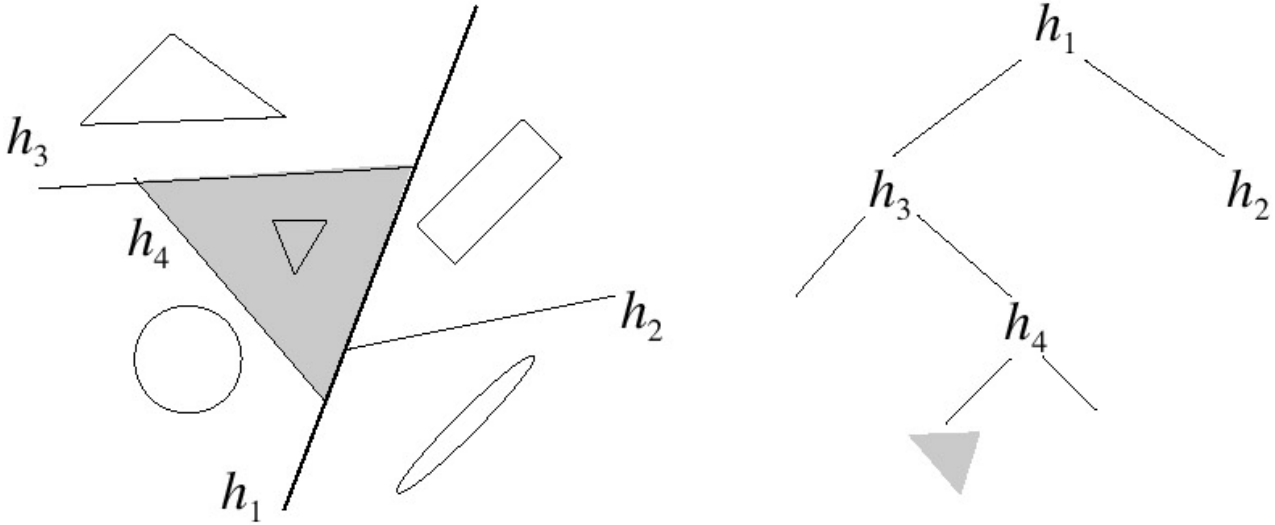
Diskrétne Geometrické Štruktúry

3. BSP

Martin Samuelčík
samuelcik@sccg.sk, www.sccg.sk/~samuelcik, I4

BSP stromy

- Binary Space Partitioning
- Zovšeobecnenie kd-stromov, rozdeľovanie priestoru ľubovoľnými nadrovinami
- Umožňuje usporiadanie objektov
- Doom, Quake, Half-life...



BSP strom

- Nech S je množina objektov (body, polygóny,...)
- $S(v)$ je množina objektov pre vrchol v BSP stromu
- BSP $T(S)$ je definovaný:
  - Ak $|S| <= 1$, tak T je list ktorý obsahuje S
  - Ak $|S| > 1$, tak v je koreň T a v obsahuje deliacu nadrovinu h_v, množinu $S(v)=\{x \in S, x \in h_v\}$ a dvoch potomkov (podstromy) pre množiny

$$S^- := \{x \cap h_v^- | x \in S\}$$

$$S^+ := \{x \cap h_v^+ | x \in S\}$$

Vytvorenie BSP stromu

```
struct HyperPlane
{
        vector<float> coefficients;
}
```

```
struct BSPTreeNode
{
        List polygons;
        HyperPlane partition;
        BSPTreeNode* front;
        BSPTreeNode* back;
}
```

```
struct BSPTree
{
        BSPTreeNode* root;
}
```

```
BSPTree* BuildBSPTree(List polygons)
{
        result = new BSPTree;
        result->root = BuildBSPTreeNode(polygons);
        return result;
}
```

```
BSPTreeNode*  BuildBSPTreeNode (list polygons)
{
        if (polygons.IsEmpty ()) return NULL;
        BSPTreeNode* tree = new BSPTreeNode;
        polygon* root = polygons.GetFromList ();
        tree->partition = root->GetHyperPlane ();
        tree->polygons.AddToList (root);
        list front_list, back_list;polygon* poly;
        while ((poly = polygons.GetFromList ()) != 0)
        {
                int result = tree->partition.ClassifyPolygon (poly);
                switch (result)
                {
                        case COINCIDENT:
                          tree->polygons.AddToList (poly);
                          break;
                        case IN_BACK_OF:
                          back_list.AddToList (poly);
                          break;
                        case IN_FRONT_OF:
                          front_list.AddToList (poly);
                          break;
                        case SPANNING:
                          polygon   *front_piece, *back_piece;
                          SplitPolygon (poly, tree->partition, front_piece, back_piece);
                          back_list.AddToList (back_piece);
                          front_list.AddToList (front_piece);
                          break;
                }
        }
        tree->front = BuildBSPTreeNode (front_list);
        tree->back = BuildBSPTreeNode (back_list);
}
```

