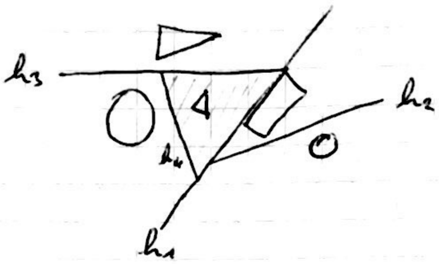
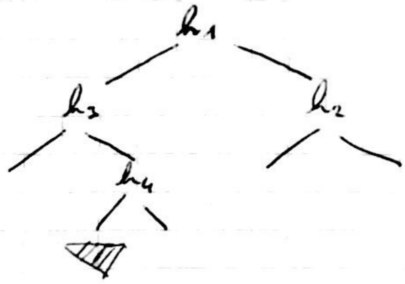
BSP – Trees

[de Berg et al.: Comp. Geom. Book]

Noch weitere Verallgemeinerung von k-d-Trees,
jetzt beliebige Orientierung der Partitionierungsebene zugelassen.

Bsp.:





(nach Def)

Def.:

h = eine Ebene, h^+ = pos. Halbraum, h^- = neg. H.-raum.

Sei S Menge von Objekten (Polygone) im $\mathbb{R}^d$.

Falls $|S| \le 1 \rightarrow$ BSP^T ist Blatt v, welches $S = S(v)$ speichert.

Fall $|S| > 1 \rightarrow$ Wurzel des BSP^T von S ist Knoten v;

v speichert Ebene h_v und $S(v) = \{ x \in S \mid x \subseteq h_v \}$

(alle Obj, die vollständig in h_v liegen);

v hat 2 Kinder T^- u. T^+ ("linkes" u. "rechtes" Kind),

T^- ist BSP zur Menge $S^- := \{ h_v^- \cap x \mid x \in S \}$,

analog T^+ zu $S^+ := \{ h_v^+ \cap x \mid x \in S \}$.
(Fragments)

Beob.:
Zu jedem Knoten v gehört eine konvexe (evtl. unbeschränkte) "Zelle" $R(v)$ (= Region) des $\mathbb{R}^d$. (wie bei kd-tree) (s. schraff. Zelle oben)

Zur Wurzel gehört $\mathbb{R}^d$ als Zelle.

Auto – Partition (im $\mathbb{R}^3$):

Verwendet nur diejenigen Splitting-Ebenen (supporting plane (s. nä. Seite f. Def.)), die durch die Polygone in S definiert werden. (d.h.: S enthält nur Polygone)

Analog in 2D: S enthält nur Linien-Segmente, definiert genau die Splitting-Lines.

In Praxis oft nur diese.

Different orders give different BSPs!

[Margin notes:]
$\bar{n}$ = # fragments ($= \sum |S(v)|$)
Case: every inner node v splits in two non-empty subsets, $S(v) = \emptyset \Rightarrow$ # fragments = # leaves = # nodes
Case: every inner node contains one polygon ($|S(v)| = 1$) $\Rightarrow$ # nodes = # fragments
$\Rightarrow$ # nodes $\in O$(# fragments) $\rightarrow$ Def. of size of BSP
Größe des BSP: # Fragmente $= \sum |S(v)| = \bar{n}$
Falls jeder Split eine $x \in S$ enthält, oder S in zwei nicht-leere Teilmengen splittet, d.h. keine nutzlosen Splits, so ist # Knoten $\le$ 4 · # Fragmente
[worst case: hin immer kein Fragm. ... zusammen Σ aller Blätter hat Fragm. → # ... = Σ # Blätter = ½ · ½ # Knoten]

Konstruktion eines BSP: Aufpartitionier-BSP in 2D (vgl Lemma unten)

S = Menge von Liniensegmenten in $\mathbb{R}^2$, ~~allen noch nicht abarbeiten~~,

bez. l_{s} = Linie die Segment s enthält ("supporting line")
$l(S)$

if $|S| \leq 1$ (also fast gleich Def.)

then

$T :=$ Blatt v mit S gespeichert

else

wähle $s_1 \in S$ als Splitting-Line

Berechne $S^+ := \{ s \cap l_{s_1}^+ \mid s \in S \}$, $S^- = \ldots$, $S(v) := \{ s \in S \mid s \in l_{s_1} \}$
(note: $s_1 \in S(v)$)

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

$T :=$ Knoten v, mit Kindern T^+, T^-, speichert l_{s_1} und $S(v)$

return T

Randomisiere Algo, indem S am Anfang zufällig permutiert wird.
(Für theor. Analyse)

Lemma (nur in 2D):

Anzahl Fragmente durch obigen Algo
ist erwartet $O(n \log n)$. (# Fragm. $\geq$ Größe des BSP)
Konstruktionszeit ist $O(n^2 \log n)$.

