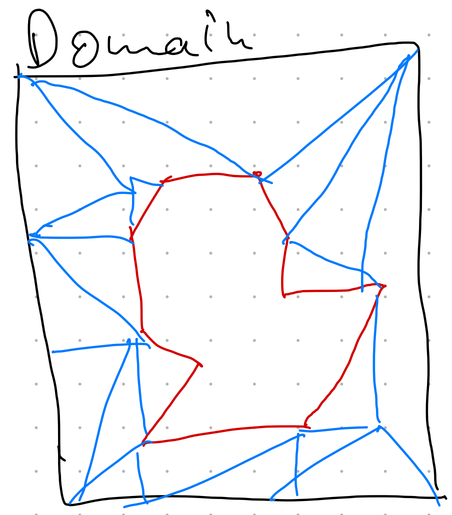
Quadtrees

Quadtrees

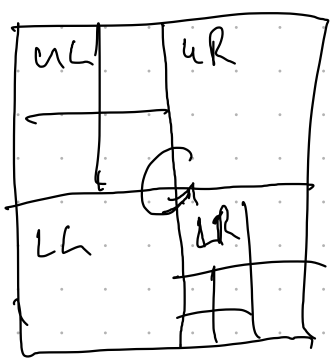


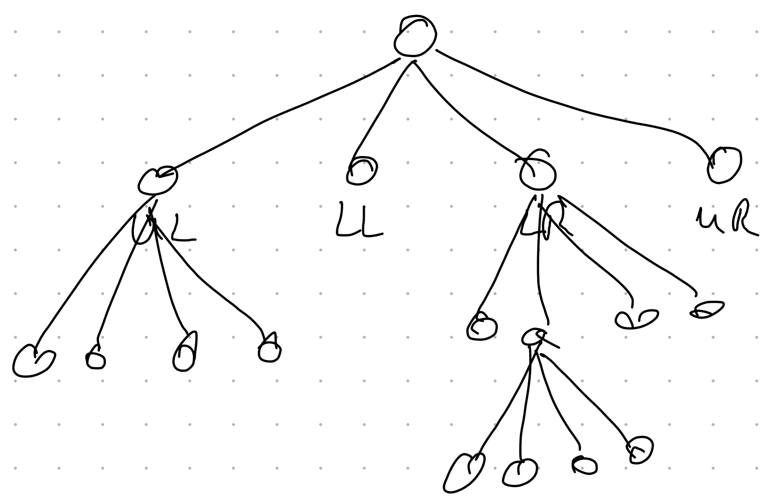
Example application: Meshing

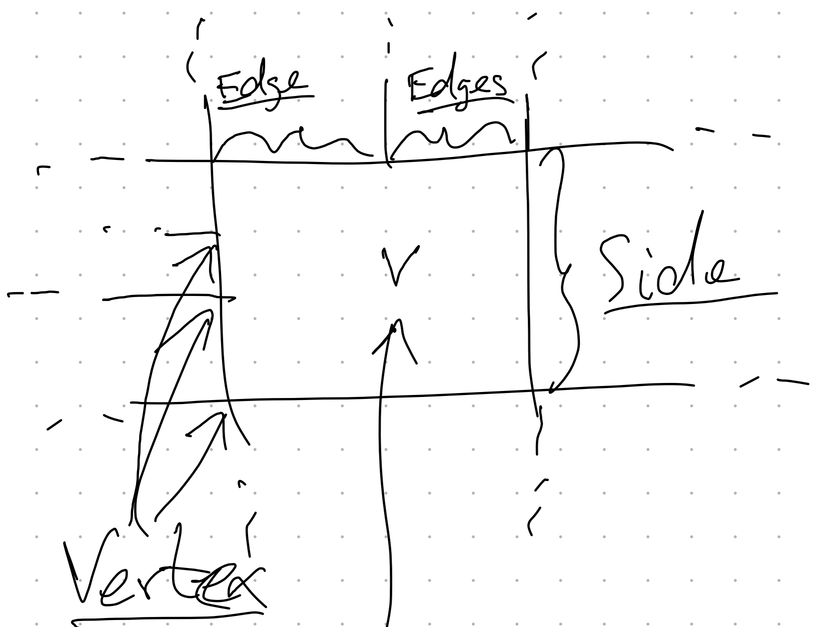
Quadtrees := Tree, where each inner node has 4 children;
each node corresponds to square in the domain;
children of a node ≙ quadrants of the node

→ Leaves of q.tree ≙ subdivision of the domain
Dito for set of nodes of a level in the q.tree

Ex. q.tree:







Squares neighbors ⟺ share common edge side

Square, Cell, Node

Def.: $q(v) := \underbrace{[x_v, x_v']} \times \underbrace{[y_v, y_v']}$

Algo: Construction of Qtree over Pts

Given $P =$ set of pts $\subseteq \mathbb{R}^2$

$Q(P) :=$ node v, where

v is leaf, if $|P| \leq 1$

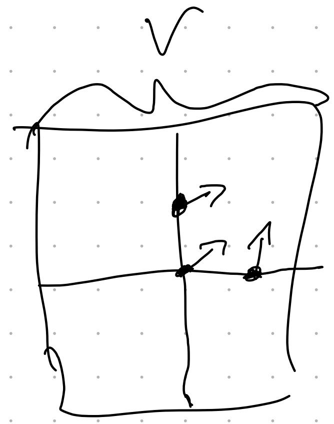
v is a quadtree with children $v_{UL}, v_{LL}, v_{LR}, v_{UR}$

if $|P| > 1$

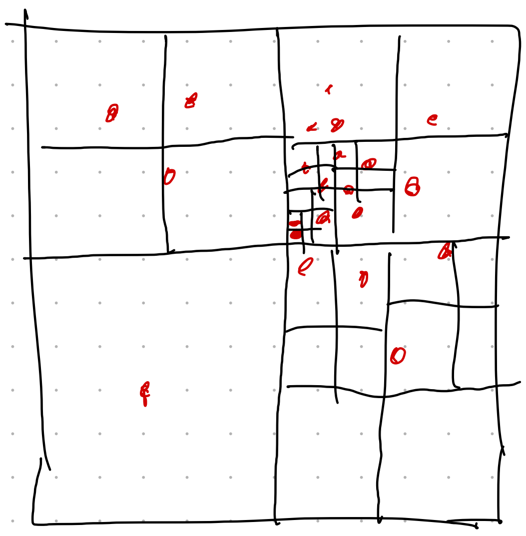
and $P(v_{UL}) := \{ p \in P \mid p_x \geq \frac{1}{2}(x_v + x_v') \text{ and } p_y \geq \frac{1}{2}(y_v + y_v') \}$

$P(v_{LL}) :=$ analogous

...



Ex.:



Depth depends on "distribution" of the geom objs

Lemma:

Let P be set of pts in $\mathbb{R}^2$,

let $s =$ side length of root,

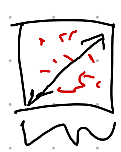
$c = \min \{ \|p_1 - p_2\| : p_1, p_2 \in P, p_1 \neq p_2 \}$.

Then, depth $d \leq \log\left(\frac{s}{c}\right) + \frac{3}{2}$

Proof: w.l.o.g. $s = 1$

Observe side length of node v at level $i = \frac{1}{2^i}$

max dist inside $v = \frac{\sqrt{2}}{2^i}$



$c \leq \min \{ \|p_1 - p_2\| : p_1, p_2 \in P(v) \} \leq \frac{\sqrt{2}}{2^i}$

$\Rightarrow i \leq \log \frac{\sqrt{2}}{c} = \underbrace{\log \frac{1}{c} + \frac{1}{2}}$

$\uparrow$ true for inner nodes

$\Downarrow$

for leaves $i \leq \log \frac{1}{c} + \frac{1}{2} + 1$

__Lemma:__ Complexity of Qtrees

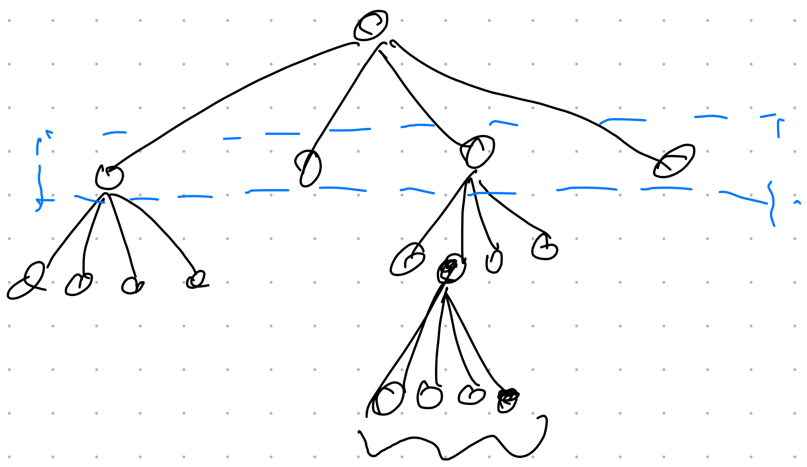
A qtree of depth d and over n pts need $O(n(d+1))$ nodes and takes $O(n(d+1))$ time to construct.

Proof:

Observe: $\#$ leaves $= (\#\text{inner nodes}) \cdot 3 + 1$ (by induction)

$\Rightarrow$ need prove bounds for inner nodes

Part 1:



$\sum_{\substack{v \text{ of} \\ \text{one layer}}} \text{pts} \leq n$

$\#$ nodes on a layer $\leq n$

$\Rightarrow$ $\#\text{nodes} \leq n \cdot (d+1) + 2n = n(d+1)$

Quadrupel of leaves, at least 2 nodes must have a pt

Part 2:

Let $m =$ #pts of node v $\Rightarrow$ $T(v) \in O(m)$

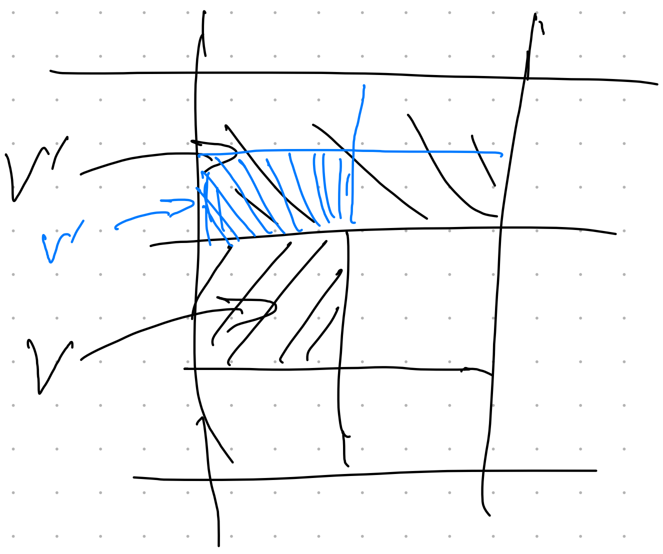
pts on one level in qtree $\leq n$

$\Rightarrow \sum_{\substack{v = \text{nodes} \\ \text{on level } i}} T(v) \in O(n)$ $\Rightarrow$ claim