Nadroviny

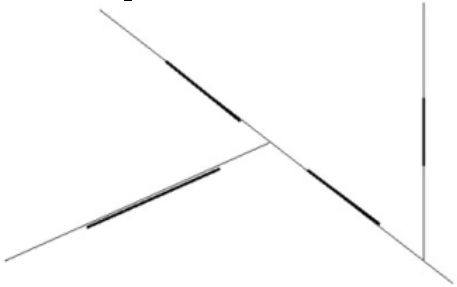
- Priamka, rovina...
- Implicitné vyjadrenie pre d-rozmerný priestor: $a_1.x_1+a_2.x_2+...a_d.x_d+a_{d+1}=0$
- $(a_1,a_2,...a_d)$ – normála, určuje aj smer pre vnútro a vonkajšok
- Test bodu – znamienko po dosadení bodu do predpisu
- Test polygónu – porovnanie znamienok po dosadení vrcholov polygónu
- Rozdeľovanie polygónov – hľadanie prieniku hraničných úsečiek s nadrovinou

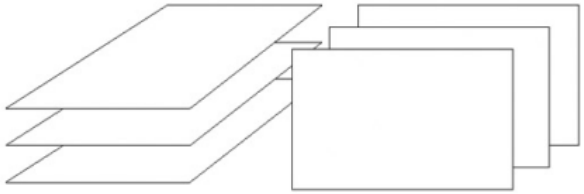
Rozdeľovacie techniky

- Auto-rozdeľovanie – $O(n^2)$
- Ľubovoľné rozdeľovacie nadroviny, čas sa dá určiť rekurzívne: $T(n) = n + 2T(\frac{n}{2} + \alpha n) \in O(n^{1+\delta})\ ,\ 0 < \alpha < \frac{n}{2},$
  - α = percento polygónov rozdelených vo vrchole

α	0.05	0.2	0.4
	$n^{1.15}$	n^2	n^7

- Pre každý polygón vyber vrchol-reprezentanta (ťažisko, stred BB, ...) a urči nadrovinu, ktorá rozdeľuje body tohoto polygónu približne na polovicu





Ohodnocovacie heuristiky

- Ohodnocovanie kvality rozdelenia
- Cena stromu $C(T) = 1 + P(T^-)C(T^-) + P(T^+)C(T^+),$
  - C – ohodnotenie, P – pravdepodobnosť navštívenia podstromu
  - Napríklad pre polohu bodu (či je dnu alebo mimo telesa) $P(T^-) = Vol(T^-)/Vol(T)$, pre raytracing to môže byť povrch bunky
- Lokálna heuristika $C(T) = 1 + |S^-|^\alpha + |S^+|^\alpha + \beta s,$
  - S – počet polygónov, s – počet rozdelených polygónov

Automatické rozdeľovanie

- Celkom efektívne - usporiadanie podľa veľkosti polygónov
  - veľké polygóny majú väčšiu pravdepodobnosť, že budú rozdelené, tak sa ich zbavme, čo najskôr
  - pre prvých k najväčších polygónov vyrátaj funkciu C(T) a vyber polygón s najmenšou cenou
- Poprípade náhodne vyber k polygónov
  - vyber ten, ktorý vyrobí najmenej rozdelení polygónov
- Používané hodnoty
  - $\alpha = 0.8, \ldots, 0.95$
  - $\beta = 1/4, \ldots, 3/4$
  - $k = 5$

Sledovanie lúča

- Organizácia BSP na základe požiadavky – napr. lúč vychádza z jedného bodu
- Ohodnotenie požiadaviek

$$C(\text{query}) = \#\text{ nodes visited} \leq \text{depth(BSP)} \cdot \#\text{ stabbed leaf cells.}$$

