DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

CONVEX HULLS

PETR FELKEL

FEL CTU PRAGUE

felkel@fel.cvut.cz

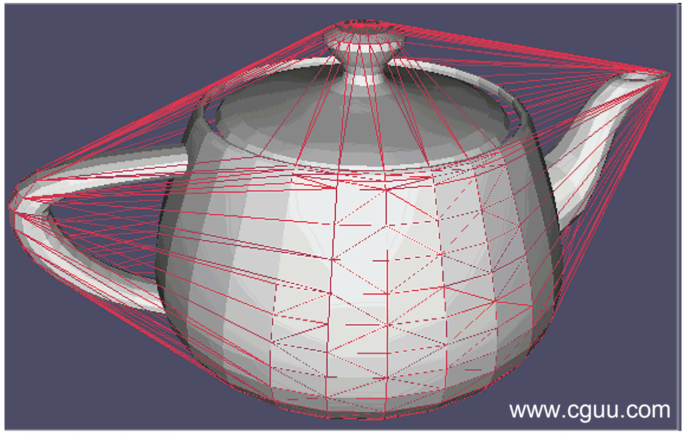
https://cw.felk.cvut.cz/doku.php/courses/a4m39vg/start

Based on [Berg] and [Mount]

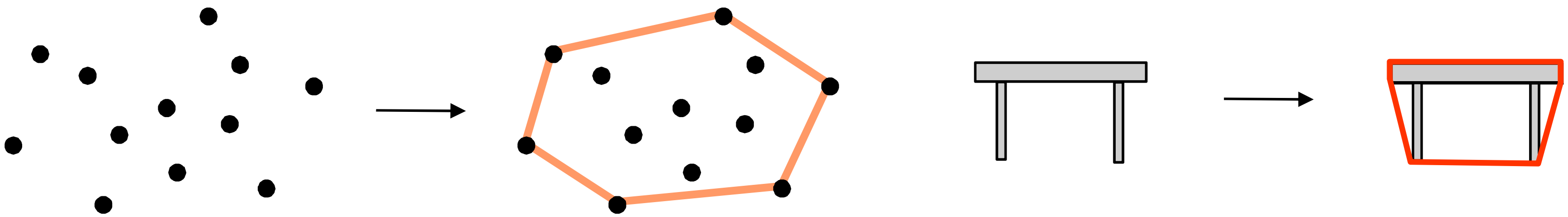
Version from 8.11.2012

Talk overview

- Motivation and Definitions
- Graham’s scan – incremental algorithm
- Divide & Conquer
- Quick hull
- Jarvis’s March – selection by gift wrapping
- Chan’s algorithm – optimal algorithm

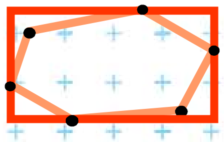


Convex hull (CH) – why to deal with it?

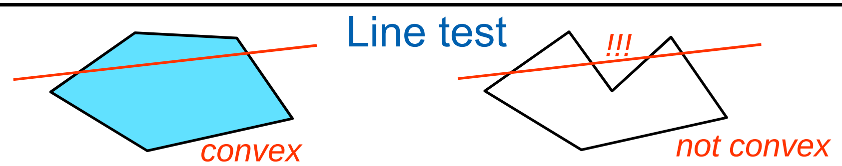


- *Shape approximation* of a point set or complex shapes (other common approximations include: minimal area enclosing rectangle, circle, and ellipse,…) – e.g., for collision detection
- *Initial stage* of many algorithms to filter out irrelevant points, e.g.:
  - diameter of a point set

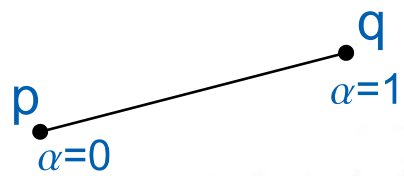
  - minimum enclosing convex shapes (such as rectangle, circle, and ellipse) depend only on points on CH



Convexity



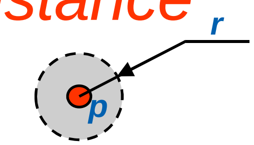
- A set S is *convex*
  - if for any points p,q $\in S$ the lines segment $\overline{pq} \subseteq S$, or
  - if any convex combination of p and q is in S
- Convex combination of points p, q is any point that can be expressed as
  $(1 - \alpha)\, p + \alpha q$, where $0 \le \alpha \le 1$



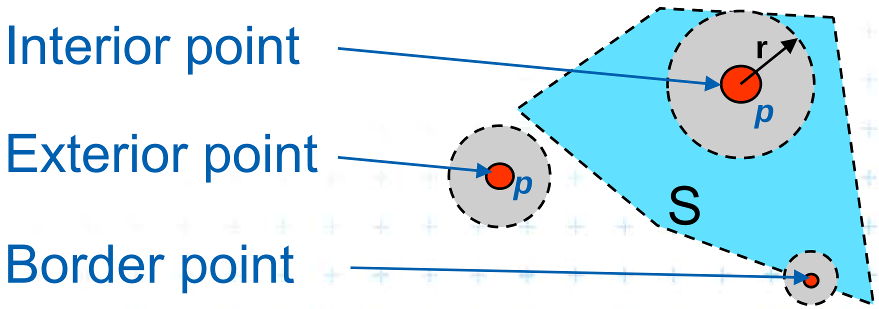
- *Convex hull* CH(S) of set S – is (similar definitions)
  - the smallest set that contains S
  - or: intersection of all convex sets that contain S
  - Or in 2D for points: the smallest convex polygon containing all given points

