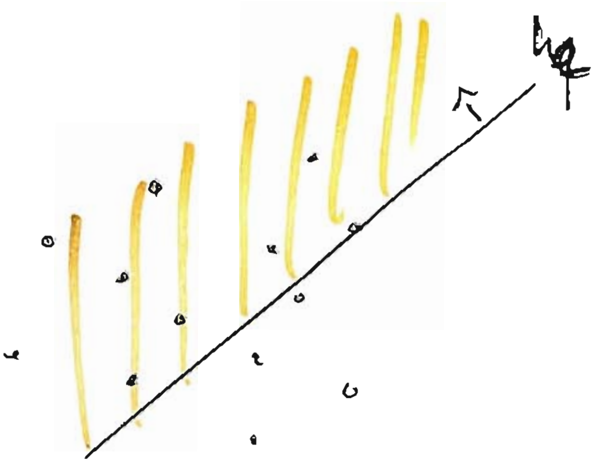
6.851 ADVANCED DATA STRUCTURES
(ANDRÉ SCHULZ' NOTES)

Lecture 6

PARTITION TREES

We want to answer halfspace & simplex queries in $\mathbb{R}^2$



How many points are above ℓ?

Halfspace query

How many points are inside t ?

Simplex Query

Can be used for polygonal queries

We study halfspace queries first

Idea: We partition the point set into Δ

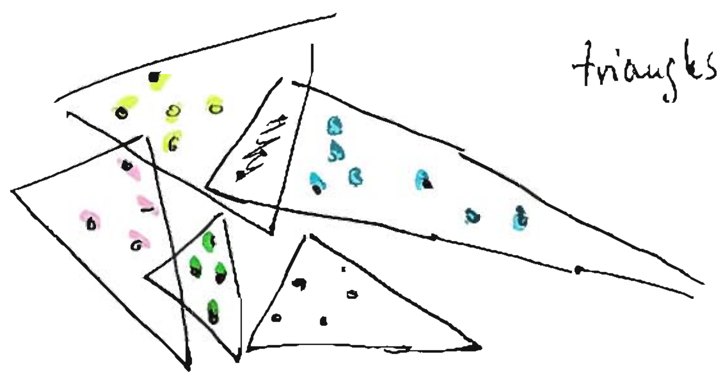
$$S=\{p_1 \dots p_n\} \Rightarrow \Psi = \{(S_1,t_1),(S_2,t_2)\dots(S_k,t_k)\}$$

