(n)$ tests for each of $O(n^2)$ edges.

In d-dimensional space, the complexity is complexity is $O(n^{d+1})$

Computing the Convex Hull

Quick Hull:

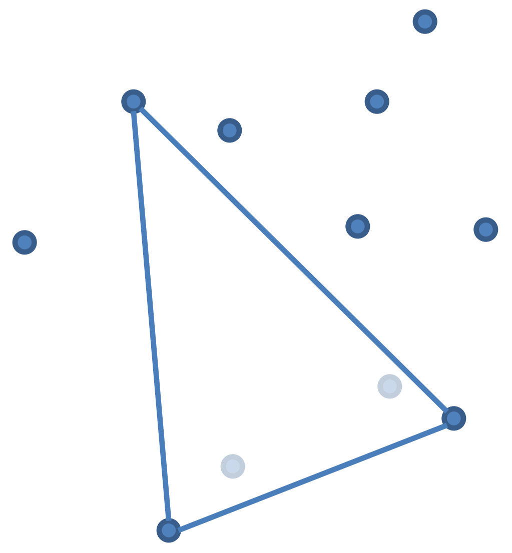
Key idea – in general, it's easy to identify internal points, so discard those quickly and focus on points nearer to the boundary.



Computing the Convex Hull

<u>Quick Hull</u>:

Observation – a point falling within any triangle has to be internal and can be discarded.



Computing the Convex Hull

```
$\underline{QH(S)}$:
Choose $a, b \in S$ with min/max x-coordinate.
$A \leftarrow$ points to the right of $\overrightarrow{ab}$.
$B \leftarrow$ points to the left of $\overrightarrow{ab}$.
return $QH(A,a,b) \cup QH(B,b,a)$
```

Computing the Convex Hull

```
$\underline{QH(S,a,b)}$:
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $QH(A,a,c) \cup QH(B,c,b)$
```

Computing the Convex Hull

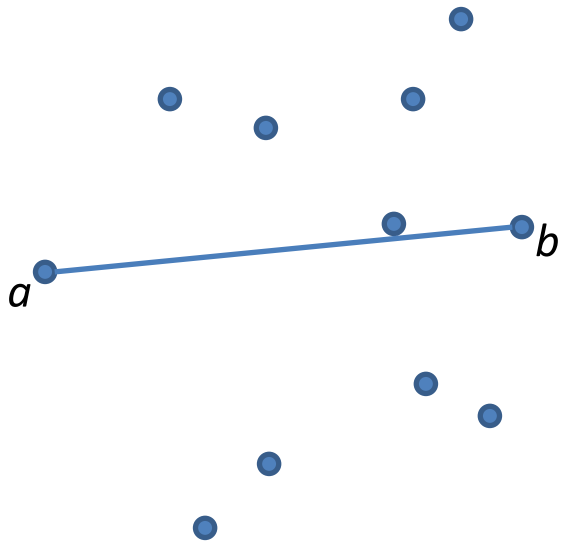
<u>*QH*(S)</u>:

Choose $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{S}$ with min/max *x*-coordinate.

$A \leftarrow$ points to the right of $\overrightarrow{ab}$.

$B \leftarrow$ points to the left of $\overrightarrow{ab}$.

return *QH*(A,a,b) ∪ *QH*(B,b,a)



Computing the Convex Hull

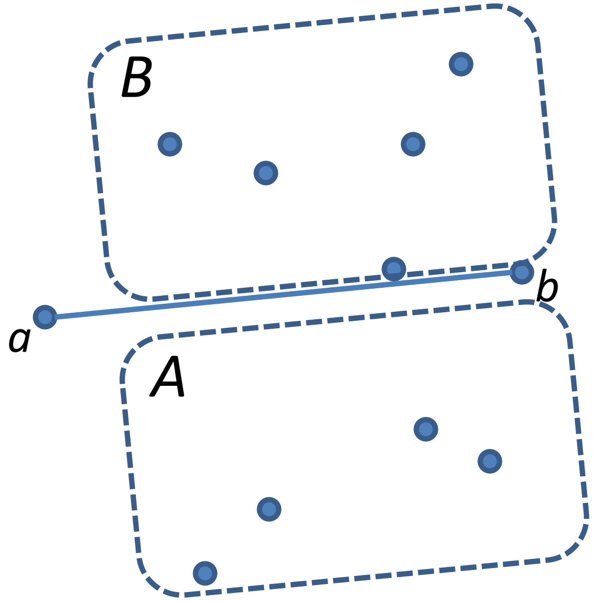
<u>*QH*(S)</u>:

Choose $a, b \in S$ with min/max *x*-coordinate.

$\boldsymbol{A} \leftarrow$ **points to the right of $\overrightarrow{\boldsymbol{ab}}$.**

$\boldsymbol{B} \leftarrow$ **points to the left of $\overrightarrow{\boldsymbol{ab}}$.**

return *QH*(A,a,b) ∪ *QH*(B,b,a)



Computing the Convex Hull

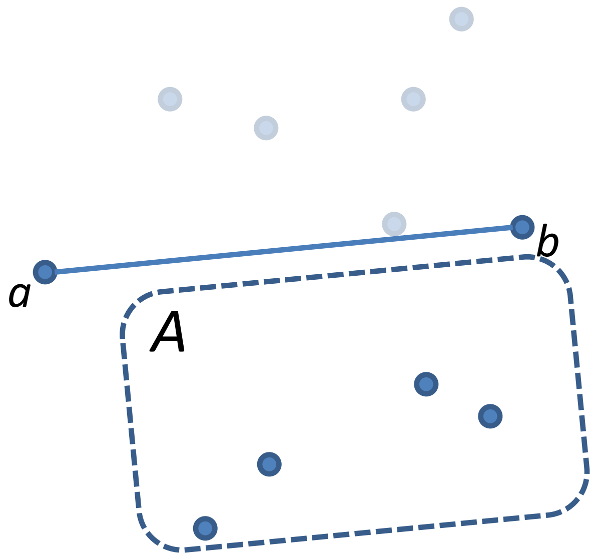
<u>$QH(S)$</u>:

Choose $a, b \in S$ with min/max *x*-coordinate.

$A \leftarrow$ points to the right of $\overrightarrow{ab}$.