Definitions from topology in metric spaces

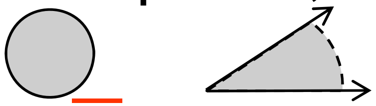
- *Metric space* – each two of points have defined a *distance*
- *r-neighborhood* of a point *p* and radius *r > 0*
  = set of points whose distance to *p* is strictly less than *r*
  (open ball of diameter *r* centered about *p*)

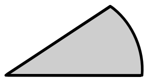
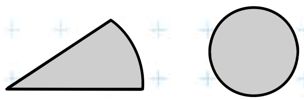


- Given set S, point *p* is
  - Interior point of S – if (r-neighborhood about *p* of radius *r*) $\subset$ S
  - Exterior point – if it lies in interior of the complement of S
  - Border point – is neither interior neither exterior



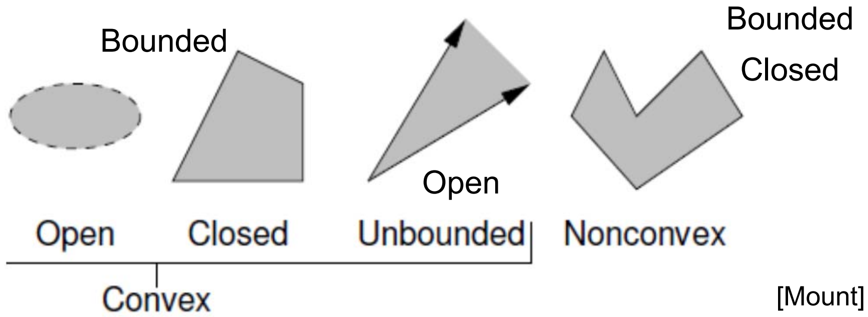
Definitions from topology in metric spaces

- Set S is Open (otevřená) 
  - $\forall p \in S \;\; \exists$ (r-neighborhood about p of radius r) $\subseteq S$
  - it contains only interior points, none of its border points
- Closed (uzavřená) 
  - If it is equal to its closure $\overline{S}$ (uzávěr = smallest closed set containing S in topol. space)
    $\forall$(r-neighborhood about p of radius r) $\cap\, S \neq \emptyset$)
- Clopen (otevřená i uzavřená) – Ex. Empty set $\emptyset$, finite set of disjoint components
  - if it is both closed and open
    (S= all positive rational numbers whose square is bigger than 2)

    space Q = rational numbers
    $S = (\sqrt{2}, \infty)$ in Q, $\sqrt{2} \notin Q$, $S = \overline{S}$
- Bounded (ohraničená)  Unbounded 
  - if it can be enclosed in a ball of finite radius
- Compact (kompaktní) 
  - if it is both closed and bounded

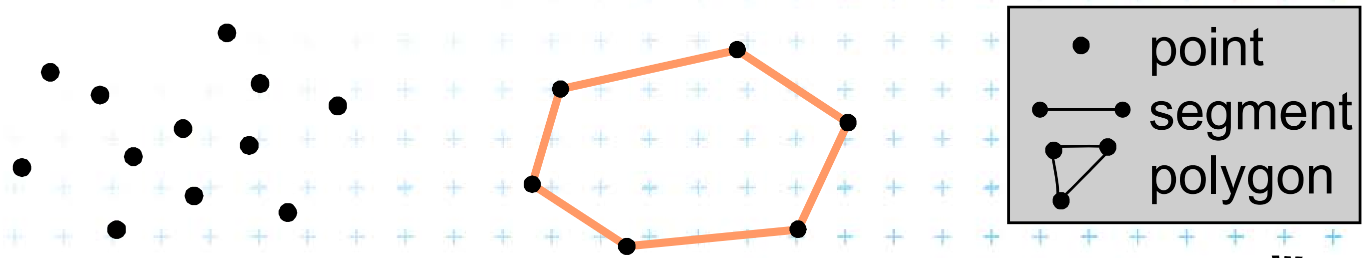
Definitions from topology in metric spaces

- *Convex set S may be bounded or unbounded*



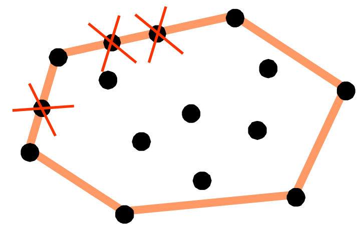
- *Convex hull* CH(S) of a finite set *S* of points in the plane

  = Bounded, closed, (= compact) convex polygon



Convex hull representation

- CCW enumeration of vertices
- Contains only the extreme points
  ("endpoints" of collinear points)



- Simplification for this semester
  Assume the input points are in general position,
  - no two points have the same *x*-coordinates and
  - no three points are collinear

  -> We avoid problem with non-extreme points on *x*
  (solution may be simple – e.g. lexicographic ordering)

Online x offline algorithms

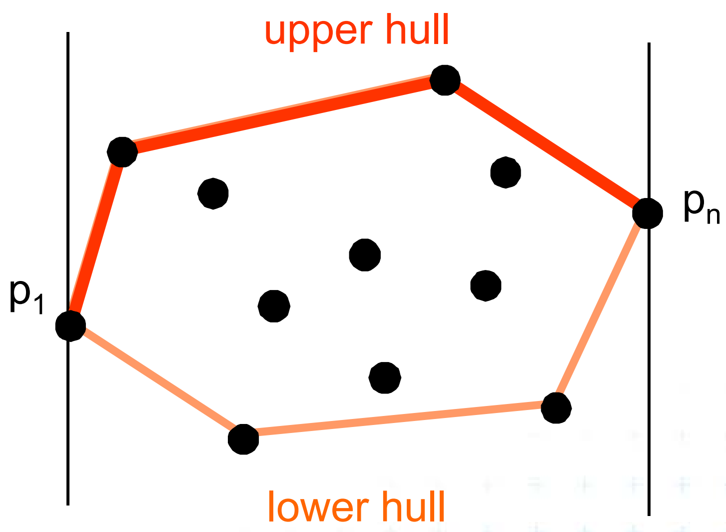
- Incremental algorithm
  - Proceeds one element at a time (step-by-step)
- Online algorithm (must be incremental)
  - is started on a partial (or empty) input and
  - continues its processing as additional input data becomes available (comes online, thus the name).
  - Ex.: insertion sort
- Offline algorithm (may be incremental)
  - requires the entire input data from the beginning
  - than it can start
  - Ex.: selection sort