Bew.:

1. $l(s_i)$ = nächste Splitting-Line (s_i = nächstes ausgewähltes Segment)
= supporting line des nächsten ausgewählten Segmentes s_i

Sei s_j weiteres noch nicht "verbrauchtes" Segment.

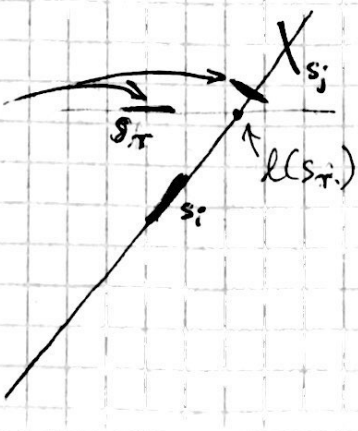
Wovon hängt es ab, ob s_j durch $l(s_i)$ gesplittet wird?

Davon, ob s_j "abgeschirmt" wird
shields

Solche Segmente "schirmen" s_j von s_i ab,

wenn sie vor s_i als Splitting-Line

ausgewählt werden.



$\Rightarrow$ Def. $dist(s_i, s_j) := \begin{cases} k & , \ell(s_i) \text{ schneidet } s_j \\ +\infty & , \text{sonst} \end{cases}$ ~~beim BSP-Aufbau~~

wobei

k = Anzahl Segmente s_r, mit Schnittpkt

$\ell(s_r) \cap \ell(s_i)$ liegt zwischen s_i und s_j.

Sei $k = dist(s_i, s_j)$, $s_{j_1}, \dots, s_{j_k}$ = Segmente "zwischen" s_i und s_j $\Rightarrow$

$\ell(s_i)$ splittet s_j $\Leftrightarrow$ $i = \min\{i, j, j_1, \dots, j_k\}$.

Reihenfolge ist zufällig $\Rightarrow$ Wahrscheinlichkeit

$Pr[\ell(s_i) \text{ splittet } s_j] = \frac{1}{dist(s_i, s_j)+2} = \frac{1}{k+2}$ $\left(\frac{\# Perm(j, \dots)}{\# Perm(i, j, \dots)} = \frac{(k+1)!}{(k+2)!} = \frac{1}{k+2} \right)$

$\Rightarrow E[\text{Anzahl Splits verursacht durch gegebenes } s] = \sum_{s' \neq s} Pr[\ell(s) \text{ splittet } s'] =$

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

(jede Distanz kommt $\leq 2\times$ vor (je $1\times$ in beide Richtungen))

$\Rightarrow E[\text{Splits insges.}] \leq 2n \ln n$

$\Rightarrow E[\# \text{Fragmente}] \leq n + 2n \ln n$ (man startet mit n Fr.)

Insbesondere: es ex. ein BSP zu S mit $\leq n + 2n \ln n$ Fragm.!

Beh. (o. Bew.): Die Hälfte aller Perm. liefert BSP mit $\leq n + 4 n \ln n$ Fragm. ?

$\Rightarrow$ Algo (prob.) für "guten" BSP:
~~Zuf.~~ wähle zufällige Perm.
generiere BSP
falls zu schlecht (= groß) ($\Leftrightarrow \geq n + 4 n \ln n$), nochmal

[Mulmuley: Comp. Geom., Intro through Randomized Algos]

Erwartete Anzahl Versuche = 2 [s.a. Blanggalles Buch]

2. Konstr. zeit:

Anz. rek. Fkt. aufrufe = Anz. Fragmente = $O(n \log n)$;

pro Fkt. aufruf wird Menge S übergeben, mit $|S| \leq n$

$\Rightarrow n^2 \log n$

Frage: warum kann man nicht, wie b. Quicksort, $n \log n$ zeigen?
$\Rightarrow$ weil hier, anders als Q.sort, S nicht garantiert partitioniert wird!