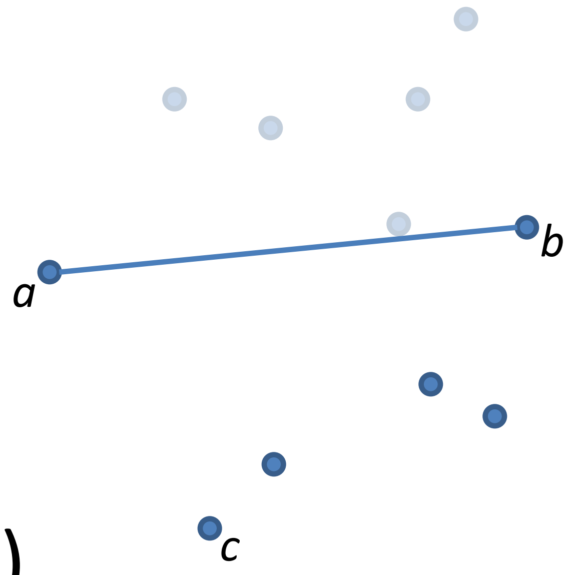
$B \leftarrow$ points to the left of $\overrightarrow{ab}$.

return $\boldsymbol{QH(A,a,b)} \cup QH(B,b,a)$



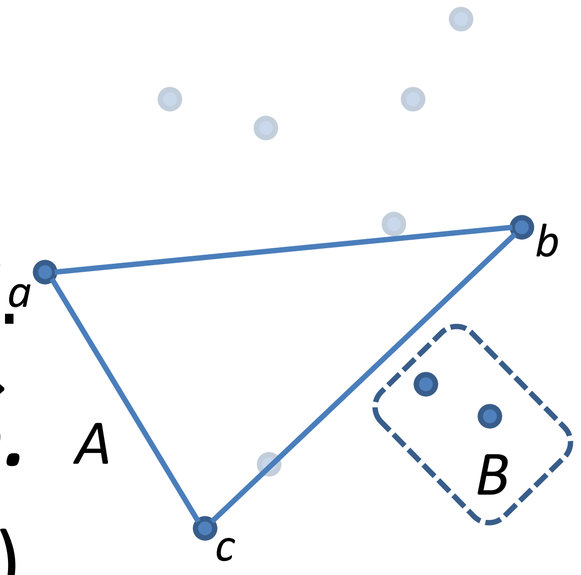
Computing the Convex Hull

```
$\underline{QH(S,a,b)}$:
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $\boldsymbol{c} \leftarrow$ **point furthest from** $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $QH(A,a,c) \cup QH(B,c,b)$
```



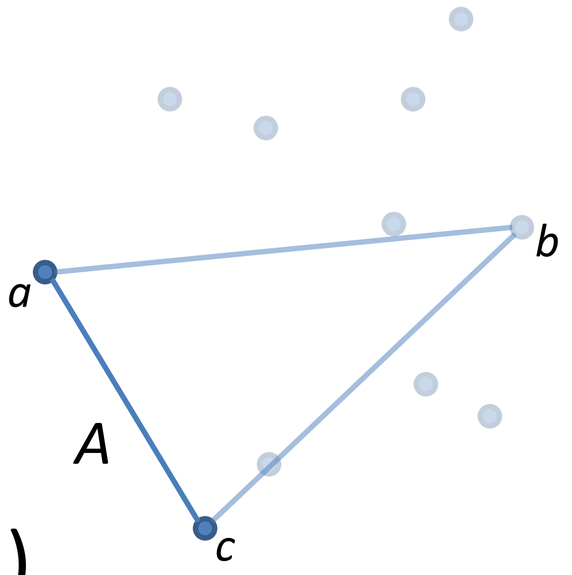
Computing the Convex Hull

```
QH(S,a,b):
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    **$\boldsymbol{A} \leftarrow$ points to the right of $\overrightarrow{\boldsymbol{ac}}$.**
    **$\boldsymbol{B} \leftarrow$ points to the right of $\overrightarrow{\boldsymbol{cb}}$.**
    return $QH(A,a,c) \cup QH(B,c,b)$
```



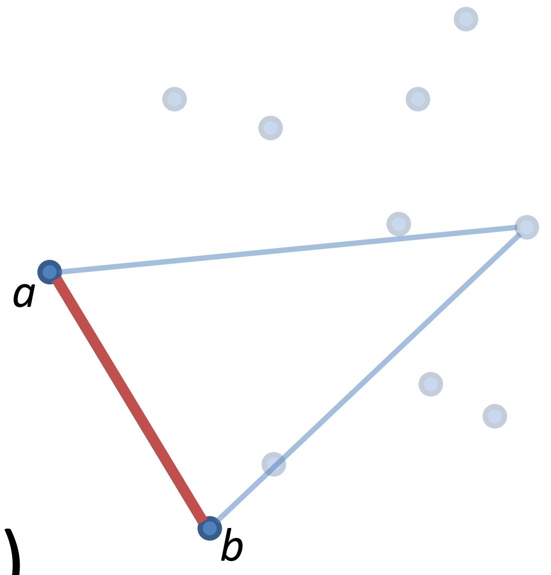
Computing the Convex Hull

```
$\underline{QH(S,a,b)}$:
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $\boldsymbol{QH(A,a,c)} \cup QH(B,c,b)$
```



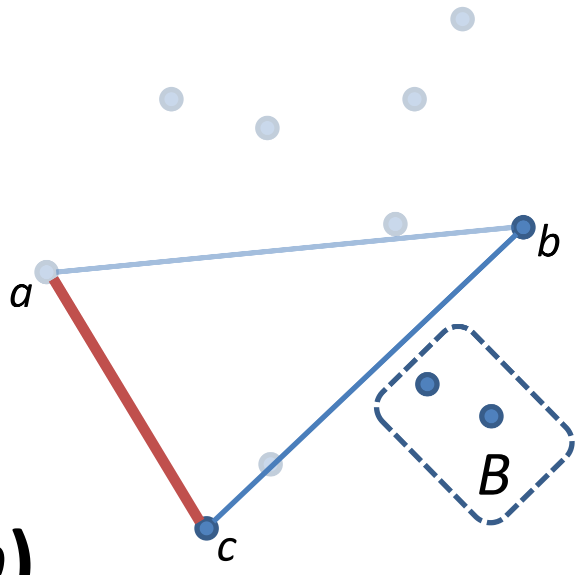
Computing the Convex Hull

```
<u>$QH(S,a,b)$</u>:
**if $S$ is empty, return $\overrightarrow{ab}$.**
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $QH(A,a,c) \cup QH(B,c,b)$
```



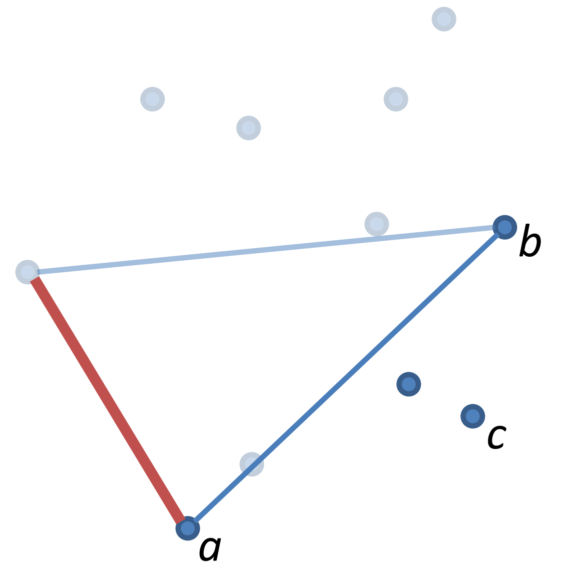
Computing the Convex Hull

```
$\underline{QH(S,a,b)}$:
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $QH(A,a,c) \cup \boldsymbol{QH(B,c,b)}$
```



