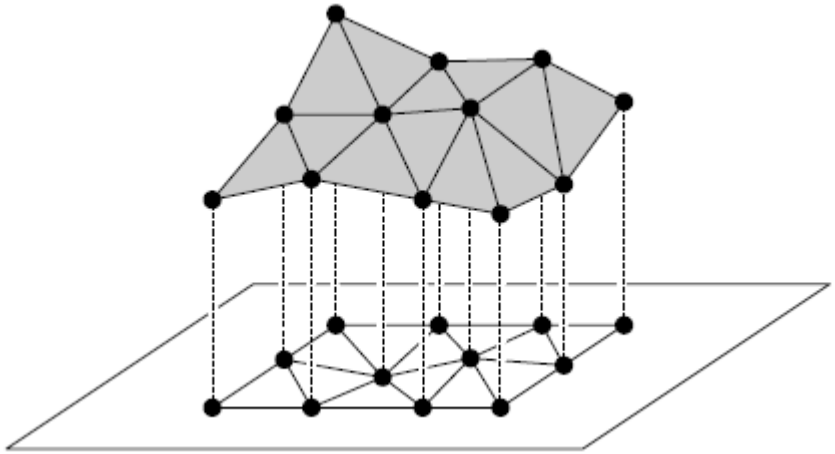
Triangulation and Convex Hull

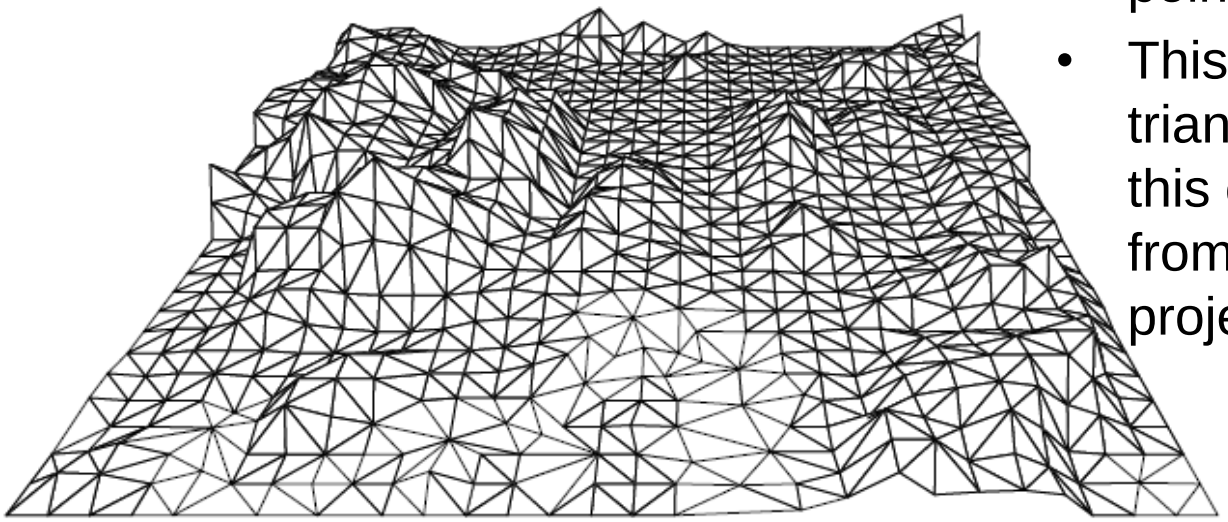
8th November 2018

Agenda

1. Triangulation. No book, the slides are the curriculum
2. Finding the convex hull. Textbook, 8.6.2

Triangulation and terrain models



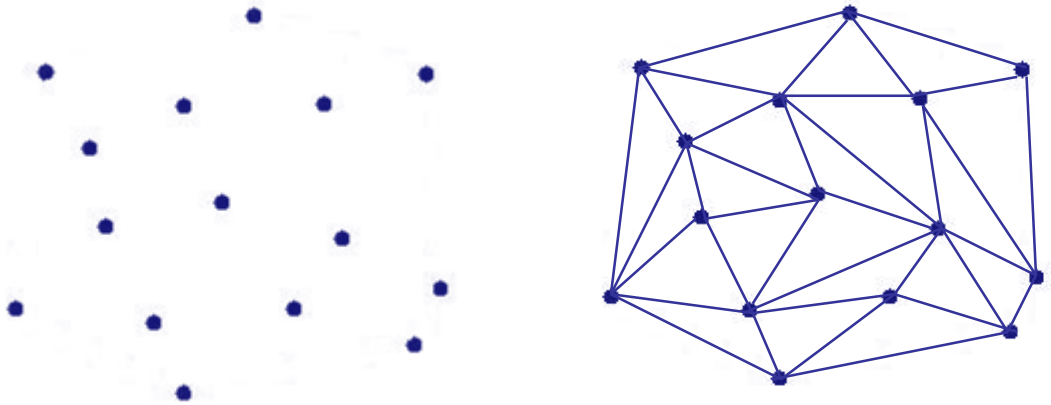


- Here we have measured the elevation of a number of points in the terrain.
- Each point is projected down to a “sea level plane”. Here we have used a “reasonable” triangulation of the resulting points.
- This triangulation also gives a triangulation of the terrain, and this can easily be drawn, e.g. from the side and in any projection

Triangulation

- The general problem is finding a triangulation of a given set of points in the plane.

  Below, a set of points are given (left), and a arbitrarily chosen triangulation of this set is drawn (right)

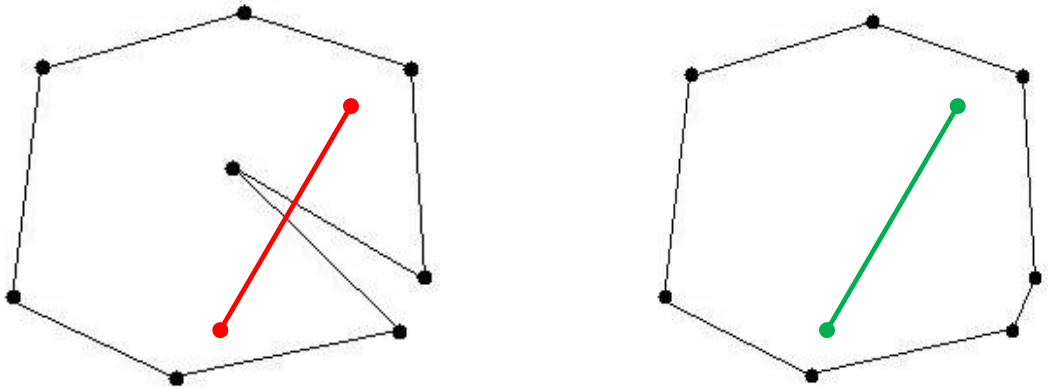


- When finding a triangulation, a polygon should also be given, where the corners are are points of the given point set, with the rest of the points inside the polygon. (To define a boundary.)
- We will assume that this polygon always is the *convex hull* of the set of points (see next slide).

The convex hull of a point set

Let P be a set of points in the plane ($\mathbf{R}^2$) (Can also be defined for $\mathbf{R}^k$, $k > 2$)

Definition: A set $Q \in \mathbf{R}^2$ is *convex* if:
for all $q_1, q_2 \in Q$ the line q_1q_2 is fully within Q.



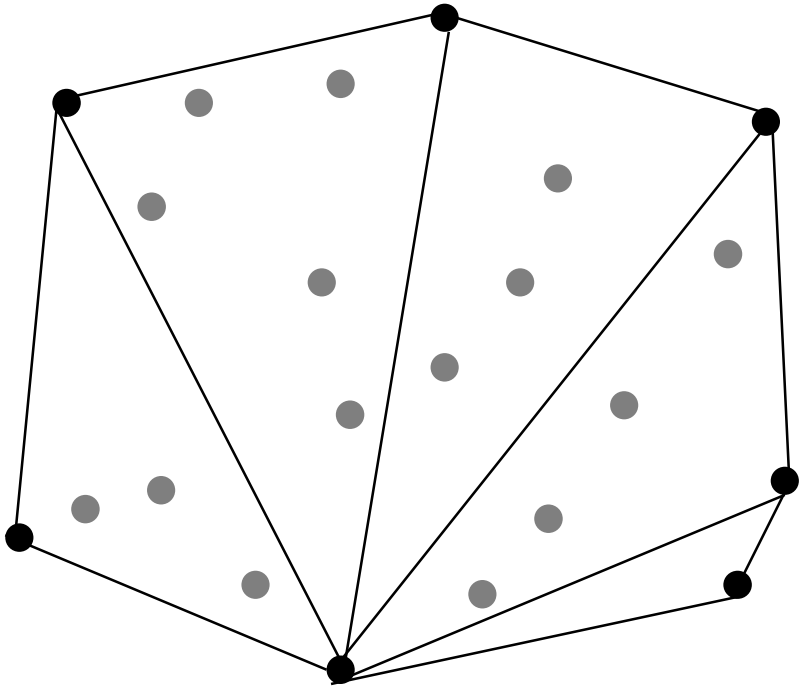
Definition: The *convex hull* of a set of points $P \in \mathbf{R}^2$ is the smallest convex set Q that contains all the points of P.

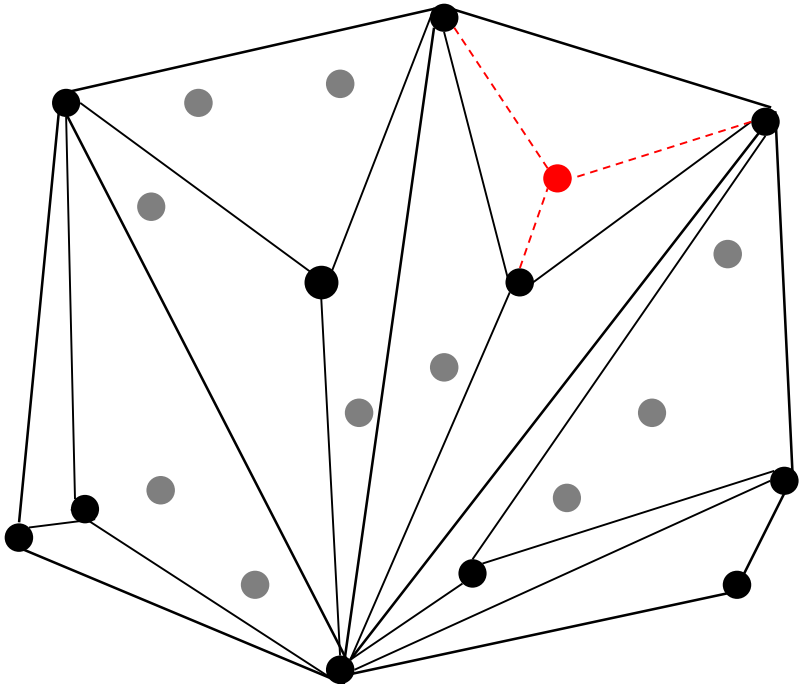
How many triangles and edges will a triangulation consist of?

- This way of counting also defines an algorithm for finding a triangulation (and we shall use a refinement of this algorithm later)
- Assume that the outer (convex) polygon has n corners, and that there are m points inside (so that the total number of points is $n+m$).
  - We can find a triangulation of the corners by choosing one of them, p, and draw an edge to the $n-3$ corners that do not already have an edge to p.
  - Together with the outer edges this gives $n + (n - 3) = 2n - 3$ edges, and $n - 2$ triangles.
  - We then take each of the inner points, q, and do as follows: We find the triangle that q resides in, and draw edges from q to the three corners of this triangle. This gives 3 extra edges and 2 extra triangles for each of the m inner points
  - We then get:

    Number of edges: $2n – 3 + 3m = 2n + 3m – 3$

    Number of triangles: $n - 2 + 2m = n + 2m - 2$
- Even if this is a special way to construct a triangulation, the answer (number of edges and triangles) holds for any triangulation, even if the outer polygon is not convex. (We do not prove this.)





Constructing a triangulation

(an illustration of the previous slide)

- We have *n* corners in the outer polygon and *m* internal points
- The upper figure shows the situation just after the *n*-3 diagonals are drawn, the lower shows a later situation, in which we want to include the red node.
- The new red node gives:
  - Three extra edges
  - Two extra triangles
- Thus the number of edges will be:

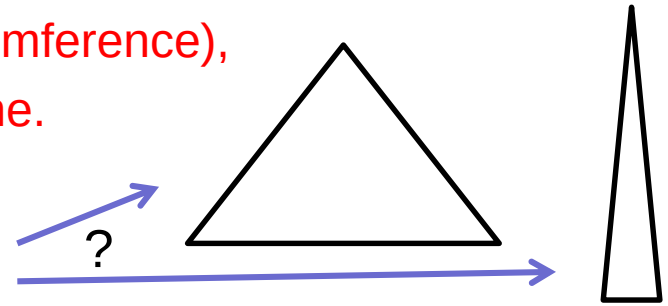
  $n + (n - 3) + 3m = 2n + 3m - 3$

  Number of triangles:

  $n - 2 + 2m = n + 2m - 2$

There are a number of different triangulations of the same set of points

- We assume that the following special cases do not occur:
  1) Four points lie on one circle (along the circumference),
  2) all the points occurs on the same straight line.



- And, what is a “good” triangulation?
  - Usually one where each triangle is as close to an *equilateral* triangle (with all angles equal to 60^{O}) as possible.

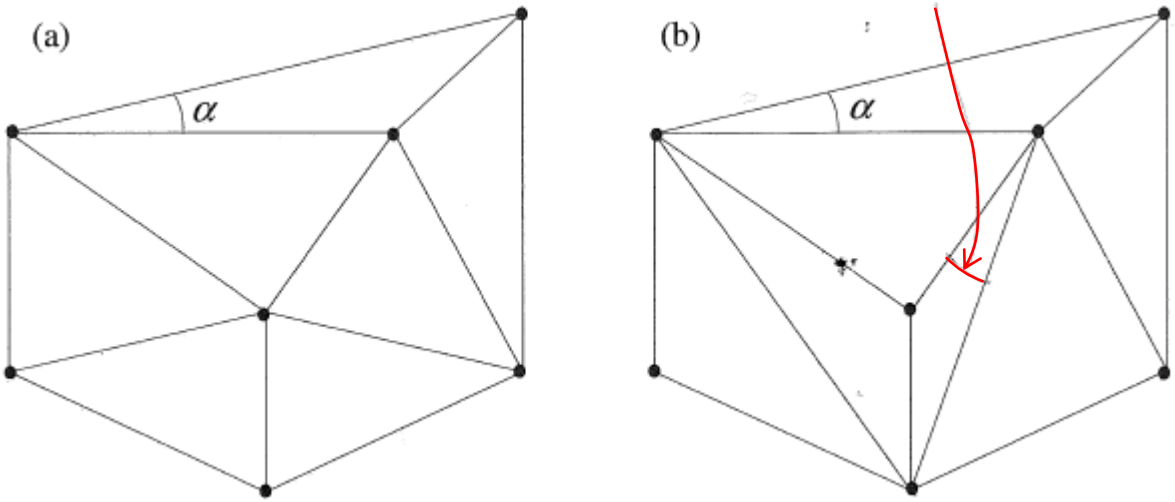
  Two reasonable goals could be
  - To minimize the maximal angle : The largest angle is as small as possible (And notice: The largest angle in a triangle is always at least 60^{O})
  - To maximize the minimal angle: The smallest angle (which is never larger than 60^{O}) is as large as possible.

- It turns out that *max-of-min* is easier to handle than *min-of-max,* so it’s most commonly used.

Definition of a Delaunay-triangulation (max-of-min)

- Roughly speaking, a triangulation of a set of points is, a *Delaunay triangulation* if, over all triangulations, the smallest angle is as large as possible.
- Or more precisely: When the angles of a triangulation are sorted from smallest to largest, the *Delaunay triangulation* is the sequence with the highest lexicographic order.