Bem.:

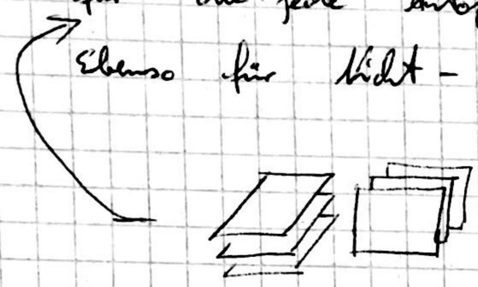
1) Dieser Algo funktioniert genauso in 3D, ⇒ expected size $= O(n^2)$ [Blanc-gelb Buch Bud edition]

~~aber man weiß nicht, wie man ihn analysieren soll!~~

~~(→ Varianten analysieren)~~

2) Es gibt Mengen von nicht-schneidenden Dreiecken in 3D, für die jede Autopartition $\Omega(n^2)$ Größe hat.

Ebenso für Nicht-Autopartitionen [s. Buch]



Proof by induction over sizes of the stacks

3) In der Praxis hat man $O(n)$ bis $O(n \log n)$ (meist Ωn)

In der Praxis gilt "principle of locality": Polygone sind klein verglichen mit Gesamtgröße

["Binary Space Partitioning (BSP) Trees"]

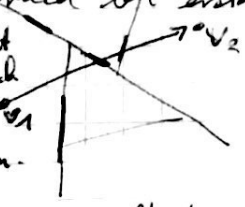
Kleine Anwendungen von BSP-Trees:

I. Ray-Shooting:

1. Blatt finden
2. Backtracking der Rekursion

[BSP FAQ]

da das Kind zuerst, das näher am Startpkt DFS nicht in-order; Abbruch bei ersten Hit



- Pgone muß man nicht splitten: einfach mehrfach einfügen.
- Pgone muß man im BSP nur nicht speichern.

II. Rendering ohne Z-Buffer:

Painter's Algo (= back-to-front):

rendere erst den BSP-Teilbaum, der Viewpoint nicht enthält, dann Polygon(e) in der Schnittebene, dann BSP-T.baum, der Vpoint enthält.

Studis den Algo in der VL entwickeln lassen anhand (T, v_1, v_2): teste ob v_1, v_2 auf versch. Seiten von h sind

Problem: viele Pixel werden 17x überschrieben → überflüssig gemacht; Ziel: jedes Pixel ≤ 1x schreiben.

Besser: front-to-back + BSP für Screen

3D-BSP umgekehrt wie bei Painter's Algo traversieren; Basis vor dem Rendern durch den Screen-BSP [BSP FAQ] "sickern" lassen (wird dabei in kleinere konvexe Teile gesplittet); nur die Teile rendern, die in "freie" Zellen kommen; an die entspr. Blätter neue BSP-Teilbäume dranhängen

Weitere Verbesserung: View-Vektor einbeziehen

Damit BSP-Teilbäume komplett culln, wenn "hinter" Viewpoint.

(Doom verwendete anscheinend BSP's zum Rendern

III. Point Classification: "in" oder "out"

(nasty detail: Pkt liegt auf Splitting-Ebene)

evtl zahlreiche Obj auf nächste Seite machen

["Binary Space Part. (BSP) Trees"]

[Graphics Gems IV, III.5]

View-Frustum Culling: Ecken des Frustums gegen Ebene des Knotens checken

Back-Face Culling: zeichne Pgon nicht, wenn Viewpunkt auf neg. Seite der Ebene. Keine extra Kosten; trotzdem ganzen BSP trav

Objekt – Repräsentation für geschlossene Obje:

Bane BSP für Menge der Pgone (Autopartition)

Für jedes Blatt:

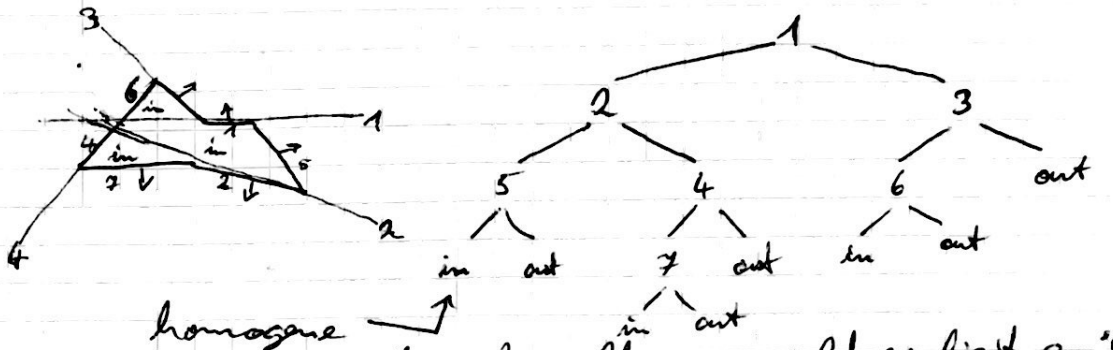
rechtes Kind (leer) ~~zeigt nicht~~ auf Seite der Normale = "out" Region

