Diskrétne geometrické štruktúry

5.

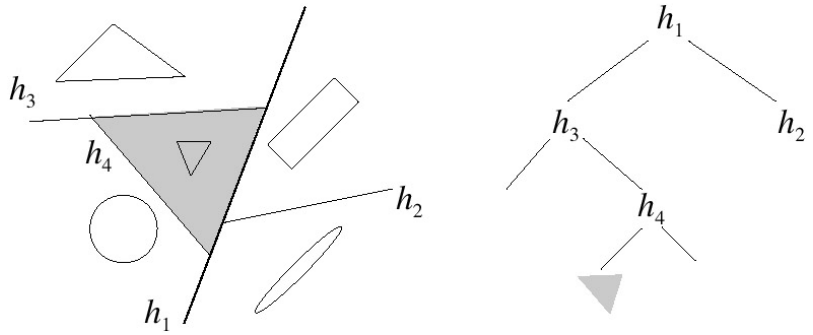
Martin Florek

florek@sccg.sk

www.sccg.sk/~florek

BSP stromy

- binary space partitioning
- zovšeobecnenie kd-stromov
- rozdeľovanie priestoru ľub. nadrovinami
- umožňuje usporiadanie objektov
- hidden surface removal
- Doom, Quake, Half-life...



Definícia BSP

- 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

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

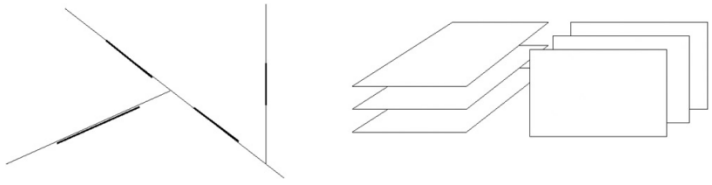
```
BSPTreeNode*  BuildBSPTree (list polygons)
{
        if (polygons.IsEmpty ()) return NULL;
        BSPTreeNode* tree = new BSPTreeNode;
        polygon* root = polygons.GetFromList ();
        tree->partition = root->GetHyperPlane ();
        tree->polygons.AddToList (root);
        list front_list, 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 = BuildBSPTree (front_list);
        tree->back = BuildBSPTree (back_list);
}
```

Nadroviny

- priamka, rovina...
- implicitné vyjadrenie: $a_1.x_1+a_2.x_2+\ldots a_n.x_n+a_{n+1}=0$
- $(a_1,a_2,\ldots a_n)$ – 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
- 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 (ci 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* 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
  - $\alpha = 0.8, \ldots, 0.95$
  - $\beta = 1/4, \ldots, 3/4$
  - $k = 5$

Raytracing

- organizácia BSP na základe požiadavky
- ohodnotenie požiadaviek $C(\text{query}) = \#\text{ nodes visited} \leq \text{depth(BSP)} \cdot \#\text{ stabbed leaf cells.}$
- chceme preťat čo najmenej listov, kým narazíme na polygón
- pravdepodobnosť prieniku lúča s polygónom:
  - ak je uhol medzi lúčom a normálou polygónu menší, pravdepodobnosť je väčšia
  - ak je polygón väčší, pravdepodobnosť je väčšia

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

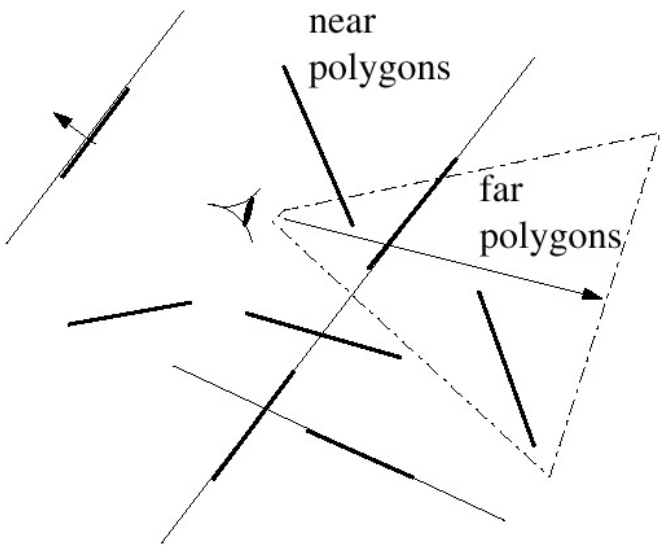
Samoorganizujúce sa BSP

- 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
- porovnávame s aktuálnou polohou pozorovateľa