Graham's scan

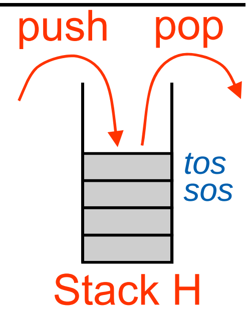
- Incremental O(*n* log *n*) algorithm
- Objects (points) are added one at a time
- Order of insertion is important
  - Random insertion
    –> we need to test: *is-point-inside-the-hull(p)* 
  - Ordered insertion
    Sort points according to *x* and add them left to right - it guarantees, that just added point is outside current hull
    - Original algorithm sorted the *angles* around the point with minimal *y*
    - Sorting *x-coordinates* is simpler to implement than sorting of angles

Graham's scan

- O(n log n) for unsorted points, O(n) for sorted pts.
- Upper hull, then lower hull. Merge.
- Minimum and maximum on x belong to CH



Graham's scan – incremental algorithm



GrahamsScan(points p)

Input: points p

Output: CCW points on the convex hull

sort points according to increasing x-coord -> $\{p_1, p_2, \ldots, p_n\}$

upper hull

push(p_1, H), push(p_2, H)

for *i* = *3* **to** *n* **do**

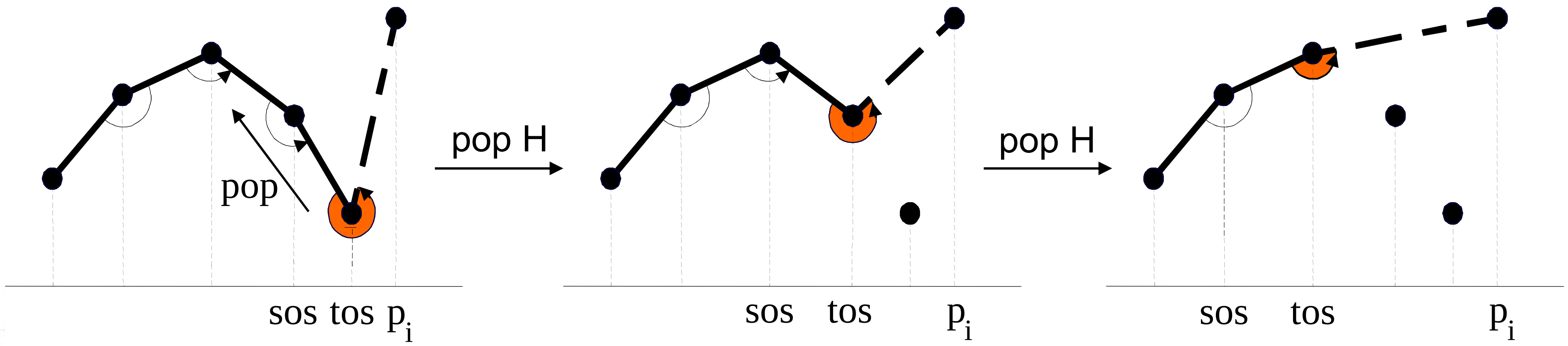
  while(size(H) $\geq$ 2 and orient(*sos*, *tos*, p_i) $\geq$ 0) // skip left turns

    pop H // (back-tracking)

  push(p_i, H) // store right turn

store H to the output (in reverse order) // upper hull

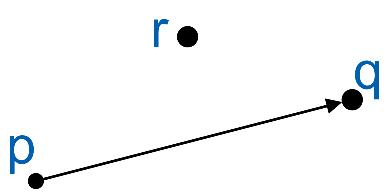
Symmetrically the lower hull

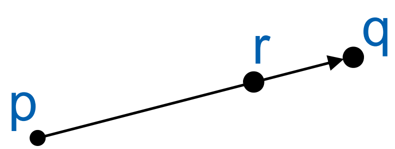


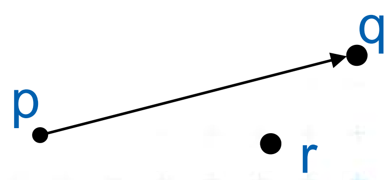
Position of point in relation to segment

$$\text{orient}(\,p,\,q,\,r\,)\begin{cases} >0 & r \text{ is left from } pq\text{, CCW orient} \\ =0 & \text{if } (\,p,\,q,\,r\,) \text{ are collinear} \\ <0 & r \text{ is right from } pq\text{, CW orient} \end{cases}$$