s.t. $S = S_1 \cup S_2 \dots \cup S_k$

$\forall i,j \quad S_i \cap S_j = \emptyset$ unless $i=j$

t_i = triangles

Example



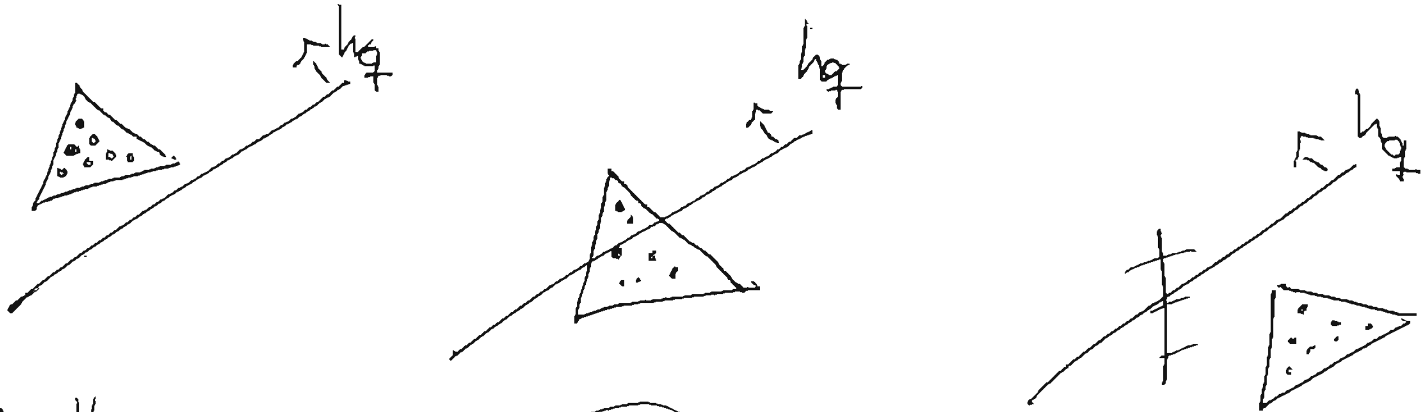
triangles may intersect

- Crossing # of Ψ = maximal # of triangles that can be intersect by some line ℓ

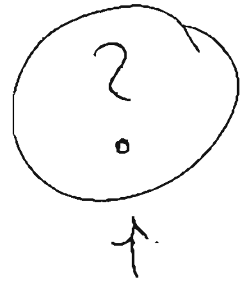
- Ψ is fine $\iff$ $|S_i| = \frac{2n}{r}$ (well distributed)

at most twice as large as the average

Observation

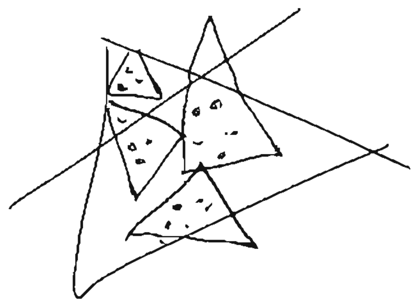


add all points to counter

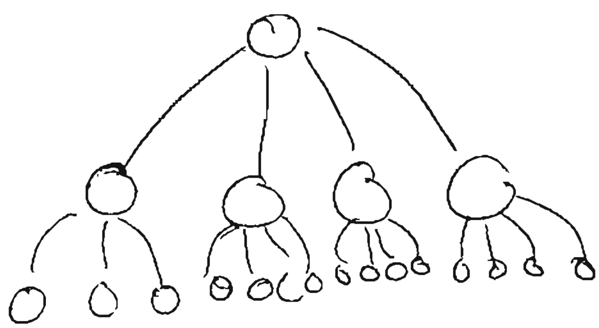


Recurse!

ignore those points



We construct a tree (PARTITION TREE) based on this idea:



to analyse the performance of queries we better have a bound on the crossing number of Ψ.

Theorem Matoušek 92

$\forall r : 1 \le \cancel{k}\, r \le n$ ~~st~~ $\exists$ fine partition Ψ of size ~~k~~ r with crossing#$(\Psi) = O(\sqrt{r})$.

Ψ can be computed in $O(n^{1+\varepsilon})$ for $\varepsilon > 0$

Since we spend at most $O(\sqrt{r}) = \text{const.}$ time / node

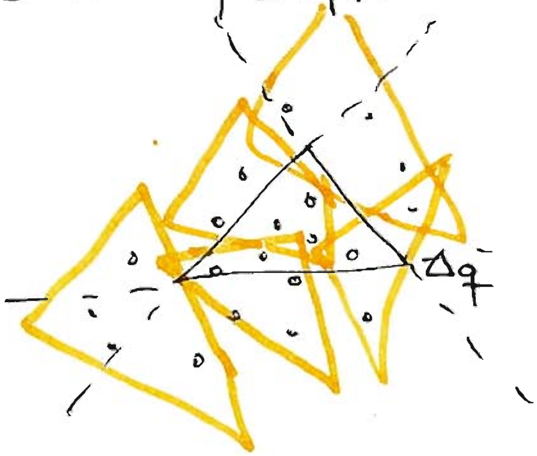
$\Rightarrow O(n^{\frac{1}{2}+\varepsilon})$ query time

Storage : $O(n)$ for $r > 2$

Preproc : $O(n^{1+\varepsilon})$

Now: Simplex Queries

Same idea, different crossing #



$\text{crossing}_\Delta \#(\Psi) \le 3 \cdot \text{crossing} \#(\Psi)$

(Obviously)

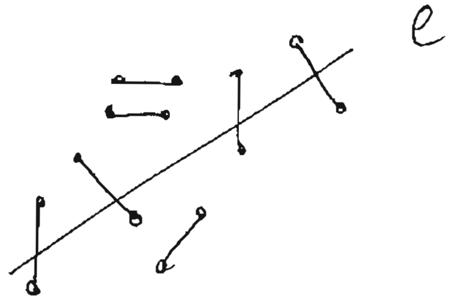
$\Rightarrow$ Same results (different constants)

MULTI LEVEL PARTITION TREES

Instead of asking # of Points we can answer all kinds of questions, just change ~~the~~ ~~the~~ what is stored in the nodes of the part. tree

(e.g. max/min queries, sums ...)

