UNIVERSITY
OF OSLO

Convex Hull

What? Why? How?

Brage Isak Keiserås

17th March 2026



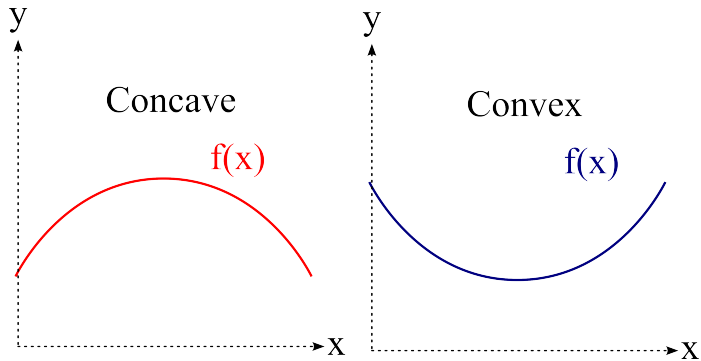
Outline

1. What?
2. Why?
3. How?

What?

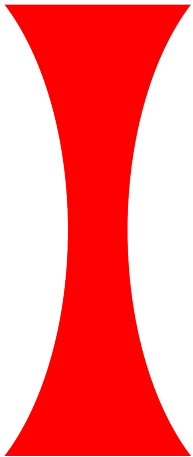
1. What?
2. Why?
3. How?

Concave/Convex Functions

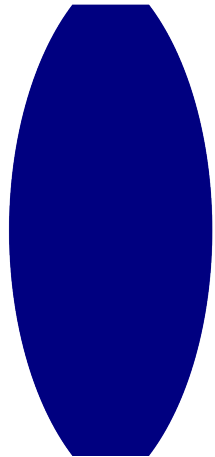


Concave/Convex Lenses

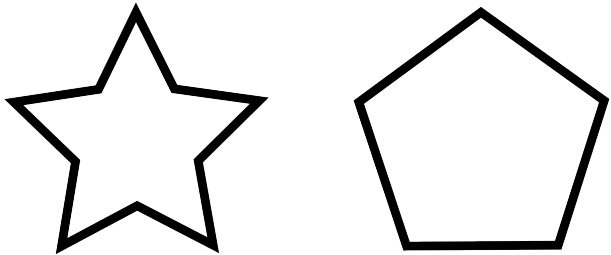
Concave



Convex



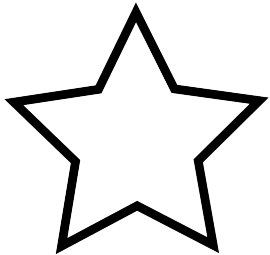
Concave/Convex

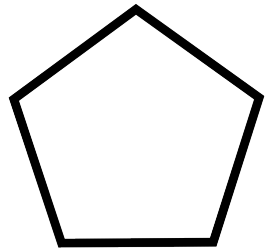


Concave/Convex

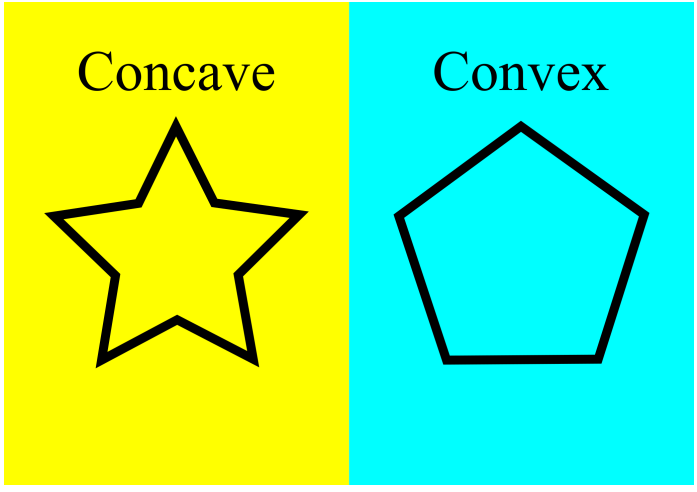
- An object is concave if it dips "inwards".
- If every line between any two points inside an object is contained inside an object then the object is convex.
  - For any points a, b inside O, ab is inside O

Concave/Convex

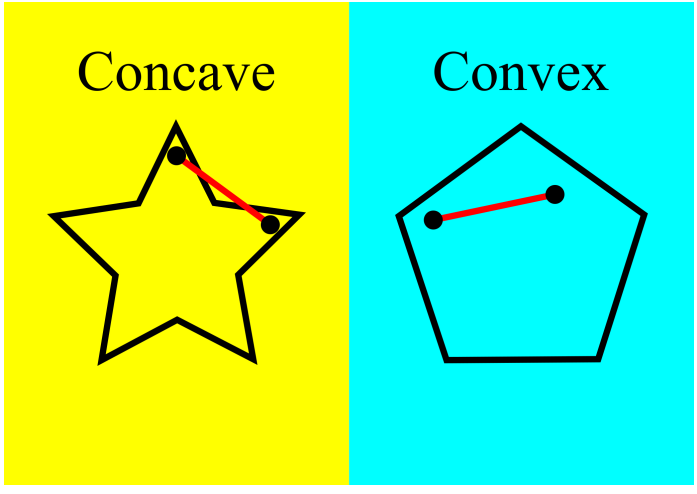




Concave/Convex



Concave/Convex



Convex Hull — What?

Convex Hull — What?

The convex hull of a set of points is the smallest convex object containing all points in the input.

Convex Hull — What?

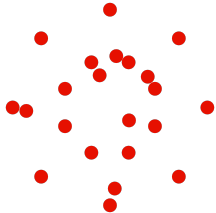
The convex hull of a set of points is the smallest convex object containing all points in the input.
If you imagine the points as nails nailed into a board, The convex hull is like a piece of string wrapped around the nails.

Convex Hull — What?

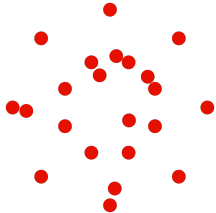
The convex hull of a set of points is the smallest convex object containing all points in the input.
If you imagine the points as nails nailed into a board, The convex hull is like a piece of string wrapped around the nails.

Convex Hull	
Input:	Set of points, P
Output:	Smallest object H such that H is convex, and all points in P are contained in H

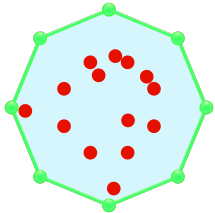
What is the convex hull of this guy?



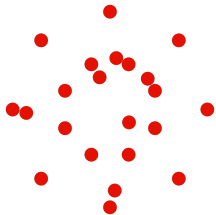
What is the convex hull of this guy?



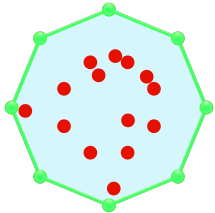
Is it this guy:



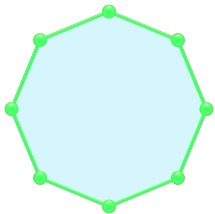
What is the convex hull of this guy?