This angle larger than alfa, but smaller than all other angles in (a)

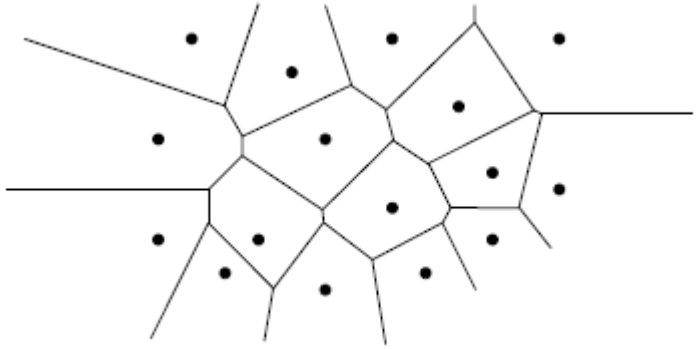


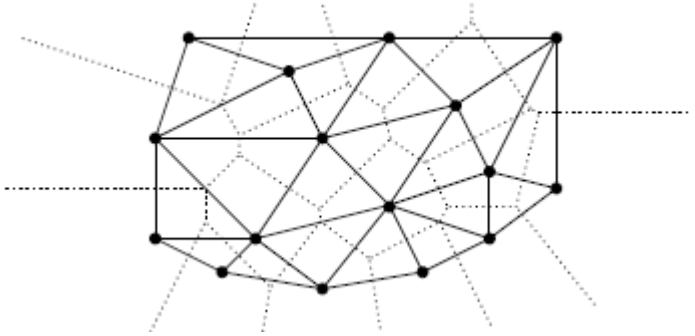
- Figure (a) is a Delaunay triangulation, but not figure (b).
- They have he same smallest angle, but the next to smallest is largest in (a)

The Voronoi diagram

Another definition of Delaunay triangulation

- The diagram to the left is the *Voronoi Diagram* of the given points
  - It is constructed as if the points are islands, and so that that the sea area closer to island x than to any other island belongs to island x. (“midtinje-prinsippet”) (“closest store”).

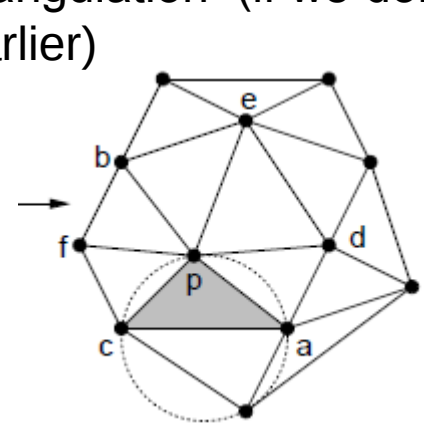


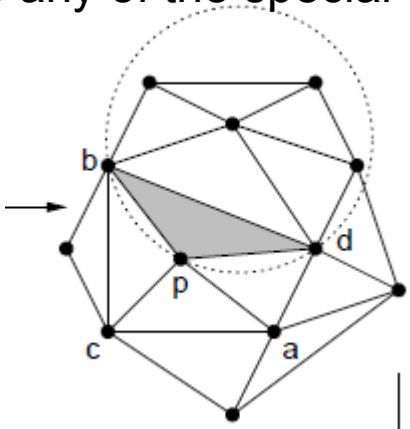


- The Delaunay-triangulation is then obtained by:
  - By drawing an edge between those points/islands that have a common border, you will get a Delaunay triangulation (and this edge will be orthogonal to the common border).
  - Note that these edges will not always pass through the common border (this is the case for the top two edges, and one at the bottom left).

A property equivalen to the two other definitions

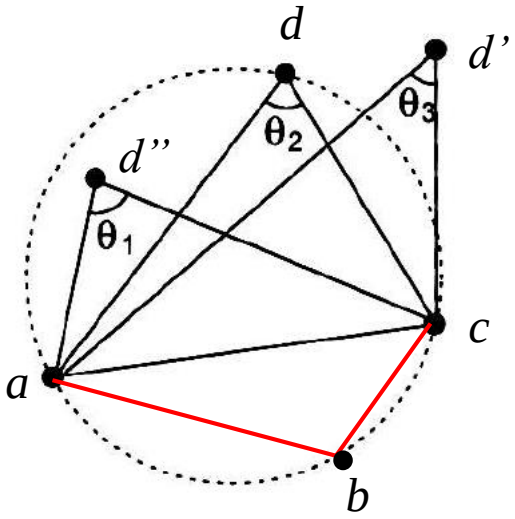
- A triangulation is a Delaunay triangulation if and only if
  for all triangles, the circle through the three corners does not contain (in its inside) any of the other points.
  - It is at the outset not clear that such a triangulation will always exist. However, it in fact does, and there is only one such triangulation (if we don't have any of the special cases mentioned earlier)





- The triangulation to the left is a Delaunay triangulation, but not the one to the right.
  - Note that we here have one of the special cases (four points on a circle), and then the Delaunay triangulation is not unique.
  - We could have replaced the edge c-a with the one from p and downwards.
- This is the definition we will use for the Delaunay triangulation.

Some facts about angles and circles

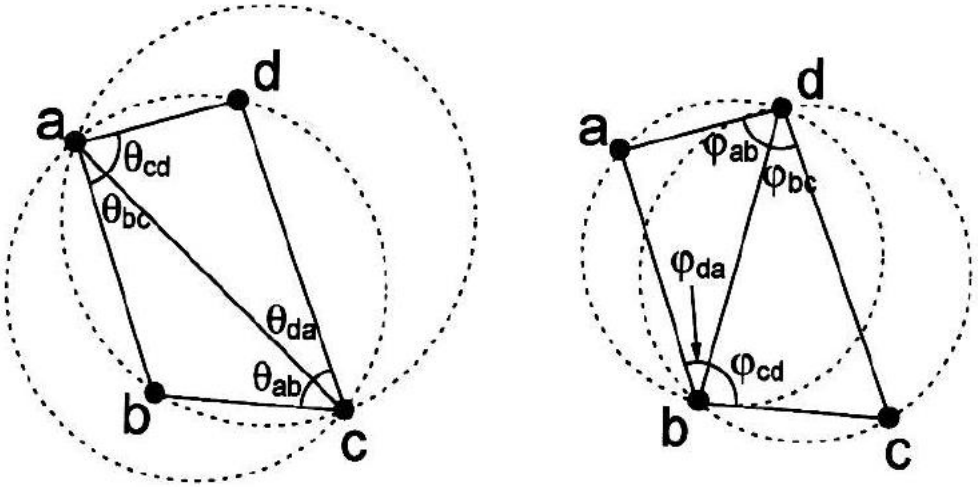


- To the left, we have: $\theta_1 > \theta_2 > \theta_3$.
- To decide whether the point *d* is inside, on, or outside the circle through *a*, *b* and *c*, we have to assure that *a*, *b*, *c* and *d* are taken in counter clockwise order, and then compute the value of the determinant:

$$\text{inCircle}(a,b,c,d) = \det \begin{pmatrix} a_x & a_y & a_x^2+a_y^2 & 1 \\ b_x & b_y & b_x^2+b_y^2 & 1 \\ c_x & c_y & c_x^2+c_y^2 & 1 \\ d_x & d_y & d_x^2+d_y^2 & 1 \end{pmatrix} > 0$$

(The computation of this can be optimized, and performed quite fast.)

The Delaunay trick



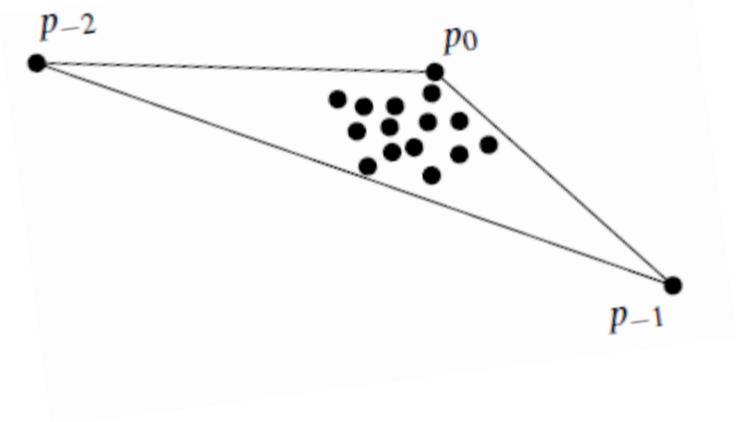
Assume the quadrangle *a-b-c-d* above is convex.

- We can observe that if *d* is inside the circle through *a, b,* and *c*, then the circle through *a, c,* and *d* will contain *b*.
  - The Delaunay requirement is not fulfilled.
- If we in (b) remove the edge *a-c* and instead insert *b-d* (see (c)), then point *a* will be outside circle through b, c and d, and c will be outside the one through a, b, and d.
  - Now the Delaunay requirement is fulfilled, at least locally.

An algorithm that finds the Delaunay triangulation

- We are given a set of points in the plane.
- **Start:** To have a triangulation to begin with, we add three points, so that the triangle defined by these points contains all the given points.
  - And we let this triangle be the initial triangle (see below).
  - This single triangle is a Delaunay triangulation of the three points.
- **Step:** We then do as we did earlier when we constructed a triangulation to count triangles and edges:
  - That is, we add one node at a time, and for each node we also add the three edges to the corners of the triangle where the node resides.
  - This may locally destroy the Delaunay property, and before we add the next node we will restore the Delaunay property, when necessary. (see next slide)

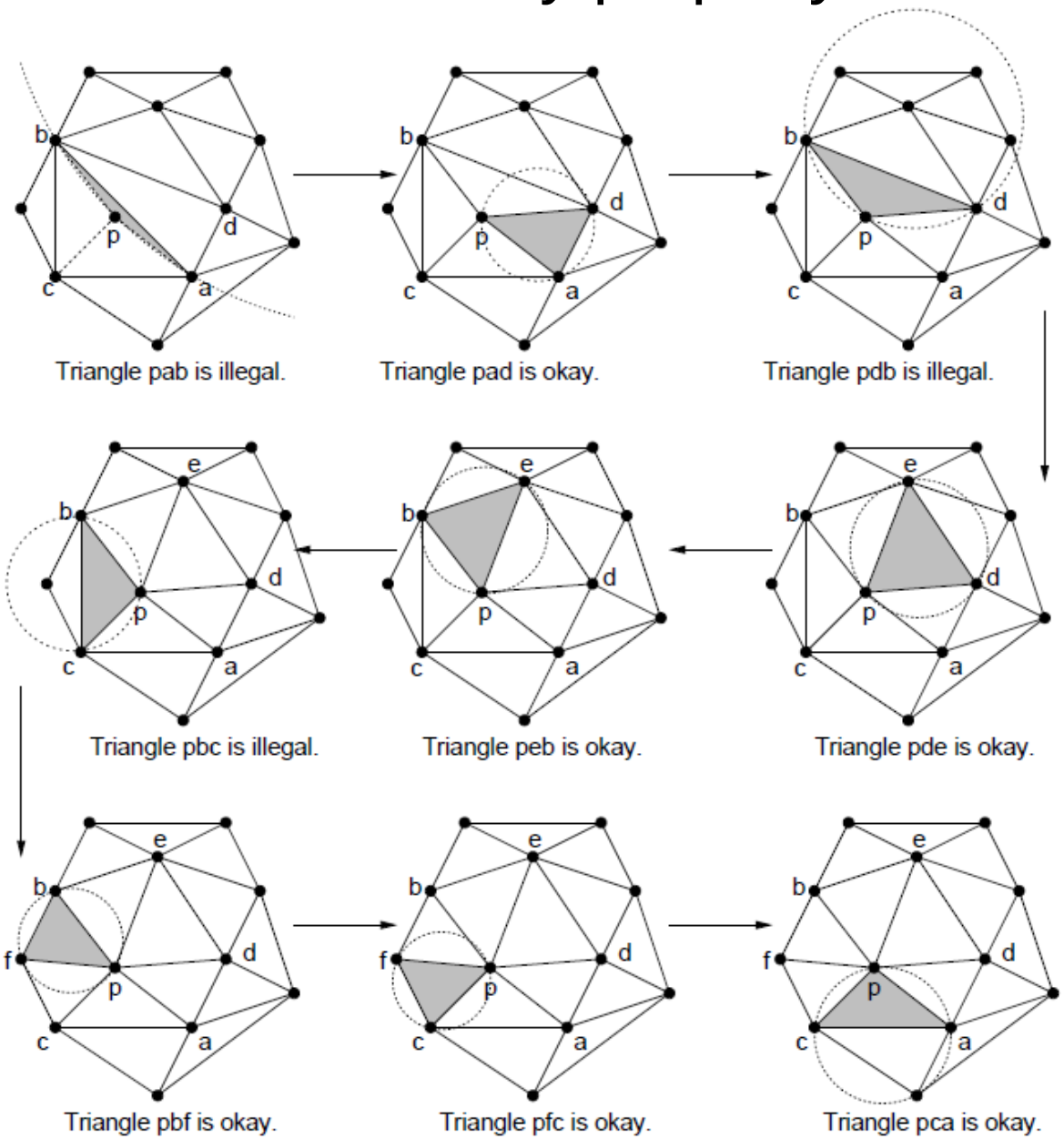
It is important here to choose two of the three starting nodes far away from the node set. Then it will be easier to remove them afterwards. The third point can in fact be one of the original points.



Restoring the Delaunay property

- Adding a point (see previous slide):
  - Before the addition we have a Delaunay triangulation (invariant).
  - We add a point *p* in one of the triangles, and draw edges from *p* to each of the three corners of this triangle.
  - This may destroy the Delaunay property, and we want to restore it by using the Delaunay trick (see earlier slide).
- An important fact that we don't prove:
  - To restore the Delaunay property we look at all triangles that have a corner in *p*. We look at each of these triangles together with the neighboring triangle opposite to *p*, and check whether the Delaunay property holds for these two triangles together. If not, we use the Delaunay trick on those two triangles (see figure next slide).
  - While this checking and repair goes on, the set of triangles with a corner in *p* can increase, and we have to go on testing until we have gone a full round without any Delaunay property being broken.
  - This process will always stop, as the number of edges to *p* will increase each time we use the Delaunay trick, and there is only a finite number of points that *p* can have edges to.

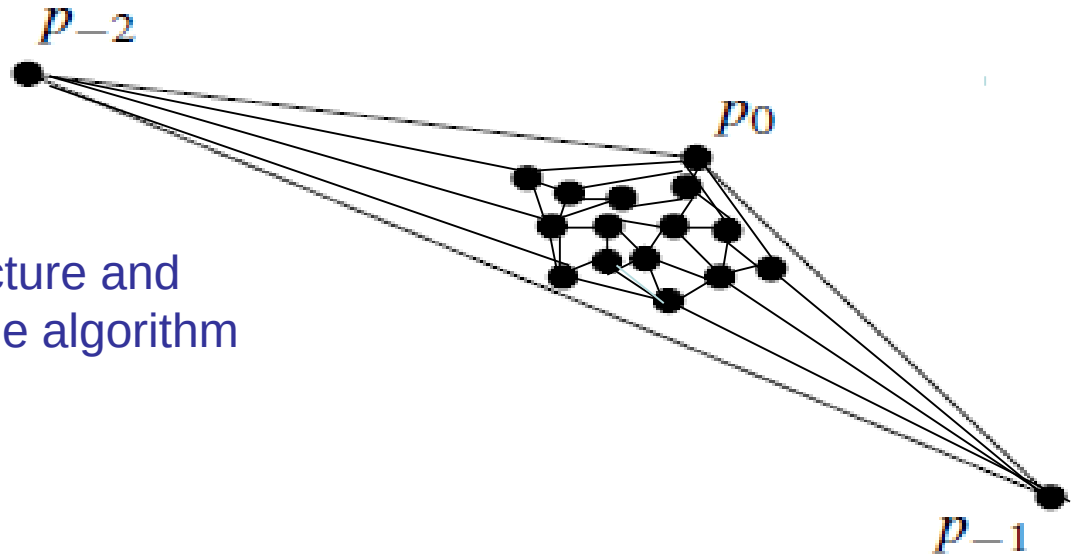
Adding a new node, and restoring the Delaunay property



Starting and ending the algorithm

As we have described the algorithm, we start the algorithm by adding three extra points, so that the triangle spanned by these contains all the given ("real") points.