Určovanie viditeľnosti 2



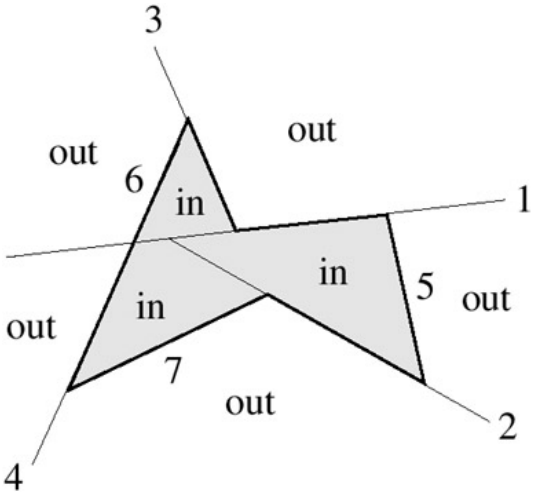

```
void DrawBSPTree (BSP_tree *tree, point eye)
{
        real result = tree->partition.ClassifyPoint (eye);
        if (result > 0)
        {
                DrawBSPTree(tree->back, eye);
                tree->polygons.DrawPolygons();
                DrawBSPTree(tree->front, eye);
        }
        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);
        }
}
```

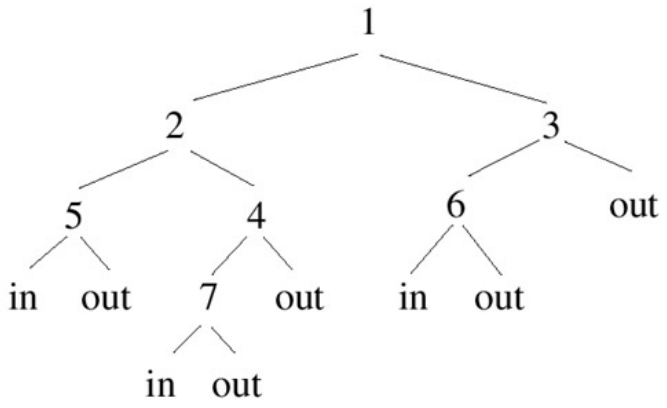
Určovanie viditeľnosti 3

- 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

Reprezentácie objektov

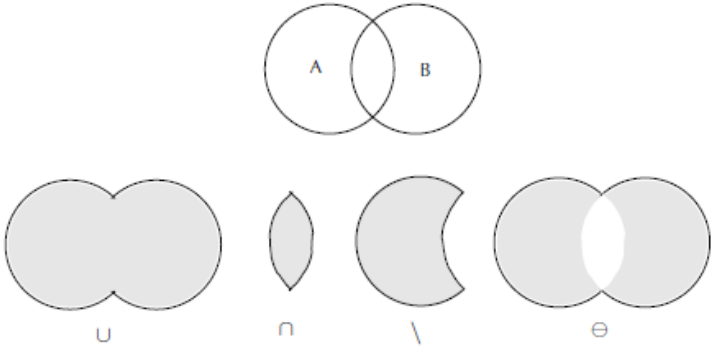
- uzavreté objekty
- hranica objektu generuje prerozdeľovacie nadroviny
- ľahké určenie príslušnosti bodu
- nevhodné pre hladké povrchy





