KD-Trees

KD-Trees

Construction:

Input: $P = \{p^1, \dots, p^n\} \subseteq \mathbb{R}^d$

choose split axis i (i-th coord) ← s.t. called "discriminator"

find median m of $\{p_i^1, \dots, p_i^n\}$

partition P into $P^- = \{p \in P \mid p_i \leq m\}$

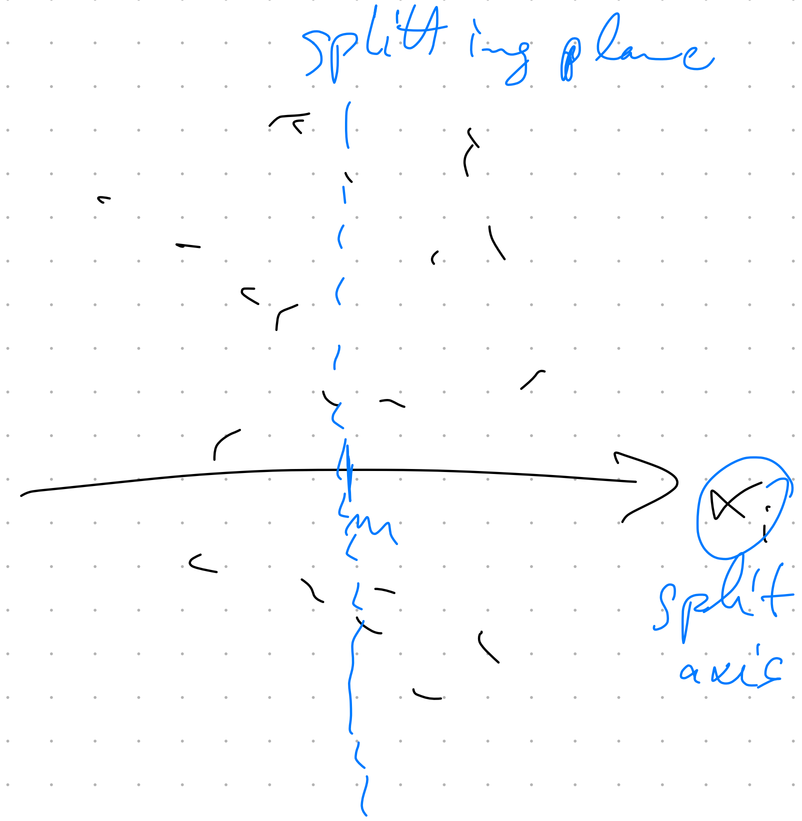
$P^+ = \{p \in P \mid p_i \geq m\}$

recursion with P^-, P^+

create node v:

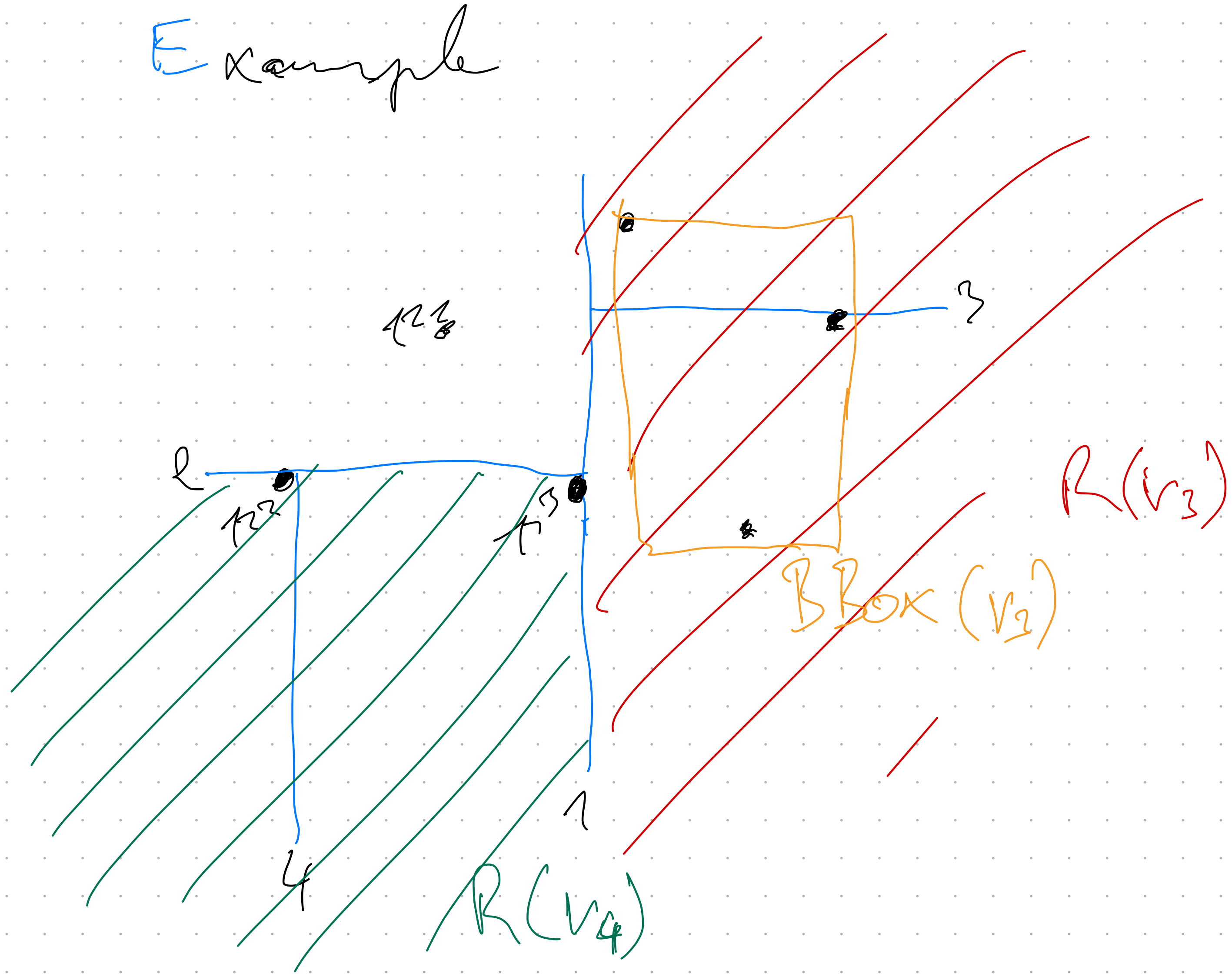
- pointers to kd-trees over P^- and P^+
- median m and axis i
- (optionally bbox(P))

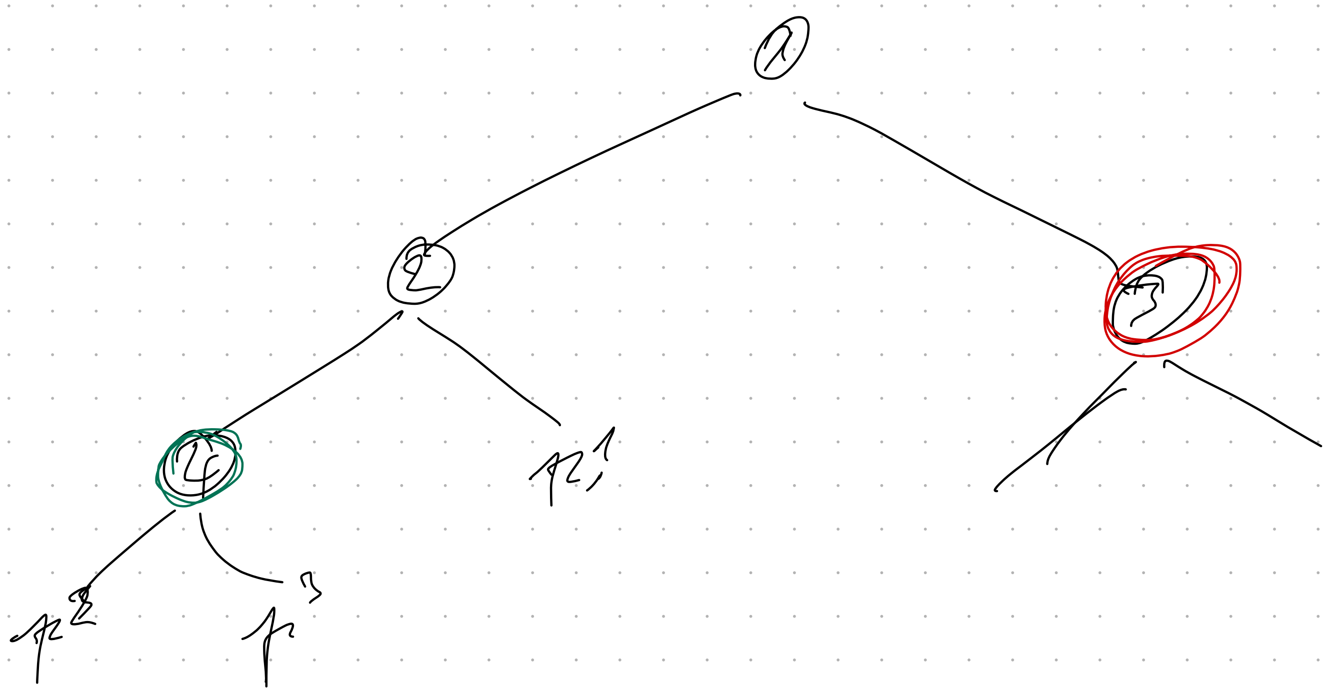
stop when $|P| = 1$



Note: m, i define a plane perpendicular to i-th coord axis

Example





Terminology:

Region of node v $\quad R(v) \subseteq \mathbb{R}^d$

$R(\text{root}) = \mathbb{R}^d$; in practice, start with BBox(P)

Derive $R(v_\ell)$ from $R(v)$ $\quad$ [v_ℓ = left child of v]
during traversal

$R(v) = [x_{min}, x_{max}, y_{min}, y_{max}]$ $\leftarrow$ split axis $= x$, m = median

$R(v_\ell) = [x_{min}, m, y_{min}, y_{max}]$

$R(v_r) = [m, x_{max}, y_{min}, y_{max}]$

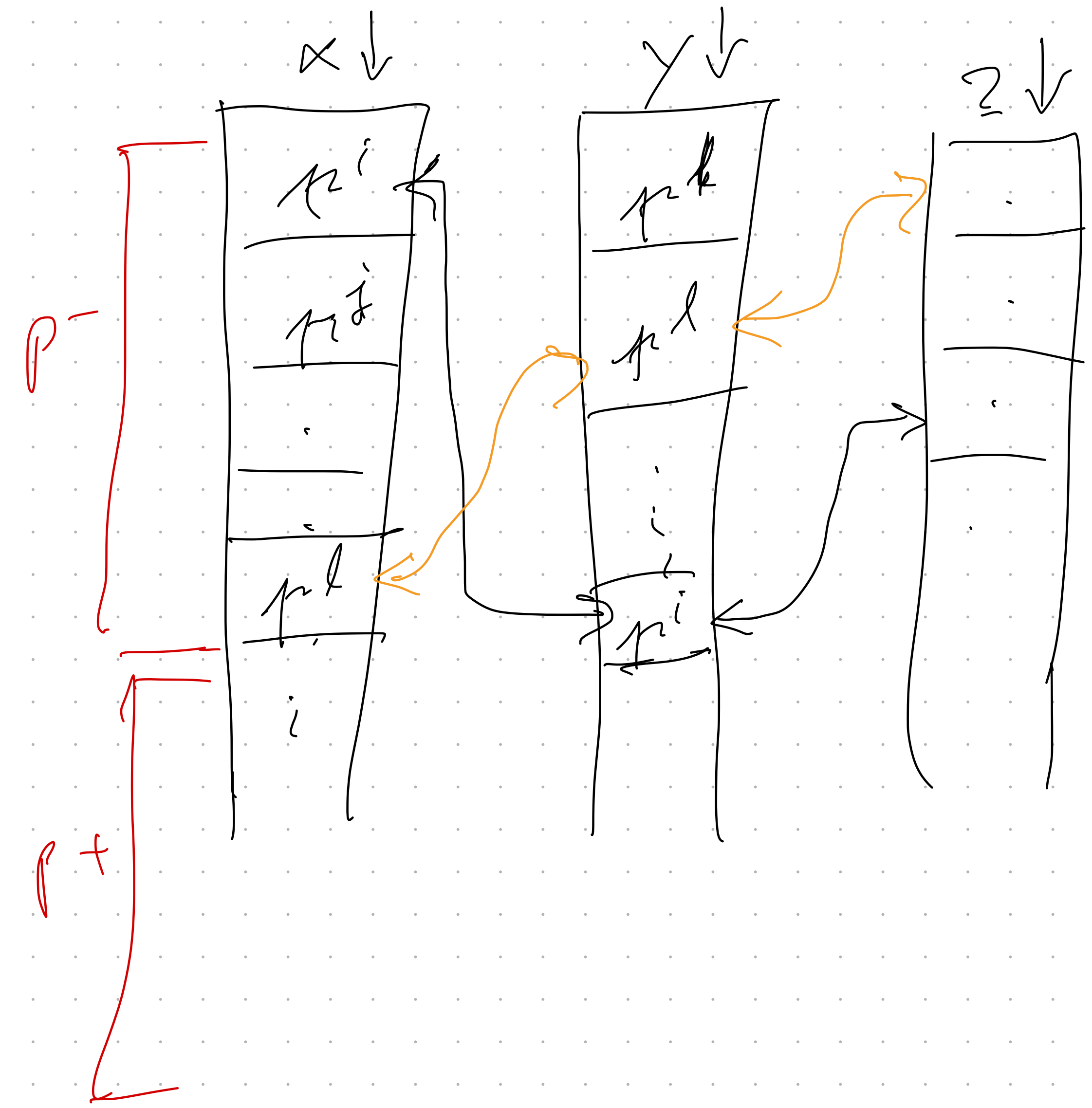
$BBox(v) = bbox(P(v))$, where $P(v)$ = pts inside $R(v)$

(implementing construction: (in 2D for simplicity)

pre-sort P once along x $\rightarrow$ "x-list"

and " y $\rightarrow$ "y-list"

introduce pointers between the lists



Observe:

- finding median $\in O(1)$
- memory $\in O(d \cdot n)$, d = dimension
- splitting $\in O(d \cdot n)$

Naive splitting: $O(n \log n)$

Complexities:

Depth: $O(\log n)$

Preprocessing: $O(d \cdot n \log n)$, d = dim.

