Řešení viditelnosti

KMI/3DG

Mgr. Markéta Trnečková, Ph.D.

Palacký University, Olomouc

Opakování z POGR

- **Ořezávání**
  - Test polohy bodu
  - Ořezávání úsečky – Cohen-Sutherland algoritmus, Cyrus-Beck algoritmus
  - Ořezávání polygonu – Sutherland-Hodgmanův algoritmus
  - Algoritmus Weiler-Atherton

Ořezávání

- Dosud jsme předpokládali, že objekt, který modelujeme/zobrazujeme je celý vidět a je uvnitř scény
- Objekt může blokovat výhled na jiný objekt, může být mimo pohledový objem
- Některé algoritmy pracují až s objekty promítnutými do průmětny – **v prostoru obrazu** (image space)
- Jiné pracují přímo se scénou – **v prostoru objektů** (object space) – zjednodušují scénu ještě před tím, než se objekty promítnou
- Algoritmy jsou svázané vždy s konkrétní reprezentací 3D objektů:
  - Dobře se zpracovávají objekty v **hraniční reprezentaci**
  - V mnoha případech se objekty převádí právě do této reprezentace
- Dělí se dle toho, v jakém tvaru poskytují výsledná data:
  - **Vektorové algoritmy** (liniové)
  - **Rastrové algoritmy**

Ořezávání – Vektorové algoritmy

- Vektorové algoritmy = liniové algoritmy
- **Výstup**: Soubor geometrických prvků (např. úseček) představujících části zobrazovaných objektů
- Úsečky popsané koncovými body lze libovolně zvětšovat a zmenšovat, takže vyhodnocená data lze zobrazit v libovolném rozlišení
- Dalším výstupem je soubor zakrytých částí objektů
- Využívají se zejména v technických aplikacích
- **HLE algoritmy** (Hidden line elimination) – vrací pouze úsečky
- Pokud zpracováváme plochy – **HSE algoritmy** (Hidden surface elimination)

Ořezávání – Rastrové algoritmy

- Pracují v rastru
- **Výstup**: Rastr s pevně danou velikostí
- Projevují se zde zaokrouhlovací chyby
- Oproti vektorovým algoritmům se tyto chyby projevují lokálně (ve vektorových ovlivní celý výsledek)

Ořezávání

- Dále algoritmy dělíme dle toho, jaké mají další paměťové a časové nároky
- Dříve bylo ořezávání důležitější, než je dnes (rychlé počítače, některé algoritmy jsou přímo součástí grafických karet)
- Principy se dají použít i v jiných aplikacích – výpočet stínu
- Dále budeme předpokládat, že pozorovatel (kamera) leží v počátku a směr pohledu je rovnoběžný s osou z (proti jejímu směru) – bez újmy na obecnosti, jednoduchými transformacemi se do tohoto nastavení snadno dostaneme

Ořezávání – základní pojmy

- Už víme, že normálový vektor ploch (které popisují hranici objektu) míří směrem ven
- Podle toho, zda tento vektor míří směrem k pozorovateli nebo od něj dělíme plochy na
  - **Přivrácené plochy** – míří k pozorovateli
  - **Odvrácené plochy** – míří od něj
- Úhel který svírá normála odvrácené plochy se směrem pohledu je vždy ostrý
- **Jak ze dvou vektorů určíme úhel, který svírají?**

Ořezávání – základní pojmy

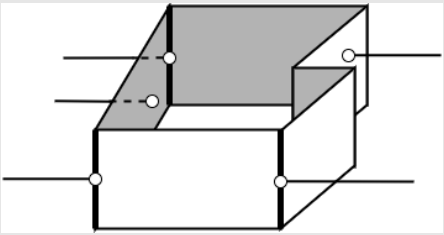
- **Jak ze dvou vektorů určíme úhel, který svírají?**
- Skalárním součinem dvou vektorů získáme cosinus úhlu
- Je-li znaménko skalárního součinu kladné $\rightarrow$ úhel je ostrý (plocha je odvrácená)
- Pokud se pozorovatel dívá proti kladné poloose z – zjistíme polohu pouze se z souřadnice – záporné číslo $=$ odvrácená strana
- Plochy, které jsou odvrácené, jsou skryty přivrácenými plochami – můžeme je odstranit aniž bychom ovlivnili viditelnost ostatních **Backface elimination**