- Chceme preťat čo najmenej listov, t.j. polygóny s väčšou pravdepodobnosťou zásahu dáme v strome vyššie
- Pravdepodobnosť prieniku lúča s polygónom:
  - ak je uhol medzi lúčom a normálou polygónu menší, pravdepodobnosť je väčšia
  - ak je polygón väčší, pravdepodobnosť je väčšia

$$\text{score}(p) = \int_D w(S,p,l)\omega(l)dl, \qquad w(S,p,l) = \sin^2(\mathbf{n}_p, \mathbf{r}_l)\frac{\text{Area}(p)}{\text{Area}(S)},$$

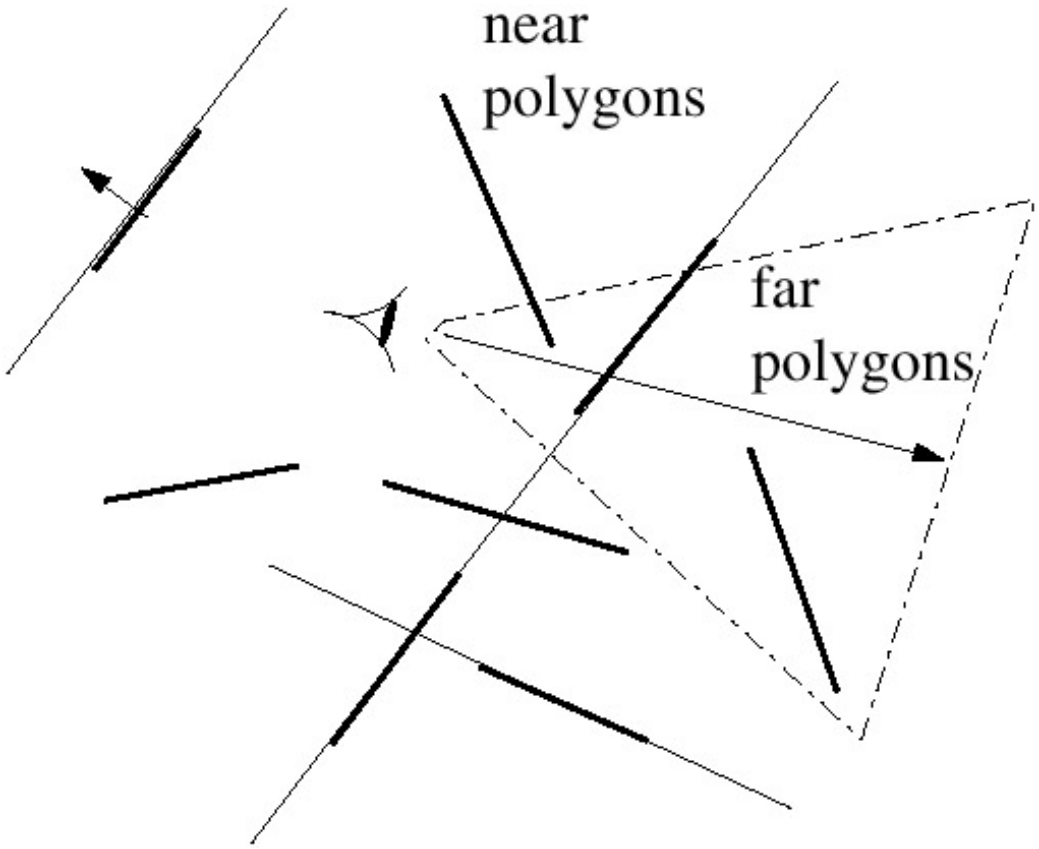
Samoorganizujúce sa BSP

- Keď nepoznáme distribúciu požiadaviek alebo je príliš drahé ju vyrátať
- Vždy máme zostrojenú len nutnú časť stromu
- Najprv vytvoríme iba časť BSP
- V každom vrchole aj informácia o nepoužitých polygónoch
- Uchovávame, koľko krát bol nejaký vrchol navštívený, ak hodnota presiahne nejaký prah, vrchol prerozdelíme
- Počítame aj počet prienikov lúča s polygónom, podľa toho vyberáme deliacu nadrovinu

Určovanie viditeľnosti

- Potrebujeme určiť, ako sa polygóny navzájom prekrývajú
- Maliarov algoritmus – kreslíme odzadu dopredu (najprv prechádzame polroviny v ktorých nie je kamera, potom kreslíme polygóny polroviny a potom prechádzame polroviny, kde sa nachádza kamera)
- BSP – máme rozdelený priestor, deliace nadroviny nám vždy určia čo je vzadu a čo vpredu vzhľadom ku kamere
- Porovnávame s aktuálnou polohou pozorovateľa