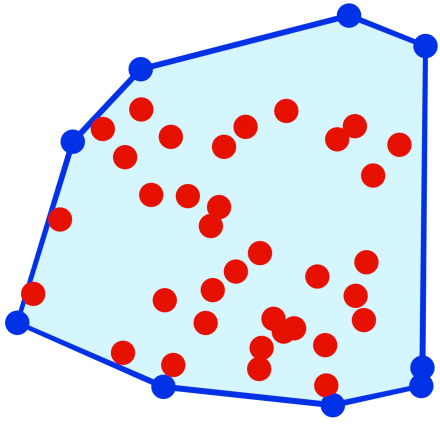
Is it this guy:



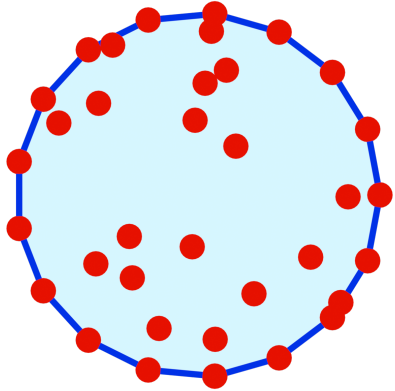
Or is it this guy:



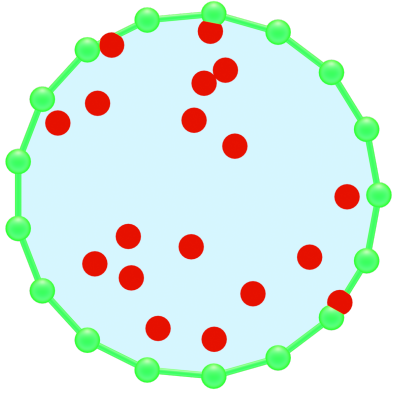
Video



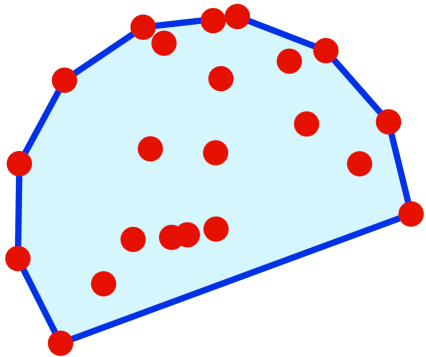
Concave or Convex?



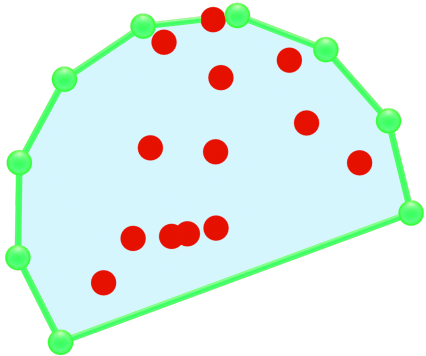
Concave or Convex?



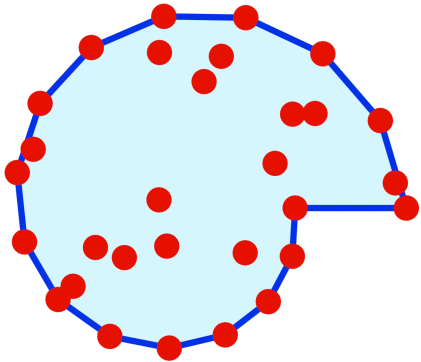
Concave or Convex?



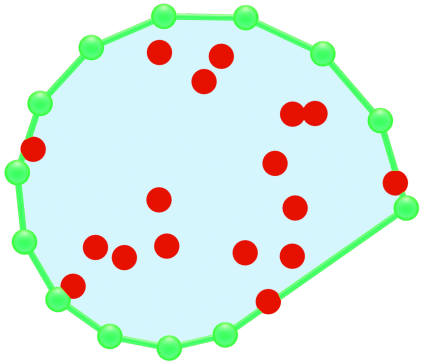
Concave or Convex?



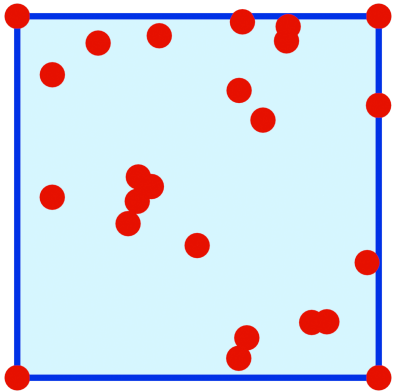
Concave or Convex?



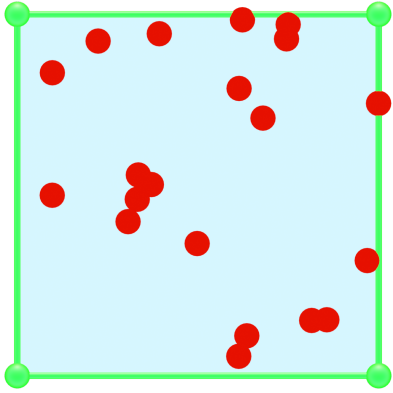
Concave or Convex?



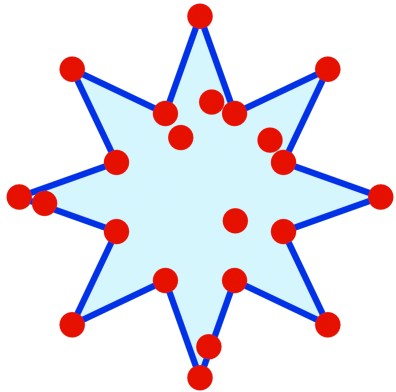
Concave or Convex?



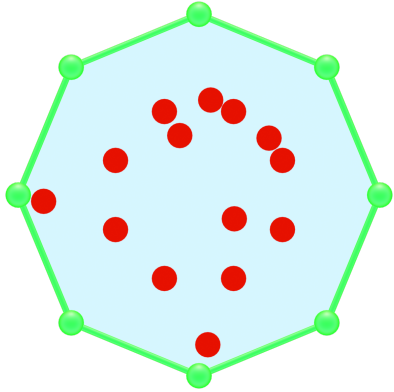
Concave or Convex?



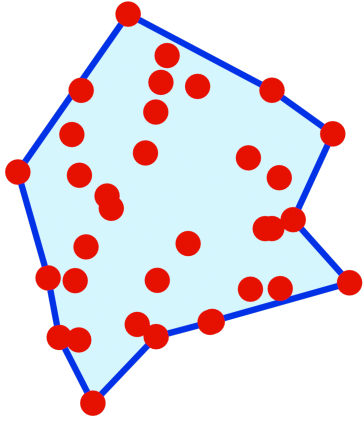
Concave or Convex?



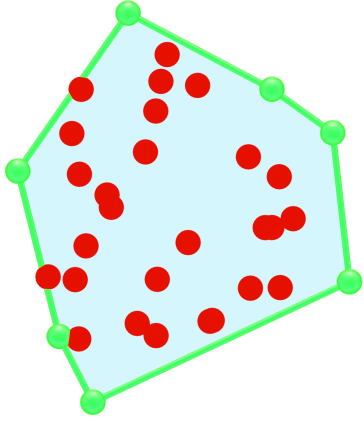
Concave or Convex?



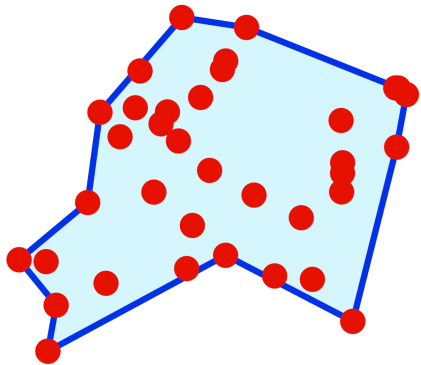
Concave or Convex?



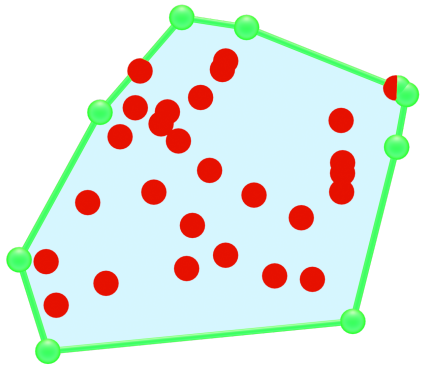
Concave or Convex?