- This triangle makes up our initial Delaunay triangulation.
- We then perform the algorithm as explained above.
- Thus, the last problem is to "get rid of" the extra points we started with.
- The trick here is to place at least two of the extra nodes far away from the given node set. Then we can simply remove the extra points, and the edges to them. This works because the convex hull of the original points will all be edges in the triangulation (not proven here).



With a good data structure and some optimizations, the algorithm will run in time:

$O(n \log n)$

Convex hull (8.6.2)

Chapter 8 is generally about the **divide-and-conquer**-method:

- Split the problem into smaller problems of the same kind.
- Solve each of the smaller problems, usually by further splitting these problems.
- Find the solution of the larger problem by combining the solutions to the smaller problems.

Divide-and-conquer is used in many algorithms, e.g. in QuickSort.

We shall use it for finding the convex hull of a set of points.

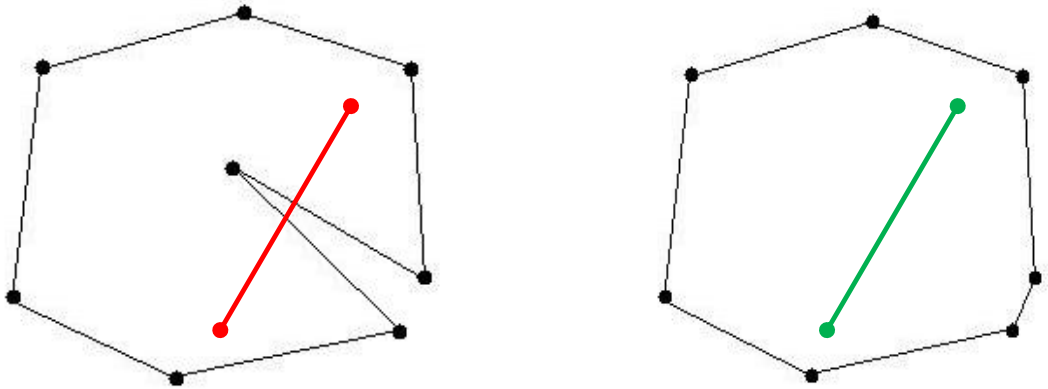
(There are also algorithms for triangulation that uses divide-and-conquer.)

Convex hull: Norwegians could wonder what type of “hull” (= hole) this is. (Some even use the phrase “Det konvekse hullet”…….)
However, it is not the type to the left, but the English version to the right.
The correct Norwegian phrase is *”Den konvekse innhylling/omslutning”.*

Convex hull (same slide as earlier)

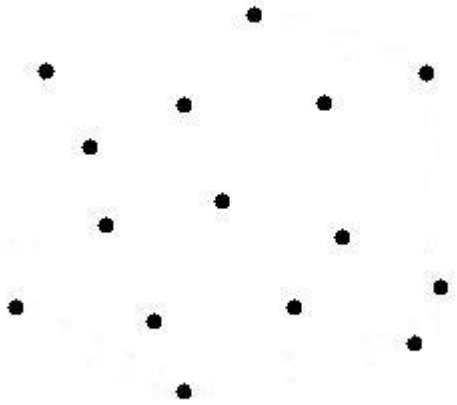
Let P be a set of points in the plane ($\mathbf{R}^2$) (Can also be defined for $\mathbf{R}^k$, $k > 2$)

Definition: A set $Q \in \mathbf{R}^2$ is *convex* if:
for all $q_1, q_2 \in Q$ the line q_1q_2 is fully within Q.

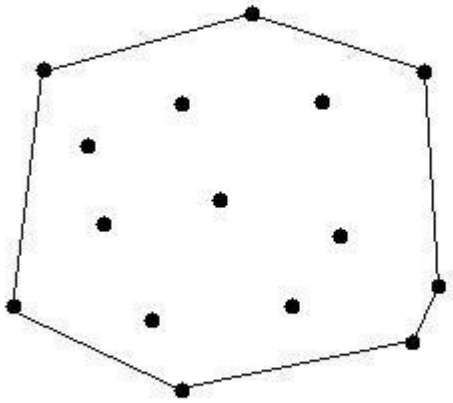


Definition: The *convex hull* of a set of points $P \in \mathbf{R}^2$ is the smallest convex set Q that contains all the points of P.

At the start: A set of points in the plane

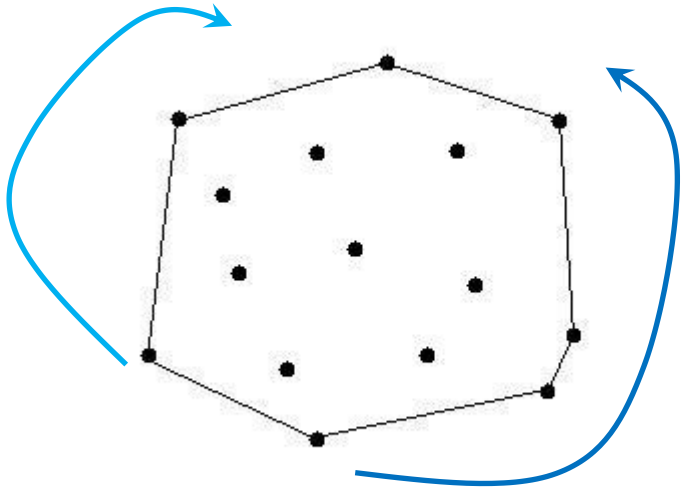


Convex hull of a set of points



Intuition: The points are pegs, and we put a rubber band around all the pegs.

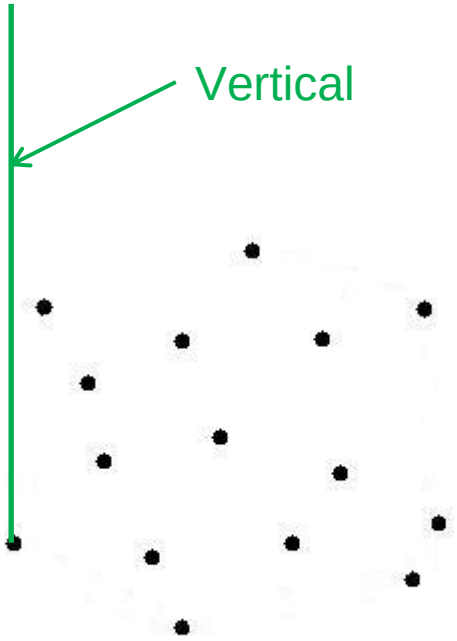
The answer from an algorithm should be given as a sequence of points (x, y) in order around the set of points.



The starting point and the direction may vary.

Counterclockwise most common, an arbitrary choice, but corresponds with the notion of a positively oriented curve in mathematics.

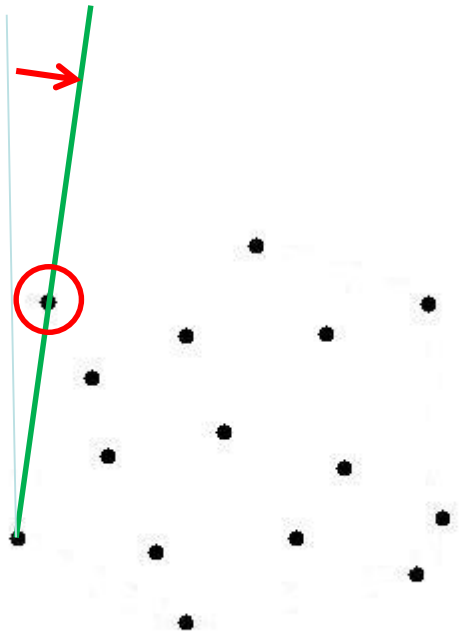
Algorithm Jarvis´ March



Idea: Use a thin rope and wrap it around the points, step by step.

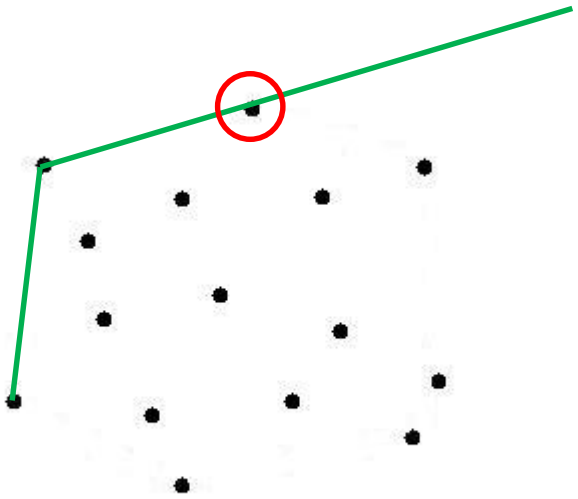
To get a staring point, choose e.g. the point with the smallest x-value. This is always a corner of the convex hull

First step



Swing the rope to the right, and stop as soon as it meets a point. This is the next corner of the convex hull.

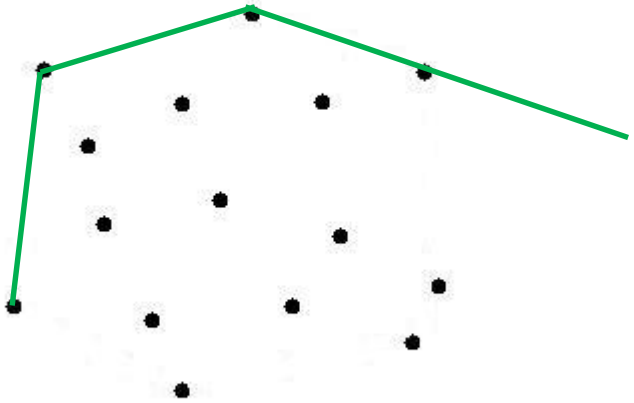
Further:



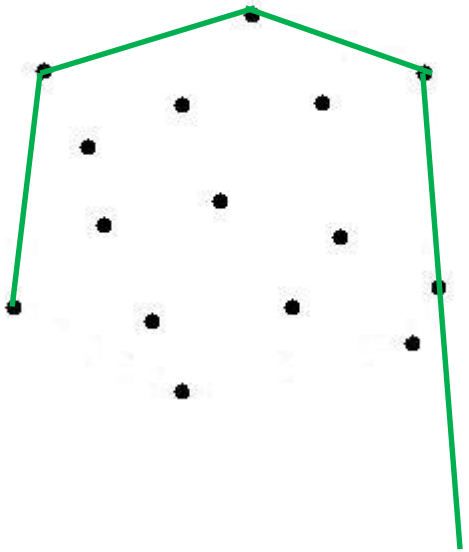
Swing the rope further to the right, always around the last identified corner, until you touch a new point.

In each step we have to find the point that has the smallest angle with the previous edge.

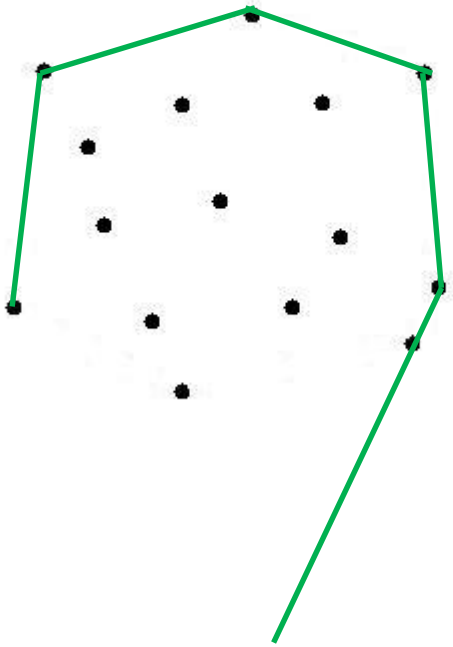
Next step:



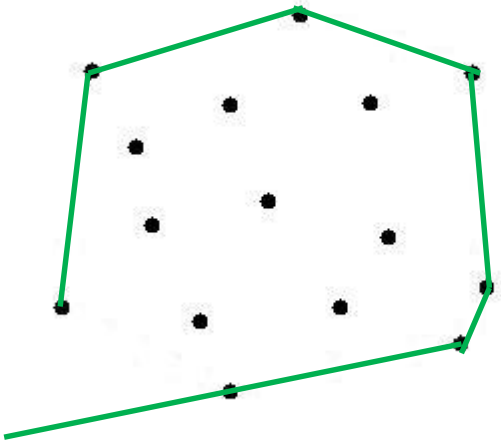
And next:



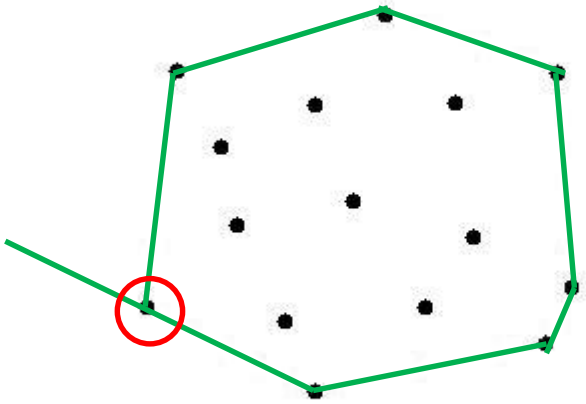
And next:



And next:

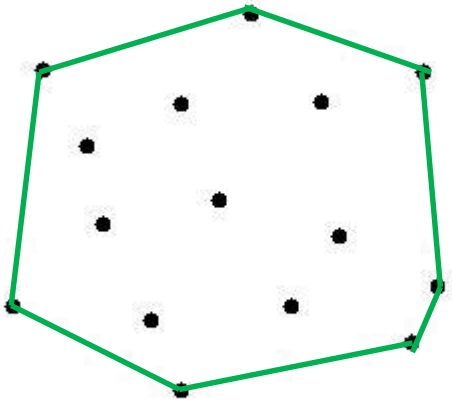


Termination:



When the starting point becomes the next corner, we are finished.