Určovanie viditeľnosti



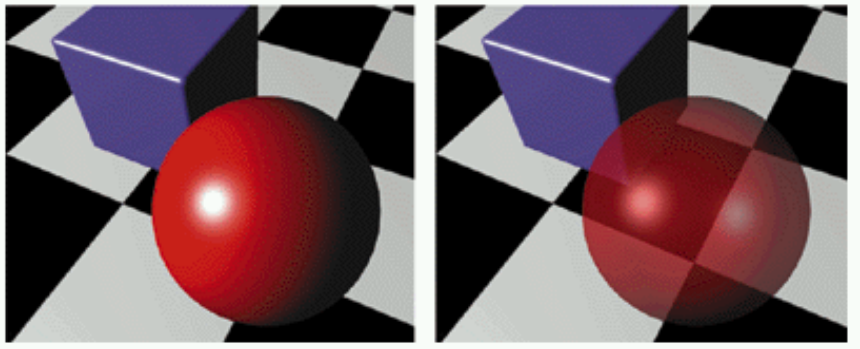

```
void DrawBSPTree (BSP_tree *tree, point eye)
{
        if (tree == NULL) return;
        real result = tree->partition.ClassifyPoint (eye);
        if (result > 0)
        {
                DrawBSPTree(tree->back, eye);
                tree->polygons.DrawPolygons();
                DrawBSPTree(tree->front, eye);
        }
        else if (result < 0)
        {
                DrawBSPTree(tree->front, eye);
                tree->polygons.DrawPolygonList();
                DrawBSPTree (tree->back, eye);
        }
        else
        {
                // the eye point is on the partition plane...
                DrawBSPTree(tree->front, eye);
                DrawBSPTree(tree->back, eye);
        }
}
```

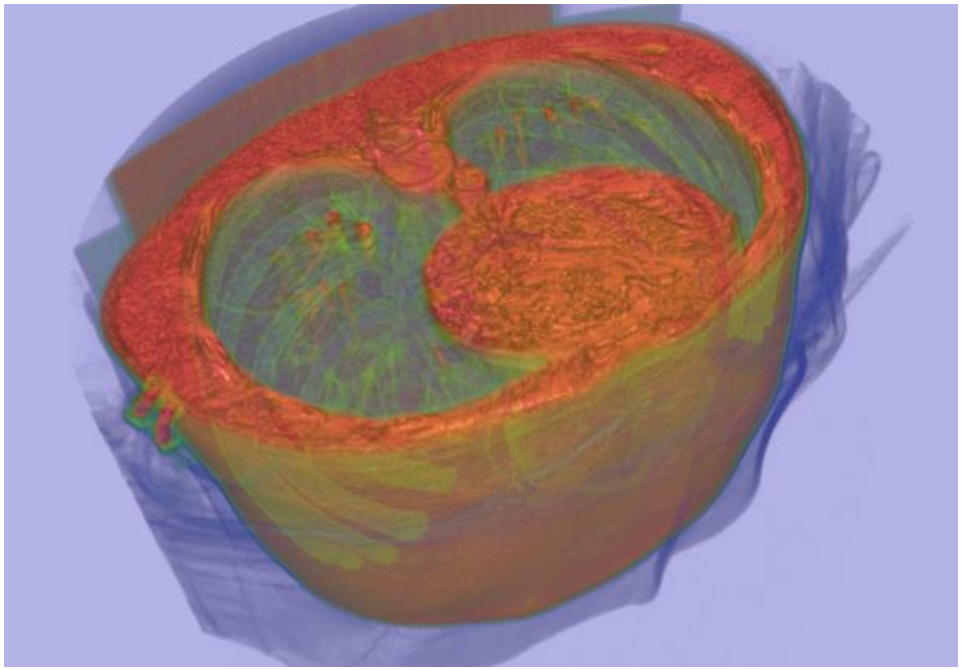
Určovanie viditeľnosti 3

- Kombinácia urýchľovacích algoritmov
- Odstránenie odvrátených polygónov (backface culling)
- Orezávanie na pohľadový objem (frustum culling)
- Prepisovanie pixelov – vykresľovanie od predu dozadu + štruktúra pre určenie častí obrazovky, ktoré sú už prepísané a ktoré nie (používa sa tiež BSP strom)