linkes Kind = "in" Region

Annahme dabei: Normalen der Pgone zeigen nach außen

→ vor Übung

Bsp.:



homogene Regionen/Blätter: braucht man nicht explizit speed

Optimale BSP-Trees: Was ist ein "guter" Tree?

Balancierung vs. Splits, hängt von App ab:

App = Klassifikation (Okt, Linie, Ray-Tracing) → Balancierung optimieren

App = Visibility (Rendering) → Depth Sorting → # fragments Splits optimieren

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

mit P^-, P^+ = Wahrscheinlichkeit, daß linker/rechter Tree traversiert wird. [Naylor: "Tutorial on..."]

Bsp. für Point-Location: $P^- = \frac{Vol(R^-)}{Vol(R)}$

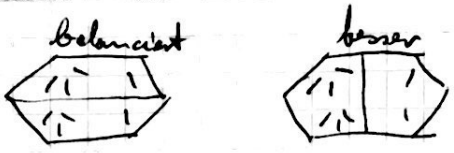
Heuristik: for construction ["Tutorial on..."]

schätze $C(T^-)$ durch $|S^-|^\alpha$, $\alpha = 0.8 \dots 0.95$

addiere zusätzlich $\beta \cdot n$, n = # gesplittete Pgone, $\beta = \frac{1}{4}, 1, \frac{3}{4}$ # Pgons split by h of rest of T

besonders große Pgone: sortiere Pgone nach Größe und betrachte nur die ersten k Stück (Rationale: wenn die großen weg sind, verursachen die kleinen weniger Splits) → werden später weniger gesplittet

Heuristik führt anscheinend zu eher "besseren" BSPs:



[Encho et al.]
["BSP Trees"]

Heuristik für Splitting-Minimierung:

wähle k zufällige Pgone aus S;

wähle davon dasjenige, das die geringste Anzahl Splits verursacht

Beob.: k=5 genügt für fast-optimale Trees.

~~Self~~ Heuristik: Distribution-Optimized BSP's | gleich: Self-Organizing BSP's

für bekannte Anfrageverteilungen:

[Ar, Chazelle, Tal: Self-Customized BSP Trees ... EG 2000]

Bsp.: Strahltest, Ray-Casting

rechte Seite

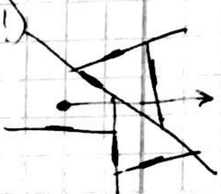
① Idee: ~~Strahlen sind~~ nicht gleichförmig verteilt → BSP anpassen

(manche Richtungen kommen häufiger vor)

Kosten (Anfrage BSP) = # besuchter Knoten ← (very rough!)

$C(T) =$ ≤ Tiefe (BSP) · # stabbed (leaf) cells

+?



Idee: minimiere # stabbed cells bevor Pgon getroffen

Was beeinflußt Wahrscheinlichkeit, daß Strahl Pgon trifft:

- Winkel zw. Strahl u. Pgon = 90° (Normale = 0°) → W.keit groß
- Pgon ~~in Ebene~~ groß (rel. zu Ges.größe) → W.keit groß
- ...

$\omega(l)$ = Dichteverteilung der Strahlen l über Domain D

$\omega: D \to \mathbb{R}$ probability density fct over D = set of all rays

wird gemessen, oder aus Geometrie abgeleitet

$l \in D$ = Strahl, mit Aufpunkt & Richtung, also $D \subseteq \mathbb{R}^5$

S = Menge Pgone, für die BSP gebaut wird.