Point *r* is: **left from *pq*** **on segment *pq*** **right from *pq***

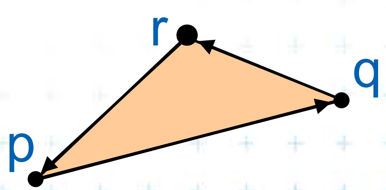


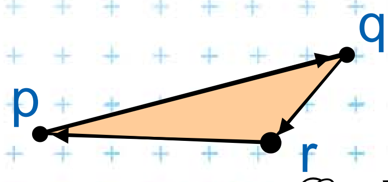




Convex polygon with edges pq and qr or

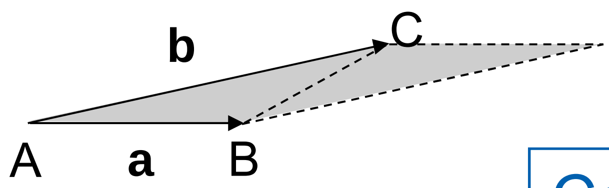
Triangle *pqr*: **is CCW oriented** **degenerated to line** **is CW oriented**





Geometric meaning: Area of Triangle ABC

- Position of point C in relation to segment AB is given by the sign of the triangle ABC area   2x Oriented area
- $T = ½ \, |\overrightarrow{AB} \times \overrightarrow{AC}|$
- $\mathbf{a} = B - A$
- $\mathbf{b} = C – A$
- $T = ½ \, (\mathbf{a}_x \, \mathbf{b}_y - \mathbf{a}_y \, \mathbf{b}_x)$



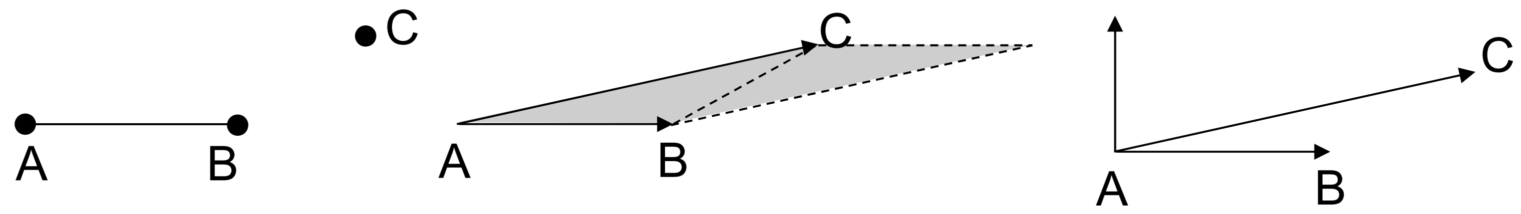
Can be computed as vector product

$$=> \; 2T = A_x B_y + B_x C_y + C_x A_y - A_x C_y - B_x A_y - C_x B_y$$

Or directly as determinant

$$2T = \begin{vmatrix} A_x & A_y & 1 \\ B_x & B_y & 1 \\ C_x & C_y & 1 \end{vmatrix} = A_x B_y + B_x C_y + C_x A_y - A_x C_y - B_x A_y - C_x B_y$$

Geometric meaning: Area of Triangle ABC



Equal to size of Vector product of vectors AB x AC

= Vector perpendicular to both vectors AB and AC

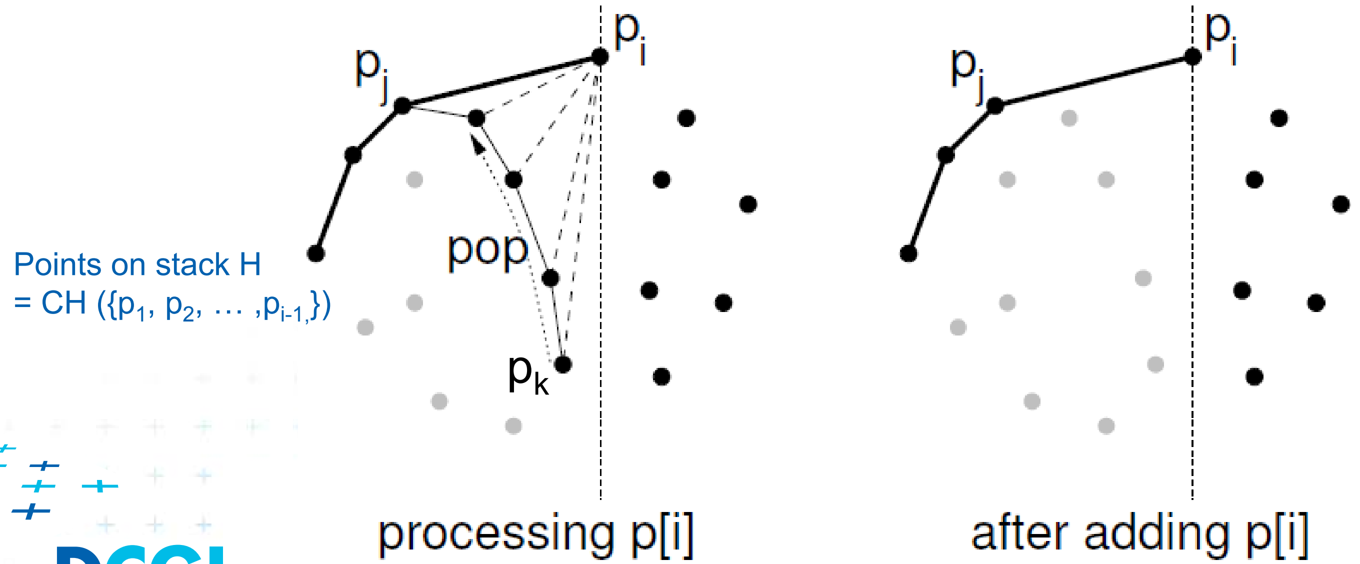
- If vectors in plane
  - it is perpendicular to the plane (*normal vector of the plane*)
  - only *z*-coordinate is non-zero
- $|\overrightarrow{AB} \times \overrightarrow{AC}|$ = z-coordinate of the normal vector

  = area of parallelopid

  = 2x area *T* of triangle ABC

Is Graham's scan correct?