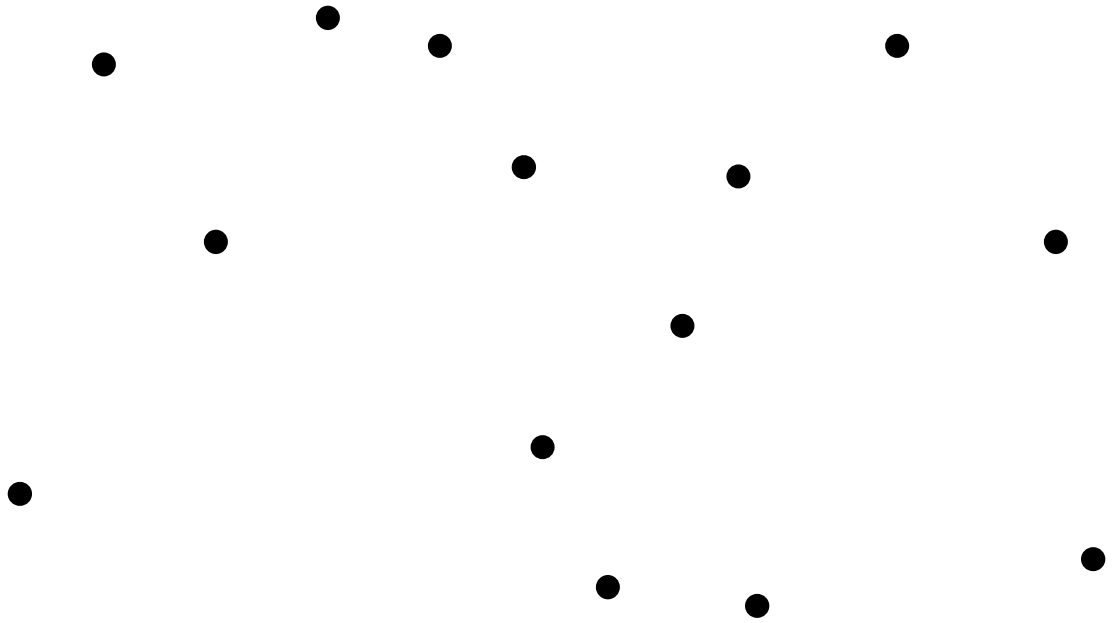
And we have the convex hull:



- In two dimensions worst case running time is: $O(n^2)$
- In d dimensions the running time is: $O(n^{\lfloor (d/2) \rfloor + 1)})$

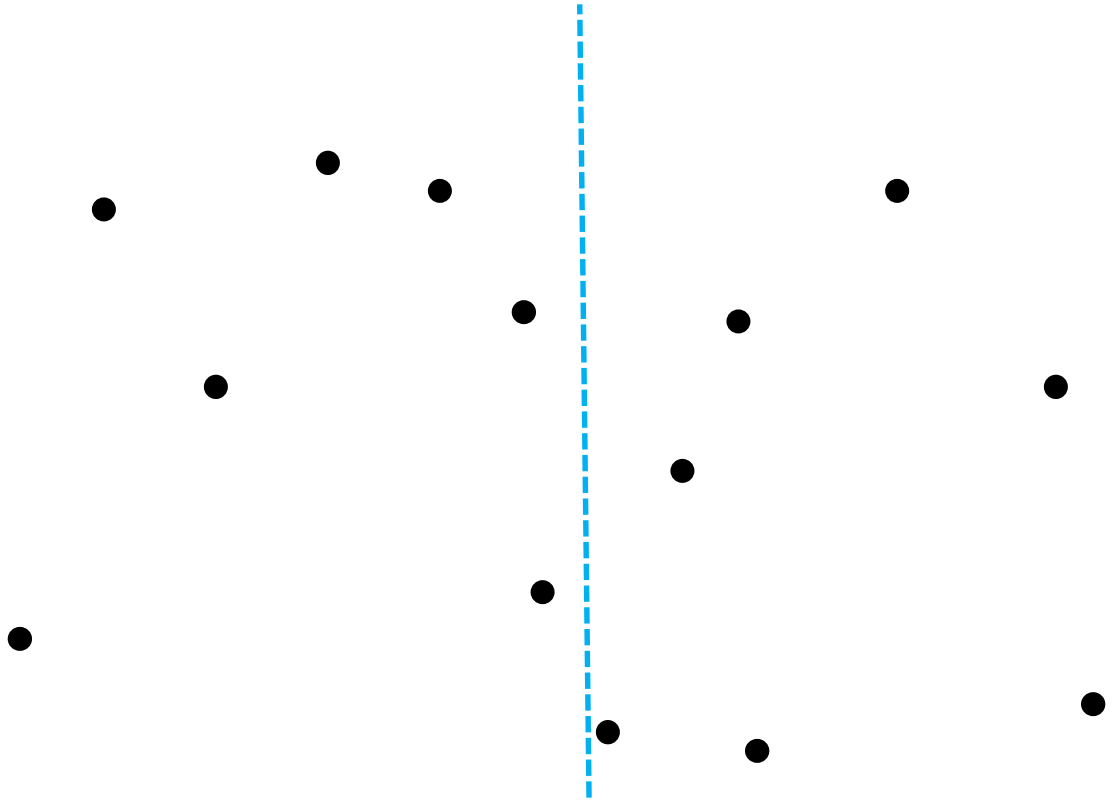
A faster method for finding the convex hull

using divide-and-conquer



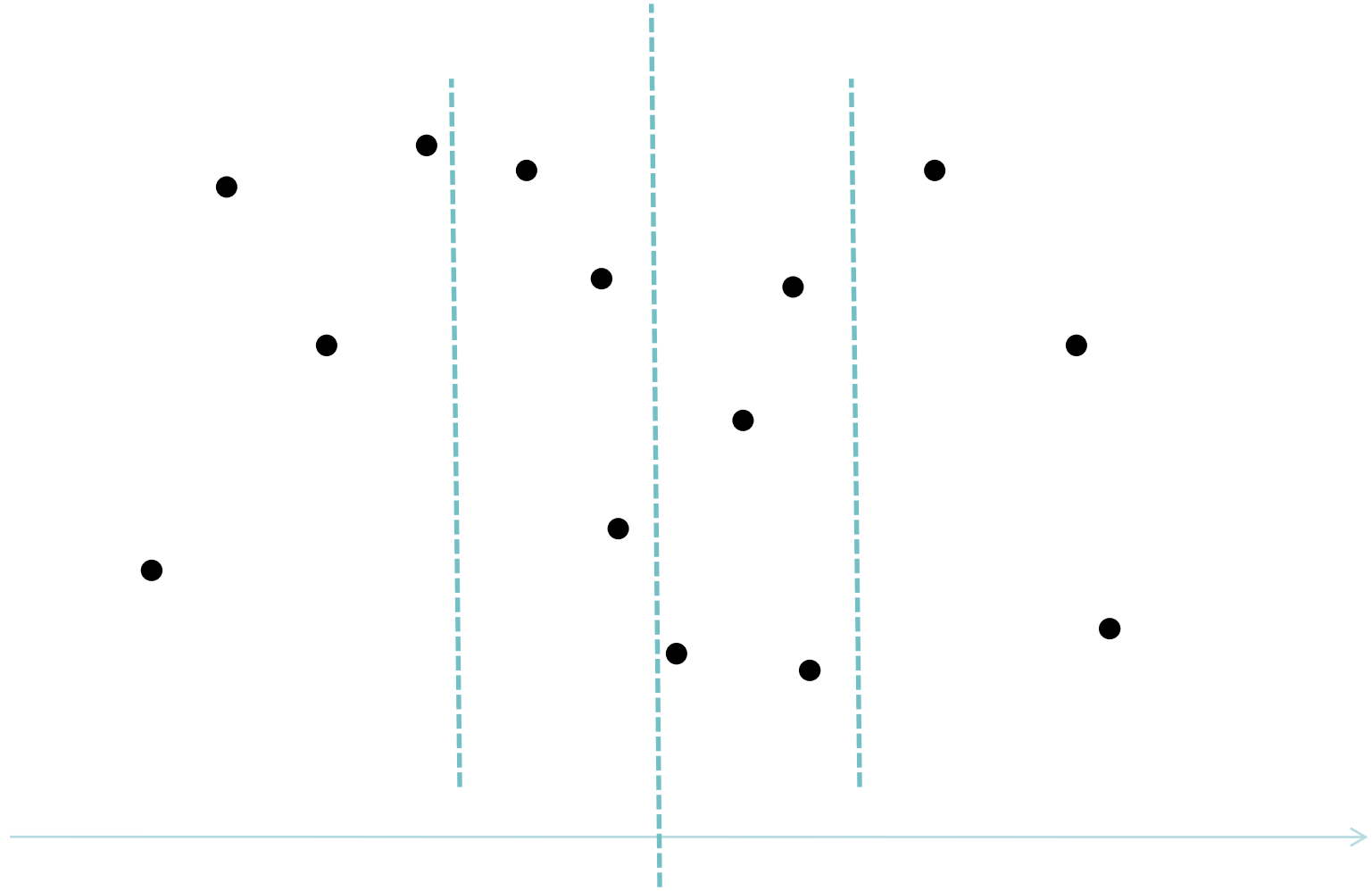
The first step is to divide the set into two sets with the same number of nodes (+1 or -1 for odd numbers), using a vertical line.

Convex hull



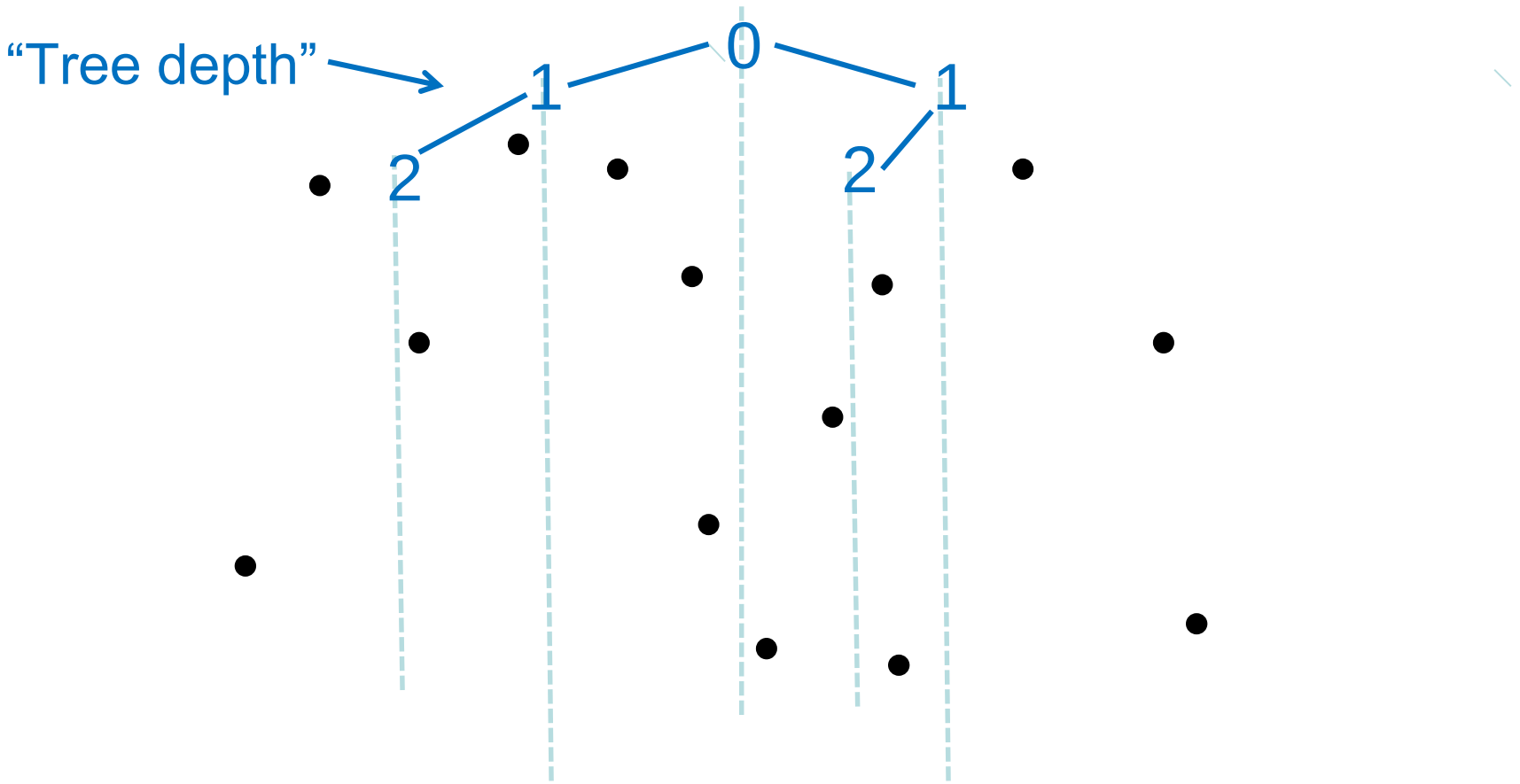
- Divide the set at the median of the *x*-values.
- Repeat this for each of the sets until you have 1, 2, or 3 points in each set.

Convex hull



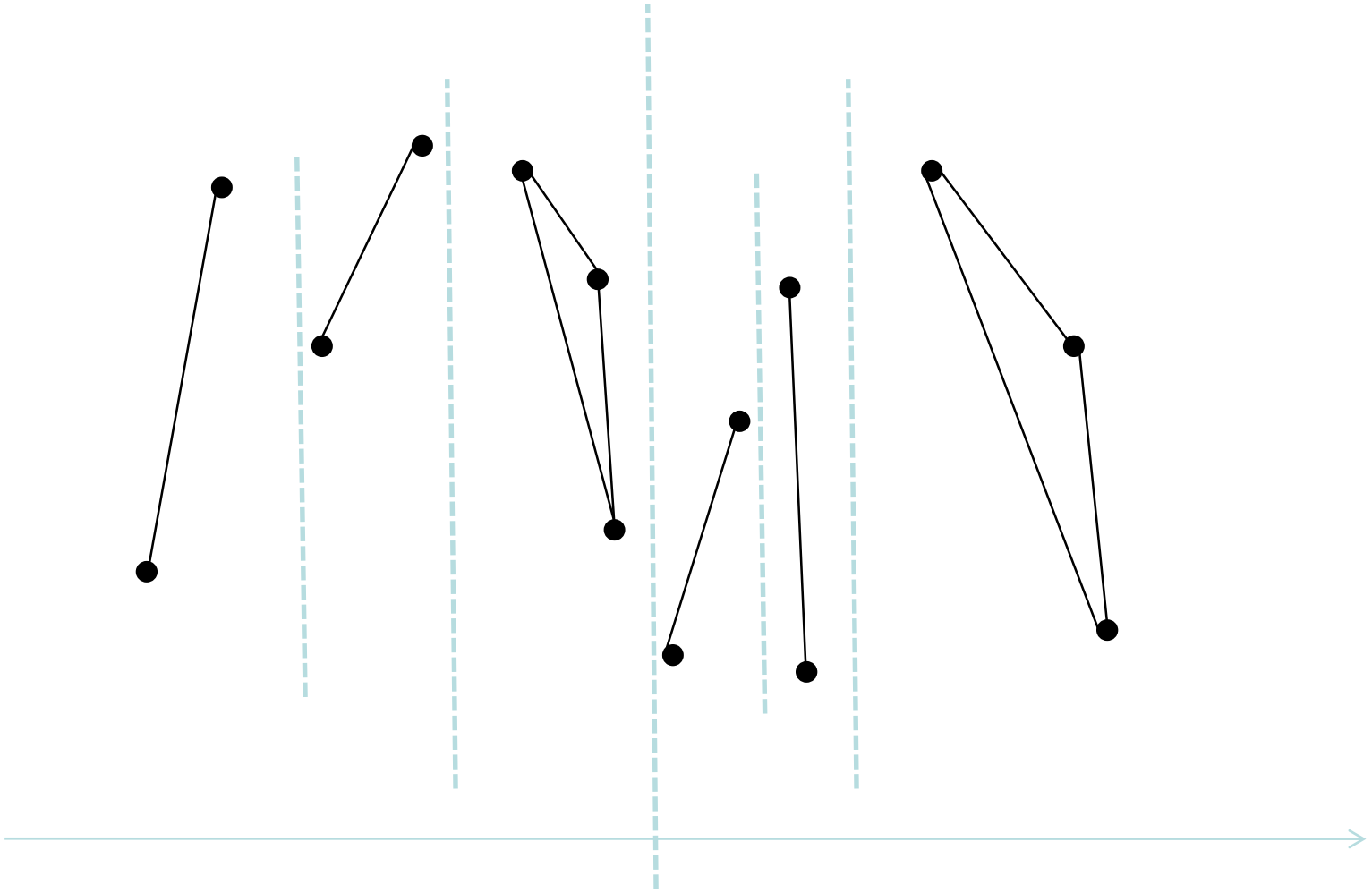
Then divide each of the smaller sets in the same way