Recurrence:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \underbrace{O(dn)}_{\text{or } O(n \log n)}$$

$$\Rightarrow T(n) = O(d \cdot n \log n)$$

$$\text{or} = O(d n \log^2 n)$$

Difference to quadtrees:

- qtree: size of nodes decreases exponentially with level
- kd-tree: size of $P(v)$ — " —

Variants:

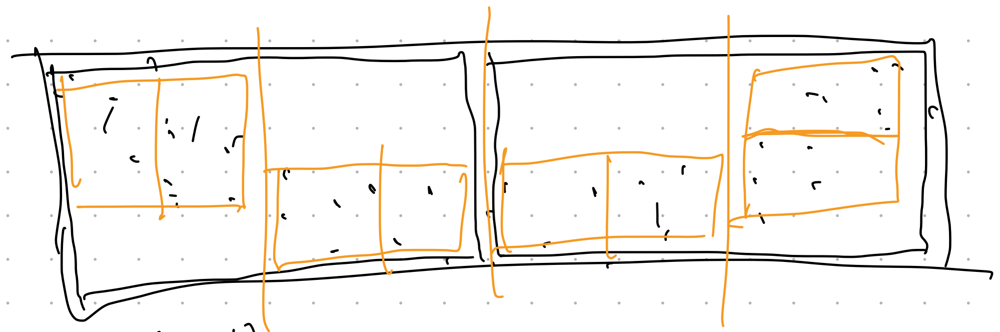
0. Store median pt $\bar{p}$ in node v $\quad (P = P^- \dot\cup P^+ \dot\cup \{\bar{p}\})$

1. "Binning": stop recursion when $|P| \le b$, $b \approx 20, \dots, 30$

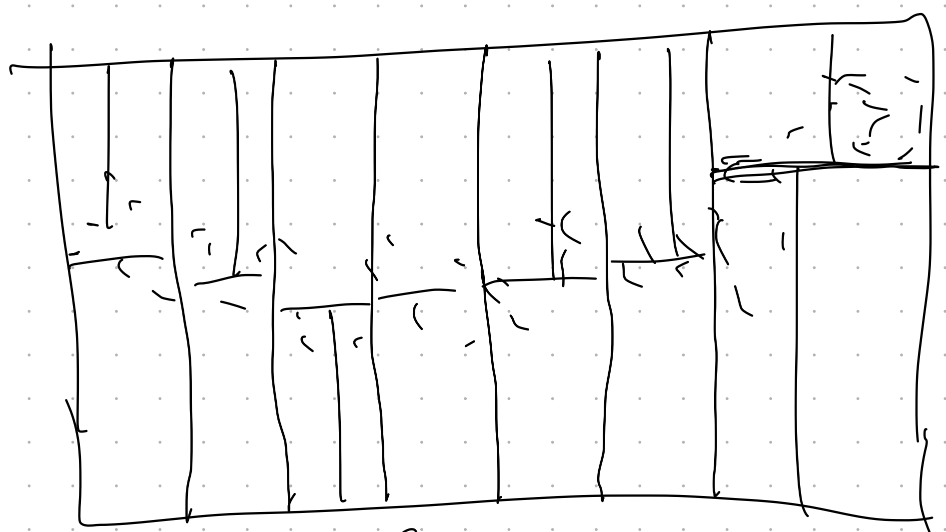
2. Split plane:

   a) trivial: round-robin
      first x axis, then y, z, x, ...

   b) "largest spread kd-tree":
      choose axis, where bbox(P(v)) has largest extent
      Problem with P lying on smaller-dim manifold:



Bbox(root)



Regions of nodes

c) "longest side kd-tree":

choose axis where $R(v)$ has longest extent.

$\Rightarrow$ seems best so far in most cases

Application: Nearest-Neighbor Problem, a.k.a. Closest Point Problem

Input: set pts $P \subseteq \mathbb{R}^d$

query pt $q \in \mathbb{R}^d$

Output: $p^* \in P$, such that $\forall p \in P: \|p^* - p\| \leq \|p - q\|$

$\uparrow$ "nearest neighbor"

Algo: $NN(v, p, r) \rightarrow p', r$

input v = node

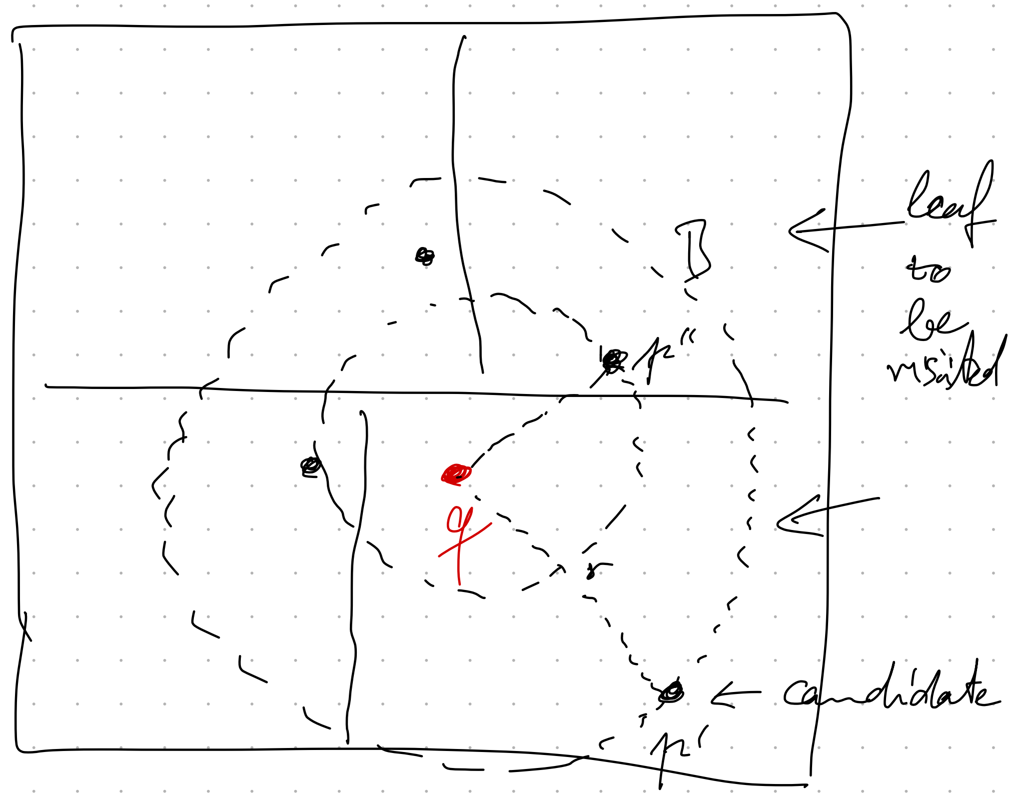
p = current candidate for p^*

$r = \|p - q\|$

output new candidate p', r

precondition: $B(q, r)$ overlaps $R(v)$

"ball with radius r, center q"



if v is leaf:

p' = nearest neighbor of $P(v)$ to q

$r' = \|p' - q\|$

if $r' < r$ then return p', r'

else return p, r

else: (v not leaf)

if $q_i \leq m$: (m, i = splitting plane)

$p, r = NN(v_\ell, p, r)$

if $B(q, r)$ overlaps $R(v_r)$: "bounds overlaps ball test"

$p, r = NN(v_r, p, r)$

else ($q_i > m$):

$p, r = NN(v_r, p, l)$

if $B(q, r)$ overlaps $R(v_\ell)$:

$p, r = NN(v_\ell, p, r)$

end if

if $B(q, r) \subseteq R(v)$:

"ball within bounds test"
(in practice: not necessary)

$p^* = p$

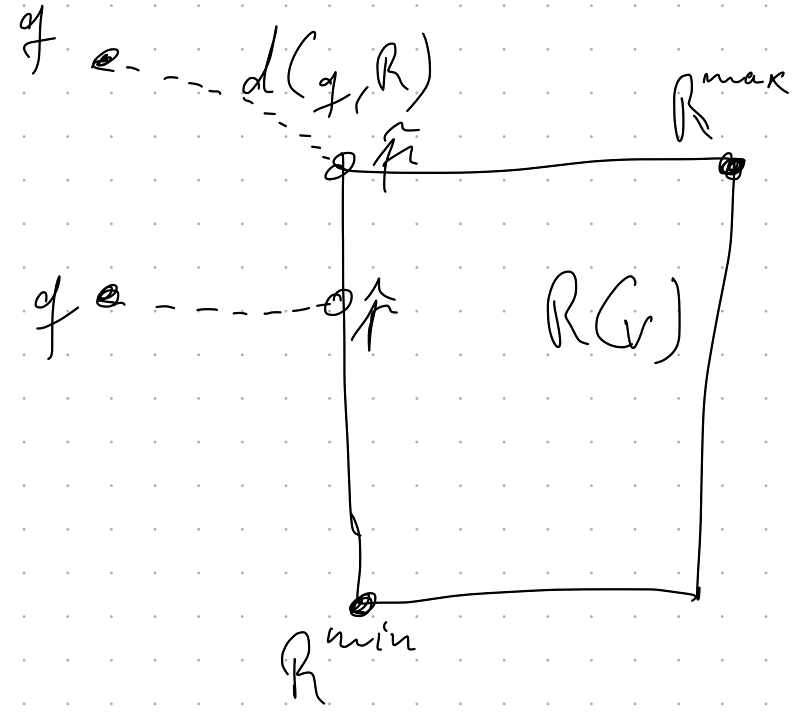
stop recursion

return p, r

Init: $NN(\text{root}, \text{NULL}, \infty)$

Analog: "farthest neighbor"

Implementation "bounds overlaps ball":



$B(q, r)$ overlaps R $\Leftrightarrow$

$d(q, R) < r$

$d(q, R) = d(q, \hat{p})$

$\hat{p} = (\hat{p}_1, \ldots, \hat{p}_d)$

where

$$\hat{p}_j = \begin{cases} R_j^{min}, & q_j < R_j^{min} \\ q_j, & R_j^{min} < q_j < R_j^{max} \\ R_j^{max}, & q_j > R_j^{max} \end{cases}$$

Test $\in O(d)$

Running time:

obvious $T(n) \in \Omega(\log n)$, $T(n) \in O(n)$

No better bounds for worst-case

Under certain assumptions about distrib. pts:

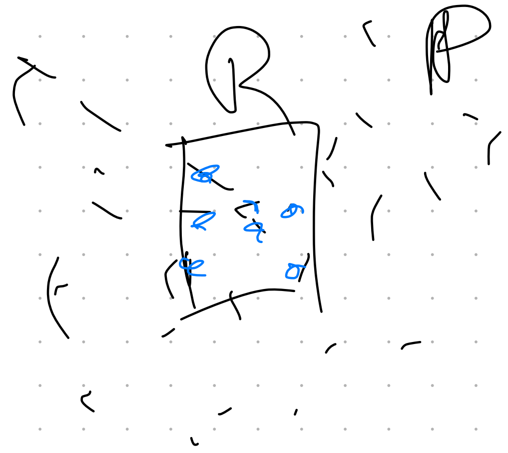
expected $T(n) \in O(\log n)$

Curse of Dimensionality

Lemma (w/o proof):

1. Given set of pts $P \subseteq \mathbb{R}^d$, $|P| = n$.
   A kd-tree over P allows for orthogonal range queries in time $O(n^{1-\frac{1}{d}} + k)$, $k = \#$ output pts.

