6. VIDITEĽNOSŤ

Trojrozmernými zobrazeniami sa v počítačovej grafike rozumejú zobrazenia priestorových scén v rovnobežnom alebo stredovom premietaní. Pretože rovnobežné premietanie je špeciálnym prípadom stredového premietania (pre nevlastný stred premietania), obmedzujeme sa často iba na stredové premietanie. Na druhej strane, vďaka tomu, že stredové premietanie možno aplikáciou vhodnej projektívnej transformácie previesť na rovnobežné premietanie, pracuje sa v niektorých algoritmoch len s rovnobežnými premietaniami.

Ak však vo zvolenom premietaní zobrazíme všetky časti zobrazovaného objektu, nemusí byť takto získaný obraz (priemet) príliš názorný. Pre zvýšenie názornosti je potrebné odstrániť obrazy neviditeľných čiar a plôch t.j. potrebujeme vyriešiť viditeľnosť na obrázku. Toto je jeden veľmi dôležitý problém počítačovej grafiky, ktorý riešia, viac-menej uspokojivo, mnohé algoritmy počítačovej grafiky.

Základný princíp riešenia viditeľnosti je testovanie každého bodu objektu či nie je pozorovateľovi zakrytý nejakou časťou zobrazovaného objektu. Tento spôsob je však nielen časovo veľmi náročný, ale aj prakticky nepoužiteľný a preto sa hľadajú jednoduchšie, viac-menej heuristické metódy na riešenie tejto úlohy. Prístupy k riešeniu tohto problému bývajú rôzne a preto aj vypracované algoritmy sú závislé nielen na type zobrazovaných objektov, ale aj na výstupnom grafickom zariadení.

Problém viditeľnosti objektov na obraze sa obyčajne rieši buď odstránením neviditeľných čiar alebo plôch (stien telies). Teoreticky, oba spôsoby riešenia možno realizovať ako v priestore objektov (3D), tak aj v priestore obrazov (2D-priemetňa, obrazovka). V prvom prípade hovoríme o objektovo orientovaných algoritmoch, lebo o tom ktoré časti objektov sú viditeľné a ktoré nie sú, sa rozhoduje na základe geometrických vzťahov medzi nimi, zatiaľ čo v druhom prípade o obrazovo orientovaných algoritmoch, lebo vychádzajú z obrazu a skúmajú ho z hľadiska metód jeho konštrukcie.

Pri zostrojovaní obrazov zložitých, hlavne hladkých priestorových objektov sa nezaobídeme bez ich matematického opisu. Obyčajne sa stretávame s dvomi prístupmi. V prvom prístupe sa na povrchu reálneho priestorového objektu vykreslí sústava čiar na základe ich matematického opisu, ktorá následne dáva geometrickú predstavu o samotnom objekte. V druhom prípade sa povrch telesa rozdelí na určitý počet častí ohraničených uzavretými čiarami. Ku každej z nich sa zostrojí jednoducho matematicky opísateľná aproximácia a súhrn obrazov všetkých takto zostrojených aproximácií častí povrchu telesa sa potom považuje za obraz povrchu telesa, čiže aj telesa samotného. Vo funkcii takýchto aproximácií konečných častí povrchu telesa sa často používajú „rovinné trojuholníky“, triangulácia plochy je pomerne jednoduchá procedúra vďaka tomu, že tri body určujú jedinú rovinu. Okrem konštrukcie trojuholníkov, ich vypĺňanie a určovanie vzájomných prienikov nespôsobuje žiadne väčšie problémy. Vážny nedostatok tejto reprezentácie geometrických objektov spočíva v tom, že na vyvolanie dojmu hladkosti potrebujeme príliš mnoho trojuholníkov alebo použitie špeciálnych metód. Aj napriek tomu sa s takto reprezentovanými mnohostenmi stretávame v počítačovej grafike najčastejšie.

Odstraňovanie neviditeľných stien a čiar.

Možno povedať, že vo vektorovej grafike sa obvykle stretávame s algoritmami na odstraňovanie neviditeľných čiar a v rastrovej grafike s algoritmami na odstraňovanie neviditeľných plôch. Keďže v našom prípade pôjde takmer vždy o zobrazovanie na zariadeniach rastrovej grafiky, tak sa budeme venovať hlavne algoritmom na odstraňovanie neviditeľných stien.

Zakrývanie (tienenie plôch).
Nech S_1*, S_2*, ...,S_n* sú steny (konečné rovinné časti) všetkých objektov participujúcich na danom obraze.

Definícia 1. Budeme hovoriť, že stena S_i* zakrýva (tieni) stenu S_j* v smere určenom vektorom **v**, ak sú splnené podmienky:
(a) $S_i^* \cap S_j^* = \emptyset \wedge S_{ij} := S_i \cap S_j \neq \emptyset$,
kde S_i, S_j sú ortogonálne priemety stien S_i*, S_j* do priemetne kolmej k vektoru **v**
(b) $\exists X_i^* \in S_i^*$: tak, že $X_i \in S_{ij} \wedge$ vzdialenosť $|O X_i^*|$ bodu X_i* od začiatku O v smere vektora **v** je menšia než takáto vzdialenosť odpovedajúceho bodu $\bar{X}_i^*$ zo steny S_j*.

Poznámka: Poslednú formuláciu obvyklú v PG, zrejme možno nahradiť jednou z ekvivalentných podmienok $|X_i X_i^*| > |X_i \bar{X}_i^*|$ resp. $|OX_i^*| > |O\bar{X}_i^*|$.

Obyčajne sa predpokladá, že smer **v** je totožný so smerom osi *z*-ovej. Pre stredové premietanie má predchádzajúca definícia nasledujúcu formuláciu:

Definícia 1´. Hovoríme, že stena S_i* zakrýva (zatieňuje) pozorovateľovi umiestnenému v bode C stenu S_j* ak sú splnené podmienky:
(a) $S_i^* \cap S_j^* = \emptyset \wedge$ stredové priemety S_i, S_j stien S_i*, S_j* na priemetňu neprechádzajúcu bodom C majú neprázdny prienik S_{ij} t.j. $S_{ij} = S_i \cap S_j \neq \emptyset$
(b) aspoň jeden bod X_i* steny S_i*, ktorého priemet X_i patrí do S_{ij} má tú vlastnosť, že pre jemu odpovedajúci bod $\bar{X}_i^*$ na stene S_j* (t.j. priesečník $\bar{X}_i^* = CX_i \cap S_j^*$) platí, že $|X_i^* C| > |\bar{X}_i^* C|$.

Takto definovaná relácia zakrývania stien pre najjednoduchšie, ale v počítačovej grafike najčastejšie používané stredové premietanie Typ1[$S = O$= [0,0,0], priemetňa π: z = -d] alebo Typ2[S = [0,0,-d], priemetňa π: z = 0] je z hľadiska riešenia úloh o rozdelení elementov scény na viditeľné a neviditeľné ekvivalentná s tou, ktorú sme definovali skôr pre ortogonálne premietanie, lebo obraz scény v tomto stredovom premietaní možno získať tak, že ju najskôr zobrazíme projektívnou transformáciou priestoru $\bar{E}_3$ určenou maticou

Typ1[$S = O$= [0,0,0], priemetňa π: z = -d] Typ2[S = [0,0,-d], priemetňa π: z = 0]

$$P1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \qquad P2 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 1 \end{pmatrix}$$

a až takto získaný priestorový obraz scény ortogonálne premietneme do priemetne pomocou príslušnej matice ortoprojekcie:

Typ1

$$M1_{orth}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&-d\\0&0&0&0\end{pmatrix}$$

Typ2

$$M2_{orth}=\begin{pmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{pmatrix}$$

Pre všetky algoritmy viditeľnosti je veľmi dôležitá rýchlosť. Preto je dôležité najskôr identifikovať z pozorovacieho miesta zakryté (od pozorovateľa odvrátené) steny a hrany, ktoré sú ich prienikmi, lebo tieto nemôžu byť na obraze viditeľné.

Algoritmy na určovanie viditeľnosti priestorových objektov možno podľa typu pracovných priestorov s prihliadnutím na rýchlosť výpočtov rozdeliť do dvoch skupín:

a) objektovo orientované (algoritmy v priestore objektov).
Ich výpočtová zložitosť je $O(n^2)$. Pracujú na princípe porovnávania vzájomnej polohy telies a ich častí, akými sú steny a hrany. Ich symbolický pracovný princíp je „ Pre každý objekt zisti, ktorá jeho časť je viditeľná.“ Zložitosť $O(n^2)$ je daná tým, že každý objekt sa testuje voči ostatným. Vhodným usporiadaním a reprezentáciou možno dosiahnuť zložitosť $O(n\log n)$.

b) obrazovo orientované (algoritmy v priestore obrazu)- rastrové algoritmy
Základom týchto algoritmov sú testy určujúce pre každý pixel obrazovky resp. jej okna, ktorá časť premietnutého objektu je v tomto pixli najbližšie k pozorovateľovi, a je teda viditeľná. Stručne povedané „Pre každý pixel sa zisťuje, ktorý objekt – stena je v ňom viditeľná“. Algoritmy pracujú s premietnutými a potom rasterizovanými plochami. Testy viditeľnosti sa realizujú lokálne v určitej časti rastra. Výsledkom je lineárna výpočtová zložitosť $O(n)$ pre danú veľkosť obrazu.