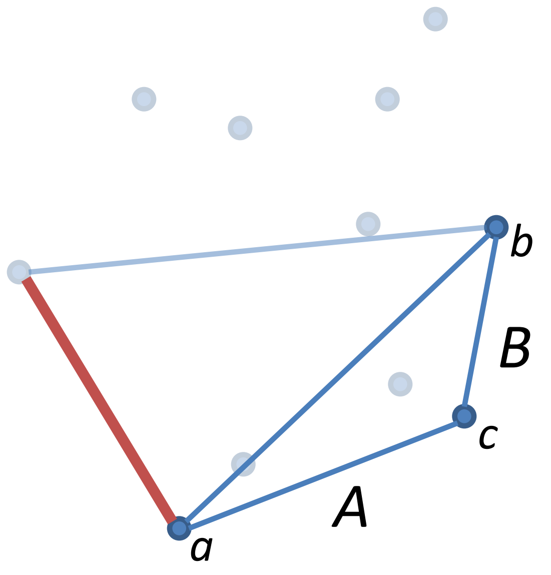
Computing the Convex Hull

```
QH(S,a,b):
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return $QH(A,a,c) \cup QH(B,c,b)$
```



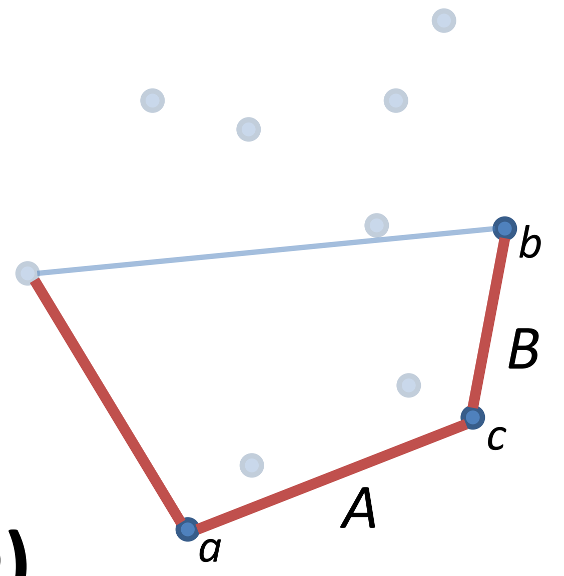
Computing the Convex Hull

```
$\underline{QH(S,a,b)}$:
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    **$A \leftarrow$ points to the right of $\overrightarrow{ac}$.**
    **$B \leftarrow$ points to the right of $\overrightarrow{cb}$.**
    return $QH(A,a,c) \cup QH(B,c,b)$
```



Computing the Convex Hull

```
QH(S,a,b):
if $S$ is empty, return $\overrightarrow{ab}$.
else
    $c \leftarrow$ point furthest from $\overrightarrow{ab}$.
    $A \leftarrow$ points to the right of $\overrightarrow{ac}$.
    $B \leftarrow$ points to the right of $\overrightarrow{cb}$.
    return QH(A,a,c) ∪ QH(B,c,b)
```



Computing the Convex Hull

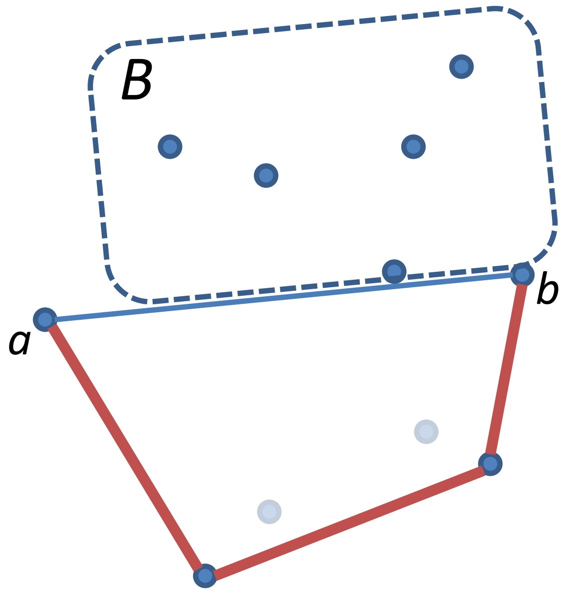
<u>$QH(S)$</u>:

Choose $a, b \in S$ with min/max *x*-coordinate.

$A \leftarrow$ points to the right of $\overrightarrow{ab}$.

$B \leftarrow$ points to the left of $\overrightarrow{ab}$.

return $QH(A,a,b) \cup \boldsymbol{QH(B,b,a)}$



Computing the Convex Hull

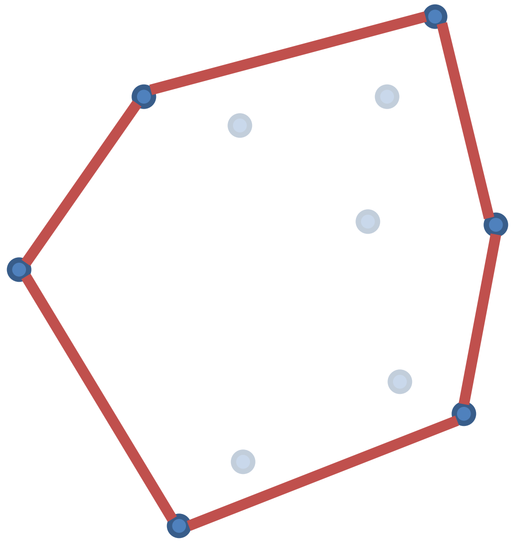
<u>$QH(S)$</u>:

Choose $a, b \in S$ with min/max *x*-coordinate.

$A \leftarrow$ points to the right of $\overrightarrow{ab}$.