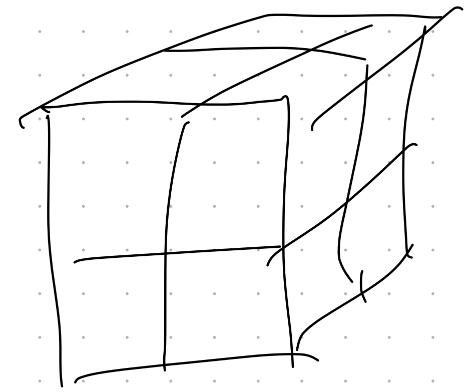
   Def.: orthogonal range query
   $R = [a_1, b_1] \times [a_2, b_2] \times \dots \times [a_d, b_d]$.
   Find all pts $p \in P$, s.t.: $p \in R$.



2. Any algorithm solving orthogonal range queries using a data structure of size $O(n)$, must have running time $\in \Omega(n^{1-\frac{1}{d}} + k)$.
   In that sense, kd-trees are optimal for orth. range queries.

Another thought experiment:

Consider $N = 10^7$ pts, uniformly distributed in cube $\subseteq \mathbb{R}^d$.



Partition cube into "octants"

$c = 2^d$ = # octants

empty octants $e \geq \frac{c-N}{c}$

expected pts per octant $p = \frac{N}{c}$

d	p	e
10	$9.8 \cdot 10^3$	~0%
30	0.009	99.1%
100	$8 \cdot 10^{-24}$	~100%

Consider hyperballs $B_d \subseteq \mathbb{R}^d$:

$$Vol(B_d) = r^d \cdot \frac{\pi^{d/2}}{(d/2)!}$$, d even , r = radius

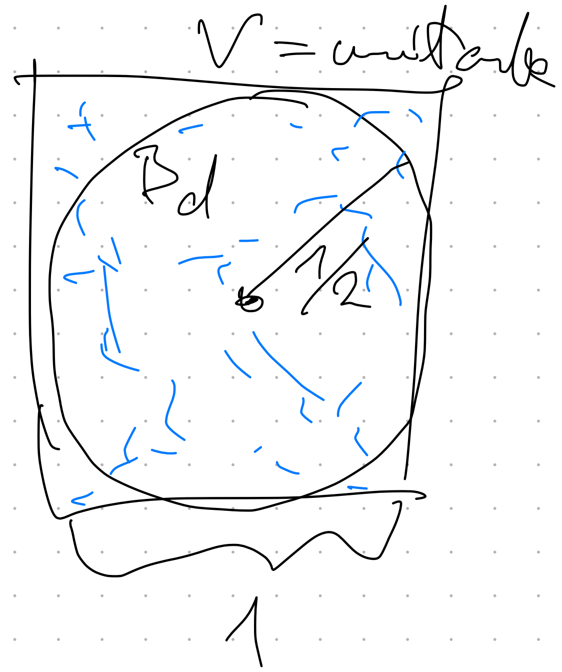
Reminder: let $R \subseteq V \subseteq \mathbb{R}^d$

distribute ~~set~~ pts $P \subseteq \mathbb{R}^d$ uniformly, randomly in V

expected pts inside R = $\frac{Vol(R)}{Vol(P)} \cdot |P|$

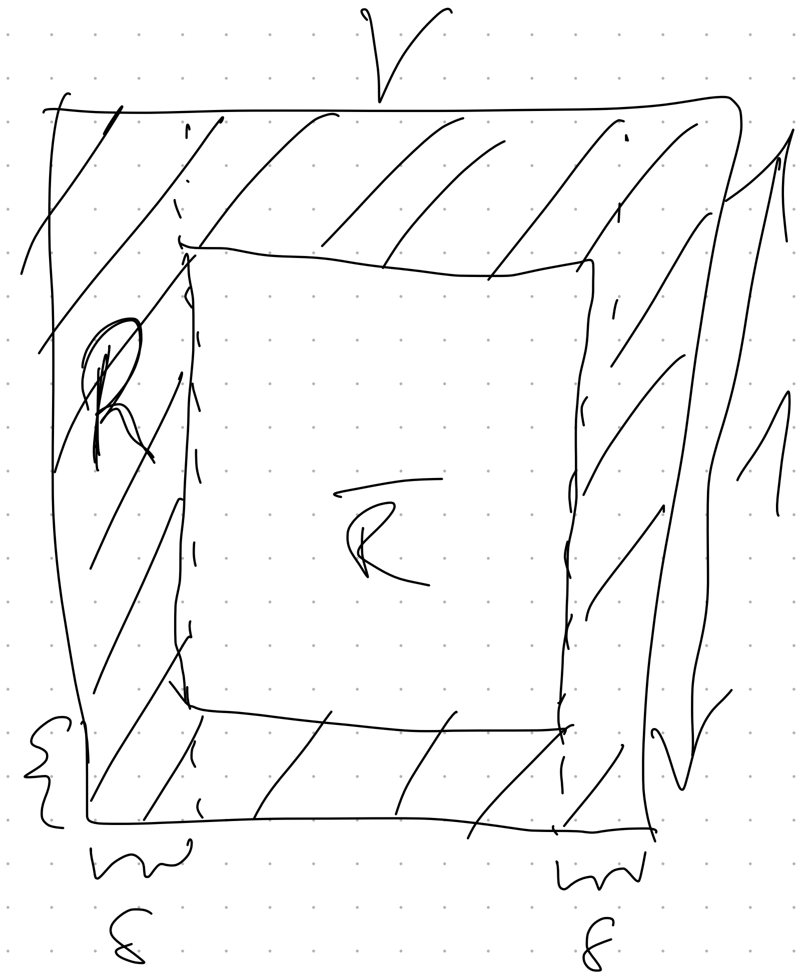
$$\frac{Vol(B_d)}{1^d} = \frac{(r^2\pi)^{d/2}}{(d/2)!} = \frac{(0.78)^{d/2}}{(d/2)!} \longrightarrow 0$$

as $d \to \infty$



Consider hypercube shell:

$$\mathrm{Vol}(R) = 1 - \underbrace{(1-2\delta)^d}_{\mathrm{vol}(\bar{R})}$$



Prob(random pt $\in R$ | pt $\in V$)

= Prob(dist pt from surface of $V \leq \delta$)

choose $\delta = 0.1$

d	Prob
3	49%
10	89%
30	99.8%

Never compare $Vol(B_d)$ with $Vol(B_{d-1})$!

Think "n" versus "n^2"

Approximate Nearest Neighbors

Def.:

Let $P \subseteq \mathbb{R}^d$ be set of pts ;

$q \in \mathbb{R}^d$ = "query pt" ;

assume $p^* \in P$ is the NN ; given $\varepsilon > 0$.

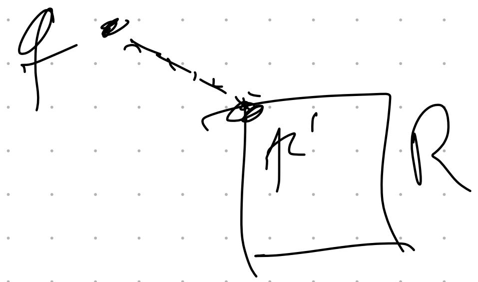
Then, $p^\circ \in P$ is called a "$(1+\varepsilon)$-approximate nearest neighbor"

$\iff \quad d(p^\circ, q) \leq (1+\varepsilon) \cdot d(p^*, q)$.

Assume, d is a metric, e.g. $d(p,q) = \|p-q\|_2$

Notation: let v = node, then

$d(q, v) = d(q, R(v))$,



Algorithm: ANN

Q = p-queue with pointers to nodes v in kd-tree, sorted by $d(q, v)$

p° = current candidate

init $p^\circ :=$ "infinite pt" (very important)

$v :=$ root

$Q :=$ empty

while $d(q, v) < \frac{1}{1+\varepsilon} d(q, p^\circ)$:

  while v is inner node:

    let v_1, v_2 = children of v, assume $d(q, v_1) \le d(q, v_2)$

    insert v_2 in Q

    $v := v_1$

  end while

  if $d(q, p_v) < d(q, p^\circ)$:

    $p^\circ := p_v$

  $v :=$ extract-min(Q)

end while

return p°

Remark: $\varepsilon = 0 \Rightarrow p^\circ = p^*$

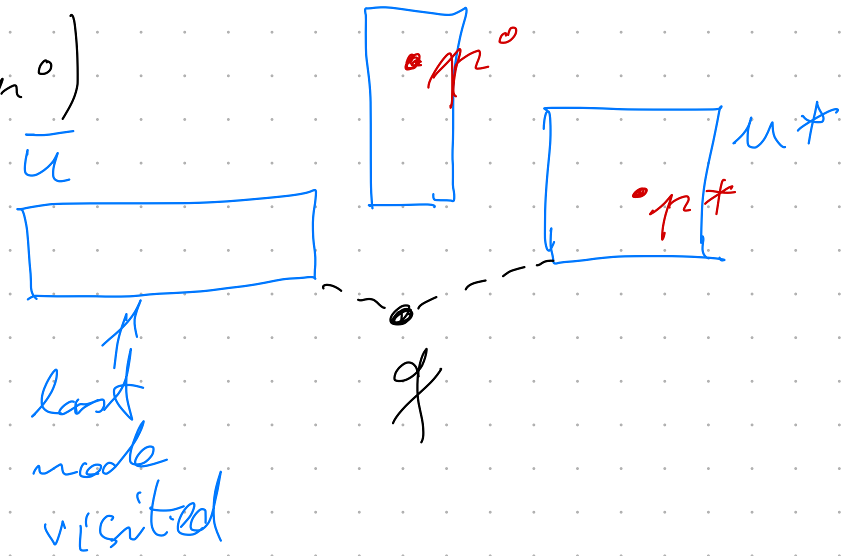
Correctness:

Let $u^* =$ leaf containing $p^* = NN$

a) Case u^* is visited $\Rightarrow$ algo returns $p^\circ = p^*$

b) Case u^* is not visited $\Rightarrow p^\circ \neq p^*$

$$\underbrace{d(q, p^*) \geq}_{\text{metric}} \; \underbrace{d(q, u^*) \geq}_{\text{closer nodes are visited first}} \; \underbrace{d(q, \bar{u}) \geq}_{\text{stop criterion}} \frac{1}{1+\varepsilon} d(q, p^\circ)$$



$$\Rightarrow d(q, p^*) \geq \frac{1}{1+\varepsilon} d(q, p^\circ)$$

Complexity:

\# outer iterations $= l =$ \# visited leaves

\# iterations in inner while loop $= O(\log n)$

Op's in outer loop: 1x extraction from p-queue $\Rightarrow O(\log n)$

1x inner loop

op's in inner loop: 1x insert in queue $\rightarrow O(\log n)$

time for inner loop $\in O(\log^2 n)$

Total time $\in O(l \cdot \log^2 n)$

Improvement: use Fibonacci heap

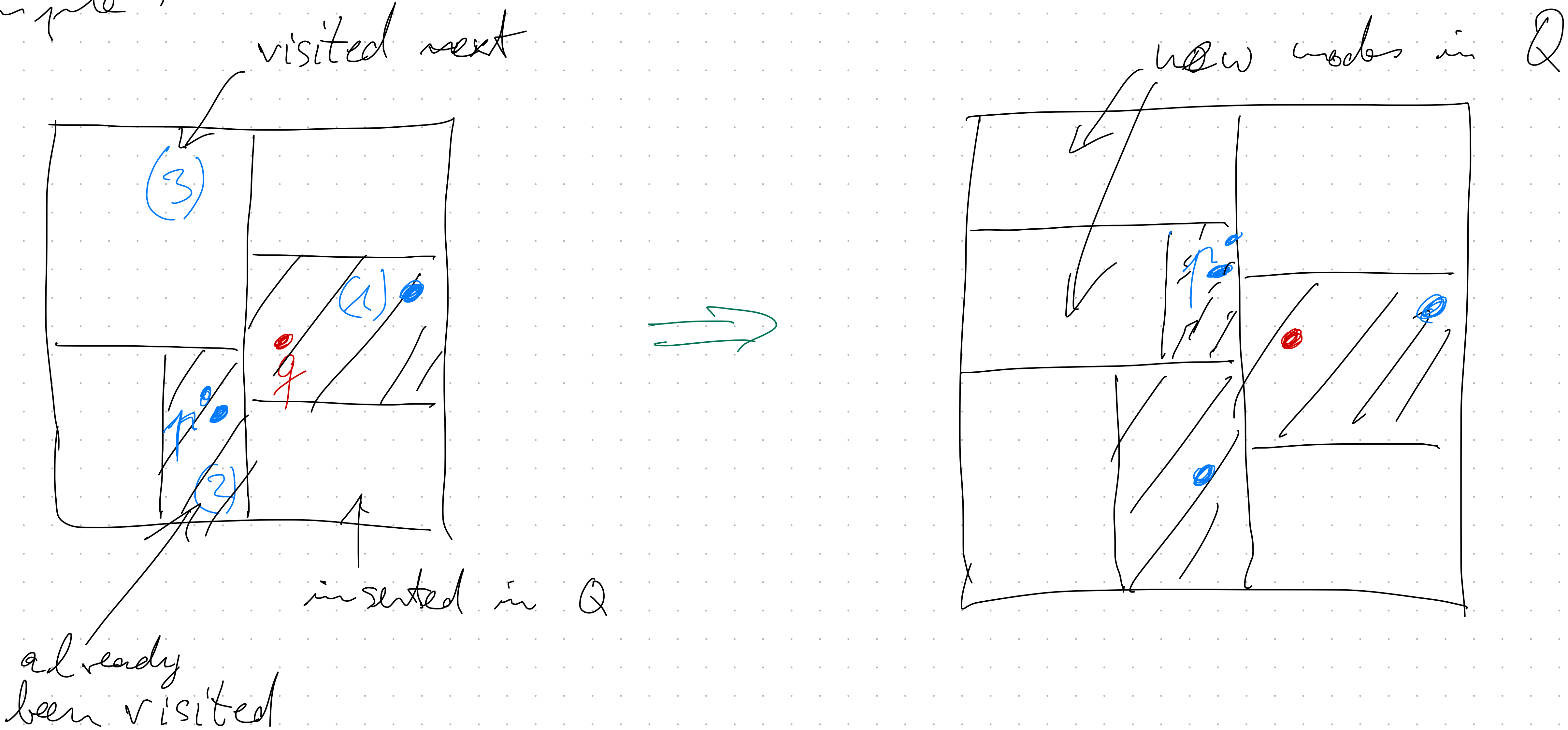
$\Rightarrow O(\log n)$ for extract op.

$O(1)$ amortized time for insert op.

$\Rightarrow$ total time for ANN $\in O(l \cdot \log n)$

Dist. calc. $\in O(1)$ / $O(d)$

Example:



Notes:

- In practice: Q remains small → use regular heaps
- Analog: "$(1-\varepsilon)$-farthest neighbor"

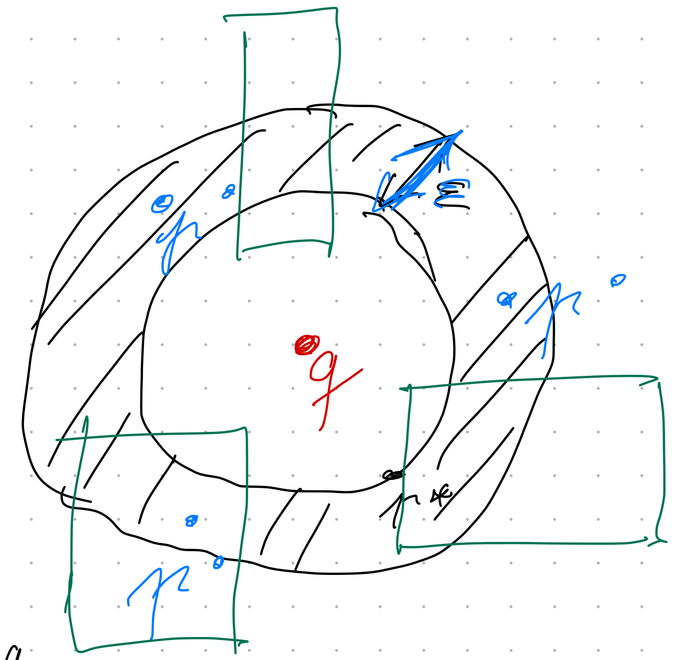
- In dimension d:

  Show # leaves visited $\ell \in O(\log^{d-1} n)$

  Approach to proof: find upper bound on # nodes stabbing an annulus around q

  $\Rightarrow O\left(\left(\frac{\log n}{\varepsilon}\right)^{d-1}\right)$

  ! works only for "longest-side kd-trees"!



Theorem:

Let $P \subseteq \mathbb{R}^d$ pt set, $|P| = n$, and query pt $q \in \mathbb{R}^d$.

Then, finding an ANN is possible in time $O\left(d \cdot \frac{\log^d n}{\varepsilon^{d-1}}\right)$.

"Best" ANN Algorithms

Another quality criterion for ANN algos:

$$\text{precision} = \frac{\#\text{ exact NN's returned}}{\#\text{ queries}}$$

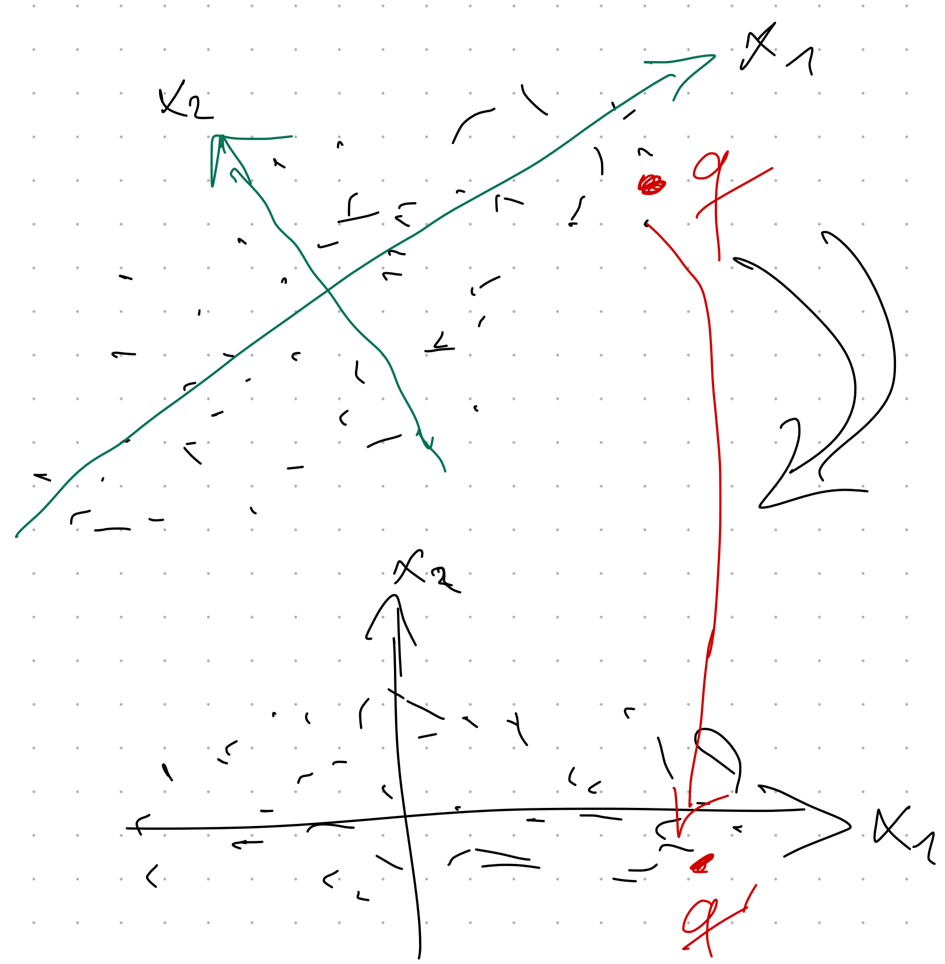
$$\text{"error"} = 1 - \text{precision}$$

Randomized kd-tree (RKD):

- Pick $D \leq d$ axes (dim's), e.g., the ones with highest variance among P
- Choose one of those randomly
- Split across median along that axis

PCA-RKD:

- Determine principal components of P
- Transform $P \rightarrow P'$
- Construct RKD over P'
- Transform $q \rightarrow q'$
- Continue with standard ANN algo

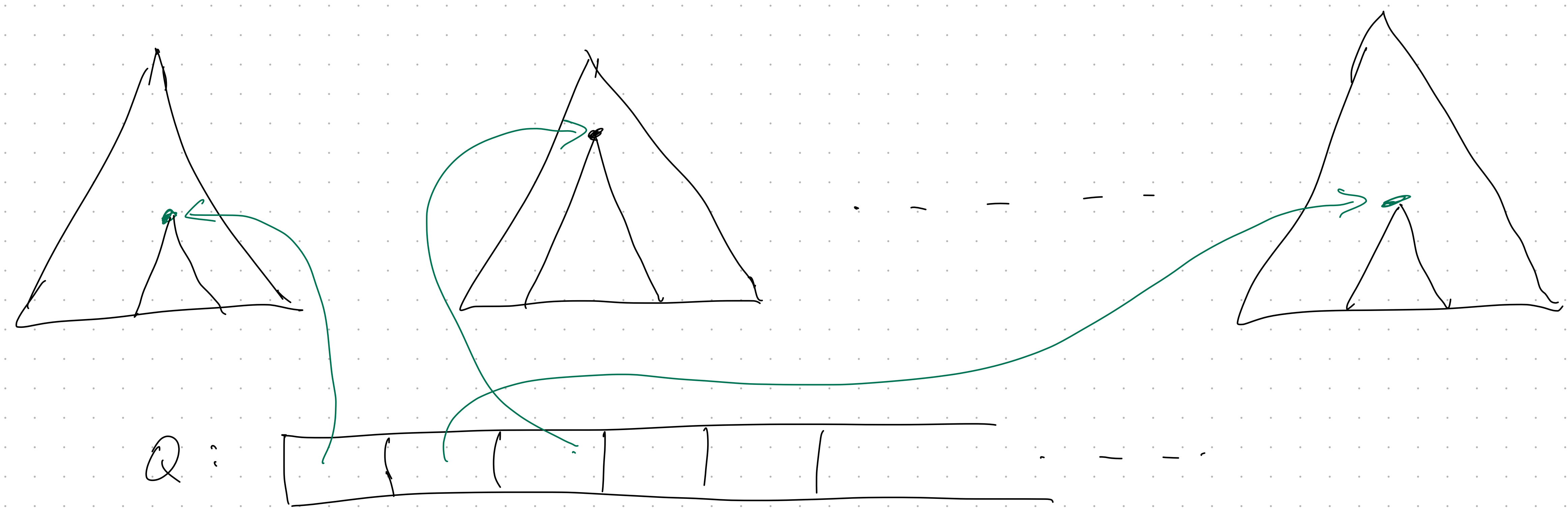


RKD-Forest (pot'ly with PCA):

- Construct N RKD-Trees over P (or P')
  each one with different subset of candidate splitting axes

ANN Search using RKD Forest:

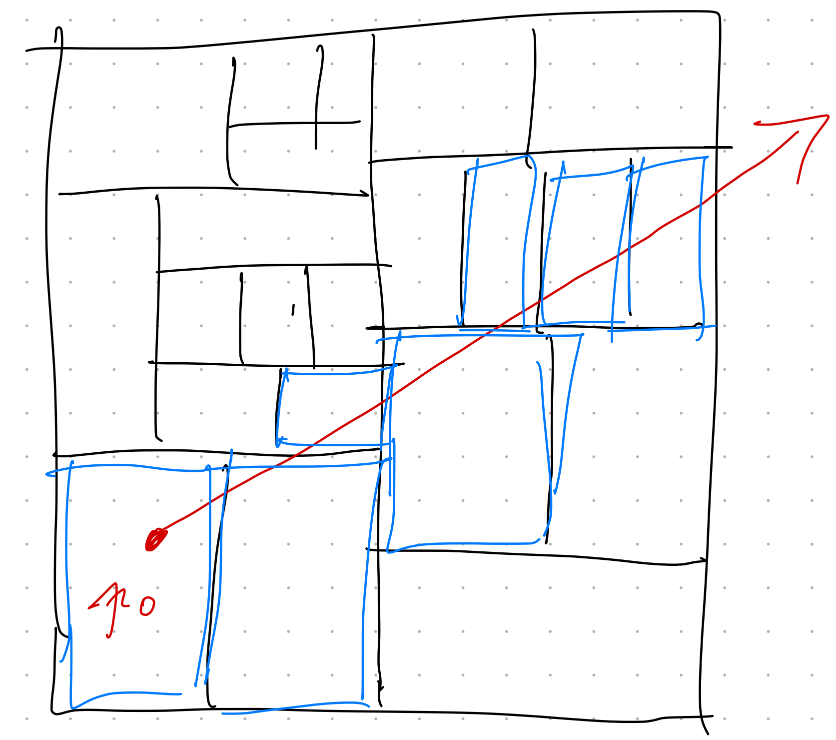
- Maintain <u>one</u> p-queue Q
- Init: descend each RKD-Tree down to closest leaf to q
- Choose p of all those leaves closest to q,
  put others in p-queue
- Proceed with ANN algo as before
  (only difference: Q points to various kd-trees)



