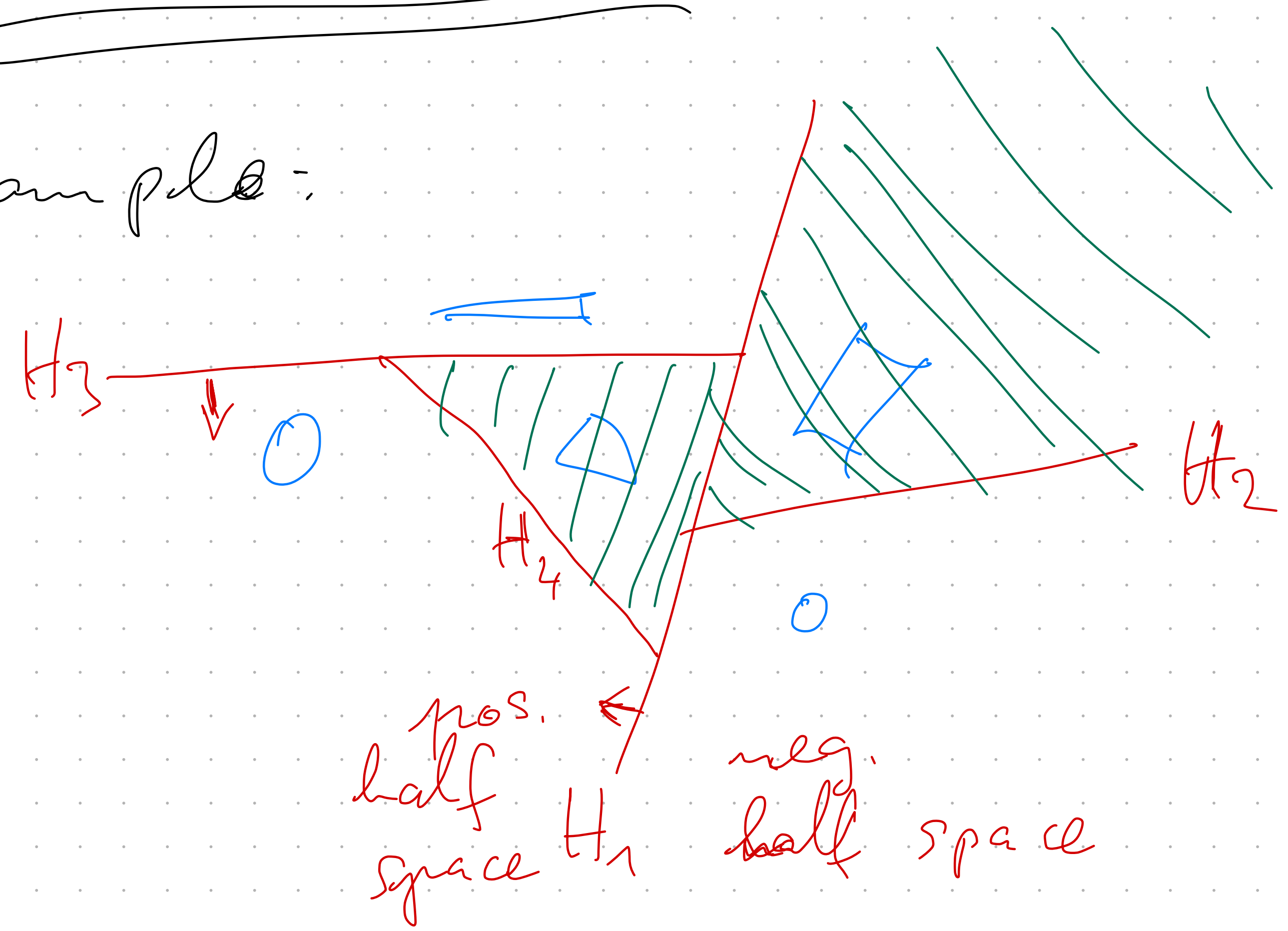
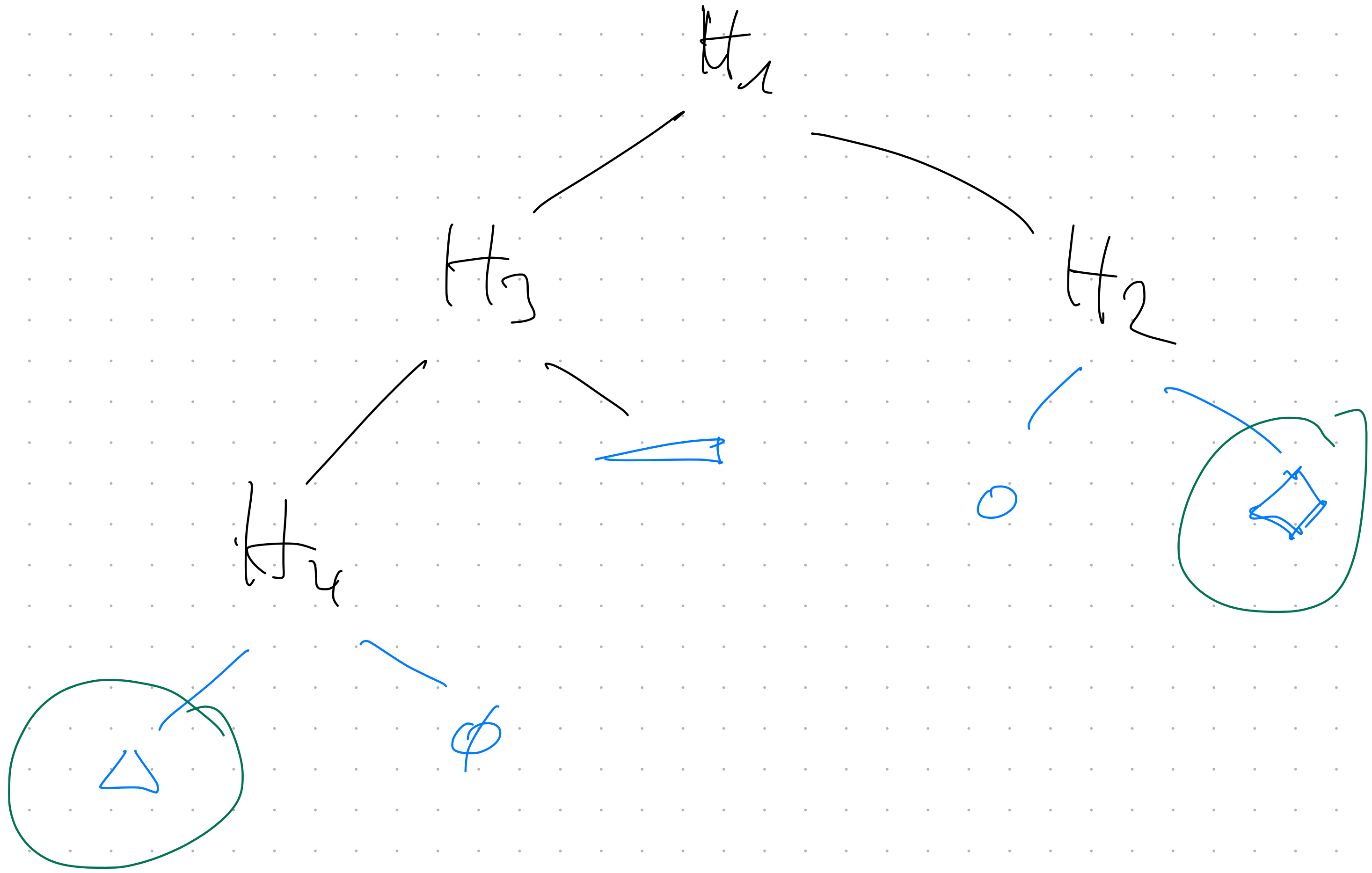
BSP Trees

BSP Trees

Example:





Def's:

Let S = set of polygons / line segments in 3D/2D, resp.