Priehladnosť

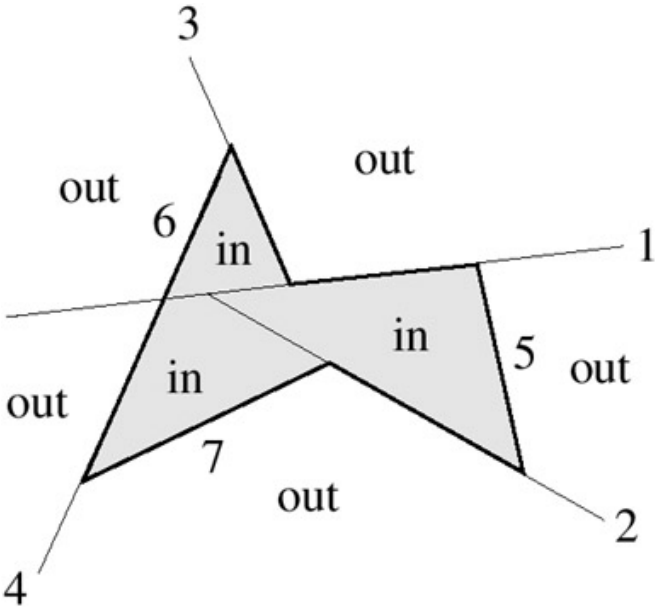
- Použitie zmiešavania (alpha-blending) v 3D
  - Fragmenty polygónu sa zmiešajú s farbou vo framebuffery v nejakom pomere
- Pri priehľadnosti potrebujeme usporiadanie
  - Od predu dozadu
  - Od zadu dopredu
- Aditívny blending

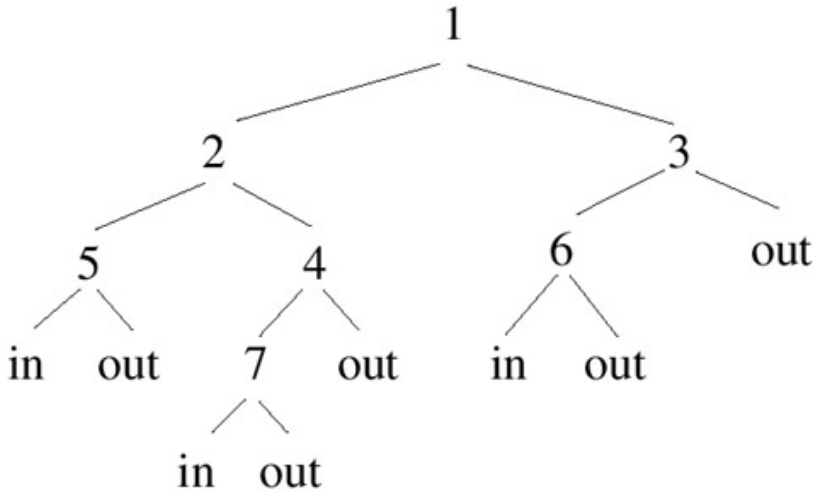




Reprezentácie objektov

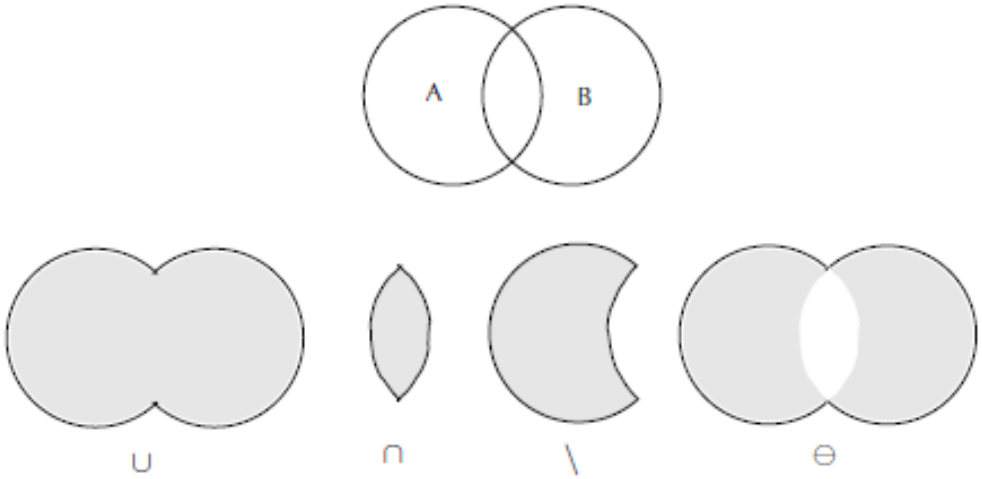
- Uzavreté objekty
- Hranica objektu generuje prerozdeľovacie nadroviny
- Ľahké určenie príslušnosti bodu
- Nevhodné pre hladké povrchy