Ořezávání – základní pojmy

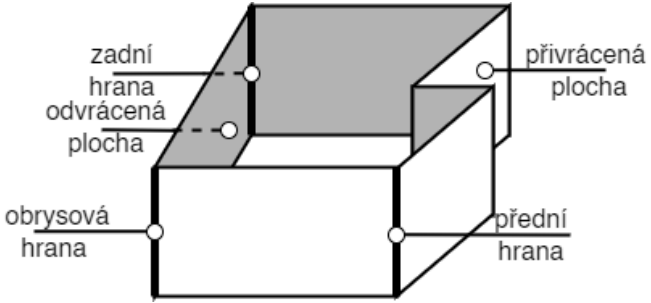
- Hrany objektů dělíme na:
  - **Zadní hrana** – hrana incidující se 2 odvrácenými stranami (není vidět)
  - **Přední hrana** – inciduje se dvěma přivrácenými stěnami
  - **Obrysová hrana** (contour edge) – inciduje s přivrácenou a odvrácenou stěnou
- Pouze přední a obrysové hrany mohou být viditelné
- V liniových algoritmech obrysové hrany indikují změnu viditelnosti

Example

Doplňte popisky do následujícího obrázku.



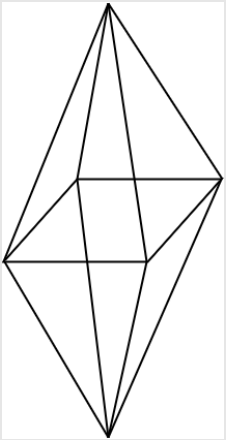
Ořezávání – základní pojmy



Ořezávání – základní pojmy

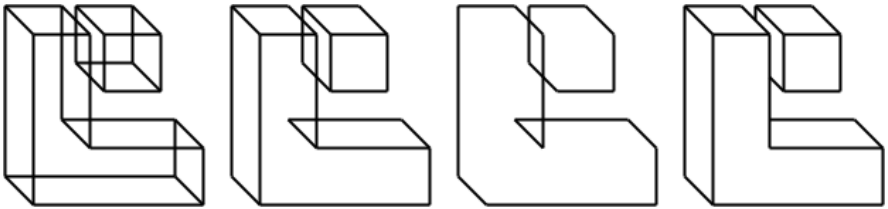
Example

Vyznačte v objektu přední a obrysové hrany.

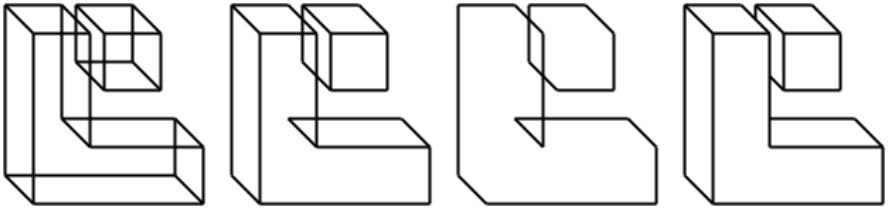


Ořezávání – základní pojmy

- Přední a obrysové hrany mohou být viditelné
- V liniových algoritmech obrysové hrany indikují změnu viditelnosti
- Pokud zobrazujeme jediné konvexní těleso, jsou všechny přední a obrysové viditelné (viz předchozí příklad)
- U více objektů a složitějších tvarů je potřeba testovat vzájemné překrytí přivrácených stěn



Ořezávání – základní pojmy



1. Bez určení viditelnosti
2. Přivrácené stěny
3. Obrysové hrany
4. Vyřešená viditelnost

Vektorové algoritmy – Robertsův algoritmus

- **Robertsův algoritmus**
- Nejstarší a implementačně nejjednodušší řešící viditelnost hran
- Původně navržen pro řešení viditelnosti skupiny konvexních mnohostěnů, lze ho zobecnit i pro obecné mnohostěny
- Postupně zpracováváme jednotlivé hrany těles
- Pro každou hranu testujeme, zda je zakryta jiným tělesem
- Pokud je zakryta jen částečně, dochází k jejímu rozdělení – na zakrytou a nezakrytou část
- Každá nezakrytá část se stává novou testovanou hranou