- Stack H at any stage contains upper hull of the points $\{p_1,\ldots,p_j, p_i\}$, processed so far
  - For induction basis $H=\{p_1, p_2\}$ … true
  - p_i = last added point to CH, p_j = its predecessor on CH
  - Each point p_k that lies between p_j and p_i lies below p_jp_i and should not be part of UH after addition of p_i => is removed before push p_i. [orient(p_j, p_k, p_i) > 0, p_i is left from p_jp_k => p_k is removed from UH]
  - Stop if 2 points in the stack or after construction of the upper hull



[Mount]

Complexity of Graham's scan

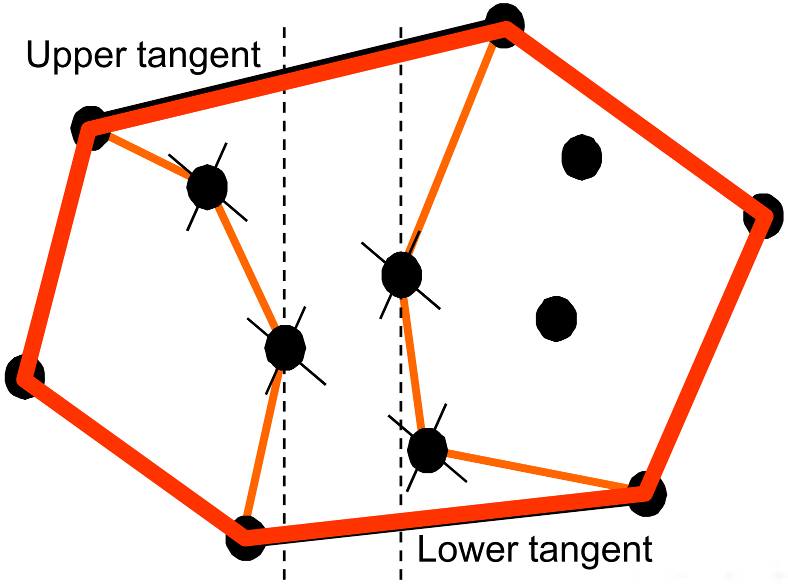
- Sorting according x – $O(n \log n)$
- Each point pushed once – $O(n)$
- Some ($d_i \le n$) points deleted while processing p_i
  – $O(n)$
- The same for lower hull – $O(n)$

- Total $O(n \log n)$ for unsorted points
  $O(n)$ for sorted points

Divide & Conquer

- $\Theta(n \log(n))$ algorithm
- Extension of mergesort
- Principle
  - Sort points according to *x*-coordinate,
  - recursively partition the points and solve CH.

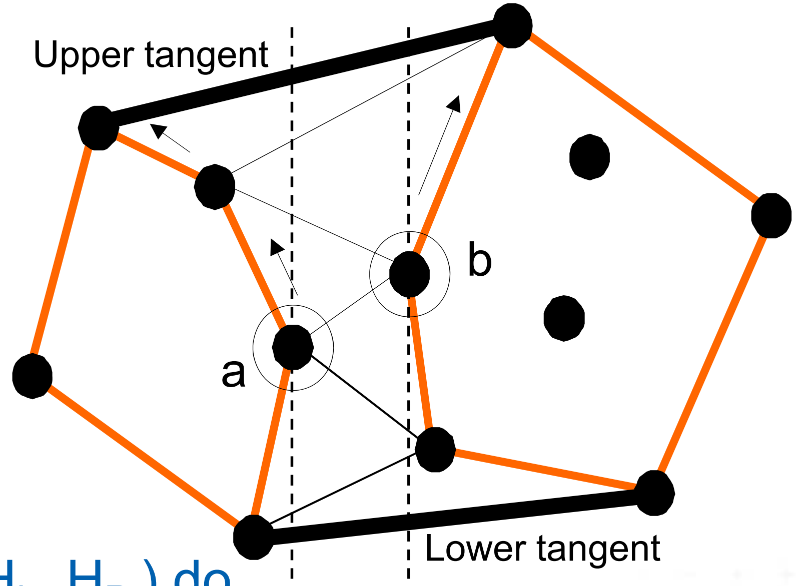
Convex hull by D&C



ConvexHullD&C(points **P)**
Input: points p
Output: CCW points on the convex hull
Sort points P according to *x*
return hull(P)

hull(points P)
    if $|P| \leq 3$ then
        compute CH by brute force,
        return
    Partition P into two sets L and R (with lower & higher coords *x*)
    Recursively compute H_L = hull(L), H_R = hull(R)
    H = Merge hulls(H_L, H_R) by computing
        Upper_tangent(H_L, H_R) // find nearest points, H_L CCW, H_R CW
        Lower_tangent(H_L, H_R) // (H_L CW, H_R CCW)
        discard points between these two tangents
    return H

Search for upper tangent (lower is symmetrical)



Upper_tangent(H_L, H_R)
Input: two non-overlapping CH's
Output: upper tangent *ab*

1. a = rightmost H_L
2. b = leftmost H_R
3. while(ab is not the upper tangent for H_L, H_R) do
4. while(ab is not the upper tangent for H_L) *a = a.succ* // move CCW
5. while(ab is not the upper tangent for H_R) *b = b.pred* // move CW
6. Return *ab*

Where: (ab is not the upper tangent for H_L) => orient(*a, b, a.succ*) ≥ 0
which means *a.succ* is left from line *ab*

$$m = |H_L| + |H_R| \leq |L| + |R| \Rightarrow \text{Upper Tangent: } O(m) = O(n)$$

