Convex Hulls (3D)

O'Rourke, Chapter 4

Announcements

- For assignment 1:
  I have posted additional polygon files for testing.

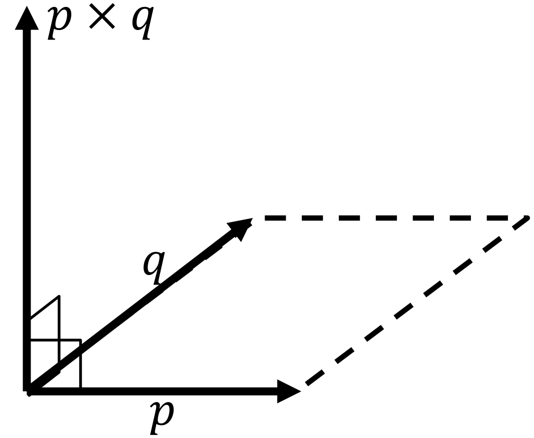
Outline

- Review
- Gift-Wrapping
- Divide-and-Conquer

Recall

Given points $p, q \in \mathbb{R}^3$, the *cross-product* $p \times q \in \mathbb{R}^3$ is the vector:

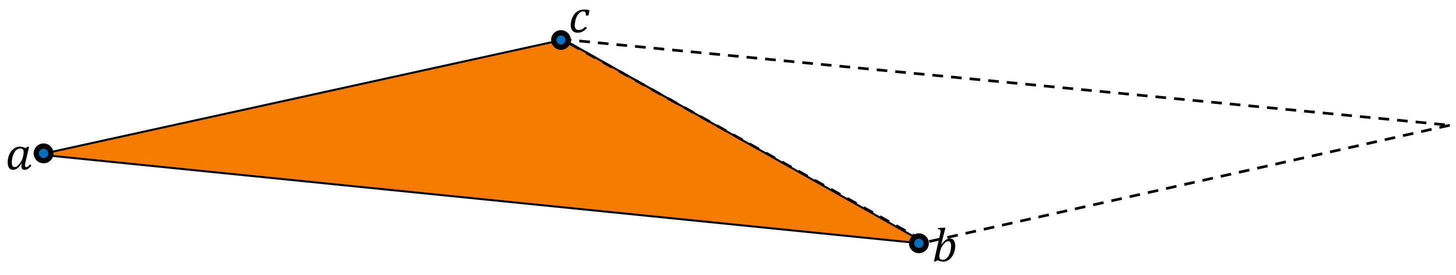
- perpendicular to both p and q,
- oriented according to the right-hand-rule,
- with length equal to the area of the parallelogram defined by p and q.



Recall

Given a triangle T with vertices $(a, b, c) \in \mathbb{R}^3$, the area of the triangle is:

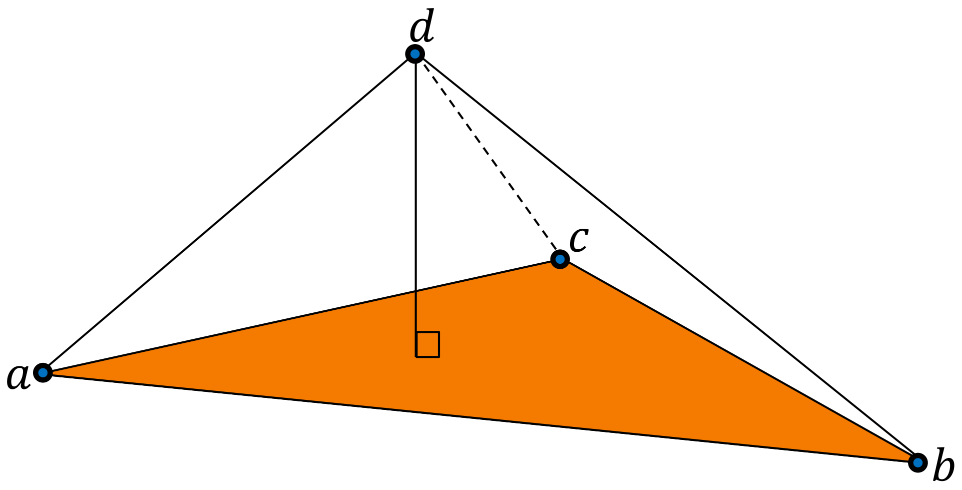
$$\text{Area}(T) = \frac{1}{2} \times \|(b - a) \times (c - a)\|$$



Recall

Given a tetrahedron T with vertices $(a, b, c, d) \in \mathbb{R}^3$, the volume of the tetrahedron is:

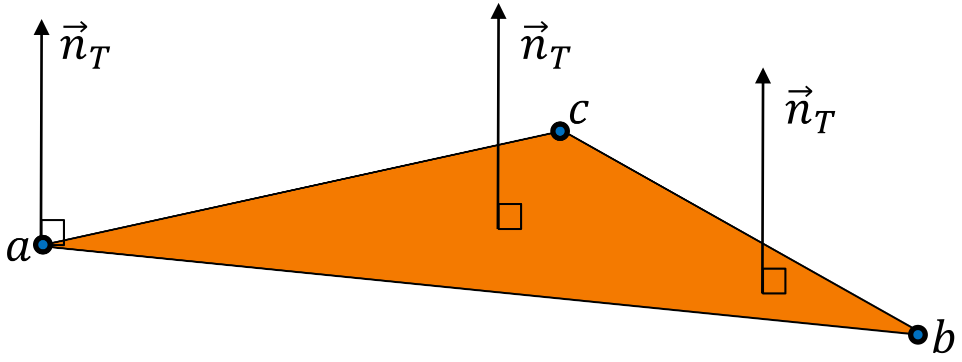
$$\text{Volume}(T) = \frac{1}{3} \times \text{base} \times \text{height}$$



Recall

Given a triangle T with vertices $(a, b, c) \in \mathbb{R}^3$, the triangle normal is:

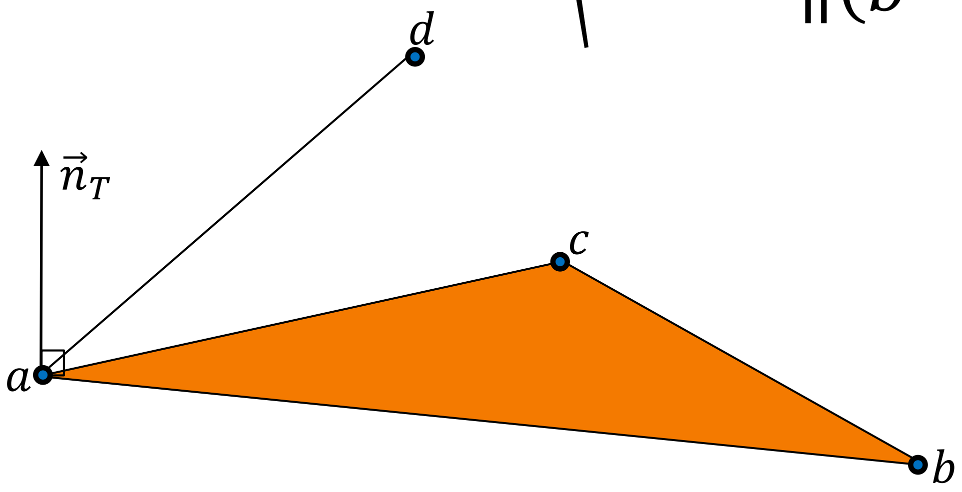
$$\vec{n}_T = \frac{(b-a)\times(c-a)}{\|(b-a)\times(c-a)\|}$$



Recall

Given a triangle T with vertices $(a, b, c) \in \mathbb{R}^3$ and given a point $d \in \mathbb{R}^3$, the signed perpendicular height of d from the plane containing (a, b, c) is:

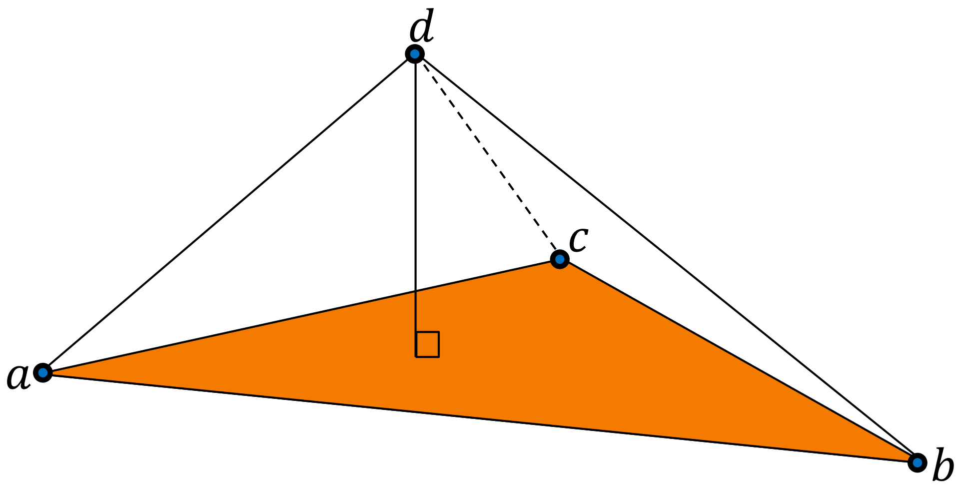
$$\begin{aligned}\text{Height}(T, d) &= \langle d - a, \vec{n}_T \rangle \\ &= \left\langle d - a, \frac{(b-a) \times (c-a)}{\|(b-a) \times (c-a)\|} \right\rangle\end{aligned}$$



Recall

Given a tetrahedron T with vertices $(a, b, c, d) \in \mathbb{R}^3$, the signed volume of the tetrahedron is:

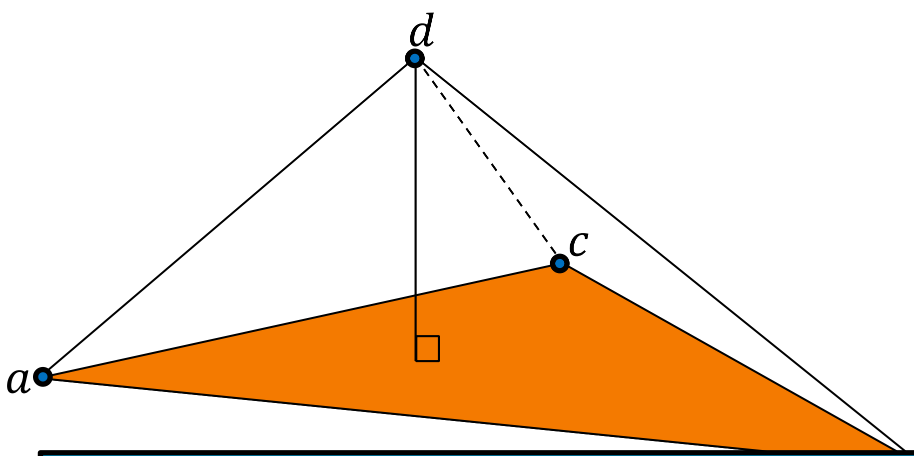
$$\text{Volume}(T) = \frac{1}{3} \times \text{base} \times \text{height}$$
$$= \frac{1}{6} \times \langle d - a, (b - a) \times (c - a) \rangle$$



Recall

Given a tetrahedron T with vertices $(a, b, c, d) \in \mathbb{R}^3$, the signed volume of the tetrahedron is:

$$\begin{aligned}\text{Volume}(T) &= \frac{1}{3} \times \text{base} \times \text{height} \\ &= \frac{1}{6} \times \langle d - a, (b - a) \times (c - a) \rangle\end{aligned}$$



The volume is positive if d is to the left of the plane defined by the triangle (a, b, c).

Recall

If we have a graph G, we can identify the connected component containing a node v by performing a flood-fill.

```
FloodFill( v , G )
    if( NotMarked( v ) )
        Mark( v )
        for w ∈ Neighbors( v )
            FloodFill( w , G )
```

Complexity: $O(|E|)$

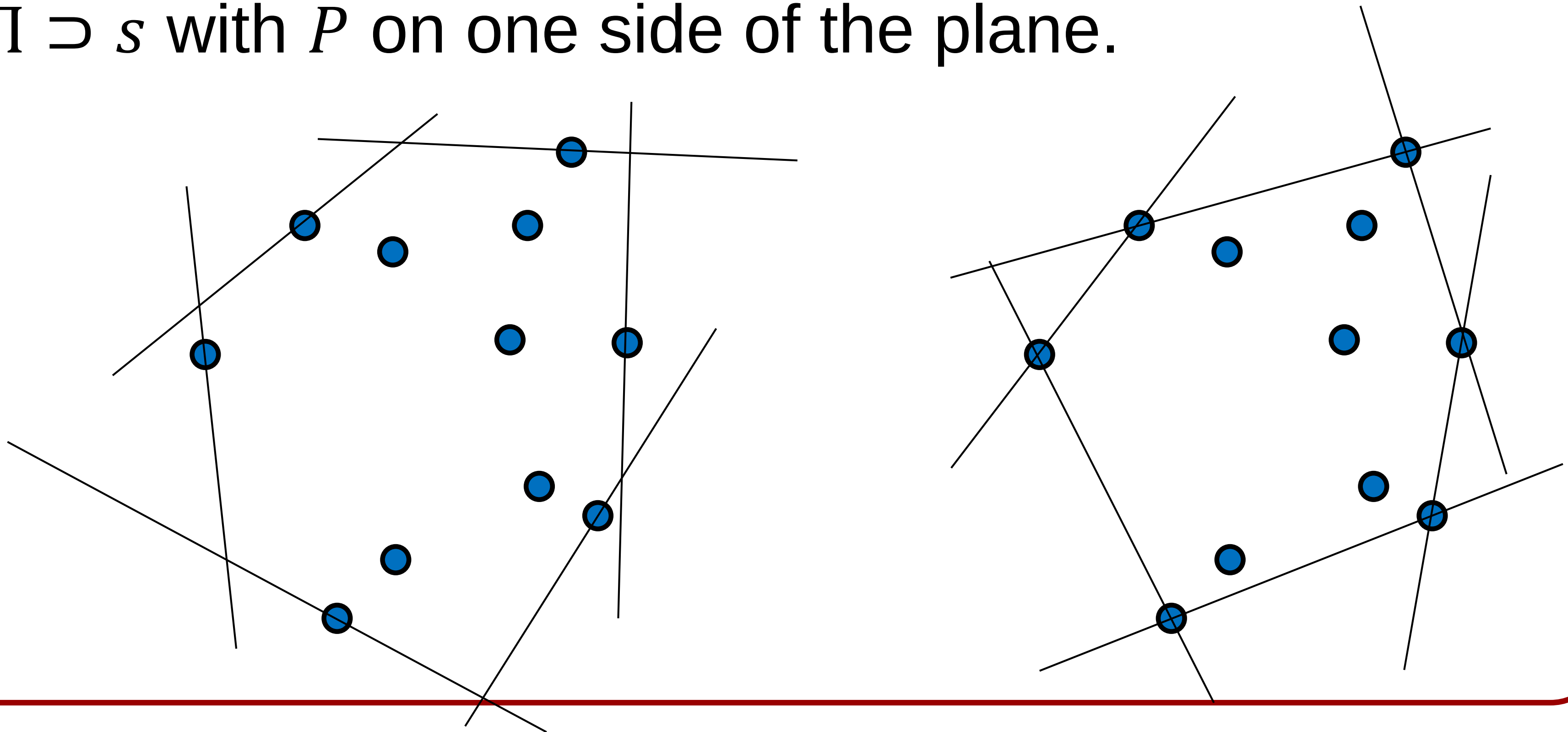
Recall

If we have a graph G, we can identify the connected component containing a node v by performing a flood-fill.

In particular, given a winged-edge representation of a triangle mesh and given a face in the mesh, we can compute the connected component of the face in linear time.

Supporting Simplices

Given a set of points $P \subset \mathbb{R}^d$, and given a simplex $s = \{p_1, \cdots, p_k\}$ (vertex, edge, triangle, etc.) formed by $k \leq d$ vertices, we say that the P is <u>supported</u> on s if there exists a $(d-1)$-dimensional hyperplane, $\Pi \supset s$ with P on one side of the plane.



Supporting Simplices

Note:

If we project P' is the projection of P onto $\mathbb{R}^{d'}$ and if P' is supported on a simplex $s' = \{p'_1, \dots, p'_k\}$ then P is supported on the simplex $s = \{p_1, \dots, p_k\}$.

Proof:

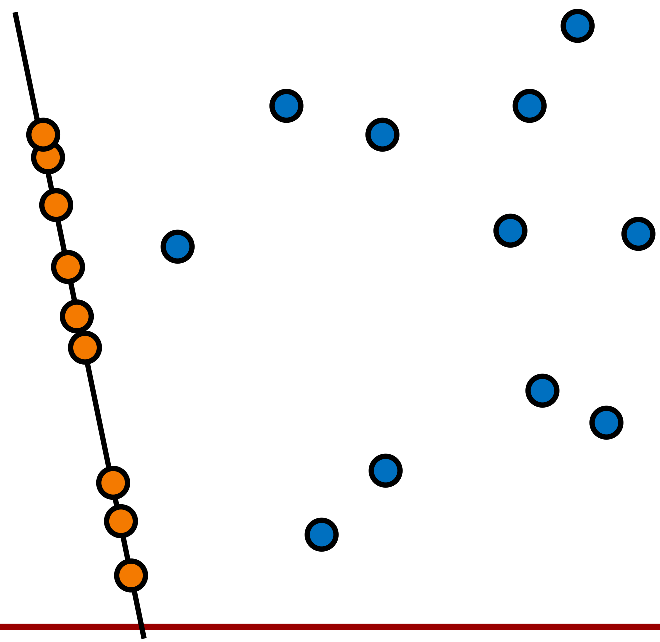
Extrude the $(d'-1)$-dimensional Π' along the direction of projection.

- The $(d-1)$-dimensional hyperplane Π has P on one side.
- The vertices of s lie on Π.

Supporting Simplices

<u>Note</u>:

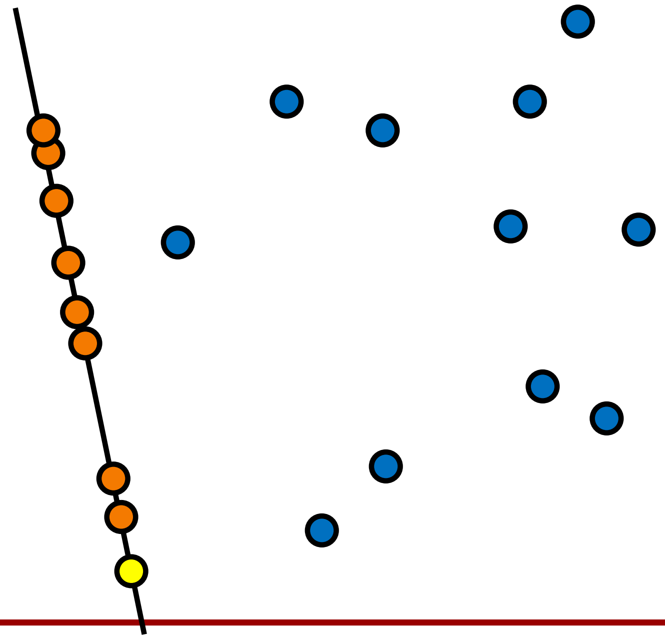
If we project P' is the projection of P onto $\mathbb{R}^{d'}$ and if P' is supported on a simplex $s' = \{p'_1, \ldots, p'_k\}$ then P is supported on the simplex $s = \{p_1, \ldots, p_k\}$.



Supporting Simplices

Note:

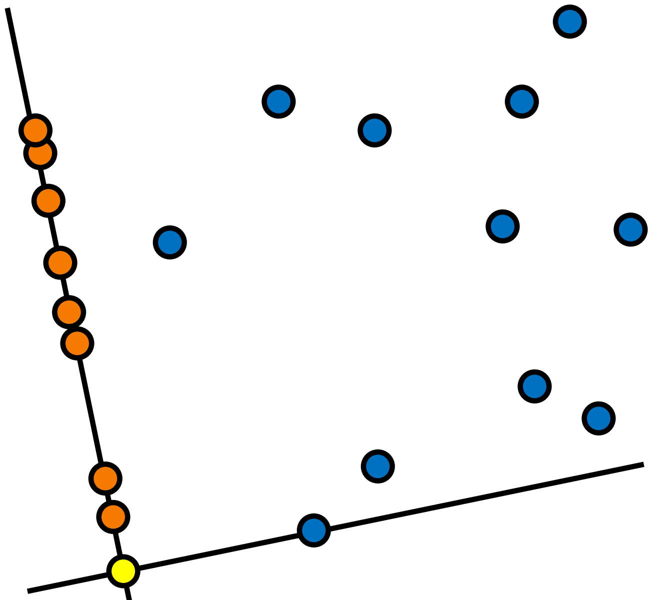
If we project P' is the projection of P onto $\mathbb{R}^{d'}$ and if P' is supported on a simplex $s' = \{p'_1, \ldots, p'_k\}$ then P is supported on the simplex $s = \{p_1, \ldots, p_k\}$.



Supporting Simplices

Note:

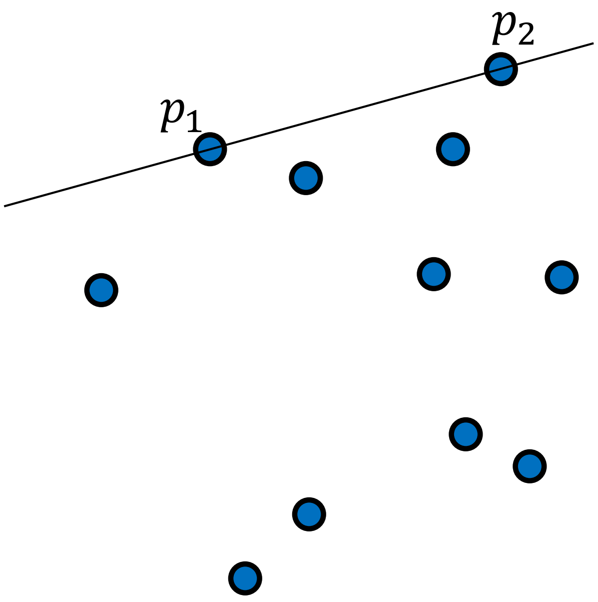
If we project P' is the projection of P onto $\mathbb{R}^{d'}$ and if P' is supported on a simplex $s' = \{p'_1, \ldots, p'_k\}$ then P is supported on the simplex $s = \{p_1, \ldots, p_k\}$.



Supporting Simplices

Note:

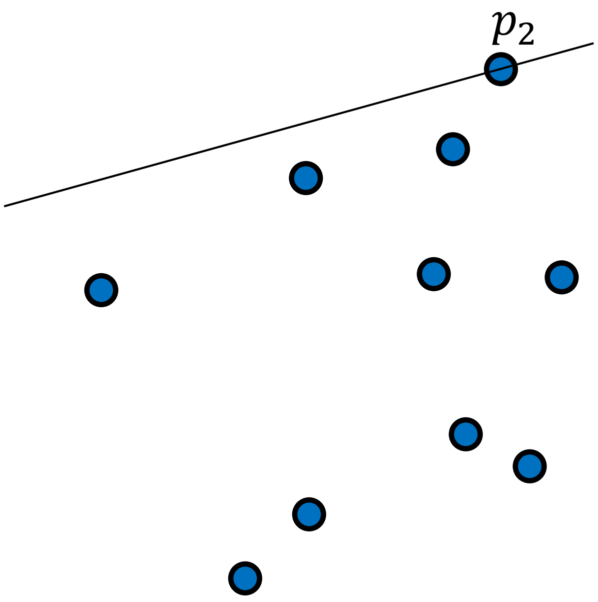
If P is supported by the simplex $s = \{p_1, \ldots, p_k\}$ then the point-set $P' = P - \{p_1\}$ is supported by the simplex $s' = \{p_2, \ldots, p_k\}$.



Supporting Simplices

Note:

If P is supported by the simplex $s = \{p_1, \ldots, p_k\}$ then the point-set $P' = P - \{p_1\}$ is supported by the simplex $s' = \{p_2, \ldots, p_k\}$.



Outline

- Review
- Gift-Wrapping
- Divide-and-Conquer

Gift-Wrapping

Initialization:

Find a triangle on the hull.

Iteratively:

Until the hull closes, pivot around a boundary edge.

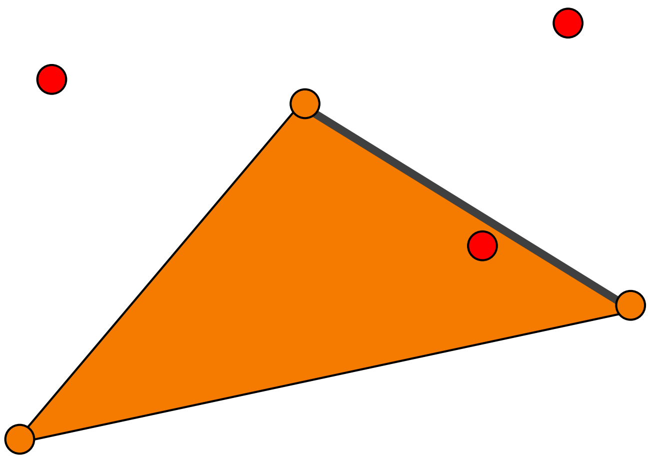
Gift-Wrapping

Initialization:

Find a triangle on the hull.