Niektoré algoritmy kombinujú oba vyššie uvedené prístupy a nezriedka sú urýchľované predspracovaním vstupných dát. Väčšinou predpokladáme, že vstupom algoritmov sú objekty opísané hraničnými reprezentáciami.

Väčšina algoritmov na viditeľnosť povrchov používa obrazovo orientované, čiže rastrové algoritmy, hoci objektovo orientované prístupy môžu v niektorých prípadoch viesť k efektívnejším algoritmom. Na druhej strane na určenie viditeľnosti wire-frame modelov objektov sa väčšinou používajú objektovo orientované prístupy, hoci mnohé rastrové algoritmy pre plôškové povrchy možno jednoducho adaptovať aj na určenie viditeľnosti v drôtených modeloch.

Hoci existujú veľké rozdiely v základných prístupoch rôznych viditeľnostných algoritmov mnohé z nich využívajú triedenie resp. usporiadanie a koherentnosť, čiže nemennosť, alebo aspoň istú pravidelnosť pri výskyte určitých vlastností susedných objektov v scéne, pixlov na tej istej úsečke, alebo skenovacej priamke, možnosti využitia interpolácií pre významné urýchlenie výpočtov a pod.

Algoritmus pre odstránenie zadných stien

• Predpokladajme, že stena priestorového telesa obsahuje tri nekolineárne body $P_1=[x_1,y_1,z_1], P_2=[x_2,y_2,z_2], P_3=[x_3,y_3,z_3]$. Potom rovnica jej stenovej roviny α má tvar:

(1) $[(P_2-P_1)\times(P_3-P_1)].(X-P_1)=0$ resp. $(X-P_1,P_2-P_1,P_3-P_1)=0$, čiže

(1) $\begin{vmatrix} x-x_1 & y-y_1 & z-z_1 \\ x_2-x_1 & y_2-y_1 & z_2-z_1 \\ x_3-x_1 & y_3-y_1 & z_3-z_1 \end{vmatrix}=0$ alebo $\begin{vmatrix} x & y & z & 1 \\ x_1 & y_1 & z_1 & 1 \\ x_2 & y_2 & z_2 & 1 \\ x_3 & y_3 & z_3 & 1 \end{vmatrix}=0$.

Ak determinant na ľavej strane poslednej rovnosti rozložíme podľa prvkov prvého riadku nadobudne (1) tvar:

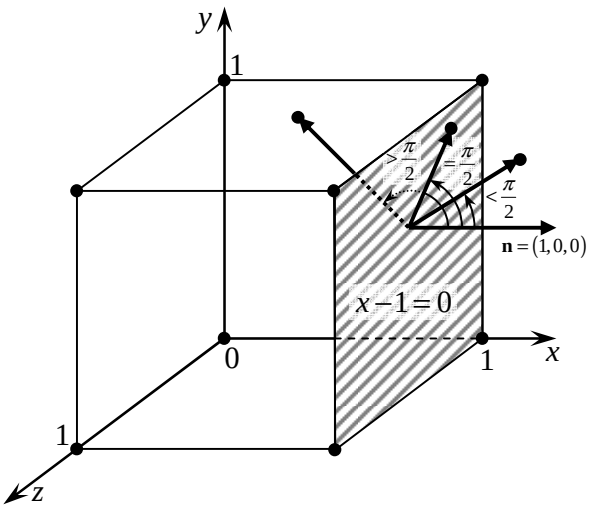
(1) $\alpha\equiv Ax+By+Cz+D=0 \iff A(x-x_1)+B(y-y_1)+C(z-z_1)=0$,

kde

(2) $A=\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix}$; $B=\begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$; $C=\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix}$; $D=-\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$.

• Každá stena (a teda aj stenová rovina) má „dve strany“: vnútornú (ktorou susedí s telesom) a vonkajšiu. Ak postupnosť bodov P_1, P_2, P_3 bola zvolená v kladnom smere (t.j. proti smeru pohybu hodinových ručičiek pri pohľade z vonkajšej strany stenovej roviny), tak vektor $\mathbf{n}=(P_2-P_1)\times(P_3-P_1)=[A,B,C]$ je normálovým vektorom steny „smerujúcim“ z jej vnútornej strany na vonkajšiu a nazýva sa jej vonkajším normálovým vektorom (lebo P_2-P_1, P_3-P_1 a normálový vektor **n** podľa definície vektorového súčinu musia tvoriť pravotočivý systém).

• Každá stenová rovina α (na obr. rovina x-1 = 0) rozdeľuje priestor na vnútornú (obsahujúcu teleso) a vonkajšiu časť.



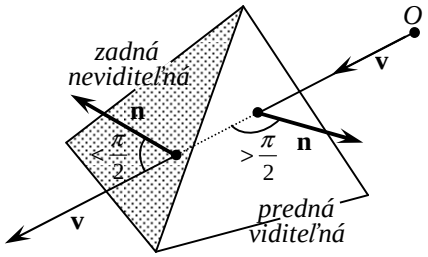
Body priestoru patriace vnútornej [vonkajšej] časti sa nazývajú vnútorné [vonkajšie].
Pretože pre každý bod $P=[x, y, z] \notin \alpha$ platí $Ax+By+Cz+D=\mathbf{n}.(P-P_1)$ $\left[:= f(P)\right]$ je zrejmé, že bod P je **vnútorný** [**vonkajší**] $\Leftrightarrow$ uhol $\left[P-P_1, \mathbf{n}\right] > \pi/2$ [$< \pi/2$] $\Leftrightarrow$ $\mathbf{n}.(P\text{-}P_1) < 0$ [>0] $\Leftrightarrow$

$\Leftrightarrow$ $Ax+By+Cz+D < 0$ [$Ax+By+Cz+\mathrm{D} > 0$].
Toto tvrdenie prirodzene platí iba v pravotočivej karteziánskej súradnicovej sústave.

• Kvôli lepšiemu pochopeniu tvrdení odporúčame ich demonštrácie napr. na jednej stene jednotkovej kocky, ktorej tri vrcholy ležia na súradnicových osiach a jeden je v začiatku súradnicovej sústavy.

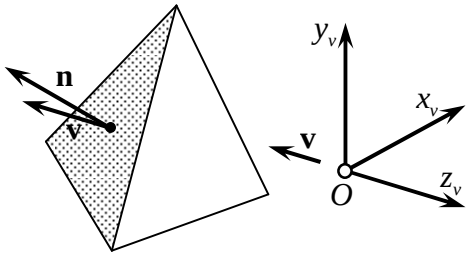
• Ak má (bodový) pozorovateľ vidieť prednú – vonkajšiu stranu steny nemôže byť umiestnený vo vnútornom bode. To však znamená, že u viditeľnej steny nemôže na žiadnej pozorovacej úsečke OS (O- pozorovateľ, $S \in$ steny) existovať vnútorný bod
$\left[M \in OS \Rightarrow f(M) = f((1-t)O+tS) = (1-t)f(O)+tf(S) = (1-t)f(O) \geq 0 \wedge f(M) < 0 \Rightarrow \text{spor}\right]$.
To ale znamená, že ak na pozorovacej úsečke k nejakej stene existuje vnútorný bod, je táto stena zadná (skrytá, neviditeľná pre pozorovateľa).

• Aby sme toto kritérium zjednodušili, predpokladajme, že $\mathbf{v} = \left[v_x, v_y, v_z\right]$ je vektor z pozorovacieho sektora oka, prípadne kamery, reprezentujúci smer pozorovania. Potom mnohouholníková stena je zadnou práve vtedy, keď $\angle\, \mathbf{v}.\mathbf{n} < \pi/2$ resp. $\mathbf{v}.\mathbf{n} > 0$.(obr.a).



Obr.a)

• Ak opis objektu (obr.b) je konvertovaný do súradnicovej sústavy pozorovateľa $\langle O-\text{oko}, x_v, y_v, z_v \rangle$,



Obr.b)

ktorá je karteziánska a pravotočivá (s pozorovacou rovinou $\langle O, x_v, y_v \rangle$) a smer pozorovania je určený osou $- z_v$ ($\Rightarrow v_z < 0$), tak $\mathbf{v}.\mathbf{n} = v_z C$ (> 0) a preto platí:
každý rovinný mnohouholník ležiaci v rovine, ktorej vonkajší normálový vektor spĺňa podmienku $C \leq 0$ je zadnou (neviditeľnou) stenou skúmaného objektu (= je tam preto, že pre C=0 je **n** je rovnobežný s rovinou $r(x_v, y_v)$ a uvažovaná stena je kolmá na rovinu $(x_v, y_v) \Rightarrow$ uvažovaná stena je rovnobežná s osou z_v, čiže je rovnobežná so smerom pozorovania).