Operation: Neighbor finding

Given: node v

Wanted: $v' =$ north neighbor such that $\text{depth}(v') \leq \text{depth}(v)$

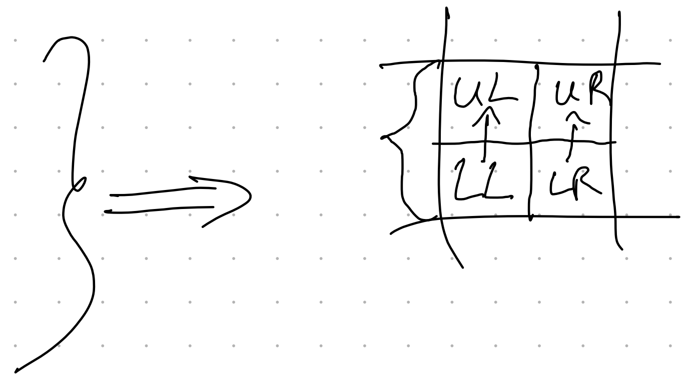


Algo: NorthNeighbor (v)

if v is root → return nil

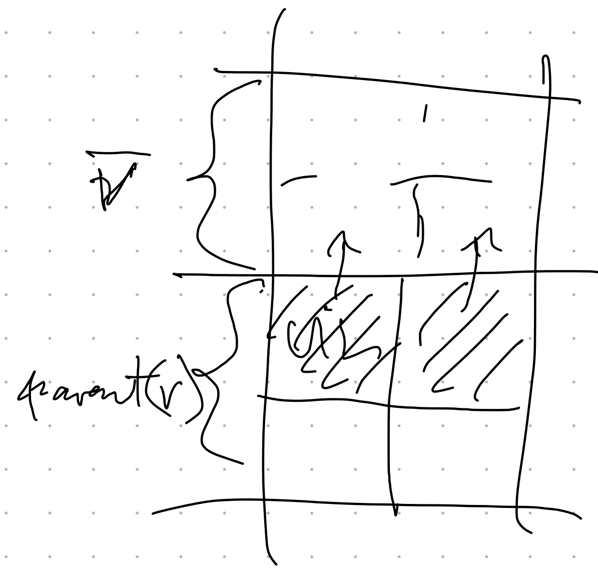
if v is LL-child of its parent
    return UL-child of parent(v)

if v is LR-child of ---
    return UR-child of parent(v)



$\bar{v}$:= NorthNeighbor(parent(v))

if $\bar{v}$ is nil or
    $\bar{v}$ is leaf
  return $\bar{v}$



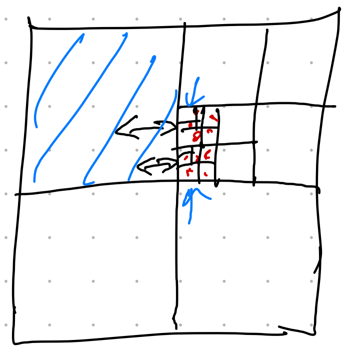
if v is UL-child
    return LL-child of $\bar{v}$

if v is UR-child
    return ---

Running time: $O(d)$

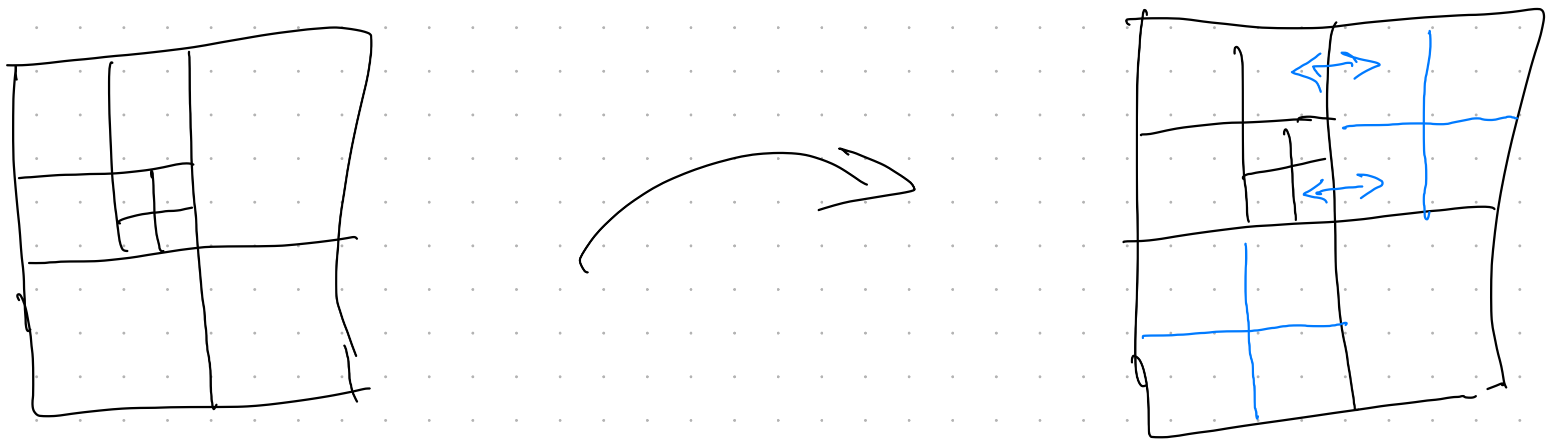
Balancing Quadtrees

Ex.: could happen



Def: "balanced quadtree"

$:\Longleftrightarrow \forall$ neighbors v, v' in Q: $|depth(v) - depth(v')| \leq 1$



Algo for balancing (Sketch):

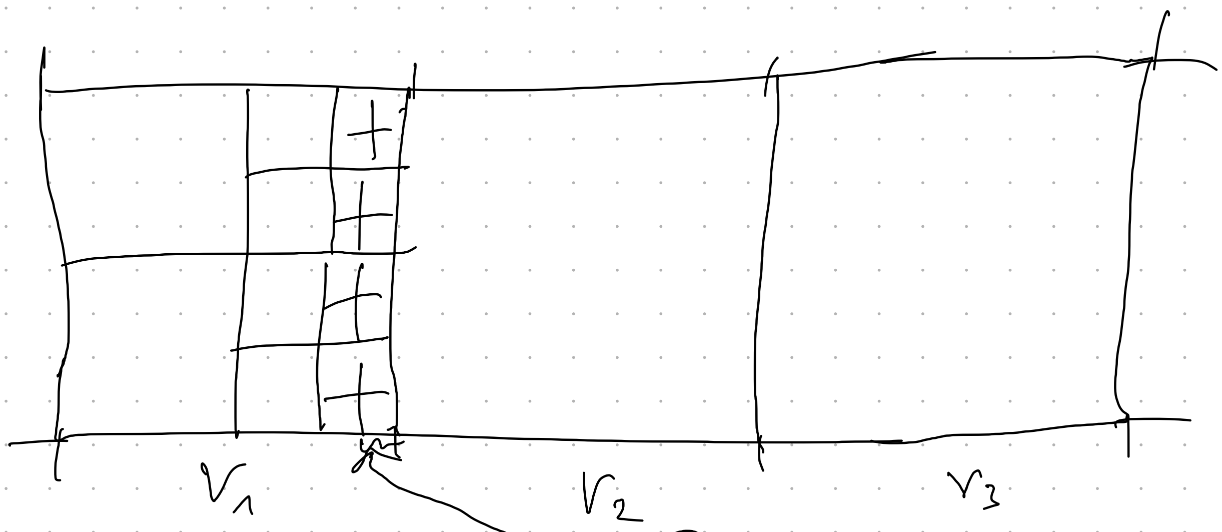
maintain list L of all leaves
init L with leaves of orig. quadtree
Iterate until balanced with following 2 steps:

1. check if leaf v needs to be split
   (use neighbor finding
   Q: do we need to go down till the way in the neighbors?)
2. When leaf is split, then check wether their neighbors need splitting $\rightarrow$ use neighbor finding

Lemma:
Let Q be a quadtree with m nodes,
$\hat{Q}$ be the balanced version of Q.
Then $\hat{Q}$ has $O(m)$ nodes and can be constructed in $O(m(d+1))$ time.

Proof:

Part 1 (size): we prove $O(n)$ splitting operations $\Rightarrow$ claim



Claim: no matter how small these nodes, v_3 never gets split.

Notation $D(v) :=$ height of the subtree underneath v.

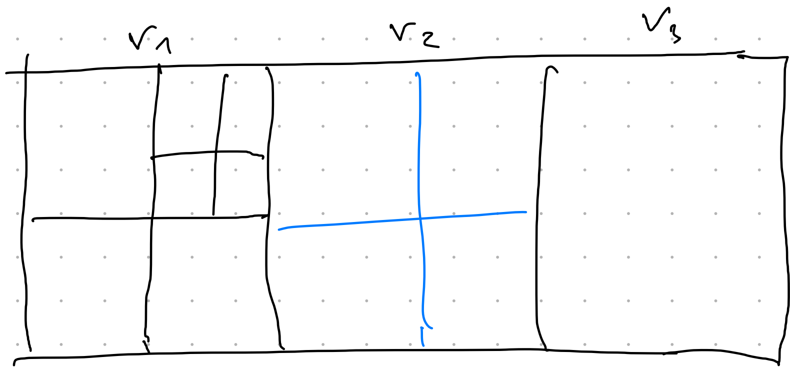
Base case:

$D(v_2) = D(v_3) = 0$

$D(v_1) = 2$

$\Rightarrow$ v_2 is split 1x

v_3 is ~~not~~ not split $\Rightarrow$ claim



Inductive step:

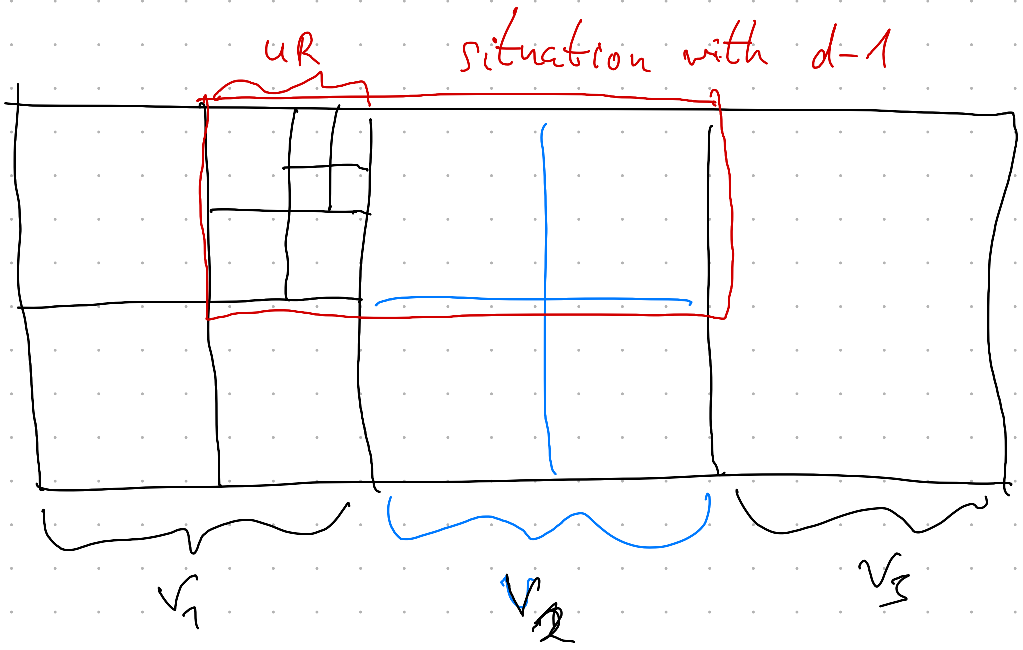
$D(v_1) = d > 2$

$\Rightarrow$ v_2 is split (at least) once

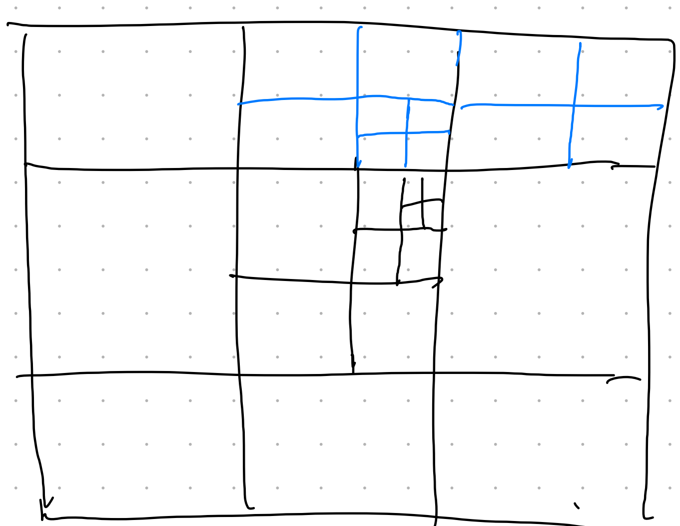
$D(\text{UR-child of } v_1) = d-1$

$\Rightarrow$ UR child of v_2 is never split (b/c induction)