Iteratively:

Until the hull closes, pivot around a boundary edge.



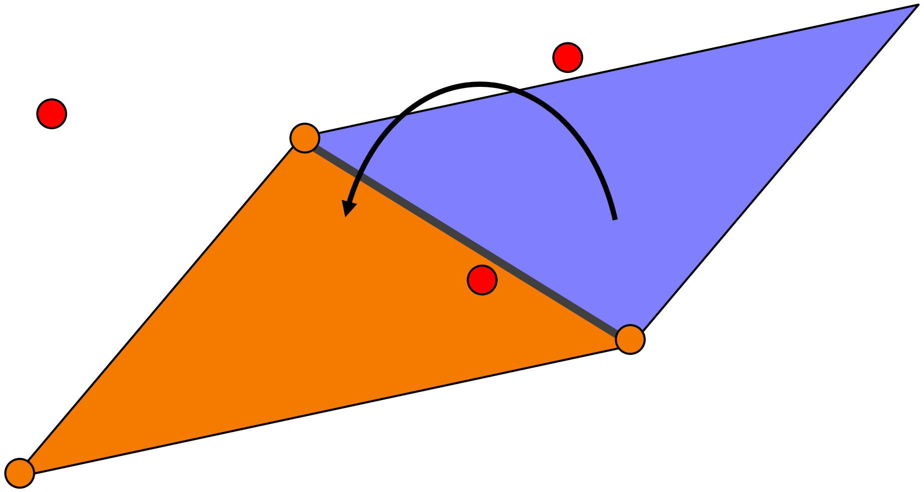
Gift-Wrapping

Initialization:

Find a triangle on the hull.

Iteratively:

Until the hull closes, pivot around a boundary edge.



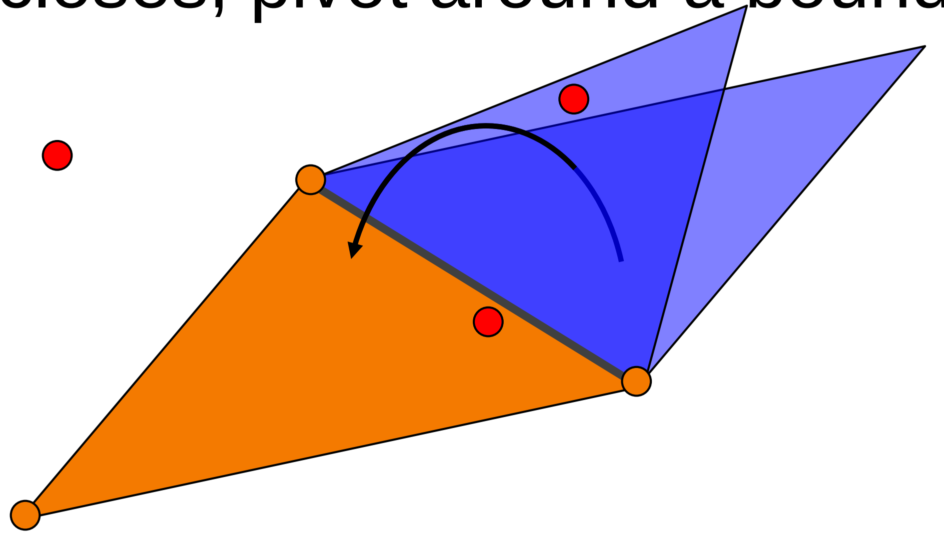
Gift-Wrapping

Initialization:

Find a triangle on the hull.

Iteratively:

Until the hull closes, pivot around a boundary edge.



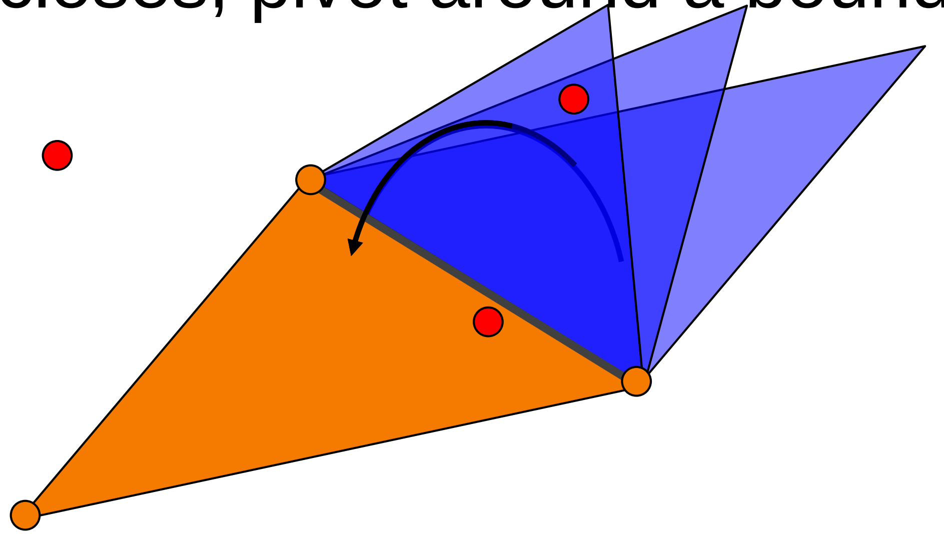
Gift-Wrapping

Initialization:

Find a triangle on the hull.

Iteratively:

Until the hull closes, pivot around a boundary edge.



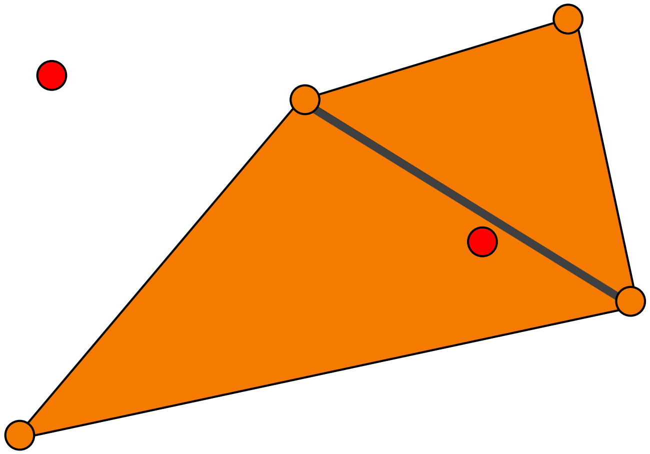
Gift-Wrapping

Initialization:

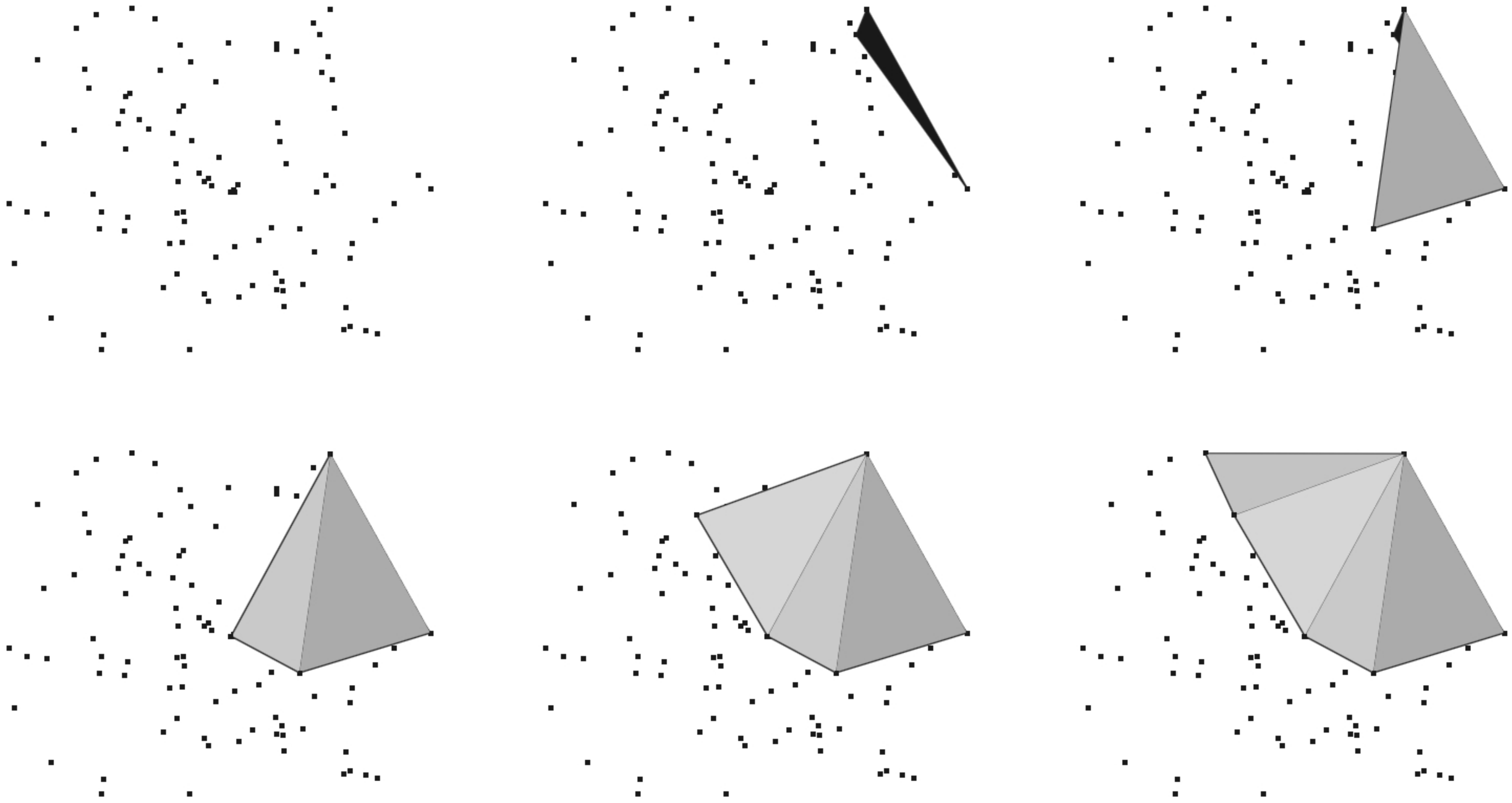
Find a triangle on the hull.

Iteratively:

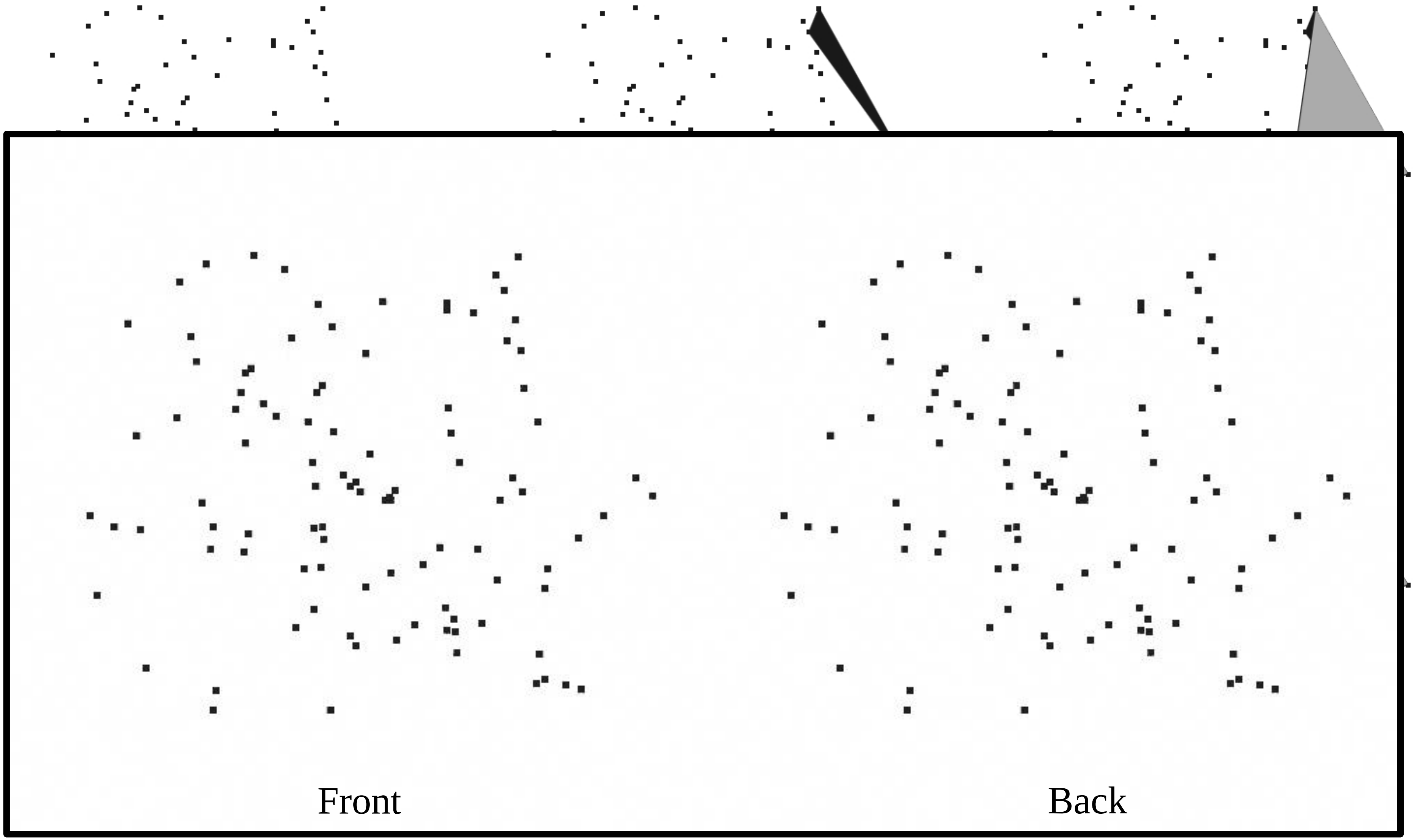
Until the hull closes, pivot around a boundary edge.



Gift-Wrapping



Gift-Wrapping



Gift-Wrapping

PivotAroundEdge($e = \{q_0, q_1\}$, $P = \{p_0, \dots, p_{n-1}\}$)

```
p ← p_0
area2 ← SquaredArea( q_0 , q_1 , p )
for p' ∈ {p_1, …, p_{n-1}}:
    volume ← SignedVolume( q_0 , q_1 , p , p' )
    if( volume<0 )
      p ← p'
    else if( volume==0 )
      _area2 ← SquaredArea( q_0 , q_1 , p' )
      if( _area2>area2 )
        p ← p'
        area2 ← _area2
return p
```

Complexity: $O(n)$

Gift-Wrapping

FindTriangleOnHull($P = \{p_0, \dots, p_{n-1}\}$)

$\{p, q\} \leftarrow$ FindEdgeOnHull(P)

$r \leftarrow$ PivotAroundEdge($\{p, q\}$, P)

return $\{p, q, r\}$

Complexity: $O(n)$ + Complexity of FindEdgeOnHull

Gift-Wrapping

FindEdgeOnHull($P = \{p_0, \dots, p_{n-1}\}$)

$p \leftarrow$ BottomMostLeftMostBackMost(P)

$q \leftarrow$ PivotOnEdge($\{p, p + (1,0,0)\}$, P)

return $\{p, q\}$

Complexity: $O(n)$

Gift-Wrapping

GiftWrap(P):

- $t \leftarrow$ FindTriangleOnHull(P)
- $Q \leftarrow \{(t_1, t_0), (t_2, t_1), (t_0, t_2)\}$ // hull boundary edges (?)
- $H \leftarrow \{t\}$ // the hull
- while($Q \neq \emptyset$)
  - $e \leftarrow Q$.pop_back()
  - if(NotProcessed(e))
    - $q \leftarrow$ PivotOnEdge(e)
    - $t \leftarrow$ Triangle(e , q)
    - $H \leftarrow H \cup \{t\}$
    - $Q \leftarrow Q \cup \{(t_1, t_0), (t_2, t_1), (t_0, t_2)\}$
    - MarkProcessedEdges(e)

Complexity: $O(n^2)$

Outline

- Review
- Gift-Wrapping
- Divide-and-Conquer

Divide And Conquer

DivideAndConquer(P):

$P \leftarrow$ SortByX(P)

return _DivideAndConquer(P)

_DivideAndConquer(P)

if($|P| < 8$) return Incremental(P)

$(P_1, P_2) \leftarrow$ SplitInHalf(P)

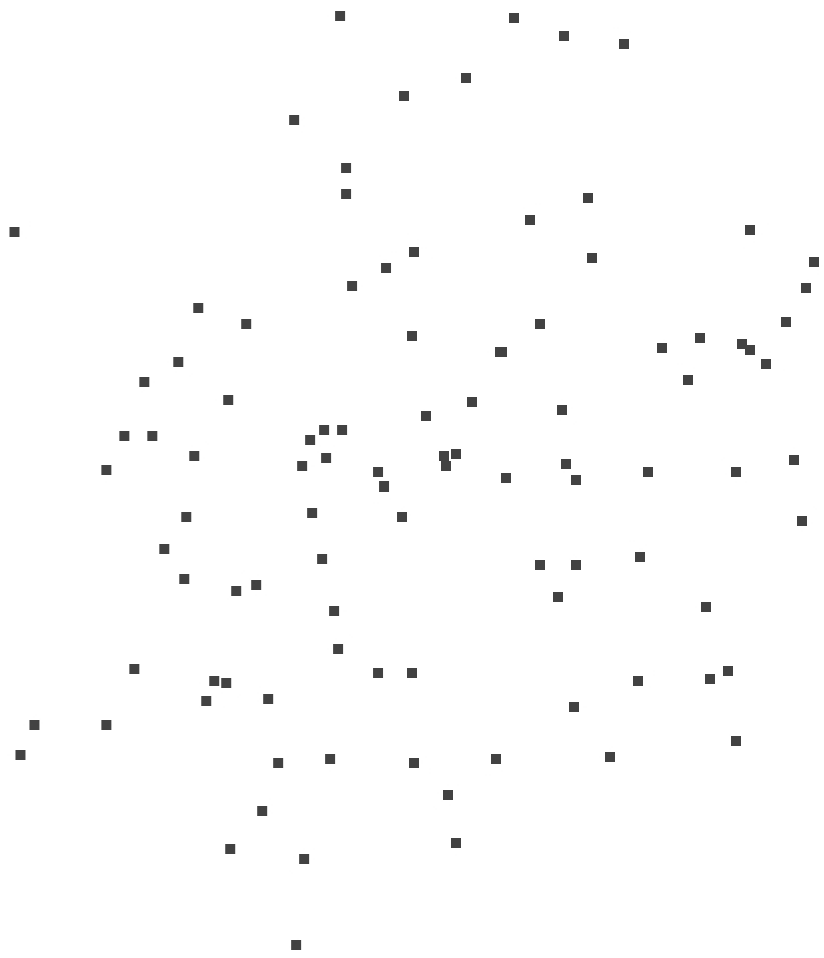
$H_1 \leftarrow$ _DivideAndConquer(P_1)

$H_2 \leftarrow$ _DivideAndConquer(P_2)

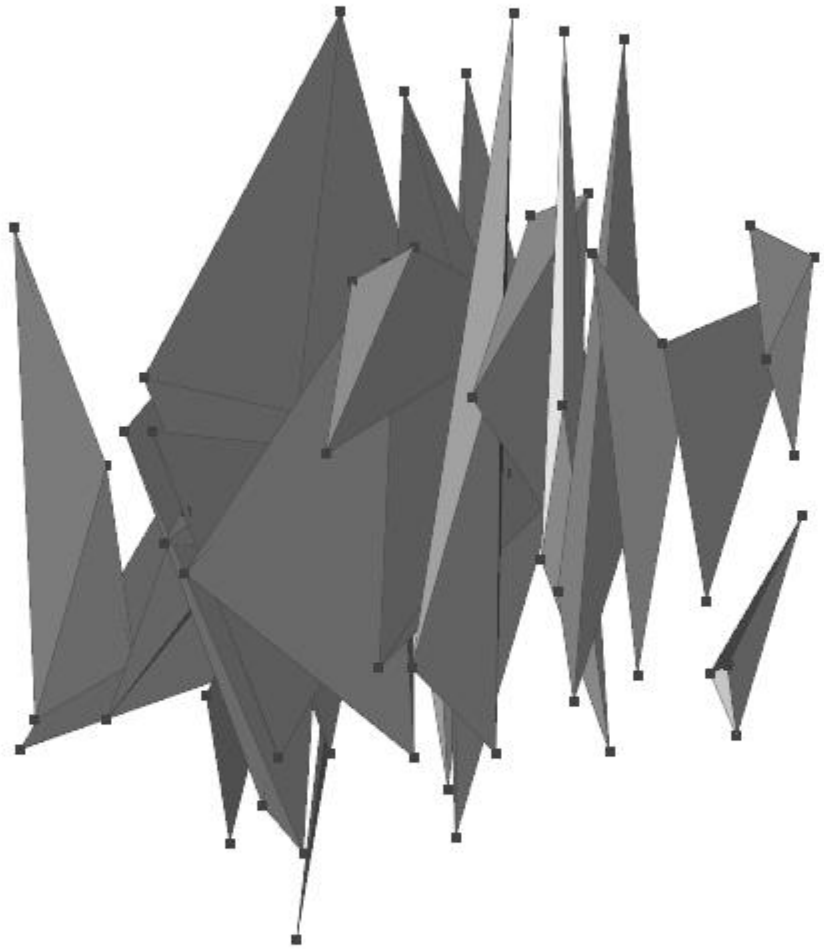
return Merge(H_1 , H_2)