- pre ľavotočivé pozorovacie súradnicové systémy so smerom pozorovania $+ z_v$ má táto podmienka tvar: $C \geq 0$.

Záver : pre jediný konvexný mnohosten (napr. štvorsten) alebo pre scénu obsahujúcu iba nezakrývajúce sa konvexné mnohosteny možno opísaným spôsobom – testom pre zadné steny, identifikovať všetky neviditeľné steny. U zložitejších scén (nekonvexné, zakrývajúce sa) je to asi polovica.

Poznámka 1. Zrejme platí:

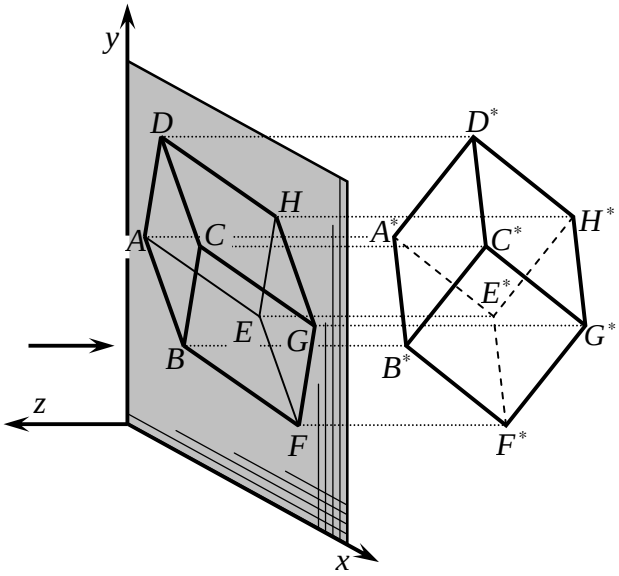
a) hrana, ktorá je prienikom dvoch predných stien je potenciálne viditeľná
b) hrana, ktorá je prienikom dvoch zadných stien je neviditeľná
c) hrana, ktorá je prienikom prednej a zadnej steny je obrysovou hranou a je potenciálne viditeľná

Poznámka 2. V prípade konvexného mnohostena sú všetky potenciálne viditeľné hrany skutočne viditeľné. U nekonvexných mnohostenov treba zistiť či potenciálne viditeľná hrana nie je zakrytá inou stenou (úplne alebo čiastočne)

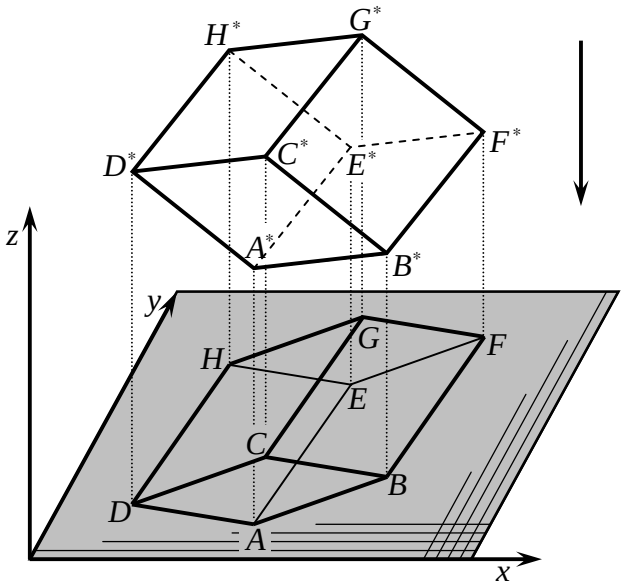
Algoritmus *z*-buffer

Tento algoritmus je jeden z najzákladnejších a najznámejších algoritmov na zisťovanie viditeľnosti priestorových objektov na obrazovke počítača. Vzhľadom na to, že sme doteraz ešte neopísali zobrazovací kanál, musíme opis trochu zjednodušiť.

Predpokladajme preto, že sa pozeráme na skúmaný objekt v smere opačnom ku kladnému smeru osi *z* (ktorú si môžeme predstaviť ako vodorovnú a smerujúcu sprava doľava (obr.A), alebo ako zvislú a orientovanú nahor (obr.B) a do priemetne kolmej k osi *z* premietame ortogonálne (ortoprojekcia).

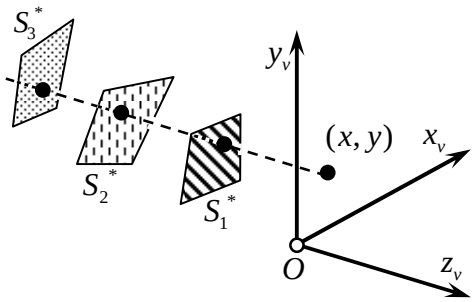


Obr.A



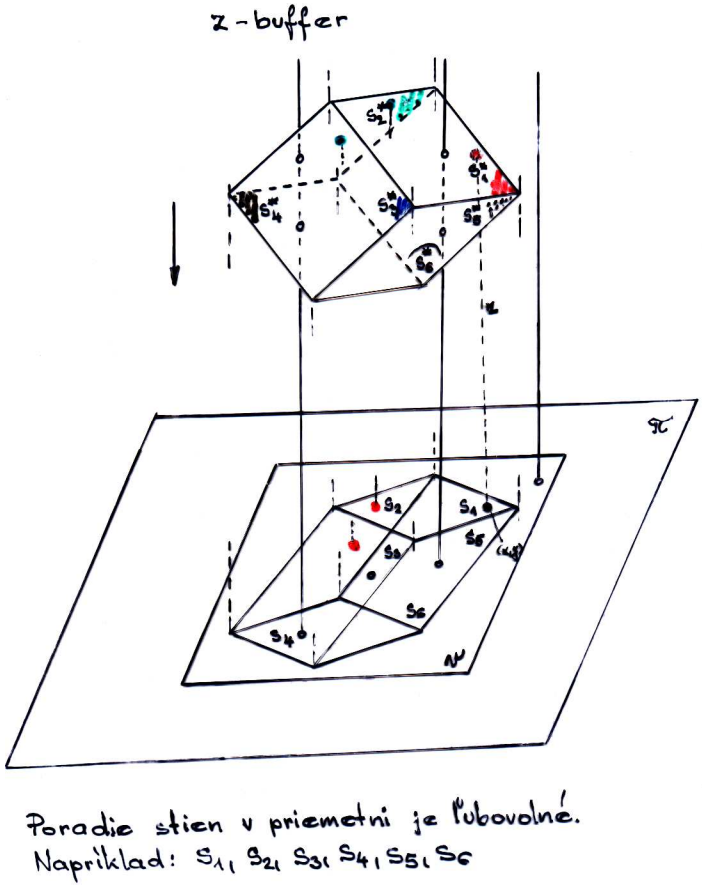
Obr.B

Teda bodu $X^*=[x, y, z]\in E^3$ priradíme v priemetni – na obrazovke bod $X=[x, y]$, pričom (x, y) sú obrazovkové resp. pixlové súradnice.



Podstata tohto algoritmu (obr.C) spočíva v nasledovnom postupe:

- priestorový objekt so stenami S_i^* odlíšenými od seba farebne alebo úrovňou intenzity tej istej farby premietneme do rovinných mnohouholníkov S_i , ktoré by mali reprezentovať obraz priestorového objektu na obrazovke počítača. Aby tento obraz bol čitateľný, musíme priemety S_i stien previesť do rastrovej formy a ich pixle zafarbiť resp. vysvietiť tou farbou, akú má stena telesa, ktorá je v nich viditeľná.



Obr. C

- Na obrazovke počítača vytvoríme *min-max box M* ohraničujúci množinu $\bigcup_i S_i$, prípadne za *M* možno zvoliť ľubovoľné okno obsahujúce túto množinu, vrátane celej obrazovky.
- Na množine *M* definujeme dva zásobníky - buffre:
  a) ***z*-buffer** $ZB : M \to R$; $(x, y) \to ZB(x, y)$, pričom v rámci inicializácie položíme $ZB(x, y) = -\infty$, pre všetky $(x, y) \in M$, kde - ∞ je ľubovoľné dostatočne malé záporné číslo, ktoré nemožno v rámci výpočtov dosiahnuť
  b) **refresh buffer** *RFB*: $M \to Z$; $(x, y) \to RFB(x, y)$, ktorý na začiatku inicializujeme tak, že pre každé $(x, y) \in M$, $RFB(x, y) = f_P$, kde f_P je číslo farby pozadia

  Hodnoty v oboch zásobníkoch sa počas behu algoritmu menia.
- Utvoríme si ľubovoľné poradie plôšok S_i (čiže priemetov stien) a postupne ich spracúvame (pre jednoduchosť zvoľme poradie $S_1, S_2, S_3, S_4, S_5, S_6$).