$B \leftarrow$ points to the left of $\overrightarrow{ab}$.

return $QH(A,a,b) \cup QH(B,b,a)$

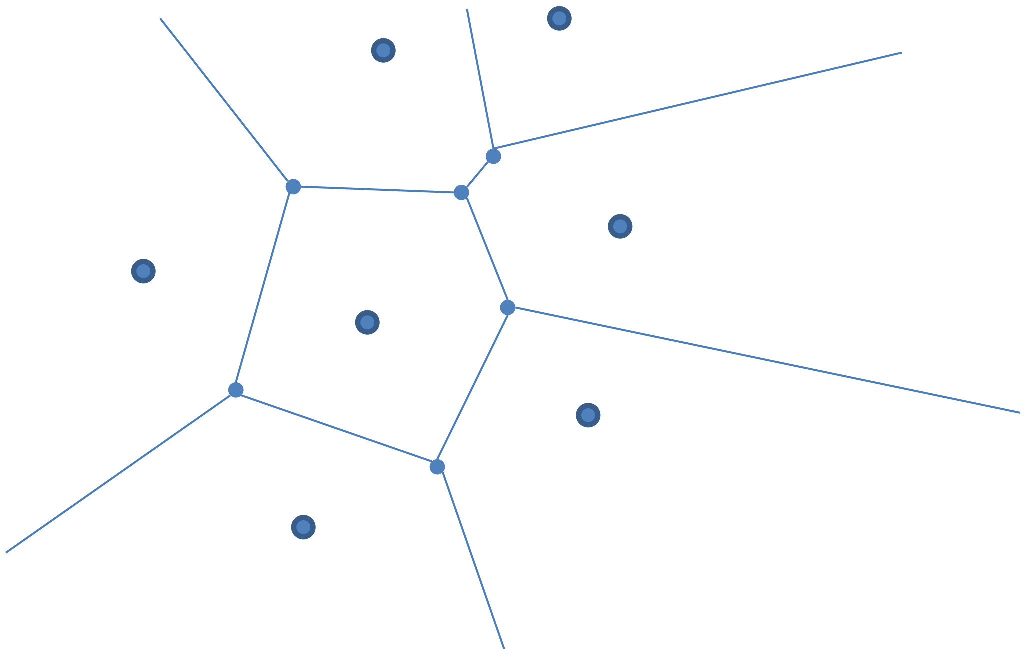


Worst-case complexity $O(n^2)$

In practice runs in $O(n \log n)$

Voronoi Diagrams

Given a finite set of points P the *Voronoi Diagram of* P is a partition of space into regions $\{R_p\}$ such that the points in R_p are closer to $p \in P$ than to any other $q \in P$.

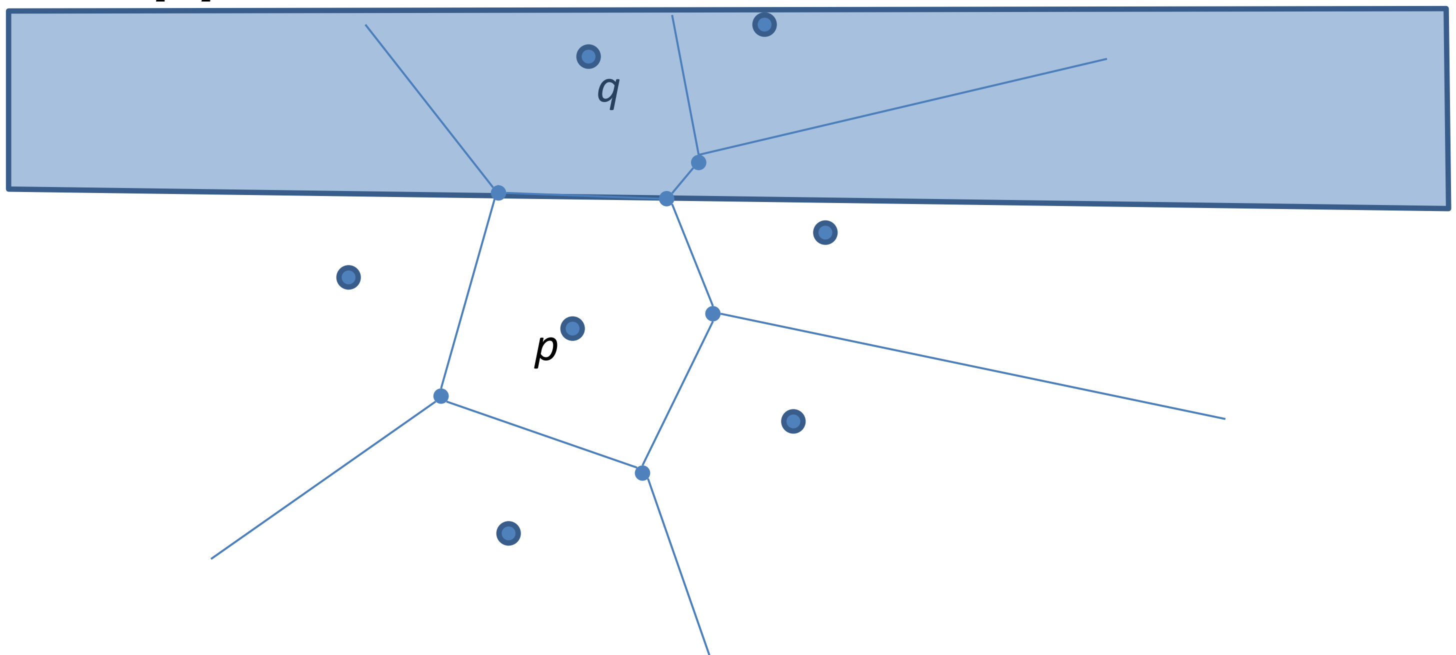


Voronoi Diagrams

<u>Claim</u>: The Voronoi regions, R_p, are convex.

<u>Proof</u>:

Given any other point $q \in P$, we can define the half-space H_{pq} of points closer to p than q.