Complexity: $O(n \log n)$

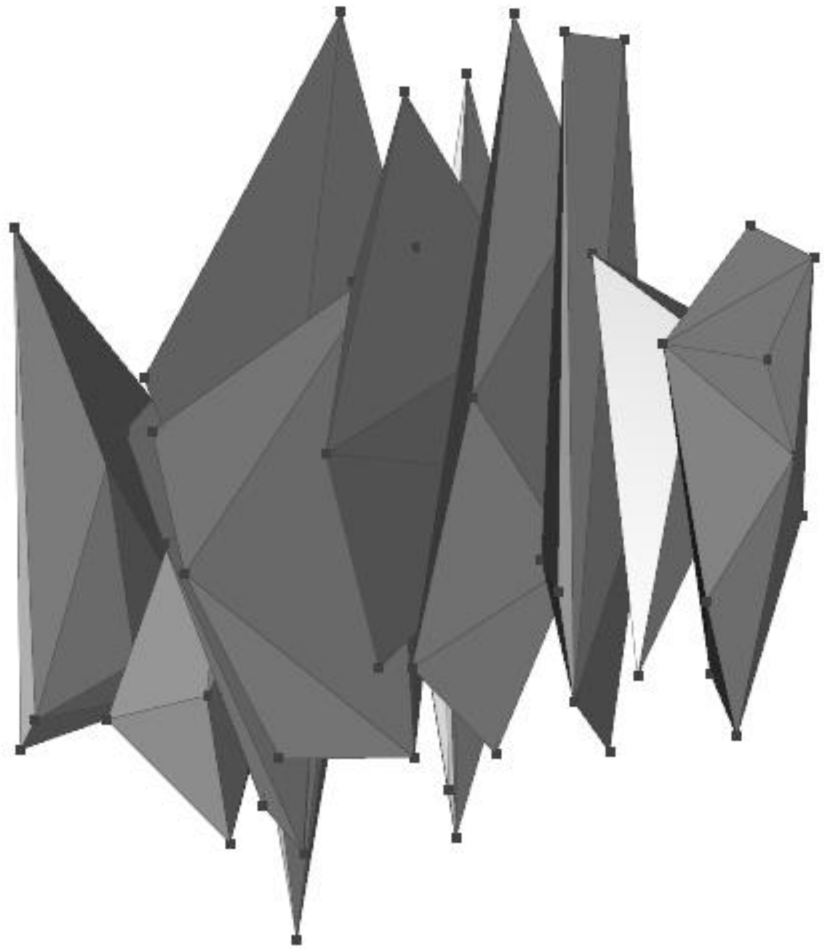
Divide And Conquer



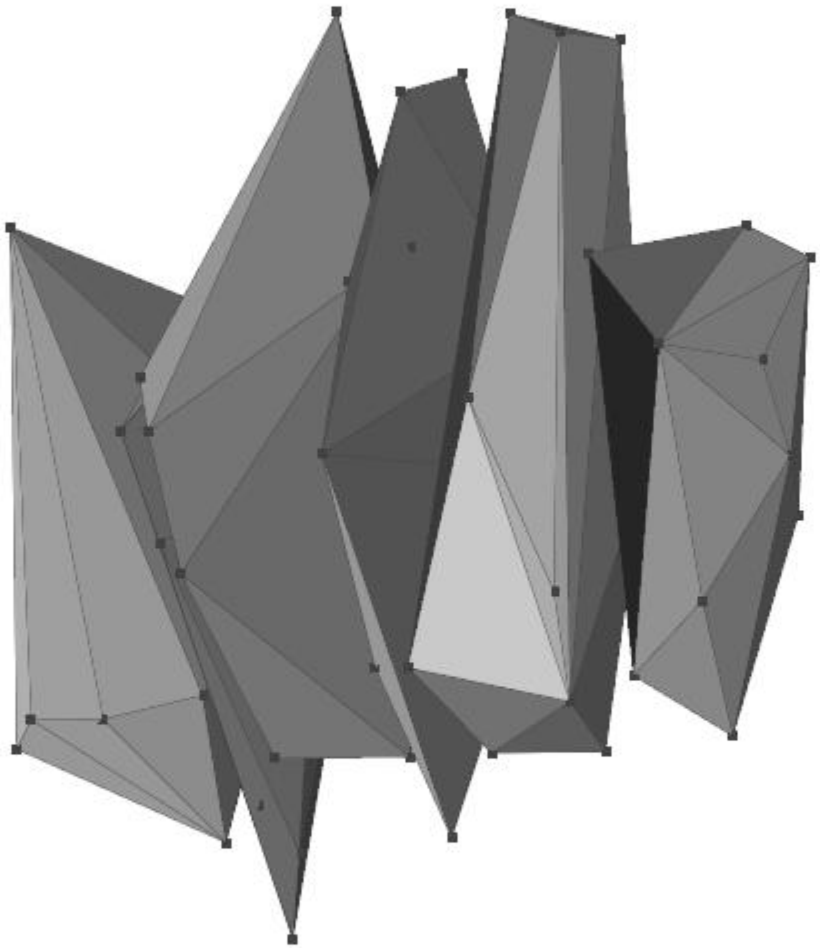
Divide And Conquer



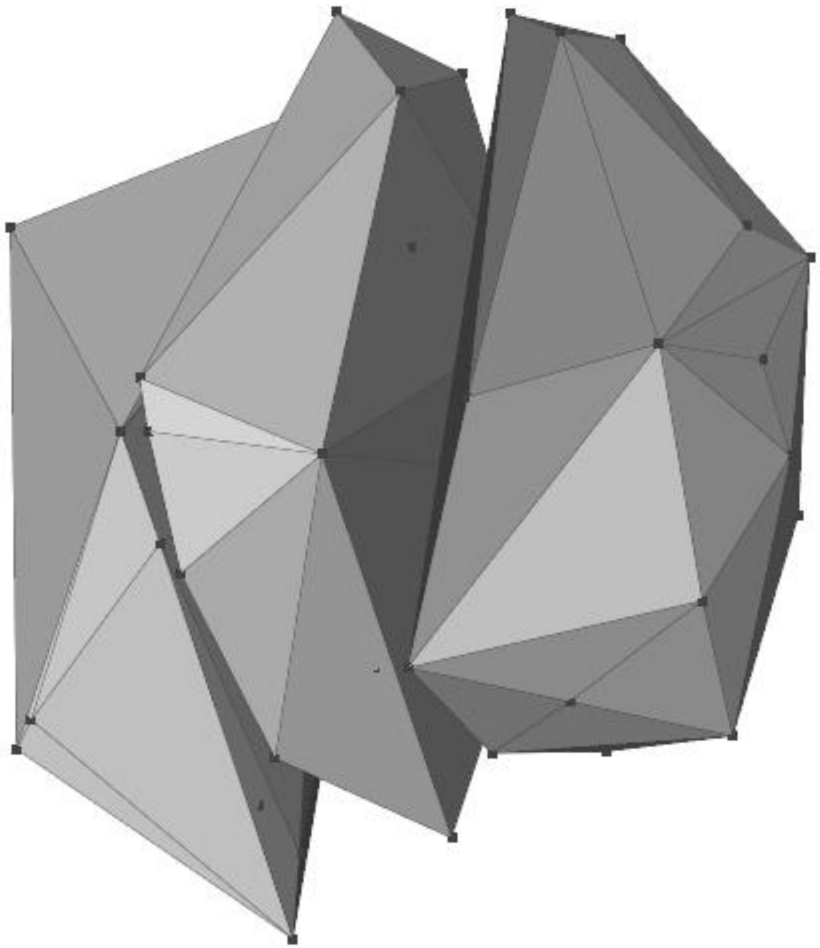
Divide And Conquer



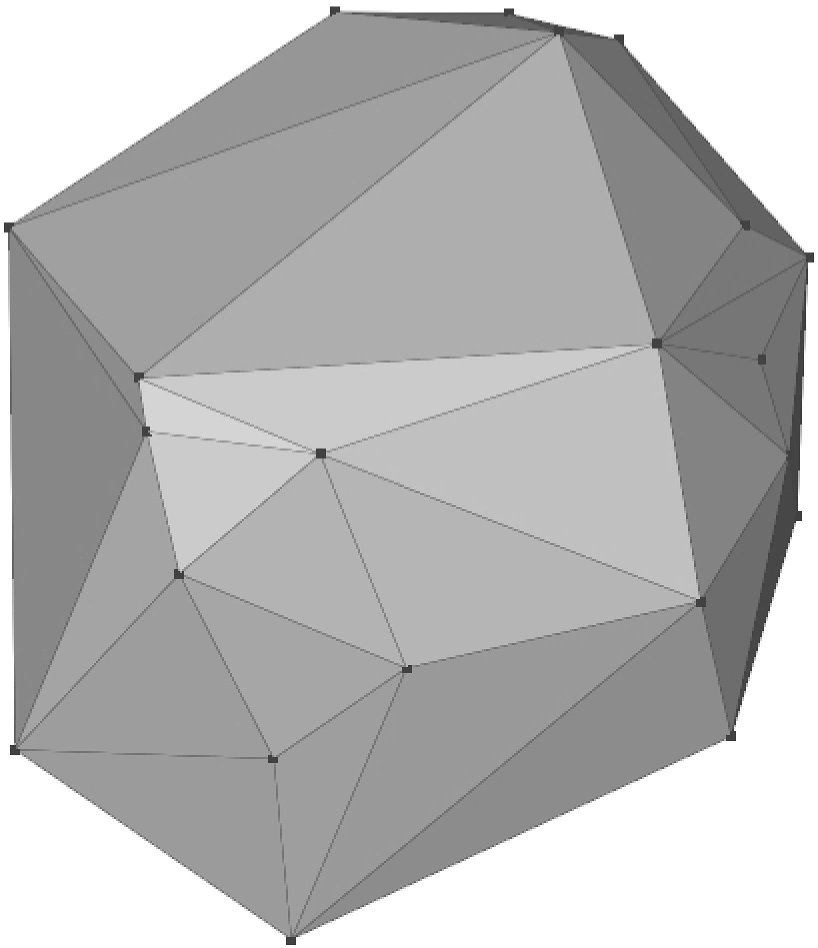
Divide And Conquer



Divide And Conquer



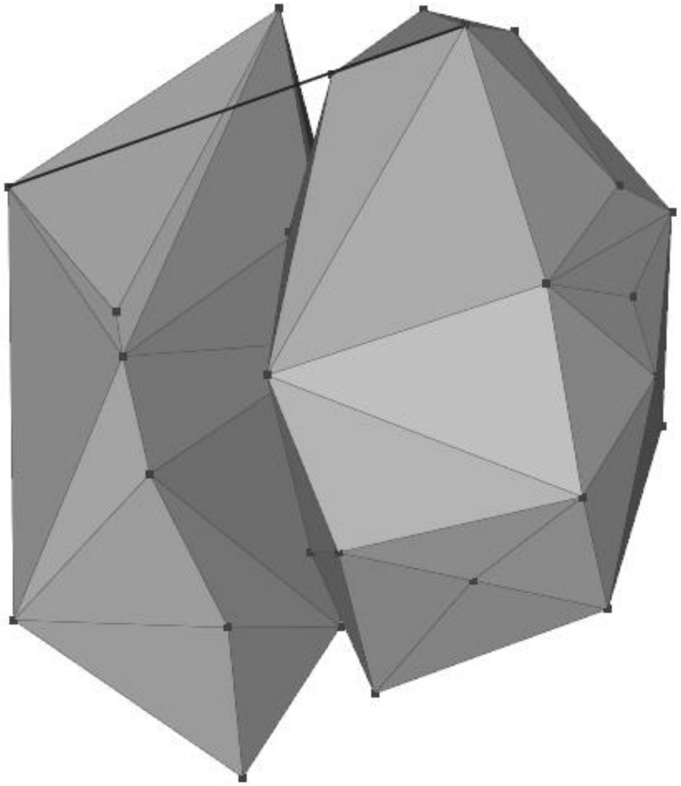
Divide And Conquer



Divide And Conquer

Merge:

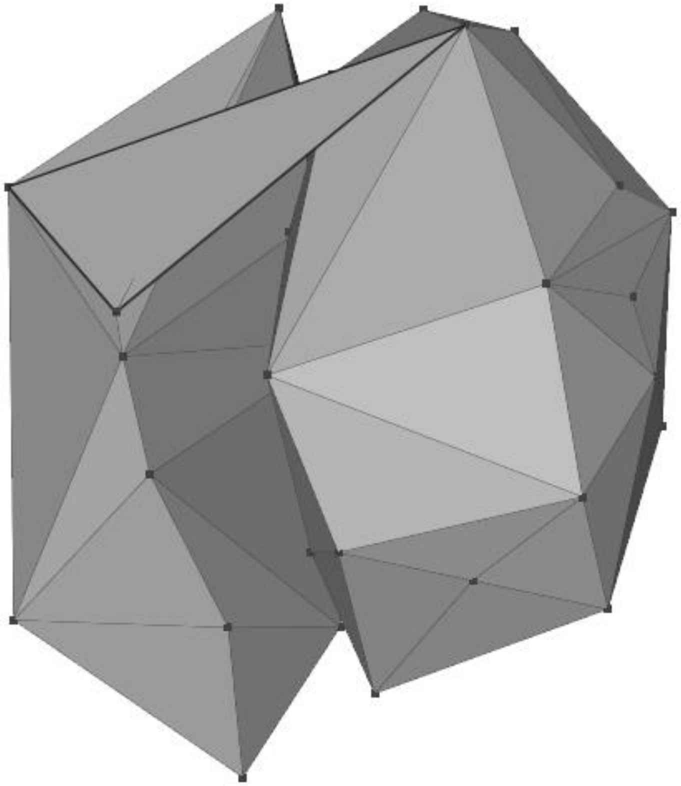
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

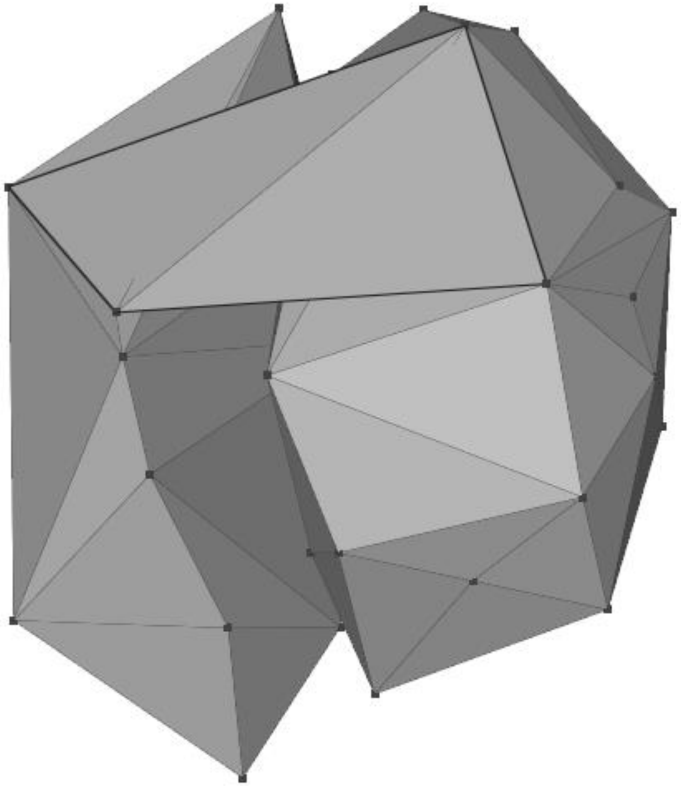
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

<u>Merge</u>:

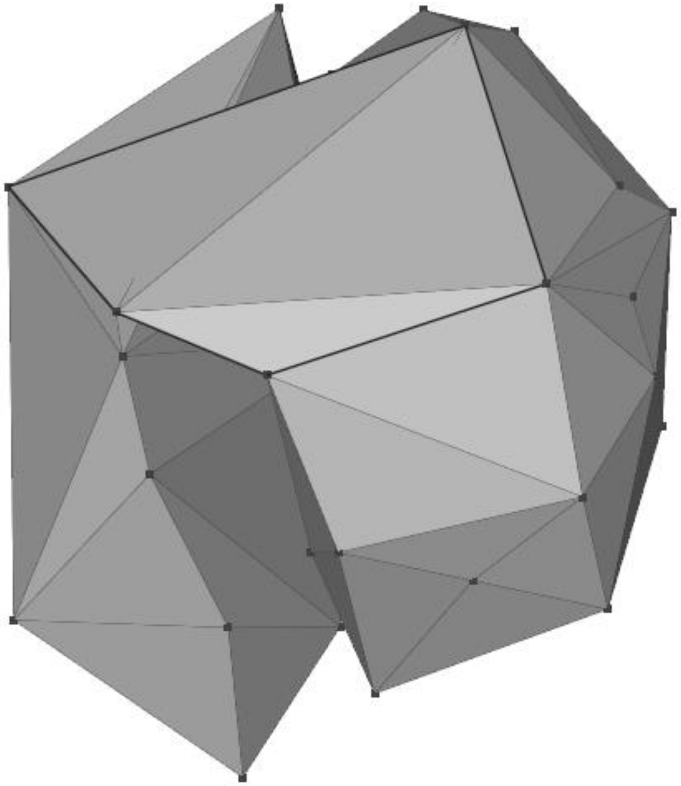
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

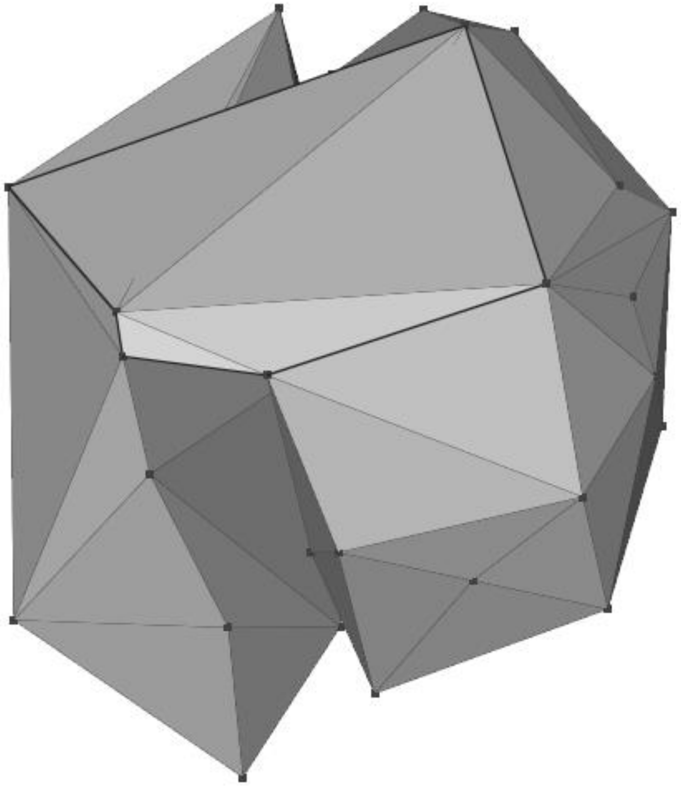
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

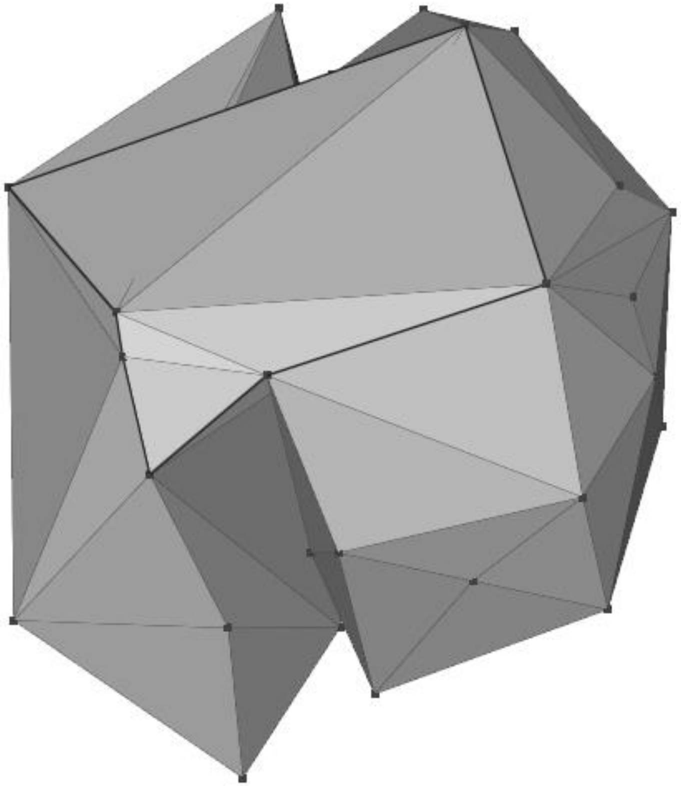
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

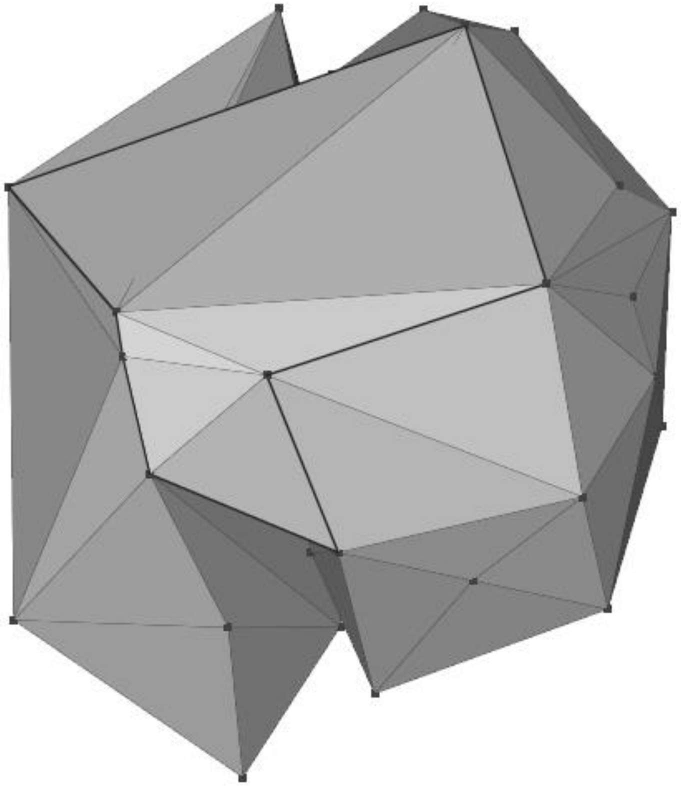
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

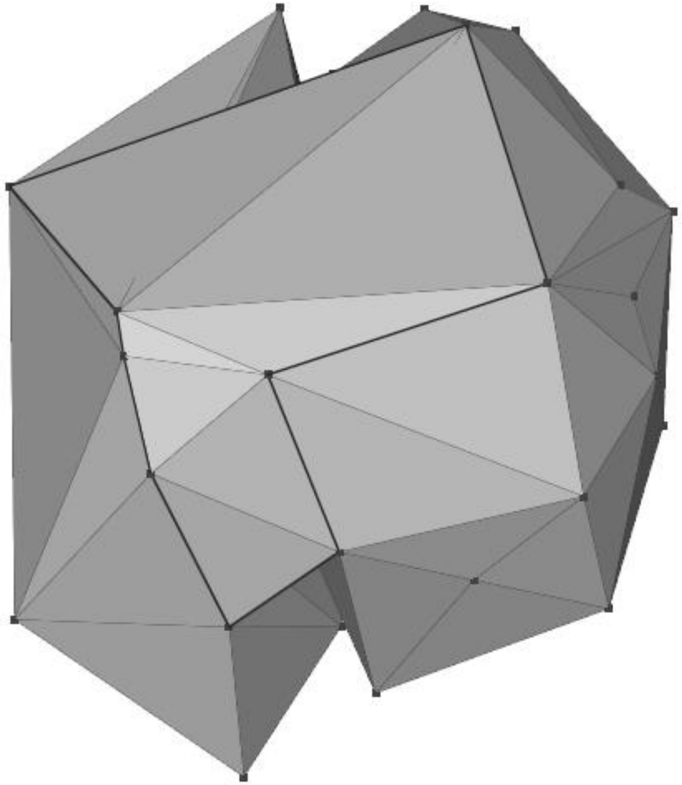
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

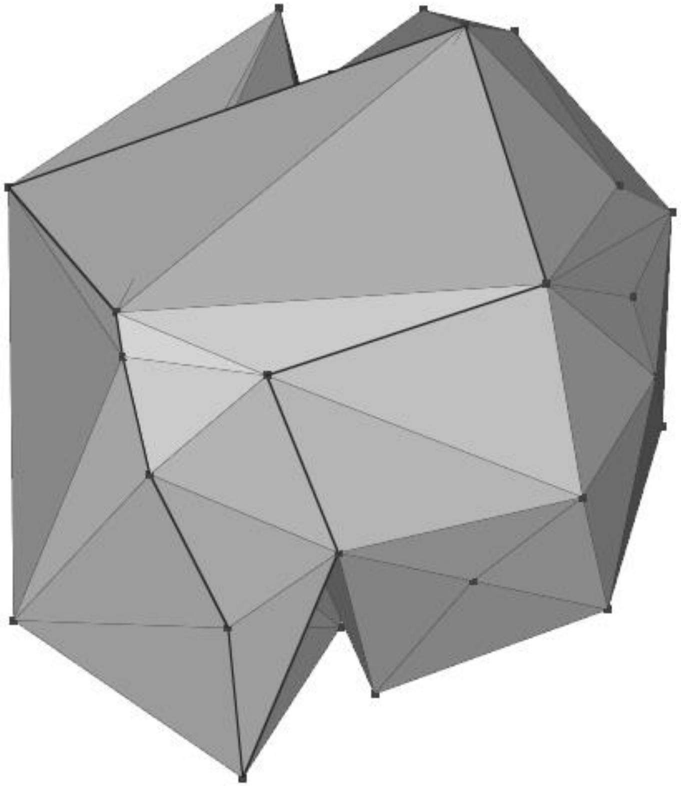
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

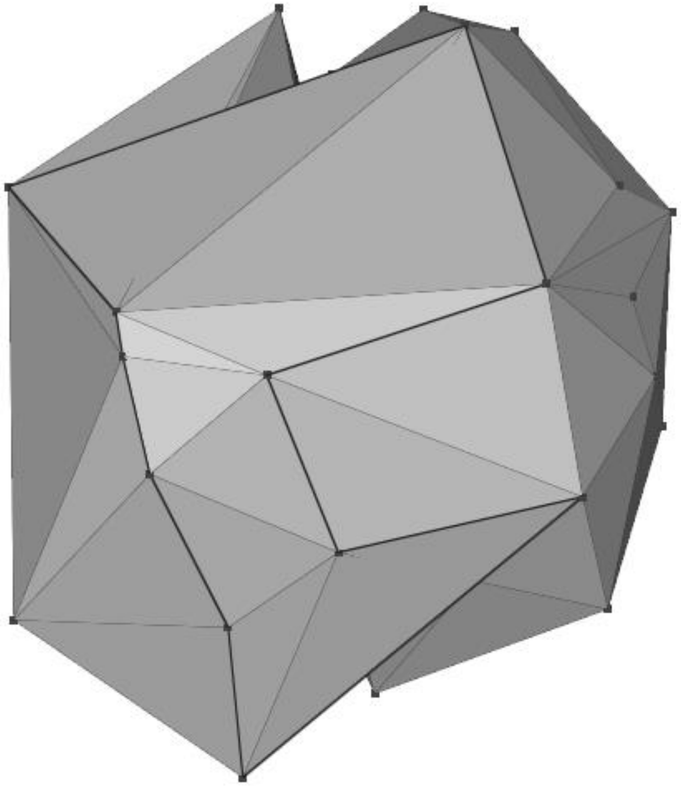
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