Now consider the following query:

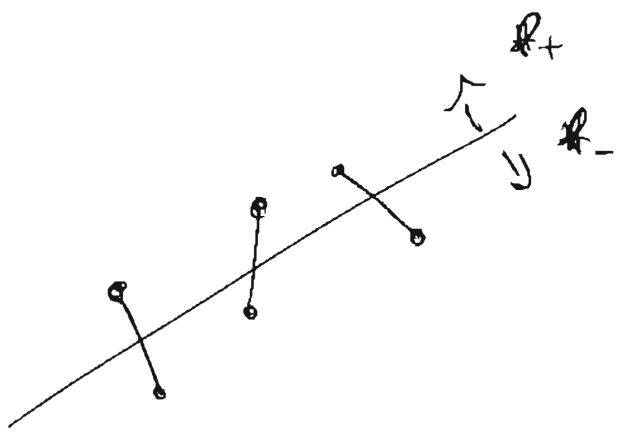


S = set of line segments

ℓ = query line

How many line segments intersect ℓ?

Observation:



The segments have one point in ℓ^*_+, one point in ℓ_-

$\Rightarrow$ report all points in ℓ_+ (1)
and check if the other endpoint lies in ℓ_- (2)

We do (1) as partition tree, in every node of the partition tree we store another partition tree for the opposite endpoints of the covered point set

More precisely: • For every segment we define the ~~left~~ left & right endpoint

• The partition tree (1st level) stores the left endpoints

• The 2nd level PT's store the (relevant) right endpoints

2 searches are necessary: 1st look for left endpoints ℓ_+ with other endpoints in ℓ_-

2nd look for left end points in ℓ_- with other endpoint in ℓ_+

Both searches use the same DS

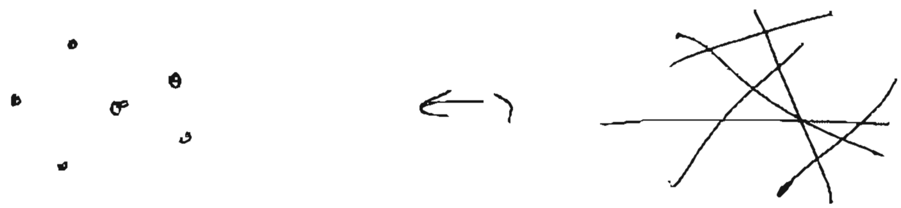
<u>Performance</u>: Storage: $O(n \log n)$ — increment

(without proof) Query time: $O(n^{\frac{1}{2}+\varepsilon})$

Faster Queries (but $O(n^2)$ storage)