Convex hull by D&C complexity

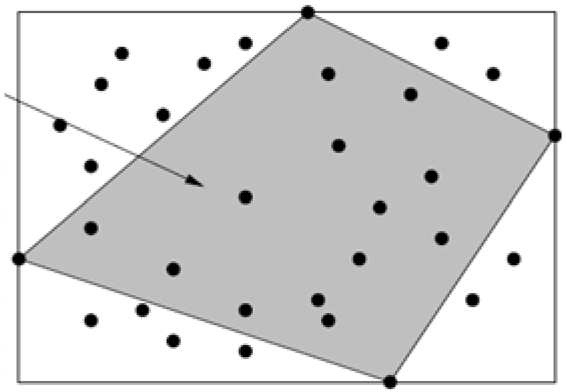
- Initial sort O(n log(n))
- Function hull()
  - Upper and lower tangent O(n)
  - Merge hulls O(1)
  - Discard points between tangents O(n)
- Overall complexity
  - Recursion

$$T(n) = \begin{cases} 1 & \ldots \text{ if } n \leq 3 \\ 2T(n/2) + O(n) & \ldots \text{ otherwise} \end{cases}$$

  - Overall complexity of CH by D&C: => O(n log(n))

Quick hull

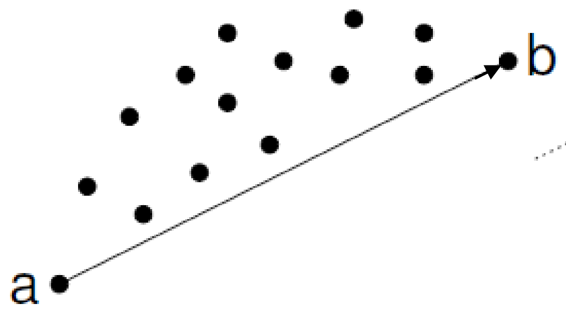
- A variant of Quick Sort
- O(n log n) expected time, max O(n^2)
- Principle
  - in praxis, most of the points lie in the interior of CH
  - E.g., for uniformly distributed points in unit square, we expect only O(log n) points on CH
- Find extreme points (parts of CH) quadrilateral, discard inner points
  - Add 4 edges to temp hull T
  - Process points outside 4 edges

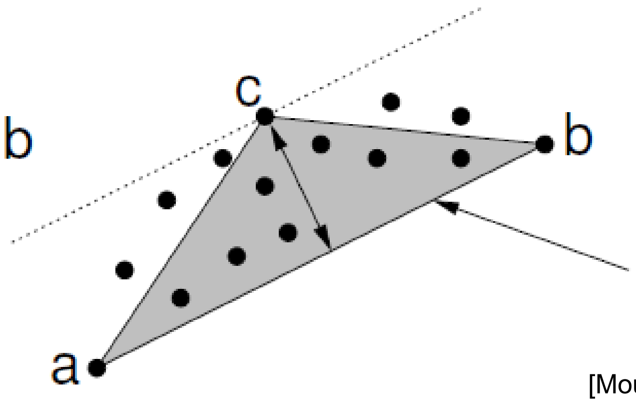


[Mount]

Process each of four groups of points outside

- For points outside *ab* (left from *ab*)
  - Find point *c* on the hull – max. perpend. distance to *ab*
  - Discard points inside triangle *abc* (right from the edges)
  - Split points into two subsets
    - outside *ac* (left from *ac*) and outside *cb* (left from *cb*)
  - Process points outside *ac* and *cb* recursively
  - Replace edge *ab* in *T* by edges *ac* and *cb*





[Mount]

Quick hull complexity

- n points remain outside the hull
- $T(n)$ = running time for such n points outside
  - O(n) - selection of splitting point c
  - O(n) - point classification to inside & (n_1+n_2) outside
  - $n_1+n_2 \leq n$
  - The running time is given by recurrence

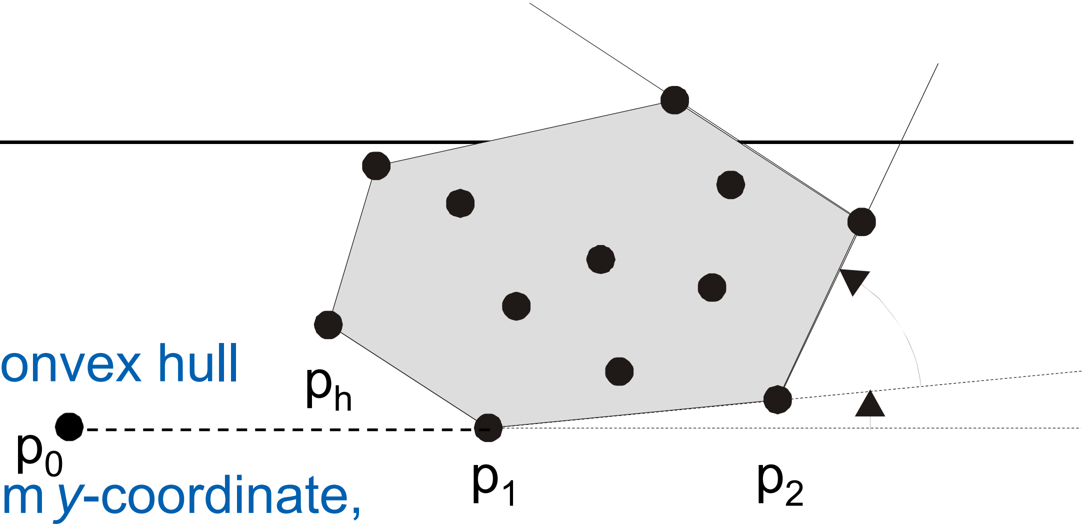
$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n_1) + T(n_2) & \text{where } n_1+n_2 \leq n \end{cases}$$