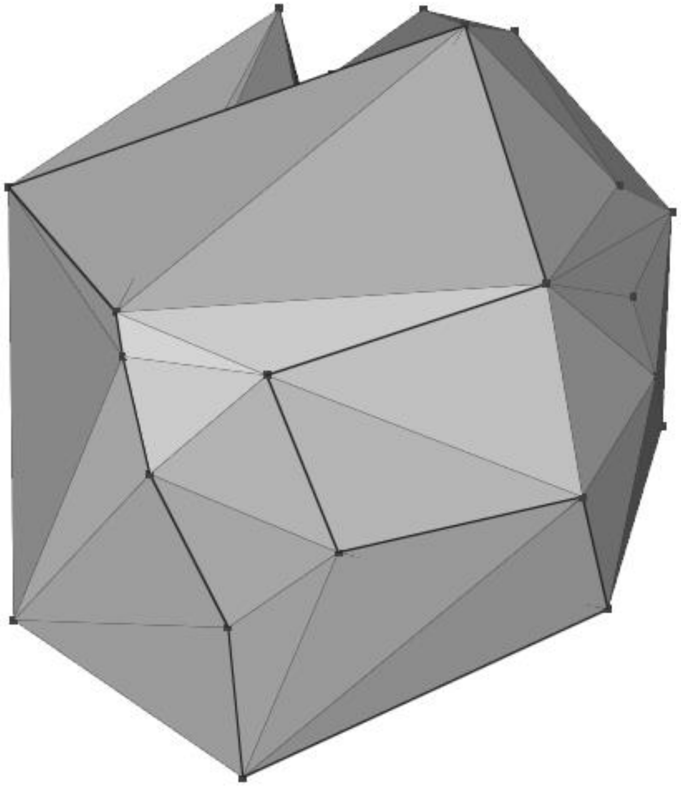
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

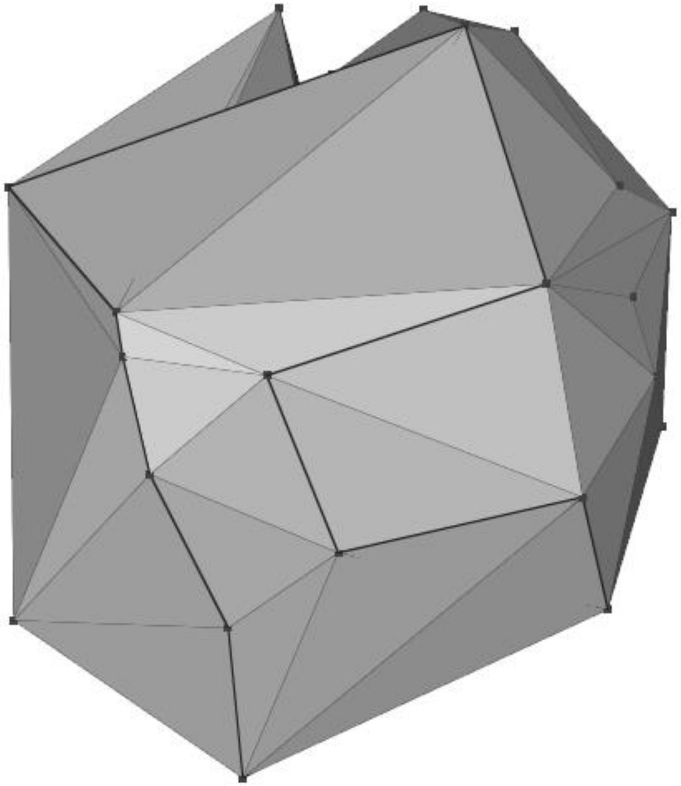
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

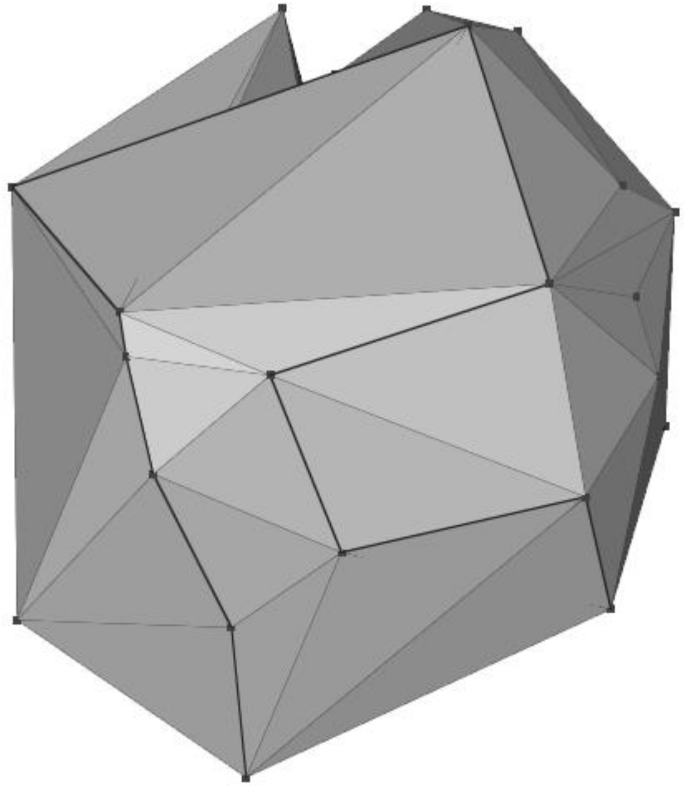
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

<u>Merge</u>:

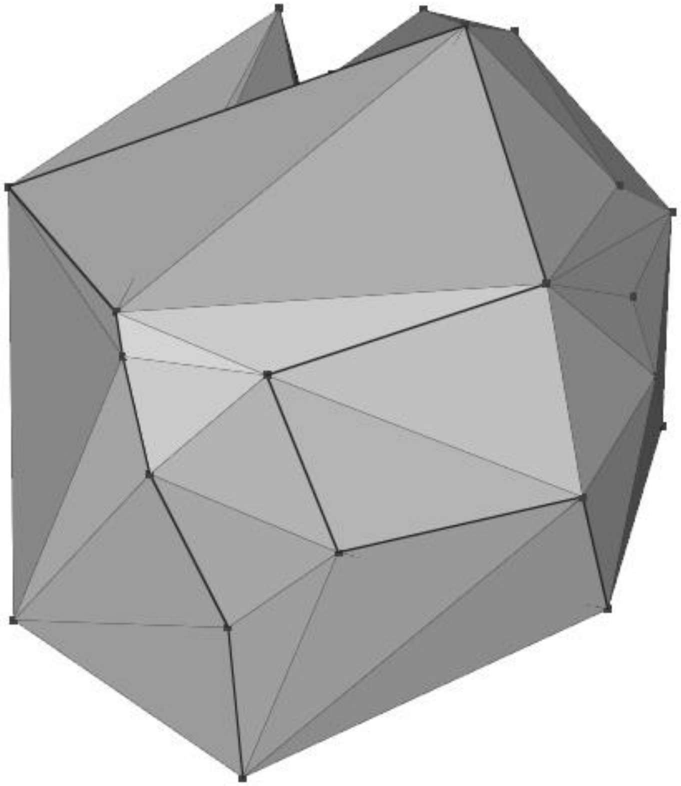
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

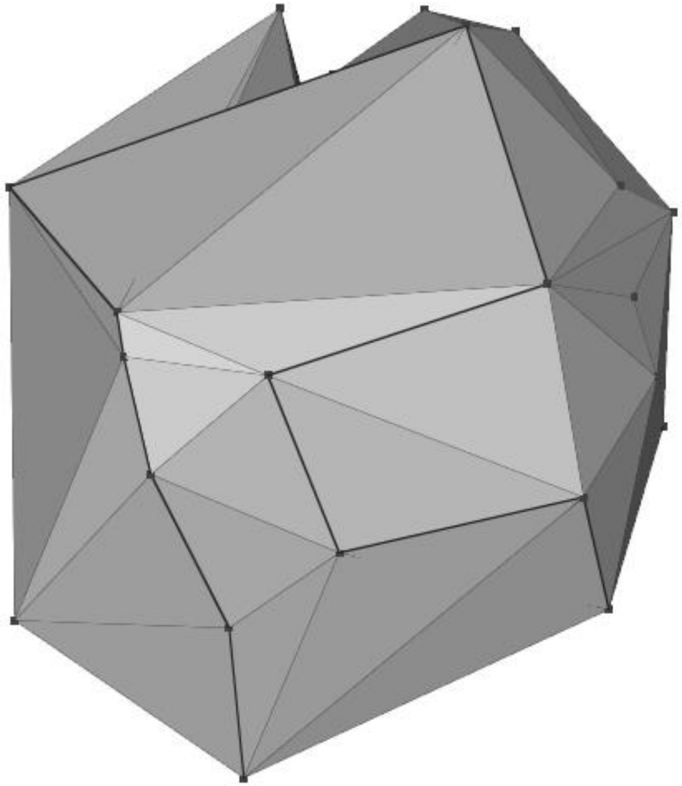
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

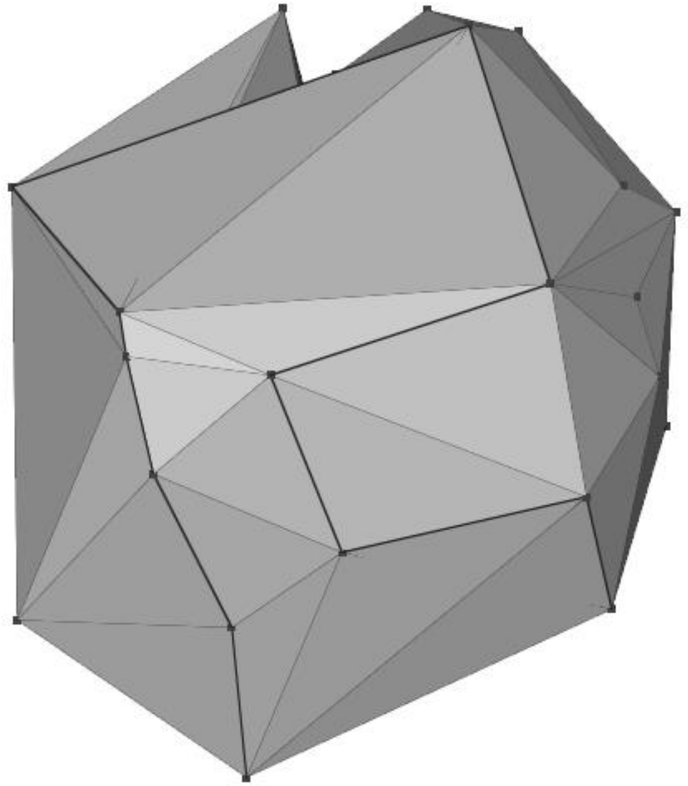
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

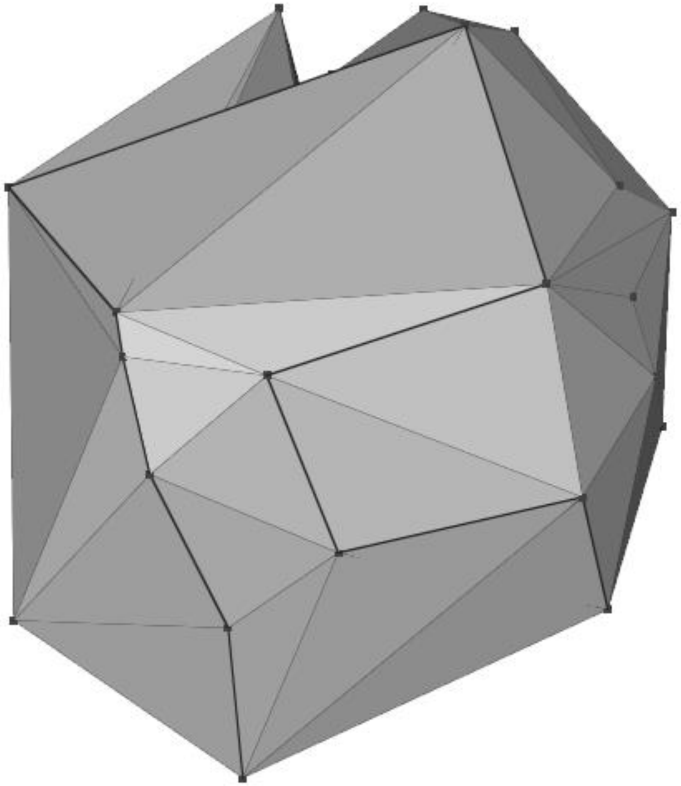
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

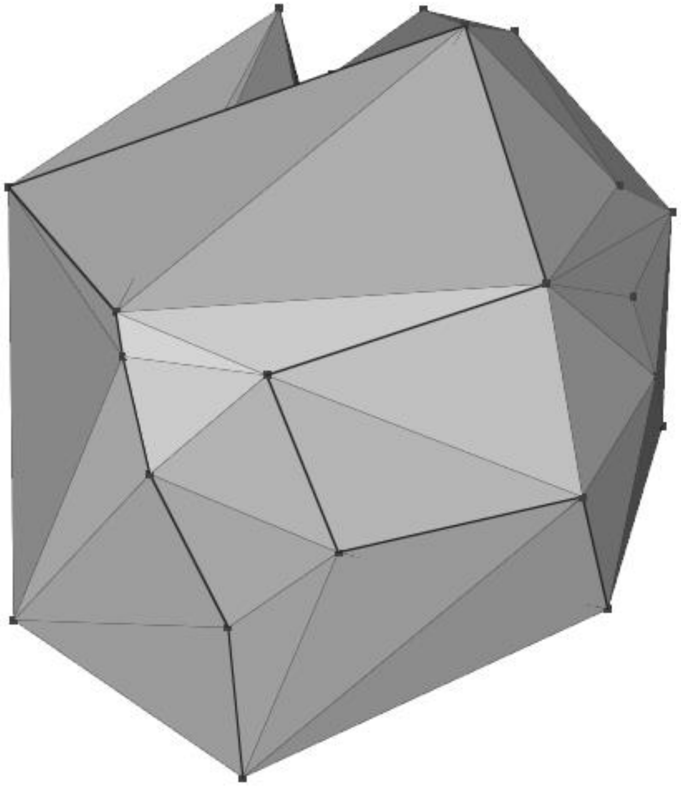
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

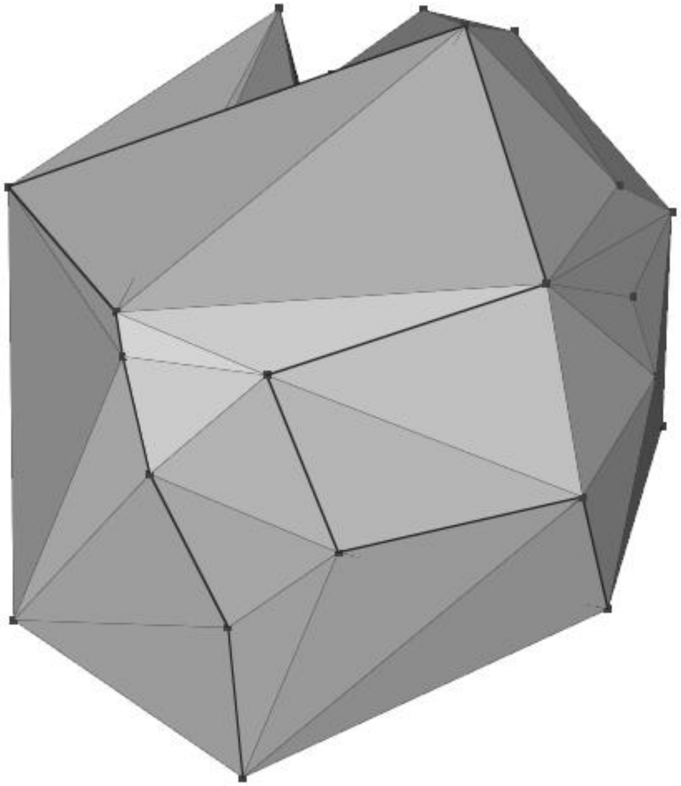
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

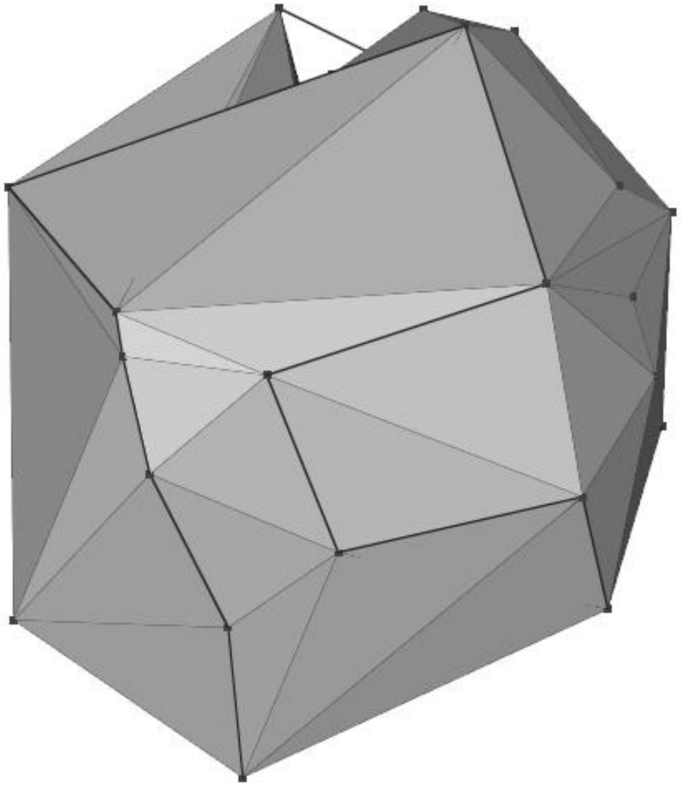
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

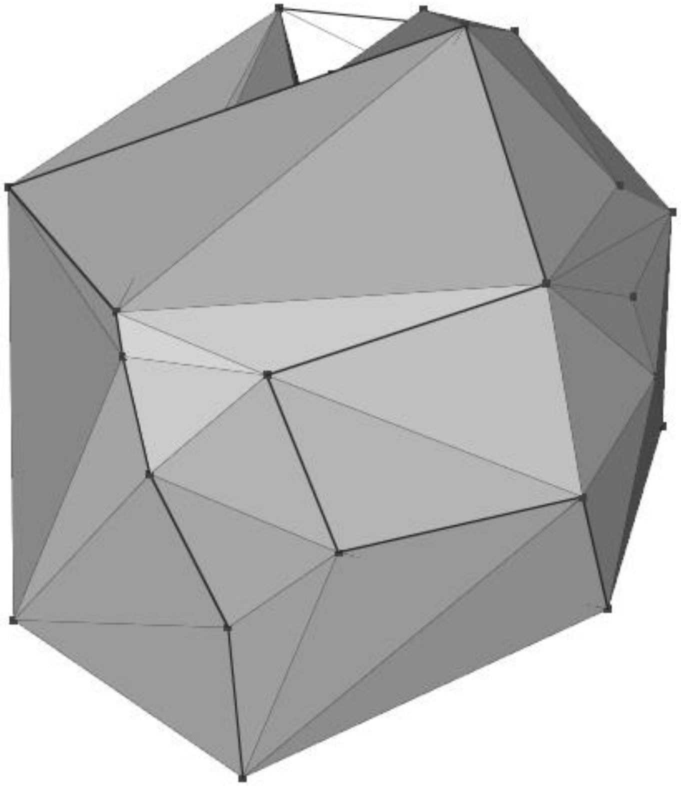
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

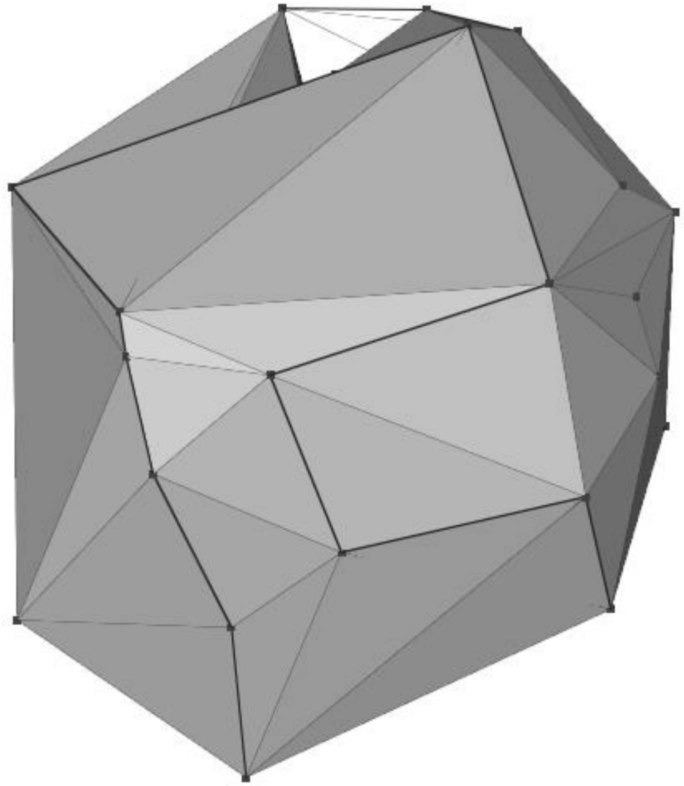
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

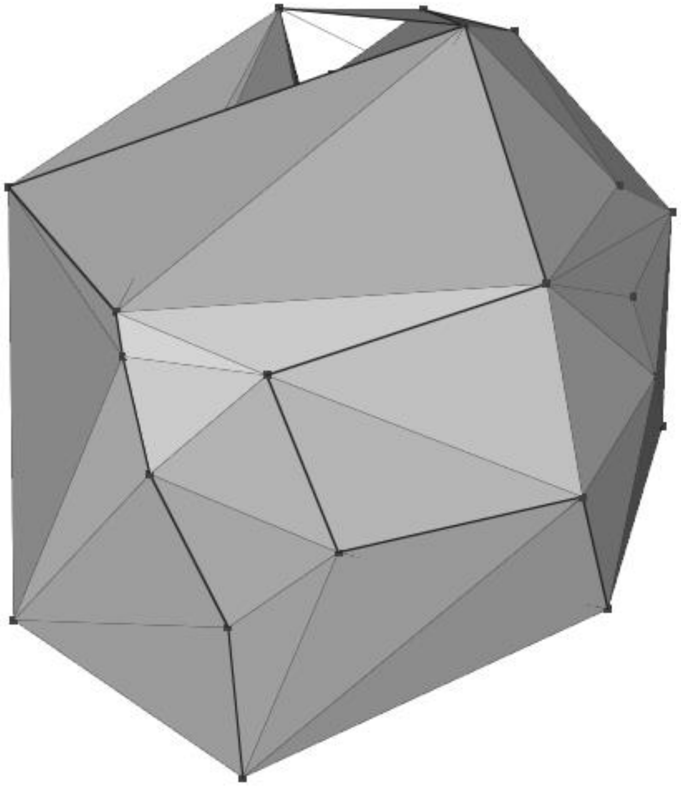
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

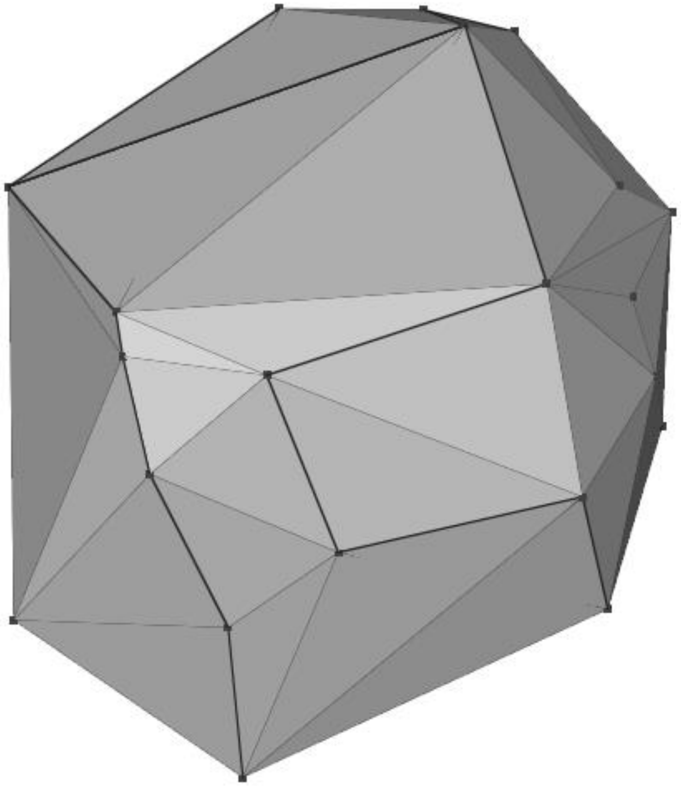
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