1. Spracúvame plochu S_1:
   a) ak nie je rozložená do rastra, urobíme tak, a označíme ju RS_1
   b) určíme rovnicu $A_1x + B_1y + C_1z + D_1 = 0$ roviny obsahujúcej stenu S_1* a upravíme ju do tvaru: $z = -\frac{A_1}{C_1}x - \frac{B_1}{C_1}y - \frac{D_1}{C_1}$ (*)

      (ak by C_1=0, tak rovina by bola rovnobežná s pozorovacím smerom a teda zadná – neviditeľná a mohli by sme ju zo zoznamu vylúčiť)
   c) pre každý pixel $(x, y) \in RS_1$ vypočítame *z* podľa vzťahu (*) t.j. položíme $z = -(A_1x + B_1y + D_1)/C_1$ (*z* sa nazýva hĺbka pixla (x, y), presnejšie $h = |z|$)
   d) Ak je takto vypočítané $z > ZB(x, y)$, tak položíme :

      $ZB(x, y) := z \quad \wedge \quad RFB(x, y)$: =číslo farby bodu $(x, y, z) \in S_1$*.

      Ak $z \le ZB(x, y)$, tak neurobíme v *ZB* a *RFB* žiadnu zmenu.
   e) Pri výpočte hodnoty *z* nie je vždy potrebné používať vzorec (*) (v ktorom sa delí), lebo ak poznáme hĺbku *z* pixla (x, y), tak pre hĺbku z' pixla $(x+1, y)$ platí $z' = -(A_1(x+1) + B_1y + D_1)/C_1$ t.j. $z' = z - A_1/C_1$.

      Tento vzorec zjednodušuje výpočet hĺbok pixlov jednej skenovacej priamky.

Ďalej, ak (x_1, y_1) sú celočíselné súradnice dolného vrcholu „ľavej“ strany h_1 plôšky S_1 a (x_1, y_1, z_1) je príslušný vrchol steny S_1*, tak z_1 je hĺbka pixla (x_1, y_1). Potom pre hĺbku z_1' v pixli $(x_1 + \frac{1}{m}, y_1 + 1)$, čo je priesečník úsečky h_1 s nasledovnou skenovacou priamkou $y = y_1 + 1$ platí:

$$z_1' = -\left[A_1(x_1 + \frac{1}{m}) + B_1(y_1 + 1) + D_1 \right]/C_1 = -\left[A_1x_1 + B_1y_1 + D_1\right]/C_1 - \left[A_1/m + B_1\right]/C_1 \text{ t.j.}$$

$z_1' = z_1 - K$, kde $K = (A_1/m + B_1)/C_1$, čo je konštanta pre celú úsečku h_1.

Podľa tohto vzťahu možno vypočítať hĺbky vo všetkých priesečníkoch úsečky h_1 so skenovacími priamkami (ak existujú) a podľa predchádzajúceho aj v jednotlivých pixloch týchto skenovacích priamok.

Ak toto urobíme pre všetky pixle plôšky S_1 prejdeme k plôške S_2 , na ktorej prevedieme tie isté procedúry .....atď. až po S_6. Po spracovaní všetkých týchto plôšok obsahuje *z*-buffer hĺbky pixlov

k plôškam, ktoré sú v nich viditeľné a refresh buffer obsahuje čísla farieb (odtieňov) bodov telesa, ktoré sa do pixlov premietajú a majú najväčšie *z*-ové súradnice, t.j. sú najbližšie k pozorovateľovi, čo po vysvietení dáva obraz viditeľných častí telesa.

Metóda triedenia do hĺbky- Maliarov algoritmus

Algoritmus používa objektovo i obrazovo orientované operácie. Je založený na nasledujúcich dvoch funkciách:

1. Roztriedenie všetkých stien vytvárajúcich povrchy telies podľa klesajúcej hĺbky
2. Plochy sa spracúvajú – zoskenovávajú (farbia) v poradí od najvzdialenejšej po najbližšiu k pozorovateľovi.

Triediace operácie sa prevádzajú v obrazovom i v objektovom priestore, skenovanie (farbenie) len v obrazovom.

V súlade so známou technikou maliara, najskôr sa zavedú do zásobníka (frame buffer) hodnoty intenzity (farebných odtieňov) najvzdialenejšej steny od pozorovacej roviny. Pri spracúvaní ďalších stien (bližších k pozorovateľovi) zapíšeme ich intenzity do frame buffra aj „cez" intenzity predtým spracúvaných plôch – prepíšeme ich. Výsledný farebný mnohouholník v zásobníku (frame buffer) sa vytvára v niekoľkých krokoch.

Ak predpokladáme, že sa pozeráme v smere osi - z_v a vonkajšok steny je oproti pozícii pozorovateľa, tak steny sa pre prvý prechod usporiadajú podľa minimálnej z-ovej súradnice bodov steny (z-ovej súradnice jej najvzdialenejšieho vrcholu). Stena S s najväčšou hĺbkou z_S (najmenšou z-ovou súradnicou) sa potom porovnáva so stenami S' vytvoreného zoznamu, aby sa zistili ich možné hĺbkové prekrytia so stenou S (hĺbkové prekrytie nastane ak min $z_{S'} <$ max z_S).

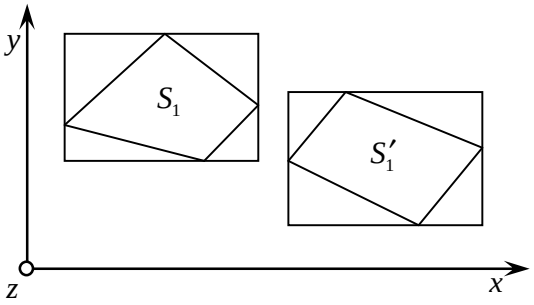
Ak stena S nemá hĺbkové prekrytie so žiadnou stenou S', tak je najvzdialenejšou od pozorovateľa, zoskenuje sa (zafarbí sa) a vyradí sa zo zoznamu. Tento proces sa potom opakuje pre nasledujúcu stenu v zozname, ...atď. Pokiaľ sa v tomto procese žiadne hĺbkové prekrytia nevyskytnú, všetky steny sa zoskenujú v pôvodnom poradí určenom zoznamom.

Ak sa však v niektorom bode zoznamu hĺbkové prekrytie vyskytne, musíme urobiť niektoré ďalšie vyšetrenia – porovnania, aby sme zistili, či sa nemusia niektoré steny v zozname vymeniť (preusporiadať zoznam).

Z tohto dôvodu, pre každú stenu S', ktorá sa hĺbkovo prekrýva so stenou S (nachádzajúcou sa v tomto okamihu v zozname najvyššie) urobíme nižšie uvedené testy T1 – T4. Ak niektorý z týchto testov je kladný, nemusíme v zozname vymeniť stenu S so stenou S'.

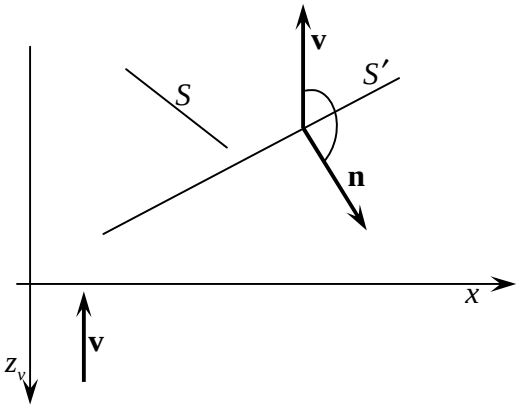
Testy sú uvedené v poradí stúpajúcej obtiažnosti. Každý z nich je určený pre riešenie situácií, v ktorých boli predchádzajúce testy neúspešné.

$T1$ Ohraničujúce obdĺžniky (min – max boxy) priemetov S_1, S_1' stien S, S' do roviny Oxy sa nepretínajú (S, S' sa nachádzajú v hranoloch nad týmito min – max boxami a preto žiadna z nich nemôže zakrývať zvyšnú pri pohľade rovnobežnom s osou z).



Realizácia :
Aby tento test bol splnený stačí, ak boxy nemajú prekrytie v smere osi x [max $x_S <$ min $x_{S'}$ alebo max $x_{S'} <$ min x_S], alebo v smere osi y [max $y_S <$ min $y_{S'}$ alebo max $y_{S'} <$ min y_S].

$T2$ Stena S je celá za rovinou obsahujúcou stenu S' (na jej vnútornej strane) vzhľadom na pozíciu pozorovateľa (reprezentovanú vektorom **v**). Ináč: Rovina α obsahujúca S' musí pozorovateľovi stenu S zakrývať, čiže pozorovateľ musí byť na prednej strane S'.(**n** musí smerovať proti **v**).