Množinové operácie

- zásadné operácie v modelovaní



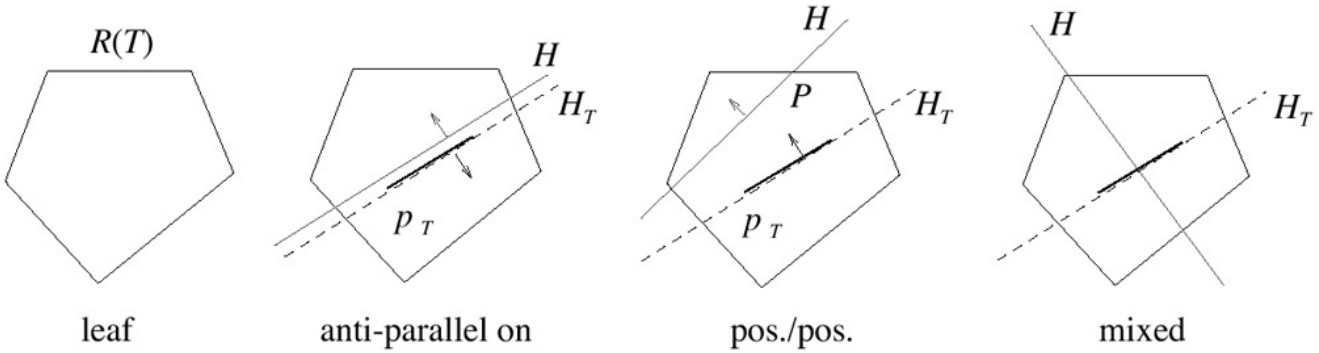
- pre BSP reprezentácie – spojenie dvoch BSP stromov
- operácie sa veľmi nelíšia, iba v elementárnych krokoch pre listy

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 2

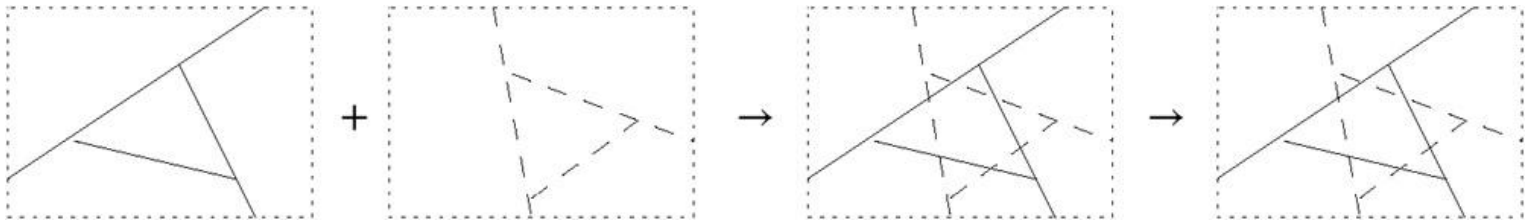
```
split-tree(T, H, P)
{
        //{P = H ∩ R(T)}  // R(T) – region of the cell of node T (it is convex)
        case T is a leaf :
                return (T, T);
        case “anti-parallel” and “on” :
                return (T+,T−)
        case “pos./pos.” :
                (T+1, T+2) = split-tree(T+,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



Spájanie stromov

- pre dané 2 BSP stromy, spojíme ich do jedného vkladaním nadrovím jedného do druhého
- ak C_i sú množiny elementárnych buniek *i*-teho stromu tak spojený strom T_3 má listy

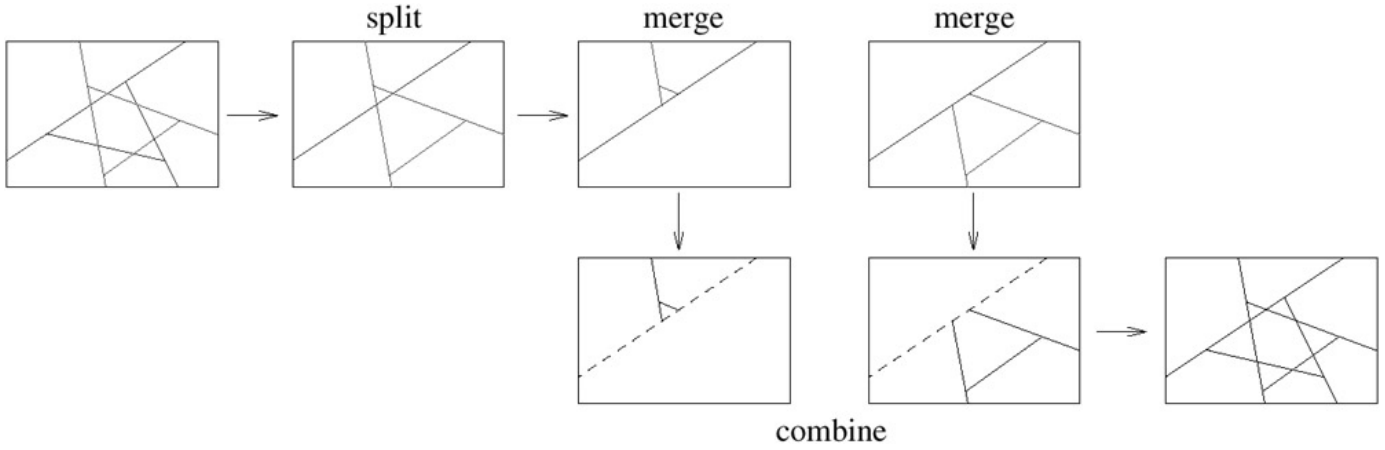
$$C_3 = \{c_1 \cap c_2 | c_1 \in C_1 \;\; c_2 \in C_2, c_1 \cap c_2 \neq \varnothing\}$$



Spájanie stromov 2