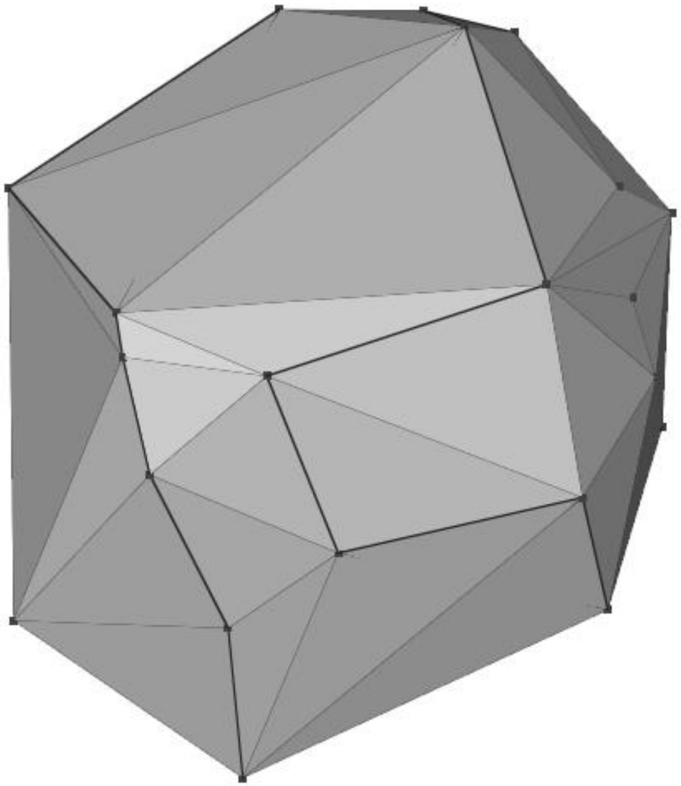
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

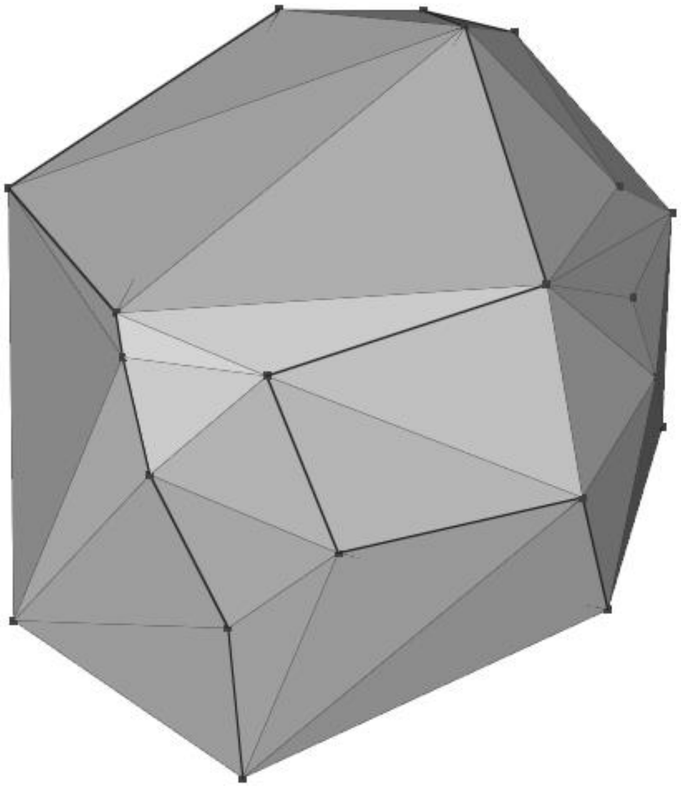
- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



Divide And Conquer

Merge:

- Construct the fillet that merges the two hulls
- Remove the triangles that are no longer visible



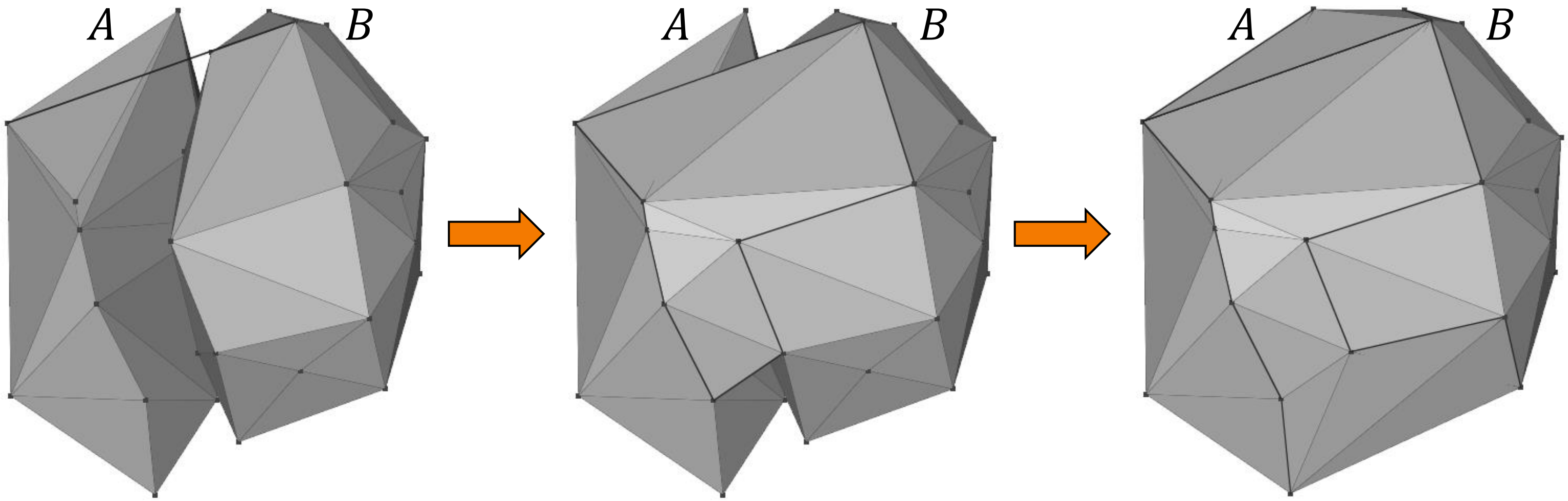
Note:

The fillet has linear complexity since each triangle on the fillet uses an edge from one of the two hulls.

Divide And Conquer

Constructing the Fillet:

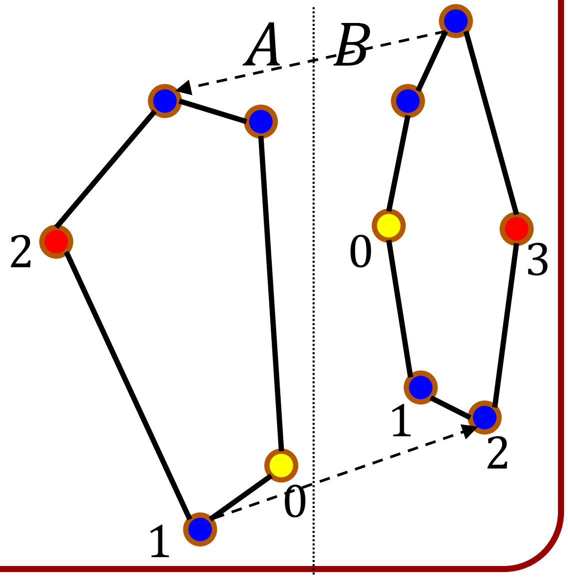
- Find a supporting line
- Pivot around the supporting line



Divide And Conquer

Finding a Supporting Line:

- While computing the 3D hull (recursively), simultaneously compute the 2D hull of the projection of the points onto the xy-plane.
- The supporting lines in 2D correspond to supporting lines in 3D.



Divide And Conquer

Pivot Around the Supporting Line:

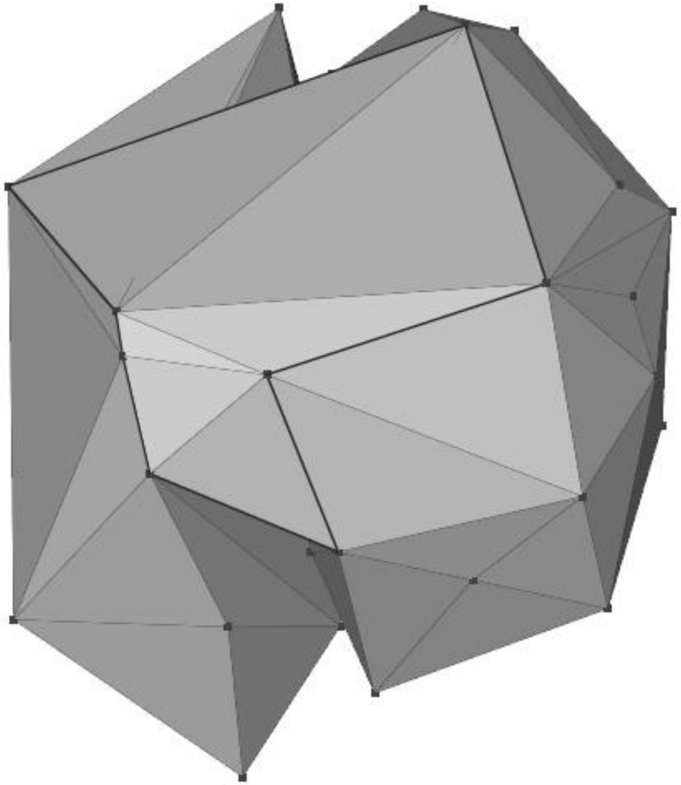
- Proceed as in the gift-wrap algorithm.

Challenge:

- To run in linear time, we can't try all points.

Observation:

- When we pivot, the first point we hit is one of the neighbors of the line's end-points.



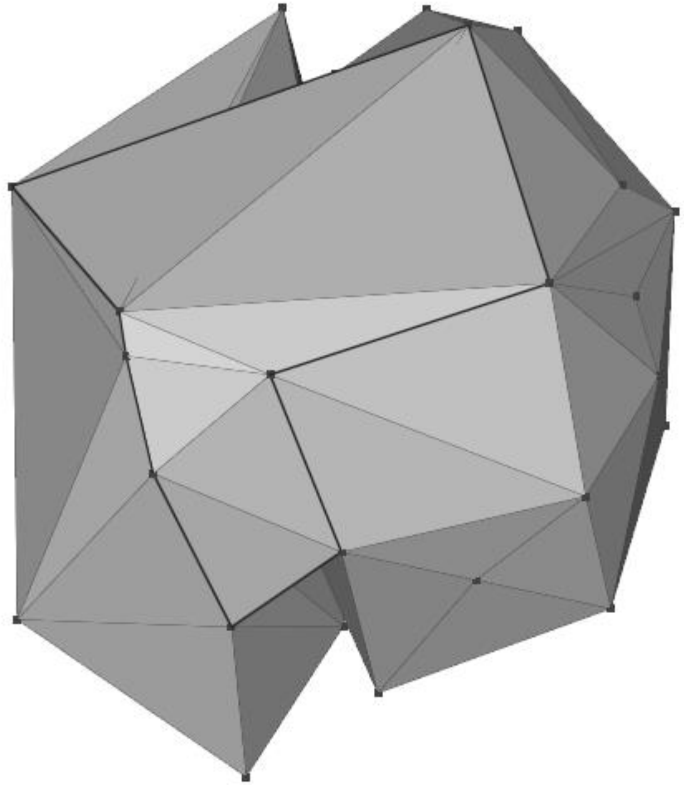
Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:

- To run in linear time, we can't try all points.



Observation:

- When we pivot, the first point we hit is one of the neighbors of the line's end-points.

Divide And Conquer

Pivot Around the Supporting Line:

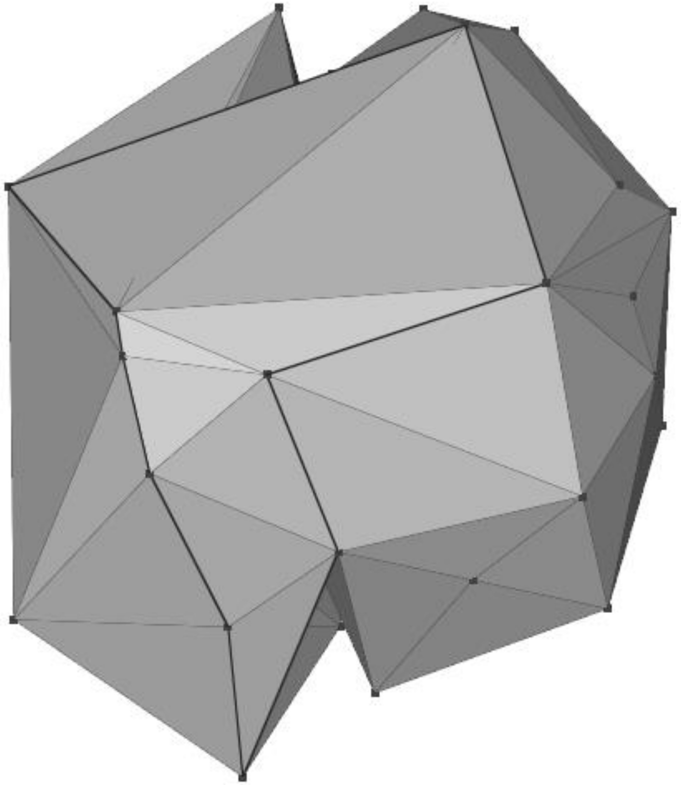
- Proceed as in the gift-wrap algorithm.

Challenge:

- To run in linear time, we can't try all points.

Observation:

- When we pivot, the first point we hit is one of the neighbors of the line's end-points.



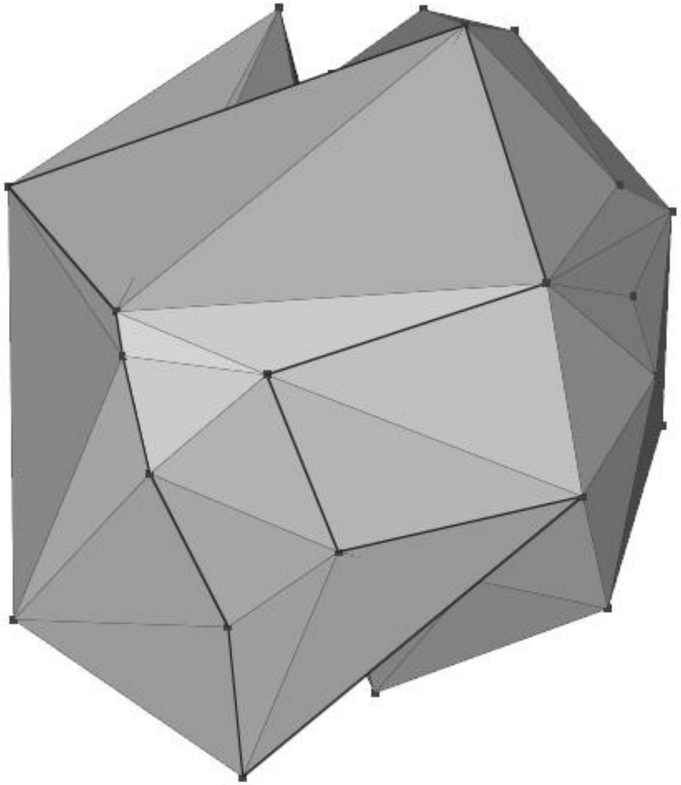
Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:

- To run in linear time, we can't try all points.



Observation:

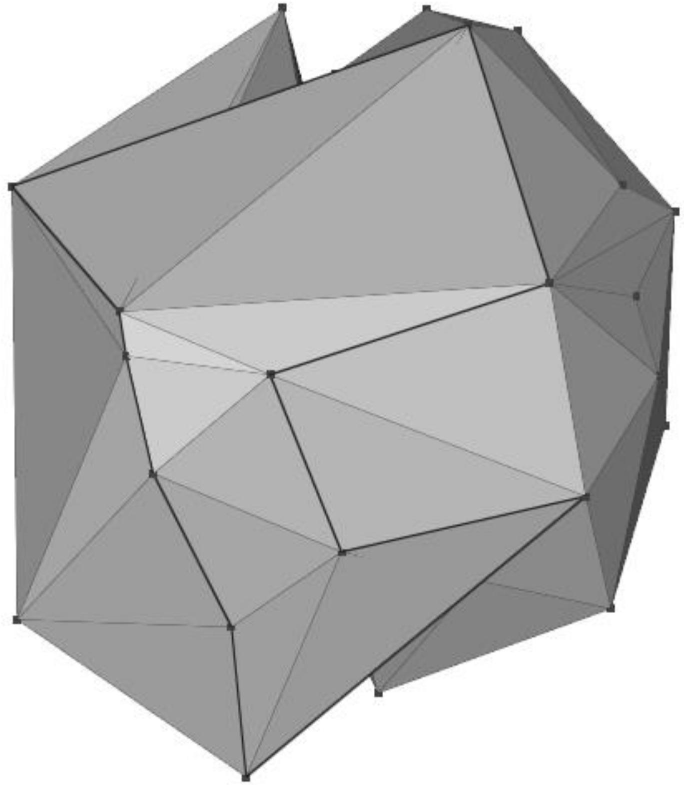
- When we pivot, the first point we hit is one of the neighbors of the line's end-points.

Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:



- This could still be costly since a vertex can have many neighbors.
  (e.g. If the right endpoint has many neighbors but the pivot keeps hitting a vertex on the left.)

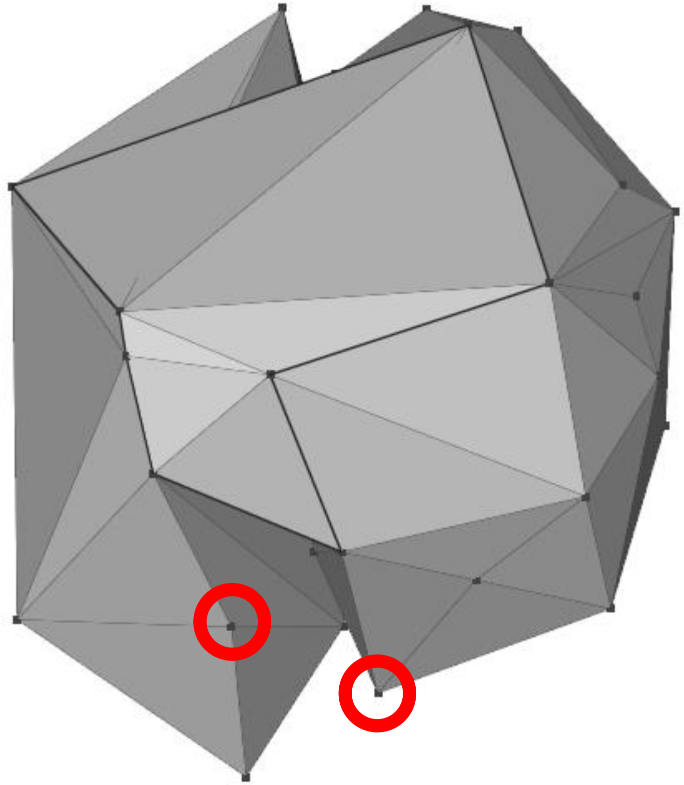
Divide And Conquer

<u>Pivot Around the Supporting Line</u>:

- Proceed as in the gift-wrap algorithm.

<u>Challenge</u>:

- This could still be costly since a vertex can have many neighbors.



<u>Observation</u>:

- We can use the previous estimated (failed) hit to constrain the next one.

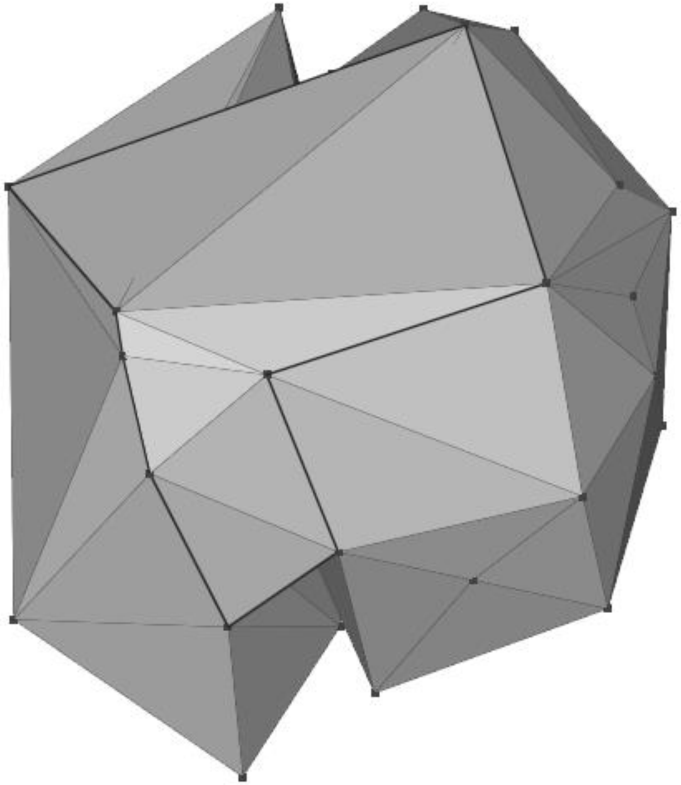
Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:

- This could still be costly since a vertex can have many neighbors.



Observation:

- We can use the previous estimated (failed) hit to constrain the next one.

Divide And Conquer

Pivot Around the Supporting Line:

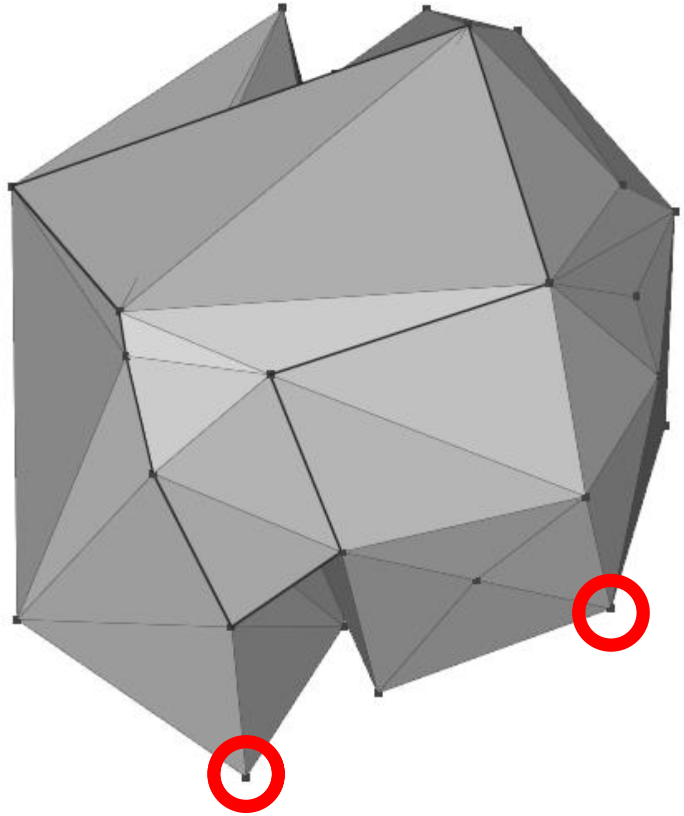
- Proceed as in the gift-wrap algorithm.

Challenge:

- This could still be costly since a vertex can have many neighbors.

Observation:

- We can use the previous estimated (failed) hit to constrain the next one.



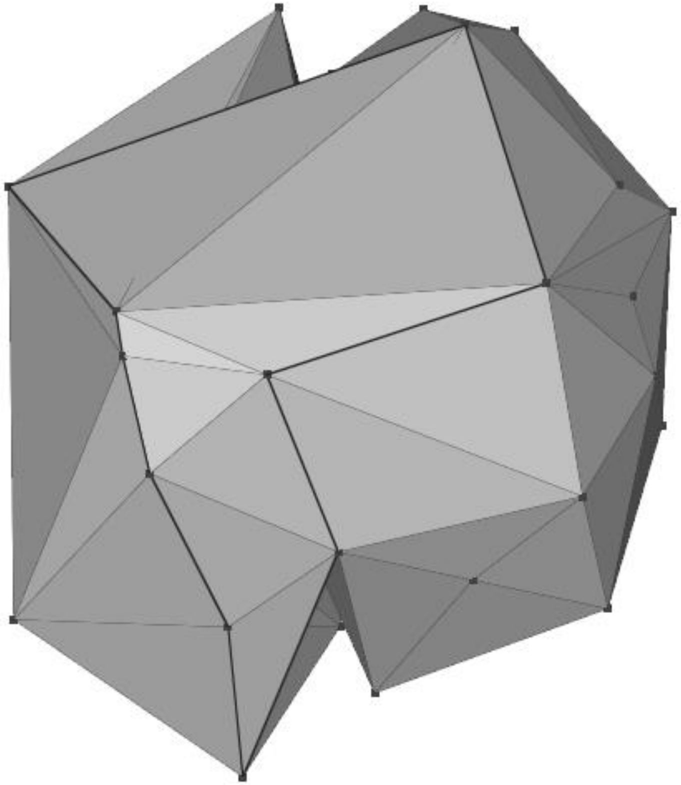
Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:

- This could still be costly since a vertex can have many neighbors.