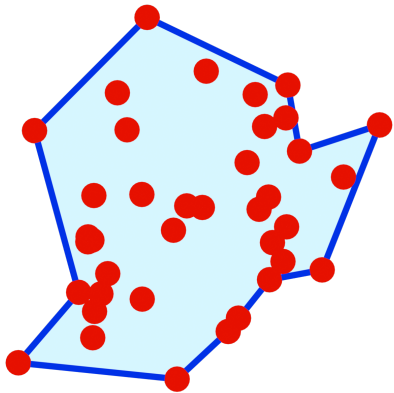
Concave or Convex?



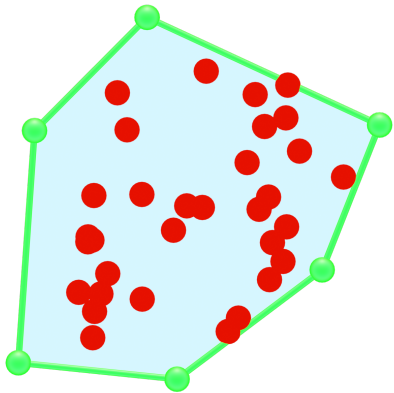
Concave or Convex?



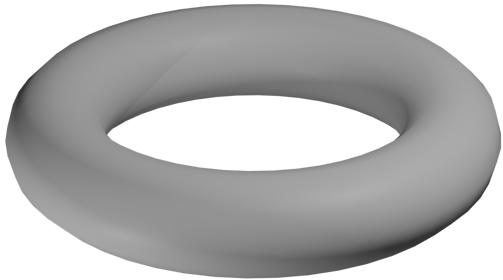
Concave or Convex?



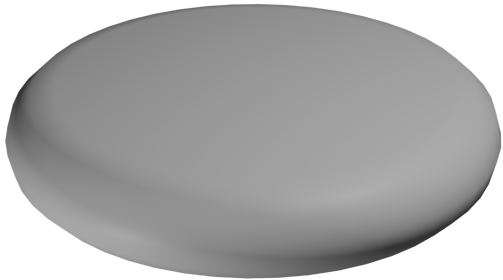
Concave or Convex?



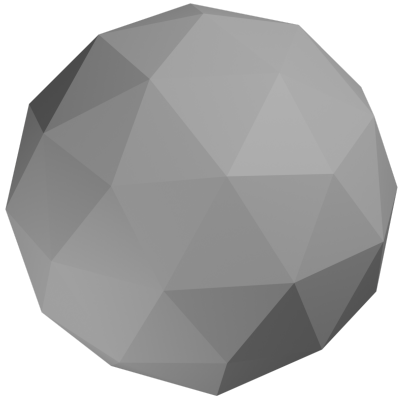
Concave or Convex?



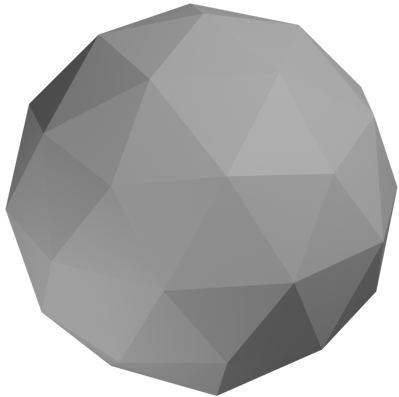
Concave



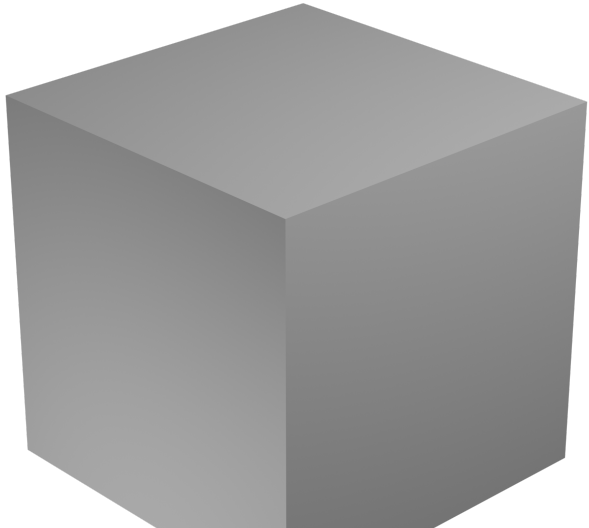
Concave or Convex?



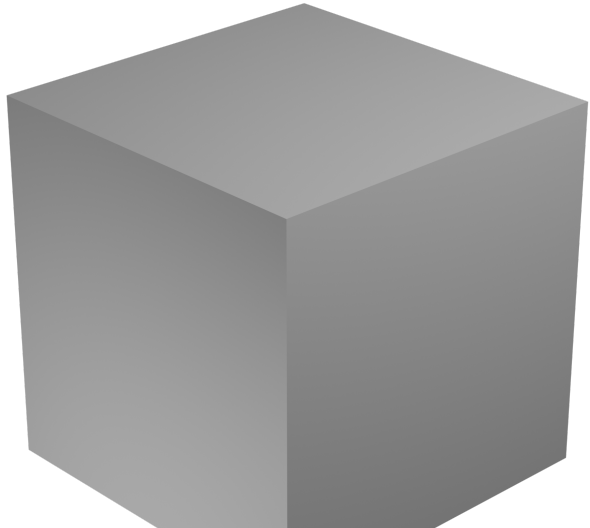
Convex



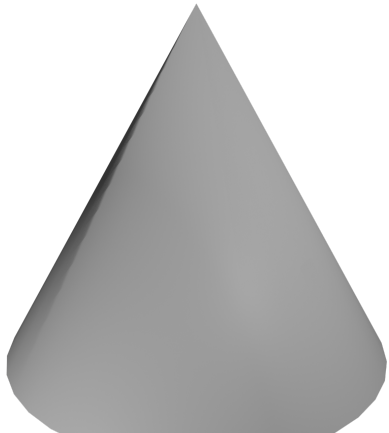
Concave or Convex?



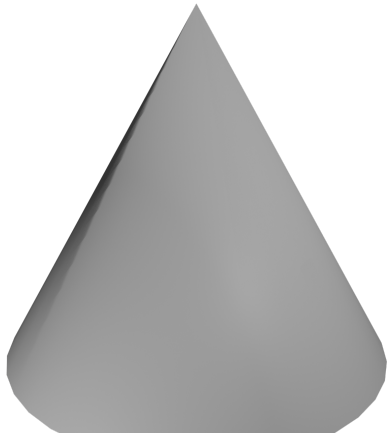
Convex



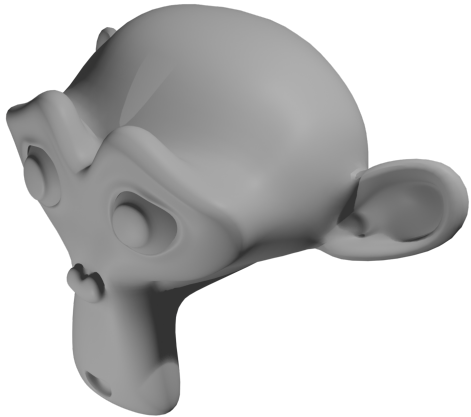
Concave or Convex?