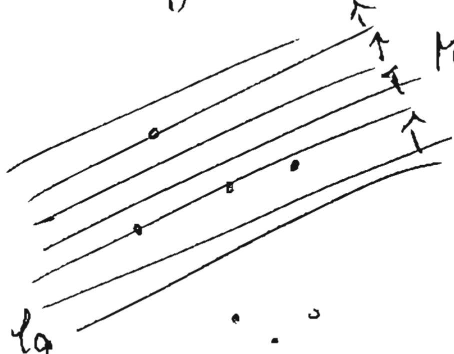
Use DUALITY TRANSFORM

2D Point set $\longleftrightarrow$ line segments in 2D



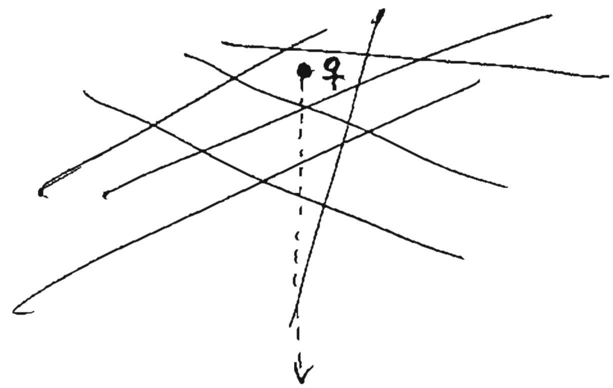
$p = (p_x, p_y) \longleftrightarrow p^* : \ell_p : y = p_x x - p_y$

original view



Move the line and report all points you hit

dual view



go from q straight down and report all lines you hit

This can be precomputed!

We have $O(n^2)$ cells in the dual arrangement and the point location can be done in $O(\log n)$ (using trapezoidal maps)

However ~~the~~ for simplex query the aproach is more complicated

(↗ CUTTING TREES -?-)

Related work: (Simplex Range searching)

- ~~Simplex~~ More complicated DS due to Matoušek 92 can answer halfspace queries in $O(\sqrt{n}\, 2^{O(\log^* n)})$ time (not usable for multilevel ~~very~~ partition trees)

- Also due to Matoušek 92, higher dimensional results ($O(n^{1-\frac{1}{d}-\varepsilon})$ query time) $O(n^{1+\varepsilon})$ space

- lower bounds due to Chazelle 89
  $\Omega(n/(m^{1/d}\log n))$ query time
  space
  in the plane
  $\Omega(n/\sqrt{m})$

- halfspace reporting: $O(\log n)$ Query time
  $O(n)$ storage
  [Chazelle, Guibas, Lee 85]

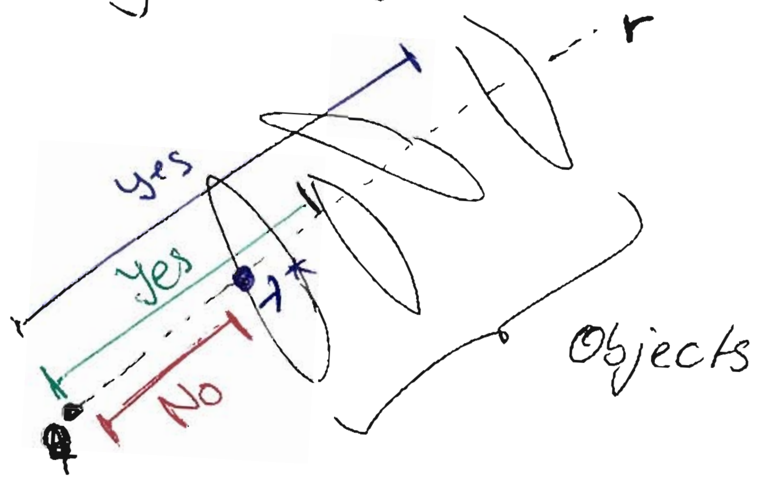
And many more

- space / time trade-off
  $O(m^{1+\varepsilon})$ space $O(\frac{n}{m^{1/d}}\log^2 n)$ query time

[Chazelle, Sharir, Welzl '92]

Back to Ray Shooting

General Ray shooting approach



Ray Shooting is an optimization problem that asks for λ^* on r

⇓

There is also a monotone! decision problem:

> Input: λ
>
> Output: Yes, if $q \to \lambda$ intersect an object
>
> No if " ——— " no object
>
> Algorithm A, running time T_A

Can we answer the optimization problem with help of the decision problem?