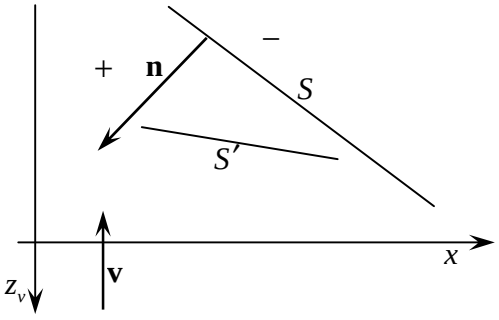


Realizácia:
Normálový vektor **n** roviny S' musí byť zvolený tak, aby:

$$\mathbf{n}.\mathbf{v} < 0 \Leftrightarrow C.v_z < 0 \Leftrightarrow C > 0 \text{ (lebo } v_z < 0).$$

Ak si teda nájdeme ľubovoľnú rovnicu $Ax + By + Cz + D = 0$ roviny $\alpha \supset S'$, v ktorej koeficient C nie je > 0, treba ju vynásobiť číslom -1 (ak $C > 0$, tak nie). Test T2 bude splnený, ak všetky čísla, ktoré dostaneme postupným dosádzaním vrcholov steny S do ľavej strany takto získanej rovnice steny S', budú záporné.

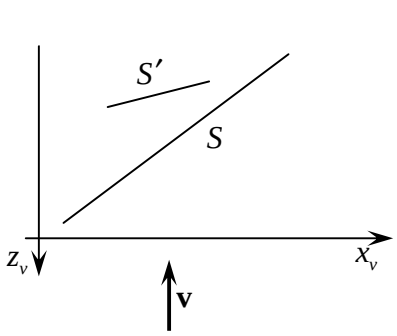
$T3$ Prekrývajúca stena S' je celá pred rovinou α obsahujúcou stenu S vzhľadom na pozíciu pozorovateľa, ale S nie je celá za S' (čiže T2 nie je splnený).



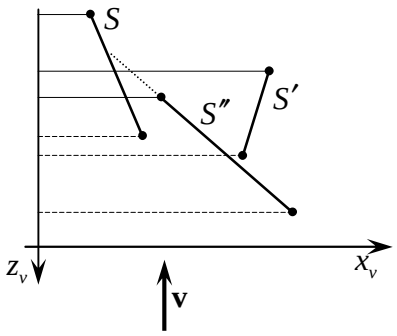
Realizácia: tohto testu je analogická ako u T2 len treba zameniť úlohu stien *S, S´* a pre vrcholy *S´* (všetky) požadovať splnenie podmienky : $Ax + By + Cz + D > 0$.

$\boxed{T4}$ Priemety S_1, $S_1´$ stien *S, S´* do pozorovacej roviny *Oxy* sa nepretínajú.
Tento test skúsime iba vtedy, keď nám všetky predchádzajúce testy T1, T2, T3 zlyhali. Využívame pritom metódy rovinnej analytickej geometrie na hľadanie prienikov medzi hranami plôch S_1, $S_1´$, zisťovanie či vrcholy jednej z plôch ležia vo vnútri druhej plochy (hlavne ak je táto konvexná).

Ako sme už povedali, testy T1- T4 uskutočňujeme v uvedenom poradí a k nasledujúcej stene *S´´* hĺbkovo sa prekrývajúcej so stenou *S* prejdeme hneď, ako náhle zistíme, že niektorý z týchto testov je splnený. Ak všetky steny hĺbkovo prekrývajúce *S* spĺňajú aspoň jeden z týchto testov, žiadna z nich nie je za plochou *S*. Žiadne preusporiadanie nie je potrebné a plochu *S* môžeme zoskenovať (vyfarbiť).



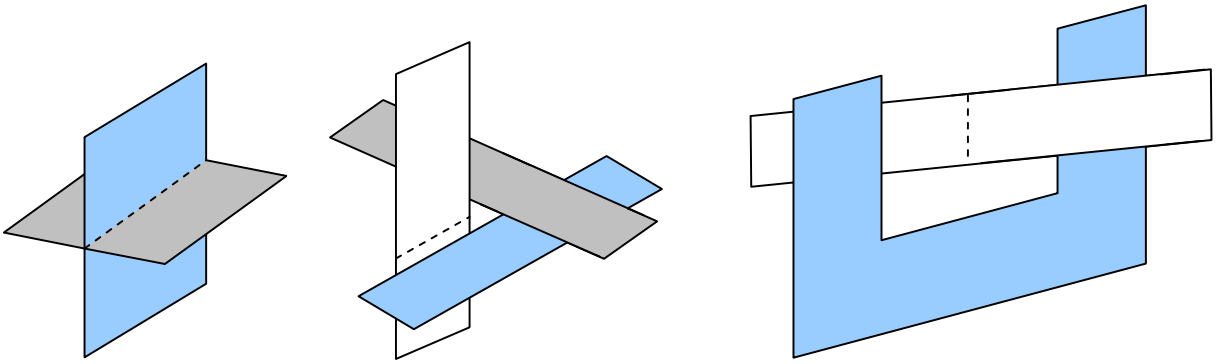
Obr.a)



Obr.b)

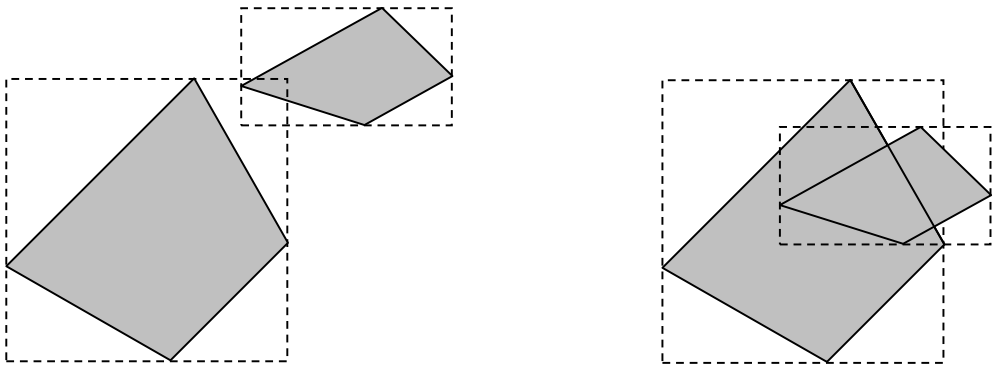
Ak všetky testy T1- T4 steny *S*, s hĺbkovo ju prekrývajúcou stenou *S´* zlyhajú, je potrebné vymeniť steny *S, S´* v zozname stien. Príklad dvoch stien *S, S´*, ktoré treba vymeniť je na obr.a) vľavo. Plocha *S* má síce väčšiu hĺbku, ale zakrýva stenu *S´* (žiadny z testov nie je splnený). Ani po tejto výmene však ešte nevieme s určitosťou povedať, že sme našli najvzdialenejšiu stenu od pozorovacej roviny. Obr.b) demonštruje situáciu, keď do usporiadaného zoznamu stien boli zaradené tri steny *S, S´, S´´*. Podľa nášho návodu (opisu) boli najskôr vymenené steny *S* a *S´´*, čím sme dostali usporiadanie *S´´, S´, S*. Avšak vzhľadom na to, že *S´´* hĺbkovo prekrýva časť *S´*, musíme ešte vymeniť *S´´* a *S´*, aby sme tak dostali ich konečné korektné usporiadanie *S´, S´´, S*. Z tohto dôvodu musíme pri každej výmene poradia dvoch stien zopakovať testovací proces pre každú stenu, ktorá bola vymenená (smerom nahor).

V načrtnutom algoritme sa možno dostať do nekonečnej slučky zloženej z dvoch alebo viacerých plôch, v ktorej alternatívne jedna zakrýva druhú.



Tento problém sa obyčajne rieši rozdelením jednej alebo viacerých plôch na časti.

Pozn1. Pri tvorbe východzieho zoznamu stien telies sa stáva, že viaceré majú rovnakú minimálnu *z*-ovú súradnicu (majú spoločný vrchol). O ich poradí v zozname môžeme niekedy rozhodnúť podľa *z*-ových súradníc ťažísk týchto stien.
Pozn2. Testy T1 a T4 nie sú totožné. Ako vidno na nasledujúcich obrázkoch, priemety stien sa môžu, ale nemusia pretínať hoci ich min – max boxy sa určite pretínajú.



a) b)

Poznámka Algoritmus nedokáže vyriešiť všetky situácie korektne. Vraj sa pri veľkom množstve objektov v scéne nejaká nepresnosť stratí. Niektorí praktici v takýchto prípadoch dokonca používajú len prvé dva testy.

Warnockov algoritmus delenia okna

Cieľ: Určiť viditeľnosť priestorových objektov reprezentovaných rovinnými stenami premietajúcimi sa do mnohouholníkových plôch, ktoré sa čiastočne, alebo úplne prekrývajú s pevne zvoleným (priesvitným) oknom rastrového charakteru nachádzajúcim sa v priemetni. Rastrové okno má slúžiť ako pomôcka pri určovaní viditeľnosti.