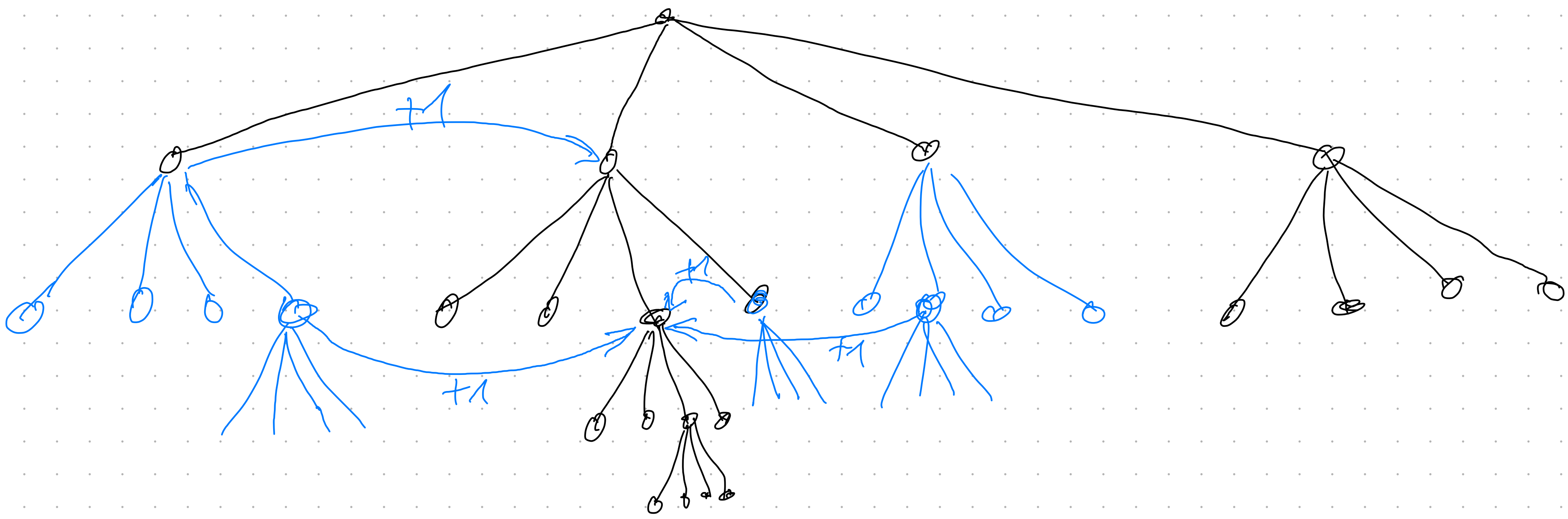
$\Rightarrow$ v_3 is never split $\Rightarrow$ claim



Note: splits can propagate around corner



"old nodes" = in orig qtree
"new nodes" = in new/balanced tree
Introduce split counter for each node:
increment it, if its old node caused a split



⇒ for each node, its split counter ≤ 8 in the end

⇒ each old node has "caused" at most $8 \cdot 4$ new nodes ⇒ part 1 of lemma

Part 2 (time):

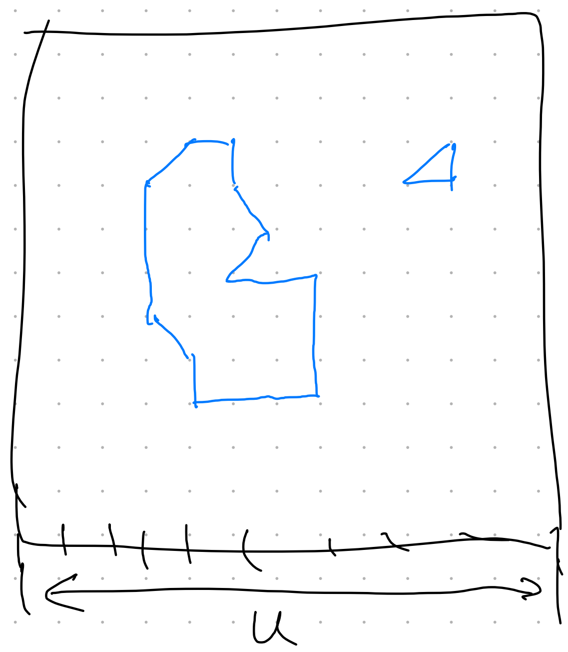
Time per node is $O(d+1)$, b/c we need only constant # neighbor finding op's for that node; each node gets "visited" only once

Meshing

Domain: square $[0,U]^2$

Input: set of poly lines

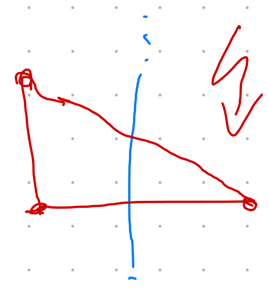
Simplification: only integer coords, only angles $\in \{0°, 45°, 90°, ..\}$



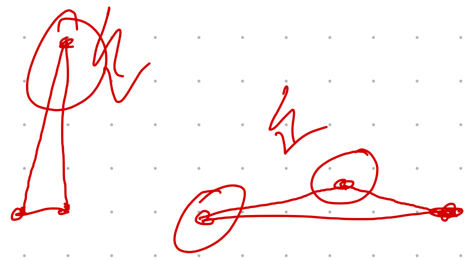
Goal: triangulation with Properties:

1) "conforming": no T vertices

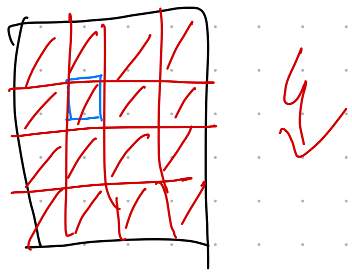
2) "constrained": poly lines are part of the triangulation



3) "well-shaped mesh":
all angles in $\{45°, 90°\}$



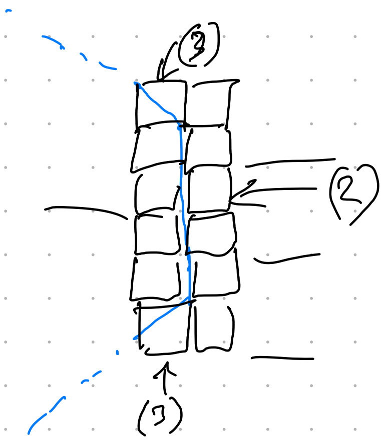
4) "non-uniform":
adaptive mesh wanted



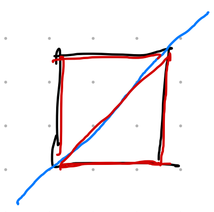
Approach (algo sketch):

Make qtree over poly lines
Similar qtree over sets of pts
Except different stopping criterion:
stop if no edge of poly lines intersects or touches the cell, or if size of cell = 1x1

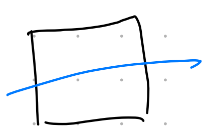
Example:

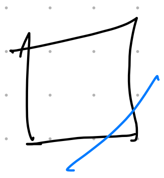


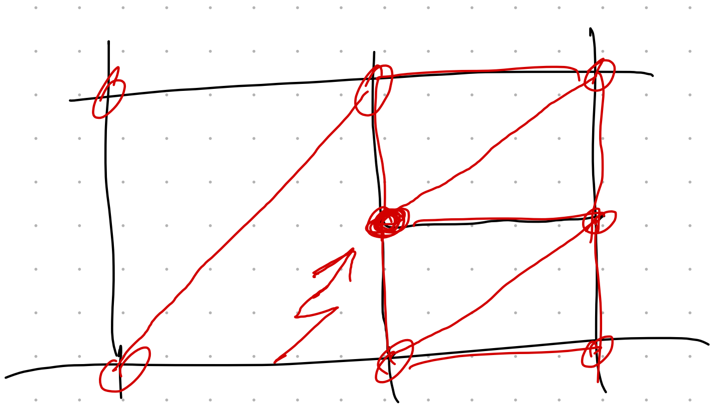
Consequence: each leaf of the qtree is
1) not intersected/touched by segment of poly line; or
2) touched along its side
3) intersected by poly line

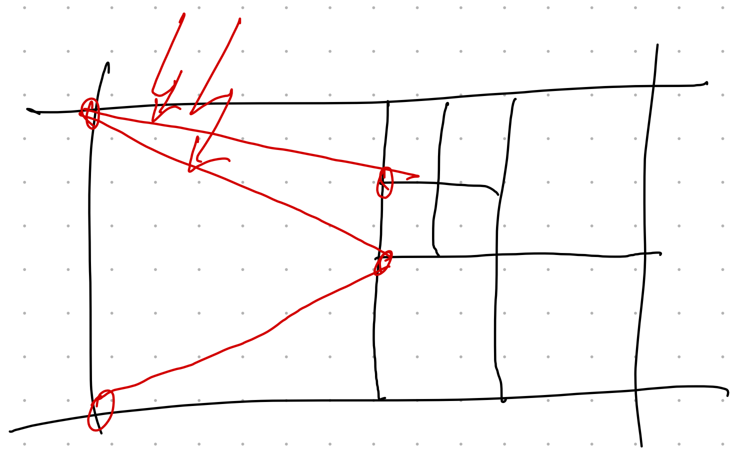


Can't happen:









Solution: balanced qtree

Algo:

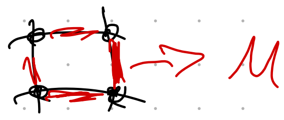
Input: polylines S, with above properties

Output: triangle mesh M, with ..

create qtree T over S

balance $T \Rightarrow Q$

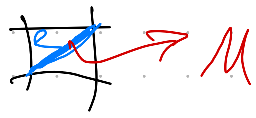
init M with all edges induced by Q



foreach leaf $q \in Q$:

 if q is intersected by edge $e \in S$:

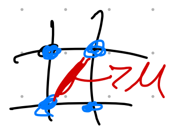
  add e to M



 else:

  if q has only vertices $\in S$ in its corners (if any):

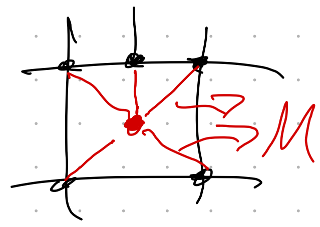
   $\Rightarrow$ add diagonal to M



  else q has vertices on sides:

   $\Rightarrow$ add center pt,

   add edges to corners of q



Lemma:

Let S be polyline with above properties (no self-intersections) inside domain $[0,U]^2$.

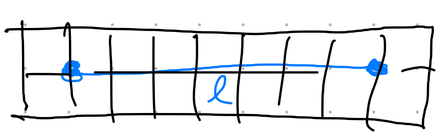
Then there is a triangle mesh M (w/ properties) that has $O(\log(U) \cdot \mu(S))$ triangles, and can be constructed in time $O(\mu(S) \log^2 U)$.

$\mu(S)$ = sum of all lengths of all polylines.