Vektorové algoritmy – Robertsův algoritmus

Algoritmus

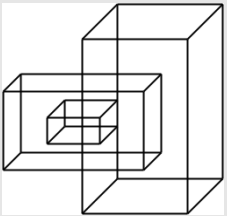
1. Rozděl hrany na zadní, přední a obrysové – zadními se dále nezabývej
2. Vytvoř prázdný seznam V potenciálně viditelných hran
3. Pro všechny obrysové a přední hrany H_i dělej
   1. Přidej hranu H_i do seznamu V
   2. Pro všechny mnohostěny M_j a pro všechny hrany H_k ze seznamu V dělej
      1. Vyjmi hranu H_k ze seznamu V
      2. Testuj hranu H_k na zakrytí mnohostěnem M_j
      3. Pokud jsou na hraně nalezeny nezakryté úseky (1 nebo 2), zařaď úseky do seznamu V
      4. Pokud mnohostěn hranu zakrývá celou, dále se jí nezabývej
   3. Vykresli všechny hrany ze seznamu V a vyprázdni jej

Vektorové algoritmy – Robertsův algoritmus

- Robertsův algoritmus je základem mnoha dalších metod – liší se seřazením hran, použitými metodami pro detekci zakrytých částí
- Všechny zachovávají myšlenku algoritmu – rozdělení potenciálně viditelných hran na elementární úseky, na nichž se může měnit viditelnost
- K nalezení úseků s neměnnou viditelností se používají obrysové hrany – pouze ty indikují změnu viditelnosti, pomocí nich tedy můžeme dělit hrany na menší úseky

Example

V objektu zakreslete přední, zadní a obrysové hrany.

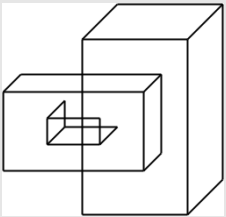


Vektorové algoritmy – Robertsův algoritmus

- Obrysové hrany dělí potenciálně viditelné hrany na elementární úseky

Example

V objektu zakreslete elementární úseky potenciálně viditelných hran.

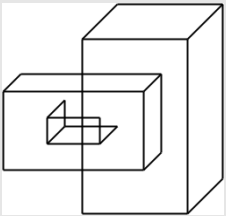


Vektorové algoritmy – Robertsův algoritmus

- Zkoumání zákrytu potenciálně viditelných elementárních úseků s nekonvexnímu mnohostěny se zdlouhavé
- **Appelův algoritmus** – potenciálně viditelné hrany se testují vůči obrysovým hranám
- Průsečíky ve kterých testovaná hrana „vstupuje“ pod obrysovou hranu (přechází za jiné těleso) jsou zaznamenány do seznamu a označeny jako vstupní/výstupní
- Seřazené průsečíky určují viditelnost hran

Example

V objektu vyřešte viditelnost.



Ořezávání – ořezávací algoritmy

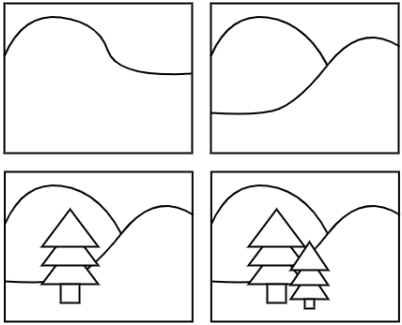
- **Ořezávací algoritmy** – clipping
- Odstraňují objekty nebo části objektů, které jsou mimo prostor pohledu (objemový pohled)
- Neřeší problém, kdy jeden objekt blokuje pohled na jiný
- Viz počítačová grafika

Ořezávání – řešení překrývání objektů

- **Klasifikace**:
  - Hidden surface removal – Algoritmy zjišťují, které povrchy není možné vidět z místa pozorovatele a ty se nerenderují – např. Backface elimination
  - Visible surface determination –Zjišťujeme, které plochy jsou viditelné a měly by být vykresleny – např. Warnockův algoritmus
- Rozdíl je nepatrný
- Další dělení – dle toho, kde pracují:
  - V prostoru obrazu (image space) – při promítání jsou všechny vzdálenosti promítány do jedné, uchováváme si informaci o vzdálenosti a dle ní pak řešíme viditelnost – např. Z-buffer
  - Ve scéně – vzdálenost se určuje ještě před promítáním