Algo: randomized greedy

p = Polygon

Define score $(p) = \int_D W(S, p, l) \cdot \omega(l)\, dl$

↑ weight

$W(S, p, l) = |n \cdot l| \cdot \frac{area(p)}{area(S)}$, n = Normale von p

→ Pgone, die sehr wahrschl. getroffen werden, landen oben im Baum. ("customized BSP")

[Analogie: Huffman-Codierung]

Modified Algo: for initial BSP sortiere Pgone ~~für~~ S bzgl. score(p)

wähle zufällig ein p aus den "top k".

Exp.: customized BSP hat 2x Knoten wie Standard-BSP, aber 2-10x weniger besuchte Knoten!

[Fr. Montag, Tal: "...", EG 2402]

Erweiterung:
Deferred, Self-Organizing BSPs:

Problem: ω is at unknown at time of construction
bei Bearbeitung von Anfragen (z.B. Strahltest)
ist Verteilung der Anfragen i.A. nicht uniform,
das wird aber bei Konstr. der BSPs nicht berücksichtigt

Lsg → self-organizing.

Problem: Verteilung der Anfragen ist a priori nicht bekannt

Lsg → deferred

Bsp.: Ray-Kollision (Casting) ①; Ziel: Pgone, die häufig von Strahl geschnitten werden, sollen mögl. weit oben im BSP stehen.

Algo / Datenstruktur:

Knoten v speichert:

Standard:
1. innere Ebene π_v + Pgon P_v
2. evtl. seine Region R_v

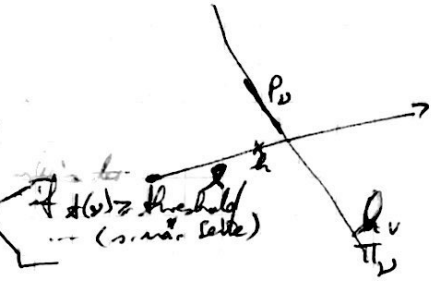
Aufgabe: teste, ob Strahl ggf. Menge von Pgonen trifft (anders bei Ray-Shooting, wo nächster Schn.-pkt. gesucht wird!)

Deferred BSP's: preliminary leaves
Erweiterung → 3. falls vorläufiges Blatt: Liste von Pgonen $\mathcal{L}(v)$
$t(v)$ = visit counter
Also, $t(p)$ = last counter for $p \in S$

testray(l, v):

l := Strahl
v := Knoten

if v ist Blatt
increment $t(v)$
teste l gegen alle Pgone $P \in \mathcal{L}_v$ mit $t(p)$ < last t's
return sobald minim. Schn.-pkt, oder "none"



else

(bei Ray-Shooting steht hier "miss"!)
v_1 := Kind von v, das Startpkt von l enthält
on same side as start pt of l
v_2 := "anderes" Kind

testray(l, v_1)

if kein Hit gefunden in v_1

testray(l, v_2)

endif

(Abbruch früher i.A. falls Strahl endlich; z.B. Walk-through

Inkrementeller BSP := 1 Knoten (= Wurzel) mit $\mathcal{L}$ = alle Pgone des Objes/Szene

Fragen:

1. wann wird ein vorläufiges Blatt gesplittet? (und welches?)
2. wie wird es gesplittet?

1. Wann: $t(v) =$ visit traversal counter for v

Zugriffszähler pro Knoten; wird erhöht, wenn traversiert wird;

Split, falls $\# \geq$ Schwellwert (absolut oder relativ)

2. Wie: $t(p) =$

Zugriffszähler pro Pgon aus $\mathcal{L}_v$;

wird erhöht, wenn Schnittpkt damit gefunden;

~~sortiere $\mathcal{L}_v$ gemäß diesem Zähler (inkrementell); self-organizing list~~

falls Split, dann mit dem ersten Pgon p^0 aus $\mathcal{L}_v$,
für das $t(p^0) = \max$

Bem.:

- Viele Pgone, die wie an einem Schnitt beteiligt sind, wandern schließlich in Blätter, die nie traversiert werden.
- Andere Arten, die Listen $\mathcal{L}$ zu verwalten (z.B. "move to front" (ähnlich zu cache org. strategies!) oder "swap"), scheinen weniger effiziente BSPs zu liefern
- Performance-Gewinn gegenüber Standard-BSP: Faktor 2–20 (angeblich) besonders bei Objen mit vielen Konkavitäten

Frage:

Wie macht man es für Ray-Tracing?

(dort nicht nur min. Schn. pkt!)