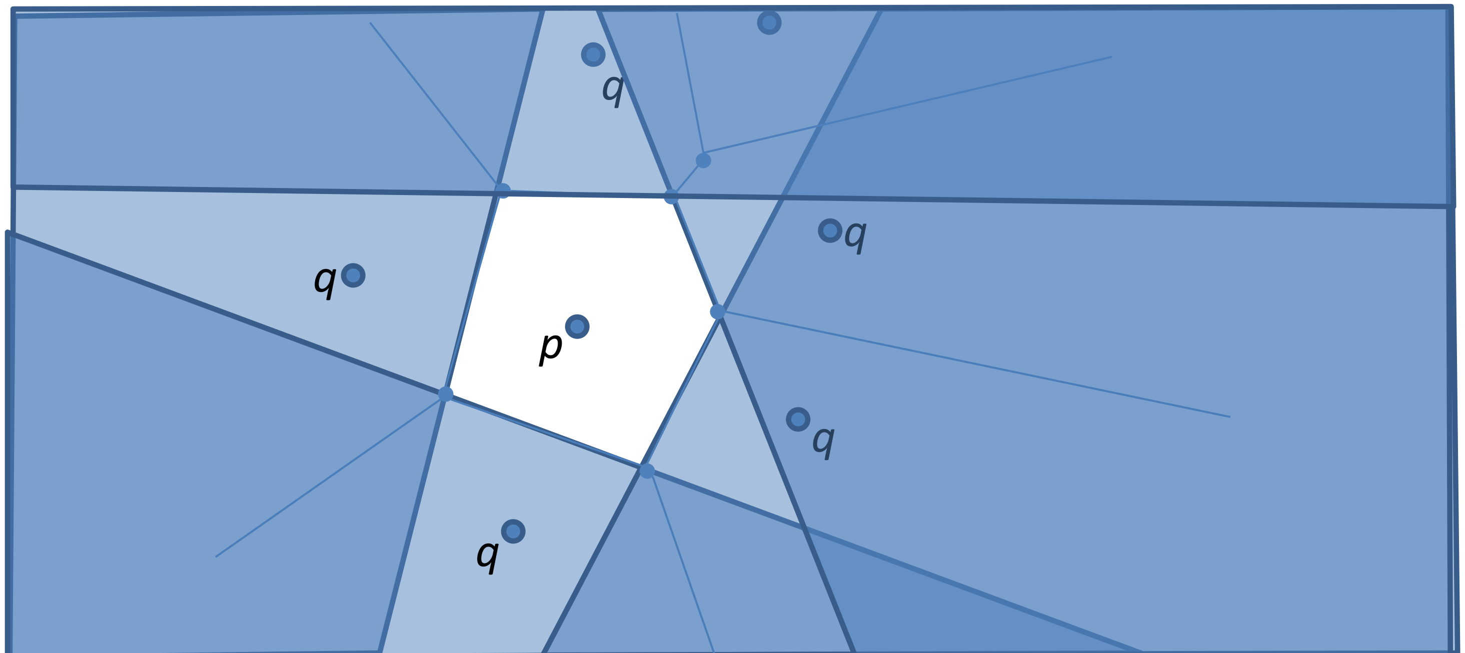


Voronoi Diagrams

<u>Claim</u>: The Voronoi regions, R_p, are convex.

<u>Proof</u>:

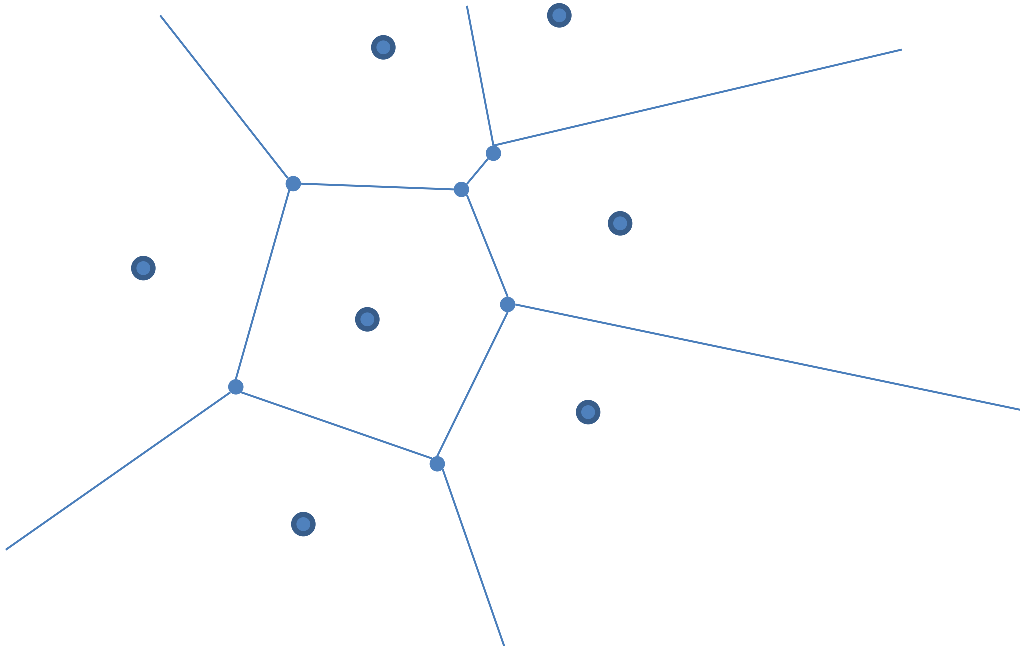
Doing this for all $q \in P$, we get R_p as the intersection of half-spaces.



Voronoi Diagrams

Properties:

- $p \in R_p$ for all $p \in P$.

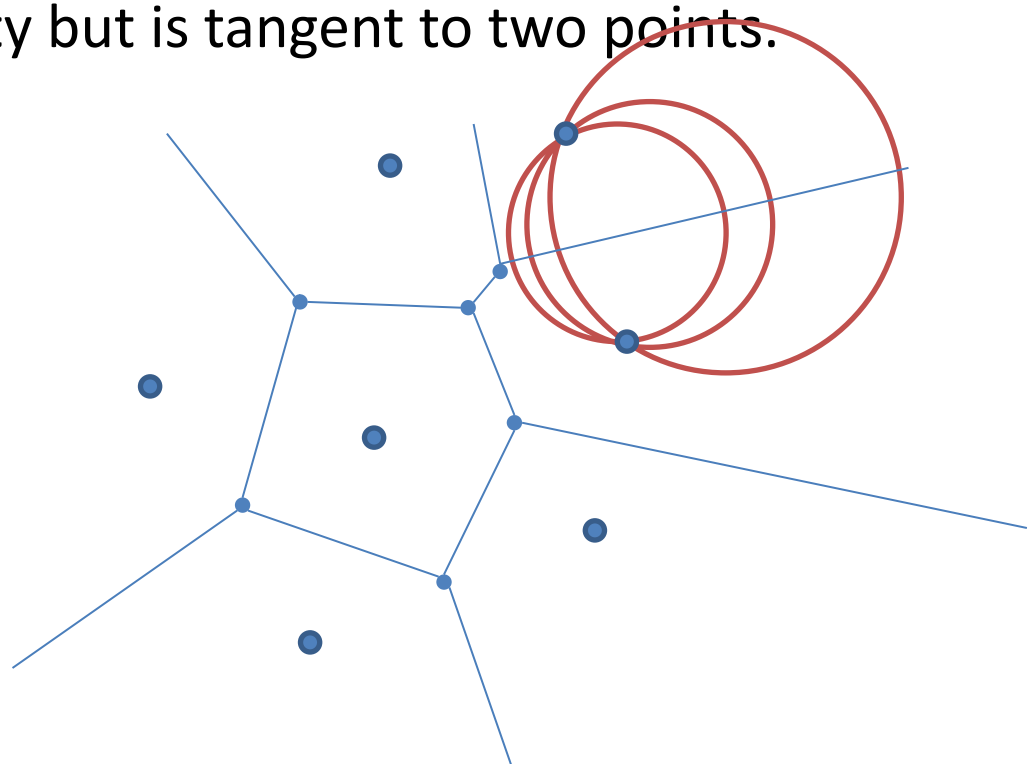


Voronoi Diagrams

<u>Properties</u>:

- Edges are equidistant to two points in P:

  We can center a circle on any point on a Voronoi edge whose interior is empty but is tangent to two points.