- Algoritmus je založený na
  - testovaní vzájomnej polohy priemetov priestorových plôch a okna
  - testovaní polohy priestorových stien vzhľadom na okno (hlavne ich „vzdialeností“ od okna – hĺbky) resp. ich vzájomnej polohy (prekrývanie, zakrývanie) pri pozorovaní z okna
- Zhruba povedané, cieľom algoritmu je zistiť, ako sa má vyplniť resp. vyfarbiť dané okno tak, aby verne odrážalo prostredie, ktoré sa do neho premieta. Ak sa tak nedá urobiť jednoduchými prostriedkami, použije sa známa metóda „rozdeľuj a panuj“ (divide et impera) t.j. okno sa rozdelí priamkami rovnobežnými so súradnicovými osami na štyri časti, na ktoré sa algoritmus opätovne aplikuje. Ak sa v tejto postupnosti delení okna niektorá časť degeneruje na riadok resp. stĺpec pixlov, pokračuje sa jednorozmerným delením.
- Algoritmus sa ukončí vtedy, keď každé „malé“ okno je priemetom časti jedinej viditeľnej plochy (vyfarbí sa jej farbou), alebo sa do neho nepremietne žiadna z plôch (vyfarbí sa farbou pozadia), alebo je pixlom (tento získa farbu tej plochy, ktorá je v tomto bode najbližšia k pozorovateľovi – *z*-buffer).
- Pri rozhodovaní o vyfarbení okna treba teda zohľadniť nasledujúce prípady:
  1. Do okna sa nepremieta žiadna časť scény - vyfarbí sa farbou pozadia.
  2. Do okna zasahuje jediná plocha – táto sa v okienku vyfarbí svojou farbou, zvyšok okna – farbou pozadia.
  3. Do okna sa síce premieta viac plôch (zasahuje do neho viac priemetov), ale plocha, ktorá je najbližšia k pozorovateľovi ho úplne zakrýva – okno sa celé vyplní farbou tejto plochy.
  4. Okno obsahuje komplikovanejšiu časť scény ako je opísané v prípadoch 1., 2.a 3. – okno sa rozdelí na štyri časti a algoritmus sa opakuje pre každú z nich.
- Pri bežnej – praktickej realizácii algoritmu sa nezvyknú testovať všetky vyššie uvedené podmienky samostatne; samostatnou zostáva podmienka 3., ktorá sa nezriedka označuje ako K1 a podmienky 1. a 2. sa zlučujú do jednej, ktorá sa označuje ako K2. Realizácia 4. prípadu sa považuje za viac-menej samozrejmú.
- Na tomto princípe je založené aj riešenie ilustračného príkladu kde vo Warnockovom algoritme sa namiesto troch testov používajú nasledujúce dva:

  K1 : K pozorovateľovi najbližšie ležiaca plocha zakrýva celé okno.

  K2 : Existuje jedna alebo žiadna plocha zasahujúca do okna.

Ak je splnená jedna z týchto podmienok možno o viditeľnosti v okne bezprostredne rozhodnúť.

Ak žiadna z nich nie je splnená, tak okno sa rozdelí a každá z jeho štyroch častí sa opäť testuje na splnenie podmienok K1, K2.

Ak sa plocha degeneruje na jeden riadok (stĺpec), tak sa pokračuje jednorozmerným delením. Najneskôr pri redukcii rastrového okna na jeden rastrový bod sa delenie ukončí a o viditeľnosti sa rozhodne pomocou z-buffra alebo metódou malého okna.

Príklad

Pre scénu pozostávajúcu z dvoch nepriehľadných plôch (oblastí):

a) trojuholník ***A*** = {*A*1=[18.5, 12.5, 0], *A*2=[22.5, 16.5, 0], *A*3=[14.5, 22.5, 14]} zafarbený červenou farbou
b) štvoruholník ***B*** = {*B*1=[12.5, 14.5, 6], *B*2=[20.5, 14.5, 6], *B*3=[20.5, 20.5, 6], *B*4=[12.5, 20.5, 6]} zafarbený modrou farbou.

Určite viditeľnosť pomocou Warnockovho deliaceho algoritmu.

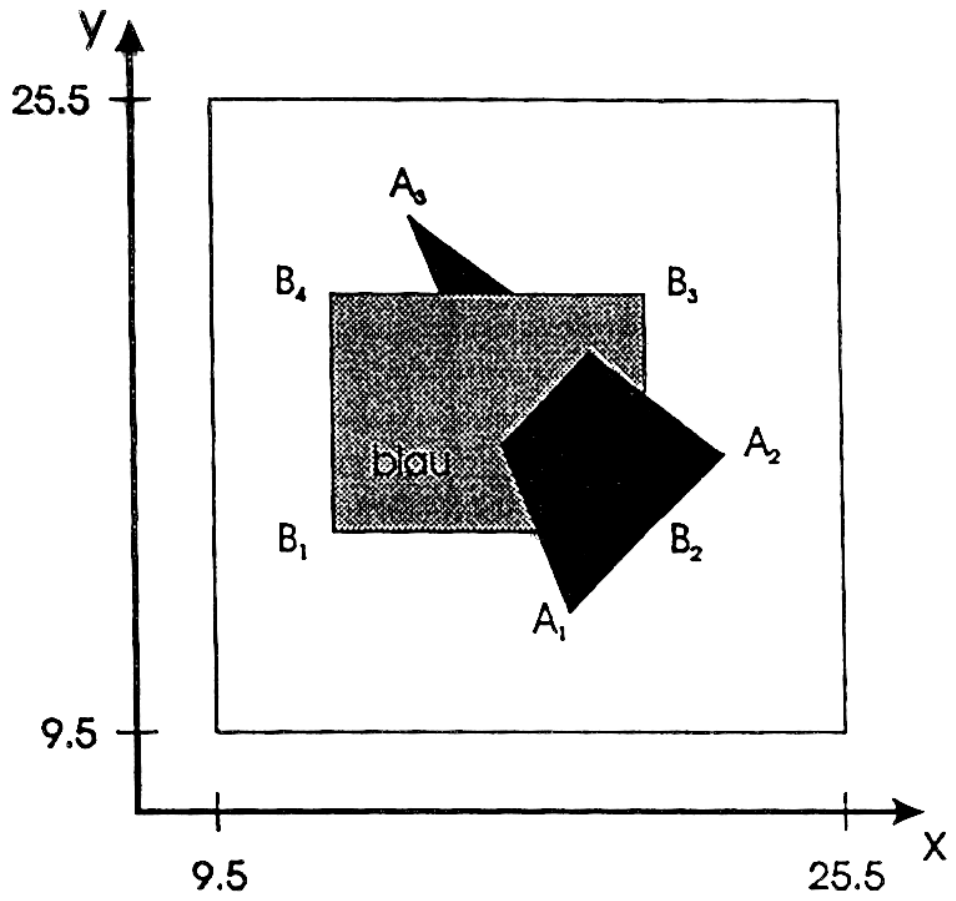
Riešenie:

Priemetňou je rovina *Oxy* ($z = 0$), 3D-systém je ľavotočivý, o pozorovateľovi sa predpokladá umiestnenie v nevlastnom bode so súradnicou $z = -\infty$ ($\uparrow$) a východzie rastrové okno je $\boldsymbol{O}$ = {*C*1=[9.5, 9.5], *C*2=[25.5, 9.5], *C*3=[25.5, 25.5], *C*4=[9.5, 25.5]}, pričom sa predpokladá, že stredy pixlov majú celočíselné súradnice, dĺžku strany 1 ($\Rightarrow$ že východzia rastrová plocha obsahuje $16 \times 16 = 2^4 \times 2^4$ pixlov $\Rightarrow n = 4$) a farba pozadia sa predpokladá biela.

Aby sme vedeli rozhodnúť, či objektová plocha, ktorej priemet okno (rastrovú plochu) úplne pokrýva je k pozorovateľovi najbližšia, vypočítame *z*-ové súradnice všetkých objektových plôch v rohových bodoch rastrovej plochy (príslušného okna); len takáto objektová plocha má na všetkých štyroch rohoch najmenšiu hodnotu, určite zakrýva všetky ostatné objektové plochy.

V rastrových plochách (oknách), u ktorých je viditeľnosť vyriešená, treba vykresliť viditeľné objektové plochy:

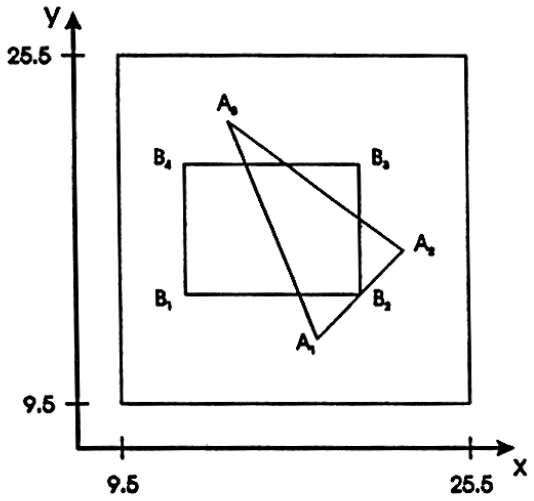
1) Ak stred pixla leží vo vnútri alebo na hranici objektovej plochy, zafarbíme ho.
2) Ak sa Warnockov deliaci algoritmus ukončí tým, že rozdeľované plochy dosiahli veľkosť pixla, zostanú nám ešte viaceré potenciálne viditeľné fragmenty o farbe ktorých nie je rozhodnuté. Tieto pixle by sa mali teoreticky ešte ďalej deliť, čo však nie je možné.
   O viditeľnosti týchto „prebytočných“ pixlov však môžeme rozhodnúť pomocou algoritmu hĺbky (z-buffer).
   Graficky vyznačte výsledok. Ak *A* a *B* majú v nejakom bode rovnakú z-ovú súradnicu, vyznačte (vykreslite) *A*.
3) Určite exaktný výsledok matematickým výpočtom.