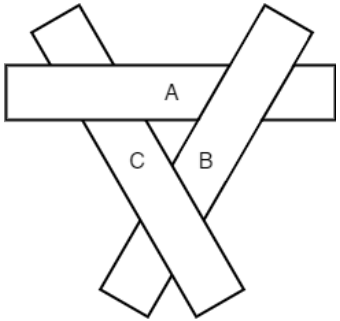
Malířův algoritmus

- **Malířův algoritmus** – setřídíme objekty ve scéně dle jejich hloubky (vzdálenosti od pozorovatele) a poté je vykreslíme
- **Princip**: Malíř nejprve nakreslí objekty v pozadí, pak je překreslí těmi, které jsou blíž
- Jak se přidávají další a další objekty, nové překrývají ty dříve vykreslené
- Vykreslíme-li objekty od nejvzdálenějších po nejbližší, scéna odpovídá tomu, čemu by měla
- V našem případě postupně vykreslujeme polygony
- Plochy třídíme dle nejvzdálenějšího bodu každého objektu – **priority list** (dle osy z)



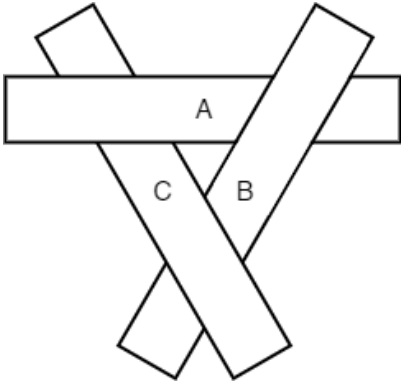
Malířův algoritmus

- Po setřízení objektů, objekty vykreslíme od nejvzdálenějšího po nejbližší
- Tímto algoritmem vykreslujeme všechny objekty ve scéně, i když nemusí být nakonec vůbec vidět
- Malířův algoritmus uvažuje jen nejvzdálenější bod objektu, což vede k problémům, když se objekty překrývají



Malířův algoritmus

- Nahoře je A za B ale před $C \rightarrow$ Vyrenderujeme C A B
- Pravá část indikuje pořadí A B C
- Levá část B C A



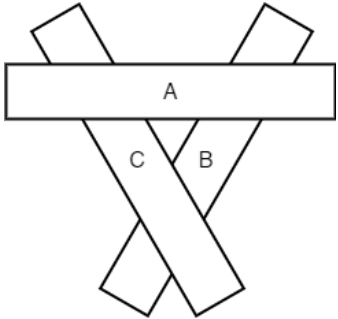
Example

Načrtněte výsledek pro uspořádání B C A.

Malířův algoritmus

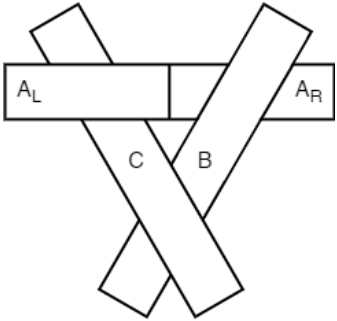
Example

Načrtněte výsledek pro uspořádání $B\ C\ A$.



Malířův algoritmus

- Problém se dá snadno vyřešit rozdělením jednoho obdélníku na dva (např. A)



- Nyní můžeme setřídit části $A_R\ B\ C\ A_L$

Malířův algoritmus

- Je potřeba najít části objektů, které se překrývají
- Překrytí – překrývají se z hodnoty objektů a zároveň se překrývají x i y souřadnice
- V takovém případě je potřeba jeden objekt rozdělit – např. vezmeme hranu jednoho obdélníku a dle ní rozřízneme druhý

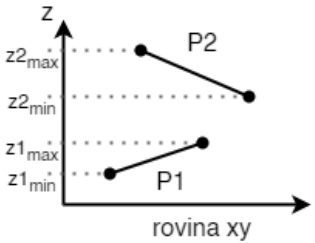
Depth-sort algorithm

- **Depth-sort algorithm** – rozšíření Malířova algoritmu o 5 testů
- Tyto testy slouží k odhalení potenciálních problémů
- Nejprve setřídíme objekty stejně jako v Malířově algoritmu
- Nejvzdálenější polygon (**aktivní plocha**) $P1$ se otestuje se všemi polygony, které leží v rozsahu z souřadnic $P1$
- Každý testujeme. Pokud testem projde je vykreslen
- Testy se provádí postupně, splnění libovolného z nich znamená, že plocha není překryta a je možné ji vykreslit