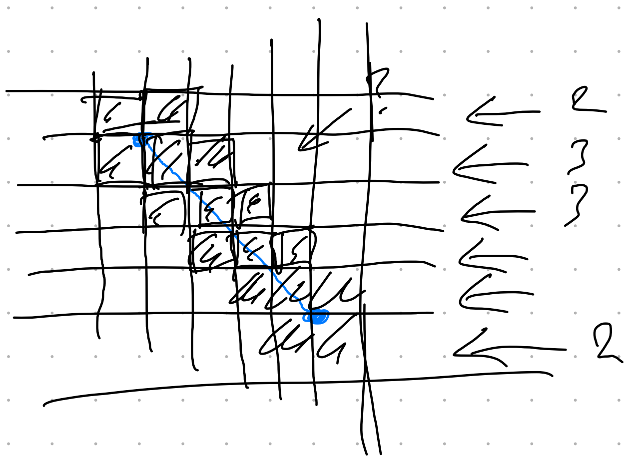
Proof:

cells that get touched/intersected by S have size 1×1.

Segment of length l can intersect/touch at most $4 + 3\frac{l}{\sqrt{2}}$ 1-cells



$2(l+2)$ cells



$\Sigma = \frac{l}{\sqrt{2}} \cdot 3 + 4$

$\Rightarrow$ # leaves touched/intersected $= O(\mu(S))$

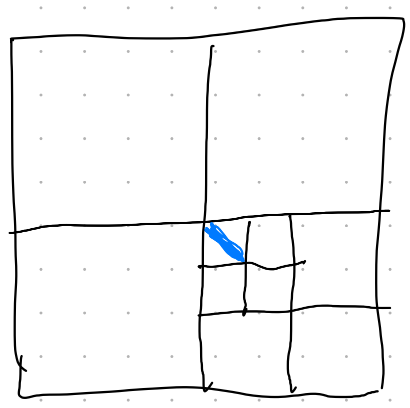
$\Rightarrow$ # leaves at bottom layer of $T \in O(4 \cdot p(S))$

$\Rightarrow$ # leaves in $T \in O(p(S) \cdot \log U)$

$\Rightarrow$ # triangles, b/c each leaf can produce at most 8 triangles.

Part 2: constr. time

Tight bound:



$p(S) = \text{const}$

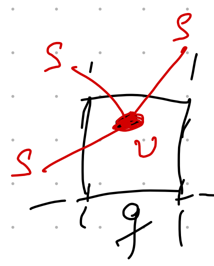
nodes $= 4 \log U + 1$

Meshing for arbitrary polylines S:

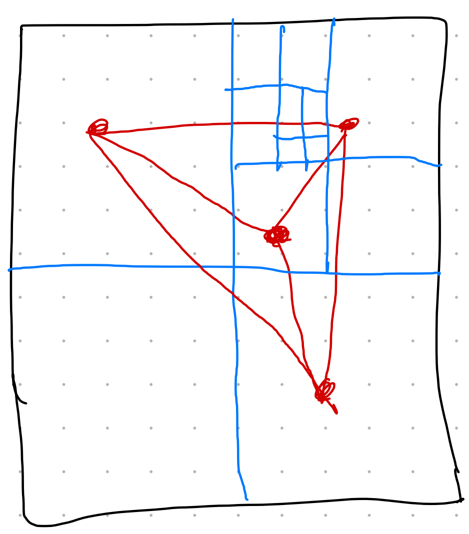
$\Rightarrow$ Different leaf types: edge nodes, vertex nodes

Stopping criterion for leaf q:

- max depth
- empty
- exactly one (part of) edge of S inside q (no vertex $\in S$)
  ↳ edge leaf
- exactly one vertex $v \in S$, with all edges intersecting q must be incident to v



Example:



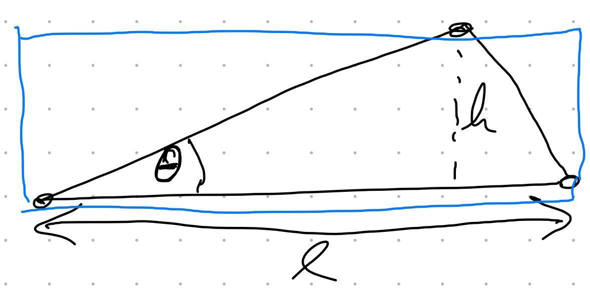
$\rightarrow$ try to triangulate with "nice" triangles

Def.: aspect ratio

$l :=$ length of longest side of triangle

$h :=$ height

$\Rightarrow$ aspect ratio $\alpha := \frac{l}{h}$



Note: smaller is better

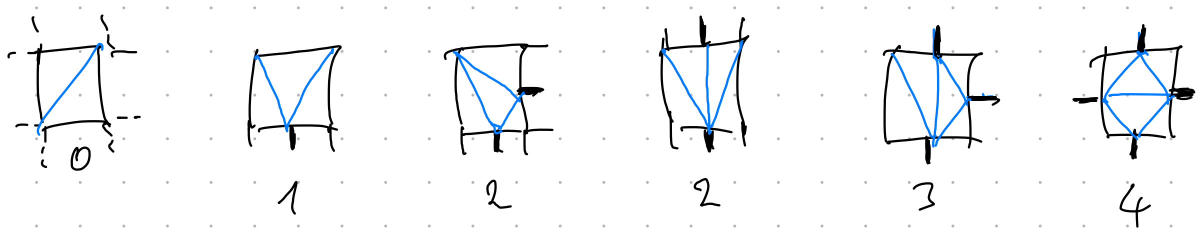
$\alpha \geq \min = \frac{2}{\sqrt{3}} \approx 1.15$

Let θ = smallest angle, then $\frac{1}{\sin\theta} \leq \alpha \leq \frac{2}{\sin\theta}$

Goal with meshing arbitrary S: create tris with aspect ratio close to optimum

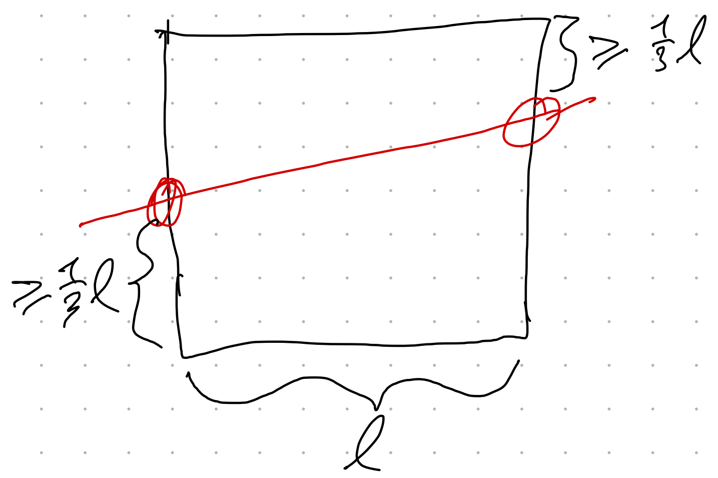
→ Modify mesh creation:

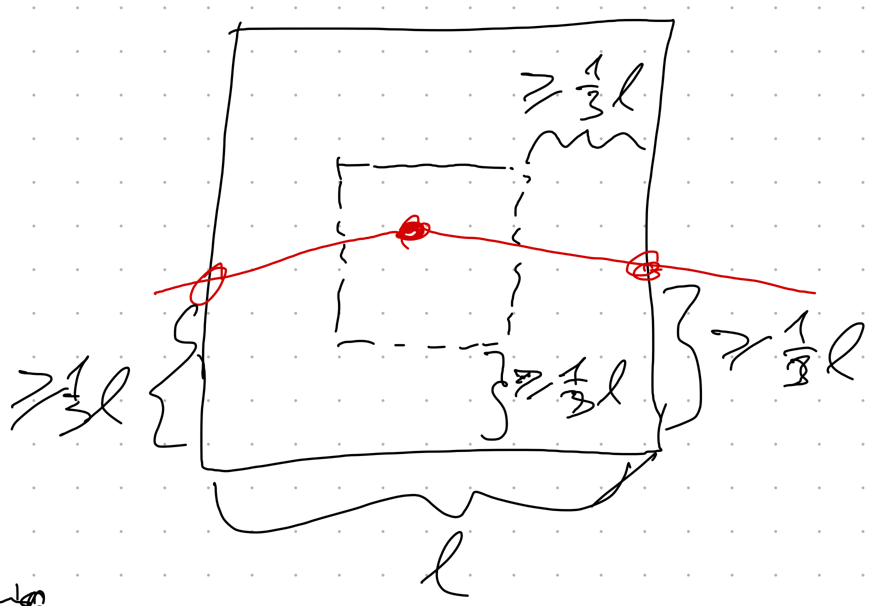
1. Create balanced quadtree over S
2. Triangulate: 3 cases
   a) empty leaf ⇒ triangulate it according to the templates



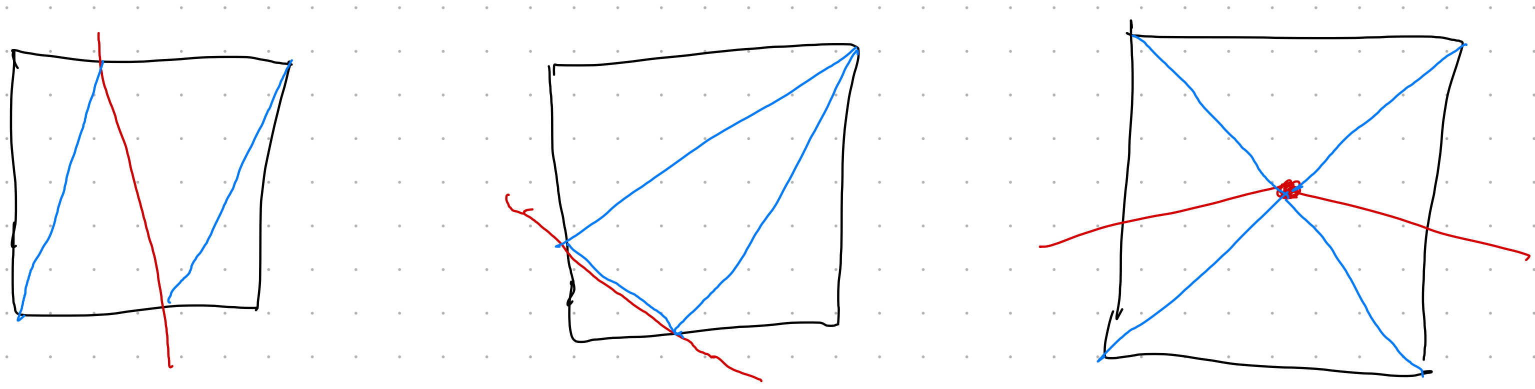
← # inner vertices on node's side

b) leaf containing edge $\in S$, or vertex $\in S$, intersections with sides of leaf are $\geq \frac{1}{3} l$ from corners of the leaf