Convex hull



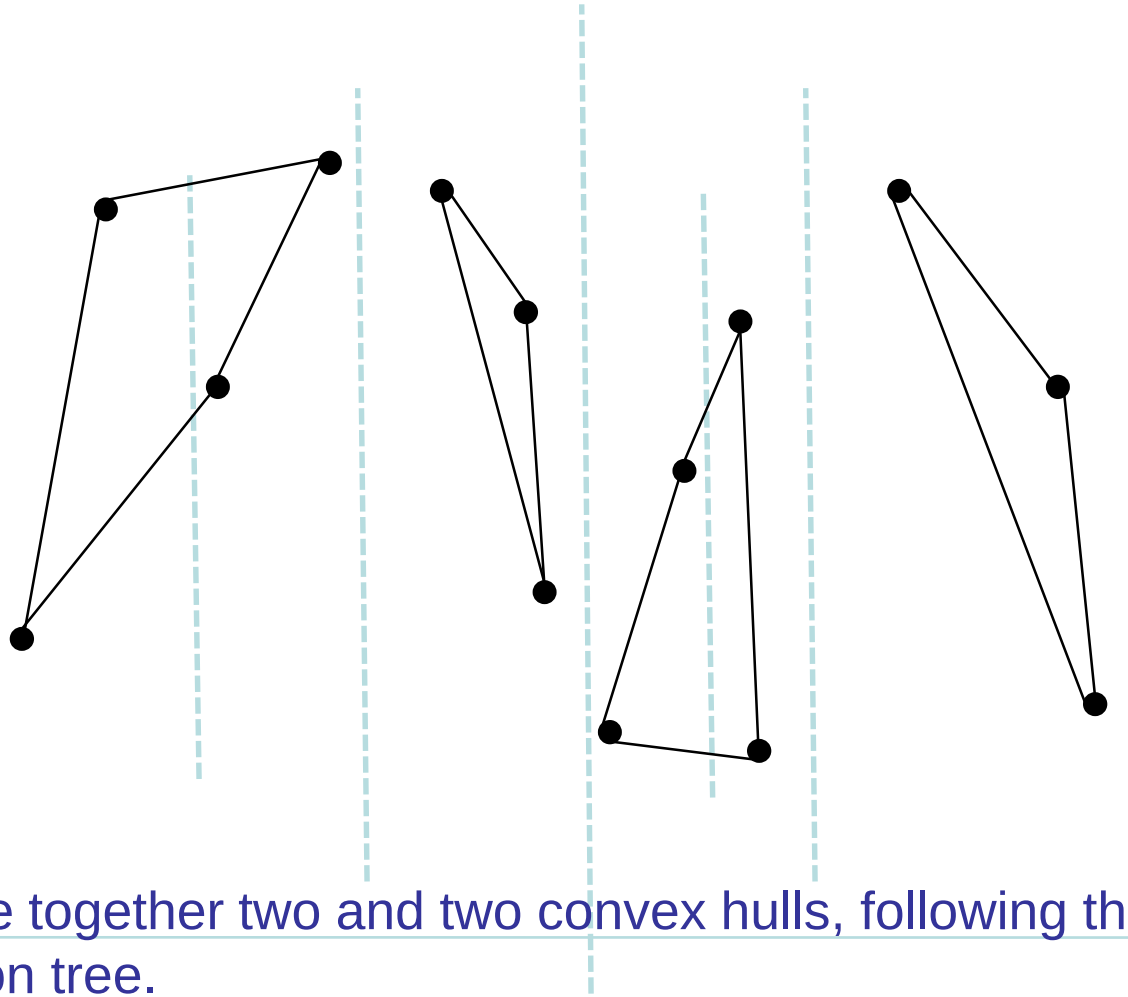
The depth of the "division tree" will be no larger than $\log_2 n$.

Convex hull



Solve lowest level: Find the convex hull for 2 or 3 points (easy).

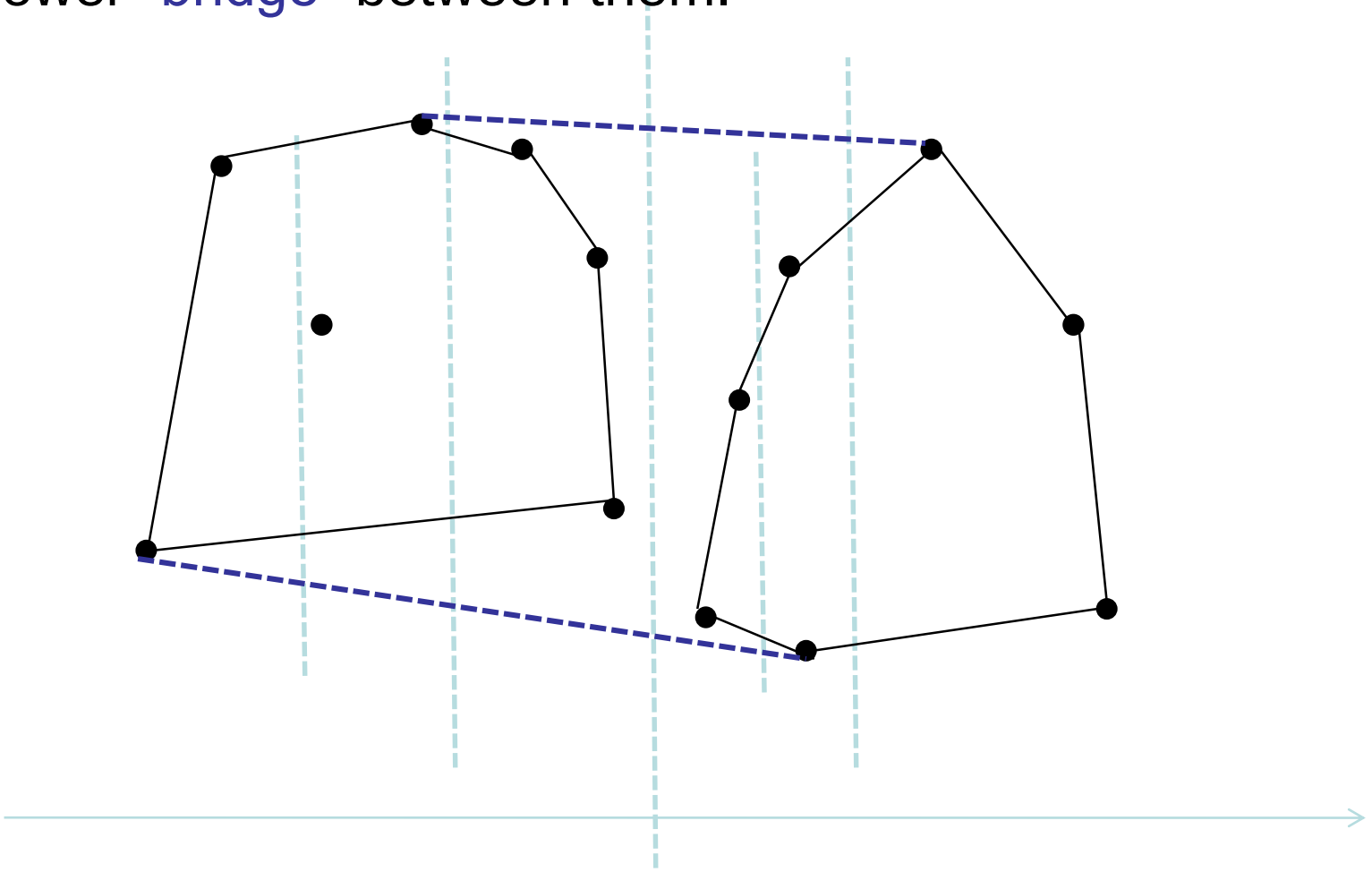
Convex hull



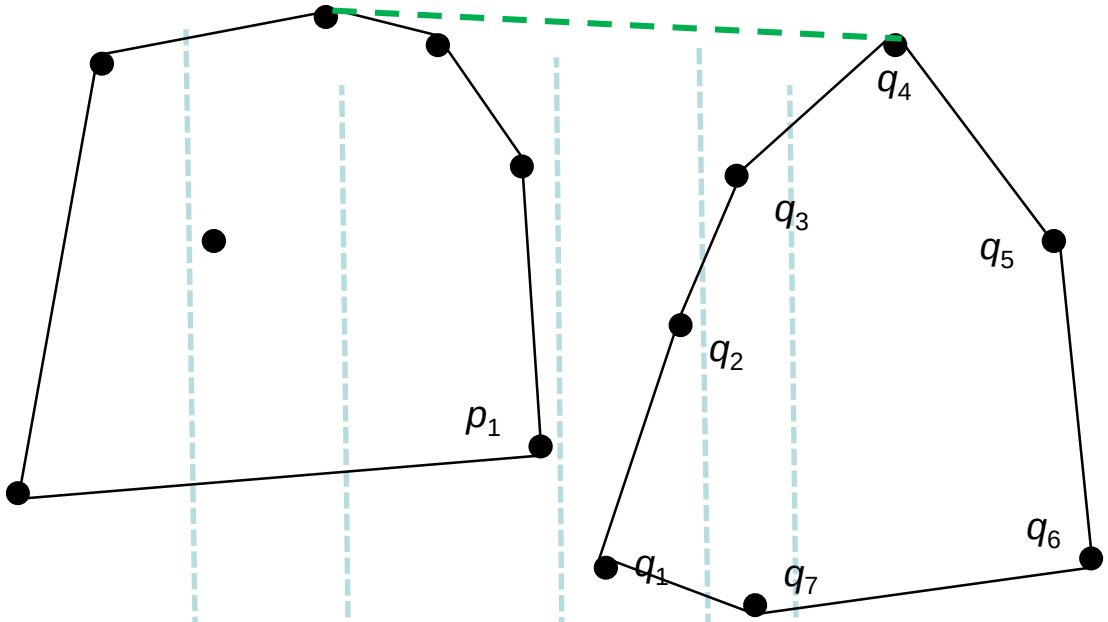
We merge together two and two convex hulls, following the structure of the division tree.

Convex hull

We merge two hulls into one by finding the upper and lower “bridge” between them.



Finding the upper bridge

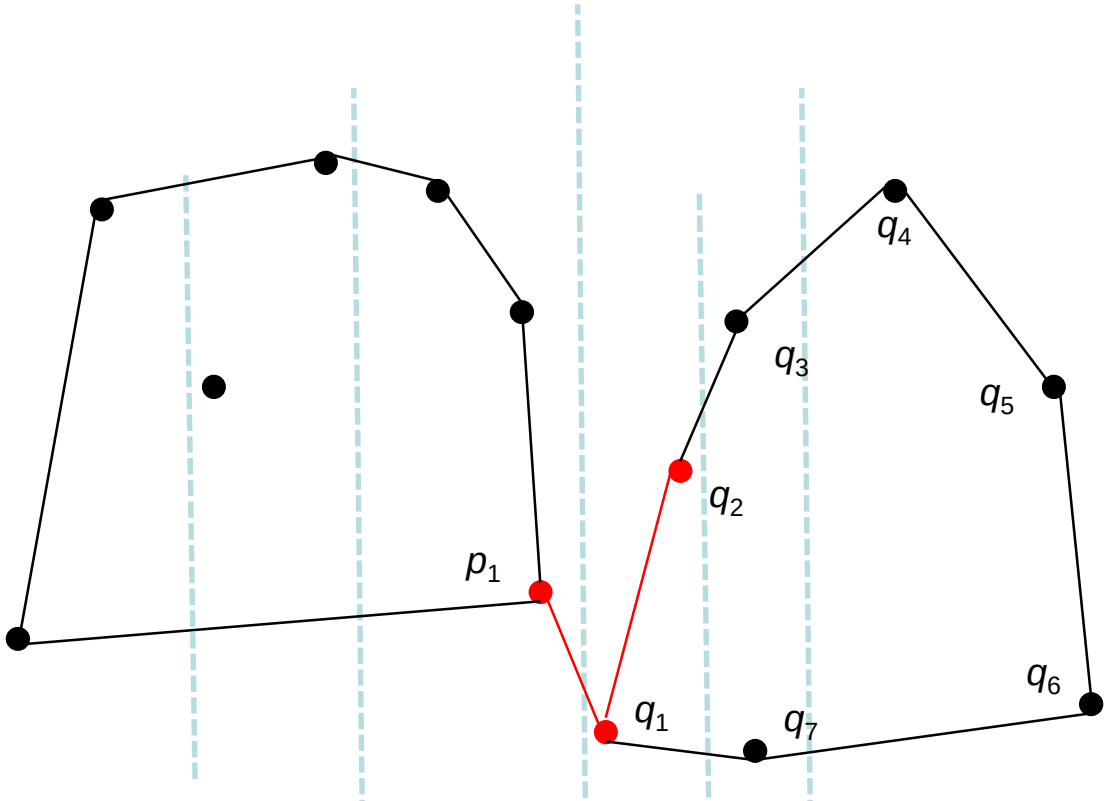


We know that all x-values in the left set are smaller than those in the right set. Let p_1 be the rightmost corner of the left hull and q_1 the leftmost of the right

Number the corners of the left hull counterclockwise p_1, p_2, …, and the corners of the right hull clockwise q_1, q_2, …

NB: This is different from what is done in the textbook! Inaccurate.

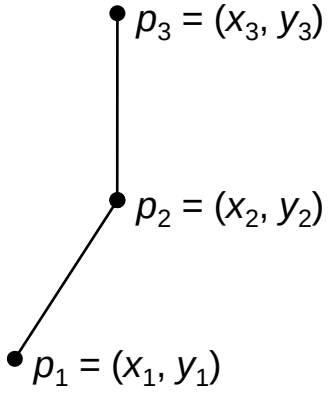
Finding the upper bridge



- We start by ”crossing” the edge p_1-q_1, and moving to the next corner, q_2, of the right convex hull (as is done above)
- To check if this was the upper bridge, we have to be able to decide whether three consecutive points represent a turn to the left or to the right. See next slides.

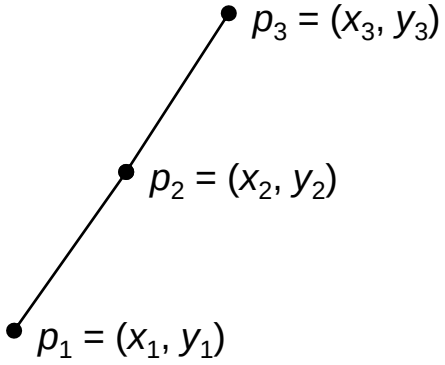
How can we find whether three consecutive points represent a turn to the left or to the right.

We assume that the three points are $p_1=(x_1, y_1)$, $p_2=(x_2, y_2)$, $p_3=(x_3, y_3)$



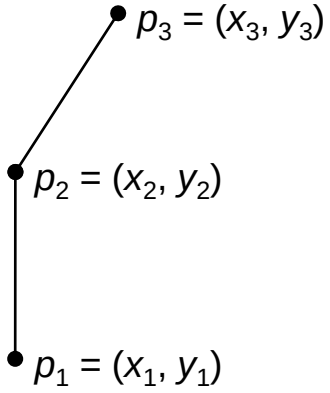
Turning left

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} > 0$$



Straight ahead

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = 0$$



Turning right

$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} < 0$$

This is a *determinant.* Computing is is described on next slide.

How to compute 3 x 3 determinants?