Depth-sort algorithm

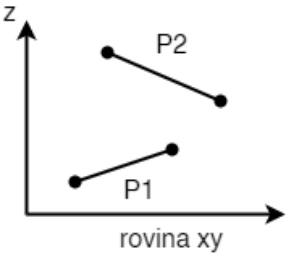
Označení

- $P1$ ... aktivní plocha
- $P2$ ... plocha vůči které testujeme
- $z1_{min}$, $z1_{max}$ ... rozsah z souřadnic aktivní plochy
- $z2_{min}$, $z2_{max}$ ... rozsah z souřadnic plochy $P2$



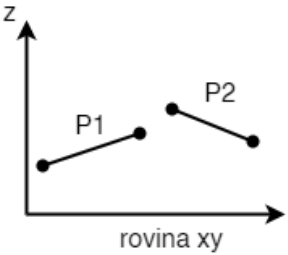
Depth-sort algorithm

- $P2$ nemá společné z souřadnice s $P1$
- $z2_{min} \geq z1_{max}$
- $P1$ leží zcela za plochou $P2$
- Tento případ nevede k nejednoznačnostem



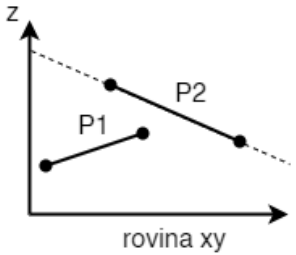
Depth-sort algorithm

- Promítneme-li plochy do roviny xy, průměty se nepřekrývají
- Porovnáme mezní souřadnice x a y souřadnic obou ploch



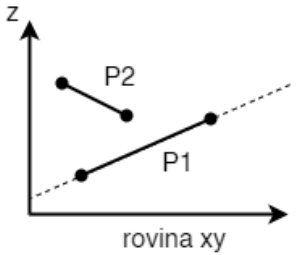
Depth-sort algorithm

- Všechny vrcholy aktivní plochy $P1$ leží v poloprostoru, který je určen testovanou plochou $P2$ a odvrácenému od pozorovatele



Depth-sort algorithm

- Všechny vrcholy plochy $P2$ leží v poloprostoru, který je určen aktivní plochou $P1$ a přivrácenému k pozorovateli



Depth-sort algorithm

- Pokud některý z testů selže je možné, že $P2$ překrývá $P1$ i přes to, že se $P1$ objevila v seznamu dříve
- Poslední dva testy opakujeme s opačným pořadím (zaměníme $P1$ a $P2$)
- Pokud neselže ani jeden, obrátíme pořadí $P1$ a $P2$ v priority listu
- Pokud selže i toto pořadí, jedna z ploch musí být rozdělena

Malířův a Depth-sort algorithm

- Malířův i Depth-sort algoritmy potřebují čas navíc pro setřízení ploch a testování/dělení ploch
- Všechny objekty, které vykreslujeme nemusí být viditelné
- Oproti následujícím algoritmům nemají tak velké paměťové nároky

Z-buffer algoritmus

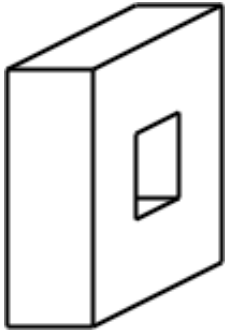
- **Z-buffer algoritmus** = paměť hloubky
- Frame-buffer . . . součást RAM, kde je uložen obraz, který se zobrazuje na obrazovce – pro každý pixel zde uchováváme jeho hodnotu (barvu)
- Z-buffer (depth buffer) . . . část paměti stejné velikosti, jako je obrazovka a uchovává informaci (1 hodnotu) o hloubce (z souřadnici) toho bodu, který je nejblíže pozorovateli a jehož průmět leží na souřadnicích tohoto pixelu v rastru
- Při renderování objektu se zjistí z souřadnice objektu, který zobrazujeme pro každý pixel na který se promítne a porovnává se se z-bufferem
- Jinak řečeno – porovnáváme souřadnice nově renderovaného objektu s již zobrazenými objekty v tomto bodě
- Pokud je nový objekt blíže – bod se vyrenderuje (nová barva je uložena do frame-bufferu) a je aktualizována hodnota z-bufferu
- Pokud je dále, již vykreslený objekt ho překrývá, tento bod přeskočíme
- Toto opakujeme pro každý bod (objekt může být skryt jen částečně)