[Obj. repräsentation ... nur 2 Seiten]

[Naylor, Amanatides, Thibault : Merging ... bsp Operations; Siggraph 1990 - Siggraph 96 Course Notes 29]

Merging :

BSP - Obj. repräsentation ; eignet sich gut für Solid Modeling (CSG) (Boolean ops), da $\cap, \cup, \setminus$ leicht zu implementieren (mit einheitlichen Algos).

Set Operation a.k.a. CSG Oper.

Abstrakte Fundamentale Operation : "Merge"

$$BSP_1 \oplus BSP_2 \to BSP_3$$

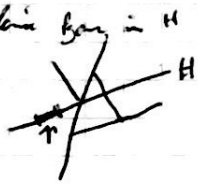
wobei die Zellen von BSP_3 durch paarweise Schnitte der Zellen aus BSP_1 u. BSP_2 entstehen

$$C_3 = \{ c_1 \cap c_2 \mid c_1 \in C_1 , c_2 \in C_2 , c_1 \cap c_2 \neq \emptyset \}$$

$c_i \hat{=}$ Regionen der Blätter , $C_i = \{$ alle Regionen aller Blätter des $BSP_i \}$

Erst mal einfachere Operation :

Geg.: BSP T , Ebene H ; (evtl. noch Polygon p_H)



ges.: neuer BSP $\hat{T}$, dessen Wurzel H ist.

Bem.: $\hat{T}^- = T \cap H^-$, $\hat{T}^+ = T \cap H^+$.

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

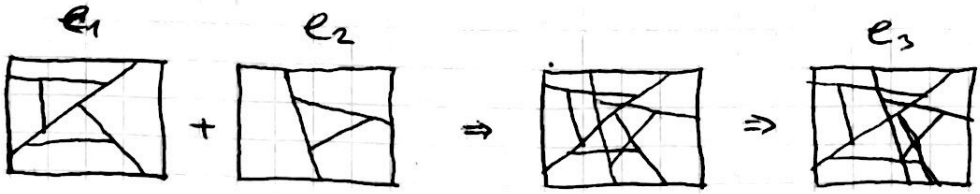
$(T^\ominus, T^\oplus) :=$ splittree p-tree (T, H, H)

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

Notation : $T = ($ Ebene p, linkes Kind, rechtes Kind $)$

Polygon in Split-Ebene

Bem.: Für das eigtl. Ziel "tree merging" braucht man eigtl. nur die Fkt. p-tree , aber der Klarheit halber tun wir so, als hätten wir $\hat{T}$ konstr.



splitting
p-tree $(T, H, P) \to (T^\ominus, T^\oplus)$ — acht man später, wenn gebraucht

$\{ T^\ominus = T \cap H^-,\ T^\oplus = T \cap H^+,\ P = H \cap R(T),\ T = (H_T, \overset{p_T}{\downarrow}, T^-, T^+) \}$

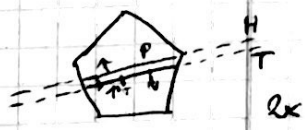
Fall T ist Blatt:

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

(*)

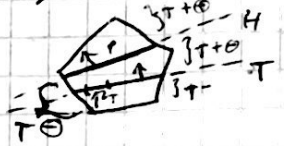
{ T kein Blatt }

Fall H und H_T koplanar, opposite normals:



$T^\ominus := T^+$; $T^\oplus := T^-$ (***)

Analog: koplanar und Normalen gleich



Fall "pos./pos.":

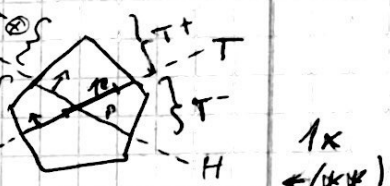
$(T^{+\ominus}, T^{+\oplus}) := \overset{\text{splittree}}{p\text{-tree}}(T^+, H, P)$

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

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

Analog die Fälle "neg/neg", "pos/neg", "neg/pos"

Fall "mixed":



←(**)

$(T^{+\oplus}, T^{+\oplus}) := \overset{\text{splittree}}{p\text{-tree}}(T^+, H, P \cap R(T^+))$

$(T^{-\oplus}, T^{-\oplus}) := p\text{-tree}(T^-, H, P \cap R(T^-))$ ← Region des Kindes T^+

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

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

analog die restlichen 4 Fälle ("parallel an", "pos/neg", "neg/pos", "neg/neg")

Bem.:

(*) kommt vielleicht einem komisch vor, wird aber klar wenn man ein Bsp. rechnet.