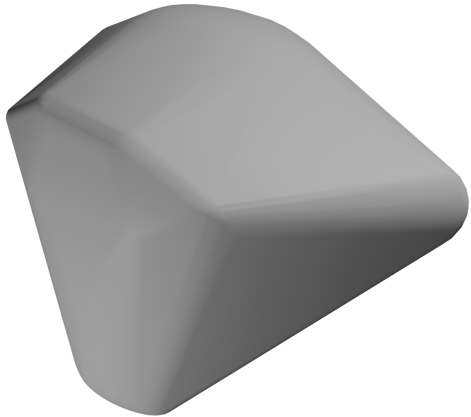
Convex



Concave or Convex?



Concave



Why?

1. What?
2. Why?
3. How?

Convex Hull — Why?

- Often easier (faster) to work with convex objects
  - First pass on convex hulls to quickly exclude objects
  - Then second pass in more detail

Convex Hull — Why?

- Often easier (faster) to work with convex objects
  - First pass on convex hulls to quickly exclude objects
  - Then second pass in more detail
- 3D models (animation and video games), simplify for
  - collision detection
  - physics simulation

Convex Hull — Why?

- Often easier (faster) to work with convex objects
  - First pass on convex hulls to quickly exclude objects
  - Then second pass in more detail
- 3D models (animation and video games), simplify for
  - collision detection
  - physics simulation
- Bézier curve lies within the convex hull of its control points
  - detecting intersections of curves

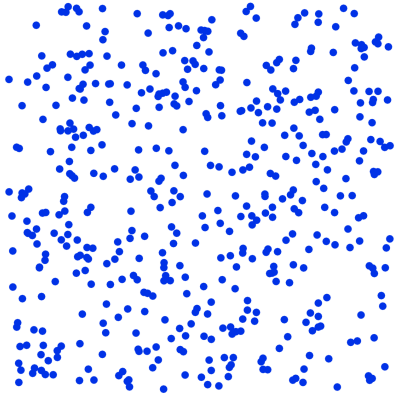
Convex Hull — Why?

- Often easier (faster) to work with convex objects
  - First pass on convex hulls to quickly exclude objects
  - Then second pass in more detail
- 3D models (animation and video games), simplify for
  - collision detection
  - physics simulation
- Bézier curve lies within the convex hull of its control points
  - detecting intersections of curves
- Calculating triangulation of points

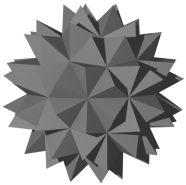
Convex Hull — Why?

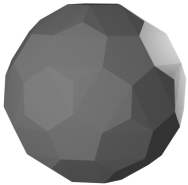
- Often easier (faster) to work with convex objects
  - First pass on convex hulls to quickly exclude objects
  - Then second pass in more detail
- 3D models (animation and video games), simplify for
  - collision detection
  - physics simulation
- Bézier curve lies within the convex hull of its control points
  - detecting intersections of curves
- Calculating triangulation of points
- Optimization

Case in point — triangulation



Case in point — collision







How?

1. What?
2. Why?
3. How?

Convex Hull — How?

Convex Hull — How?

What points are always on the hull?

Convex Hull — How?

The "boundary" points will be in the convex hull

Convex Hull — How?

The "boundary" points will be in the convex hull
Which points will be on the "boundary"?

Convex Hull — How?

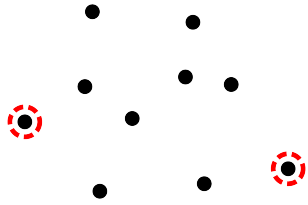
The "boundary" points will be in the convex hull
Which points will be on the "boundary"?

Observation
The points with Min_X and Max_X are both on the boundary of the hull

Convex Hull — How?

The "boundary" points will be in the convex hull
Which points will be on the "boundary"?

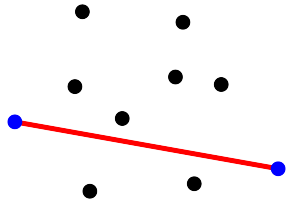
Observation
The points with Min_X and Max_X are both on the boundary of the hull



Convex Hull — How?

The "boundary" points will be in the convex hull
Which points will be on the "boundary"?

Observation
The points with Min_X and Max_X are both on the boundary of the hull

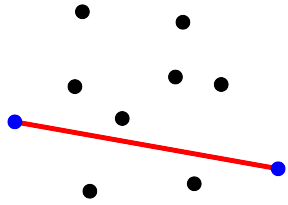


Convex Hull — How?

In-fact, since there is nothing "special" about the *X*-axis, this holds for any line

Observation

For any line, the point(s) furthest from this line is on the boundary of the convex hull

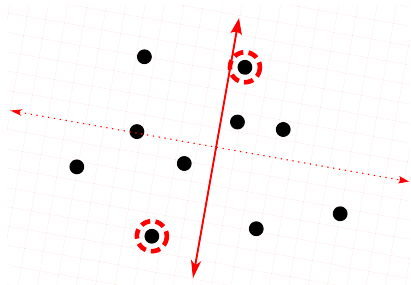


Convex Hull — How?

In-fact, since there is nothing "special" about the *X*-axis, this holds for any line

Observation

For any line, the point(s) furthest from this line is on the boundary of the convex hull



Convex Hull— How?

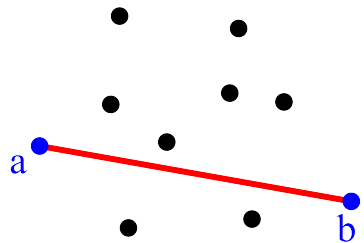
Can you utilize these two facts to create an algorithm?

Convex Hull— An idea

- For any line *ab*, if we rotate this line by 90 degrees we obtain a new line *e*
- The point *c* furthest from this line *e* is on the Convex Hull boundary
  - This gives us two new line segments *ac*, *cb*.

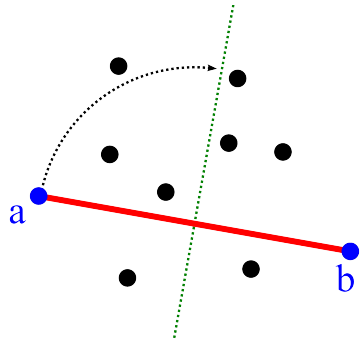
Convex Hull— An idea

- For any line *ab*, if we rotate this line by 90 degrees we obtain a new line *e*
- The point *c* furthest from this line *e* is on the Convex Hull boundary
  - This gives us two new line segments *ac*, *cb*.



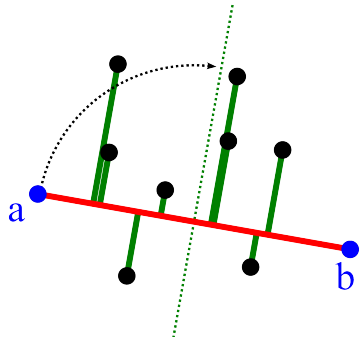
Convex Hull— An idea

- For any line *ab*, if we rotate this line by 90 degrees we obtain a new line *e*
- The point *c* furthest from this line *e* is on the Convex Hull boundary
  - This gives us two new line segments *ac*, *cb*.



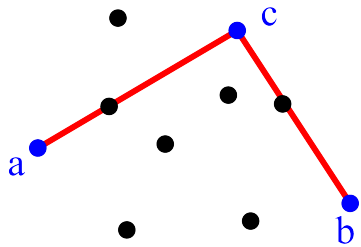
Convex Hull— An idea

- For any line *ab*, if we rotate this line by 90 degrees we obtain a new line *e*
- The point *c* furthest from this line *e* is on the Convex Hull boundary
  - This gives us two new line segments *ac*, *cb*.



Convex Hull— An idea

- For any line *ab*, if we rotate this line by 90 degrees we obtain a new line *e*
- The point *c* furthest from this line *e* is on the Convex Hull boundary
  - This gives us two new line segments *ac*, *cb*.