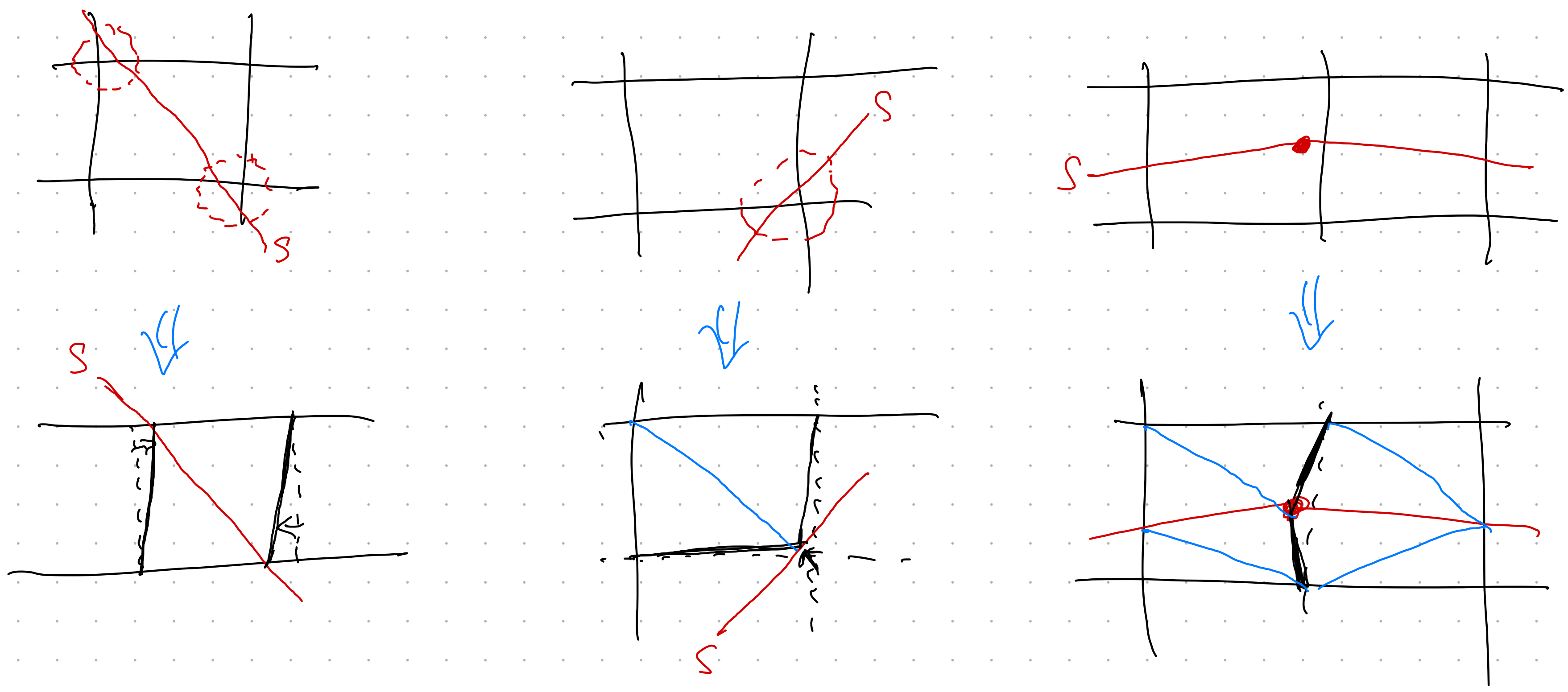




→ triangulate according to following schemas:



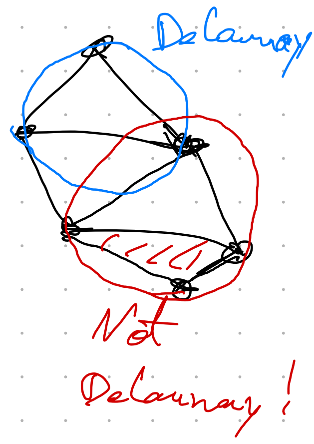
c) Else: deform the leaf ("warping"):



3. Improve mesh:

Def.: <u>Delaunay condition</u>

A triangle is Delaunay triangle, iff its circum circle does not contain any other vertex from the mesh.
A mesh containing only Delaunay triangles is called a Delaunay mesh.



Operation: "edge flip"

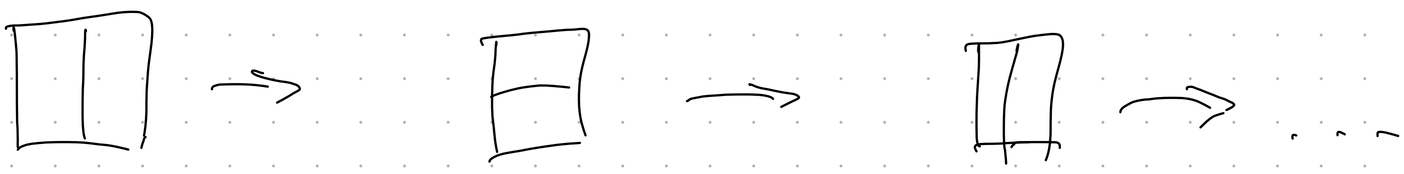
→ Try to establish Delaunay triangulation by continued edge flips.

! works completely only in 2D !

Variants / Generalizations

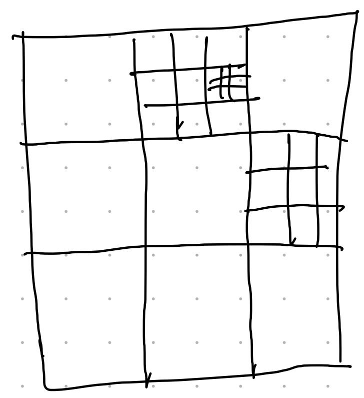
0. Quadtree in higher dimensions
   In 3D: "octree", in d-dim. "d-dim. octree"

1. Bintree:
   split in round-robin fashion in 2 children: along xy-plane, then xz, then yz, then xy, ...



2. N^2-tree: subdivide into N^2 children
   ($N=2 \triangleq$ quadtree)

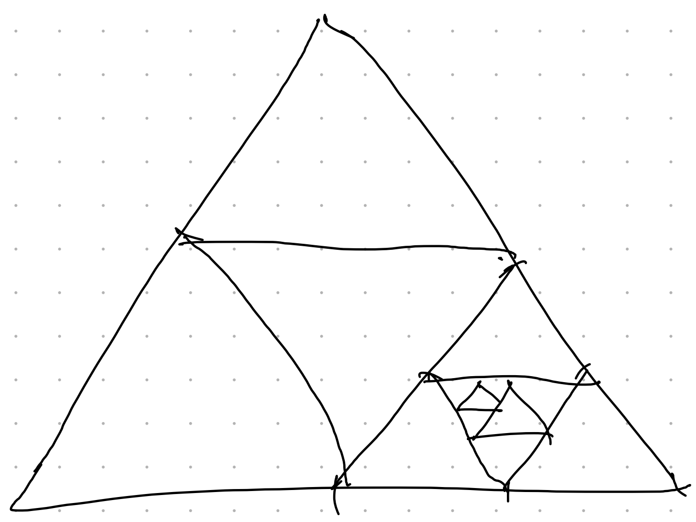
Example: $N=3$

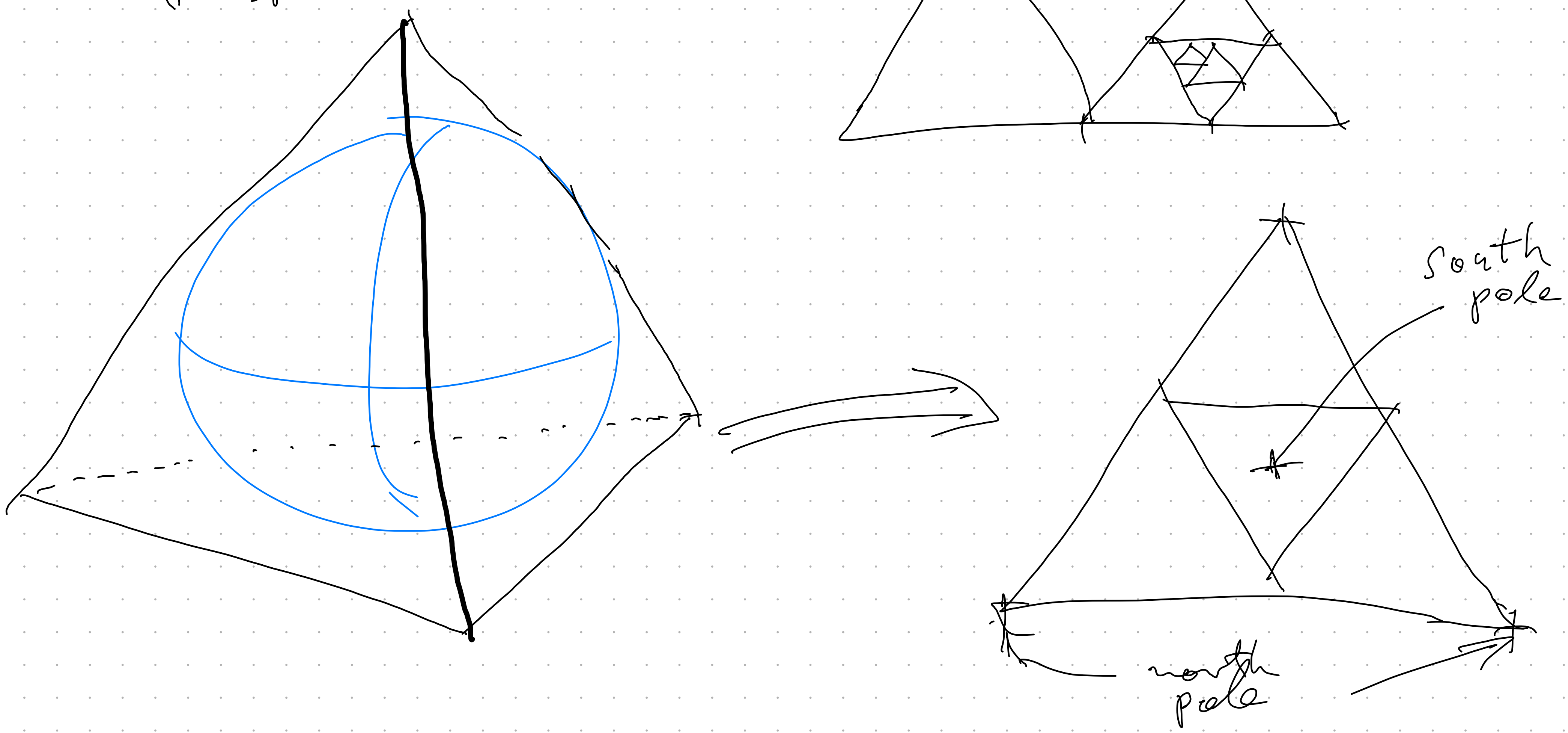


Establishes kind of a "continuum" between quadtree and full grid.

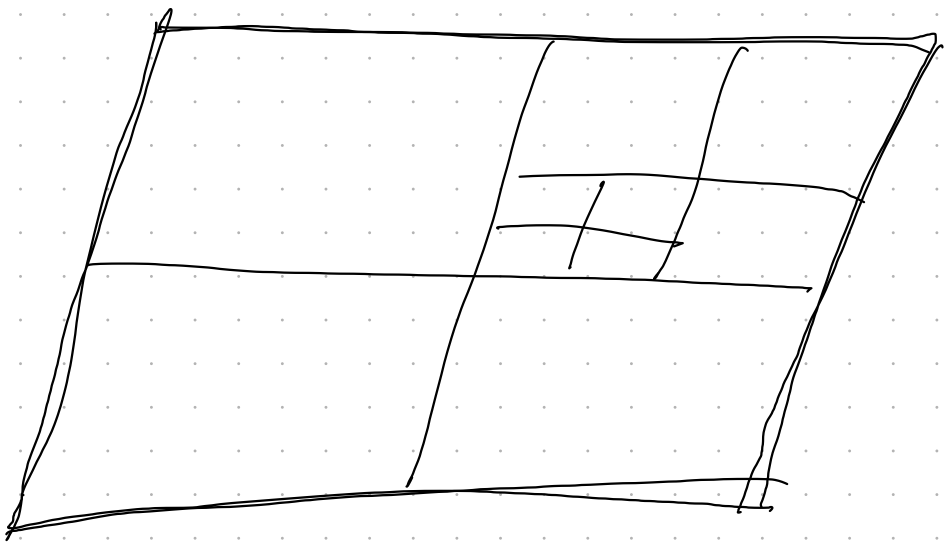
3. Triangle quadtree: Start with triangular domain
   subdivide into triangles

