Kapitola 5

Řešení viditelnosti

Řešit viditelnost ve scéně umí většina grafických programů. Cílem je určit ty objekty, resp. jejich části, které jsou viditelné z určitého místa. Tyto algoritmy jsou vždy svázány s určitou reprezentací objektů ve scéně. Podle toho, v jakém tvaru jsou výstupní data, rozdělujeme algoritmy do dvou skupin:

1. Vektorové algoritmy – jejich výstupem je soubor geometrických prvků, např. úseček, které představují viditelné části zobrazovaných objektů. Používá se především v technických aplikacích. Bez změny viditelnosti lze plynuje úsečky zvětšovat. Nevýhodou je, že při numerické chybě je špatně celá úsečka, což výrazně mění celý pohled.

2. Rastrové algoritmy – pracují pouze v rastru. Výsledek je obraz, jehož jednotlivé pixely obsahují barvu odpovídajících viditelných ploch.Nevýhodou je pevný rozměr obrázku. Většina metod patří do této skupiny. Mezi rastrové algoritmy patří z-buffer, malířův algoritmus či algoritmus plovoucího horizontu.

5.1 Paměť hloubky – z-buffer

Tato metoda patří k nejznámějším a nejefektivnějším. Základem je použití paměti hloubky – z-bufferu. Ta tvoří dvojrozměrné pole, jehož rozměry odpovídají velikosti obrazu. Každá položka paměti hloubky obsahuje souřadnici z toho bodu, který leží nejblíže pozorovateli a jehož průmět leží v odpovídajícím pixelu v rastru. Na obr. vidíme scénu a odpovídající z-buffer. Čím blíže je předmět k pozorovateli, tím světlejší má odstín.

Výhody metody:

- není třeba třídit,
- umí správně vykreslit nestandardní situace (např. průseky),
- rychlost, jednoduchost výpočtu.

Obrázek 5.1: Z-buffer

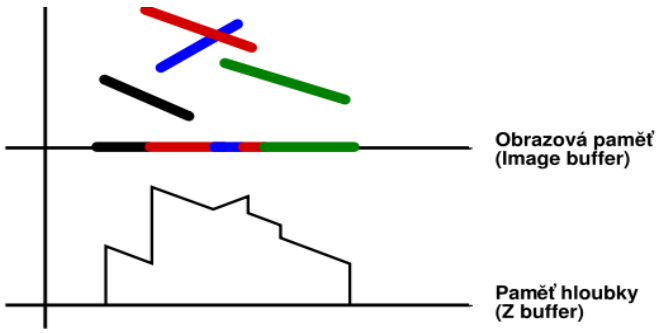
Nevýhody metody:

- větší paměťová náročnost,
- některé pixely se vícekrát překreslují.

Algoritmus

Pro každý pixel ukládáme jeho barvu a vzdálenost od pozorovatele (z-buffer). V inicializační části nastavíme barvu pro všechny pixely na barvu pozadí, paměť hloubky vypníme hodnotou minus nekonečno.

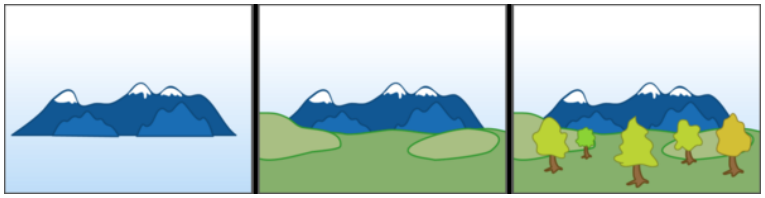
1. Každou plochu rozlož na pixely a pro každý její pixel $[x_i, y_i]$ stanov hloubku z_i
2. Má-li z_i větší hodnotu než položka $[x_i, y_i]$ v z-bufferu pak:
   a. obarvi pixel$[x_i, y_i]$ v obrazové paměti barvou této plochy
   b. položku $[x_i, y_i]$ v z-bufferu aktualizuj hodnotou z_i



Obrázek 5.2: Princip fungování algoritmu z-buffer

5.2 Malířův algoritmus – Painter's algorithm

Algoritmus je založen na myšlence vykreslování všech ploch postupně odzadu dopředu - přes objekty v pozadí se kreslí objekty v popředí. Inspirováno představou práce malíře který na podkladovou barevnou vrstvu nanáší další vrstvy (viz obrázek 5.3).