Convex Hull — How?

What about now, can you make an algorithm?

Convex Hull — How?

QuickHull *P* (unoptimized):

- find leftmost and rightmost points in *P*
- We then recursively find the rest of the points using
  - findHull *P*, *ab* — finding the convex hull points above *ab*
  - findHull *P*, *ba* — finding the convex hull points below *ab*

Convex Hull — How?

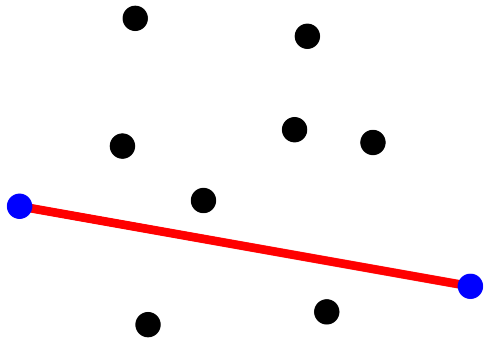
QuickHull *P* (unoptimized):

- find leftmost and rightmost points in *P*
- We then recursively find the rest of the points using
  - findHull *P*, *ab* — finding the convex hull points above *ab*
  - findHull *P*, *ba* — finding the convex hull points below *ab*

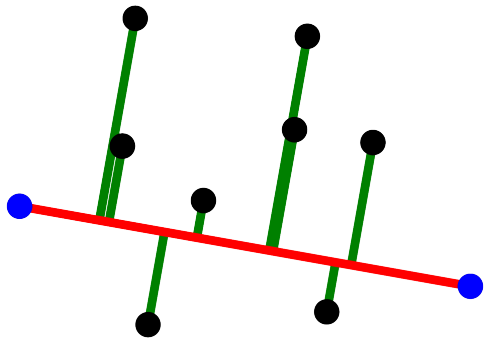
FindHull *P*, *ab* (unoptimized):

- We are adding the points on the convex hull between *a* and *b*
- Find the point *c* furthest above the line *ab*
  - We then get two new lines *ac*, *cb*
  - Recursively call findHull(*P*, *ac*), findHull(*P*, *cb*) on them

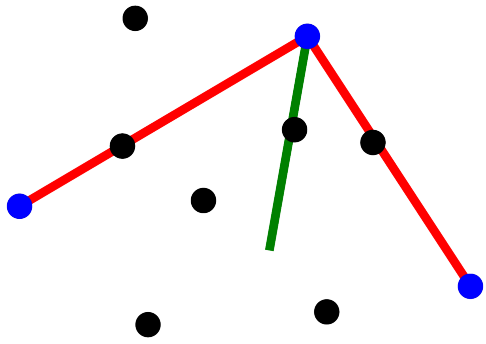
Example execution (unoptimized)



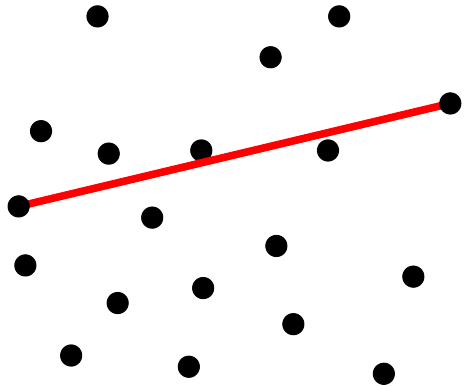
Example execution (unoptimized)



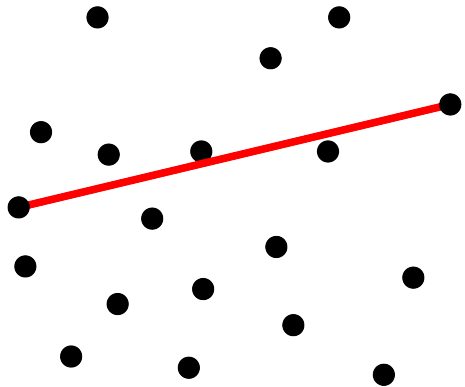
Example execution (unoptimized)



Quick-Hull Optimization

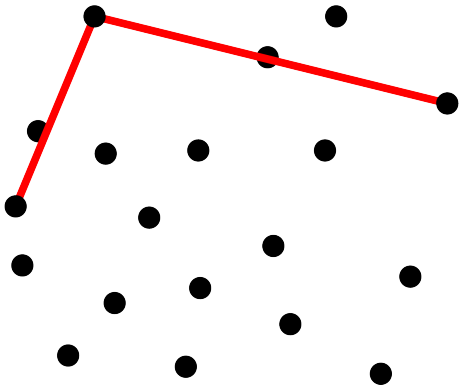


Quick-Hull Optimization



- Find the point *c* furthest above the line *ab*

Quick-Hull Optimization

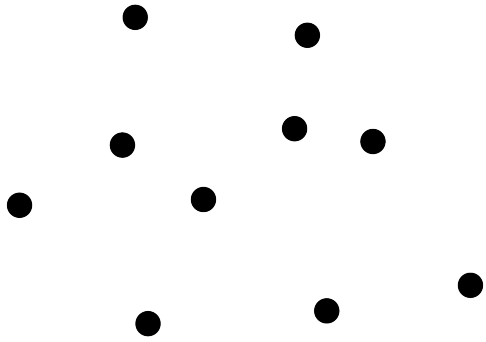


- Find the point *c* furthest above the line *ab*
  - We then get two new lines *ac*, *cb*
  - Recursively call
    - findHull(*P*, *ac*)
    - findHull(*P*, *cb*)

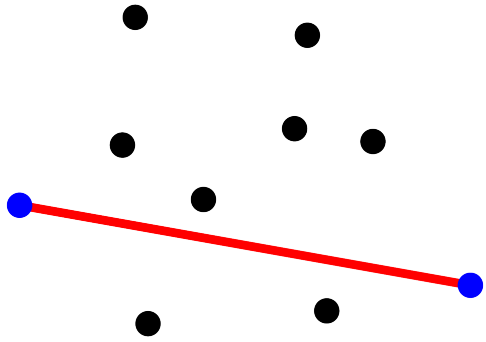
Quick-Hull Optimization

At each line we can discard all points below that line
This cleanly partitions the rest of our points

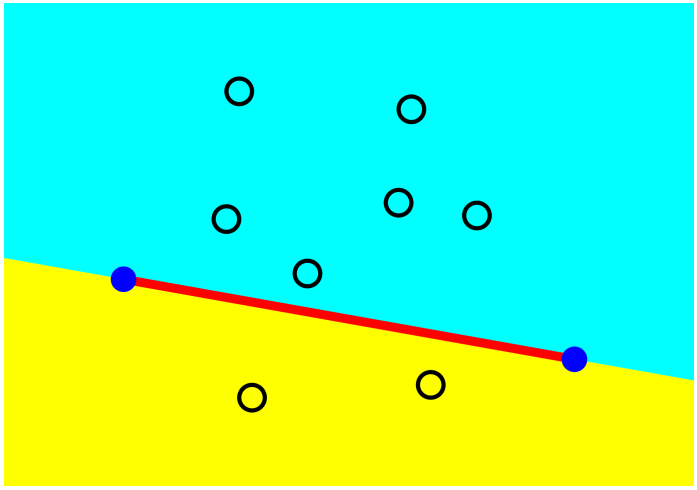
Quick-Hull



Quick-Hull — create line



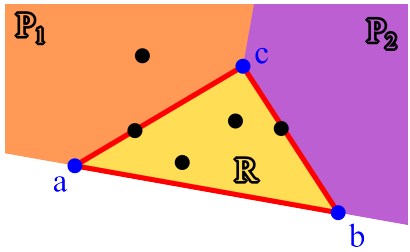
Quick-Hull — call findHull on cyan and yellow