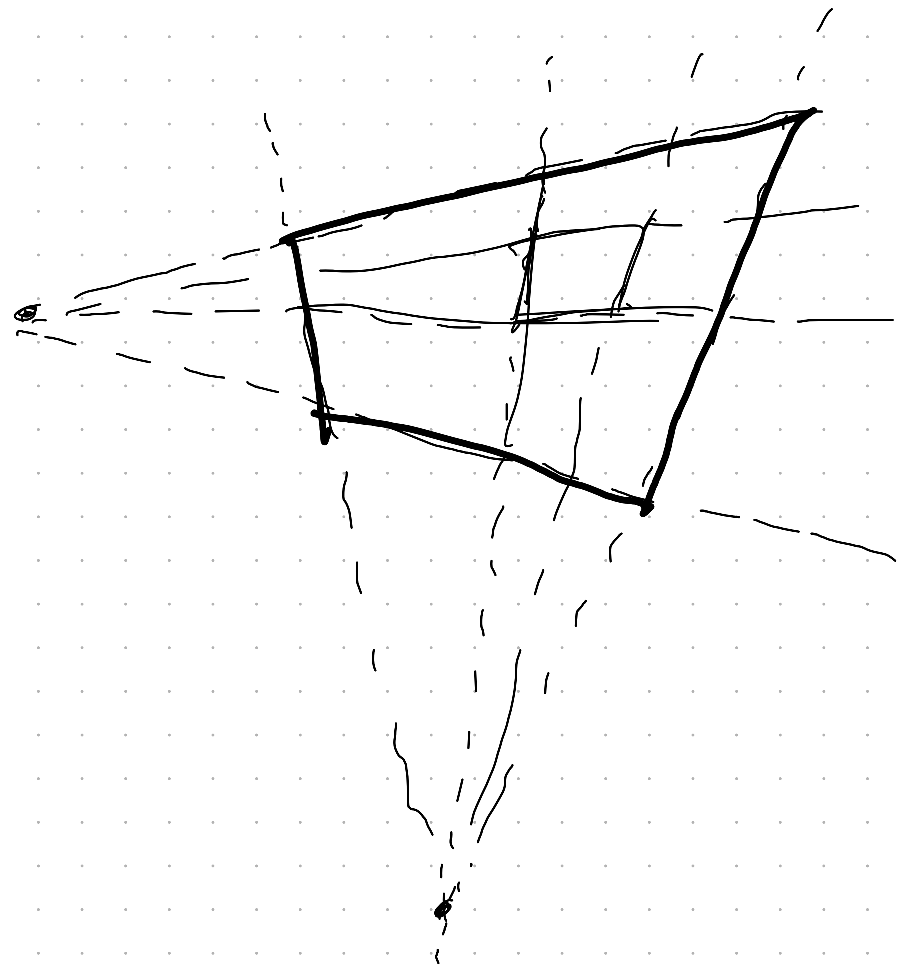
Well-suited to generate hierarchical partitioning of sphere





4. Oblique quadtrees / Vantage quadtrees:





Algo: Point Location in Quadtree

Given: point $(x,y) \in [0,1)$

Sought: leaf containing (x,y)

convert $(x,y) \to (X,Y) = \lfloor x \cdot 2^d \rfloor, \lfloor y \cdot 2^d \rfloor \to M =$ morton code over (X,Y)

cell := root

bitnum = 2d-2 // bits we are interested in

bitmask = 0b11 << bitnum // start w/ 1100...00

while cell has children: shift op

  childidx = (M & bitmask) >> bitnum
  (& = bitwise and, >> = shift)

  cell = cell.children[childidx]

  bitnum = bitnum - 2

  bitmask = bitmask >> 2

return cell

MSBs

yxyxyxyxyx

11 00 00

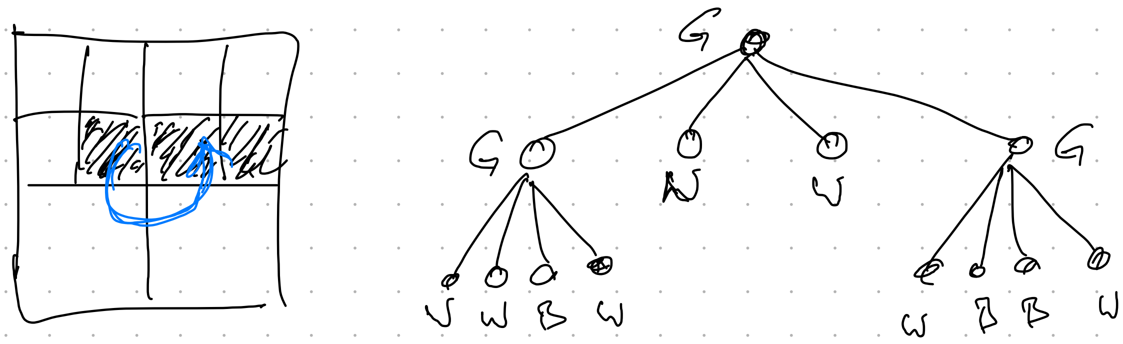
00 ...... 00 yx ∈ 0,..,3

Provided: children are stored in z-order!

Implementing Octrees

1) Pointers or indices into a 1D array
   (Optimization: use 1 pointer per parent, store all 8 children in 1 block)

2) Treecode:
   Represent qtree as sequence of nodes in DFS traversal
   Especially useful for BW images
   Ex.:



$\Rightarrow$ Treecode: GGWWBWWWGWBBW

3) Linear Quadtree:

"location code"

Represent nodes v by just (x,y) of lower left corner
Let d = depth of qtree

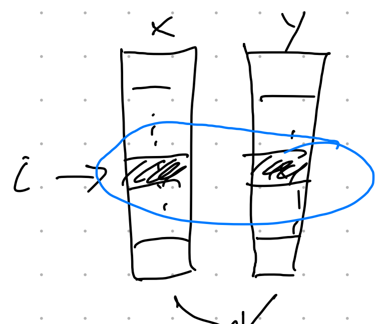
$x, y \in [0, U-1] \subseteq \mathbb{N}^2$, where U given by smallest leaf, $U = 2^d$

$l \in [0, .., d]$ level of v

$\Rightarrow$ Need $2d + \log d$ bits per node

↑ for x,y ← for l

BTW: path from root $\rightarrow v$ is described by (x,y)



→ direction of path from node at level i to child:
$00 \rightarrow LL, \ldots, 11 \rightarrow UR$

Morton code

Store each and every node by (x,y,l)

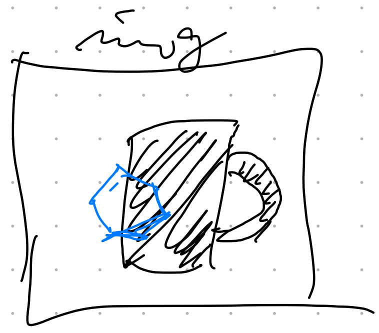
3D Obj Representation by Space Carving

Obj : voxel grid , "black" = inside , "white"
Given: set imgs BW with projection trf
Sought : voxel grid for obj

Approach : use octree , black node = inside
white = outside
gray = don't know (border)

Start : one black root node
fore each img :
for each leaves :
project leaf into img space
modify leaf as follows :

leaf is... \ old color	B	G	W
inside	B	G	W
ambiguous	G	G	W
outside	W	W	W



After one round of imgs ,
subdivide gray leaf nodes , init. with black

Boolean Operations on Images

Demo interactively

Image compression

Goal: simple & efficient algo

Given: grayscale image w/ values $\in [0,1]$

Compression:

1. Build a complete quadtree Q bottom up; propagate min, max, sum bottom-up
2. Prune $Q \to Q$:

   prune subtree $\Leftrightarrow$ max(block of pixels) − min(block) $\leq \theta_1$ ← user-defined param.

3. Calc grayscale at leaves

   $a = \frac{1}{n} \cdot \text{sum}$ (= avg), or $a = \frac{1}{2}(\text{min} + \text{max})$

4. Encode grayscale val's:

   Traverse Q in DFS in z-order

   At each leaf:

   let s = side length, p = predictor fct

   $$\text{code}(a, s; \theta_2, p) := \text{round}\begin{cases} (a-p)\cdot\frac{s}{\theta}\cdot 255 & \text{, if } s < \theta_2 \\ (a-p)\cdot 255 & \text{, if } s \geq \theta_2 \end{cases}$$

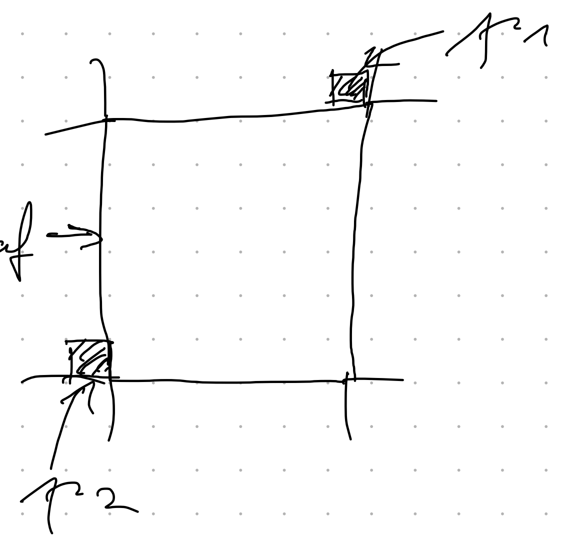
   If $s < \theta_2$: $\text{code} = \left[\frac{a-p}{\theta}\right] \to [-255, 255]$

   Observe: more codes around 0, provided p is "good"

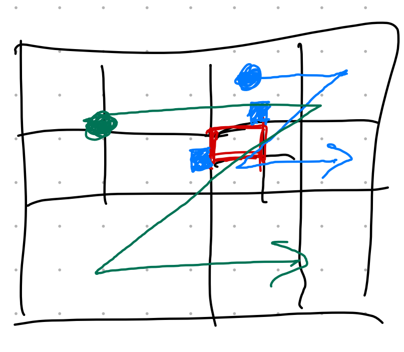
Predictor:

$$p = \frac{1}{2}(p_1 + p_2)$$

p_1: NE pixel w.r.t. current leaf, p_2: SW

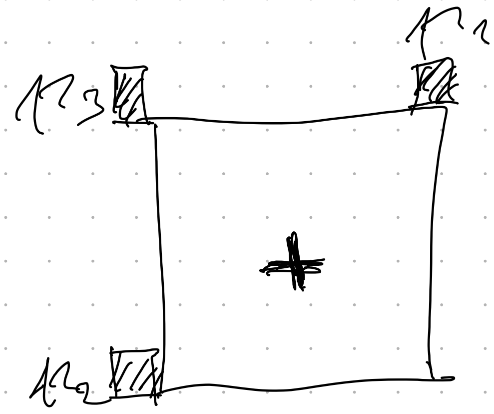


Works b/c of z-order!



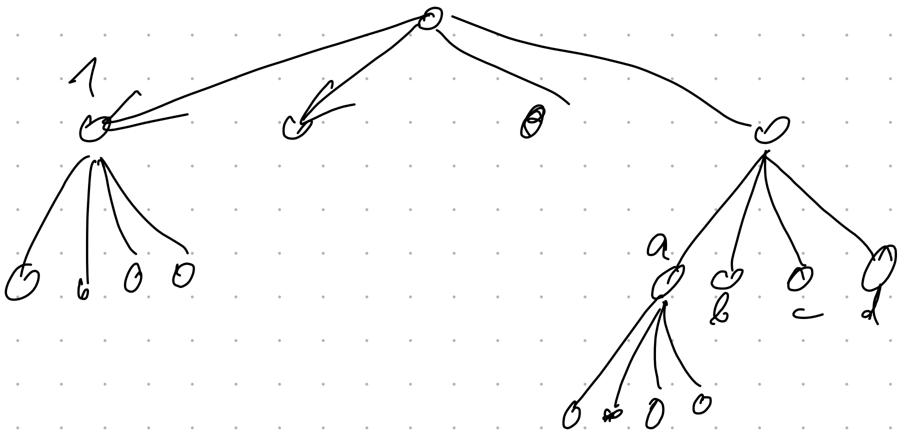
Important: use encoded values for p!