  - If evenly distributed that $\max(n_1, n_2) \leq \alpha n$, $0 \leq \alpha \leq 1$ then solves as QuickSort to $O(cn \log n)$ where $c=f(\alpha)$ else $O(n^2)$ for unbalanced splits

Jarvis's March – selection by gift wrapping

- Variant of $O(n^2)$ selection sort
- Output sensitive algorithm
- $O(nh)$ … *h = number of points on convex hull*

Jarvis's March



JarvisCH(points P)
Input: points p
Output: CCW points on the convex hull

1. Take point p_1 with minimum y-coordinate,
   // p_1 will be the first point in the hull
2. Take a horizontal line, i.e., create temporary point $p_0 = (-\infty, p_1.y)$
3. i = 1
4. repeat
5. | Rotate the line around p_i until bounces to the nearest point q
   | // compute the smallest angle by the smallest orient(p_{i-1} , p_i , q)
6. | i++
   | p_i = the bounced nearest point q
7. until ($q \neq p_1$)

Output sensitive algorithm

Complexity: $O(n) + O(n) * h$ => $O(h*n)$
good for low number of points on convex hull

Output sensitive algorithm

- Worst case complexity analysis analyzes the worst case data
  – Presumes, that all (const fraction of) points lie **on** the CH
  – The points are ordered along CH
    => We need sorting => $\Omega(n \log n)$ of CH algorithm
- Such assumption is rare
  – usually only much less of points are on CH
- Output sensitive algorithms
  – Depend on: input size n and the size of the output h
  – Are more efficient for small output sizes
  – Reasonable time for CH is $O(n \log h)$

Chan's algorithm

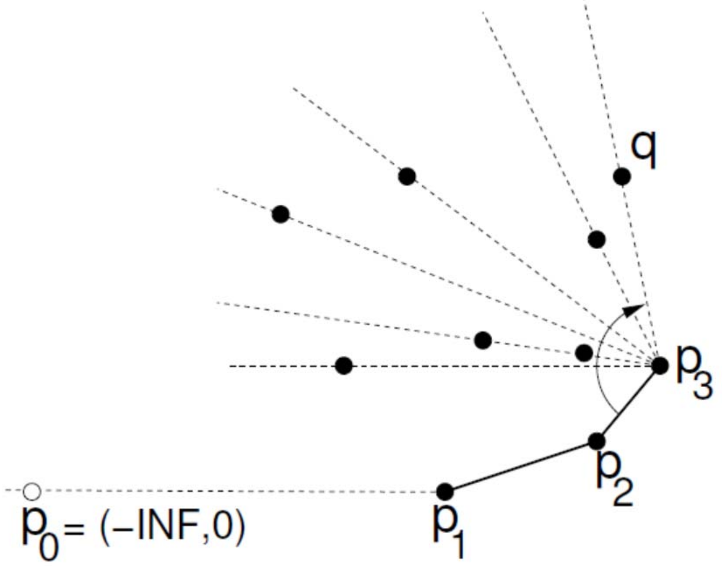
- Cleverly combines Graham's scan and Jarvis's march algorithms
- Goal is $O(n \log h)$ running time
  - We cannot afford sorting of all points - $\Omega(n \log n)$

  => Idea: limit the set sizes to polynomial h^c

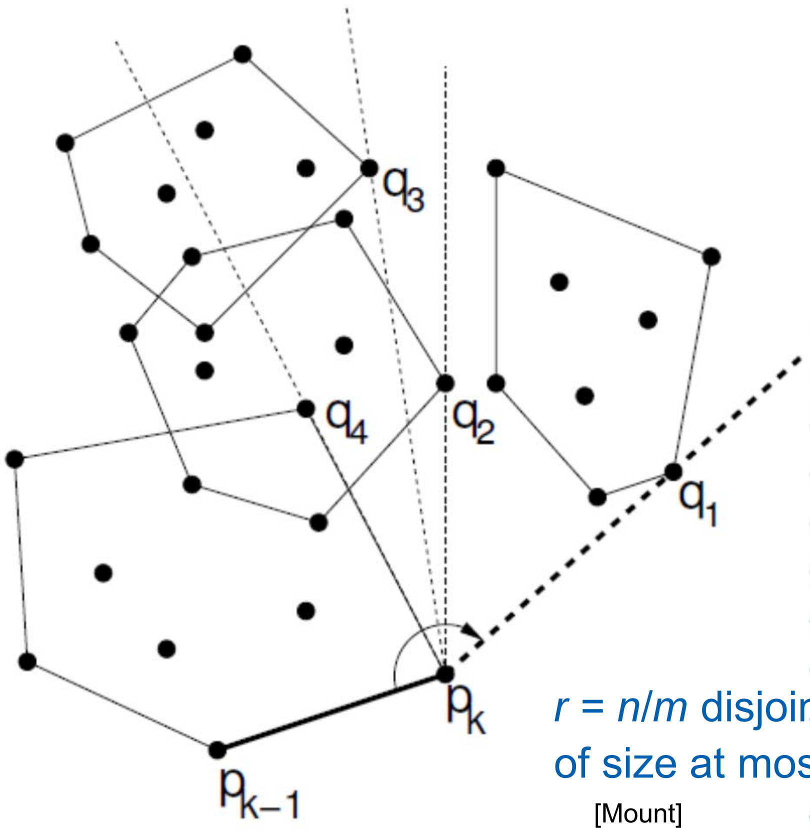
  the complexity does not change => $\log h^c = \log h$
  - h is unknown – we get the estimation later
  - Use estimation m, better not too high => $h \leq m \leq h^2$
- Partition points P into r-groups of size m, $r = n/m$
  - Each group take $O(m \log m)$ time - sort + Graham
  - r-groups take $O(r\, m \log m) = O(n \log m)$ - Jarvis