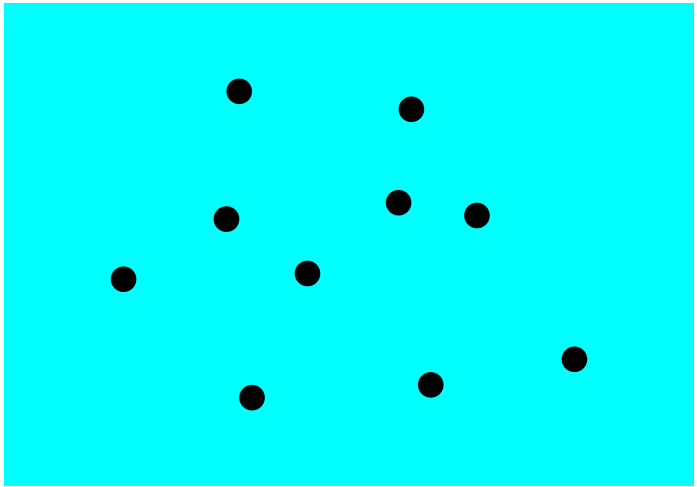
Convex Hull — How?

Find-Hull *ab*:

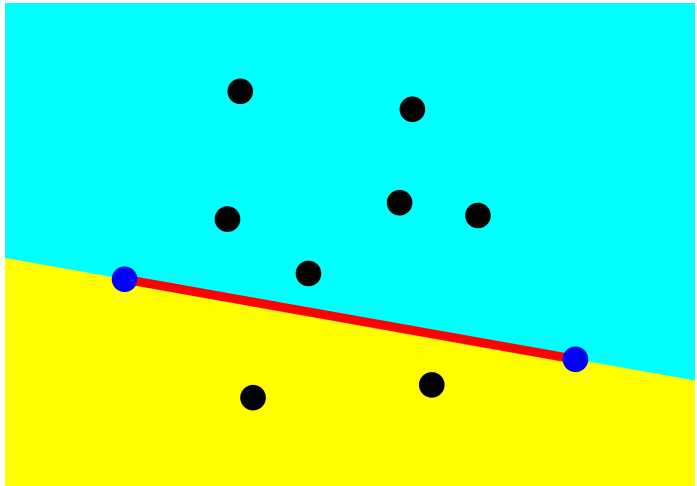
- find point $c \in P$ furthest from line *ab*
- *c* splits (parititions) *P* in three: P_1, P_2, and *R*
  - line segments *ac*, *bc*, and *ba* form a triangle
    - *R*: The points inside the triangle
    - P_1: The points "left" of the triangle
    - P_2: The points "right" of the triangle
- recursively findHull on *ac* in P_1 and *cb* in P_2