K-Means-Tree : works using clustering
could randomize

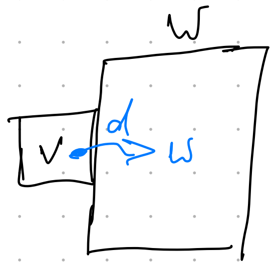
Raytracing using Stackless kd-tree Traversal

Goal: given kd-tree,
given ray;
find all leaves "along" ray



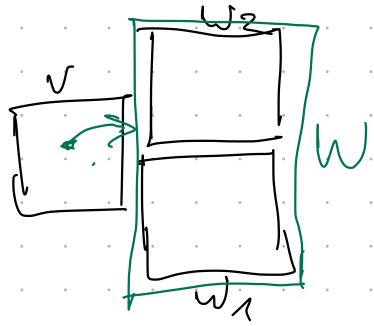
Def.:

1. Direction: $D = \{L, R, T, Bo, F, Ba\}$
   = "directions"
2. Opposite direction: let $d \in D$, then
   $\bar{d}$ = "opposite";
   ex. $d = L \Rightarrow \bar{d} = R$
3. Rope: let v = node of kd-tree.
   Denote by $v \xrightarrow{d} w$ a pointer from v to w, and w is in direction d relative to v, where
   a) in case v has just one neighbor in direction d:
      w = that one neighbor
   b) in case v has several neighbors in dir d:
      $w_1, \dots, w_n$

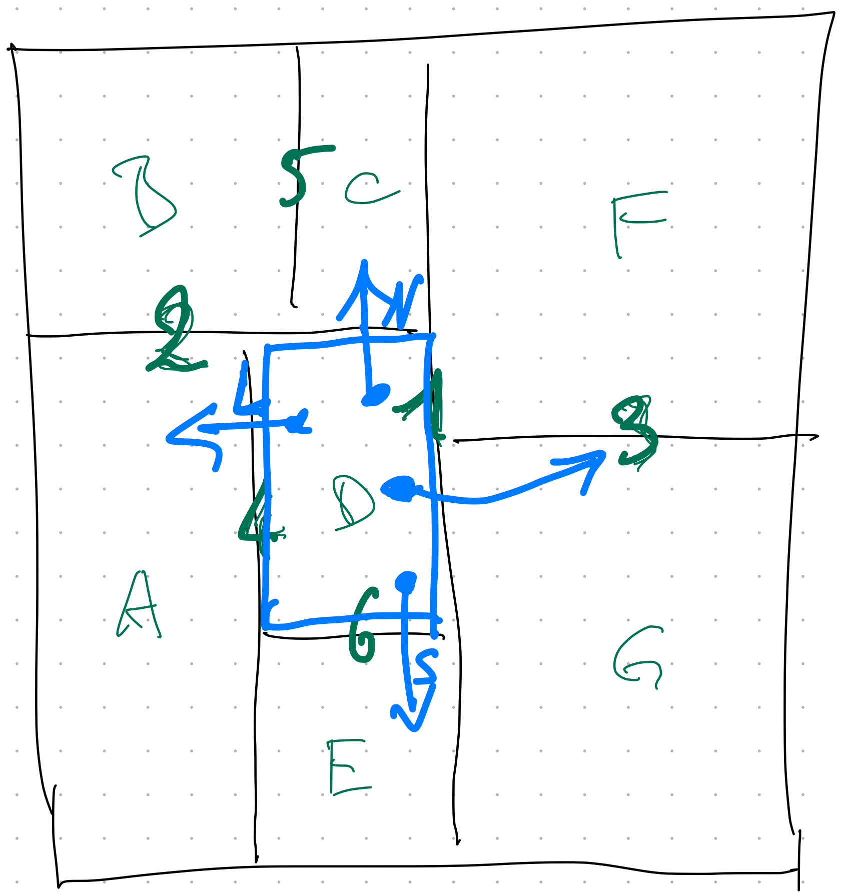


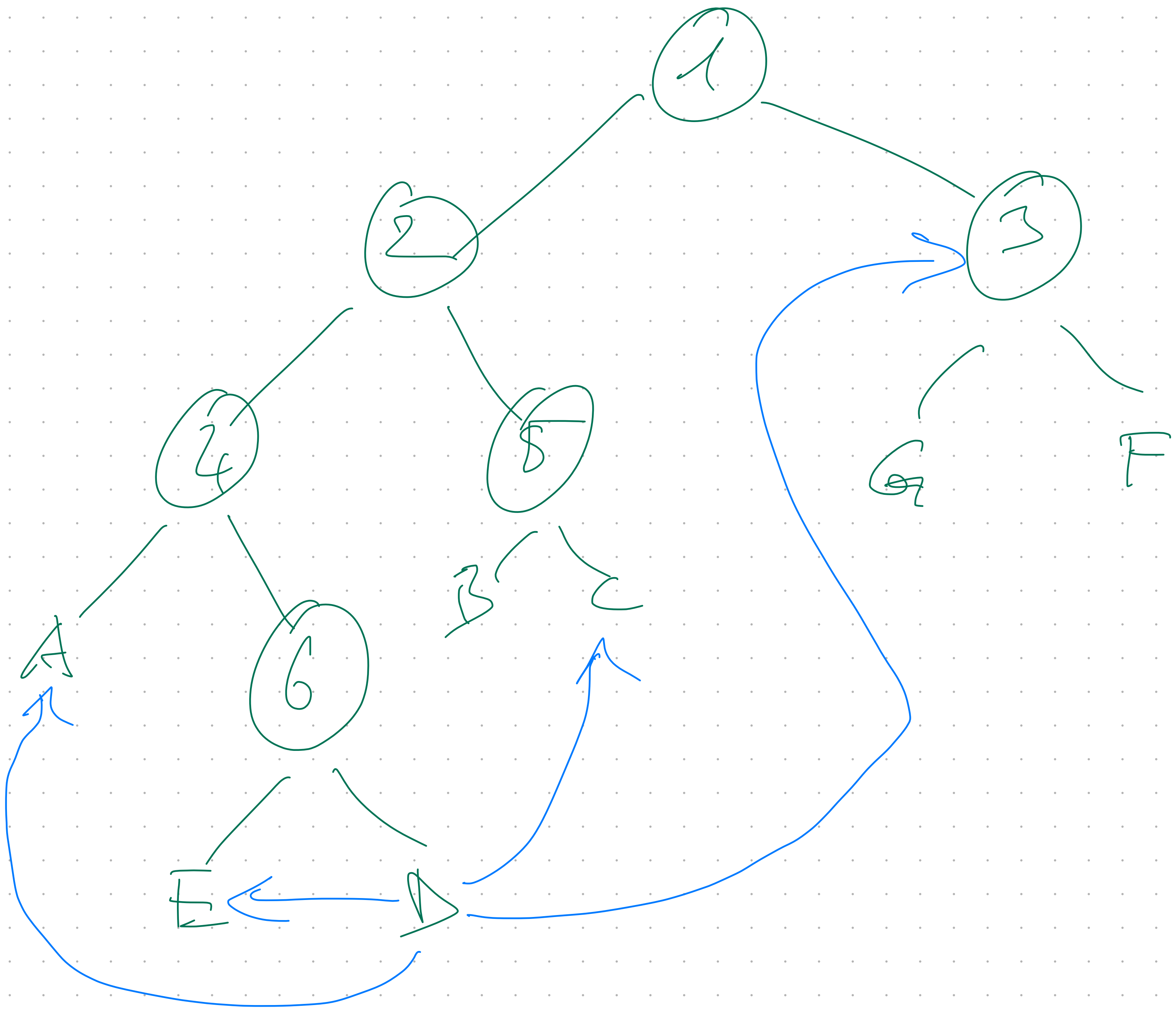
$w =$ common ancestor of $w_1, \dots, w_m$

[d-side of $R(w)$ has to contain the d-side of $R(v)$]



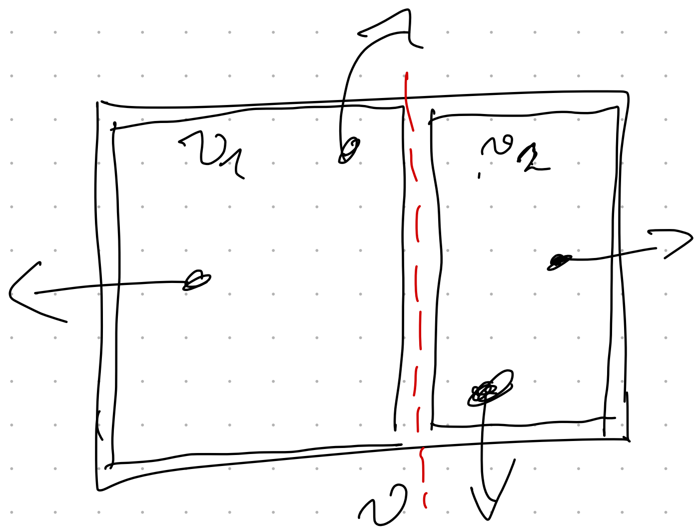
Example:

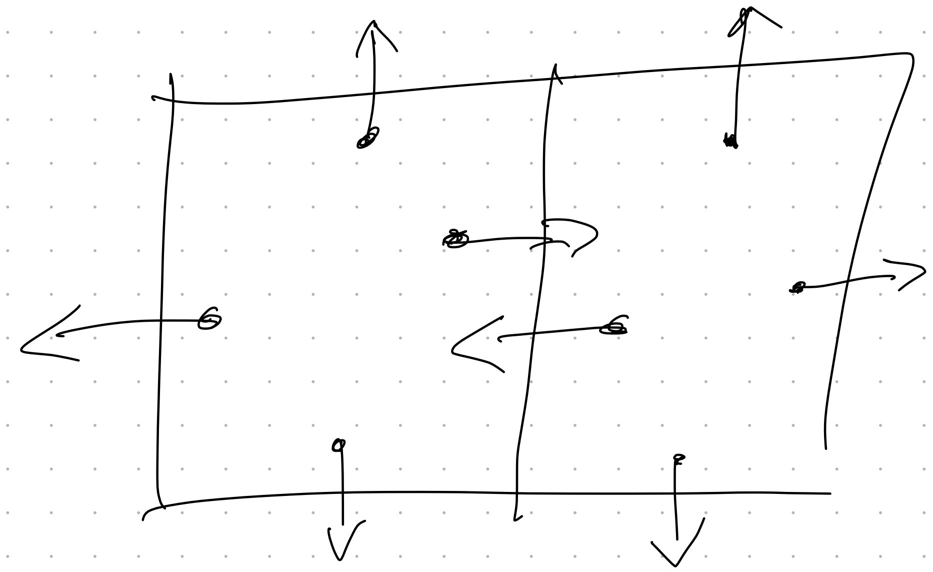




Task: Introduce ropes

Approach: propagate ropes top-down





Aux algo: replace rope $v \to w$ by pointers to children of w, if possible

pushDown(v, w, d):

  input: rope $v \xrightarrow{d} w$, d = dir. of rope

  output: new rope to child w_1

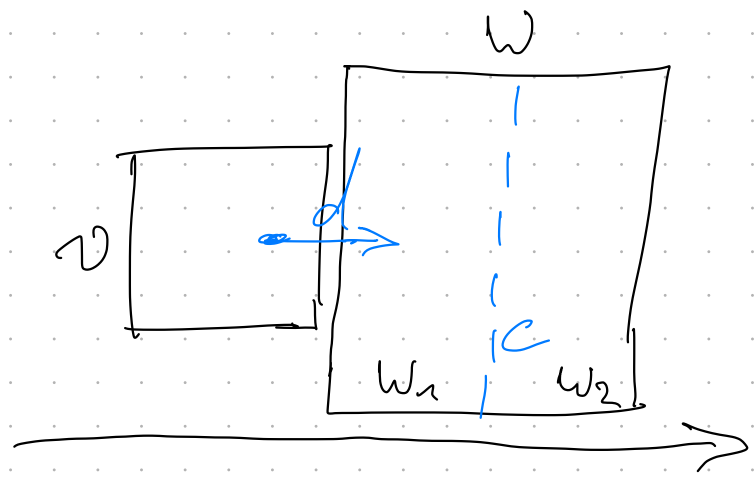
  let c = clipping plane of w

  if $c \perp d$:

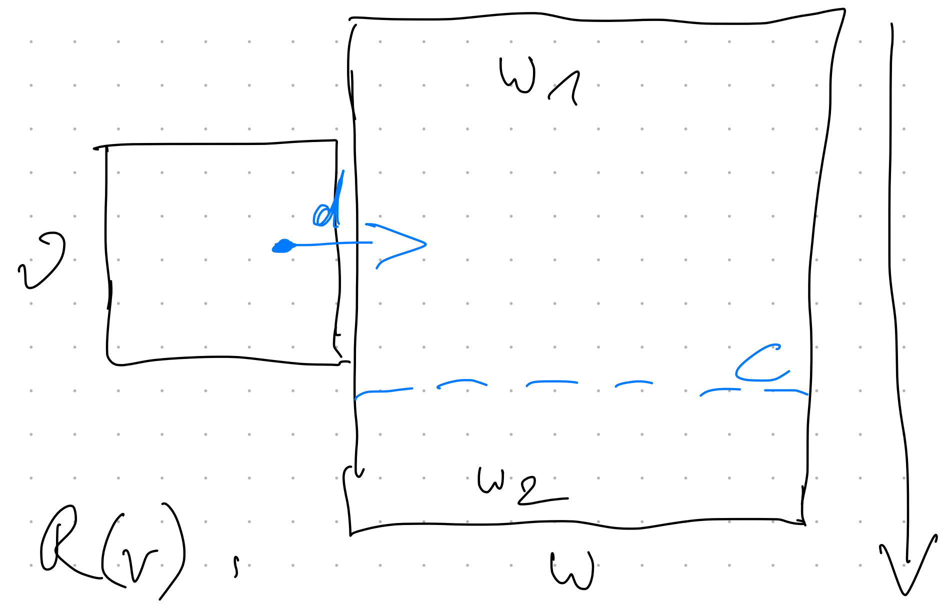
    if $d \in \{R, T, Ba\}$:

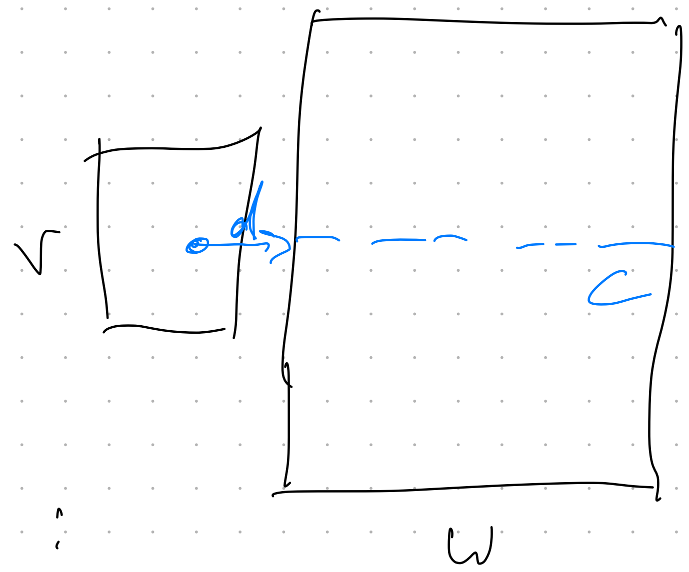
      return w_1

  else:



```
        return w_2
else:   // c parallel to d
    if side of R(w_1) in dir d contains
       side of R(v):
        return w_1
    if side of R(w_2) in d contains side of R(v):
        return w_2
return w   // no further propagation possible
```





Algo for introducing ropes in existing kd-tree:

```
propagateRopes(v):
  if v is leaf:
     return
  for all d ∈ D:
     if rope v -d-> w exists:
        repeat:
           w' = w
```

$w'' = \text{pushDown}(v, w', d)$

until $w'' == w'$

set new rope $v \xrightarrow{d} w'$ for node v

let c = splitting axis of v, $c \in \{X, Y, Z\}$

$$d = \begin{cases} R, & c = X \\ T, & c = Y \\ F, & c = Z \end{cases}$$

v_1 = left child, v_2 = right child

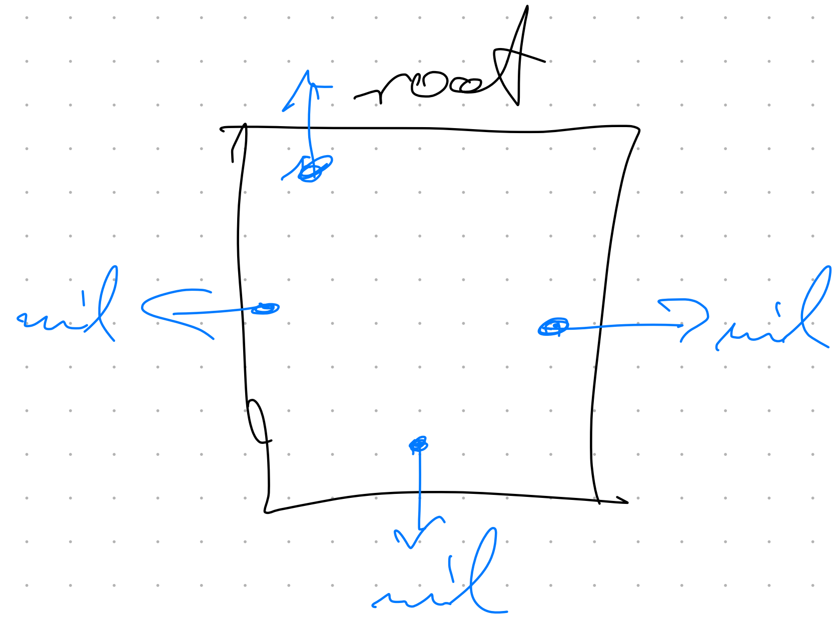
set ropes of v_1, v_2 pointing to v_2, v_1 respectively

$v_1 \xrightarrow{d} v_2$, $v_2 \xrightarrow{\bar{d}} v_1$

copy ropes of v to v_1, v_2 // "outward" pointing ropes

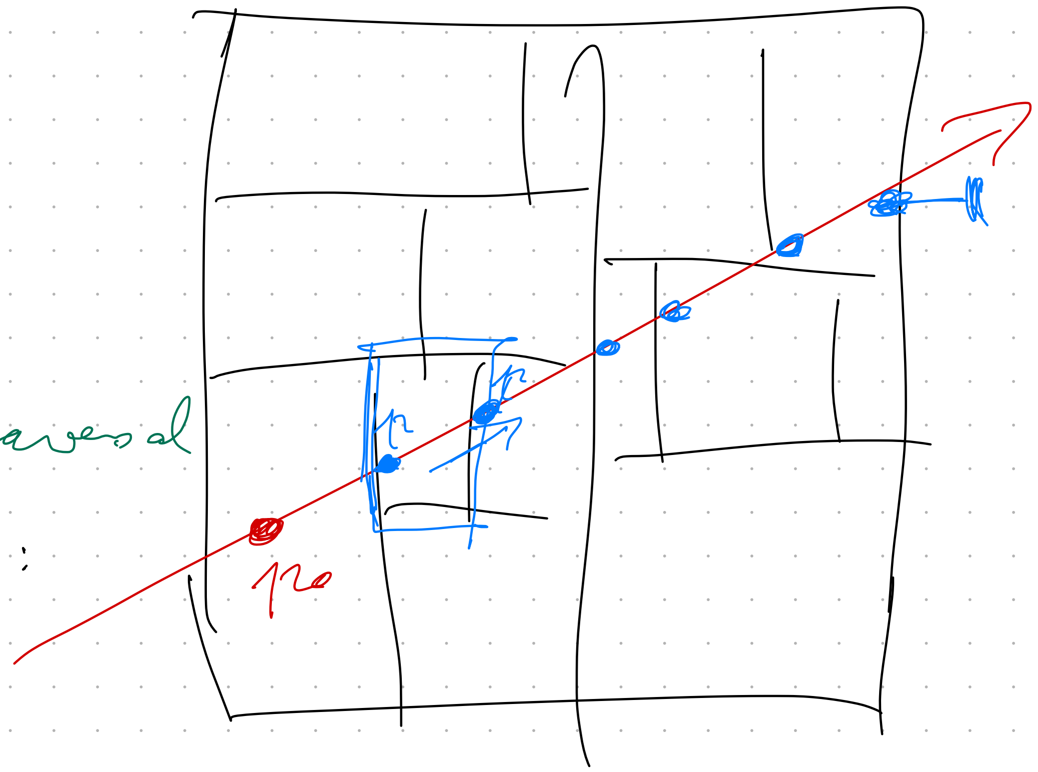
Start: set ropes of root to nil

propagateRopes(root)



Kd-Tree Traversal using "roped kd-tree"

p_0 = start pt of ray
v = root
$p = p_0$
while $v \neq$ nil :
    while v is not leaf : // down traversal
        if p is left of splitting plane :
            $v = v_1$
        else:
            $v = v_2$
    end while
    find p' = closest intersection pt with geom in v // if any
    if p' exists and $p' \in R(v)$:
        return p'
    find "exit wall" of v , "exit pt" p
    follow rope of v in direction exit wall
    if rope $v \xrightarrow{d} w$ exists :
        v := w
end while
return "no intersection"



Q for you: prove anything for rsoped kd-trees ?
in case "longest side" construction

Texture Synthesis

Problem: Given input texture I

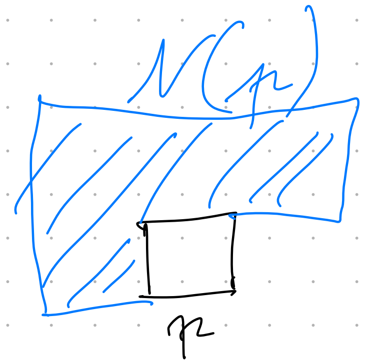
Construct larger/different output texture T

that <u>looks similar</u>

Def.: p_i = input pixel $\in I$

p_o = output $\in T$

$N(p)$ = neighborhood around p



Algo:

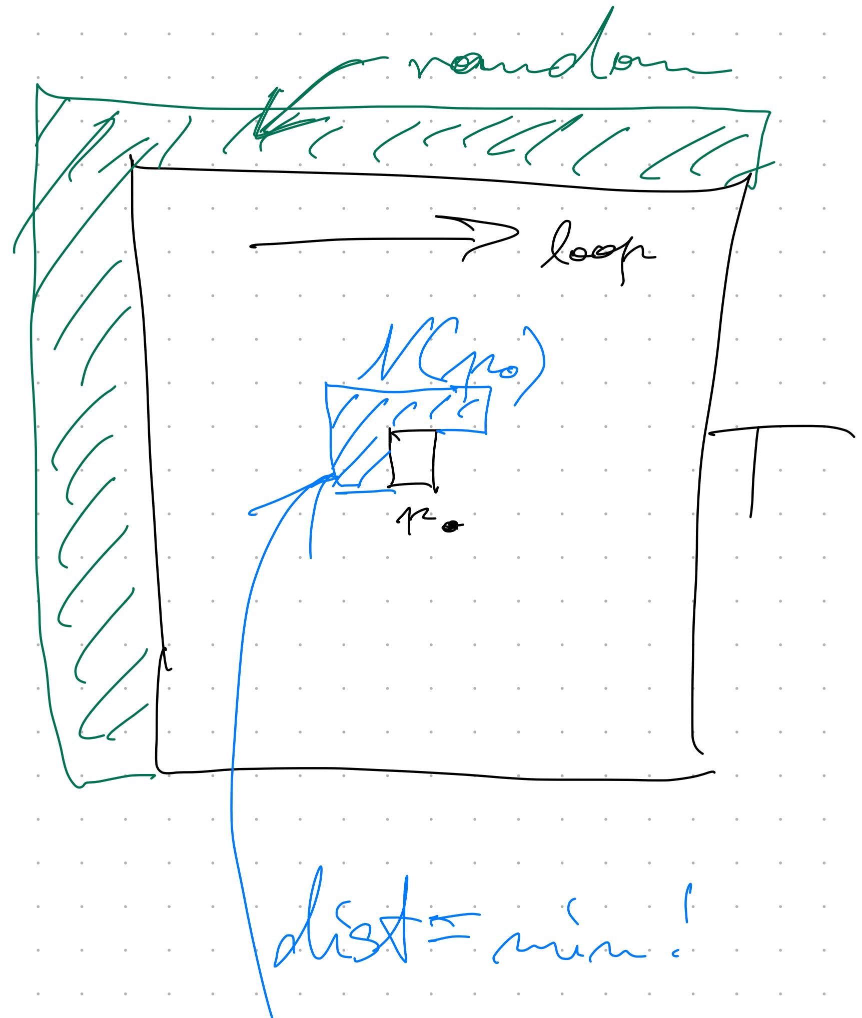
init T with random border

for all $p_o \in T$ in scanline order:

find $p_i \in I$ such that

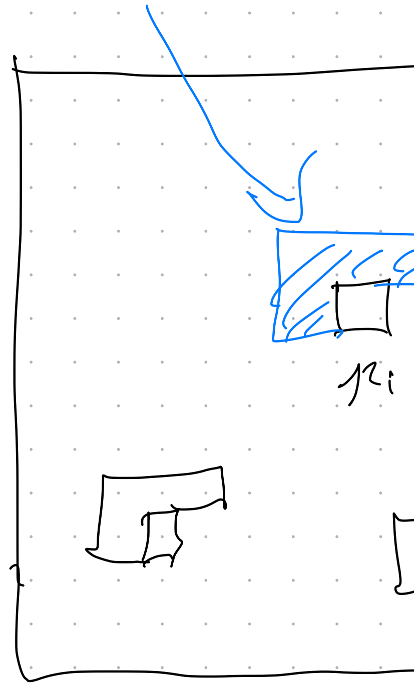
$$\text{dist}(N(p_o), N(p_i)) = \min \quad (*)$$