Given the matrix *A*:

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

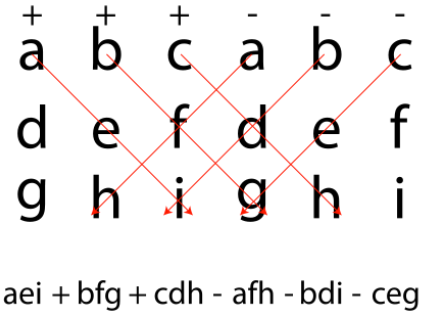
$$\det(A) = a\begin{vmatrix} e & f \\ h & i \end{vmatrix} - b\begin{vmatrix} d & f \\ g & i \end{vmatrix} + c\begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= aei - afh - bdi + bfg + cdh - ceg$$

Since c, f and i are all 1, we get the formula:

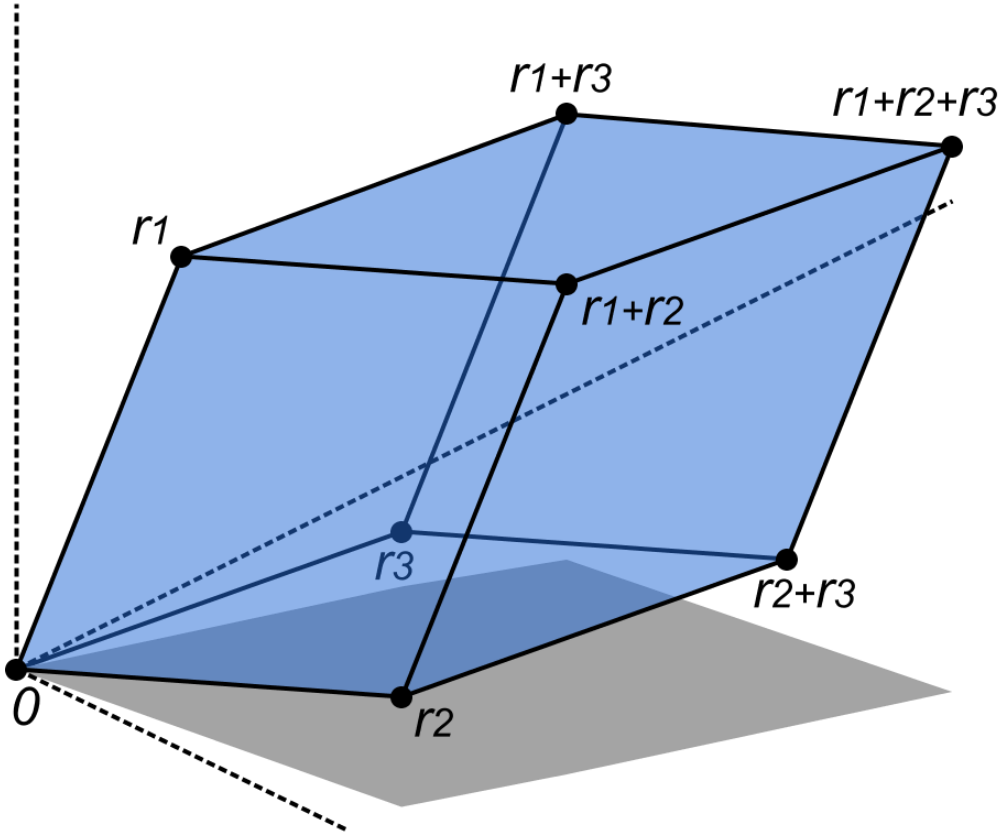
ae – ah – bd + bg + dh – eg

A scheme for computing a 3x3 determinant



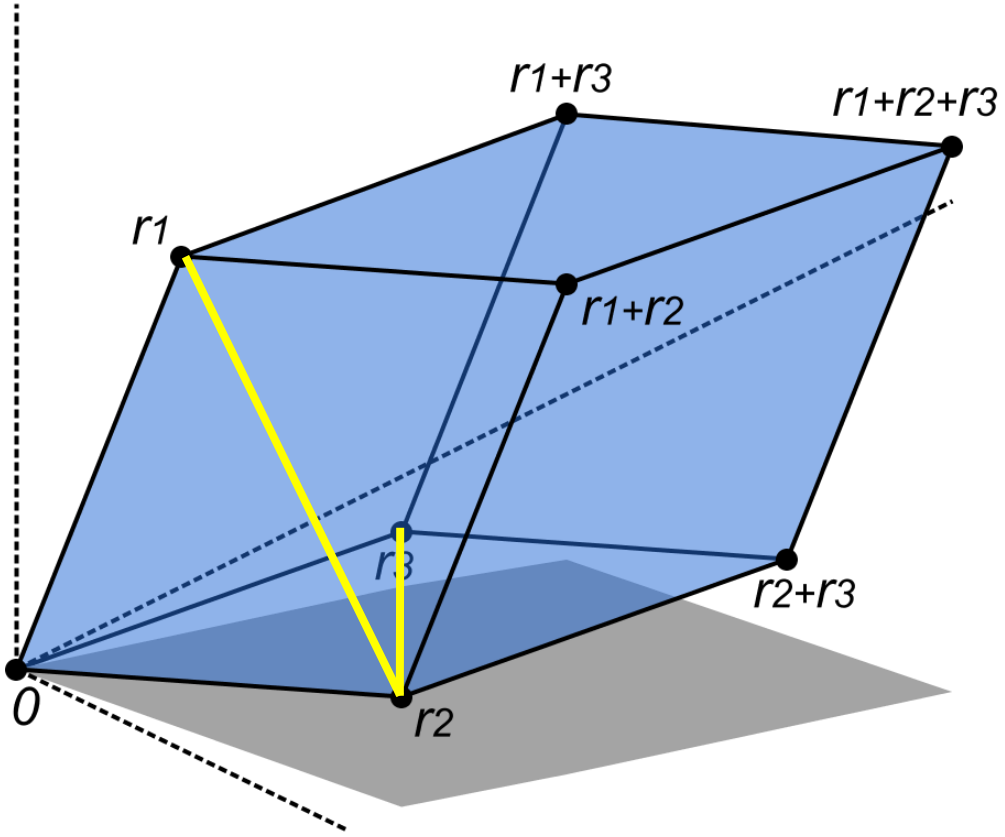
Determinants

$$A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$



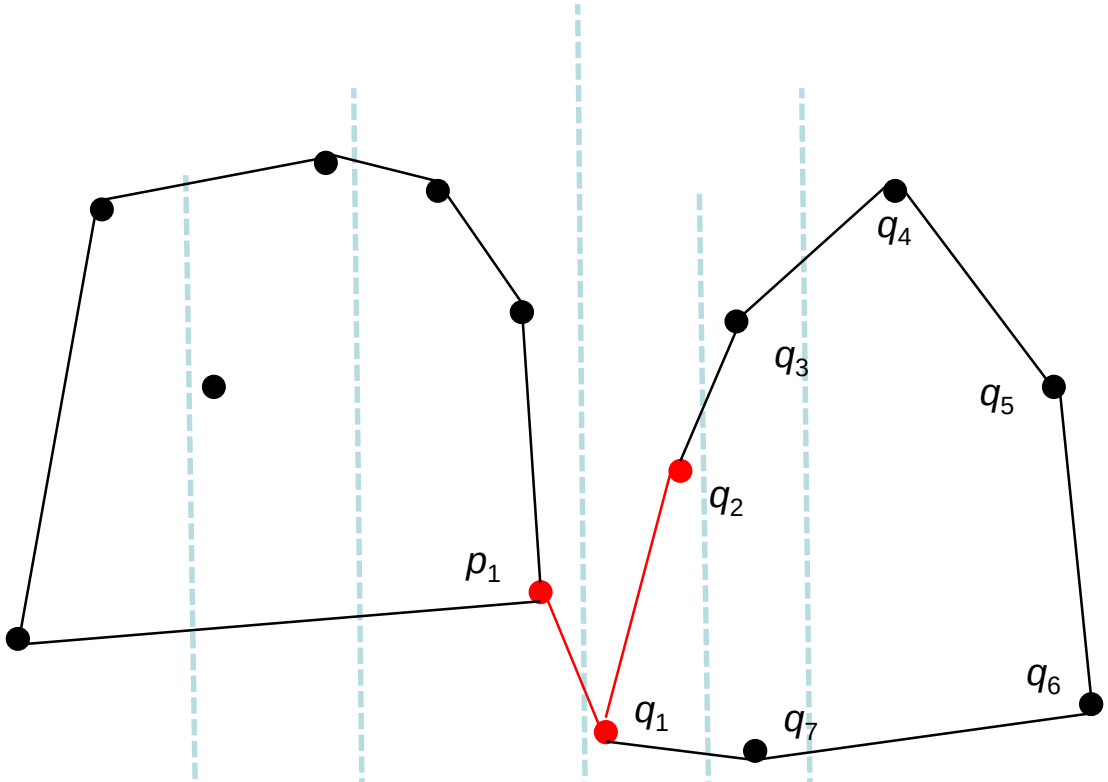
The geometric interpretation of the determinant of matrix *A* (det(*A*), or simply |*A*|) is the volume of the parallelepiped spanned by the row vectors of *A*. (Or the column vectors, it is the same volume).

Determinants



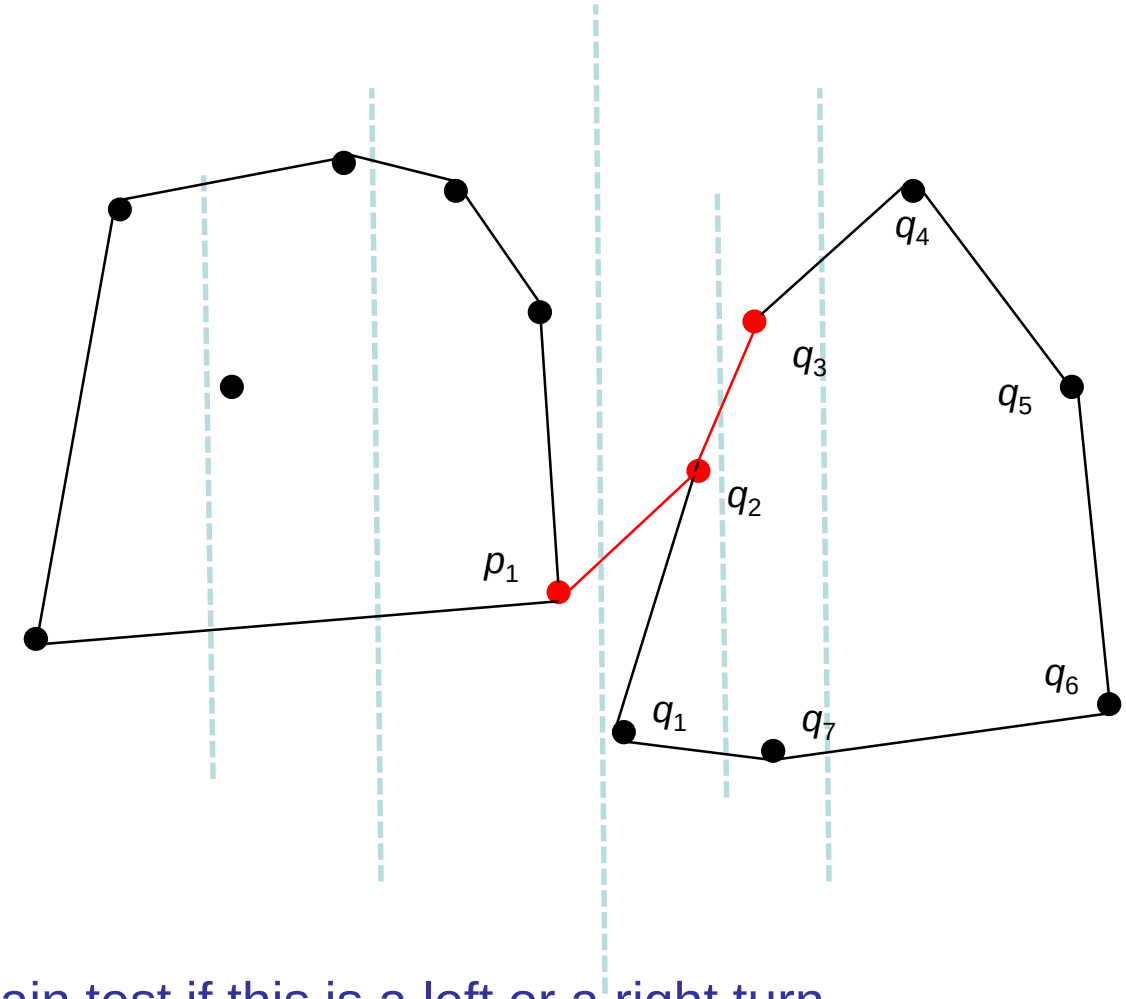
Finding the upper bridge

same figure as earlier



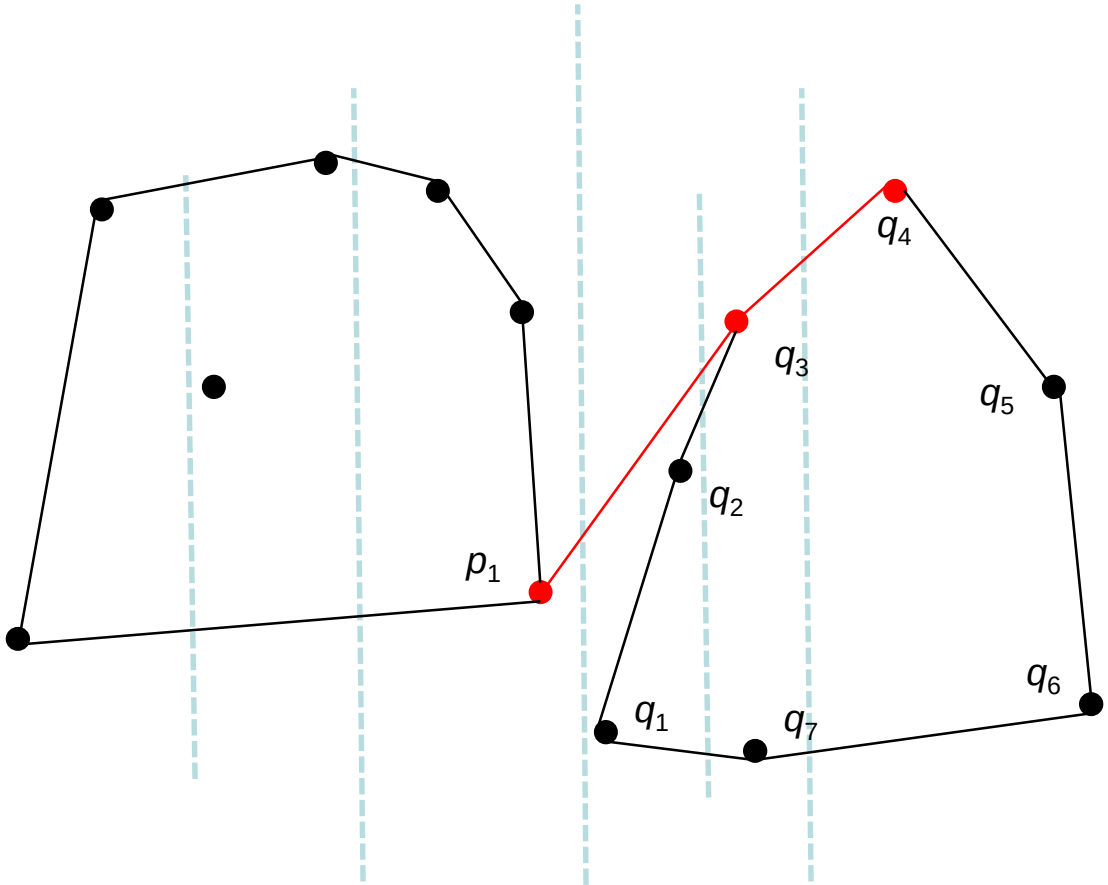
- We start by “crossing” the edge p_1-q_1, and moving to the next corner, q_2, of the right convex hull (as is done above)
- We then use the determinant to decide whether p_1-q_1-q_2 is a turn to the right or to the left.
- If it turns to the left (as above), we move one step to the tipple p_1-q_2-q_3 (otherwise we ”change side”, see later).