Množinové operácie

- Zásadné operácie v modelovaní

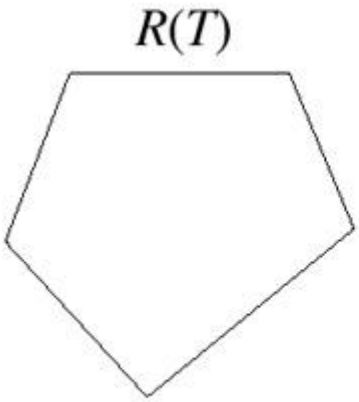


- Pre BSP reprezentácie – spojenie dvoch BSP stromov
- Operácie sa veľmi nelíšia, iba v elementárnych krokoch pre listy

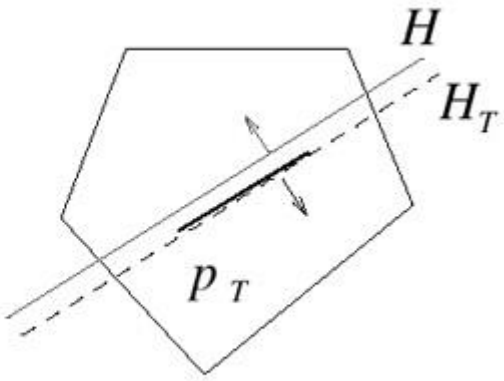
Rozdelenie stromu

- Pre daný BSP strom *T* a nadrovinu *H*, vytvor nový BSP strom T_1, tak,aby $T_1^- = T \cap H^-$ a $T_1^+ = T \cap H^+$
- *H* bude nový koreň
- Vrchol *T* obsahuje (H_T, p_T, T^-, T^+)
  - *H* rozdelovacia nadrovina
  - *p* polygón ležiaci v *H*
- Pri skúmaní *H* vo vrchole *T* máme niekoľko prípadov
- Potreba rátania ohraničujúcich polygónov v bunke

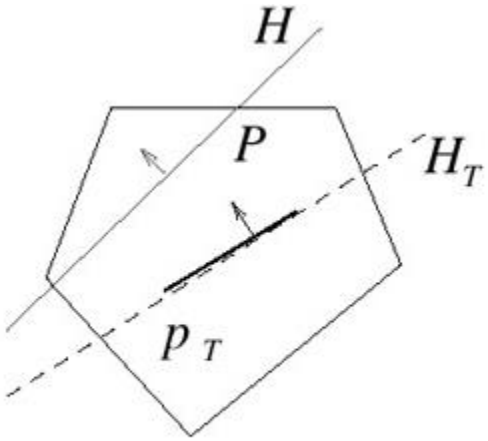
Rozdelenie stromu



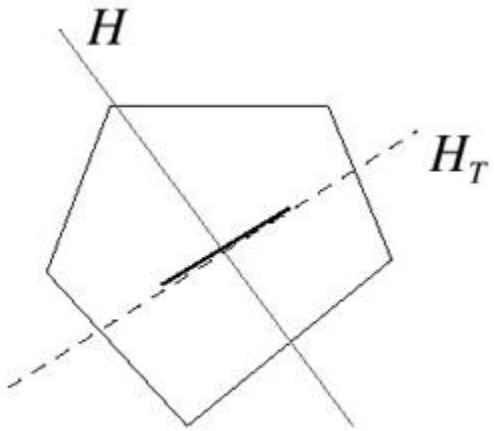
leaf



anti-parallel on



pos./pos.