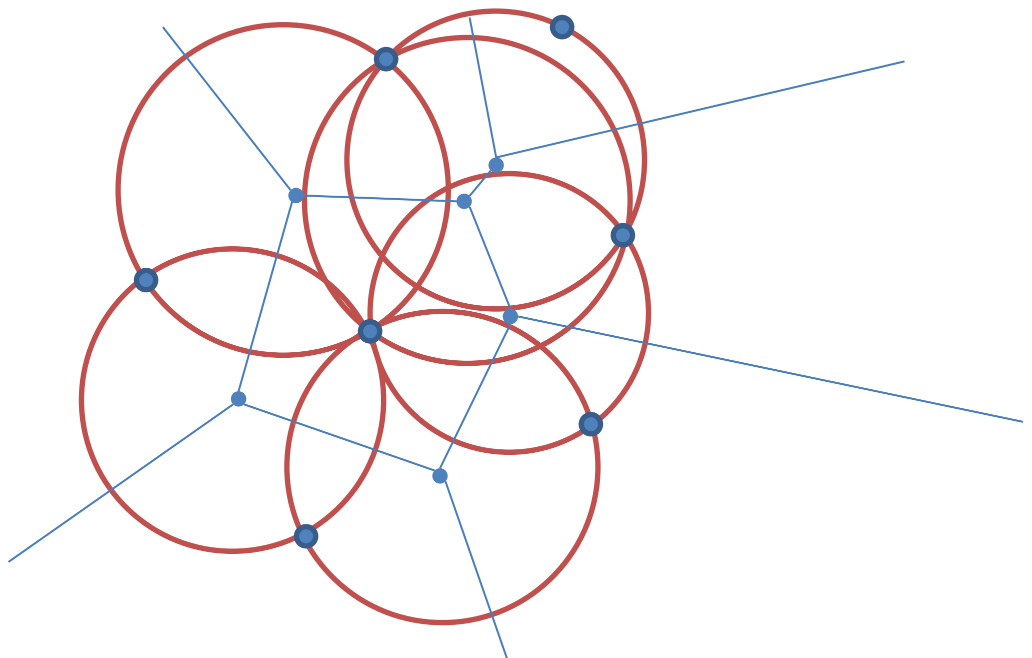


Voronoi Diagrams

<u>Properties</u>:

- Vertices are equidistant to three points in P.

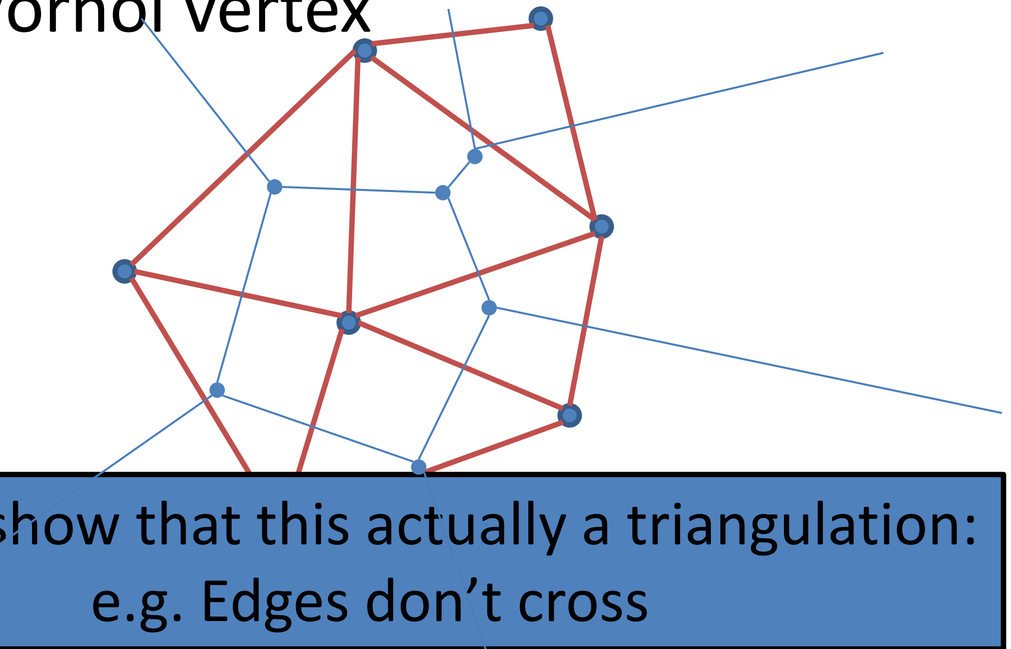
  We can center a circumscribing circle on any Voronoi vertex whose interior is empty but is tangent to three points.



Delaunay Triangulations

We can construct the dual graph by:

- Adding edges between pairs of points defining a Voronoi edge
- Adding triangles between triplets of points defining a Vornoi vertex

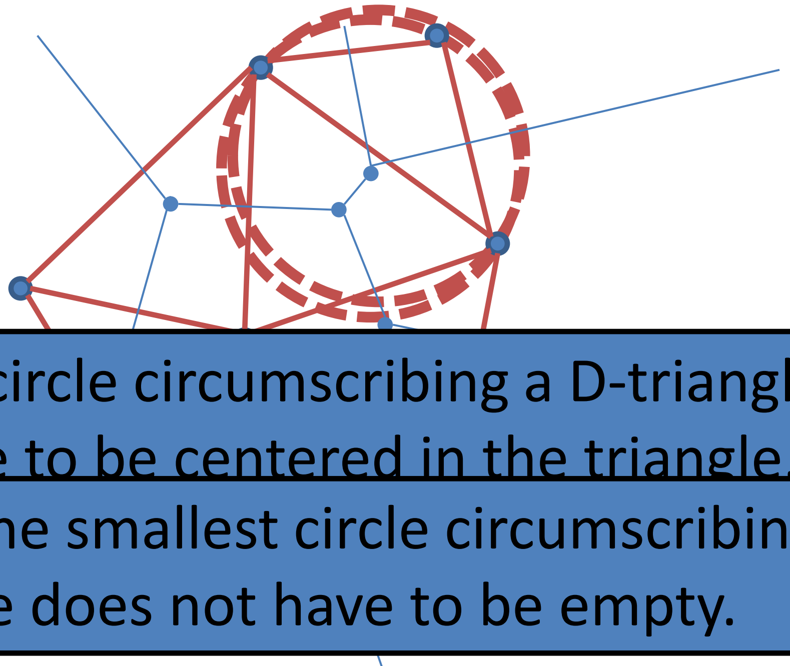


Need to show that this actually a triangulation: e.g. Edges don't cross

Delaunay Triangulations

Properties:

- The DT is a triangulation of the convex hull.
- Each D-edge/triangle can be circumscribed by an empty circle.



Note that the circle circumscribing a D-triangle does not have to be centered in the triangle.

Note that the smallest circle circumscribing a D-edge does not have to be empty.