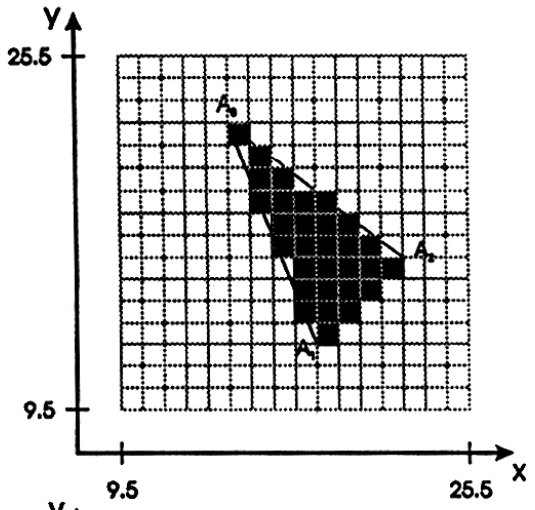
Lahko sa možno presvedčiť, že rovina trojuholníka má rovnicu $Z = -X+Y+6$ a rovina 4-uholníka : $Z = 6$.
Ich priesečnica je potom implicitne daná dvojicou rovníc: $Y = X$, $Z = 6$. Okrem toho $A_1A_3 \cap B = [16.78, 16.78, 6]$ a $A_2A_3 \cap B = [19.07, 19.07, 6]$

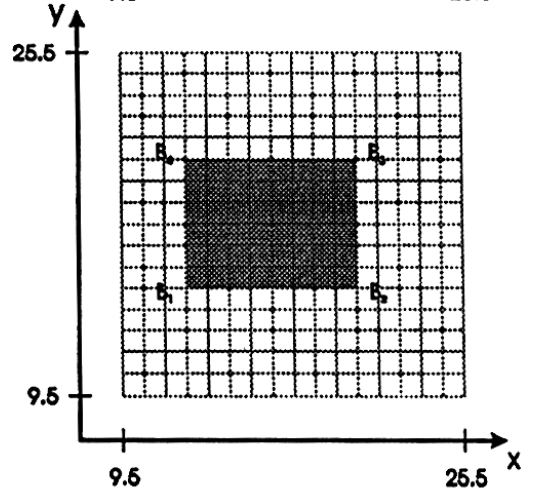
Priamka $Y = X \wedge Z = 6$ oddeľuje viditeľnú dolnú časť trojuholníka od zakrytej. Na obrázku je grafická prezentácia situácie.

b) Na nasledujúcom obrázku sú zobrazené obe plochy A a B.

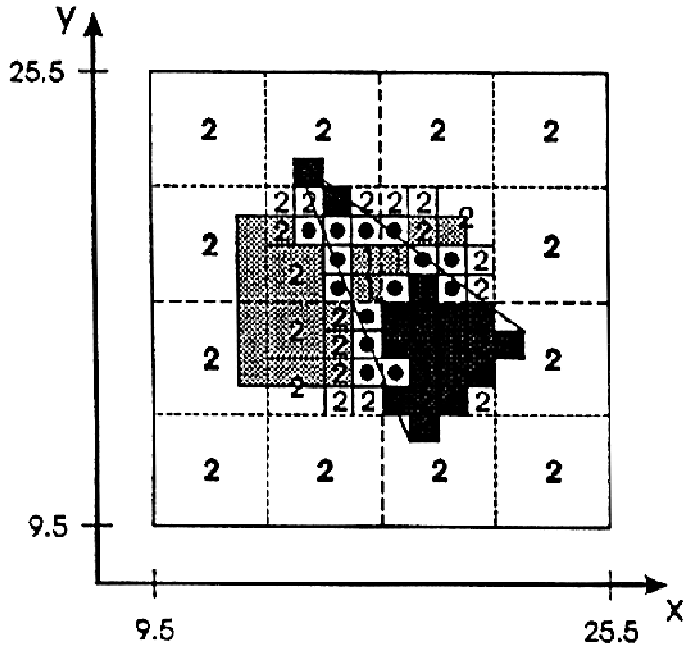


Ak štvoruholník a trojuholník prevedieme do rastrovej formy (každý zvlášť) dostaneme nasledujúce kresby



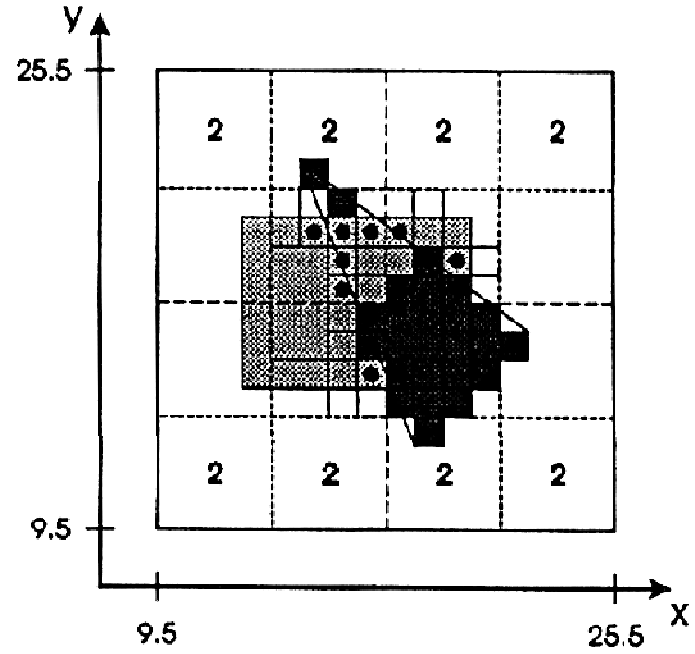


b1) Warnockov deliaci algoritmus a následné vyfarbenie pixlov patriacich viditeľným plochám dáva nasledujúci výsledok



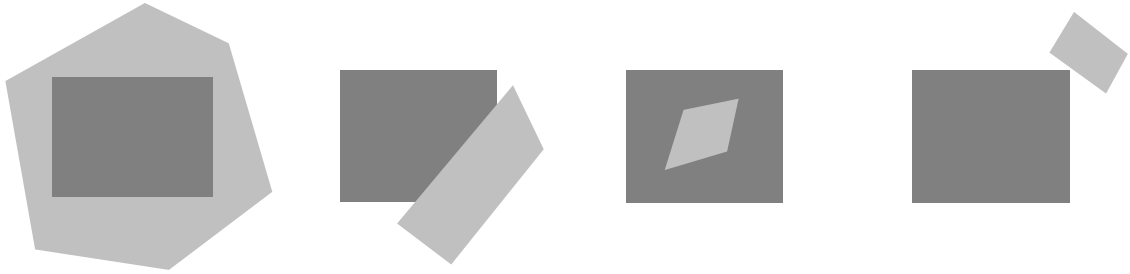
Čísla na obraze označujú prípady, v ktorých sa rozhodlo podľa testov; pixle označené tmavým krúžkom predstavujú ešte ďalšie potenciálne viditeľné plôšky - pixle (nespĺňajú ani jednu podmienku testu)
Mali by sa vlastne ďalej deliť.

b2) Použitie z-buffra pre všetky krúžkom označené pixle dáva nasledujúci obrázok:



V medzinárodných učebniciach (napr. Hearn – Baker: Computer Graphics – International edition, 1997) sa možno stretnúť aj s podrobnejším opisom tohto algoritmu pod názvom:
AREA-SUBDIVISION METHOD, v tomto algoritme sa vychádza z testov, ktoré rýchlo identifikujú, že okno je časťou jedinej plochy resp. že situácia v okne je príliš zložitá na jednoduchú a rýchlu analýzu a preto ho treba rozdeliť (na okno 1024x1024 treba max.10 delení).