mixed

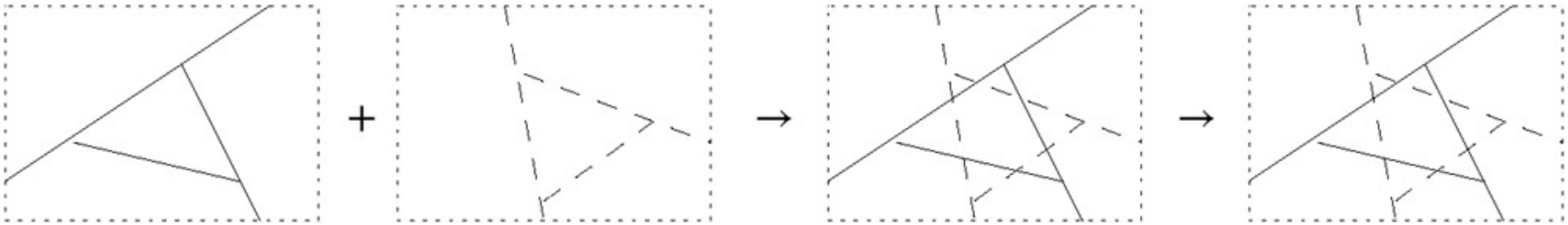
```
split-tree(T, H, P)
{
  //{P = H ∩ R(T)}
  // R(T) – region of the cell of node T (it is convex)
  case T is a leaf :
    return (H, T, T);
  case "anti-parallel" and "on" :
    return (H, T+,T−)
  case "pos./pos." :
    (T+1, T+2) = split-tree(T+, H, P);
    T1 = (HT, pT, T−, T+1);
    T2 = T+2;
    return (H, T1, T2);
  case "mixed" :
    (T+1, T+2) = split-tree(T+, H, P ∩ R(T+));
    (T−1, T−2) = split-tree(T−, H, P ∩ R(T−));
    T1 = (HT, pT ∩ H−, T−1, T+1);
    T2 = (HT, pT ∩ H+, T−2, T+2);
  return (H, T1, T2);
}
```

+ analogické prípady

Spájanie stromov

- Pre dané 2 BSP stromy, spojíme ich do jedného vkladaním nadrovín jedného do druhého
- Ak C_i sú množiny elementárnych buniek i-teho stromu tak spojený strom T_3 má listy

$$C_3 = \{c_1 \cap c_2 | c_1 \in C_1 \;\; c_2 \in C_2, c_1 \cap c_2 \neq \varnothing\}$$

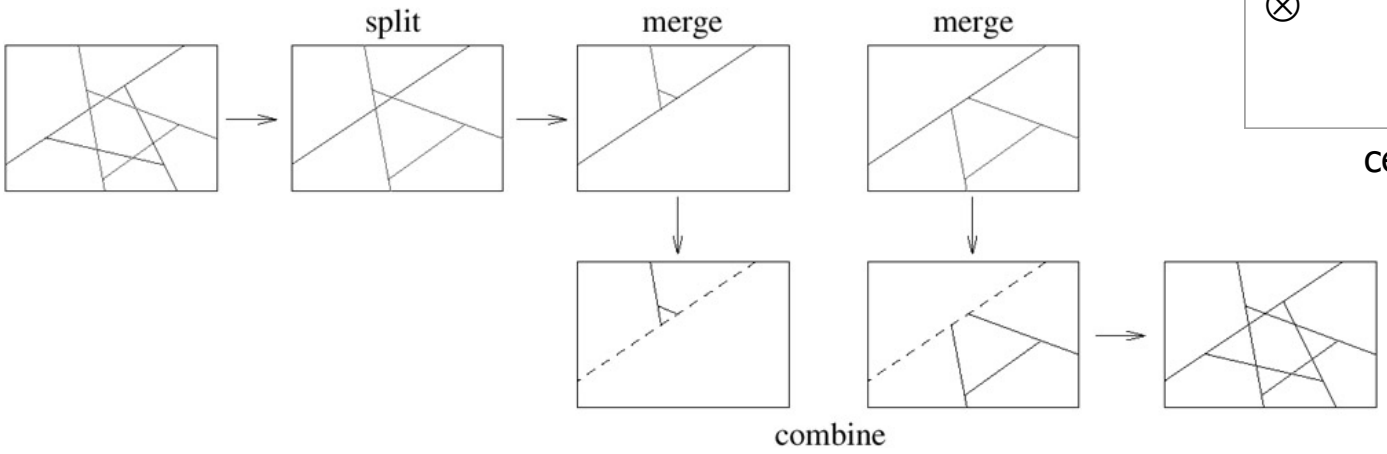


Spájanie stromov

```
merge(T1, T2) → T3
{
        if (T1 or T2 is a leaf)
        {
                perform the cell-op as required by the Boolean
                operation to be constructed
        }
        else
        {
                (T2+, T2-) = split-tree(T2, H1, …);
                T3- = merge (T1-, T2-);
                T3+ = merge (T1+, T2+);
                T3 = (H1, T3-, T3+);
                return T3;
        }
}
```