H = plane / line, H^+ = pos. halfspace, H^- = neg half space

If $|S| \le 1 \Rightarrow$ BSP T over S is leaf v,

v stores $S(v) = S$

If $|S| > 1 \Rightarrow$ BSP T over S is node v, where

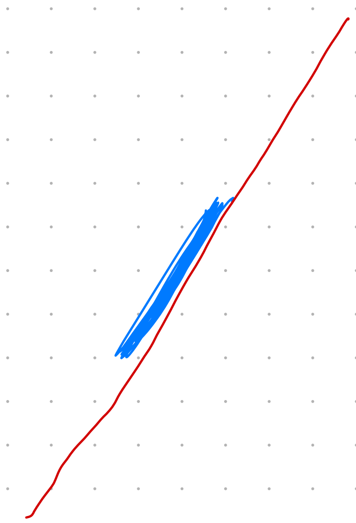
v stores
- H_v = plane
- $S(v) = \{ p \in S \mid p \subseteq H_v \}$
- pointers to children T^- and T^+

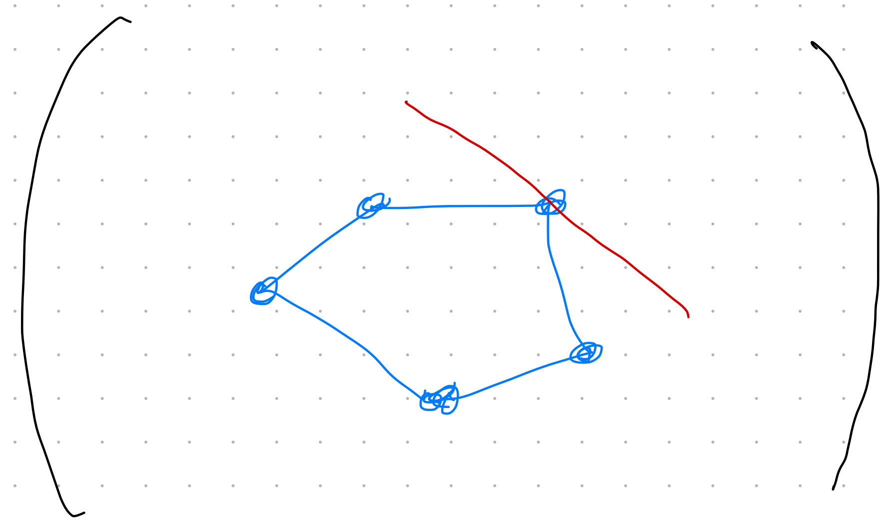
where T^- is BSP over $S^- := \{ p \cap H_v^- \mid p \in S \}$

T^+ is BSP " $S^+ := \{ \underbrace{p \cap H_v^+}_{\text{fragments}} \mid p \in S \}$

$R(v) :=$ region of v = convex subset of $\mathbb{R}^d$ covered by v

Supporting plane / line:

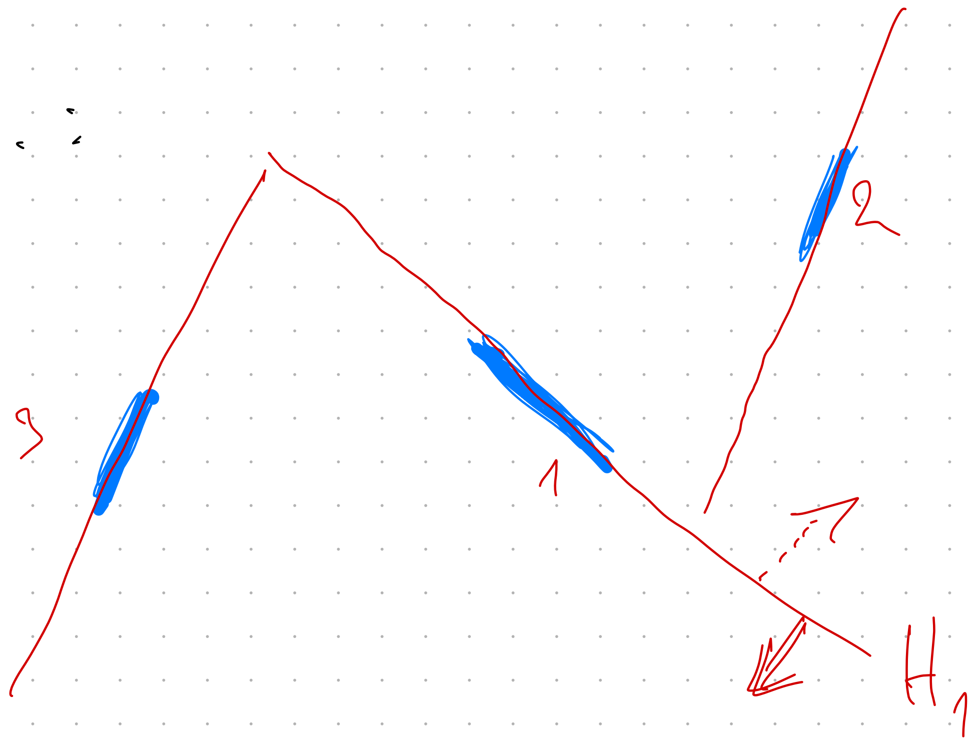


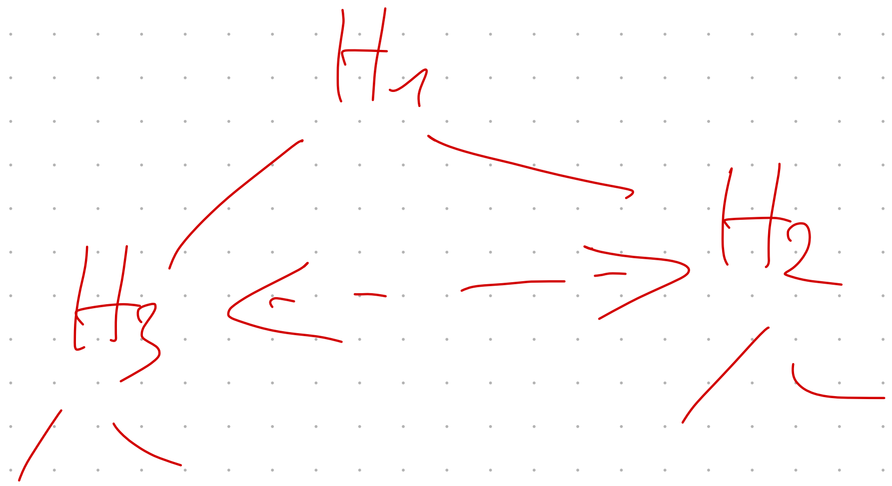


Def.: Auto-Partition

A BSP where all splitting planes/lines H_v are supporting planes/lines of one pgon/line in S.

Ex.:





Order of splitting planes / orientation ... ?