Merging of *m* parts in Chan's algorithm

- Merge *r*-group CHs as "fat points"
  - Tangents to convex *m*-gon can be found in O(log *m*) by binary search



[Mount]



r = *n*/*m* disjoint subsets of size at most *m*

[Mount]

Chan’s algorithm complexity

- *h* points on the final convex hull

  => at most *h* steps in the Jarvis march algorithm

  – each step computes *r*-tangents, O(log *m*) each

  – merging together O(*hr* log *m*)

  r-groups of size *m*, $r = n/m$

- Complete algorithm O(*n* log *h*)

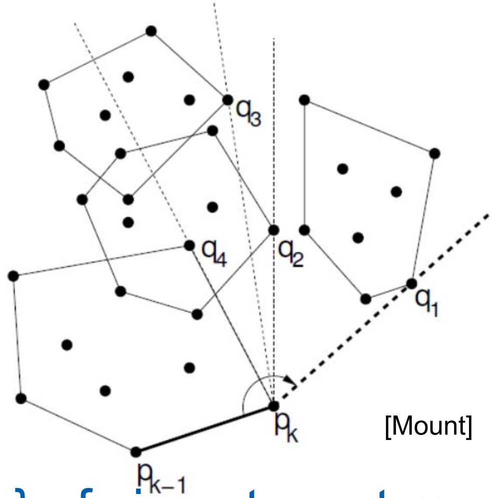
  – Graham’s scan on partitions O(*r* .*m* log *m*)=O(*n* log *m*)

  – Jarvis Merging: O(*hr* log *m*) = O(*h n*/*m* log *m*), …4a)

  $h \leq m \leq h^2$ = O(*n* log *m*)

  – How to guess *m*? *Wait!*

Chan's algorithm for known *m*



PartialHull(*P*, *m*)
Input: points P
Output: group of size *m*

1. Partition *P* into $r = \lceil n/m \rceil$ disjoint subsets $\{p_1, p_2, \ldots, p_r\}$ of size at most *m*
2. for *i=1 to r do*
   a) Convex hull by GrahamsScan(P_i), store vertices in ordered array
3. let p_1 = the bottom most point of P and $p_0 = (-\infty, p_1.y)$
4. for k = 1 to m do // compute merged hull points
   a) for i = 1 to r do // angle to all r subsets
      Compute the point $q_i \in P$ that maximizes the angle $\angle\, p_{k-1}, p_k, q_i$
   b) let p_{k+1} be the point $q \in \{q_1, q_2, \ldots, q_r\}$ that maximizes $\angle\, p_{k-1}, p_k, q$
      (p_{k+1} is the new point in CH)
   c) if $p_{k+1} = p_1$ then return $\{p_1, p_2, \ldots, p_k\}$
5. return "Fail, *m* was too small"

O(log *m*)

Chan's algorithm – estimation of *m*

ChansHull
Input: points P
Output: convex hull $p_1 \ldots p_k$

1. for $t = 1, 2, \ldots$ do {
   a) let $m = \min(2^{2^t}, n)$
   b) L = PartialHull(P, m)
   c) if $L \neq$ "Fail, m was too small" then return L

   }

Sequence of choices of m are { 4, 16, 256,…, 2^{2^t},…, n } … squares

Example: for h = 23 points on convex hull of n = 57 points, the algorithm will try this sequence of choices of m { 4, 16, 57 }

1. 4 and 16 will fail
2. 256 will be replaced by n

Complexity of Chan's Convex Hull?

- The worst case: Compute all iterations
- t^{th} iteration takes $O(\, n \log 2^{2^{\wedge}t}) = O(n\, 2^t)$
- Algorithm stops when $2^{2^{\wedge}t} \geq h$ => $t = \lceil \lg \lg h \rceil$
- All $t = \lceil \lg \lg h \rceil$ iterations take:

Using the fact that $\sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

$$\sum_{t=1}^{\lg \lg h} n 2^t = n \sum_{t=1}^{\lg \lg h} 2^t \leq n 2^{1+\lg \lg h} = 2n \lg h = O(n \log h)$$

2x more work in the worst case

Conclusion in 2D

- Graham's scan: $O(n \log n)$, $O(n)$ for sorted pts
- Divide & Conquer: $O(n \log n)$
- Quick hull: $O(n \log n)$, max $O(n^2)$ ~ distrib.
- Jarvis's march: $O(hn)$, max $O(n^2)$ ~ pts on CH
- Chan's alg.: $O(n \log h)$ ~ pts on CH

References

- [Berg] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars: Computational Geometry: *Algorithms and Applications*, Springer-Verlag, 3rd rev. ed. 2008. 386 pages, 370 fig. ISBN: 978-3-540-77973-5, Chapter 5, http://www.cs.uu.nl/geobook/
- [Mount] David Mount, - CMSC 754: Computational Geometry, Lecture Notes for Spring 2007, University of Maryland, Lectures 3 and 4. http://www.cs.umd.edu/class/spring2007/cmsc754/lectures.shtml
- [Chan] Timothy M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions., *Discrete and Computational Geometry*, 16, 1996, 361-368.

  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.44.389

DCGI