Observation:

- We can use the previous estimated (failed) hit to constrain the next one.

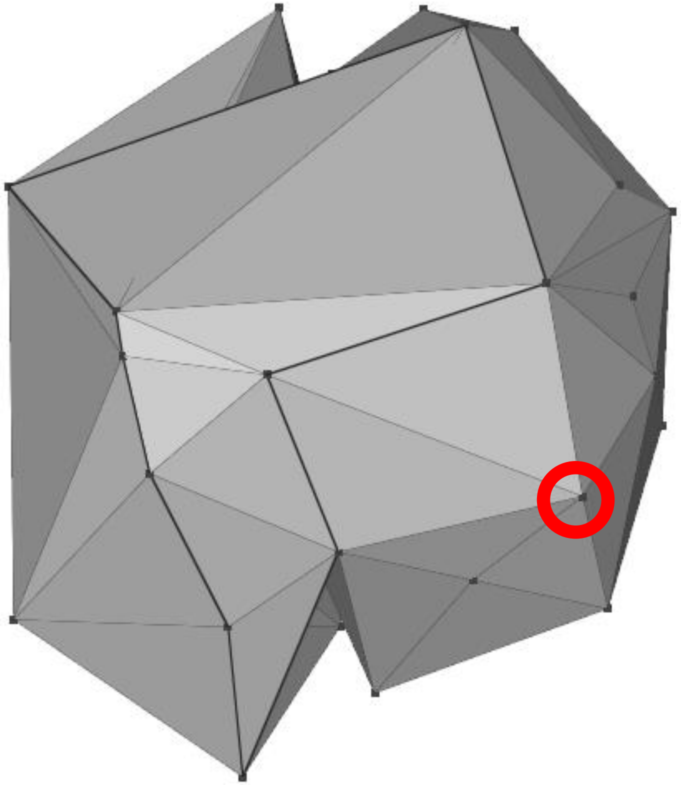
Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.

Challenge:

- This could still be costly since a vertex can have many neighbors.



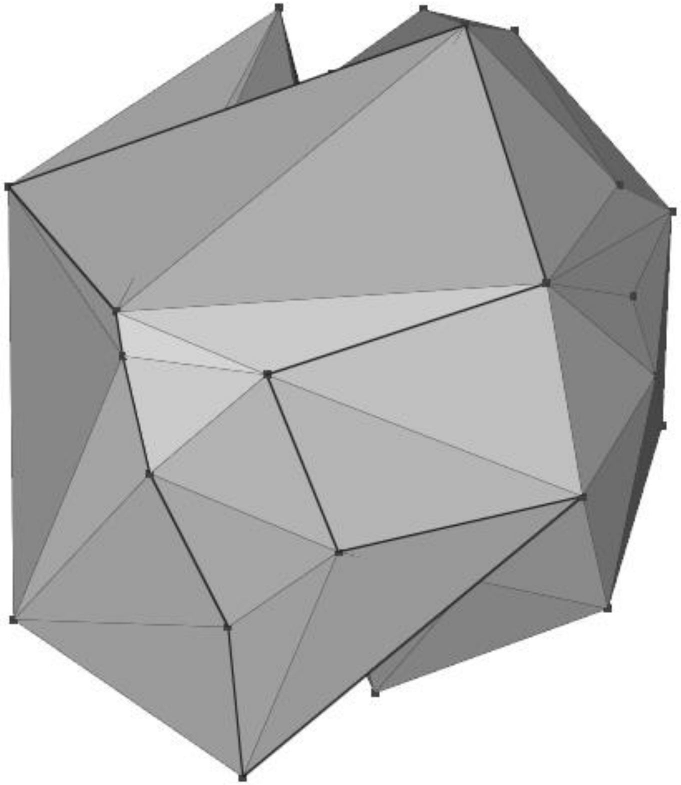
Observation:

- We can use the previous estimated (failed) hit to constrain the next one.

Divide And Conquer

Pivot Around the Supporting Line:

- Proceed as in the gift-wrap algorithm.



Challenge:

- This could still be costly since a vertex can have many neighbors.

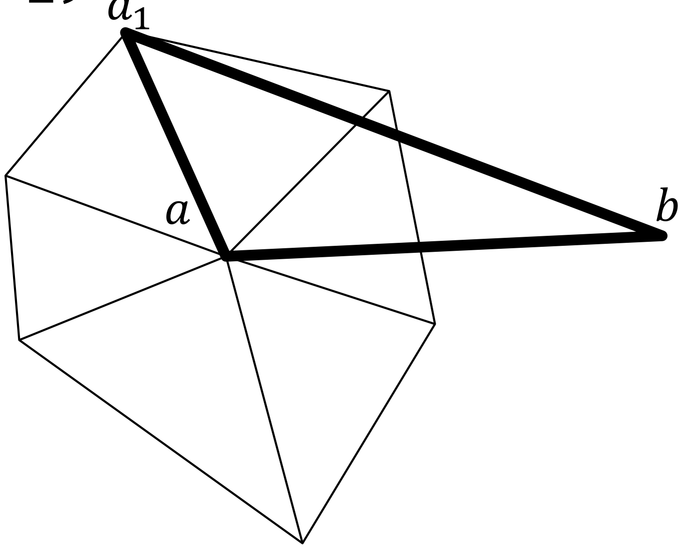
Observation:

- We can use the previous estimated (failed) hit to constrain the next one.

Divide And Conquer

More Specifically:

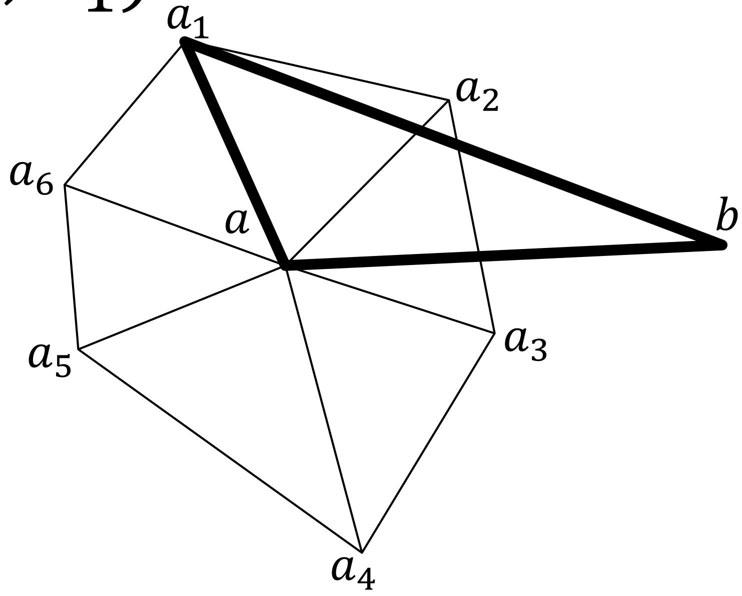
- Assume the fillet is at edge (a, b) having just added triangle (a, b, a_1).



Divide And Conquer

More Specifically:

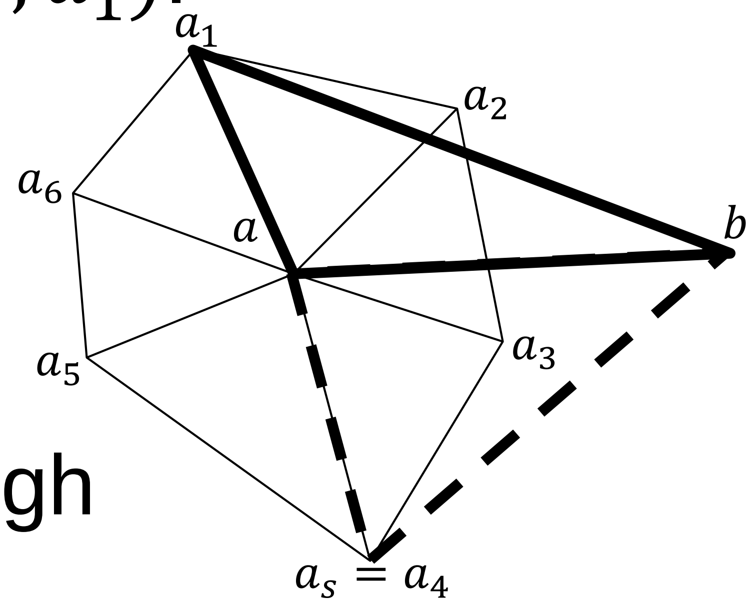
- Assume the fillet is at edge (a, b) having just added triangle (a, b, a_1).
- Sort the neighbors of a CW starting from a_1.



Divide And Conquer

<u>More Specifically</u>:

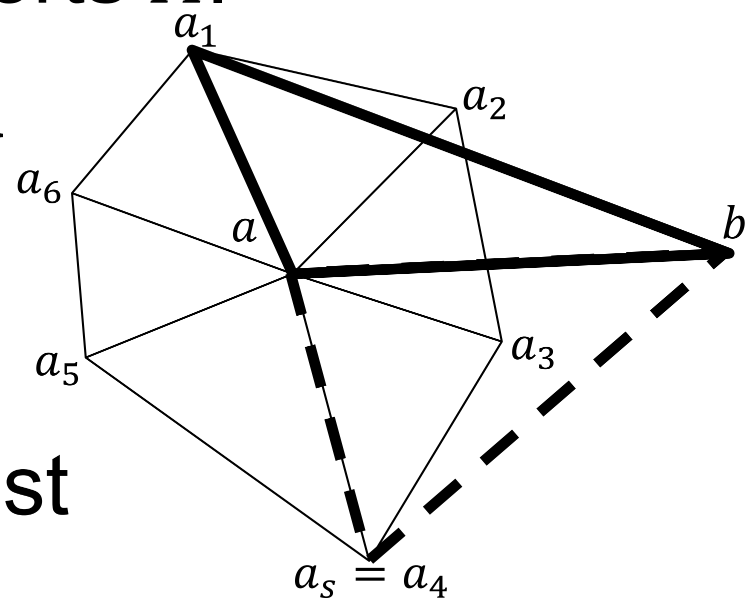
- Assume the fillet is at edge (a, b) having just added triangle (a, b, a_1).
- Sort the neighbors of a CW starting from a_1.
- Let a_s be the neighbor of a s.t. the plane through (b, a, a_s) supports A.



Divide And Conquer

More Specifically:

- Let a_s be the neighbor s.t. the plane through (b, a, a_s) supports A.
- The points $\{a_2, \ldots, a_{s-1}\}$ must be inside the hull.
- Even if we advance on b we won't need to retest these points.



Divide And Conquer

Merge(H_1 , H_2):

$(v_1, v_2) \leftarrow$ FindSupportingLine(H_1 , H_2)

$Q \leftarrow \{(v_1, v_2)\}$

$F \leftarrow \emptyset$

While ($Q \neq \emptyset$)

  $e \leftarrow Q$.pop_back()

  if($e \neq \{v_2, v_1\}$)

    $t \leftarrow$ SupportingTriangle(H_1 , H_2 , e)

    $F \leftarrow F \cup \{t\}$

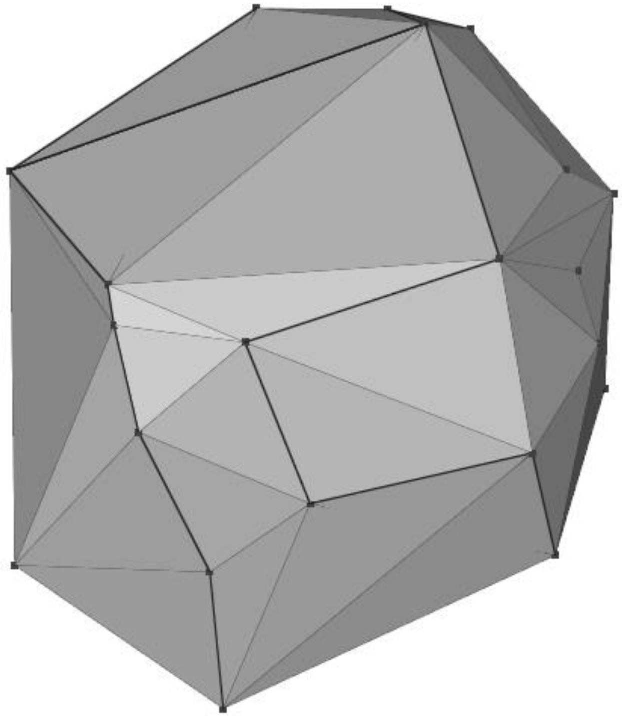
    $Q \leftarrow Q \cup$ CrossingEdges(t) / $\{e\}$

CleanUp

Divide And Conquer

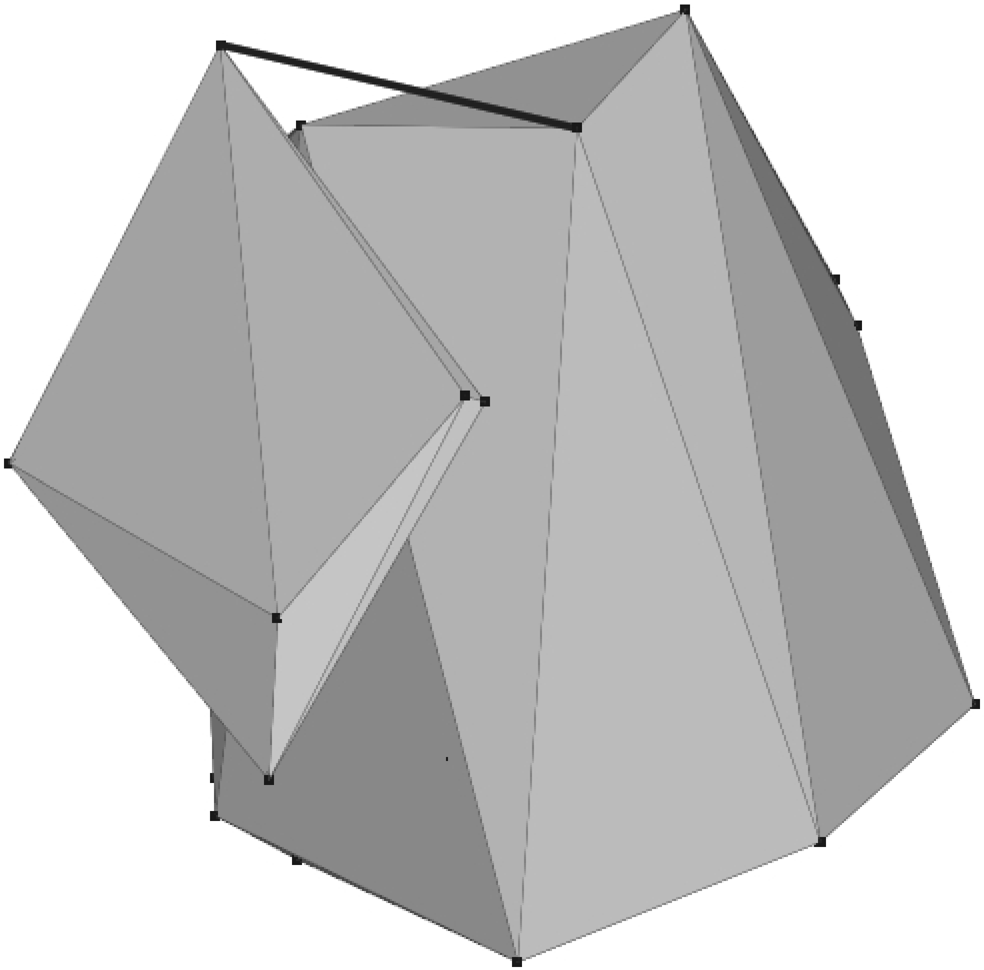
Clean-Up:

- Represent the two hulls with a winged-edge data structure.
- Replace the opposite edges of the silhouette with the edges of the new triangles.

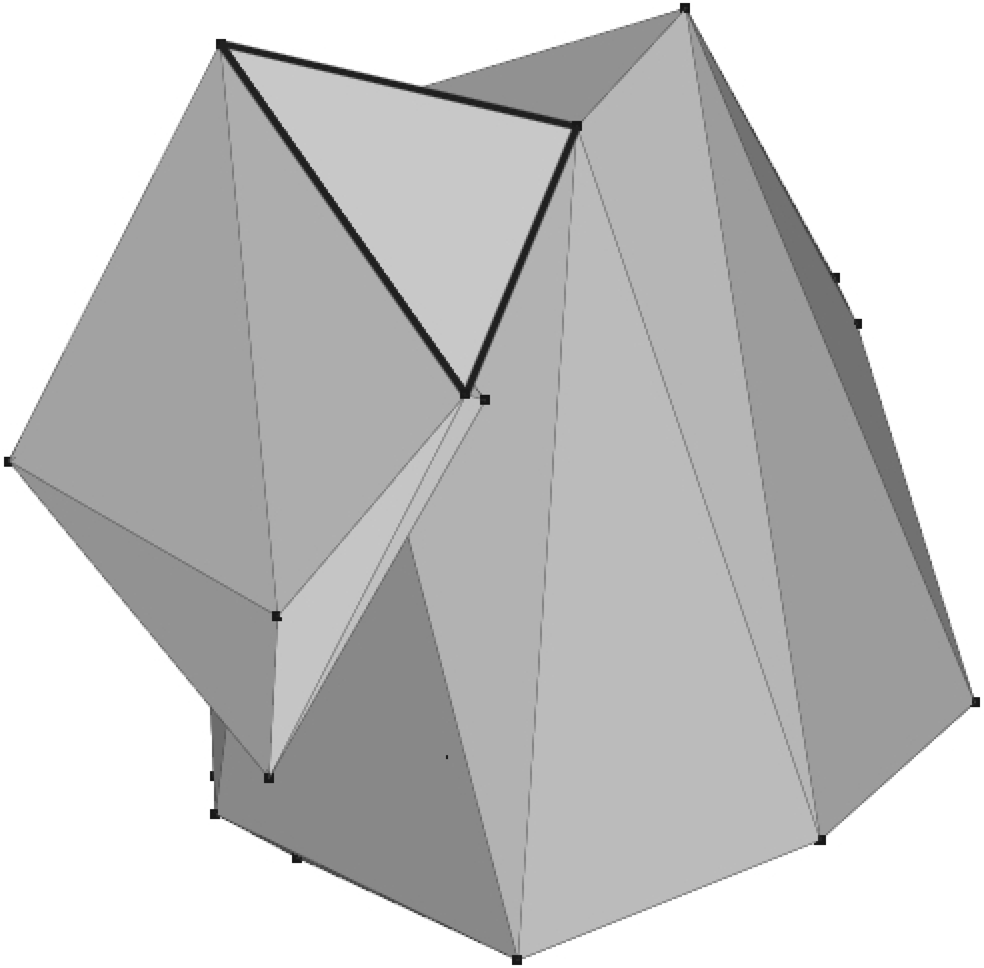


- Flood-fill to find interior triangles.