Construction of Auto-Partitions (in 2D)

Input: S = set of line segments in $\mathbb{R}^2$

if $|S| \leq 1$:

   $T :=$ leaf v with S

else:

   choose $s_1 \in S$ as splitting line,

   let L_{s_1} = supporting line of s_1

   compute $S^+ := \{ s \cap L_{s_1}^+ \mid s \in S \}$, $S^- := \ldots$

   $T^+ := BSP(S^+)$, $T^- := BSP(S^-)$

   $T :=$ node v, children T^+, T^-, store L_{s_1} and $S(v) \subseteq S$

$S(v) \ni s_1$

return T

Randomize: scrambling S randomly at the beginning

Lemma: (only in 2D!)

Let $n = |S|$; then expected # fragments in BSP $\in O(n \log n)$, and construction time $\in O(n^2 \log n)$.

Size of BSP = # fragments $= \sum_{v} |S(v)| = \bar{n}$

Case: every inner node v splits S in two non-empty subsets,

$\Rightarrow$ # nodes $= 2 \cdot$ # leaves $- 1 = 2 \cdot$ # fragments $- 1$

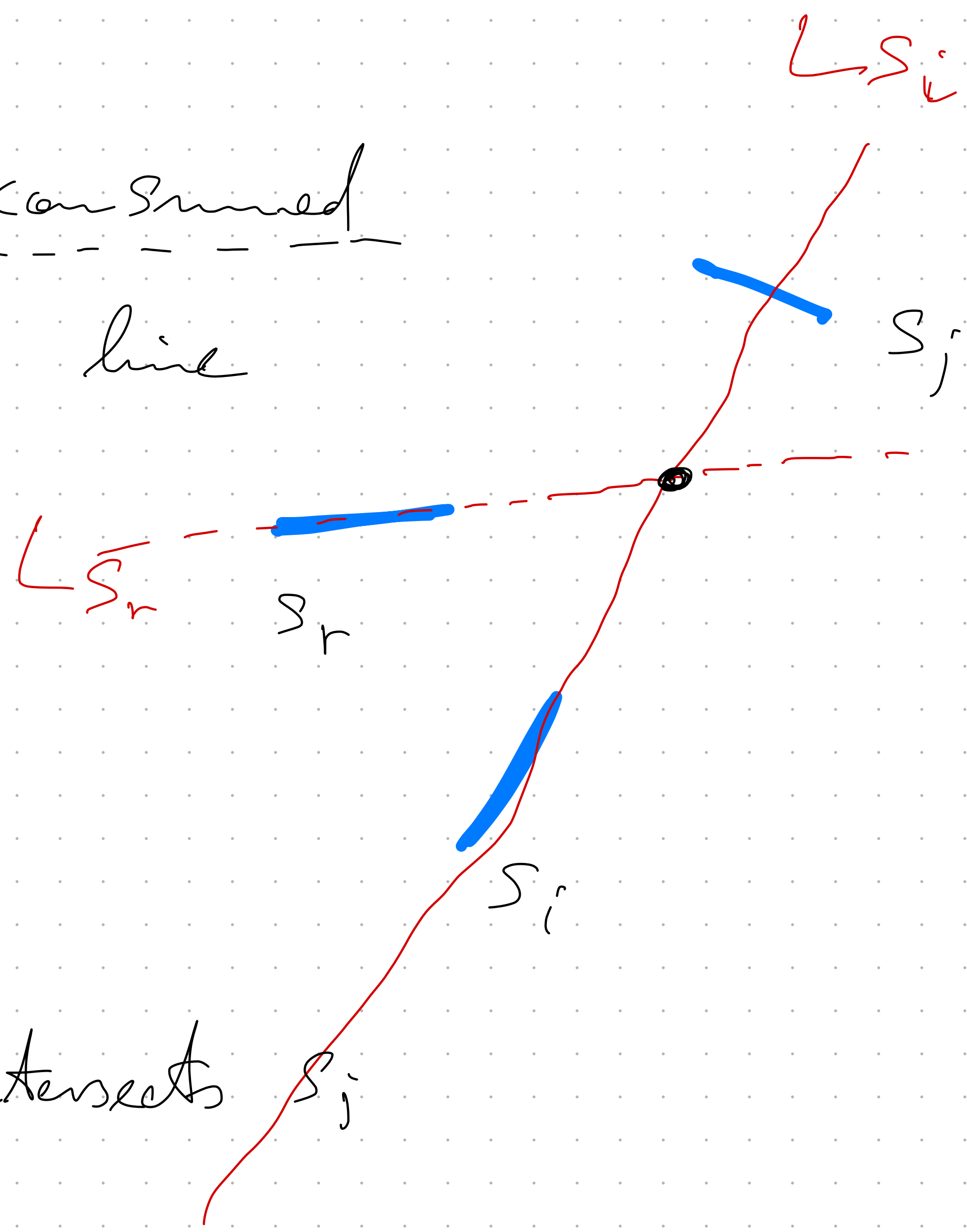
Case: every inner nodes stores exactly one polygon $\in S$, $|S_v| = 1$

$\Rightarrow$ # nodes $\in O($# fragments$)$

Proof of lemma:

Let s_j = another segment $\in S$ not yet consumed

s_i = currently chosen for splitting line



Condition for s_j being split / not split by s_i:

if s_r is chosen before s_i,

then if "shields" s_j from s_i

Define $\mathrm{dist}(s_i, s_j) := \begin{cases} k, & L(s_i) \text{ intersects } s_j \\ +\infty, & \text{else} \end{cases}$

where k = # segments s_r with intersection pt $L_{s_r} \cap L_{s_i}$ between s_i and s_j

L_{s_i} splits s_j $\Leftrightarrow$ $i = \min\{i, j, j_1, \dots, j_k\}$

Randomization $\Rightarrow$ probability $\Pr[L_{s_i} \text{ splits } s_j] =$

$$\frac{\#\text{ permutations }(j, j_1, \dots)}{\#\text{ permutations }(i, j, j_1, \dots)} = \frac{(k+1)!}{(k+2)!} = \frac{1}{k+2}$$

$$= \frac{1}{\mathrm{dist}(s_i, s_j) + 2}$$

$$\Rightarrow \quad E[\#\text{ splits caused by } s] = \sum_{s' \neq s} \Pr[L_s \text{ splits } s']$$

$$= \sum_{s' \neq s} \frac{1}{\mathrm{dist}(s, s') + 2} \leq 2 \cdot \sum_{i=0}^{n-2} \frac{1}{i+2} \leq 2 \cdot \ln n$$

all distances occur $\leq$ twice

$\Rightarrow$ expected # splits caused by all segments $\leq 2 \cdot n \ln n$

$\Rightarrow$ # fragments $\leq \underline{n} + 2n \ln n$

Time: # recursive calls = # fragments $\in O(n \log n)$

per call: $|S| \leq n \implies O(n^2 \log n)$

Lemma (w/o proof):

Exists a BSP with size $n + 2n \ln(n)$.