Z-buffer algoritmus

- Dříve byla implementace náročná – velké paměťové nároky (pro každý pixel hodnota)
- Současné grafické karty mají vlastní z-buffery a pracují s nimi efektivně
- Stejná technika se používá i v jiných algoritmech – například stínování (tvorba stínu)
- Pomocí této techniky lze slučovat více obrázků dohromady (paralelně počítat části scény a pak je sjednotit)
- N bufferů, výsledná hodnota se bere z toho, kde je nejmenší z hodnota

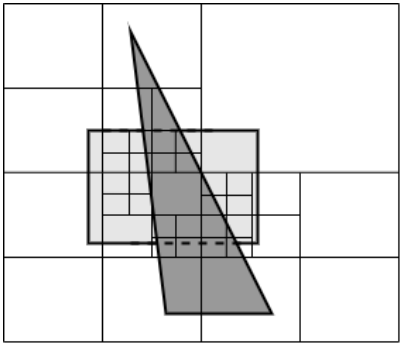
Backface elimination

- Jak bylo řečeno dříve – Plochy, které jsou odvrácené, jsou skryty přivrácenými plochami – můžeme je odstranit aniž bychom ovlivnili viditelnost ostatních **Backface elimination**
- Odstraňujeme části objektu, které nejsou viditelné – jsou překryty jinými částmi stejného objektu
- Odstraněním všech odvrácených stran neodstraníme všechny části, které nejsou viditelné – jiná viditelná část ho překrývá



Warnocks algorithm

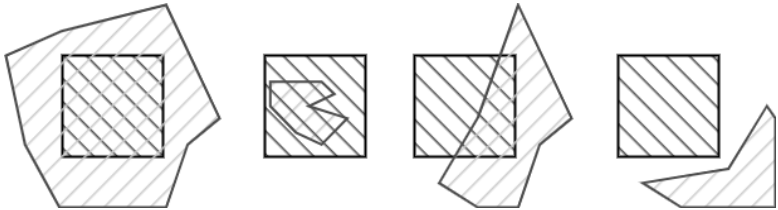
- **Warnocks algorithm** – dělení obrazovky
- Řeší viditelnost tak, že rozdělí scénu na menší části a v nich řeší viditelnost
- Prostor je dělen tak dlouho, dokud se v podokně objevuje nějaká nejasnost → jakmile je možné jednoznačně určit, jak má být dané okno vykresleno
- Pokud je těžké rozhodnout, jak dané okno vykreslit, rozdělí se dvěma kolmými řezy v rovině na 4 části
- Dělíme tak dlouho, dokud se nedostaneme na velikost pixelu



Warnocks algorithm

Vztah plochy k podoknu (regionu)

1. Obklopující – plocha překrývá celý region
2. Obsažená – plocha je celá uvnitř regionu
3. Protínající – plocha částečně překrývá region
4. Disjunktní – plocha je celá mimo region



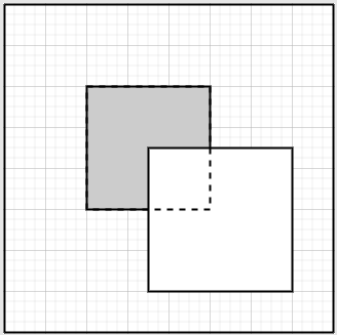
Warnocks algorithm

- Pokud v regionu nedochází k nejasnostem – **simple region**, můžeme ho vykreslit
- **Prázdný region** – všechny plochy jsou s ním disjunktní $\rightarrow$ vykreslíme ho barvou pozadí
- **Region obsahuje jen jednu plochu** – plocha ho může obklopovat, být v něm obsažená nebo ho jen protínat – vykreslíme ho barvou plochy případně pozadí
- **Region obsahuje více ploch** – pokud ho nejbližší plocha celý obklopuje, je snadné ho vykreslit – vykreslíme ho barvou nejbližší plochy
- **Ostatní případy** je nutné řešit rozdělením regionu
- existují i varianty, které dělí region optimálněji, ne na 4 stejné části

Warnocks algorithm

Example

Rozdělte obrázek na regiony a určete o jaký typ regionu (viz předchozí slide) se jedná.