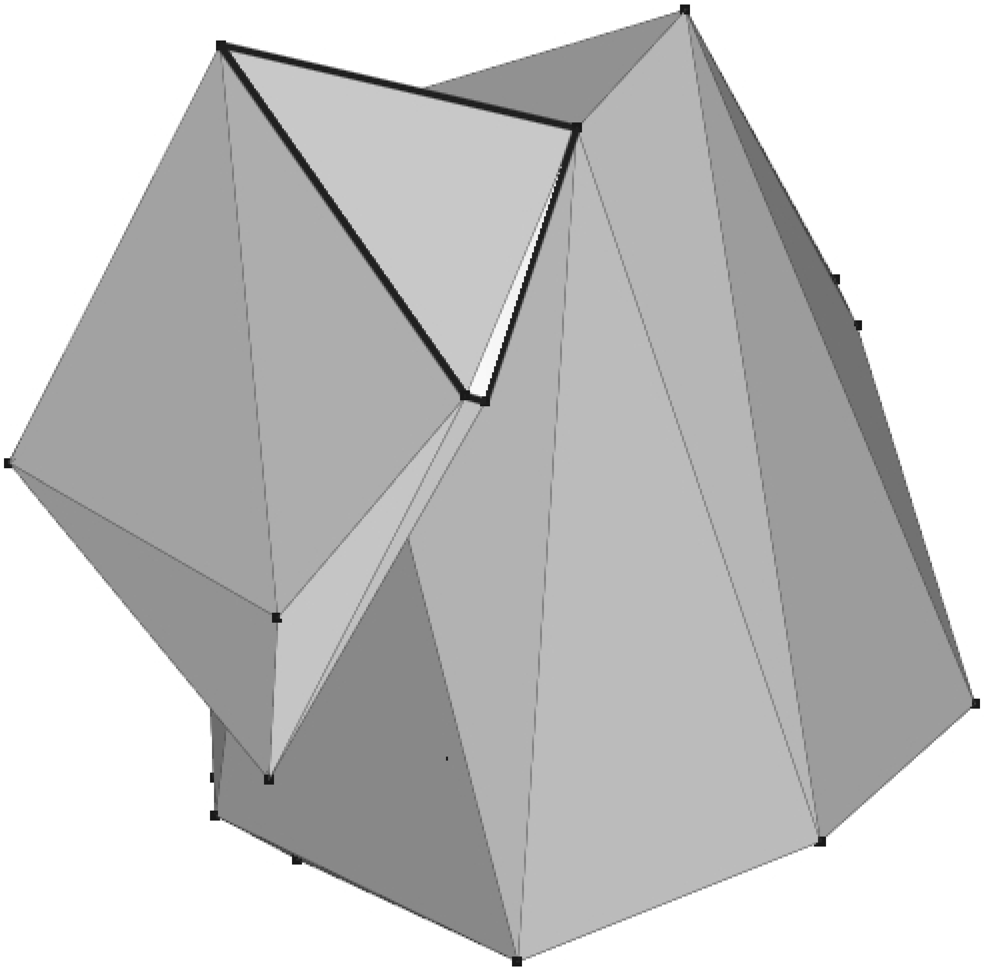
Divide And Conquer



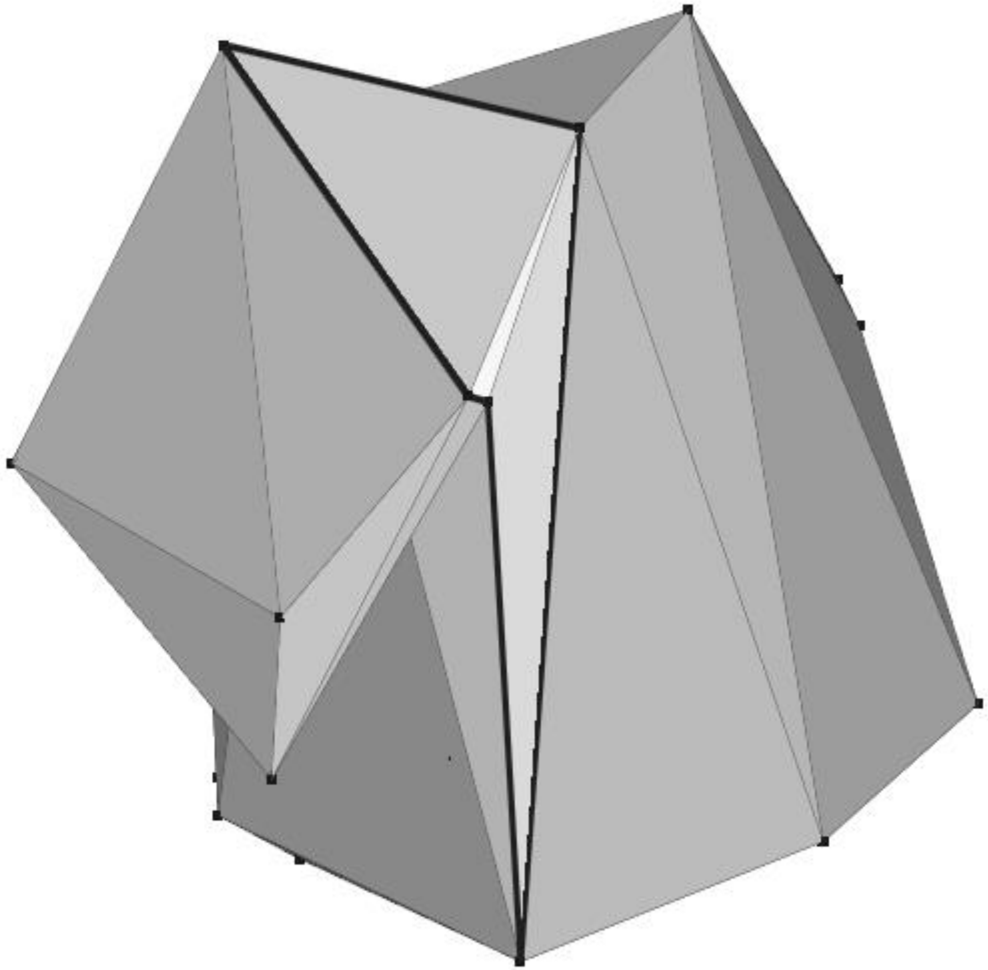
Divide And Conquer



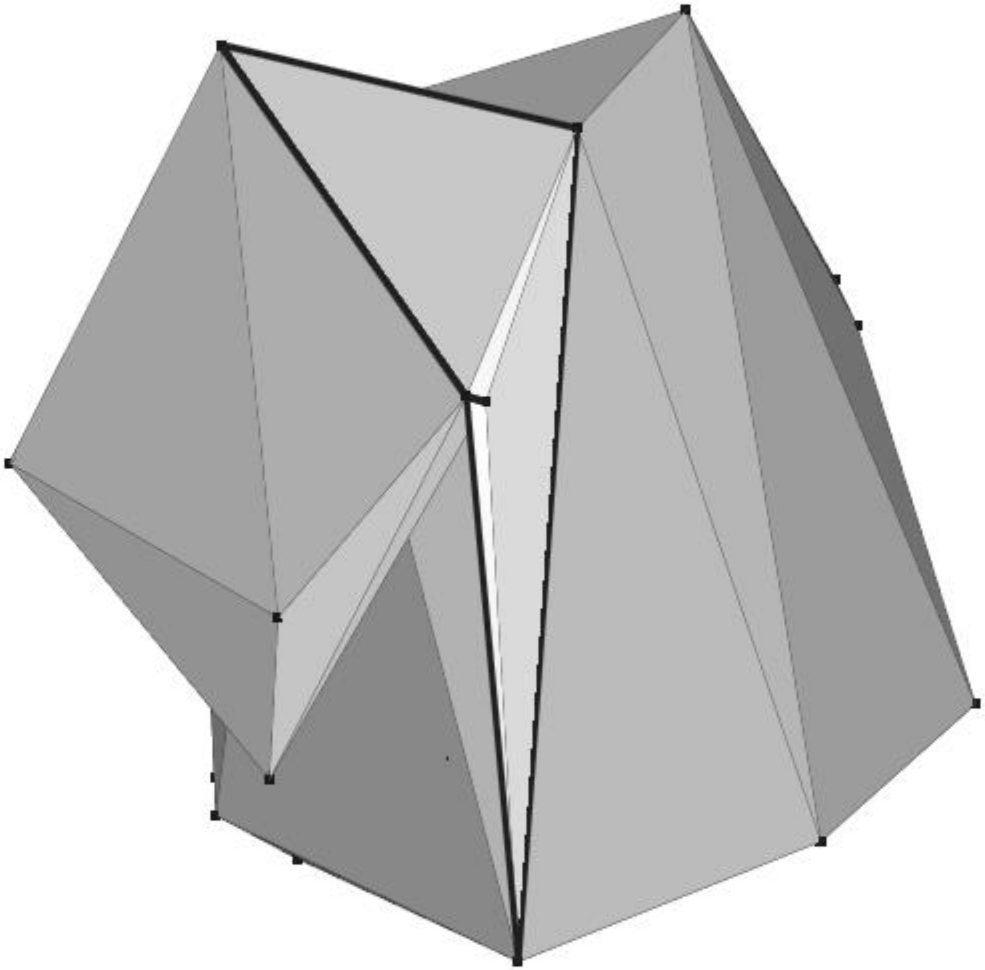
Divide And Conquer



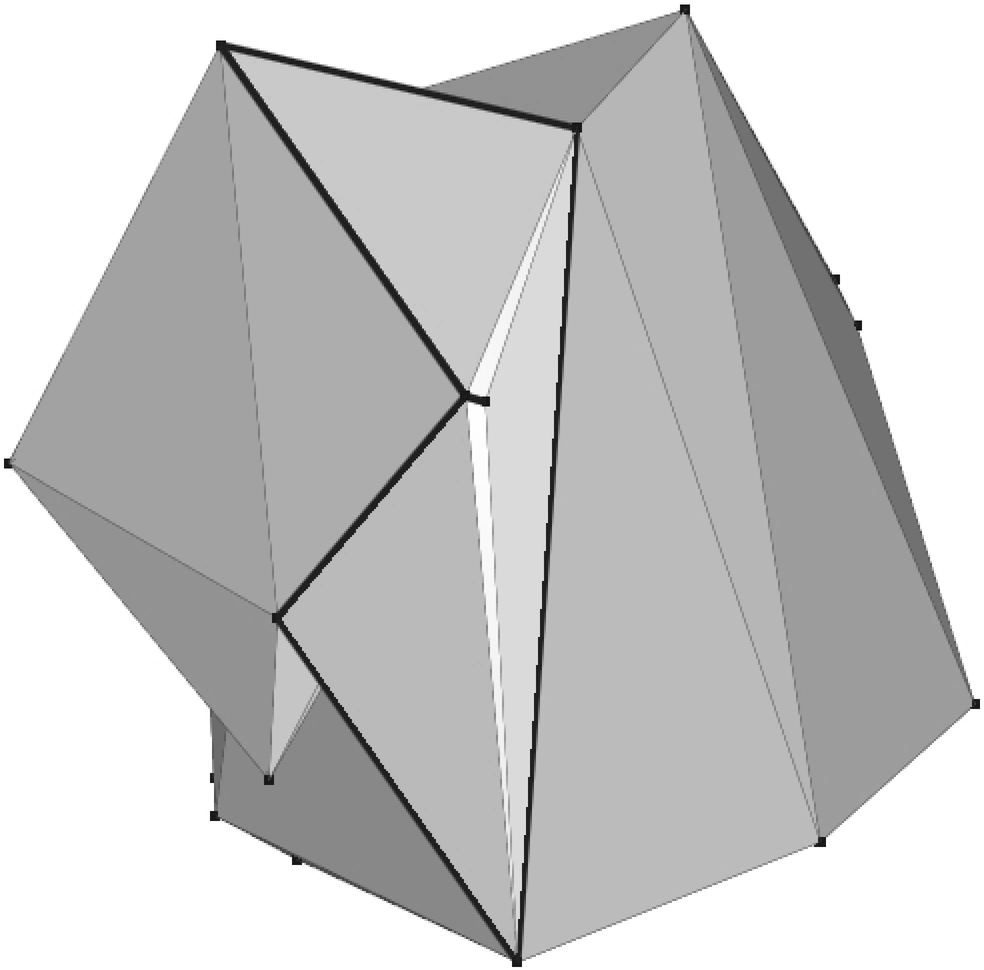
Divide And Conquer



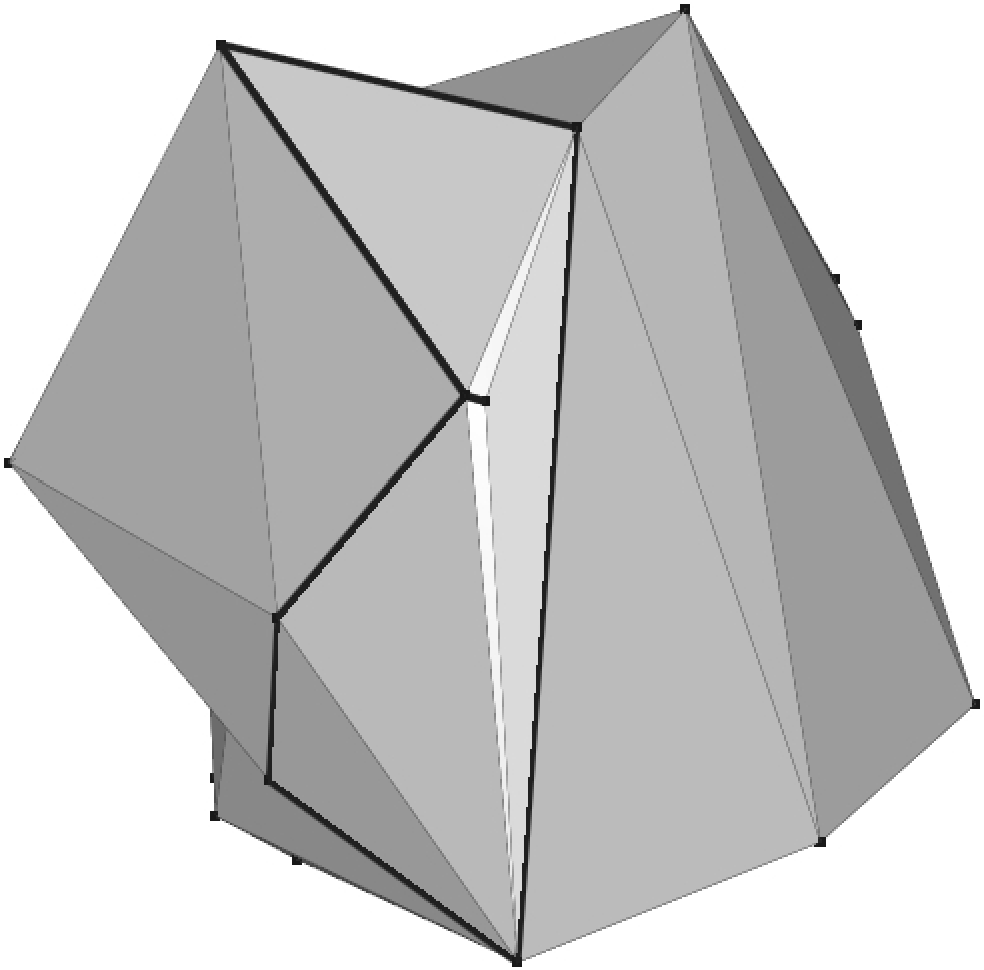
Divide And Conquer



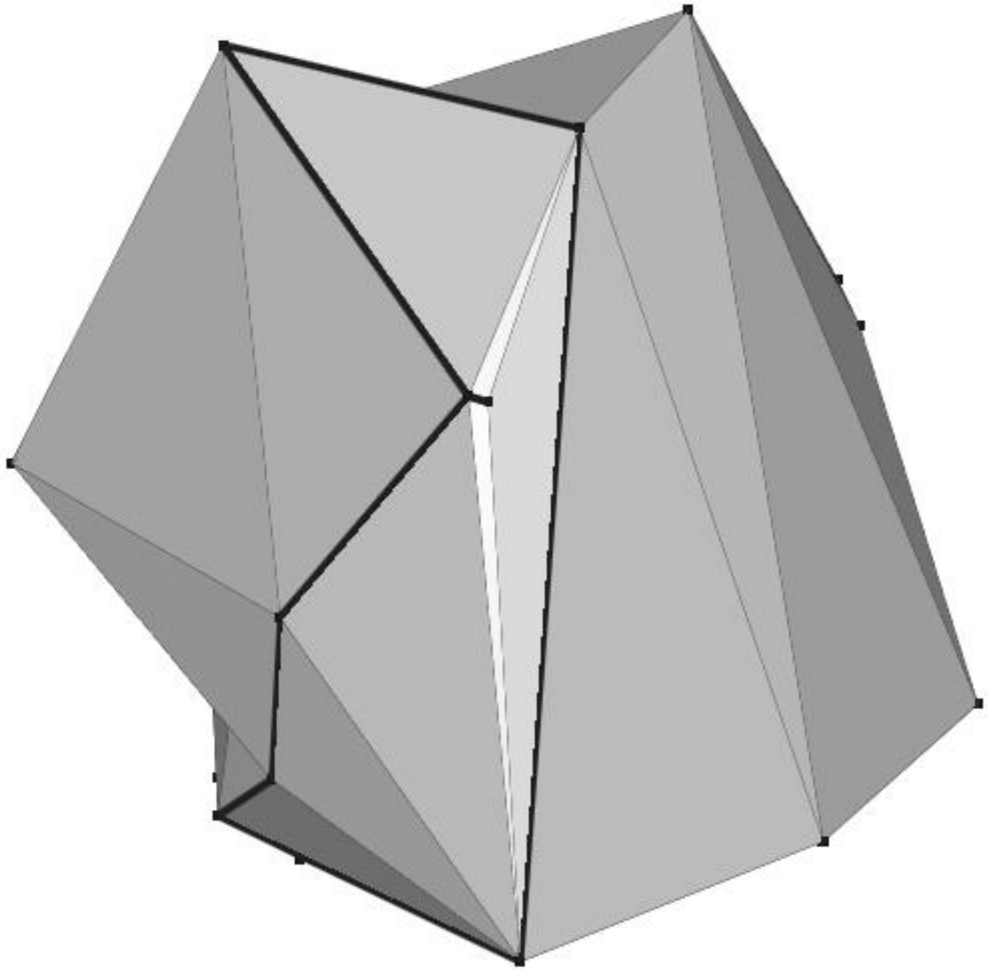
Divide And Conquer



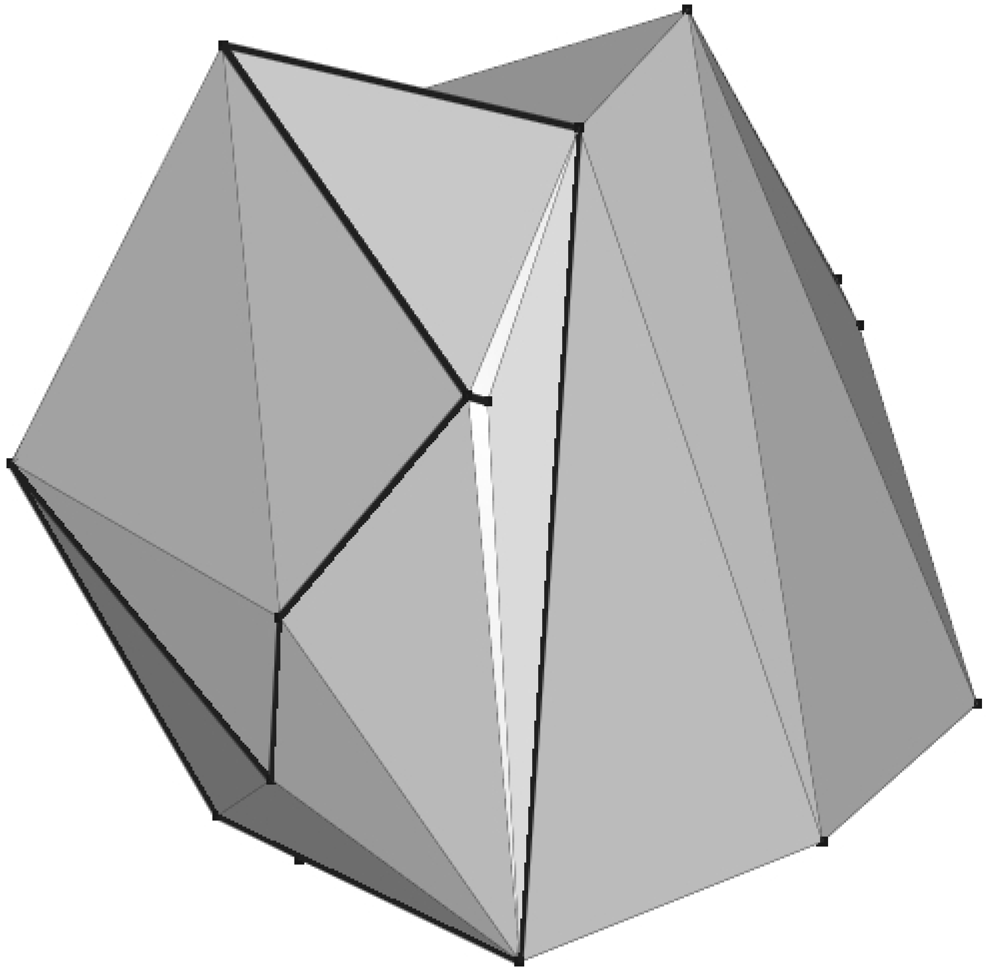
Divide And Conquer



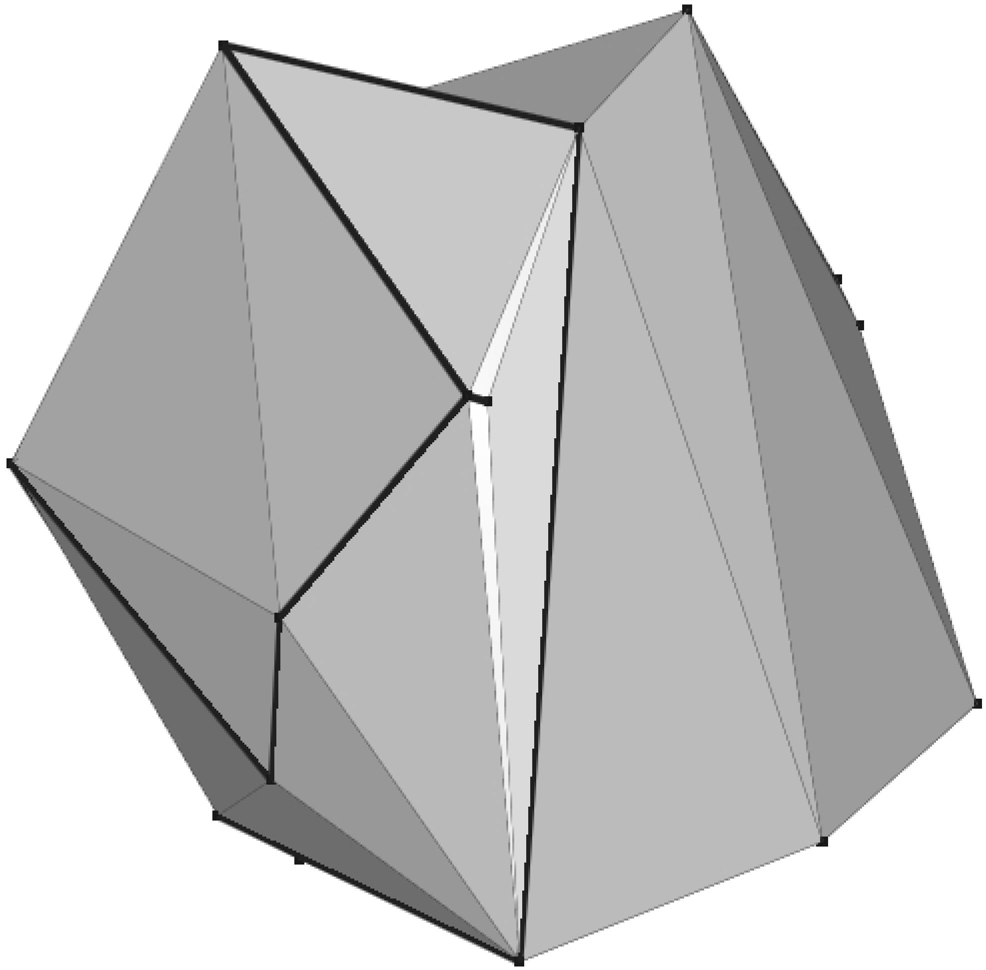
Divide And Conquer



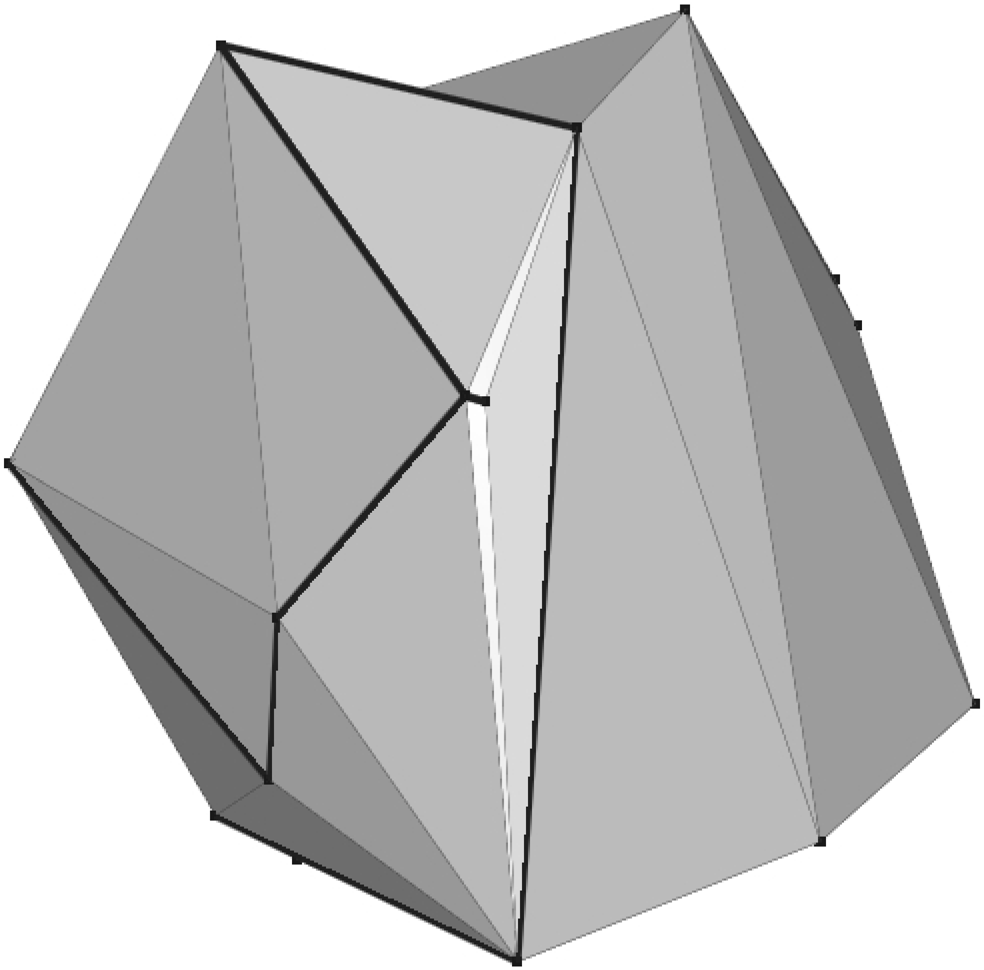
Divide And Conquer



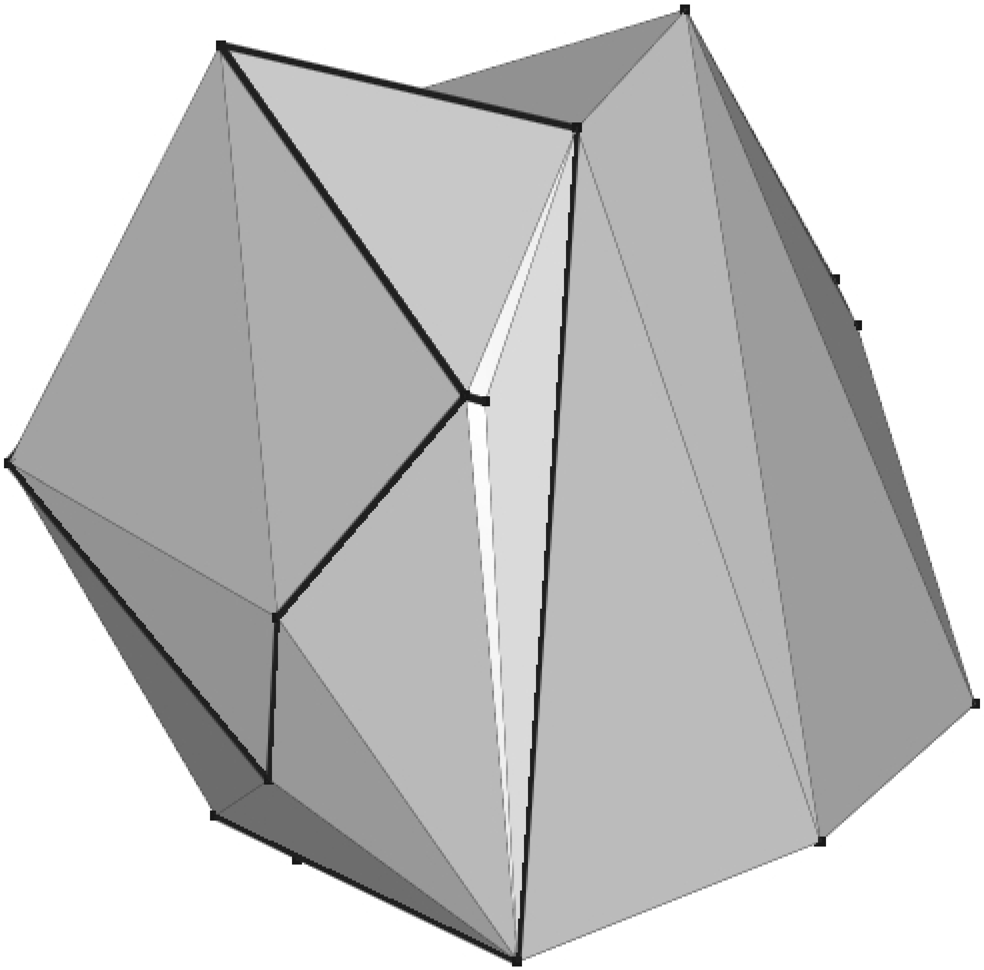
Divide And Conquer