(**) $P \cap R(T^+)$ bekommt man natürlich ganz einfach, indem man $P \cap H_T^+$ bildet (genau 2 Kanten von P schneiden).

(!) Numerisch muß man aufpassen, daß man nicht aus Versehen einen der anderen Fälle als "parallel an" klassifiziert! Kann leicht passieren, wenn Zellen klein werden.

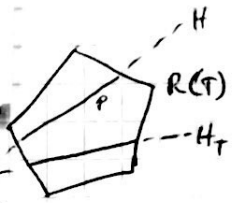
(***) Das P_{gen} ^{p_T} muß man eigtl. ganz nach oben transportieren und in neuen Kindknoten einfügen.

- splitting p-tree tut fast keine Arbeit; nur Fälle klassifizieren (s.u.) und $p \cap H^+$, $p \cap H^-$ berechnen.

"mixed" Fall

s.a. Graphics Gems III, Kap. V.2

Fall-Erkennung:



vergleiche $P = H \cap R(T)$ mit H_T.

brute-force: alle Ekte von P in H_T einsetzen

Beob.: P konvex → Min u. Max mit Binärsuche finden.

Dem.: $P \cap R(T^-)$ u. $P \cap R(T^+)$ fällt hier als Nebenprodukt ab.

Damit kann man also für Tree-Merging machen.

merge $(T_1, T_2) \to T_3$ {precond.: $R(T_1) = R(T_2)$}

T_1 oder T_2 ist Blatt (= elem. konvexe Zelle in der Space-Partitioning) (= homogene Region / Blatt).

return → cell-op (T_1, T_2)

Sonst:

T_1 = BSP mit Wurzel (H_1, T_1^-, T_1^+)

$H_1 \cap R(T_2) = H_1 \cap R(T_1)$

$(T_2^\ominus, T_2^\oplus) := \underset{\text{splitting}}{p\text{-tree}}(T_2, H_1, ..)$

$T_2^\ominus, T_2^\oplus$ beliefert jetzt exakt die gleiche Region als kamen wie T_1^-, T_1^+.

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

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

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

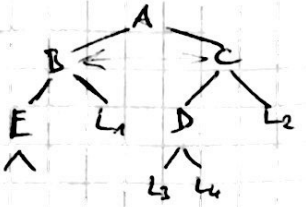
wg. precondition ↑ darf man jetzt die Teile einfach wieder zus. bauen

FRAGE: was macht hier hier? mit H_1 wieder zus. bauen → p_1 (OK, wegen $R(T_3^-) = R(T_1^-)$, $R(T_3^+) = R(T_1^+)$)

Bem.:

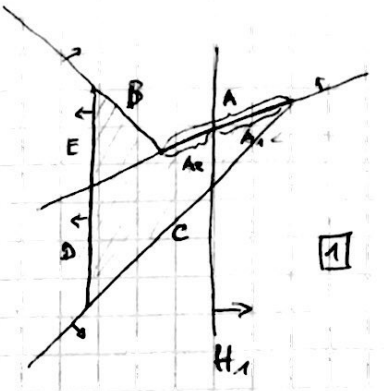
- Ekt cell-op liefert die "Semantik" $(\cup, \cap, -)$; für den ursprgl. gesuchten Zellkomplex (BSP$_2$) ersetzt man einfach das eine Blatt^{T_1} durch den anderen BSP-T. baum T_2.
- Symmetrisch: ob man T_2 mit H_1 partitioniert oder T_1 mit H_2 ist total egal, es kommt dasselbe heraus. (Man muß nur aufpassen, falls cell-op nicht-kommutativ)

Bsp. für p-tree:



Rek. stufe

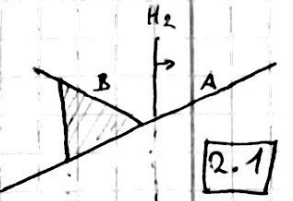
1. (A, H_1) = "mixed":



kommt später im Text

1

2.1 (B, H_2) = "pos/pos":

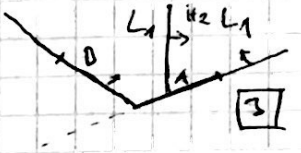


2.1

3. (L_1, H_2) = "leaf":