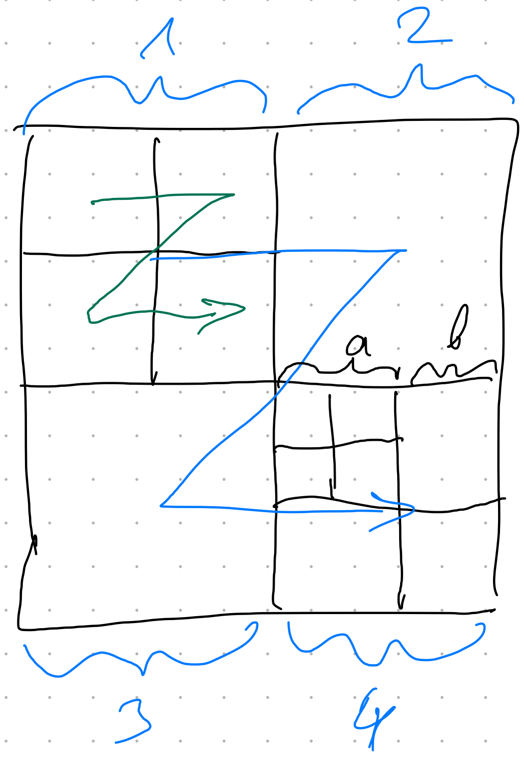
Use better predictors?



5. Tree encoding:

a) encoding of topology of Q: use tree code





output "1" for inner nodes,
"0" for leaves,
in DFS z-order

Example:

1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0

root, node 1, 2, 3, 4, a, b c d

b) Output encoded grayscale val's (separate string):

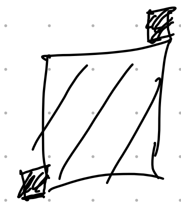
use unary code: 0 → 0

−1 → 100   +1 → 101
−2 → 1100   +2 → 1101
−3 → 11100   +3 → 11101

"end", sign

Remark: rationale for μ

- grayscale val should lie between μ_1, μ_2
- variable scale quantization



Decompression

1. Build Q from treecode
2. Reconstruct grayscale val's: $a = code \cdot \max\left(1, \frac{\theta_2}{3}\right) + \mu$
3. optional: remove block artifacts

Performance:

- decompression always faster than JPEG
- compression w/ 1 bit/pixel → 2x faster than JPEG

Isosurface Construction

Given: $f : \mathbb{R}^3 \to \mathbb{R}$ "scalar field"

Definition:

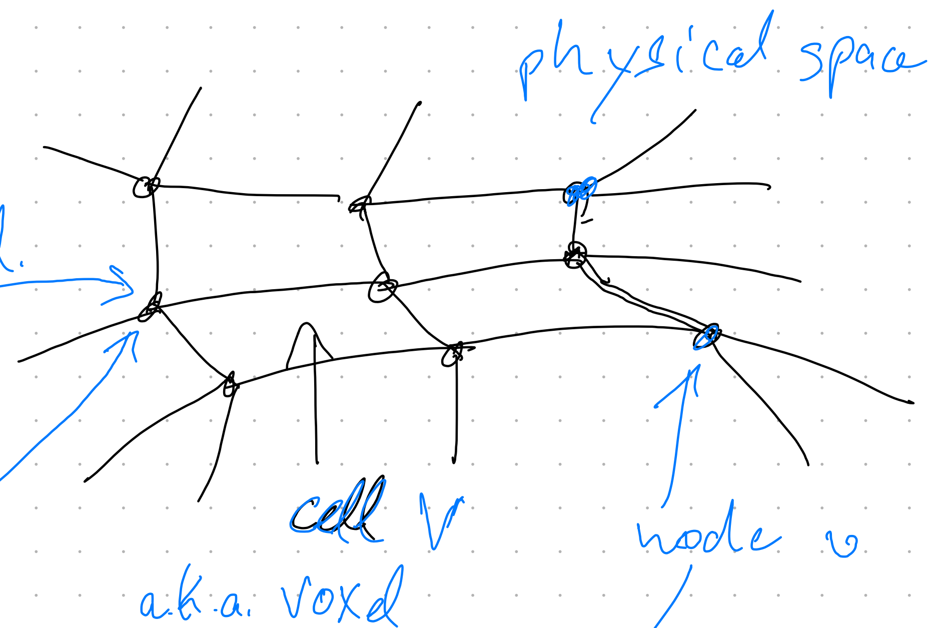
Isosurface A_τ with isovalue τ is the set

$$A_\tau := \{ x \in \mathbb{R}^3 \mid f(x) = \tau \}$$

Curvilinear grid:

Represent by 3D array

$$F[i][j][k] = (p_{ijk} \in \mathbb{R}^3 , f_{ijk} \in \mathbb{R})$$



Isosurface in discrete space

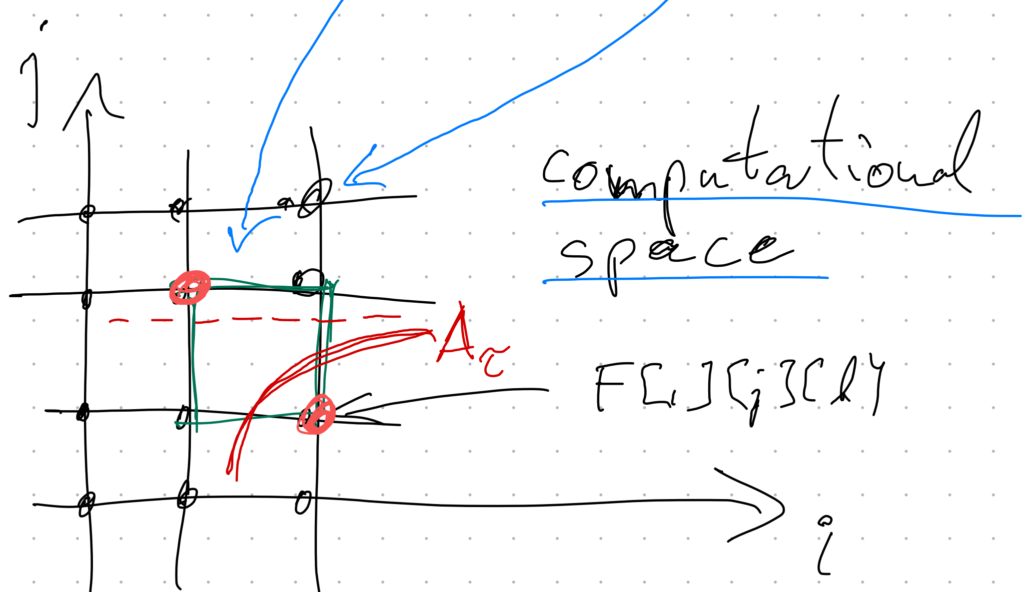
Surface A'_τ, such that

$\forall$ voxels V intersecting A'_τ:

$\exists v_i \in V : f(v_i) \leq \tau \ \wedge$

$\exists v_j \in V : f(v_j) \geq \tau$

$V = \{ v_1, \ldots, v_8 \}$

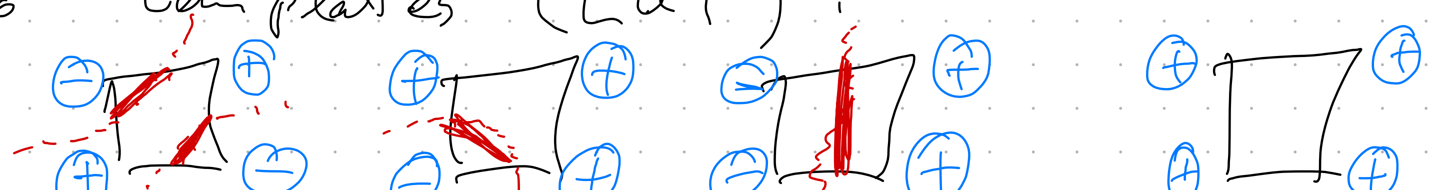


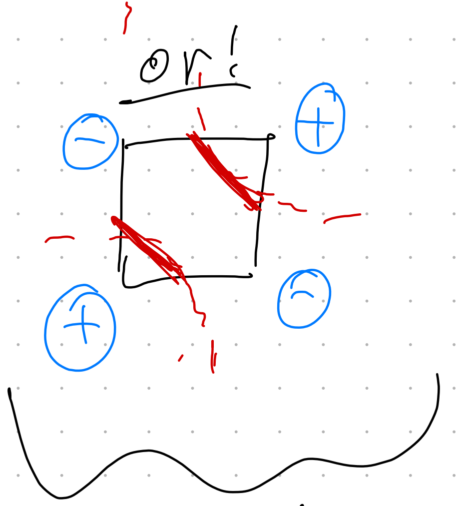
Earliest algo: "marching cubes" (1987)

iterate over all voxels V:

calc signs of nodes of V ($\ominus$: $v_i < \tau$, $\oplus$: $v_i > \tau$)

triangulate according to templates (LUT):





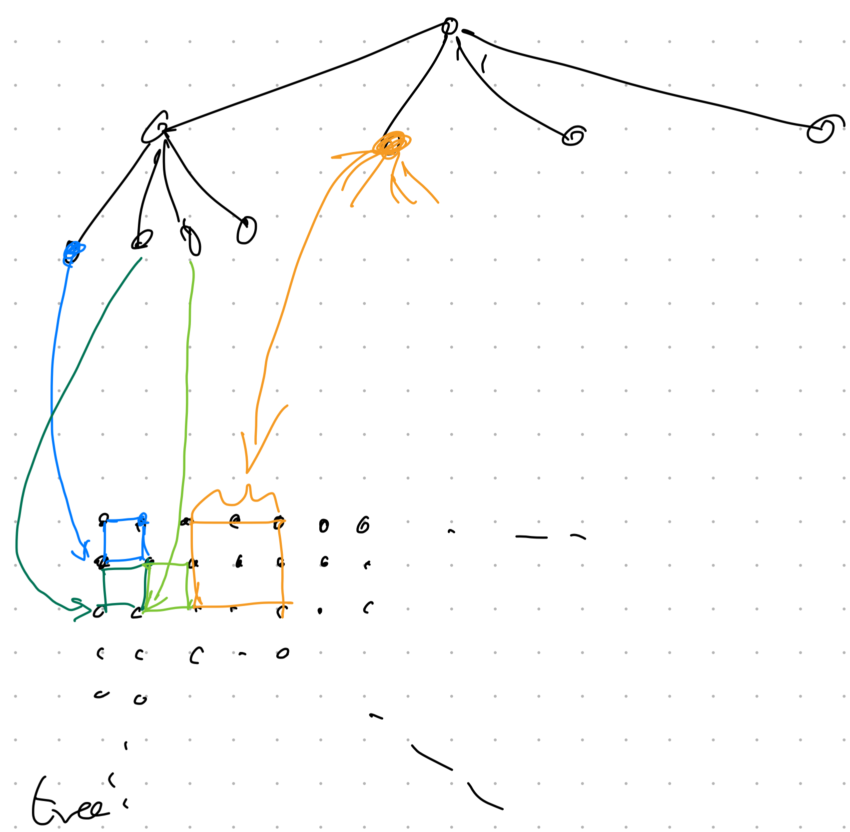
ambiguous

Isosurface-Construction using Octrees

Solution: Min-Max Octree

Construct complete octree over voxels nodes of scalar field.

Leaves point to lower left corner of voxels



Each leaf stores

$\min\{v_i\}$, $\max\{v_i\}$

where v_i = nodes of the voxel

Propagate min/max up through tree:

→ inner nodes store min(children), max(...)

Note:

Iso surface must pass through a voxel region of an octree node v $\Leftrightarrow$ $\min(v) \leq \tau \leq \max(v)$.