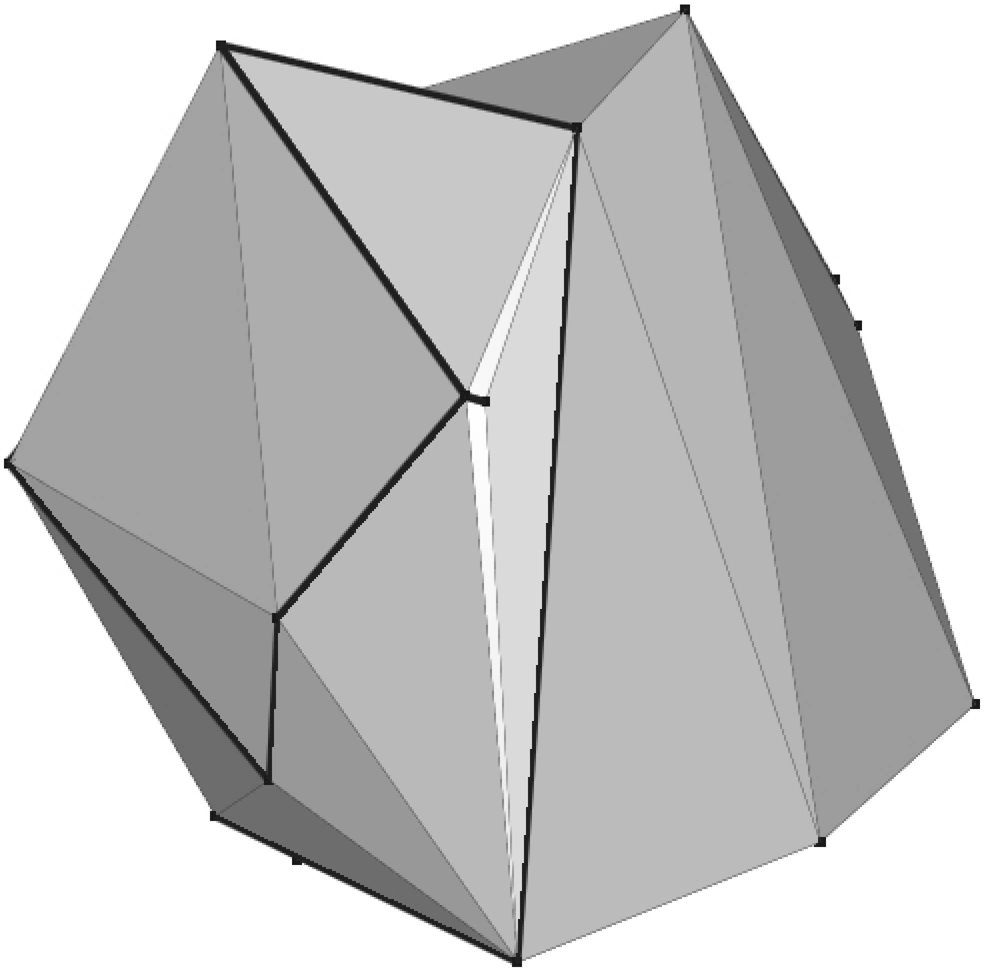
Divide And Conquer



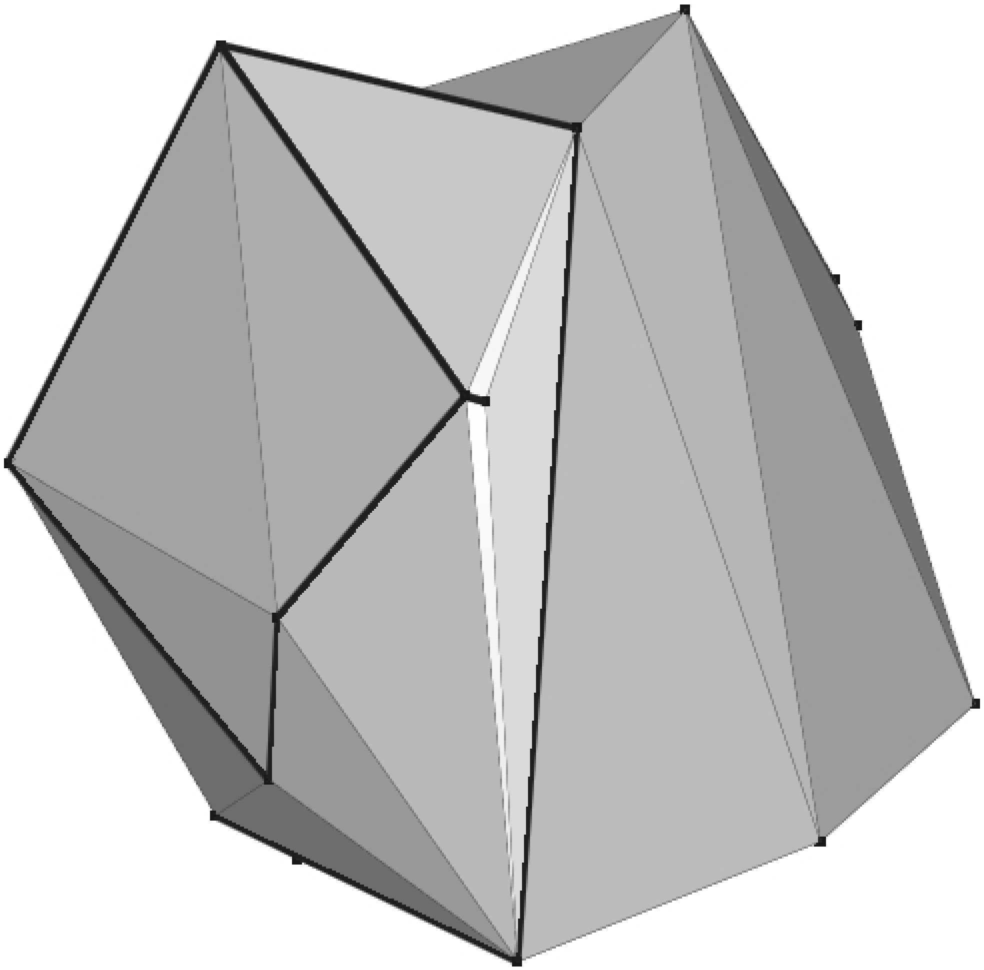
Divide And Conquer



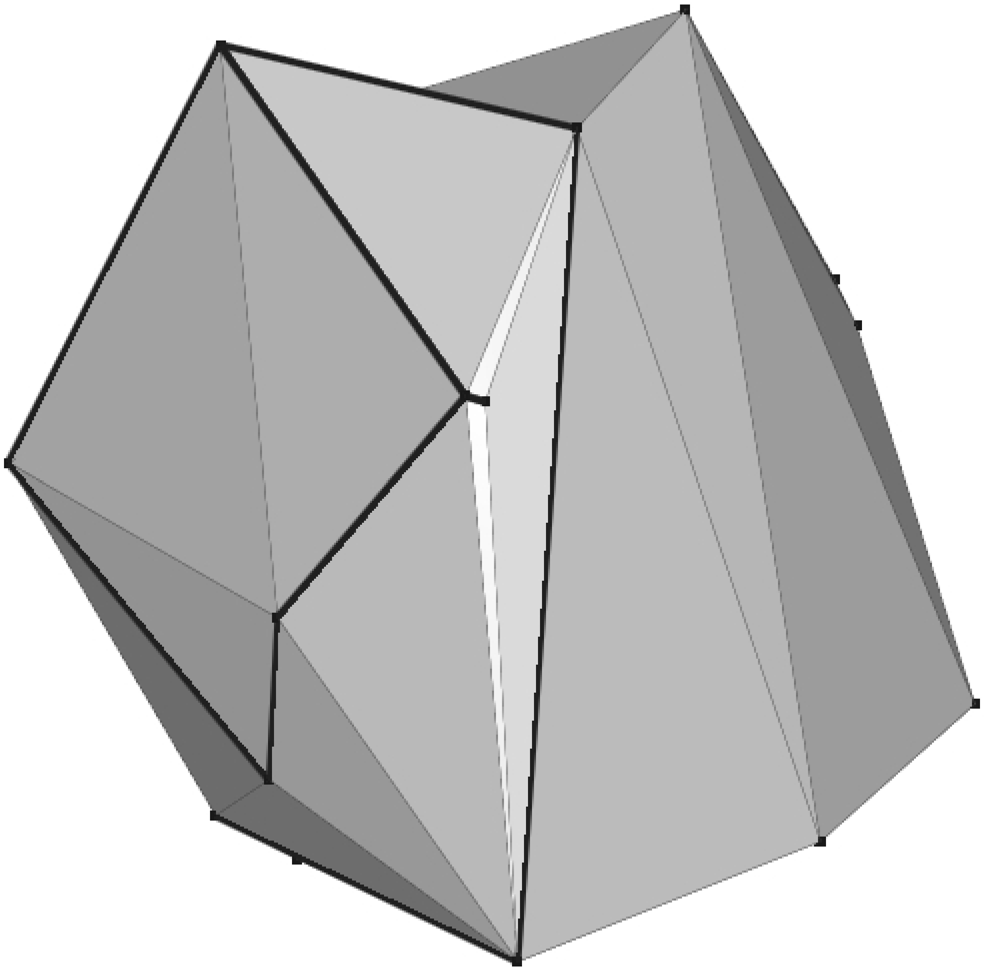
Divide And Conquer



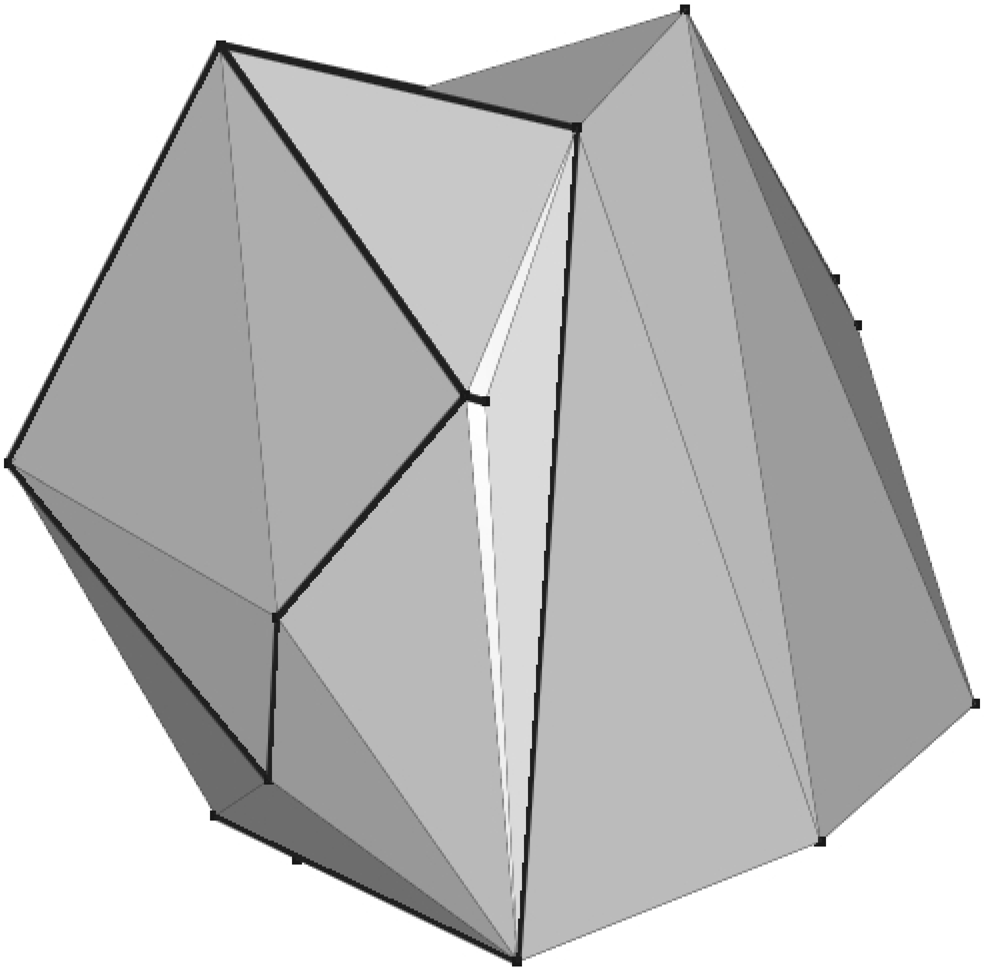
Divide And Conquer



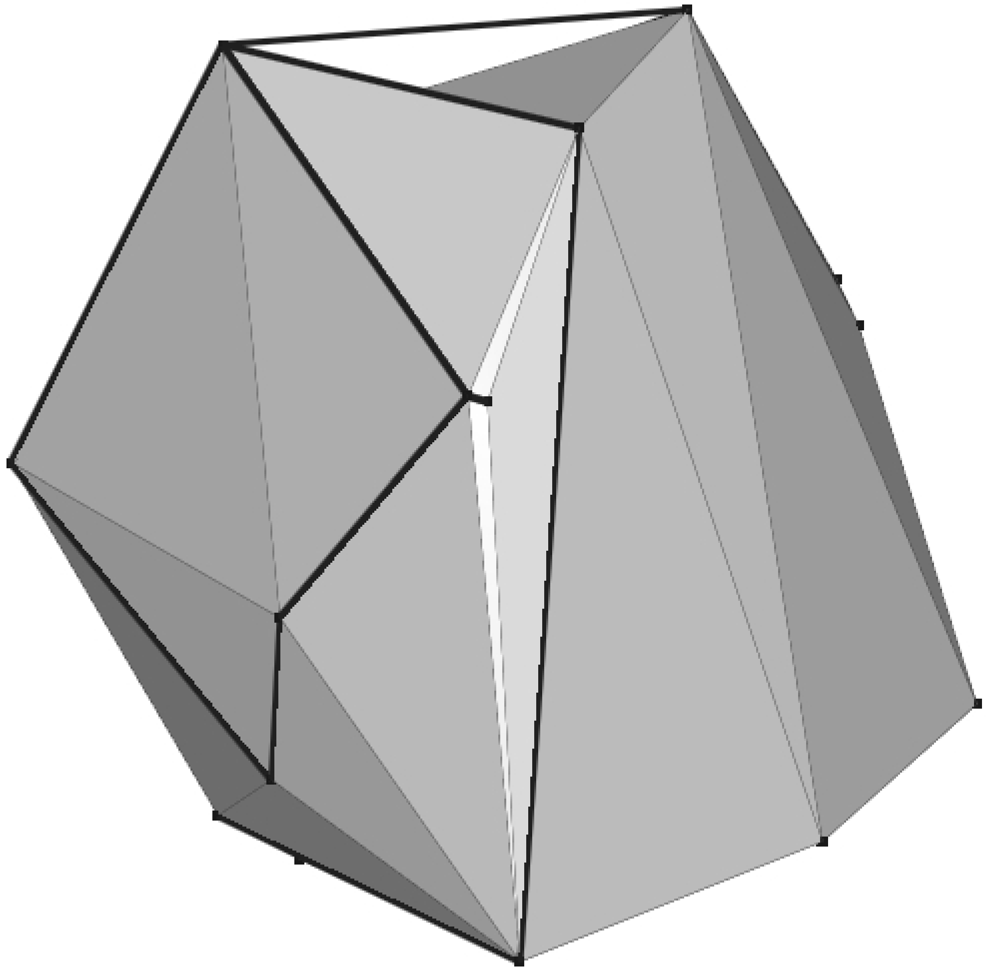
Divide And Conquer



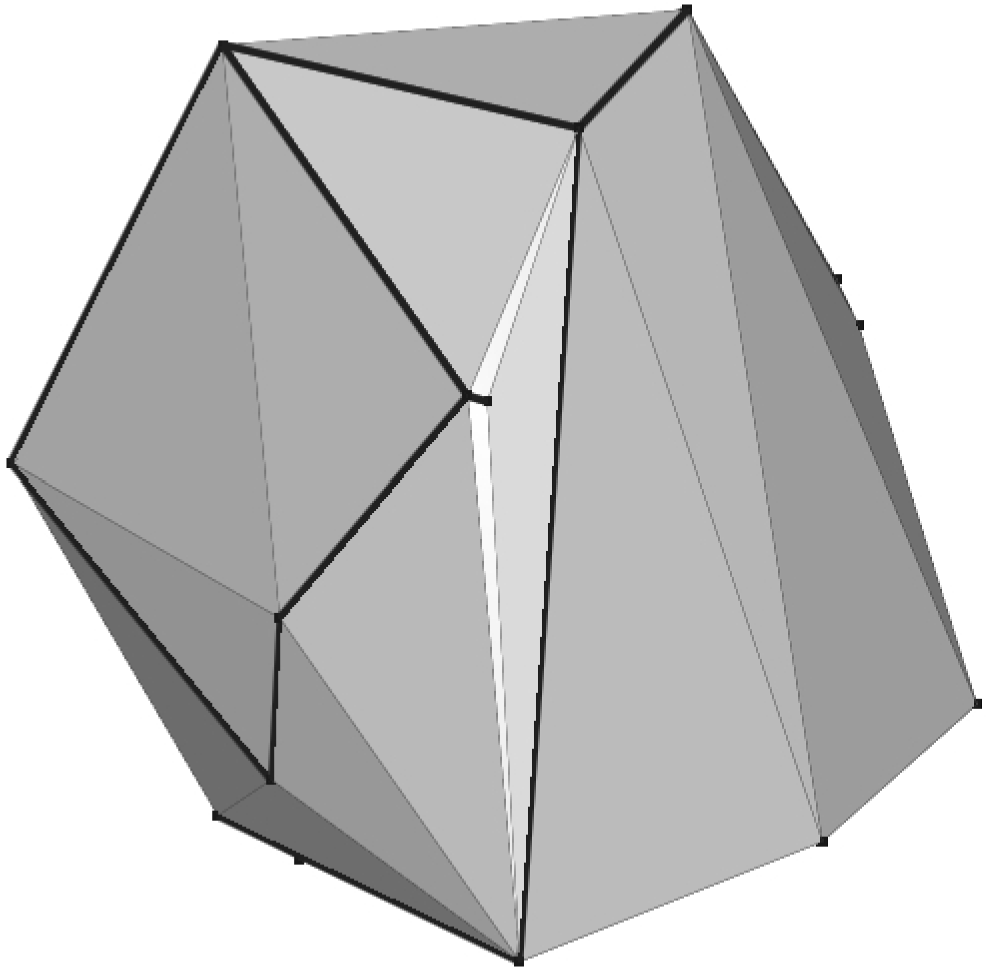
Divide And Conquer



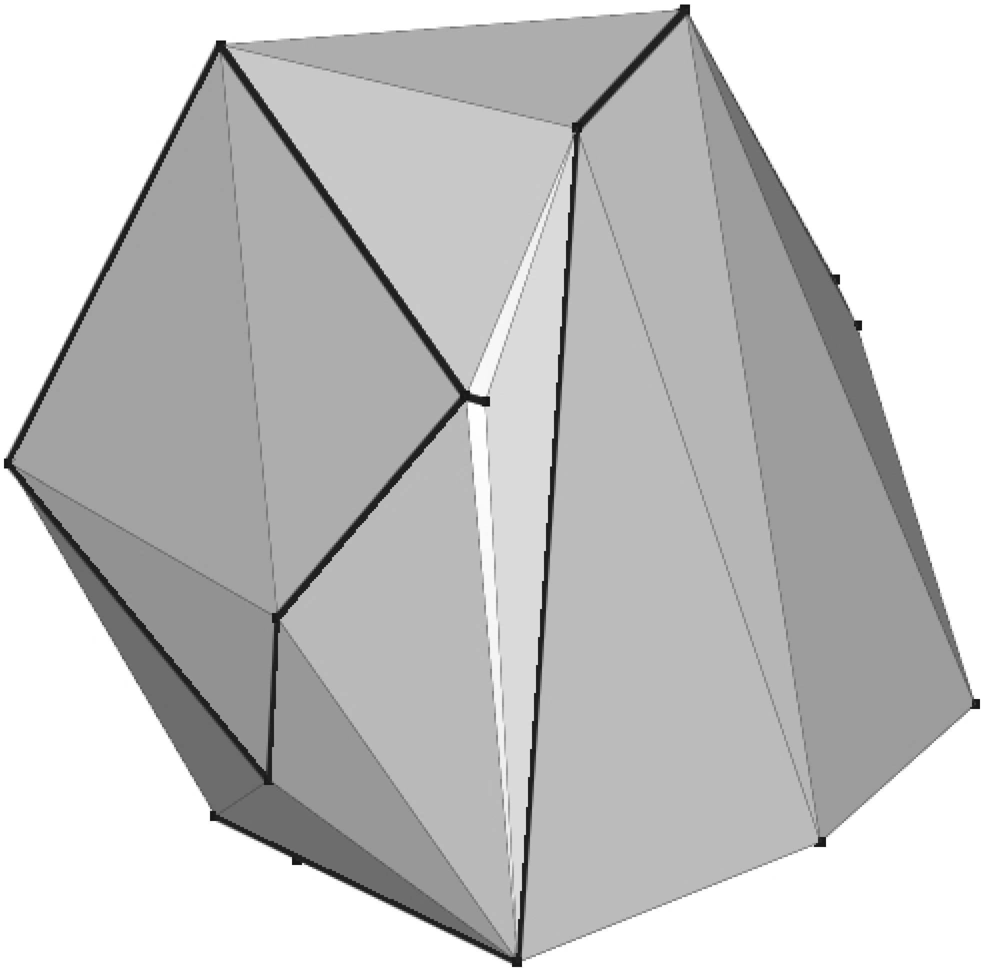
Divide And Conquer



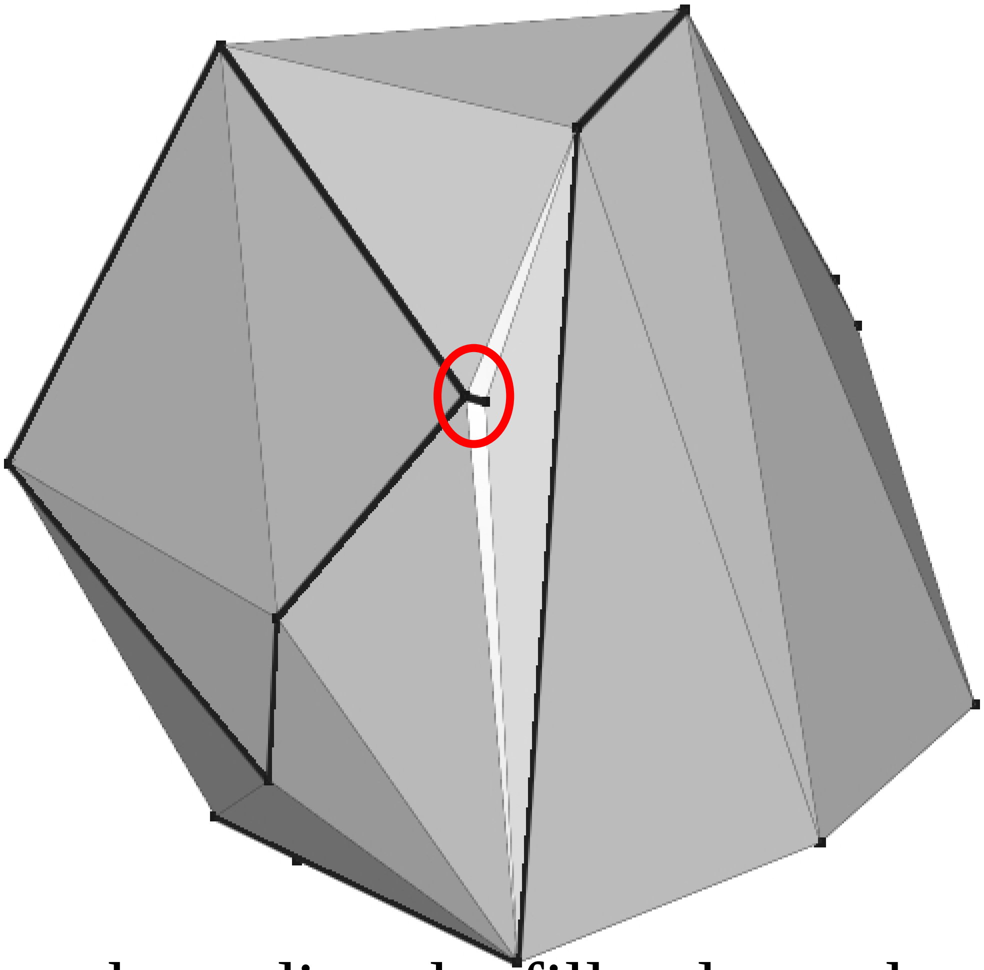
Divide And Conquer



Divide And Conquer



Divide And Conquer



Note: The curves bounding the fillet do not have to be simple