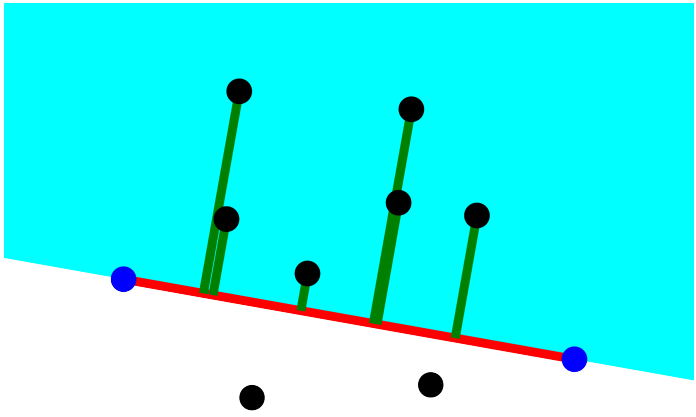
Example – start



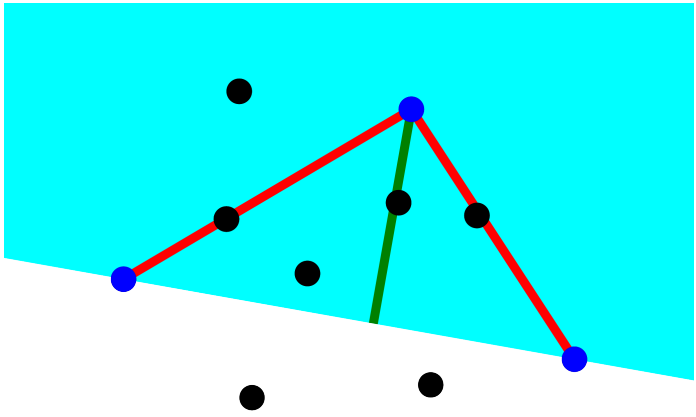
Example – minX maxX



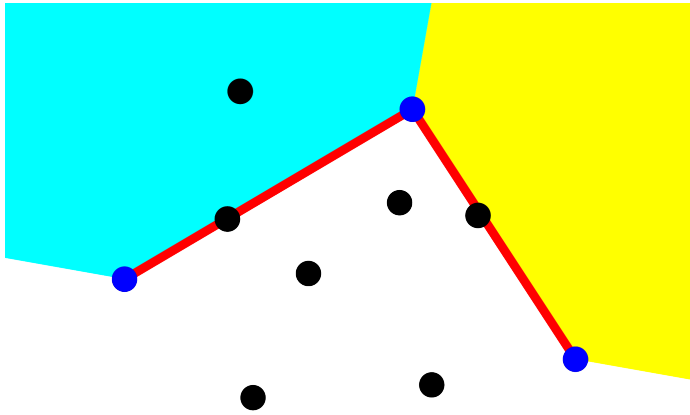
Example – recursion depth = 0



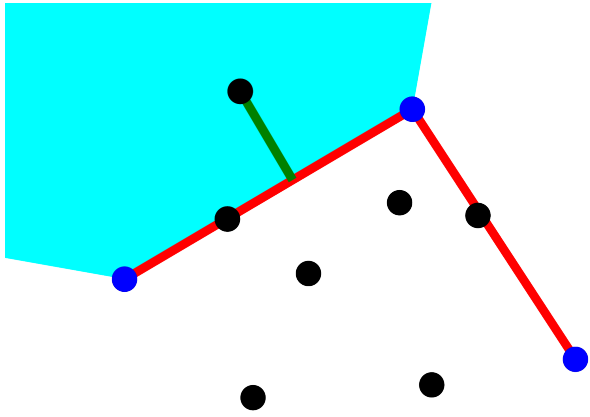
Example – recursion depth = 0



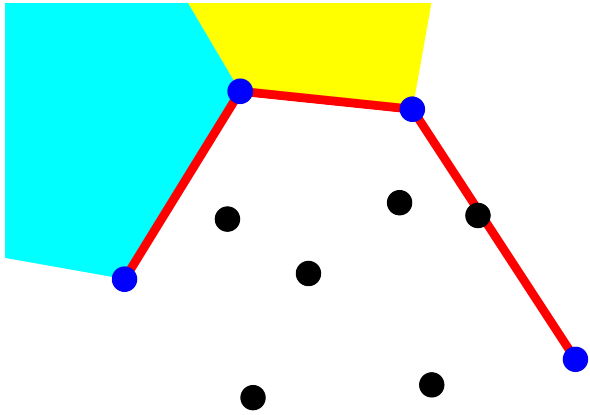
Example – recursion depth = 1



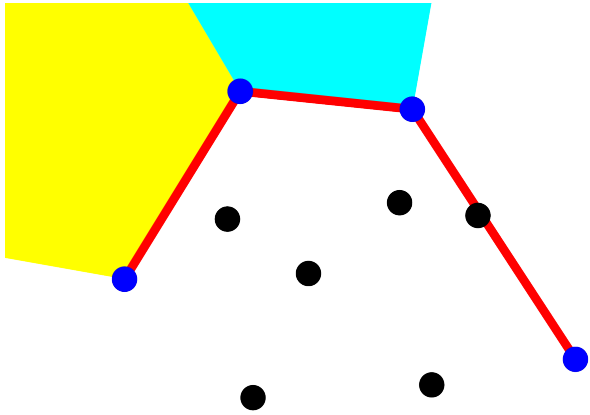
Example – recursion depth = 1



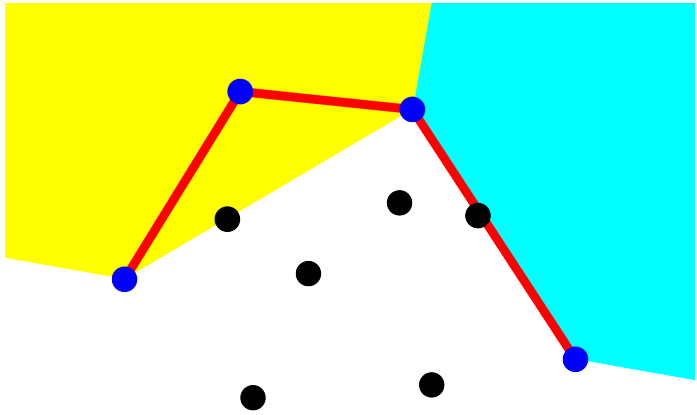
Example – recursion depth = 2



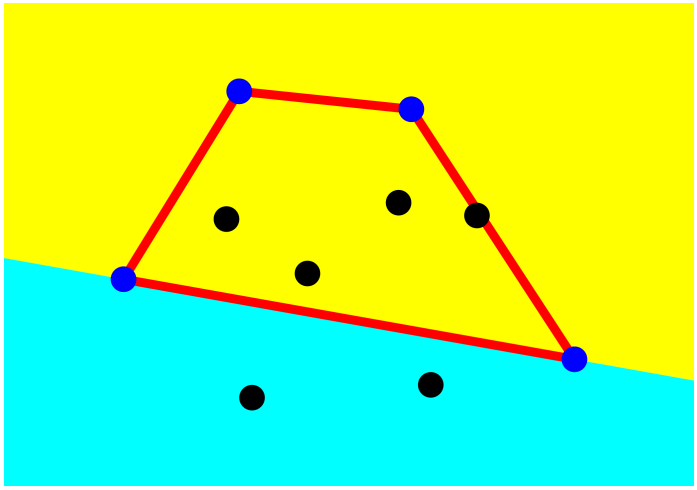
Example – recursion depth = 2



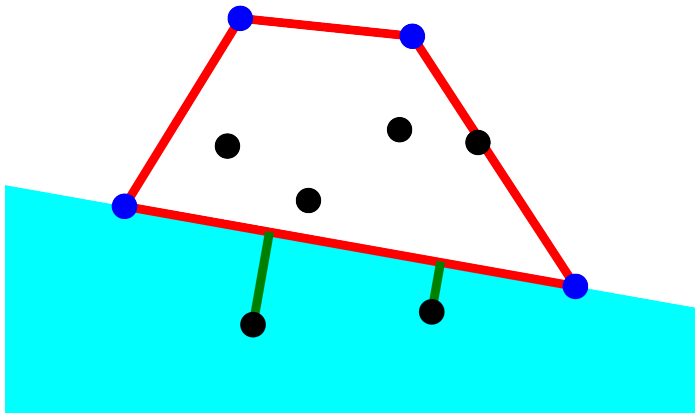
Example – recursion depth = 1



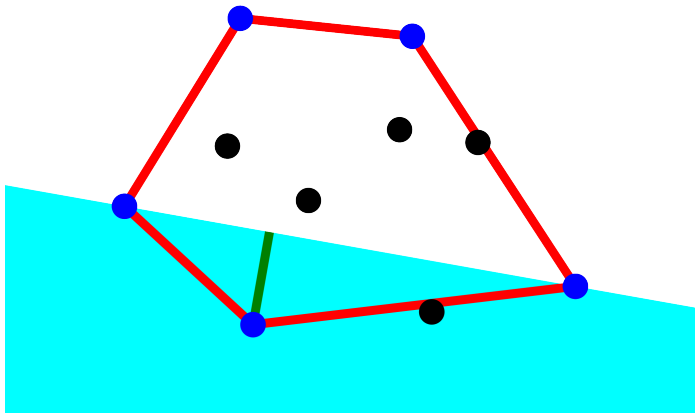
Example – recursion depth = 0



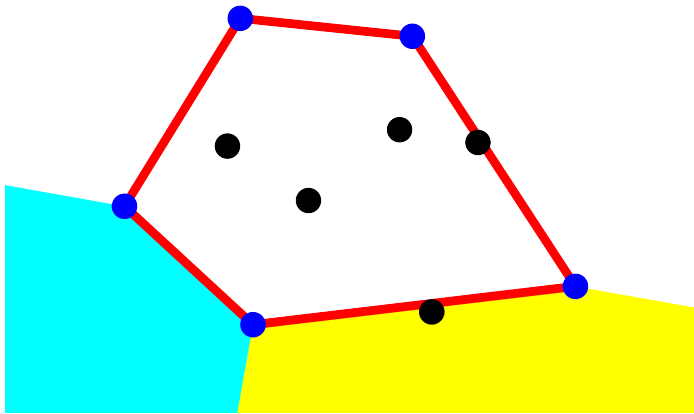
Example – recursion depth = 0



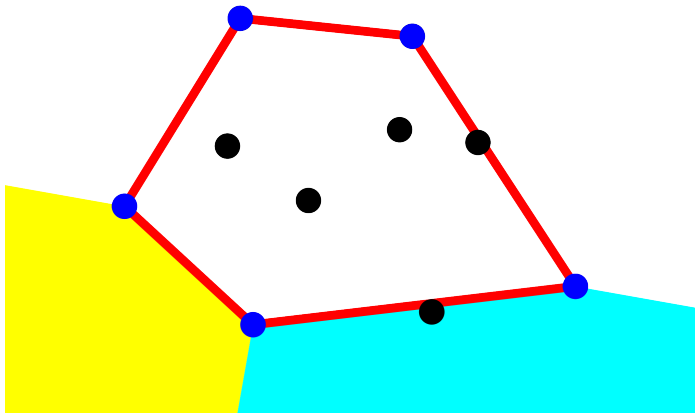
Example – recursion depth = 0



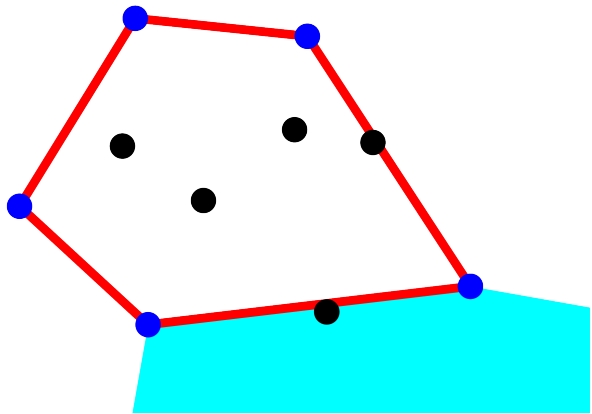
Example – recursion depth = 1



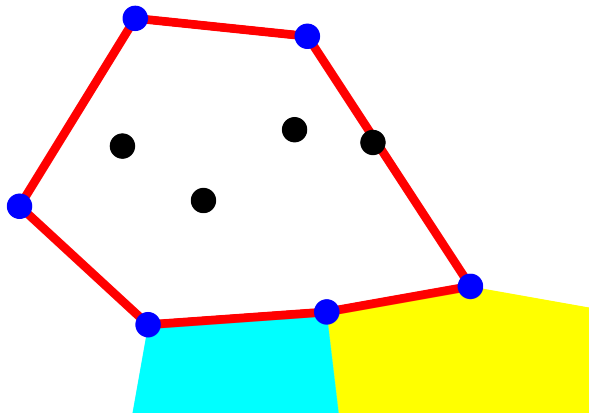
Example – recursion depth = 1



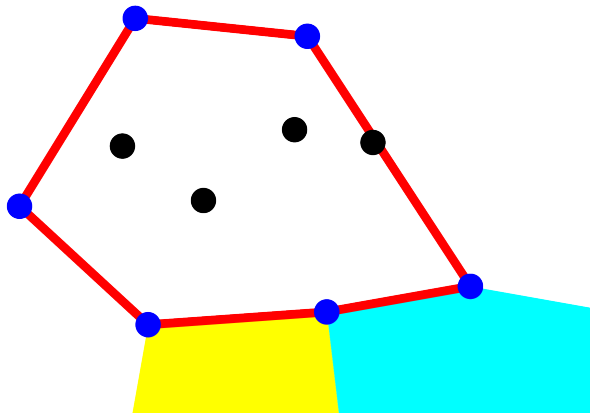
Example – recursion depth = 1



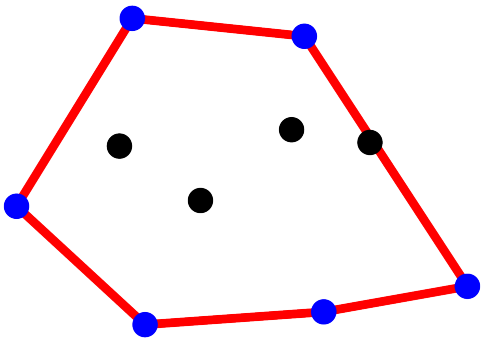
Example – recursion depth = 2



Example – recursion depth = 2



Example – finished



QuickHull

```
1: Function quickHull(S)
2:     a = point in P with min x;
3:     b = point in P with max x;
4:     addToHull(a); addToHull(b);
       // S1 : points above line ab
5:     S1 = pointsToLeft(S, ab);
       // S2 : points below line ab
6:     S2 = pointsToLeft(S, ba);
7:     findHull(S1, a, b);
8:     findHull(S2, b, a);
```

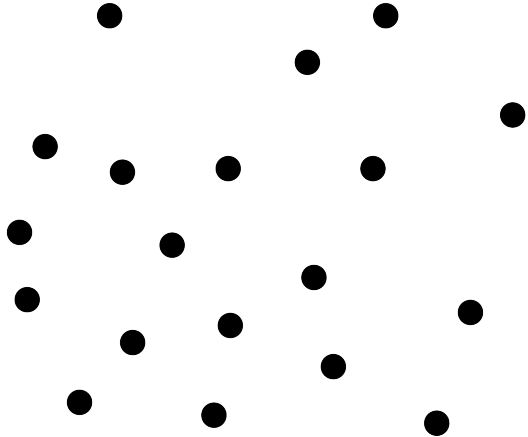
```
1: Function findHull(S, a, b)
2:     if S == ∅ then
3:         return;
4:     c = point in S furthest from ab;
5:     addToHull(c);
6:     S1 = pointsToLeft(S, ac);
7:     S2 = pointsToLeft(S, cb);
8:     findHull(S1, a, c);
9:     findHull(S2, c, b);
```

Utilities

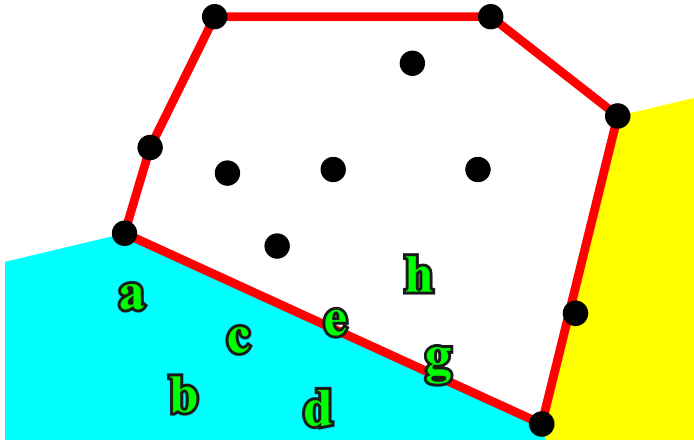
```