Computation

- Given a Voronoi diagram, we can obtain the Delaunay triangulation by computing the dual graph, w/ vertices at points $p \in P$.
- Given a Delaunay triangulation, we can obtain the Voronoi diagram by computing the dual graph, w/ vertices at circum-centers.
- How do we compute either?

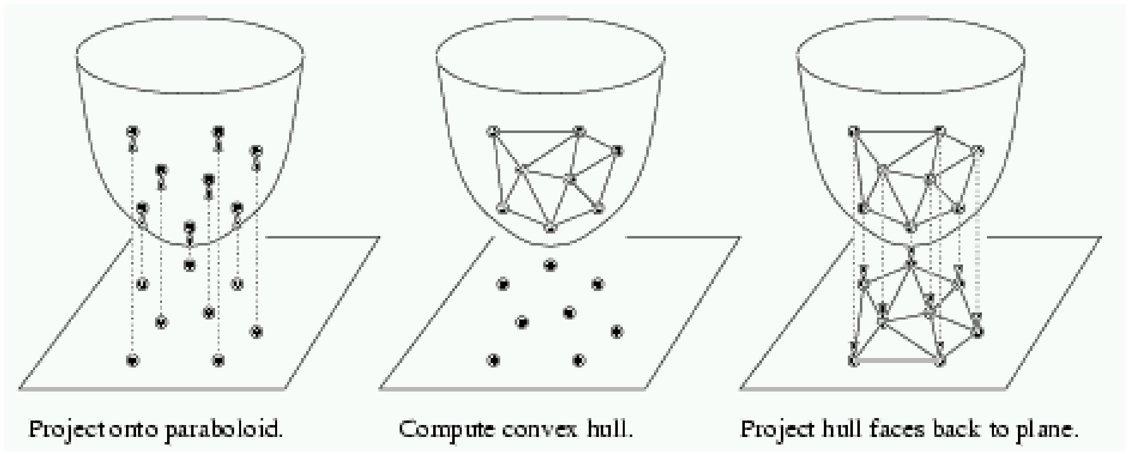
Computation

Claim:

We can compute the Delaunay triangulation of a point set by lifting the points to a paraboloid:

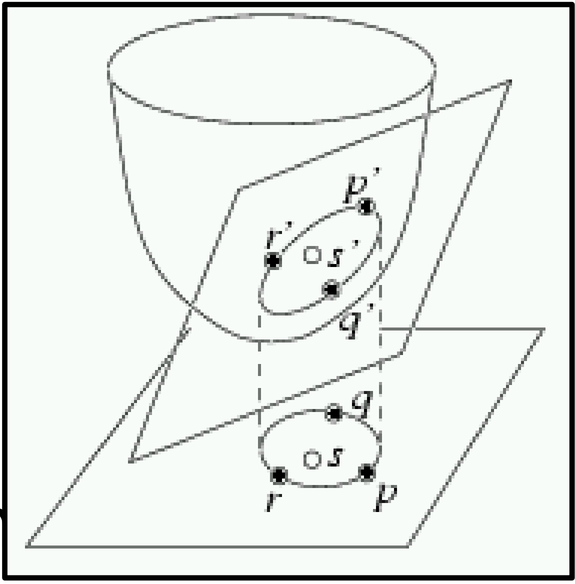
$$(x, y) \to (x, y, x^2 + y^2)$$

and computing the lower convex hull.



http://research.engineering.wustl.edu/~pless/506/l17.html

Computation



Proof:

- Given a point $(a, b, a^2 + b^2)$ on th
  par

The projection of the points of intersection onto the 2D plane is a circle with radius r around (a,b)!

- Shifting the plane up by r^2 we get the plane:

$$z = 2ax + 2by - (a^2 + b^2) + r^2$$

- The shifted plane intersect the paraboloid at:

$$z = x^2 + y^2 = 2ax + 2by - (a^2 + b^2) + r^2$$
$$\Rightarrow (x - a)^2 + (y - b)^2 = r^2$$

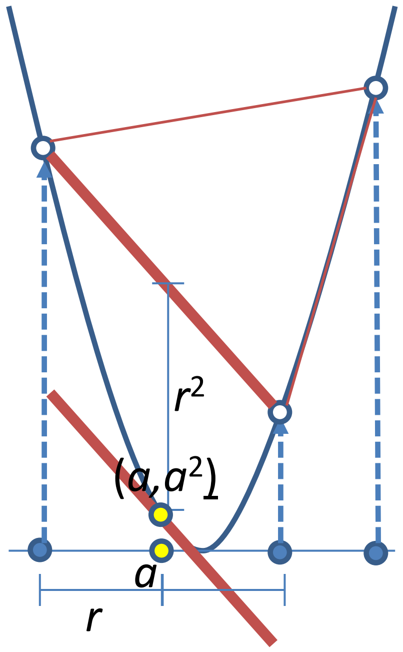
Computation

Proof:

If we have a triangle on the lower convex hull, we can pass a plane through the three vertices.

We can drop the plane by r^2 so that it is tangent to the paraboloid.

Then the projected vertices of the triangle must lie on a circle of radius r around the point (a, b).



Computation

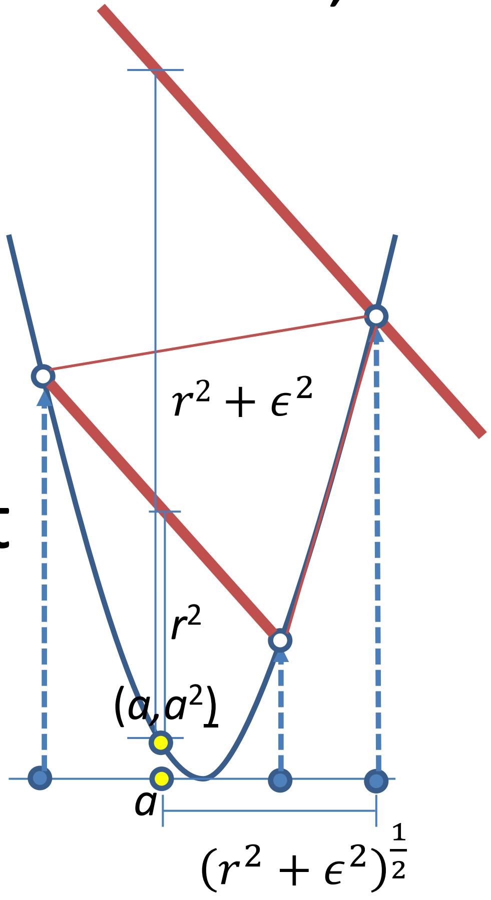
Proof:

Since the original plane was on the lower hull, all other points must be above.

We can raise the plane until it intersects any other point.

The distance from the projection of the point onto the 2D to (a, b) must be larger than r.

The circle of radius r around (a, b) contains no other points.