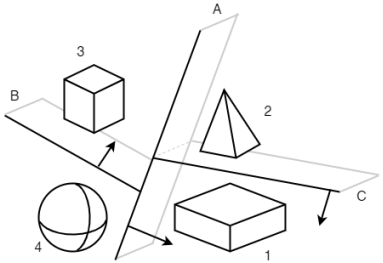


BSP stromy

- **Binary space partition trees** – BSP
- Řešíme polohu objektů (jejich viditelnost) v prostoru obrazu
- Výpočet musí být opakován, když se změní pozice pozorovatele nebo směr pohledu
- Alternativou je sestavit BSP stromu scény, který je nezávislý na pozici pozorovatele, ale díky němuž můžeme pro libovolnou jeho pozici jednoznačně určit pořadí vykreslování
- Je vhodný pro statické scény
- **BSP strom** – binární strom jehož každý vnitřní uzel představuje řez rozdělující prostor na 2 části
- Při tvorbě stromu postupujeme shora dolů – rekurzivně dělíme 3D prostor obsahující zobrazované objekty

BSP stromy

- **Základní myšlenka** – prostor je plochou rozdělen na 2 části
- Objekty seskupíme podle toho, na které polovině od plochy leží (přivrácená/odvrácená strana plochy dle normály)
- Tyto prostory dále dělíme dalšími rovinami



- Nejprve rozdělíme prostor rovinou A (šipka představuje normálu a tedy určuje, který podprostor leží na kladné a který na záporné straně)
- Objekty 1 a 2 leží v kladné (levý podprostor) a 3 a 4 v záporné (pravý)
- Kladná část je dále rozdělena rovinou B a záporná rovinou C

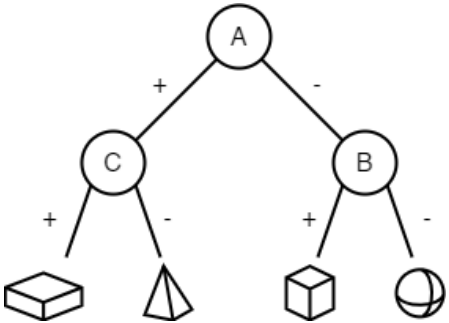
Example

Nakreslete BSP strom výše uvedeného příkladu.

BSP stromy

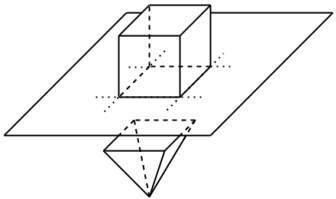
Example

Nakreslete BSP strom.



BSP stromy

- **Zobrazení**
  - Procházíme strom od kořene a v každém uzlu určujeme, který poloprostor je vůči pozorovateli vzdálenější a který bližší
  - Nejprve vykreslíme vzdálenější část a pak oblast bližší k pozorovateli
- Hledání řezných rovin je složitá úloha
- Mnohdy se jako řezné plochy berou strany těles (plocha v níž strana tělesa leží)

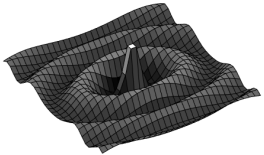


- V případě nejednoznačnosti (jako u Malířova algoritmu) se může stát, že jde řezná rovina skrz nějaký objekt $\rightarrow$ rozdělíme ho na 2 části

Algoritmus plovoucího horizontu

- Předchozí algoritmy byly univerzální – zpracovávaly množinu prostorových ploch bez využití případných dalších informacích o jejich topologii
- **Algoritmus plovoucího horizontu** – předpokládá speciální uspořádání vstupních dat
- **Příklad**:
  - Chceme zobrazit plochu určenou explicitní funkcí dvou proměnných $w = f(u, v)$ $u \in \langle u_{min}, u_{max} \rangle$, $v \in \langle v_{min}, v_{max} \rangle$
  - Takovou plochu nejsnáze zobrazíme soustavou řezů kolmých na osy u a v
  - Po provedení pohledové transformace kreslíme jednotlivé řezy od pozorovatele směrem dozadu

$$f(u,v) = \frac{\cos(\sqrt{u^2+v^2}}{\sqrt[3]{u^2+v^2}}$$
$$u, v \in \langle -10, 10 \rangle$$



Algoritmus plovoucího horizontu

- Předpokládejme, že vzdálenější řezy mají větší souřadnici v
- Viditelnost budeme určovat metodou **plovoucího horizontu**, terá vykresluje grafické prvky (úsečky) každého dalšího řezu jen tehdy, pokud se nachází nad nebo pod již zaplněnou oblastí na obrazovce
- Využíváme 2 pomocná pole – **horní horizont (HH)** a **dolní horizont (DH)**
- Počet pixelů každého pole je roven počtu pixelů obrazovky ve vodorovném směru
- Horizonty obsahují pro každou souřadnici x její odpovídající nejvyšší (resp. nejnižší) souřadnici y dosud nakresleného pixelu na obrazovce
- Při kreslení dalšího řezu jsou souřadnice bodů řezu porovnány s oběma horizonty a nakresleny pouze v případě, že nepadnou do oblasti těmito horizonty omezené