1: Function pointsToLeft(S, ab)
2:     S_left = {};
3:     foreach p ∈ S do
4:         d = distanceToLine(p, ab);
5:         if d ≤ 0 then
6:             S_left.add(p);
7:     return S_left;
```

```
1: Function distanceToLine(P, P1P2)
2:     P = (x, y);
3:     P1 = (x1, y1);
4:     P2 = (x2, y2);
5:     a = y1 − y2;
6:     b = x2 − x1;
7:     c = x1 · y2 − x2 · y1;
8:     return (a · x + b · y + c) / √(a² + b²);
```

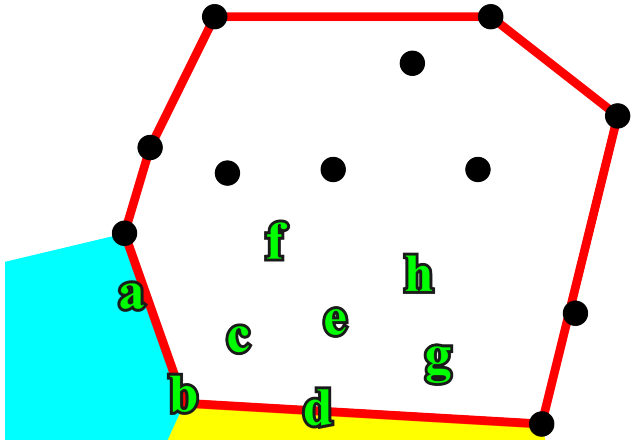
Guess the execution



Guess the execution



Guess the execution



Running time

Running time

worst-case:

Running time

worst-case: $O(n^2)$
average-case:

Running time

worst-case: $O(n^2)$
average-case: $O(n \cdot lg(n))$

- quickHull divides input in roughly two
- findHull divides input in roughly two

Running time

Running time

worst-case:

Running time

worst-case: $O(n^2)$

- If the input almost is a convex hull

Running time

worst-case: $O(n^2)$

- If the input almost is a convex hull

Best-case:

- The input has most points inside the hull, not close to the boundary

A note on implementation

Take note that the order in which the points are added to the hull matters for how we construct the edges.

A note on the name

QuickHull ~ QuickSort?

A note on the name

QuickHull ~ QuickSort?

- Both partition the input in two parts at each step

A note on the name

QuickHull ~ QuickSort?

- Both partition the input in two parts at each step
- Both are average-case $O(n \cdot lg(n))$, worst-case $O(n^2)$

A note on the name

QuickHull ~ QuickSort?

- Both partition the input in two parts at each step
- Both are average-case $O(n \cdot lg(n))$, worst-case $O(n^2)$
- Both are in-place

A note on the name

QuickHull $\sim$ QuickSort?

- Both partition the input in two parts at each step
- Both are average-case $O(n \cdot lg(n))$, worst-case $O(n^2)$
- Both are in-place

Implies the existance of ...?

Generalization - 3D?

Generalization - parallel