$p_o := p_i$



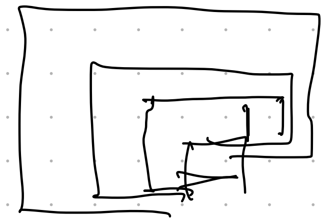
$(*)$ is NN search!

→ use kd-tree
with some metric / similarity
over $N(p) = \begin{pmatrix} r_1 \\ g_1 \\ b_1 \\ r_2 \\ g_2 \\ b_2 \\ \vdots \end{pmatrix}$



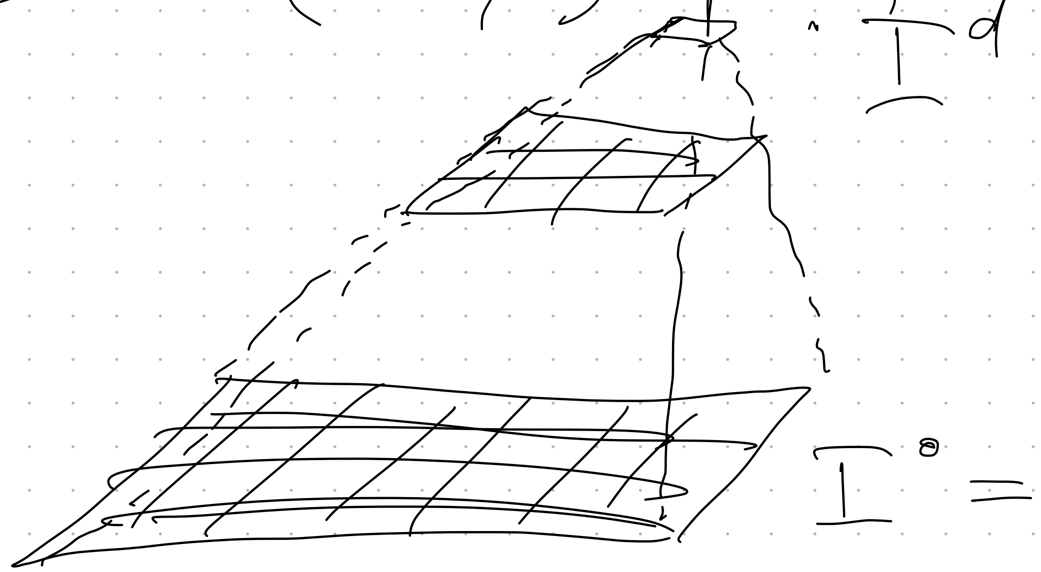
I

Solution for "proper" size of $N(p)$?
→ image pyramid



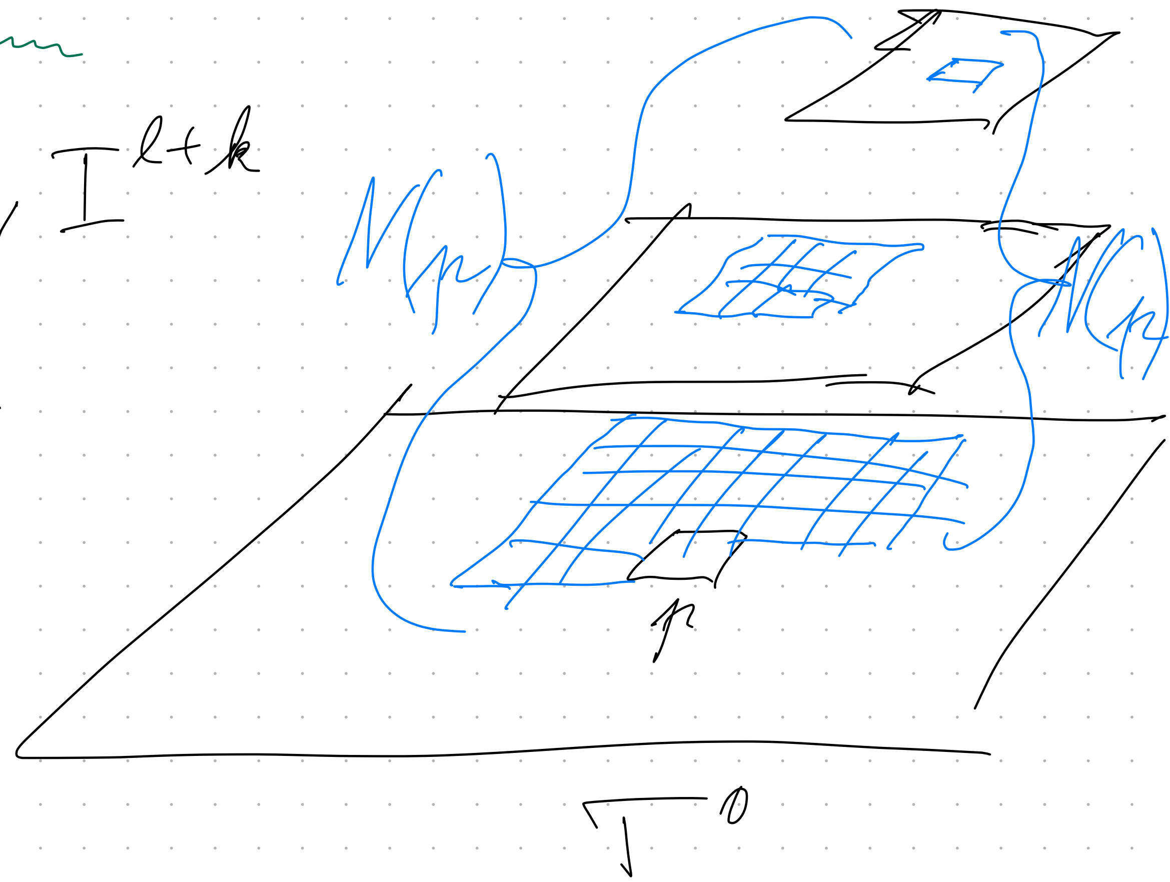
I^{l+1} from I^l by averaging each 2×2 pixels in I^l (or, Gauss filter + subsampling)

$d = \log(\text{resolution})$



Algo.:
build img pyramid $I^0, \dots, I^d$

for $i := d-k, \ldots, 0$:   // k = param
    construct T^ℓ out of $I^\ell, I^{\ell+1}, \ldots, I^{\ell+k}$
    with $N(p)$, $p \in T^\ell$
        stretches over layers $\ell, \ell+1, \ldots, \ell+k$



Appl. Shape Matching

Problem: given database of content, e.g. images, 3D geom, ..

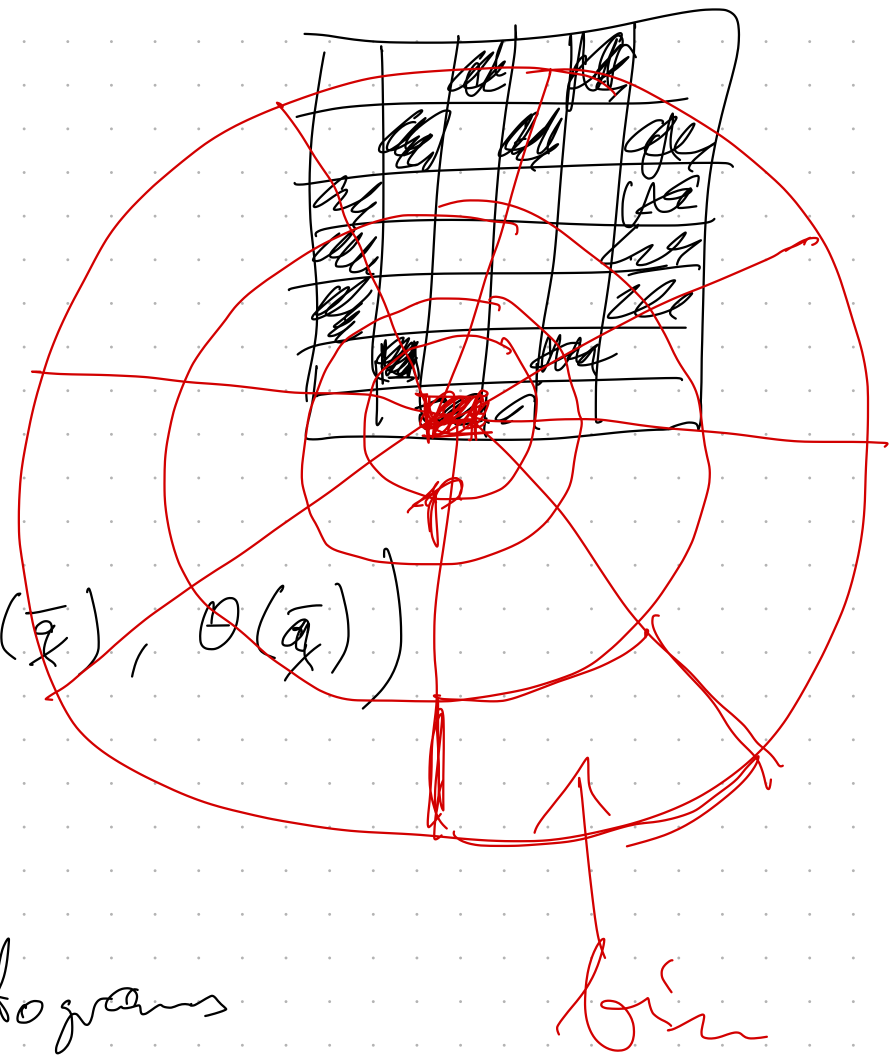
Shape = 2D curve, 3D surface

Approach in general:

1. Define: transformation : shapes → descriptor "feature vector" $\in \mathbb{R}^d$
   ideally: invariant w.r.t. rotation & translation & scaling
2. Define "dissimilarity measure" d:
   f_1, f_2 feature vectors,
   $d(f_1, f_1)$ big ⇒ shapes S_1, S_2 look very different
3. Database → set of feature vector $F \subseteq \mathbb{R}^d$ → kd-tree
4. Given query shape q → feature vector f_q → (A)NN search
   or k-NN search

→ content-based information retrieval

Example: "shape context"



Represent 2D shape as grid

For each black pixel p:

generate 2D histogram

for all other black pixels q:

$\bar{q} = q - p$

represent $\bar{q}$ in polar coords $\left(r(\bar{q}), \Theta(\bar{q}) \right)$

$$\mathrm{bin}(\bar{q}) := \begin{pmatrix} \lfloor (\log r^2) \cdot N \rfloor \\ \lfloor \Theta \cdot N \rfloor \end{pmatrix}$$