Algo: recursion through octree

Optimization:

- Hash table for edges of the voxel grid
- Remove entries when 2x visited
- Proceed in z-order → small hash table

Interlude : Span Space

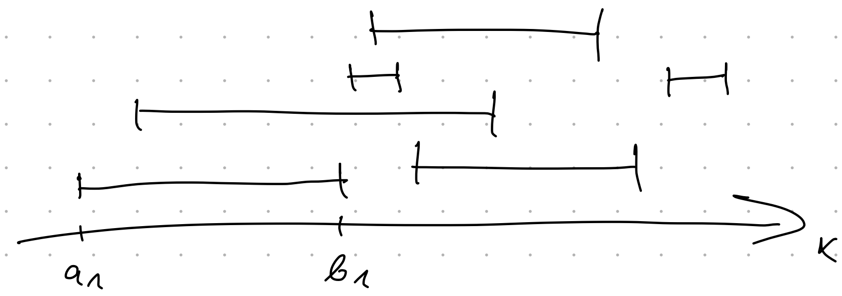
Problem: "1-dim stabbing query"

Given: N intervals $[a_i, b_i] \subseteq \mathbb{R}$

θ = "query pt"

Sought: all intervals with

$\theta \in [a_i, b_i]$



Standard algo: interval tree, segment tree

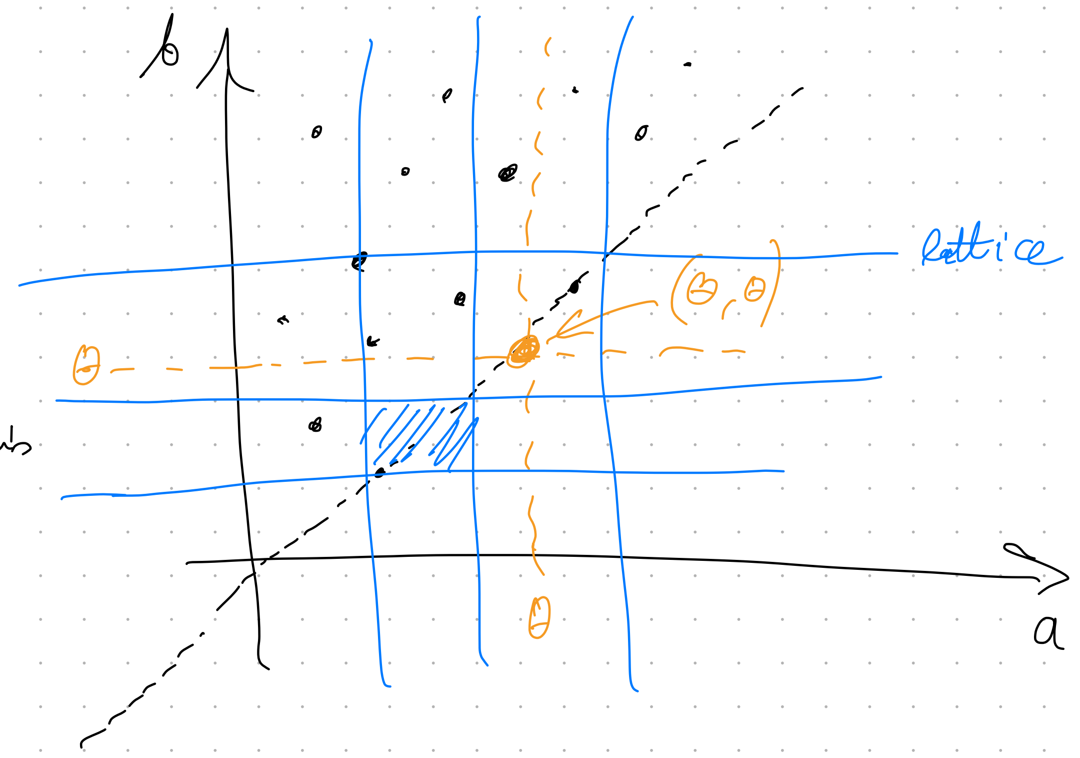
Idea: consider intervals $[a,b]$ as pts (a,b) in $\mathbb{R}^2$

→ "span space"

Data structure:

Overlay span space with lattice of $L \times L$, s.t.

1. lattice line/grid distances are equal along a- and b-axis
2. pts are distributed approx uniformly among lattice lines



(1) ⇒ lattice cells are intersected diagonally, or not at all

(2) can be achieved by sorting a-/b-values together.

For each row i of lattice, store two lists:

1) L_i^a = { pts in row i up to cell $(i-1)$ }
   sort L_i^a by a-value ascending
2) L_i^b = { ..... }, sort by b-value descending

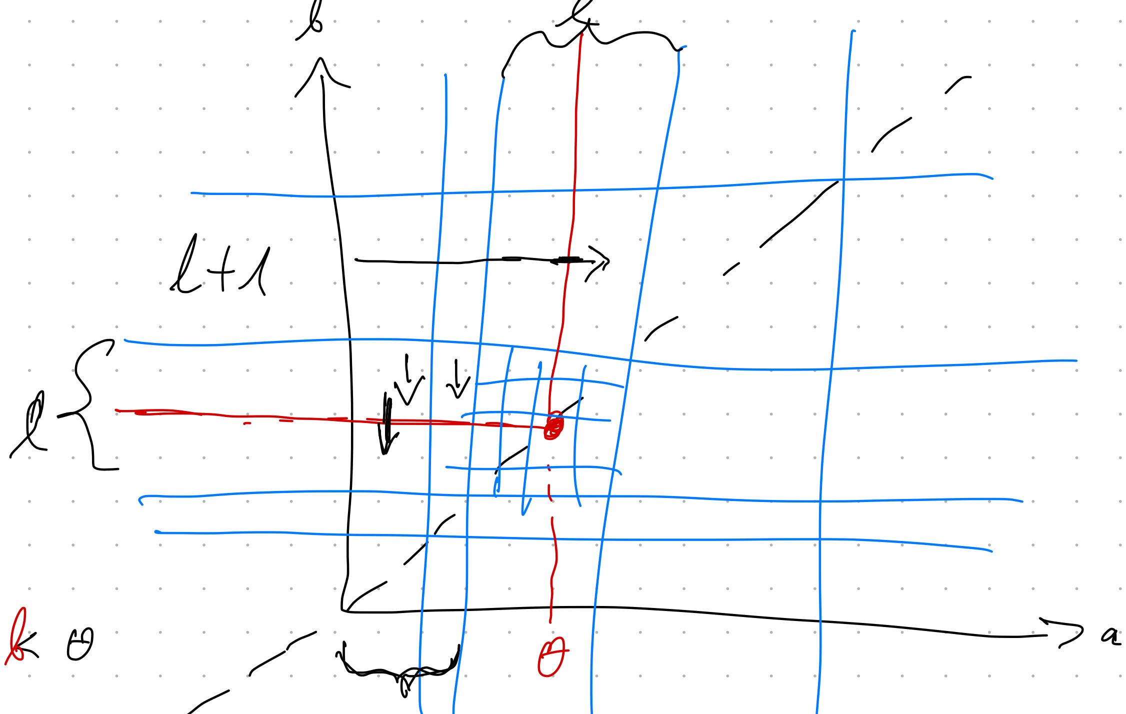
For pts on diagonal cell: construct lattice recursively
(or: store in simple array, if not "too many")
(or: use interval tree)

Algo:

find lattice cell (l, l) containing θ

for rows $i = l+1, \ldots, L$:
  traverse L_i^a up to $L_i^a[j].a > \theta$

for row l:
  traverse L_l^b up to $L_l^b[j].b < \theta$



for cell (ℓ, ℓ):
recursion into "sub-lattice"
(or: exhaustive search)

Running time: expected r.t. assuming uniform distribution
→ each cell contains $\frac{n}{L^2/2} = \frac{2n}{L^2}$ pts

$$T(n) = O(\log L) + O(L) + T\left(\frac{2n}{L^2}\right) + O(k)$$

where k = # output set

⇒ choose $L = \log n$ ($L = \sqrt{n}$?)

$$T(n) = O\left(L \cdot \log_{L^2/2}(n) + k\right) = O\left(\frac{\log^2 n}{\log\log n}\right)$$

Isosurface over time-varying field

Given: N 3D scalar fields, for $t_i \in \{t_0, t_{N-1}\}$

Def.:

$\min_t(v) :=$ min { nodes of voxel v at time t }

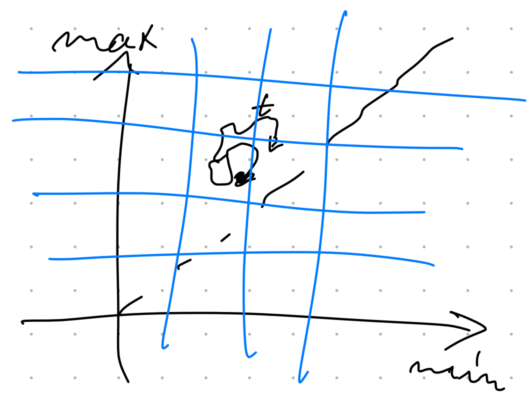
$\max_t(v) = \ldots$

$\min_i^j(v) := \min\{\min_{t_i}(v), \ldots, \min_{t_j}(v)\}$

$\max_i^j(v) = \ldots$

Consider $(\min_t(v), \max_t(v))$ in span space, consider its trajectory over time

Def.: cell $v \in V$ has "small temp. variation" :⇔ all pts $(\min_t(v), \max_t(v))$, $t = i, \ldots, j$, are contained in 2×2 contiguous cells in span space



Construct Temporal Index Tree (TI-tree):

Start V (all voxels) and T_0^N

create span space of V and interval $[0,N]$

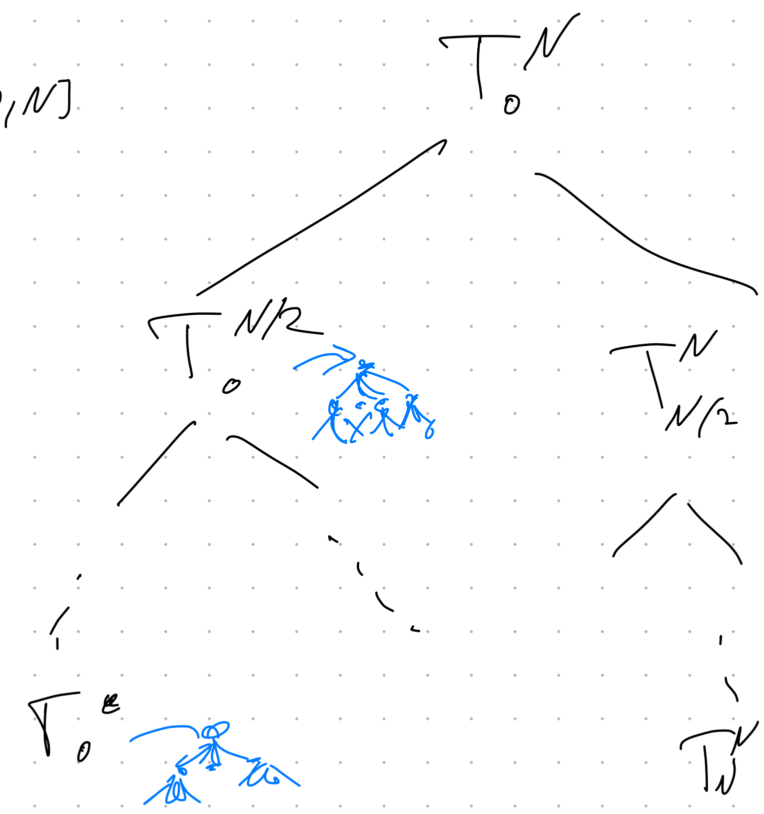
for each $v \in V$:

  check cond. "small temp. variance" over time $[0,N]$

  if yes $\Rightarrow$ add v to $V(T_0^N)$

recursion with $T_0^{N/2}$ and $T_{N/2}^N$

  and $V \setminus V(T_0^N)$



build octree for each node in the TI-tree for $V(T_i^j)$