Half of all permutations yield BSP with size $\leq n + 4n^{\ln(n)}$

→ Prob. algo for "good" BSP:

pick random permutation of S

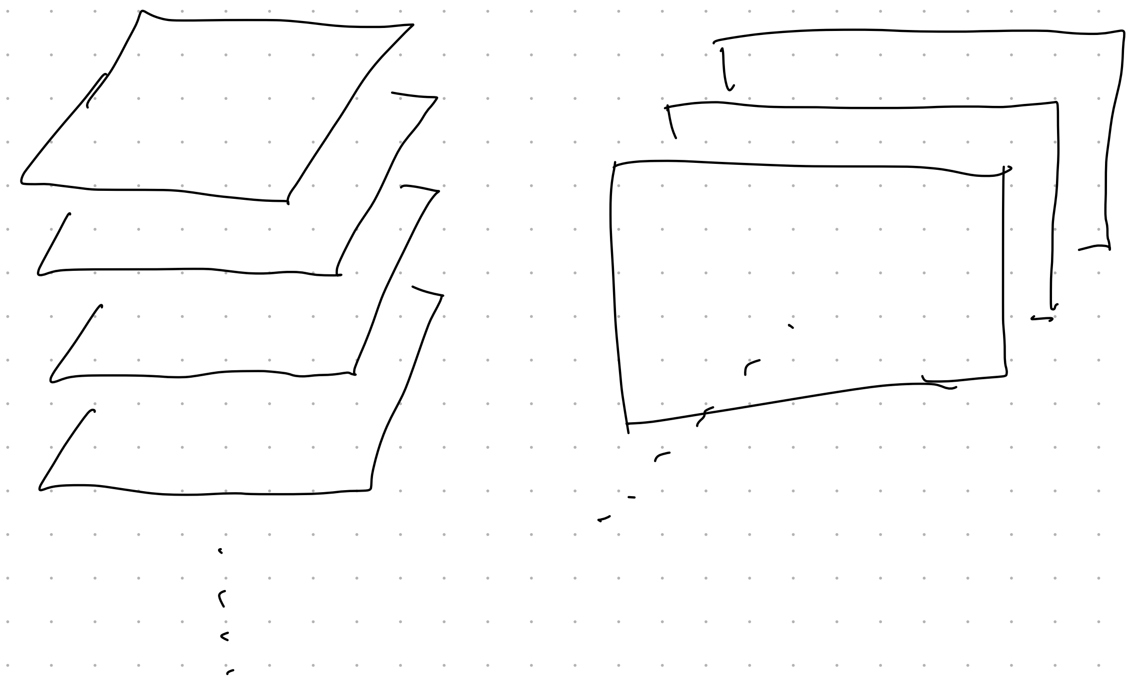
construct BSP over S

if size(BSP) > $n + 4n \ln n$: scramble S again, and construct new BSP

→ expected # iteration = 2

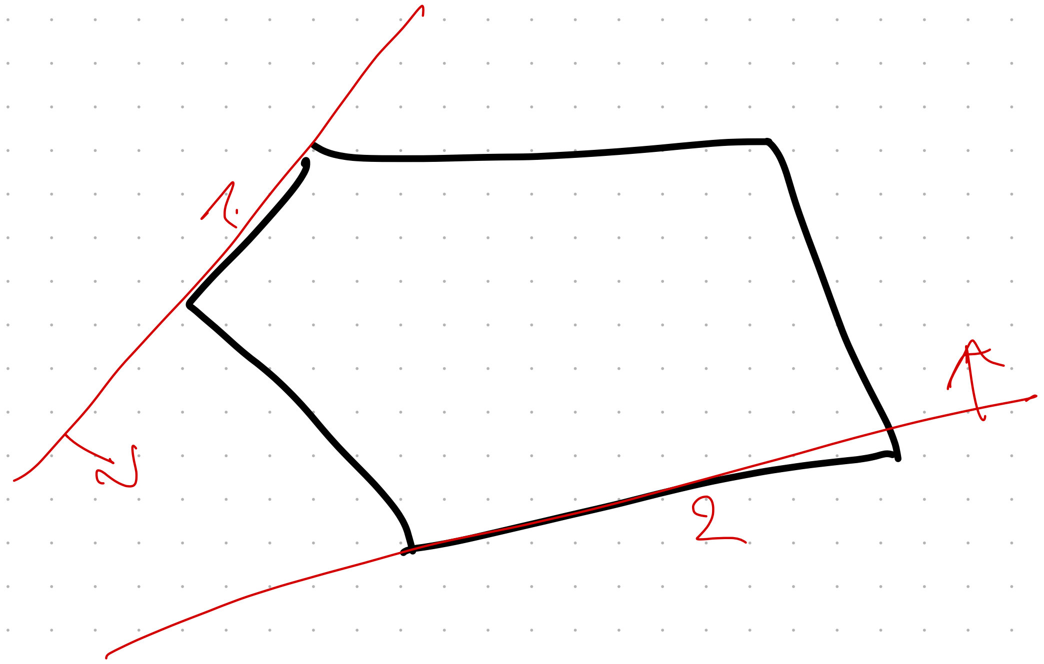
Algo in 3D : expected size $\in O(n^2)$

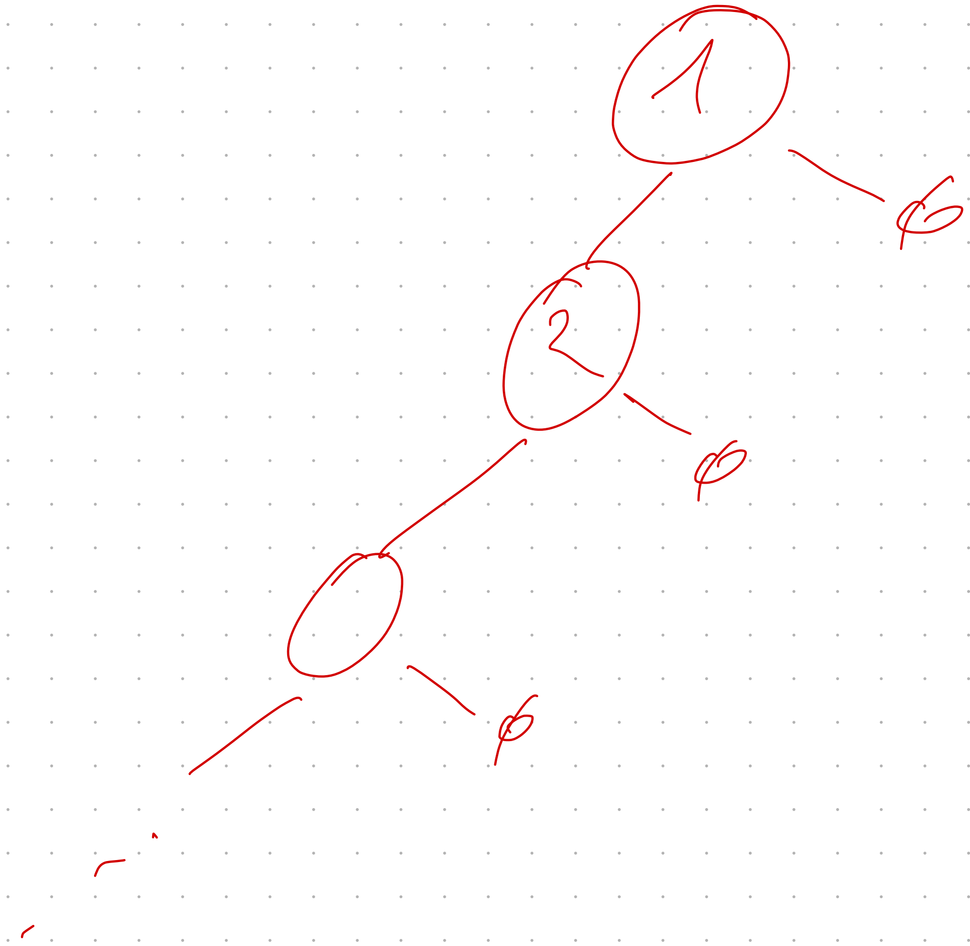
Exists example input :