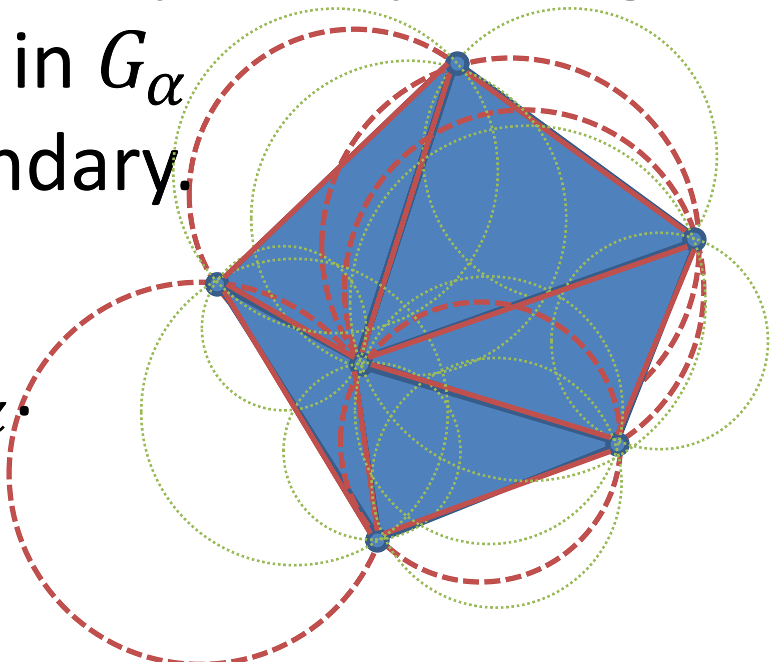


Definition

Given a DT of a set of points, we set G_α to be the simplices whose circum-spheres are empty and whose radii are smaller than α.

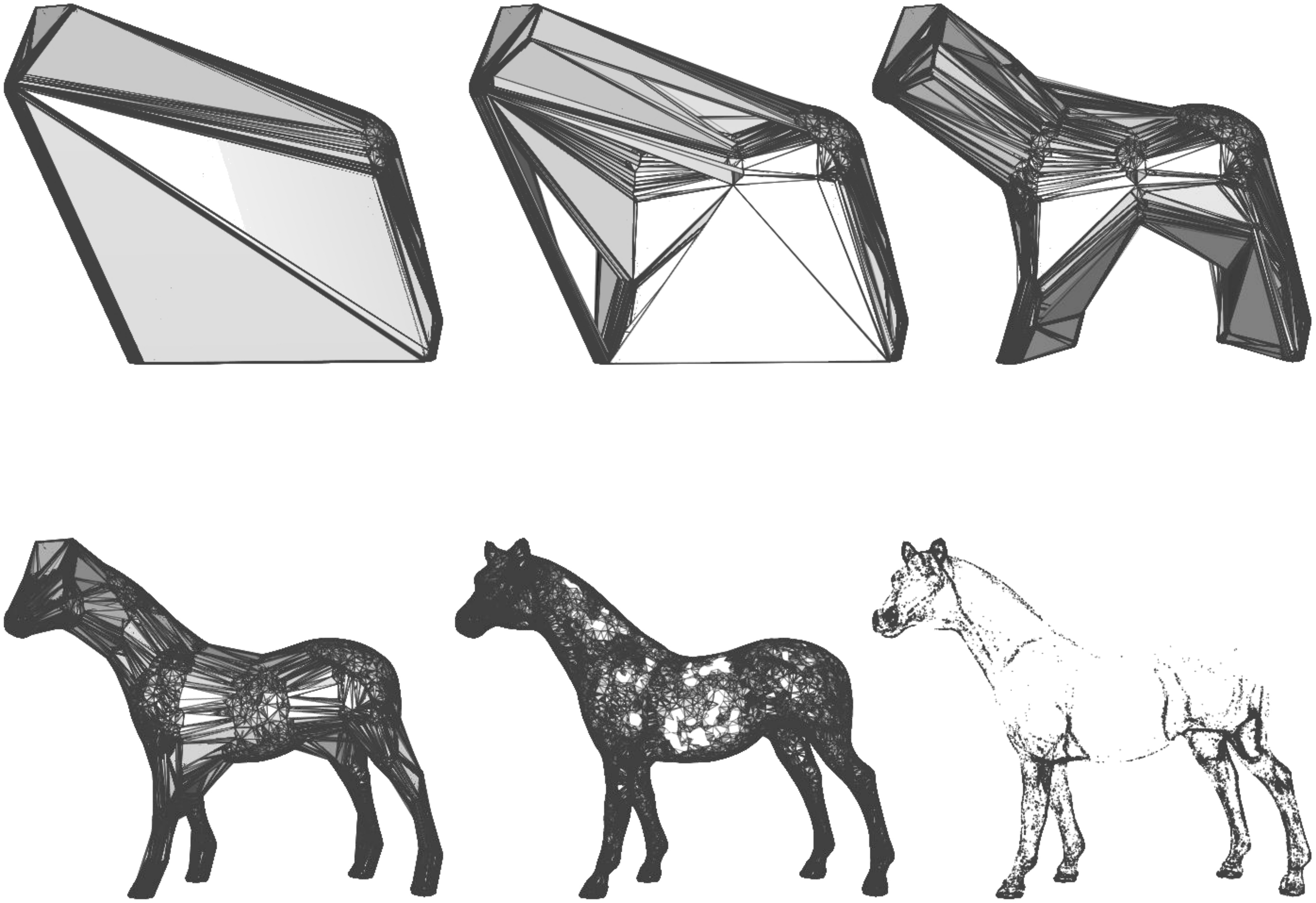
We set the C_α to be the *α-complex*, by taking the union of the elements in G_α with simplices on the boundary.

The *α-shape,* $|C_\alpha|$, is the union of all simplices in C_α.



Alpha Shapes

- Can be comprised of triangles, as well as dangling edges and disconnected vertices. Provides a family of shapes ranging from the convex hull ($\alpha = \infty$) to just the points ($\alpha = 0$).



Contribution

The paper presents a new approach for surface reconstruction. It extends earlier work on 2D alpha shapes, describing implementation in 3D.

Key Idea

The Delauany Triangulation encodes the building blocks for tets/faces/edges of possible reconstructions.

Select from a family of possible reconstructions by grading the simplices and keep those finer than a specified cut-off.

Limitations

- The method is not guaranteed to generate a manifold surface.
- The method implicitly assumes that sampling is uniform.
- It is not clear how well the method works for points sets sampled from a 2D manifold in 3D (can have sliver simplices with large radii).

Guarantees

- The family of shapes nest $(|C_\alpha| \subset |C_\beta|$ for all $\alpha < \beta)$.

Computational Complexity

- The single computation of the Delaunay Triangulation, $O(n^2)$, can be used to define the whole family of shapes.

Generalizability

- The method can be extended to arbitrary dimensions.
- It can be applied to use weighted points (e.g. if sampling density is known)