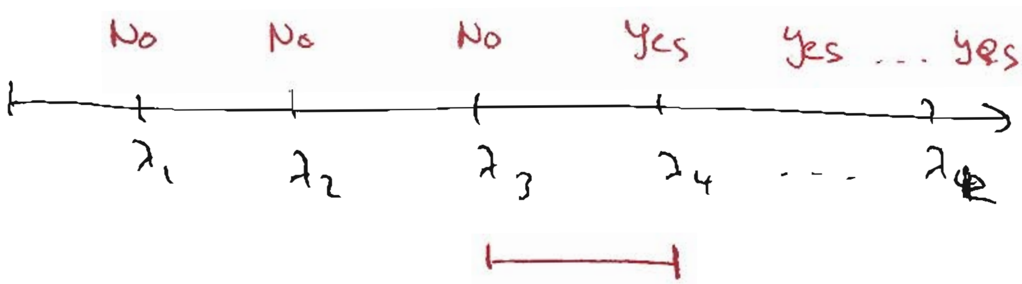
→ Yes with a technique called parametric search

(Applicable for many Geometric opt. problems) [Megiddo]

We assume that all conditional "branches" in A depend on a sign of a polynomial

<u>Idea:</u> • Run A with <u>generic input</u> $\lambda = (-\infty, \infty)$

- If we read a condition, compute the roots of the polynomial
  - → critical values $\lambda_1, \lambda_2 \dots \lambda_k$
  - → Run A on the critical values



There is one switch from No → Yes (monoton problem)

$\Rightarrow$ λ^* has to lie in $(\lambda_3, \lambda_4]$

- Run A with generic input $\lambda = (\lambda_3, \lambda_4]$

We can speed-up by binary search
(→ do it only for on set of critical values does not help → this is a constant)

- We batch up ~~p~~ p searches
- this takes $p + \log p \cdot T_A$ time

Median   Search

- If we could run A as parallel Algorithm on p processors in T_p time

⇒ total time : $T_p \cdot p + T_p T_A \log p$

Notice that the parallelism is virtual
(we need that we have p independent comparisons)

~~#~~ Ray Shooting on Hyperplanes in $\mathbb{R}^d$

Decision problem:

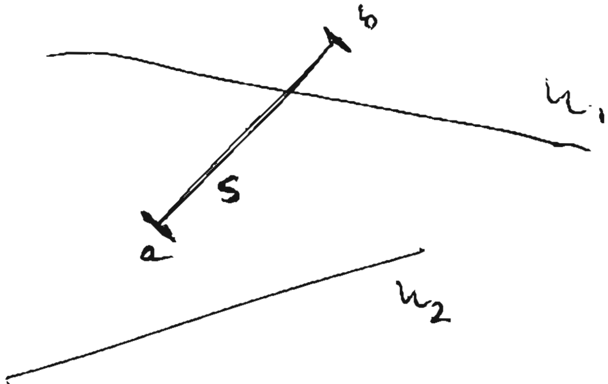
Input	: segment s
Output	: yes if $s \cap h$
	no else

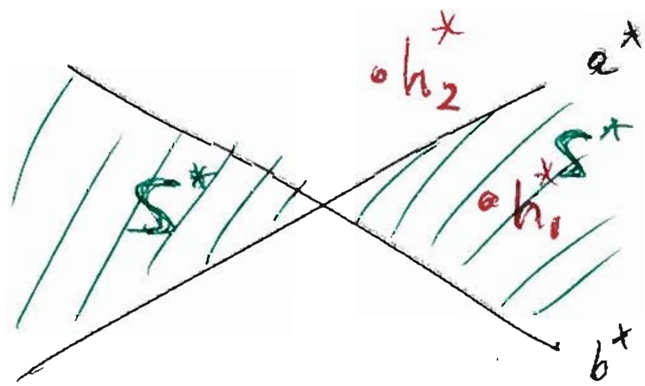
Hyperplane

We use again duality transform

schematic

primal $\longrightarrow$ dual





Does some h_i lies in the ~~upper~~ double wedge s^*

$\Rightarrow$ The dual problem can be answered with simplex range searching

We can use the DS of ~~Sharir~~ Chazelle, Sharir, Welzl (which can be executed in $O(\log n)$ parallel steps using $O\left(\frac{n}{m^{1/d}} \log n\right)$ processors)

Using the reverse search technique we get

Storage : $O(n^{1+\epsilon})$

preproc : $O(n^{1+\epsilon})$

query time : $O\left(\frac{n}{m^{1/d}} \log^4 n\right)$ simplified