$\Rightarrow n^2$ fragments

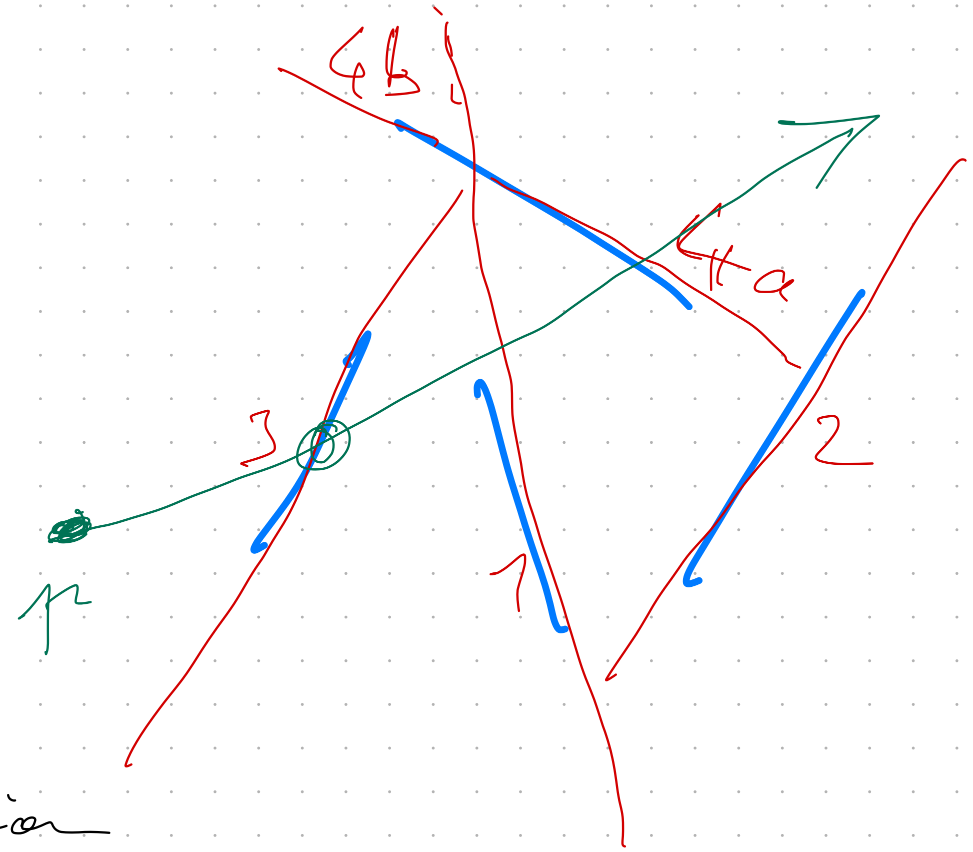
Example of BSP over convex obj:





→ auto-partition is not good

Example applications



1. Ray casting:

   Do DFS traversal,
   traverse first into subtree,
   where p is on positive side;
   in-order traversal: check intersection
   with $S(v)$

2. Rendering set of pgons without Z-Buffer:
   → render pgons back to front (sorting of pgons w.r.t. view point)
   ("painter's algo")

   render (v, p):
   if v_1 does not contain ~~p~~ viewpt:
   render (v_1, p)
   render $S(v)$

render (v_2, p) "closer subtree"

else:

render (v_2, p)

render S (v)

render (v_1, p)

Advantages: easy integrate with frustum culling

and back face culling

Quality of BSP?

Balancing vs Splits

Case: appr "=" classification (pt location query, ray query, ...)

→ reduce depth → balance splits

Case: appr "=" depth sorting (rendering, ...)

→ reduce size → reduce # fragments

Measure: costs

$$C(T) = 1 + P^- \cdot C(T^-) + P^+ \cdot C(T^+)$$

where P^- / P^+ = probabilities that traversal at runtime enters T^- / T^+ subtrees

Example: pt location problem

Query: q, which leaf BSP contains q

$$P^- = \frac{Vol(R(T^-))}{Vol(R(T))}$$

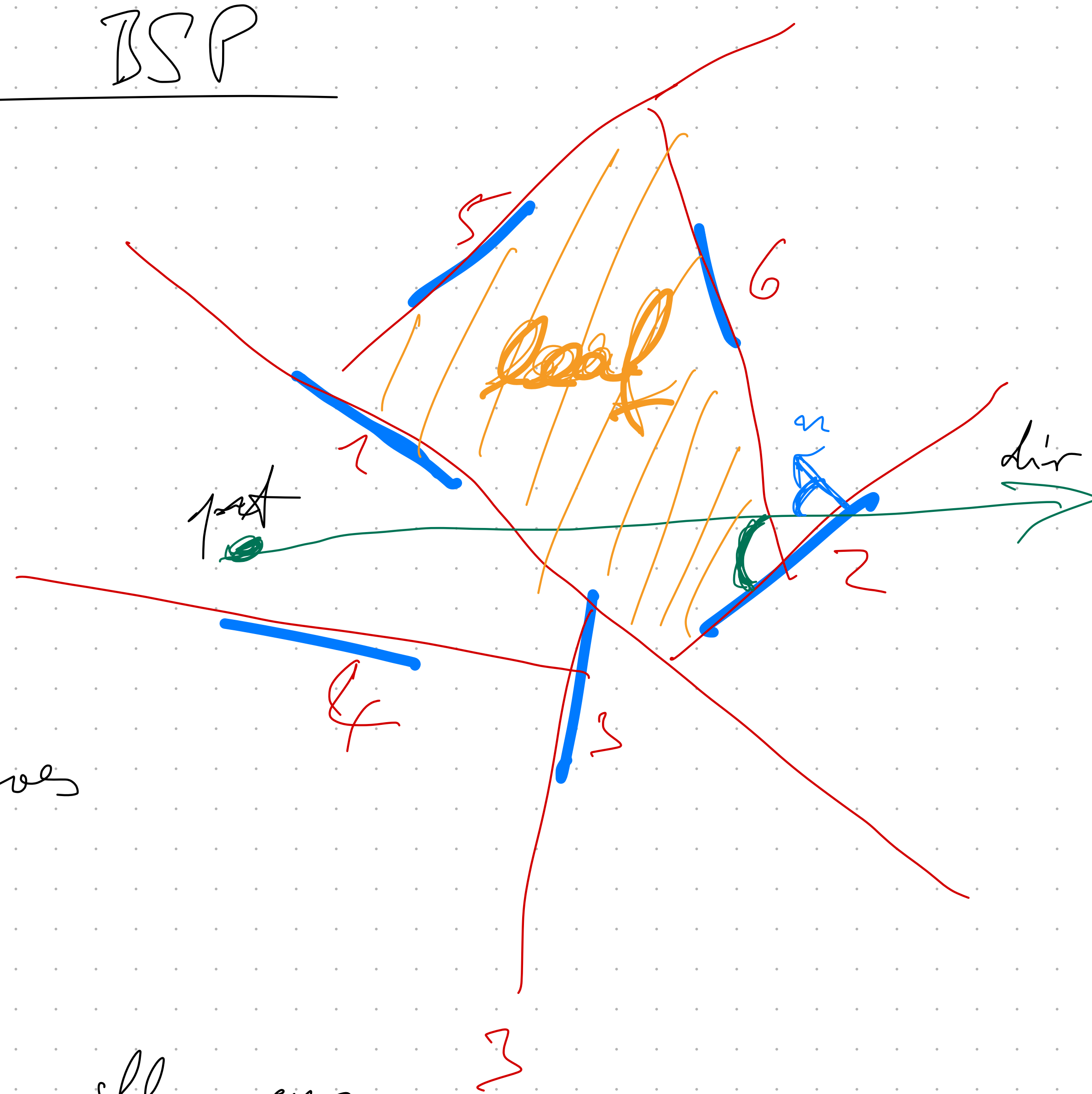
Deferred, Distribution-Optimized BSP

(a.k.a. Self-Organizing BSP)