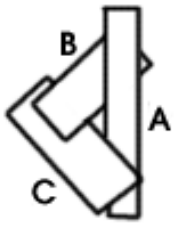


Obrázek 5.3: Princip malířova algoritmu

Algoritmus

Nejprve nalezneme pro každou plochu její nejmenší z-ovou souřadnici. a podle ní plochy uspořádáme. První plocha v seznamu je označena jako aktivní a je podrobena několika testům překrývání s ostatními plochami a pokud lze po nich rozhodnout, že leží za všemi ostatními je vykreslena a vyřazena ze seznamu. V opačném případě dojde v seznamu k výměně aktivní plochy s plochou u které dopadly všechny testy negativně.

Malířův algoritmus není vhodný pro všechny případy. Některé pozice (protínání) zobrazovaných objektů mohou vést ke špatnému zobrazení. Pokud se objekty navzájem překrývají může to vést až k zacyklení algoritmu. Takový případ nastane například při situaci na obrázku 5.4.



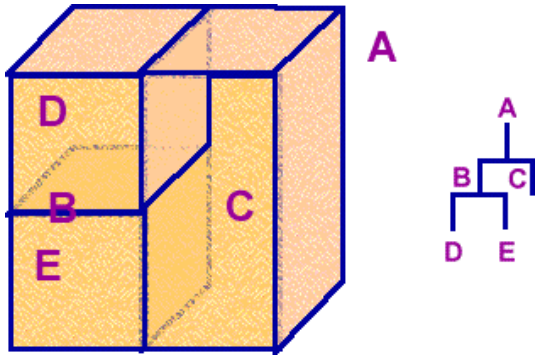
Obrázek 5.4: Problémový případ pro malířův algoritmus

```
Setřid' objekty podle hloubky.
Vstup: plošky, které se neprotínají
for všechny objekty v určeném pořadí do {
  for každý pixel[x][y] pokrytý objektem do{
    Obarvi pixel[x][y] barvou objektu.
  }
}
```

5.3 Malířův algoritmus se stromem BSP

BSP (Binary Space Partitioning trees) jsou datové struktury, které popisují danou scénu s jejími vazbami. Pomáhají urychlit řadu algoritmů, ale jsou vhodné pouze pro statické scény. Popisují složení objektů a jejich vzájemné vazby.

Při zobrazování procházíme strom BSP od kořene a v každém uzlu určujeme, který poloprostor je vůči poyorovateli vzdálenější a který bližší. Nejprve vykreslíme vzdálenější část prostoru, poté obsah aktuálního uzlu a nakonec oblast bližší pozorovateli.



Obrázek 5.5: BSP strom

5.4 Metoda vržení paprsku

Tato metoda spočívá ve vysílání paprsku od pozorovatele a následné hledání průsečíků s objekty ve scéně a určení toho, který je pozorovateli nejblíže. Pokud paprsek protíná pouze jeden objekt, je situace jednoduchá. Jestliže protíná průnik více těles, je nutné užít množinové operace.

```
for každý pixel[x][y] obrazu do {
/* Nejbližší objekt zasažený promítacím paprskem */
   for každý objekt ve scéně do
       if existuje průsečík paprsku s objektem
          { Porovnej hloubku z a vyber bližší objekt }.
   if nalezen objekt
         { Obarvi pixel[x][y] barvou objektu }
   else
         { Obarvi pixel[x][y] barvou pozadí }
}
```

Kontrolní otázky

1. Na jaké druhy rozdělujeme algoritmy viditelnosti podle typu výstupu?
2. Jaké jsou výhody a nevýhody z-bufferu?
3. Popište princip z-buffer určování viditelnosti.
4. Popište princip malířova algoritmu.
5. Jaké jsou nevýhody malířova algoritmu?
6. Co je to BSP strom?
7. Popište základní myšlenku metody vržení paprsku.

5.4.1 Literatura

Z-buffer
http://cs.wikibooks.org/wiki/Viditelnost
http://cgg.ms.mff.cuni.cz/~pepca/lectures/pgr003.html
http://www.pctuning.cz/index.php?option=com_content&task
=view&id=8721&Itemid=44&limit=1&limitstart=3

Malířův algoritmus
http://pocitacova-grafika.kvalitne.cz/maliruv-algoritmus
http://www.fi.muni.cz/~sochor/M4730/Slajdy/Viditelnost.pdf
http://en.wikipedia.org/wiki/Painter's_algorithm
http://medialab.di.unipi.it/web/IUM/Waterloo/node67.html
http://www.ibiblio.org/e-notes/3Dapp/Hidden.htm

BSP stromy
http://cgg.ms.mff.cuni.cz/~pepca/prg022/vondrak.html