Algoritmus plovoucího horizontu

Algoritmus

1. Inicializuj všechny prvky HH hodnotou $-\infty$ a všechny prvky DH hodnotou ∞
2. Po krocích rostoucí $v_j \in \langle v_{min}, v_{max} \rangle$ urči křivku $f(u, v_j)$
3. Pro všechna $u_i \in \langle u_{min}, u_{max} \rangle$ vypočítej $w_{ij} = f(u_i, v_j)$
   1. Pohledovou transformací transformuj vypočítaný bod $[u_i, v_j, w_{i,j}]$ na bod $[x, y]$ v souřadnicích obrazovky
   2. Je-li $y \leq HH[x]$ a zároveň $y \geq DH[x]$ pokračuj zpracováním dalšího u_i
   3. Nakresli bod $[x, y]$ a podle hodnoty y aktualizuj buď $HH[x]$ nebo $DH[x]$

Algoritmus plovoucího horizontu

Example

Předpokládejme, že máme funkci $f(u, v)$ zadanou následující tabulkou.

		u								
		1	2	3	4	5	6	7	8	9	10
v	1	0	0	1	1	2	2	1	1	0	0
	2	0	0	0	0	1	1	1	1	2	2
	3	2	2	2	2	1	1	1	1	0	0
	4	-1	-1	-1	0	0	0	0	1	1	1

Pohledová transformace zobrazí $[u, v, w] \to [u, w]$. Simulujte algoritmus Plovoucího horizontu.

Algoritmus plovoucího horizontu

- Pouze základní řešení – rasterizace je prováděna v 3D prostoru, takže krok měnící se souřadnice u_i musí odpovídat velikosti 1 pixelu
- Praktické rozšíření – vzorkování funkce ve větších intervalech a následná rasterizace vzniklé posloupnosti úseček
- Algoritmus kreslí pouze řezy kolmé na osu v
- Je nutno vyřešit vykreslování spojnic řezů – obdobným způsobem

Zpracování poloprůhledných objektů

- Poloprůhledné objekty (tělesa s poloprůhlednými stěnami) v některých algoritmech řešících viditelnost působí problémy
- Poloprůhledné plochy nezakrývají vzdálenější objekty, naopak je potřeba barvu poloprůhledných objektů smíchat s barvou objektů v pozadí (morfing, α míchání z počítačové grafiky $\alpha A + (1 - \alpha B)$)
- Při vykreslování poloprůhledných ploch je nutno dodržet vykreslování zezadu dopředu (míchání barev není komutativní!)
  Máme za sebou 2 poloprůhledné plochy, přední má průhlednost α_N a zadní α_F.
  Mícháme barvy C_N a C_F s barvou pozadí C_B. Ve správném pořadí získáme:
  $\alpha_N \cdot C_N + (1 - \alpha_N) \cdot [\alpha_F \cdot C_F + (1 - \alpha_F) \cdot C_B]$
  Pokud vykreslíme plochy v opačném pořadí
  $\alpha_F \cdot C_F + (1 - \alpha_F) \cdot [\alpha_N \cdot C_N + (1 - \alpha_N) \cdot C_B]$
  Subtraktivní člen $\alpha_N \alpha_F$ není aplikován na barvu C_F ale na C_N.

Zpracování poloprůhledných objektů

Malířův algoritmus

- Pracuje bez problému bez modifikací

Z-buffer – modifikovaný

1. Inicializuj z-buffer a frame-buffer
2. Postupně načítej jednotlivé plochy. Je-li plocha neprůhledná, okamžitě jí zpracuj jako v základním algoritmu.
   V opačném případě její zpracování odlož (plochu si zapamatuj).
3. Po zpracování všech neprůhledných ploch seřaď zapamatované plochy podle vzdálenosti
4. Postupně zpracuj poloprůhledné plochy od nejvzdálenější k nejbližší jako v základním algoritmu s tím, že namísto zápisu do obrazové paměti použij α míchání