Finding the upper bridge



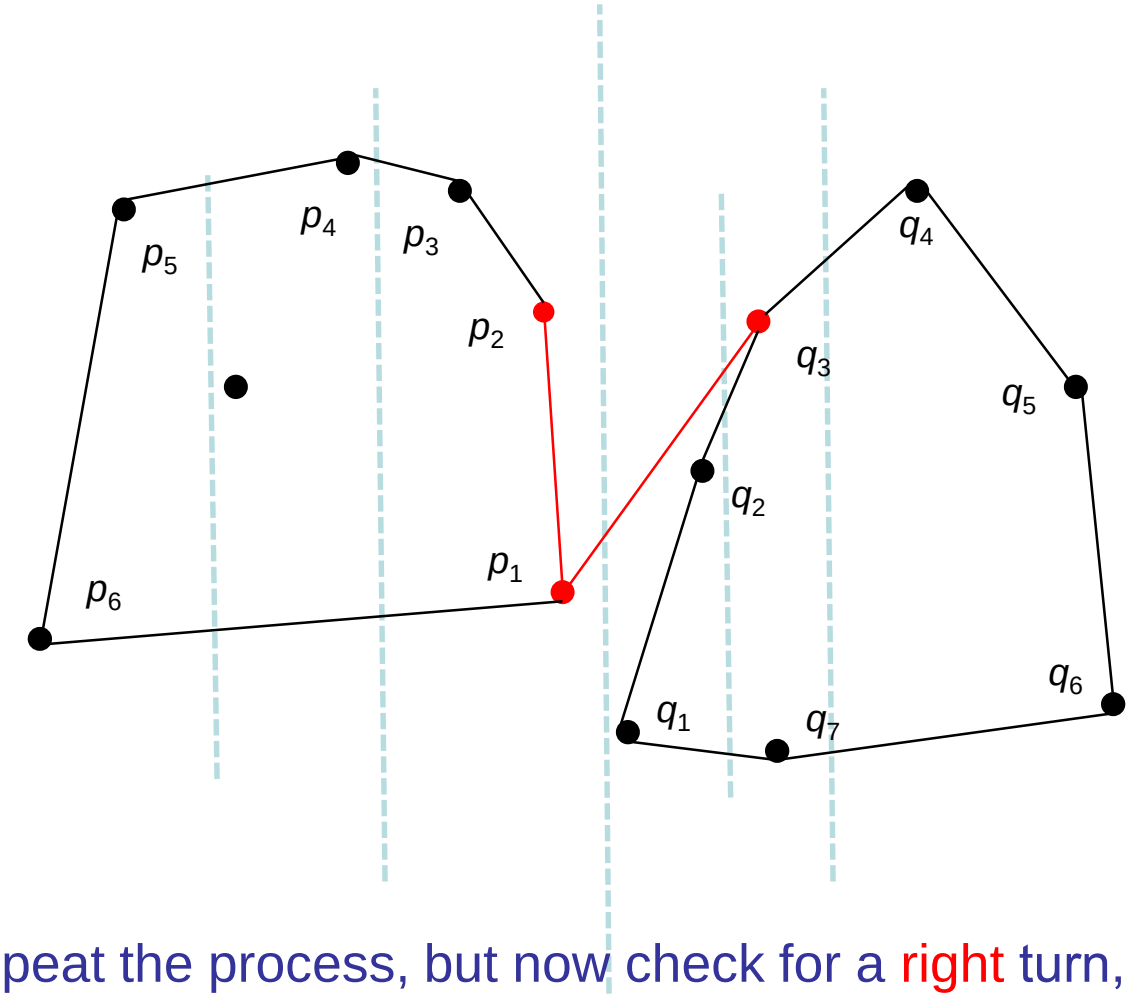
- We again test if this is a left or a right turn.
- As it too is a left turn, we go one step further along the right hull.

Finding the upper bridge



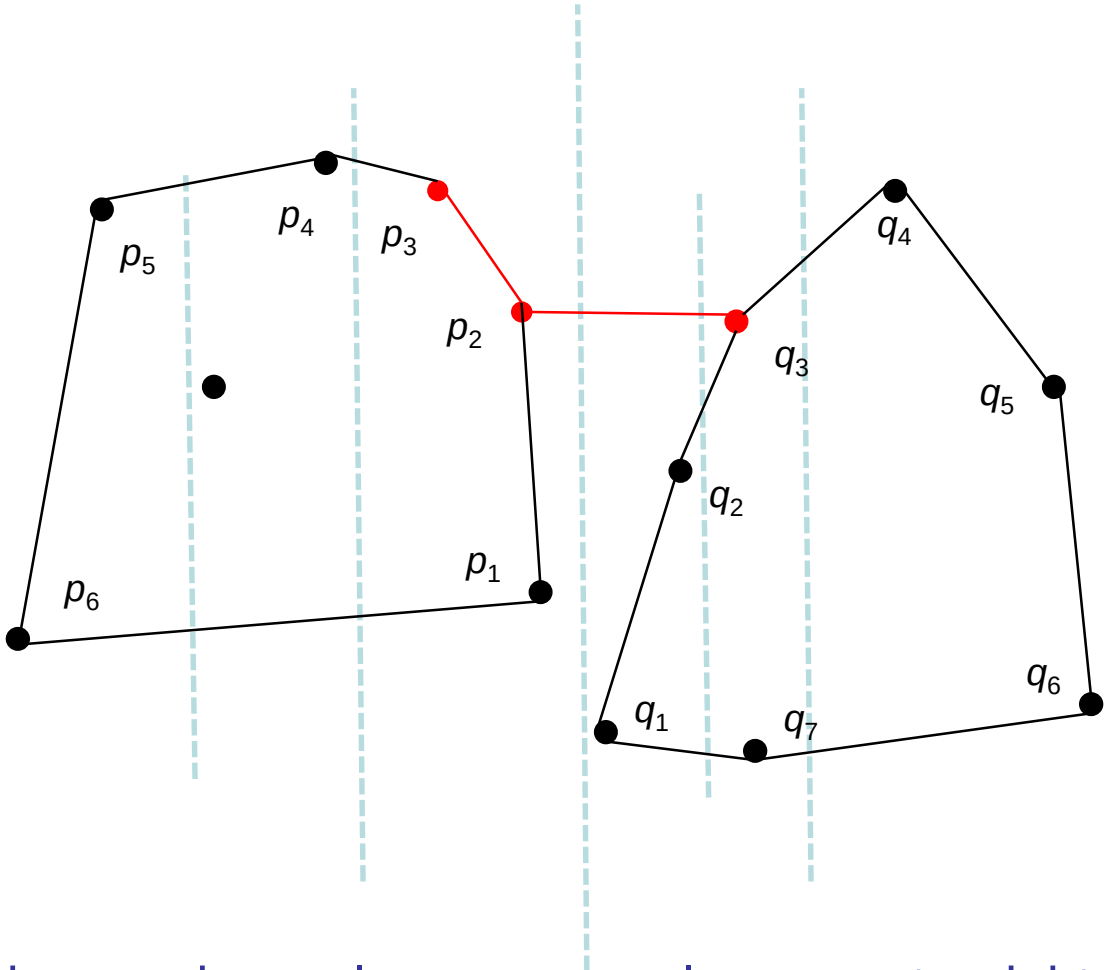
- Now p_1-q_3-q_4 is a turn to the right, and we "move to the other side". We have "crossed over" from p_1 to the right hull and followed the corners to end up in q_3, now we "cross back" from q_3 to p_1 and follow the corners of the left hull.

Finding the upper bridge



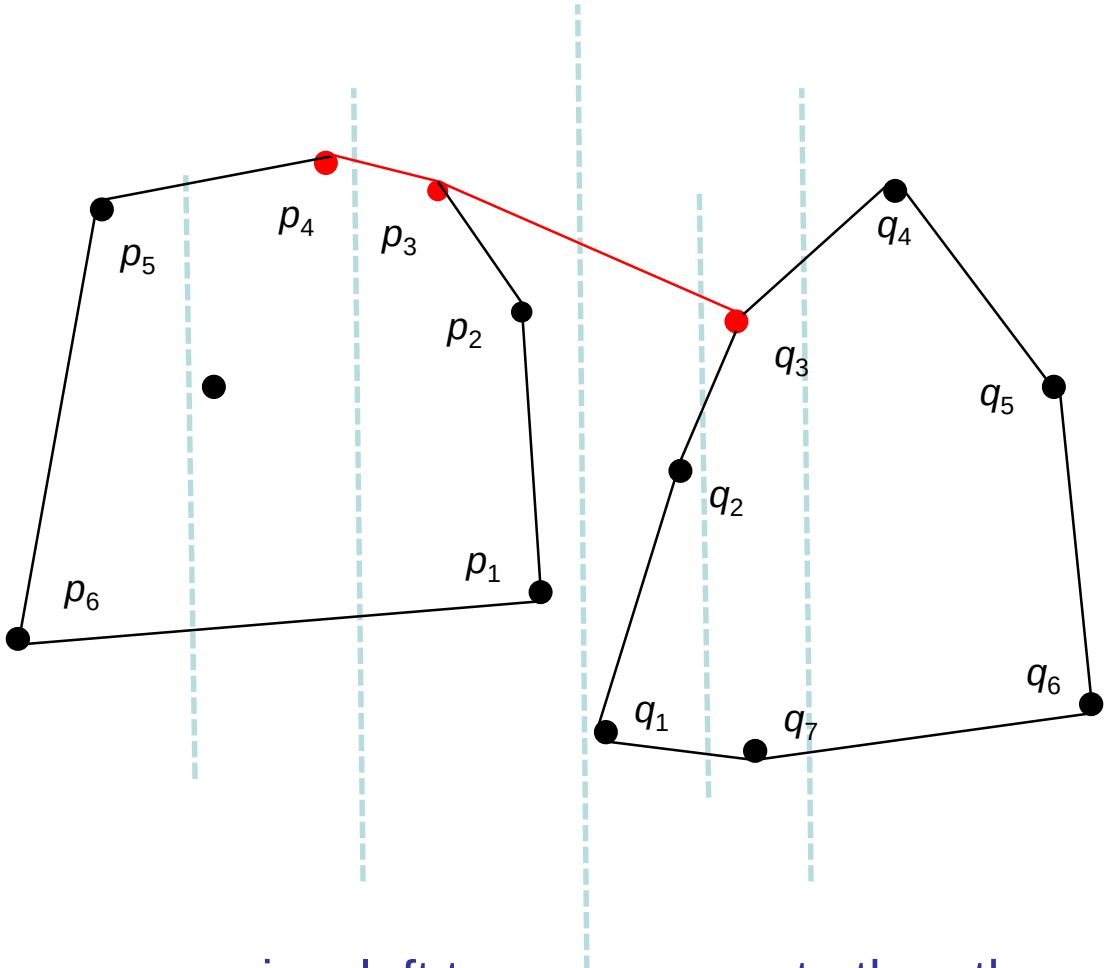
- We repeat the process, but now check for a right turn, and “move to the other side” if we encounter a left turn.

Finding the upper bridge



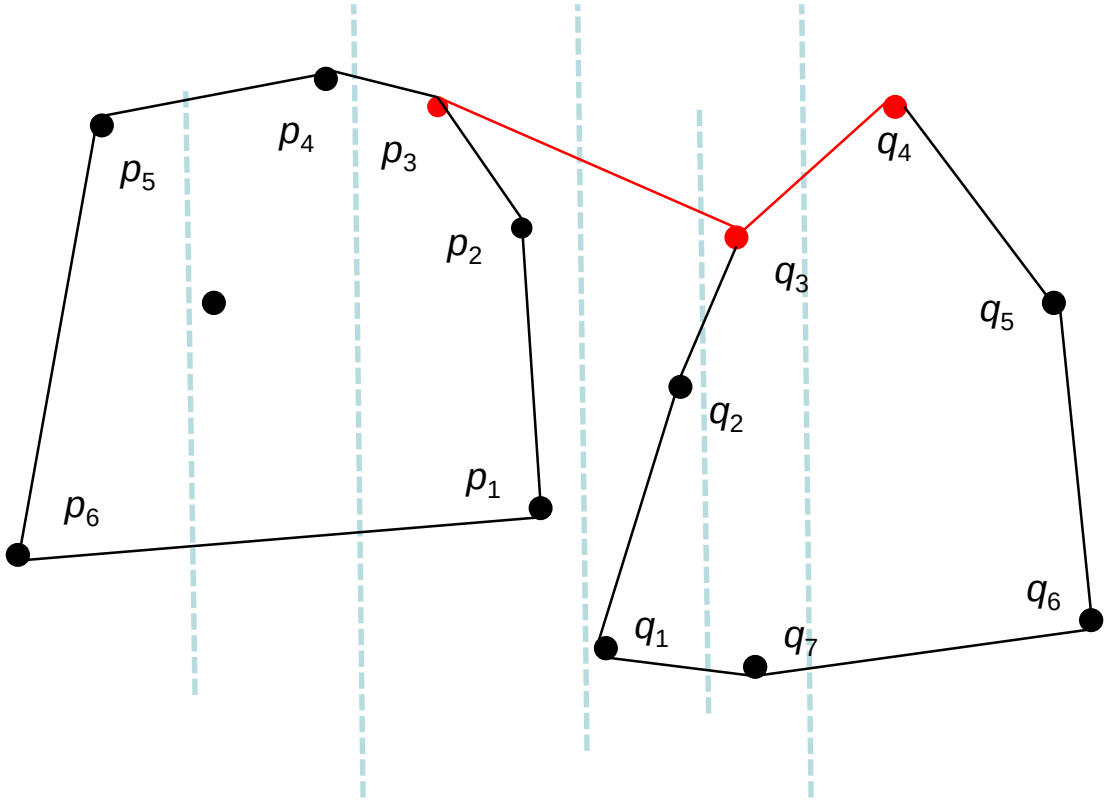
- We keep going as long as we only encounter right turns.

Finding the upper bridge



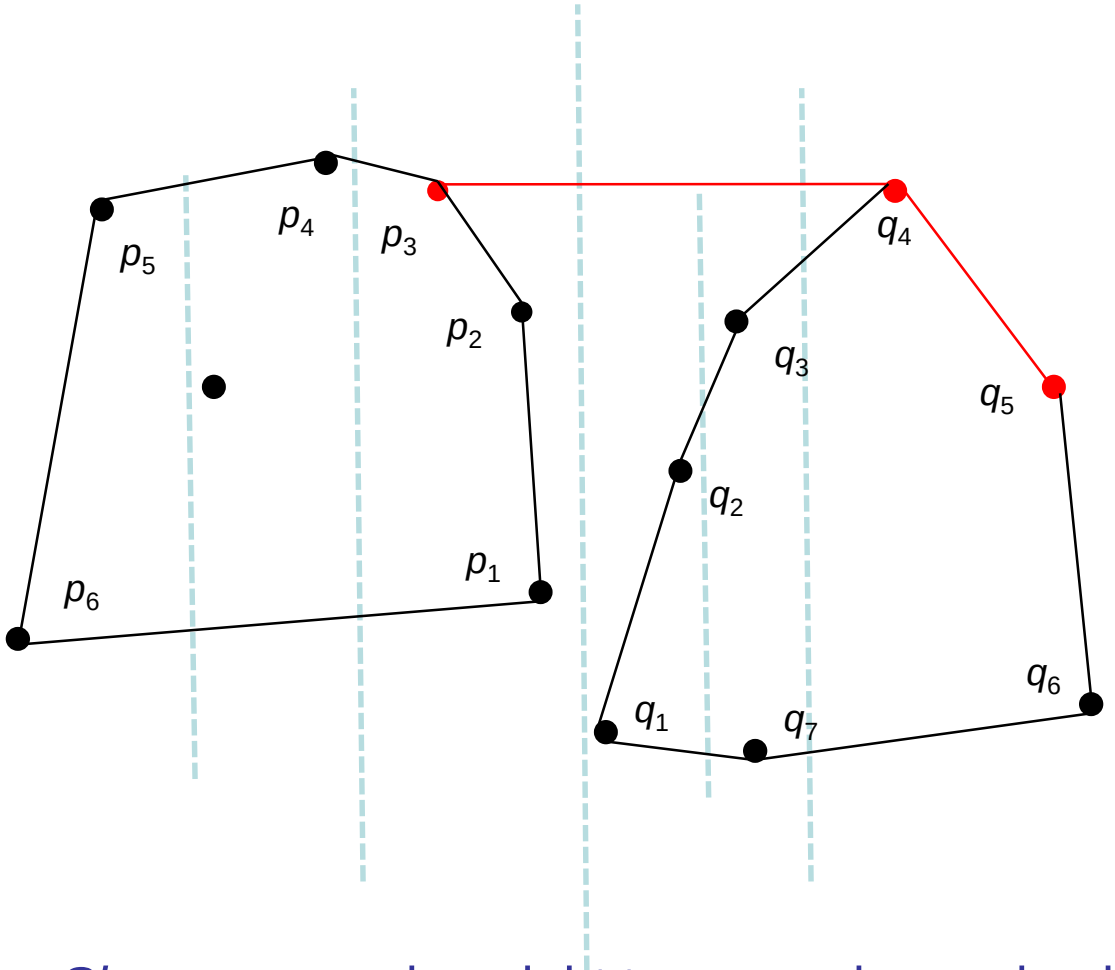
- Since q_3-p_3-p_4 is a left turn, we move to the other side again.