Operation	T_1	Result
∪	in	T_1
	out	T_2
∩	in	T_2
	out	T_1
\	in	T_2^c
	out	T_1
⊗	in	T_2^c
	out	T_2

cell-op, T_1 is leaf



Detekcia kolízií

- Zistenie v ktorých bunkách sa nachádza
- Ako raytracing
- Výpočet prieniku s nadrovinami medzi bunkami
- Pri pohyboch sa jedná najčastejšie o prienik s úsečkou

Tieňové objemy

- BSP strom z polygónov tieňového telesa
- Tieňové objemy – shadow volumes

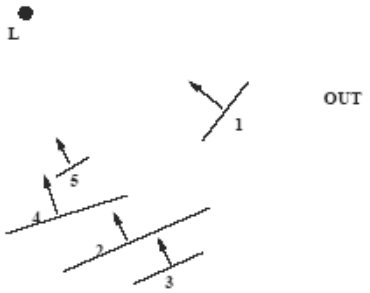


Figure 3: Initial scene

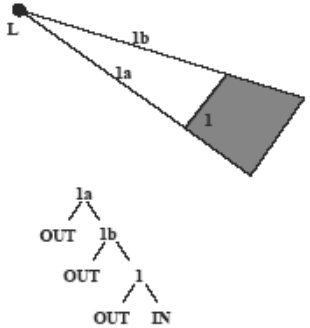


Figure 4: Insert poly 1

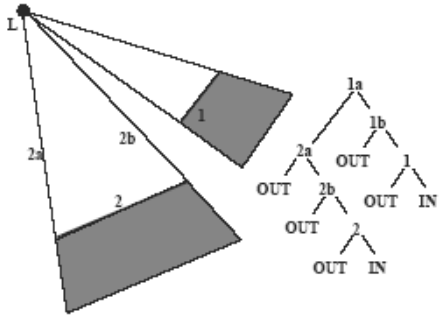


Figure 5: Insert poly 2

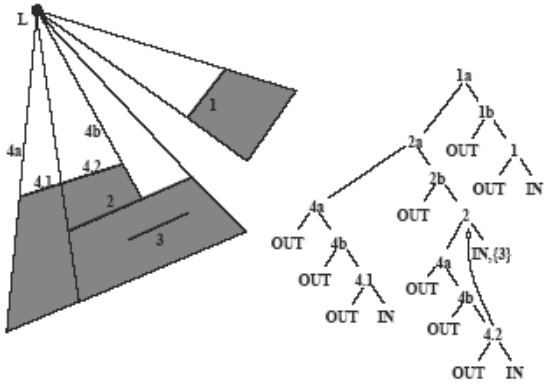


Figure 6: Insert poly 3 and 4

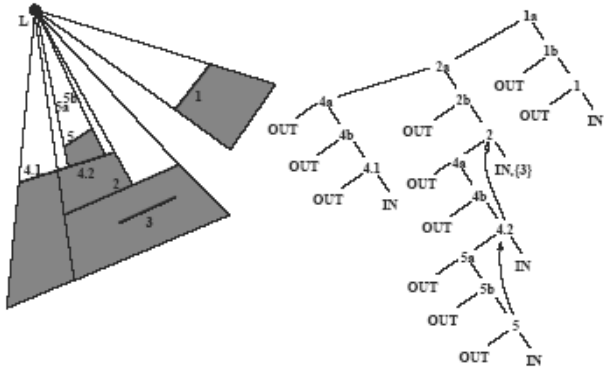


Figure 7: Insert poly 5

Tieňové objemy

- Algoritmus
  - Z pozície svetla nájdi všetky siluetové hrany objektov v scéne
  - Každú siluetovú hranu rozšír do priestoru, vyniknú tým polygóny tieňových telies
  - Ulož steny tieňových telies do BSP stromu
  - Pre nejaký bod scény sa dá zistiť či je v tieni alebo nie podľa toho v ktorom liste sa nachádza
  - Možnosť použiť stencil buffer namiesto BSP

Dynamické scény

- Dynamické objekty sú nanovo vkladané do stromu zo statických objektov pri každom vykreslení
- Často sú dynamické objekty aproximované pomocou bodov (potom sú dynamické objekty vykreslené vždy pred statickými)
- Vloženie jedného bodu je lacné narozdiel od vloženia všetkých polygonóv dynamického objektu
- Ďalšia možnosť je vloženie nadroviny kolmej na smer pohľadu

Otázky?