```
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



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

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

Tiene

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

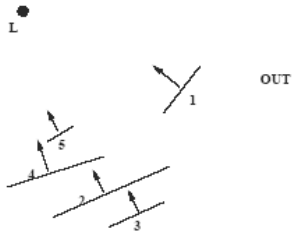


Figure 3: Initial scene

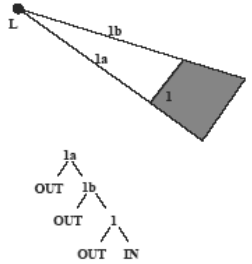


Figure 4: Insert poly 1

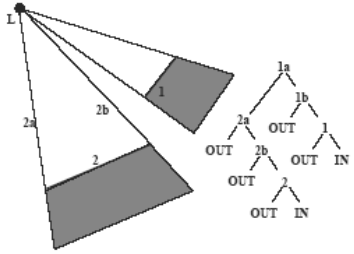


Figure 5: Insert poly 2

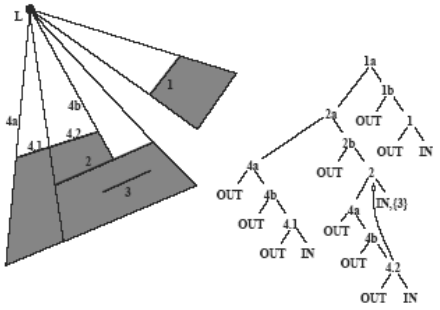


Figure 6: Insert poly 3 and 4

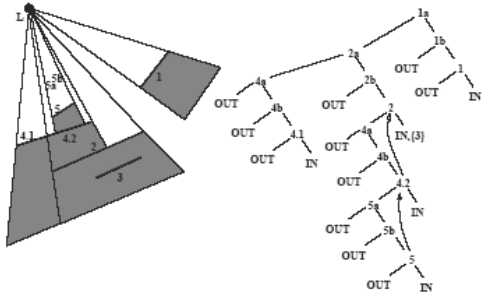


Figure 7: Insert poly 5

Ďalšie použitie

- reprezentácia obrazu
- kompresia obrazu
- radiosity

koniec (-:

florek@sccg.sk