Finding the upper bridge



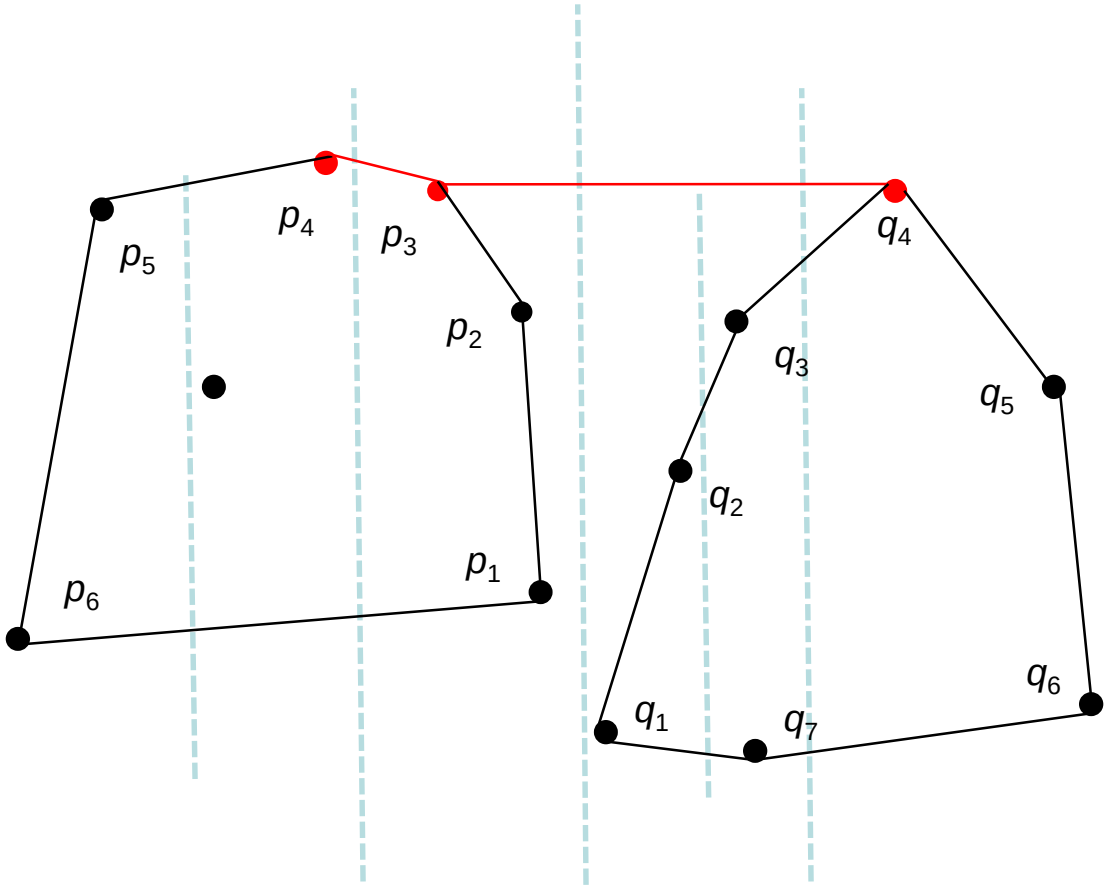
- As p_3-q_3-q_4 turns left, we keep going along the right hull.

Finding the upper bridge



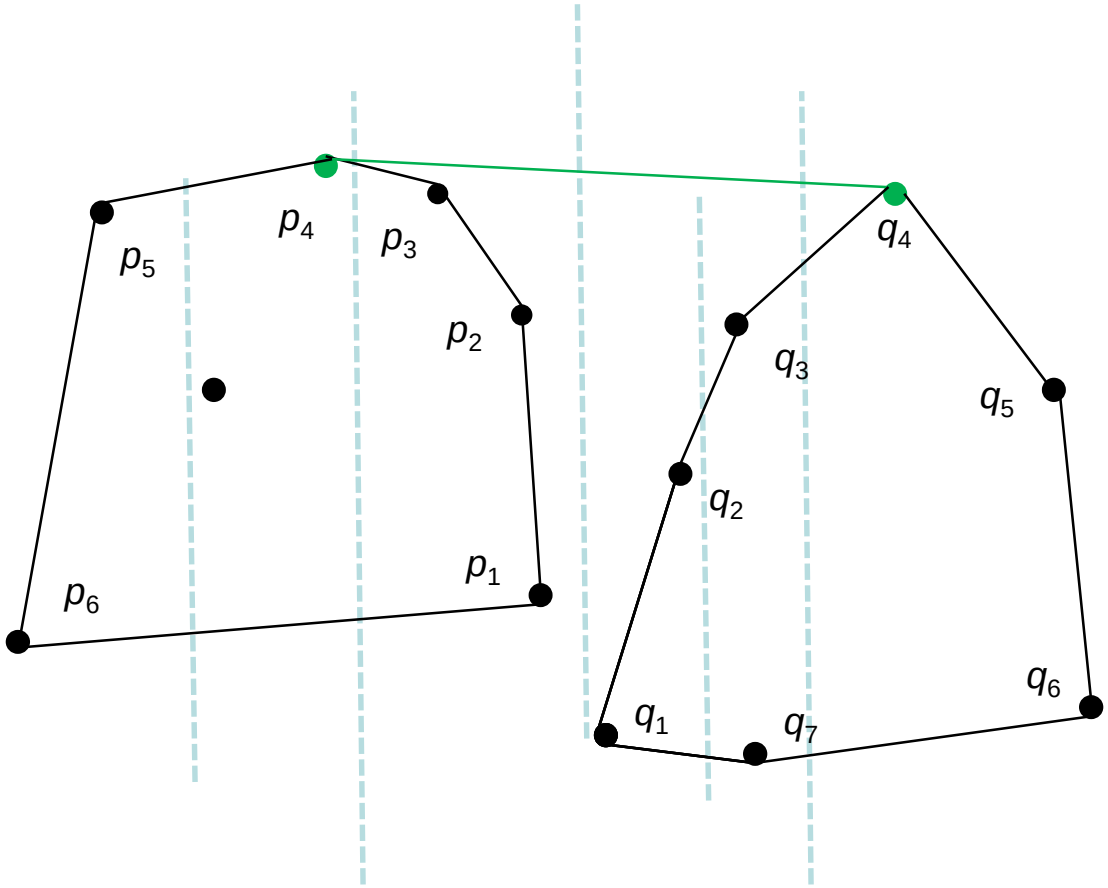
- *Since* p_3-q_4-q_5 is a right turn, we change back again.

Finding the upper bridge



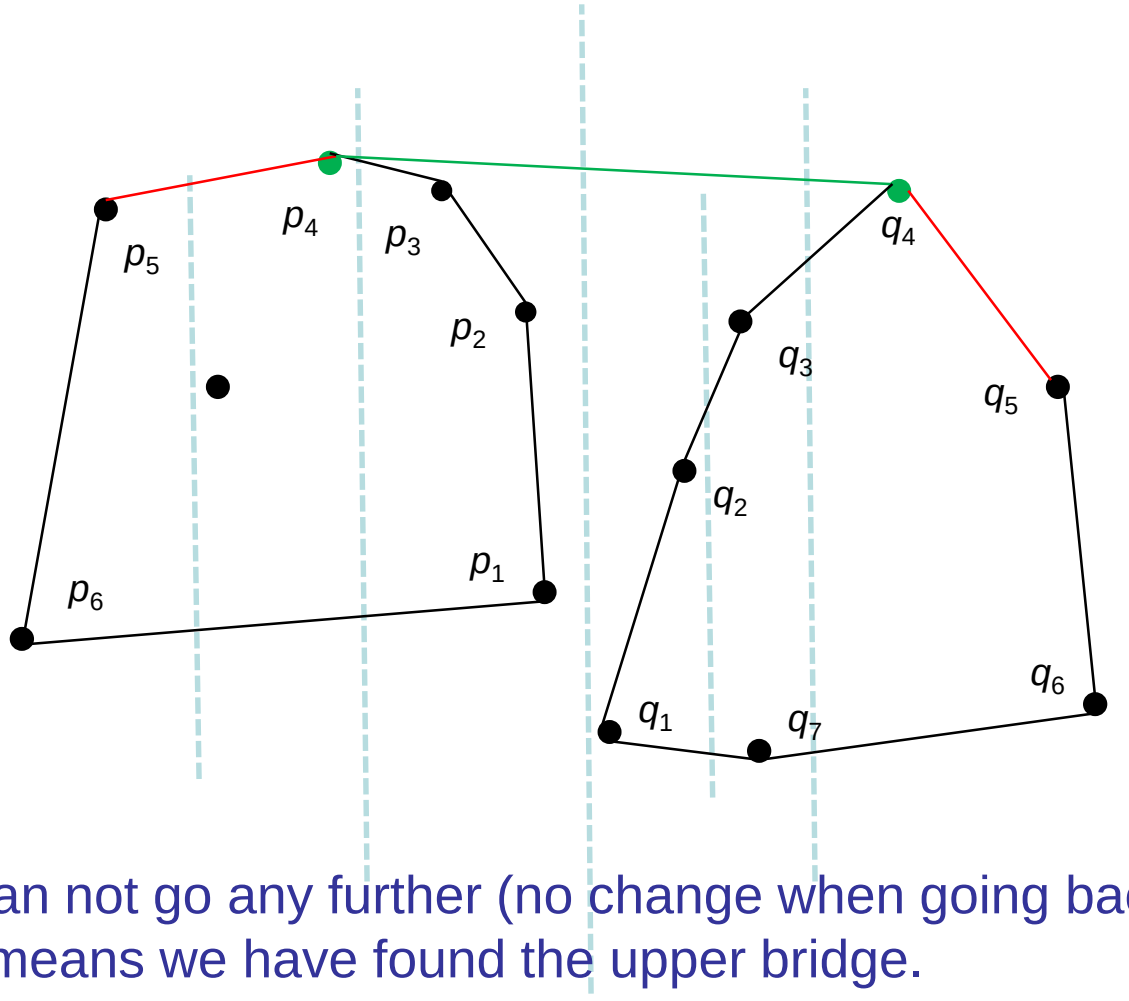
- As q_4-p_3-qp_4 turns right, we keep going along the left hull.

Finding the upper bridge



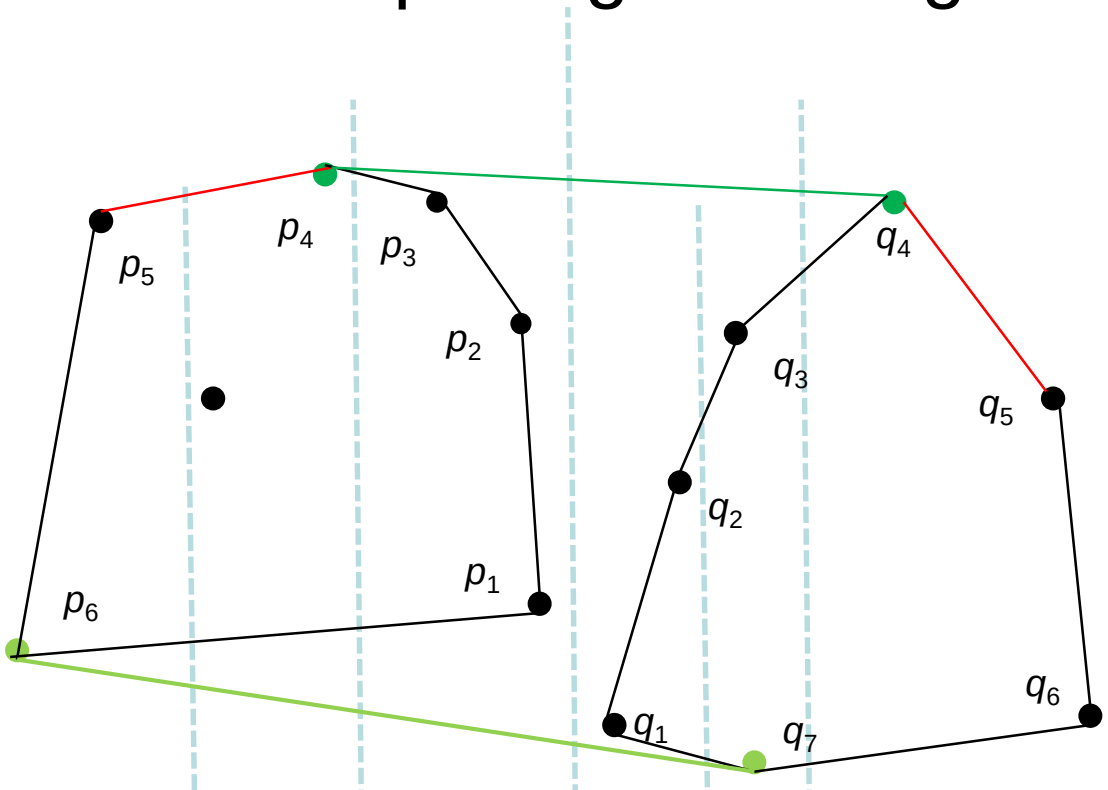
- Finally we have the upper bridge!

Finding the upper bridge



- We can not go any further (no change when going back and forth). This means we have found the upper bridge.
  (This process is inaccurately described in the book.)

Completing the merge



- The lower bridge is found the same way (upside down).
- Afterwards we need to remove some “old” corners (now interior points), and renumber the current corners.
  - All points visited during the search for the bridges, except the endpoints of the bridges are no longer corners of the merged hull.
  - We re-number remaining corners so that we get a continuous numbering of the corners around the merged hull.

Time complexity

(of the divide-and-conquer method for finding the convex hull)

- We first sort the points according to their *x*-value. This takes time $O(n \log n)$, where n is the total number of points. We can then easily do the partitioning.

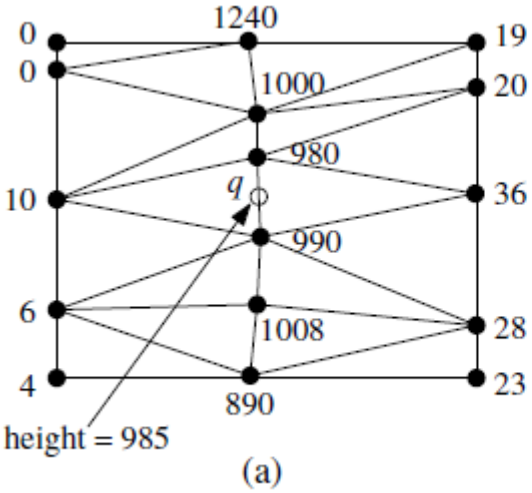
- Each time we merge two convex hulls, we may have to move the endpoints of the potential bridges m times, where m is the total number of nodes in the two merged sets. Thus, each merge takes time $O(m)$.

- All merging at one tree-depth will therefore take time $O(m_1) + O(m_2) + O(m_3) + \ldots + O(m_k)$ which becomes $O(n)$ since $m_1 + m_2 + m_3 + \ldots + m_k = n$.

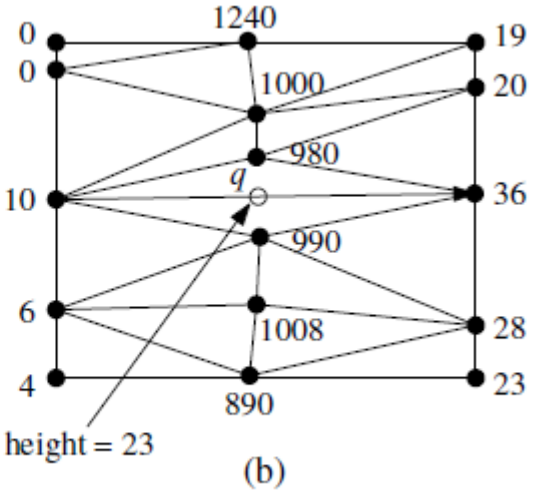
- The number of tree-levels do not exceed $\log_2 n$, so the total running time is therefore: $O(n \log n)$.

Example: Why large angles are best
The figure shows heights at different points

- One usually assumes that the edges of the triangulation are straight lines in the terrain.
- That means that the height of a point on an edge can be found by interpolation of the height of the endpoints of the edge
- We can see that the left triangulation below gives an intuitively better height for q than the triangulation to the right.



(a)



(b)