Consider: ray shooting query

Costs of BSP:

$$C(T) = \#\text{ nodes visited}$$
$$\leq \text{depth}(T) \cdot \#\text{ stabbed leaves}$$



Consider probability density fct

$\omega : D \to \mathbb{R}$, D = set of all possible rays

let $\ell \in D$ a ray, has pt & dir $\to$ $D \subseteq \mathbb{R}^5$

S = set pgons of scene, let $p \in S$

Define $\text{score}(p) = \displaystyle\int_D \omega(\ell) \cdot W(p, \ell)\, d\ell$

weight $w(p, \ell) = |n \cdot \ell_{dir}| \cdot \dfrac{\text{area}(p)}{\text{area}(S)}$ $\leftarrow$ Normalization

→ easy improvement of BSP constr.:

sort S by score(p)

at each node: pick random pgon among "top k"

Augment BSP: on-demand construction

Nodes v store:

1. Plane H_v, pgon(s) P_v, potent'ly region $R(v)$
2. if preliminary leaf:

   list $\mathcal{L}(v) \subseteq S$, pgons associated w/ v

   $A(v)$ = visit counter

For each pgon $p \in S$, $A(p)$ = "hit" counter

Algo: testray (ℓ, v):

if v is leaf:

increment $A(v)$

test ℓ against all $\mathcal{L}(v)$,

increment $A(p)$ for all $p \in \mathcal{L}(v)$ that are intersected

if $A(v) \geq$ threshold:

subdivide v

return (min) hit pt, or "none"

else

let v_1 = child of v on same side as start pt of ℓ

hit-pt := testray (ℓ, v_1)

if no hit in v_1:

hit-pt := testray (ℓ, v_2)

return hit-pt

Subdivision of preliminary leaves:

1. When:
   split, if $A(v) \geq$ threshold

   ↑ absolute, better relative

2. How:
   $A(p)$ = hit ctr of polygon p, increment whenever hit

   if split, use p^* where $A(p^*) = \max$

   maintain L as a heap

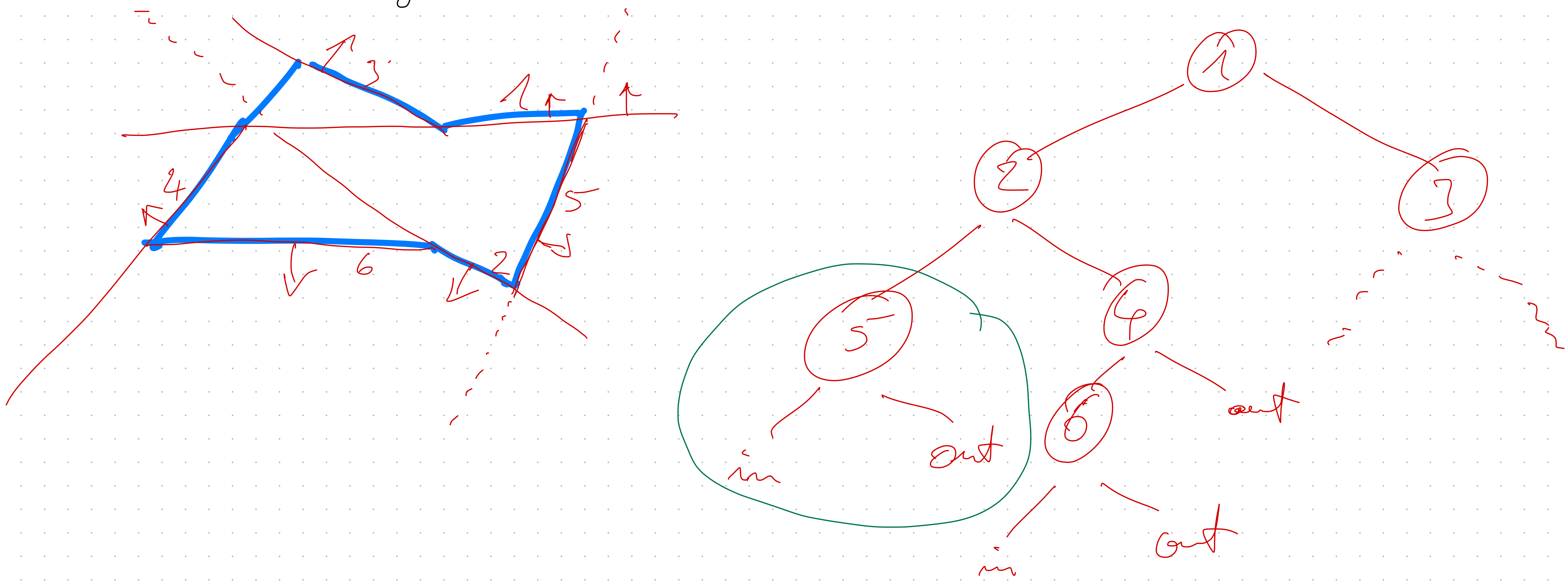
Notes:

Alternative to (2): caching strategies, e.g. LRU → not as good

Performance gain: factor 2-20

Object representations using BSP

Leaves: meaning = "inside" or "outside"



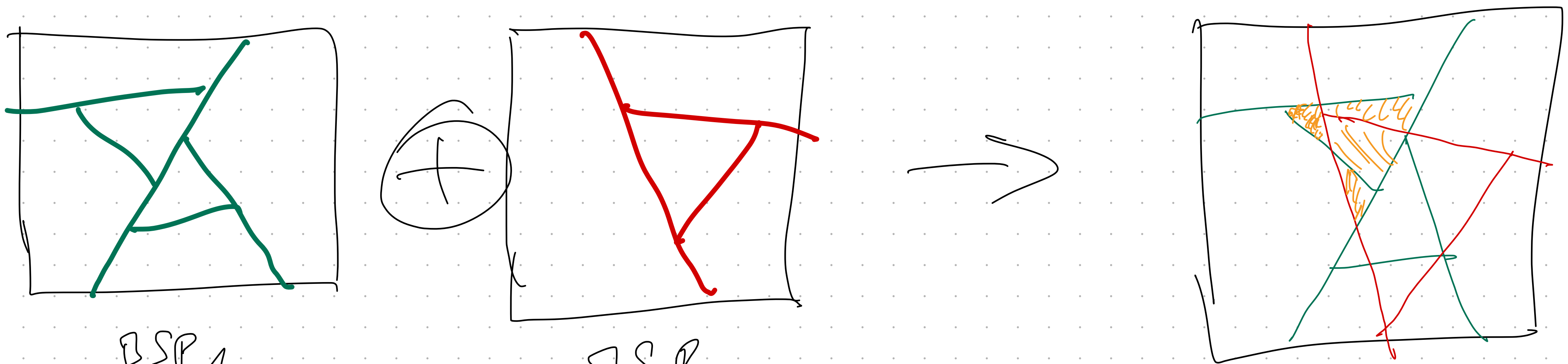
Merging BSP's:

$BSP_1 \oplus BSP_2 \longrightarrow BSP_3$

↑ operation $\in$ "$\cap$", "$\cup$", "$-$" (set op's)

BSP_3 new BSP where all leaves are

$$\mathcal{C}_3 = \{ c_1 \oplus c_2 \mid c_1 \in \mathcal{C}_1 ,\ c_2 \in \mathcal{C}_2 \}$$



BSP_1 $\qquad$ BSP_2

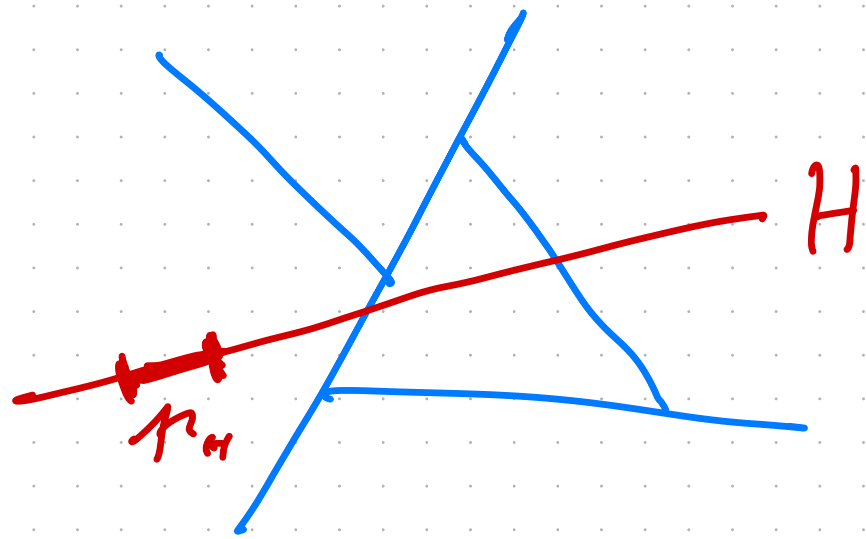
Start: consider

given BSP T, plane H (including pgon $p_H \subseteq H$)

sought: new BSP $\hat{T}$, with root H

partition-tree $(T, H) \to \hat{T}$

$$(T^{\ominus}, T^{\oplus}) := \text{splittree}(T, H, H)$$



$$\hat{T} := (H,\ p_H,\ T^{\ominus},\ T^{\oplus})$$

↑ pgon in H, for autopartition

$\mathrm{SplitTree}(T, H, P) \to T^{\ominus}, T^{\oplus}$

output: $T^{\ominus} = T \cap H^-$, $T^{\oplus} = T \cap H^+$

input: T = root of BSP $= (H_T, p_T, T^-, T^+)$ in case of inner node

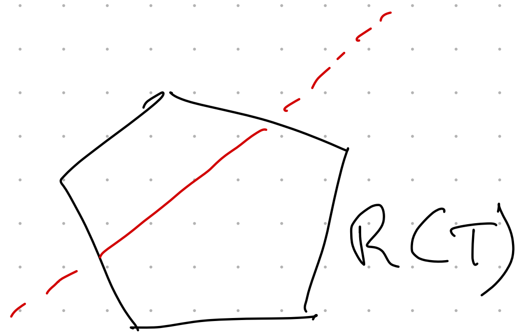
in case of leaf: T = "in" / "out" label

H = split plane

$P = H \cap R(T)$

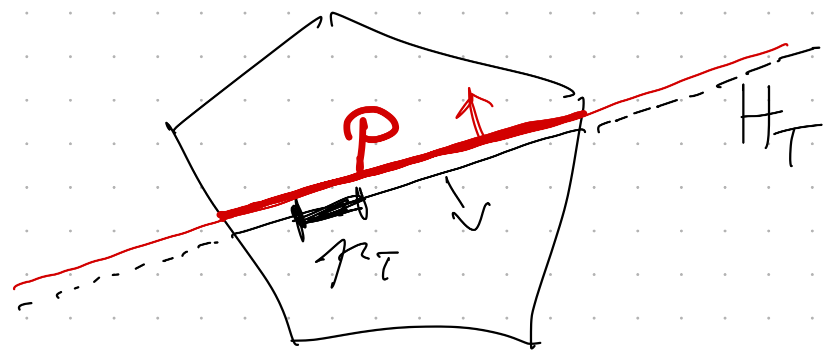
case T is leaf:

return $T^{\ominus}, T^{\oplus} := (T, T)$



Case: H and H_T coplanar, but with opposite normals:

$$T^{\ominus} := T^+, \quad T^{\oplus} := T^-$$



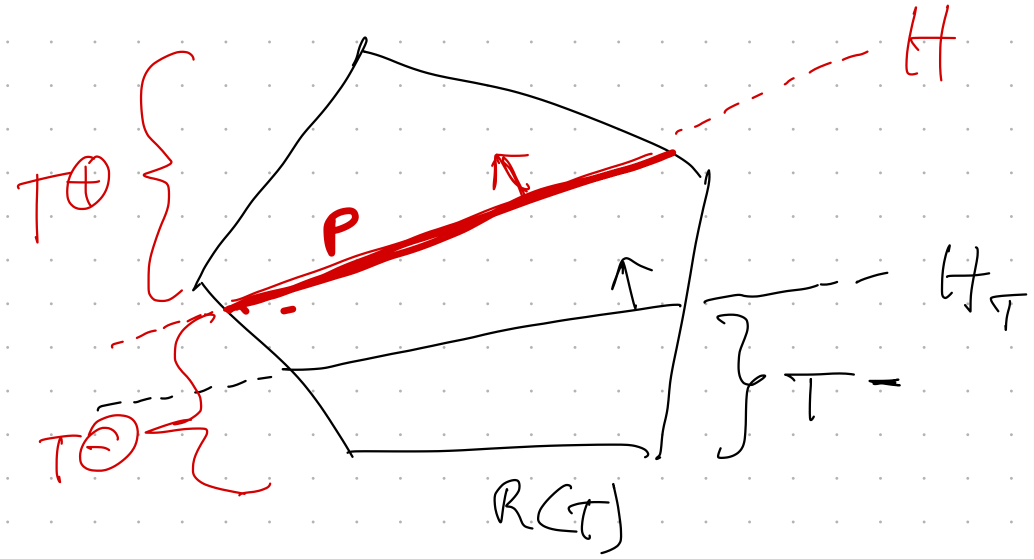
Cases: coplanar with normals equal: analog

Case: "pos./pos."