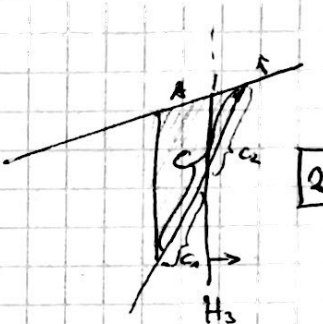
$T^{\oplus} := L_1$, $T^{\ominus} := L_1$

$(T^{+\oplus}, T^{+\ominus})\begin{cases} T^{\oplus} := L_1 \\ T^{\ominus} := (B, E, L_1) \end{cases}$



3

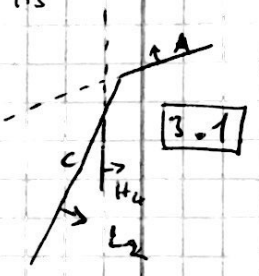
2.2 (C, H_3) = "mixed":



2.2

3.1 (L_2, H_4) = "leaf":

$(T^{+\oplus}, T^{+\ominus})\{T^{\oplus} := L_2$, $T^{\ominus} := L_2$



3.1

3.2 (D, H_5) = "neg / neg":

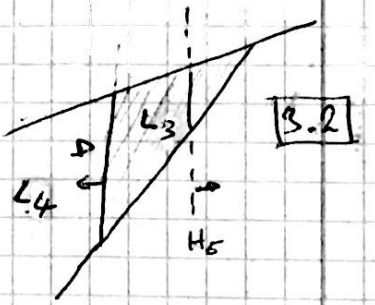
4. (L_3, H_5) = "leaf":

$T^{\oplus} := L_3$, $T^{\ominus} := L_3$

$(T^{-\oplus}, T^{-\ominus})\begin{cases} T^{\oplus} := L_3 \\ T^{\ominus} := (D, L_3, L_4) \end{cases}$

↖ einig mit rechtem Kind von D

$(T^{\oplus}, T^{\oplus})\begin{cases} T^{\oplus} := (C_2, L_3, L_2) \\ T^{\ominus} := (C_1, D, L_2) \end{cases}$

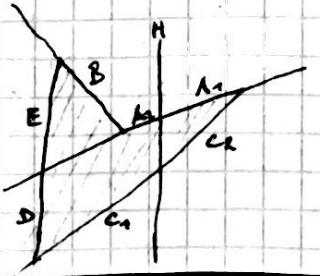


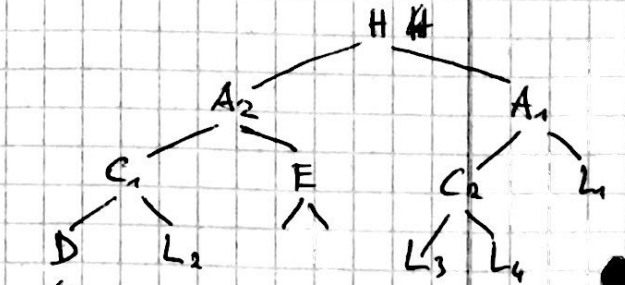
3.2

$T^{\oplus} := (A_1, C_2, L_1)$

$T^{\ominus} := (A_2, C_1, E)$

(H, A_2, A_1)

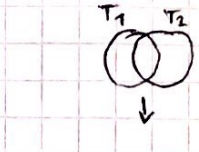




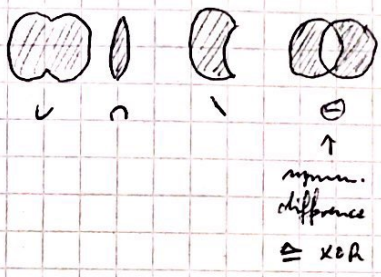
einige Li's mehrfach verwendet, aber einfachheit halber.

Precondition: $R(T_1) = R(T_2)$

call-op (T_1, T_2): { oBdA T_1 Blatt (T_2 evtl. auch) }



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	(T_2 komplementiert)
	out	T_1	(= out)
$\ominus$ KOR	in	T_2^c	
	out	T_2	



Die Zellen-Op. liefert evt die rigtl. Semantik!

Kondensation:

Diese Vereinfachungsop. sollte man ständig während eines Merge durchführen; bringt Perf.!

Wie bestimmt man die Pgone auf dem Rand von Obj1 ⟨op⟩ Obj2 ?