Accumulate all histograms $\longrightarrow$ overall histograms

$\Downarrow$

feature vector

Properties: invariant w.r.t. translation

Dissimilarity measures: $f, g \in \mathbb{R}^d$

1. $d(f, g) = \|f - g\|_p$, L_1-norm works rel. well

2. Kullback-Leibler-Divergence (KL):
compare histograms h_1, h_2

$$K(h_1, h_2) = \sum_{i \in \text{bins}} h_2(i) \cdot \ln \frac{h_2(i)}{h_1(i)}$$

Properties: $K \geq 0$

$K = 0 \iff h_1 = h_2$

$K(h_1, h_2) \neq K(h_2, h_1)$

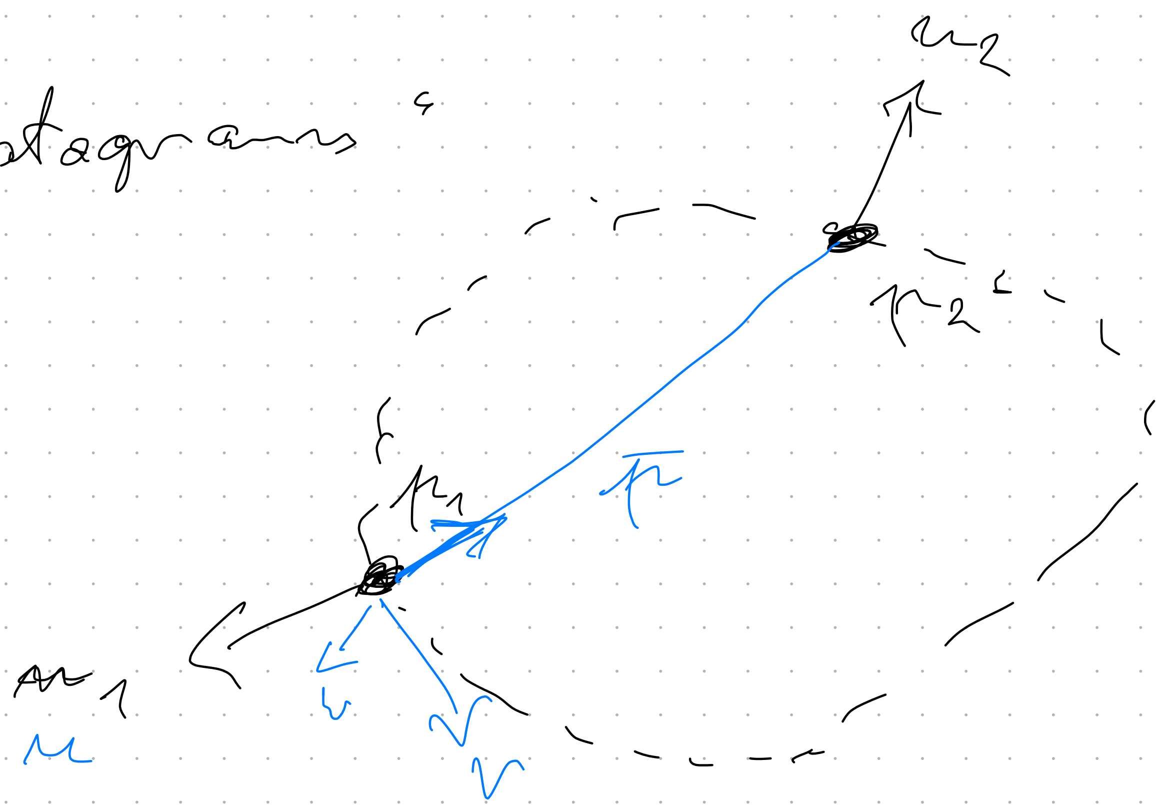
Deal with $h_1(i) = 0$ $\rightarrow$ ignore them, or penalize

($h_2(i) = 0$ OK, because $\lim_{x \to 0} x \ln x = 0$.)

Example: 3D shapes

Input: meshes in 3D

Define features: "surflet pair histograms"



Consider p_1, p_2 vertices
with normals n_1, n_2

Set $\bar{p} := \text{normalize}(p_2 - p_1)$

Construct coord syst. in p_1:

$u := n_1$

$v := \text{normalize}(\bar{p} \times u)$

$w := u \times v$

Describe p_2 rel. to p_1:

$\alpha = \arctan 2(w \cdot n_2, u \cdot n_2)$

$\beta = \measuredangle(v, n_2)$

$\gamma = \arctan 2(\bar{p} \cdot u, \bar{p} \cdot v)$

$\delta = \measuredangle(\bar{p}, w)$

$\varepsilon = \| \bar{p} \|$

Discretize $(\alpha, \dots, \varepsilon)$ in k levels $\to [0, k]^5 \in \mathbb{N}^5$
"feature vector"

for all pairs $(p_i, p_j) \in$ mesh :
bin in $[0, k]^5 \to$ histogram of shape

Properties: invariant w.r.t. translation, rotation

Example : ICP

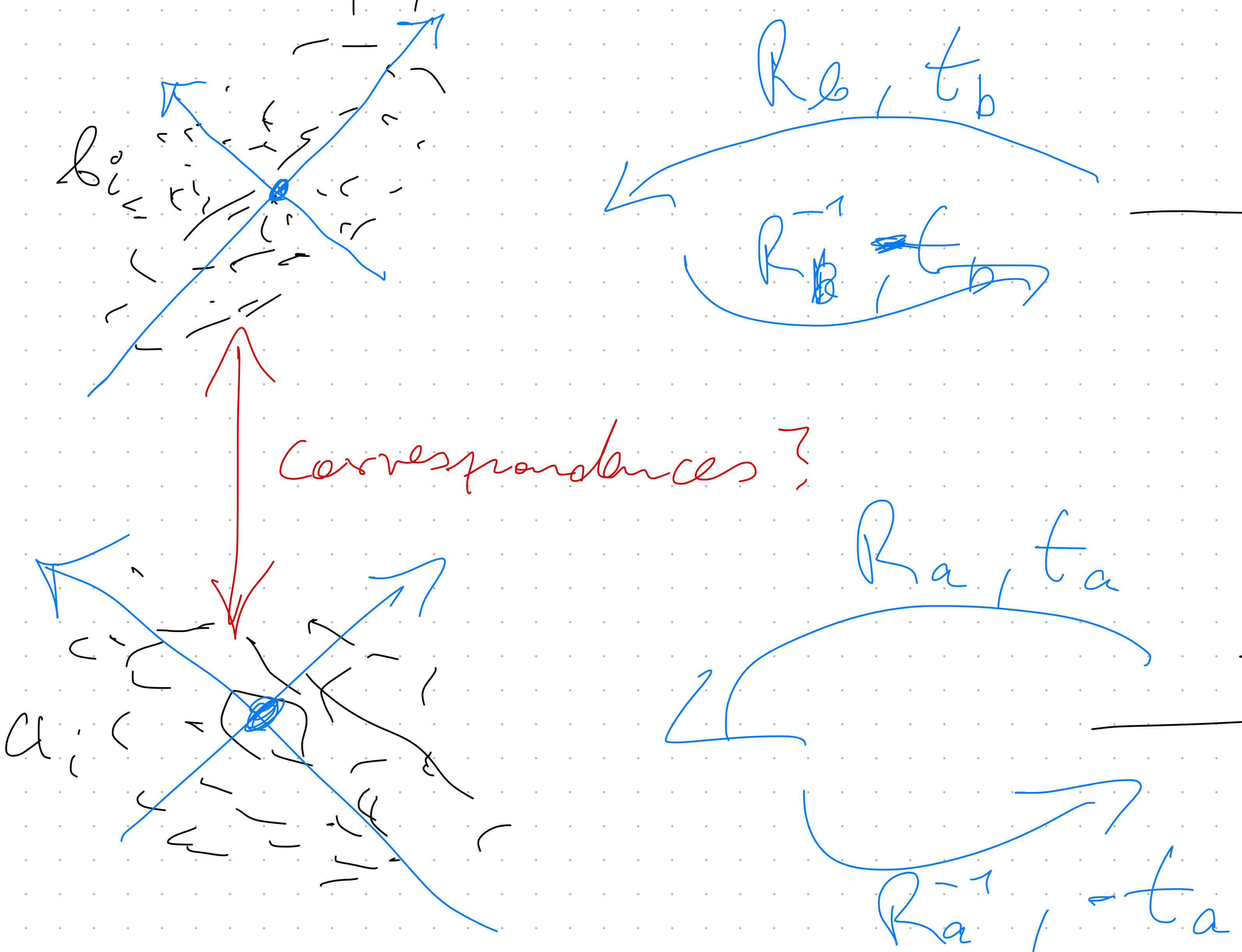
Given: two pt sets $A, B \subseteq \mathbb{R}^3$, $A = \{a_i\}$, $B = \{b_i\}$

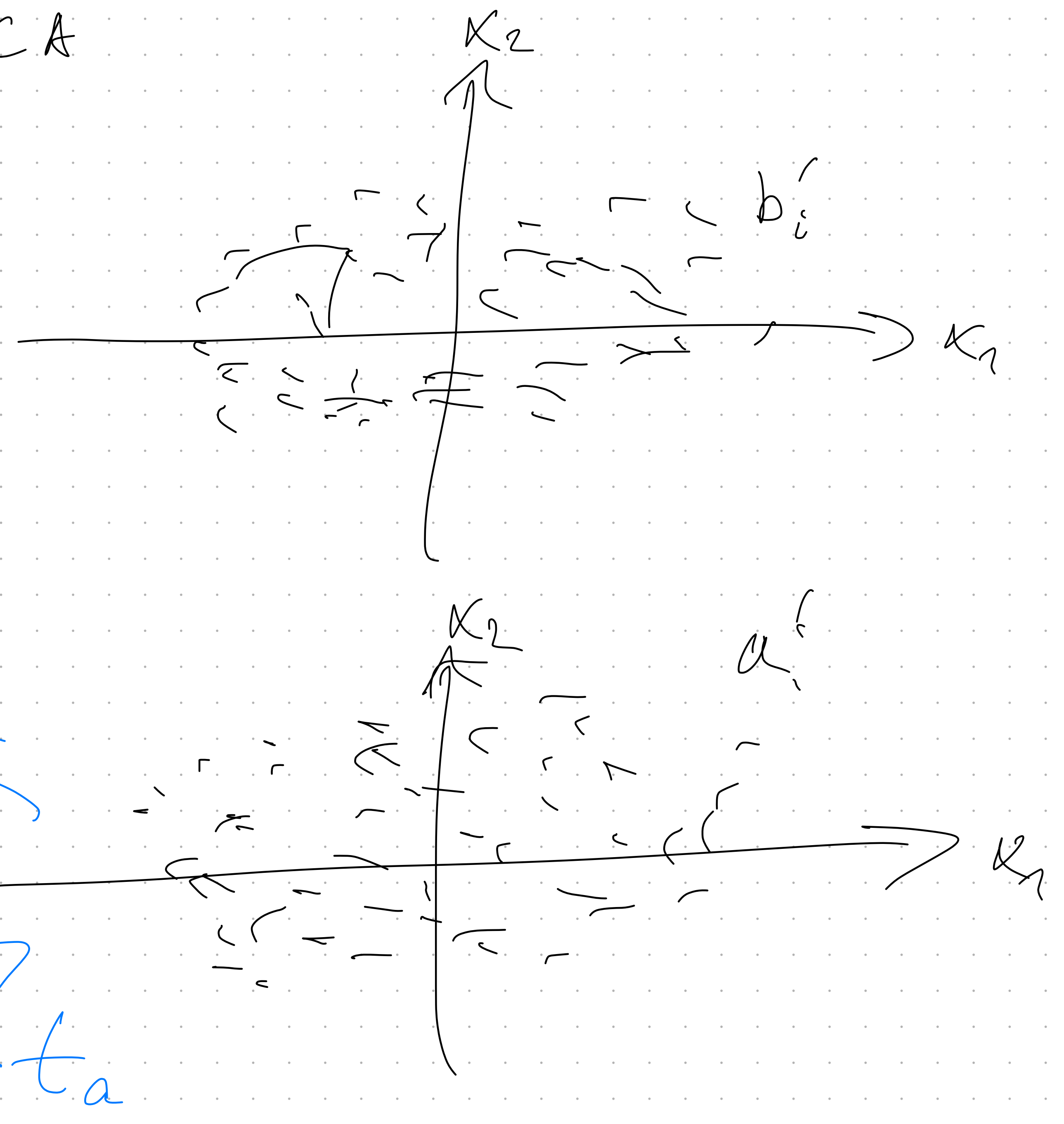
Assume: perfect match, i.e., R, t exist s.t.

$$a_i = R \cdot b_i + t$$

Wanted: R, t

Simple approach: use PCA





$$a_i = R_a R_b^{-1}(b_i - t_b) + t_a$$

To be continued on the slides