$T^{+\ominus}, T^{+\oplus} = \text{splittree}(T^+, H, P)$

$T^{\ominus} := (H_T, p_T, T^-, T^{+\ominus})$

$T^{\oplus} := T^{+\oplus}$



Cases: "neg/neg", "pos/neg", "neg/pos":

analog

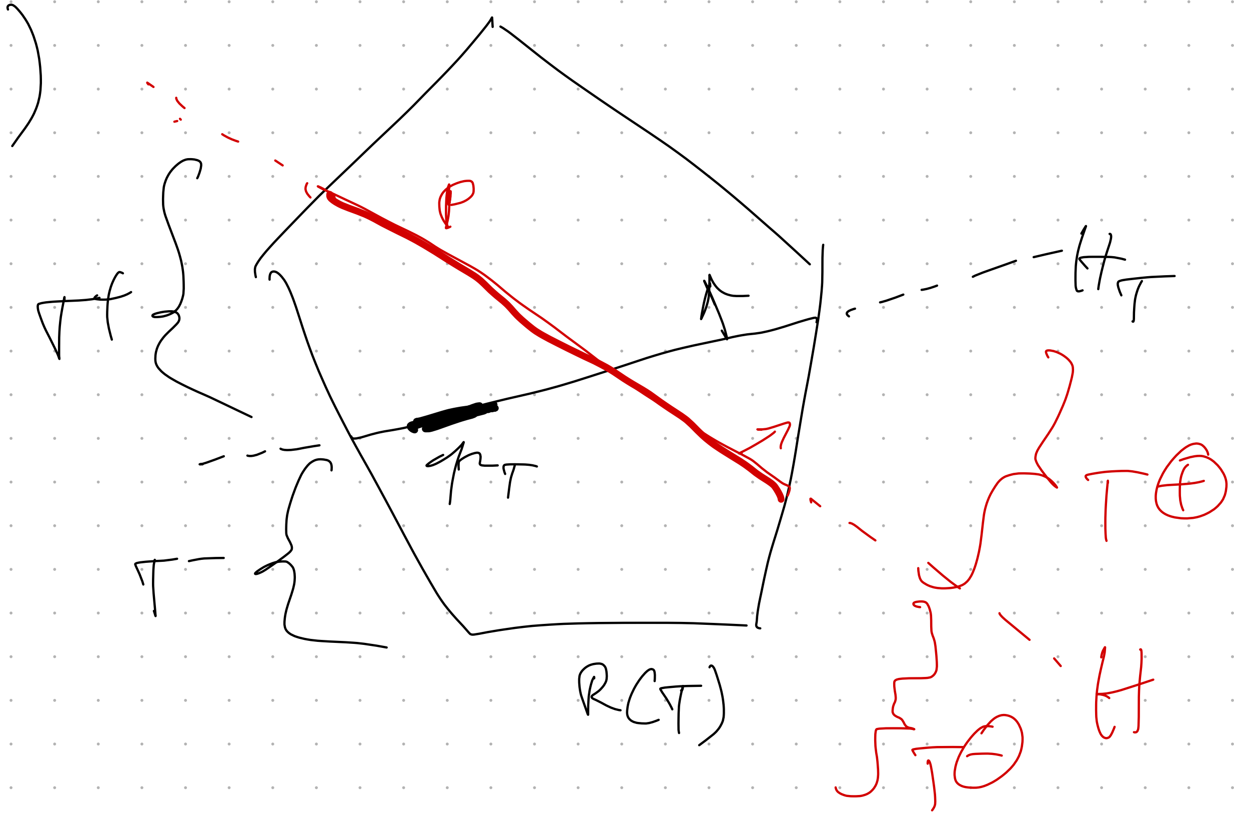
Case: "mixed"

$T^{+\ominus}, T^{+\oplus} := \text{splittree}(T^+, H, P \cap R(T^+))$

$T^{-\ominus}, T^{-\oplus} := \text{splittree}(T^-, H, P \cap R(T^-))$

$T^{\ominus} := (H_T, p_T \cap H^-, T^{-\ominus}, T^{+\ominus})$

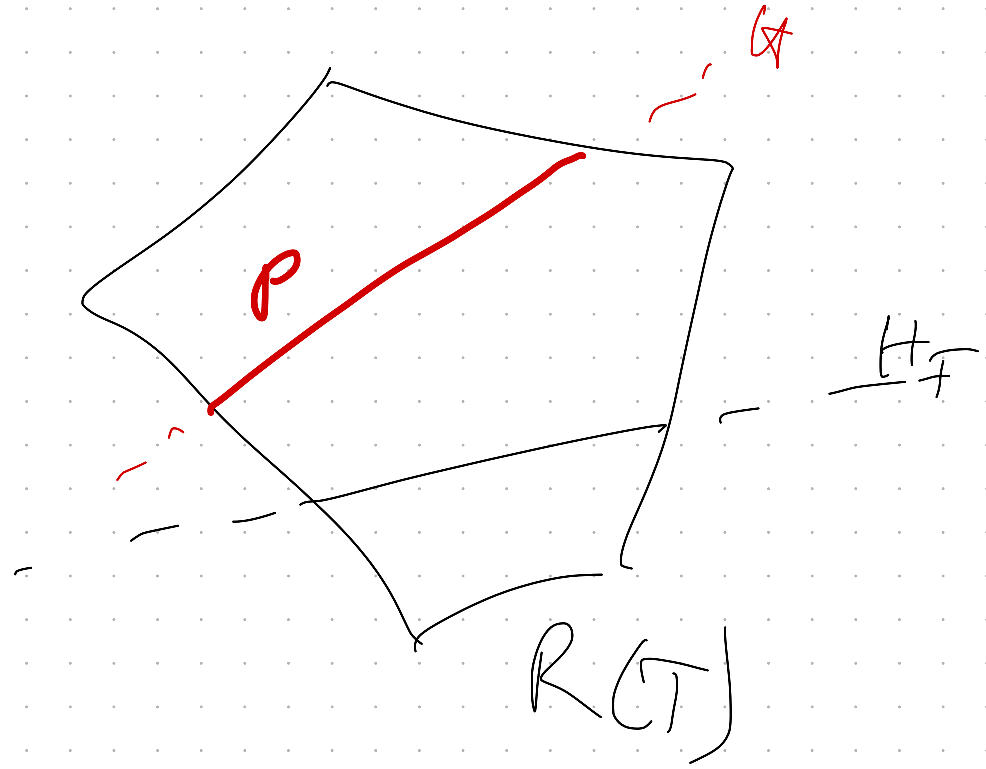
$T^{\oplus} := (H_T, p_T \cap H^+, T^{-\oplus}, T^{+\oplus})$



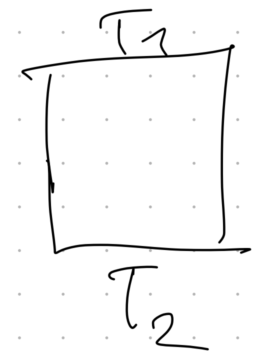
Note: splittree doesn't do much work!
classification, and calculating $p \cap H^+$, $p \cap H^-$
or $P \cap R(T+)$ ...

brute-force:
check all vertices of P
against H_T

Calc. $P \cap R(T-)$:
brute-force check all edges,
or binary search,
or golden section search



Algo: merge$(T_1, T_2) \longrightarrow T_3$



Precond.: $R(T_1) = R(T_2)$

if T_1 or T_2 is leaf:

return merge-op(T_1, T_2) (perform set op.)

else:

let T_i = BSP with root node (H_i, r_i, T_i^-, T_i^+)

$T_2^{\ominus}, T_2^{\oplus}$:= splittree$(T_2, H_1, \dots)$

opt.

$R(T_2^{\oplus}) = R(T_1^+)$

$R(T_2^{\ominus}) = R(T_1^-)$

T_3^- := merge$(T_1^-, T_2^{\ominus})$

T_3^+ := merge$(T_1^+, T_2^{\oplus})$

$T_3 := (H_1, r_1, T_3^-, T_3^+)$

merge-op(T_1, T_2):

oper.	T_1	Result
$\cup$	in	T_1 (= in)
	out	T_2
$\cap$	in	T_2
	out	T_1 (= out)
$\setminus$	in	T_2^c (complement of T_2)
	out	T_1