- V algoritme sa plochy (priemety) podľa svojej polohy vzhľadom na okno delia na **obklopujúce, prekrývajúce, vnútorné a vonkajšie**:



obklopujúce prekrývajúce vnútorné vonkajšie

Na týchto pojmoch sú založené testy pre určovanie viditeľnosti plôch vzhľadom na okno. Ich formulácia je nasledovná:

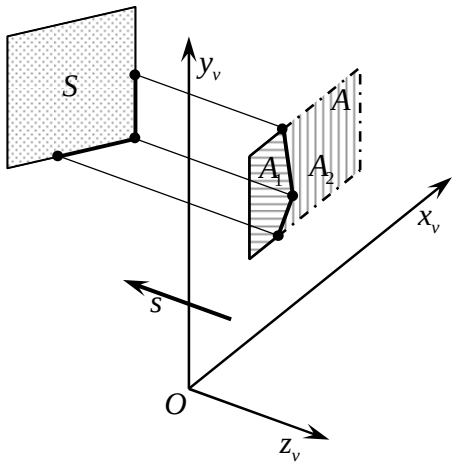
1. Všetky plochy (priemety) sú vonkajšími vzhľadom na okno.
2. S oknom inciduje len jediná vnútorná alebo prekrývajúca alebo obklopujúca plocha.
3. Existuje okno obklopujúca plocha, ktorá zatemňuje (zacláňa) všetky ďalšie plochy medzi hranicami okna (orezané časti) .

- Pri realizácii testov 1. a 2. sa využívajú min-max boxy. Pre detailné zistenie, či daná plocha je okno obklopujúca, prekrývajúca resp. vonkajšia, sú však potrebné aj jemnejšie testy. Ak sme zistili, že určitá plocha vyhovuje testu 2., prenesú sa intenzity (farby) jej pixlov do príslušnej oblasti pamäte obrazu (frame buffer) a prepíšu sa aj predtým zapísané farby pozadia.
- Najzložitejšie je overenie testu 3. Obyčajne sa využíva jedna z nasledujúcich možností:
  1. Plochy sa usporiadajú podľa ich minimálnej vzdialenosti (hĺbky) od roviny pozorovateľa. Pre každú obklopujúcu plochu sa potom vypočíta maximálna hĺbka od uvažovaného okna. Ak maximálna hĺbka jednej z týchto obklopujúcich plôch je bližšia k rovine pozorovateľa ako minimálna hĺbka všetkých ďalších plôch k oknu, je test 3. splnený.
  2. Ďalšia možnosť pre vykonanie tohto testu, nevyžadujúca hĺbkové triedenie plôch, používa rovnice rovín na výpočet hodnôt hĺbky v štyroch rohoch okna pre všetky obklopujúce, prekrývajúce a vnútorné plochy. Ak vypočítané hĺbky pre jednu z obklopujúcich plôch sú menšie ako vypočítané hĺbky pre všetky ostatné plochy, tak je test 3 splnený. Potom sa okno vyplní (zafarbí) hodnotami intenzity (farbou) tejto obklopujúcej plochy.
- Niekedy zlyhajú obe uvedené metódy – vtedy sa okno rozdelí.

Týmto sme opísali podstatu Warnockovho prerozdeľovacieho algoritmu. Ako sme videli, algoritmus rekurzívne prerozdeľuje obrazovku na štyri pravouhlé oblasti bez ohľadu na skutočný tvar mnohouholníkov a zavádza tak primnoho prebytočných vertikálnych a horizontálnych hrán. Weiler a Atherton si všimli tento fakt a pokúsili sa počet prebytočných hrán znížiť. Ich

algoritmus síce rekurzívne prerozdeľuje okno, ale namiesto pravouholníkov využíva všeobecnejšie štvoruholníky, ktorých niektoré strany sú hranice (ich priemety) aktuálnych mnohouholníkov, presnejšie okná sa rozdeľujú pozdĺž hraníc plochy miesto delenia na polovicu. Ak teda plochy už boli usporiadané podľa minimálnej hĺbky, môžeme plochu s najmenšou hodnotou hĺbky (najbližšou k pozorovateľovi) použiť na rozdelenie daného okna.

Obrázok ilustruje túto metódu prerozdeľovania okien:



Priemet hranice plochy *S* bol použitý na prerozdelenie pôvodného okna *A* na podokná A_1 , A_2. Plocha *S* je potom obklopujúcou pre A_1 a viditeľnostné testy 2 a 3 možno použiť na zistenie, či je potrebné ďalšie delenie alebo nie. Vo všeobecnosti použitie tohto prístupu vyžaduje menej delení, ale viac výpočtov na rozdelenie okien a na analýzu vzájomných vzťahov, presnejšie na určenie vzájomnej polohy plôch (priemetov stien) s hranicami okien.

Weiler – Athertonov algoritmus

• Algoritmus vychádza zo zoznamu $L = \{P_1, P_2,...,P_n\}$ vstupných mnohouholníkov približne usporiadaných podľa hĺbky tak, že najbližšie mnohouholníky sú na začiatku a najvzdialenejšie na konci zoznamu. Tento triediaci krok sa urobí len raz na začiatku výpočtov, nemá záväzný charakter, ale ovplyvňuje efektívnosť algoritmu.

• Vyberme zo zoznamu *L* prvý mnohouholník P_1 a použime ho ako rozdeľovací mnohouholník pre mnohouholníky množiny $L \setminus \{P_1\}$ do dvoch nových zoznamov:

$$I = \left\{P_1^I, P_2^I, ..., P_m^I\right\} \text{ a } O = \left\{P_1^O, P_2^O, ..., P_M^O\right\}; \quad m, M \le n$$

kde do *I* patria tie mnohouholníky alebo ich časti z $L \setminus \{P_1\}$, ktoré z hľadiska pozorovateľa padnú do vnútra P_1 a do *O* tie, ktoré padnú do doplnku mnohouholníka P_1 v rovine $\pi_1 \supset P_1$.

• Algoritmus začína vyšetrovaním (prieskumom) zoznamu *I* a vyraďovaním všetkých mnohouholníkov (častí), ktoré sú umiestnené v *L* za P_1, pretože pozorovateľovi sú zakryté mnohouholníkom P_1. Ak zvyšný zoznam I' je prázdny (orezávací mnohouholník zakrýva všetky mnohouholníky z *I*), tak sa orezávací mnohouholník vykreslí a východzí zoznam *L* sa nahradí vonkajším zoznamom *O*, s ktorým sa prevedie rovnaká deliaca procedúra ako s *L*.

• Ak však zvyšný zoznam I' obsahuje aspoň jeden mnohouholník, čiže aspoň jeden mnohouholník nám padne pred orezávací mnohouholník P_1, tak to znamená, že došlo k chybe vo

východzom predbežnom triedení mnohouholníkov. Napravíme ju tak, že skôr, než zrealizujeme predchádzajúci krok, podrobíme aj I' rovnakej procedúre ako východzí zoznam L.

• Aby nedošlo k cyklickému prekrývaniu sa mnohouholníkov, vyvolávajúcemu nekonečnú rekurziu, v algoritme zavádzame a počas celého procesu udržiavame množinu S, do ktorej sa zaradí názov (identifikátor) každého mnohouholníka vo chvíli, keď bol vybraný ako orezávací a vyradí sa z nej ihneď po jeho spracovaní (vykreslení alebo odstránení). Vloženie sa prirodzene robí iba vtedy, ak mnohouholník P už nie je v S (prečo?).

• Podrobnosti sú v nasledujúcom algoritme.

WeilerAtherton (*L*)

```
 P = prvá položka v L; S = {}
 if P ∈ S then vykresli P; return; endif
 vlož P do S;
 I= Clip( L,P);
 O = Clip(L, P̄);     || P̄ = complement P
 for každý polygón Q ∈ I ;
       if Q je za P then
          vyber Q z I;
           if Q ∈ S then vyber Q z S; endif
        endif
 endfor
 if I = {} then
     vykresli P;
      odstráň P z S;
 else
      WeilerAtherton (I);
 endif
 WeilerAtherton(O);
end
```

Poznámka: Rekurzívny algoritmus sa volá s východzím utriedeným zoznamom L vstupných mnohouholníkov na “hornej” úrovni, po inicializácii množiny S na {}.

Maliarov algoritmus s BSP-stromom

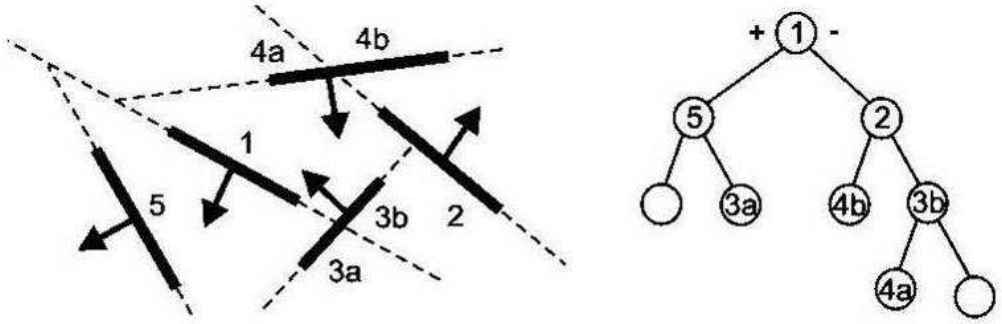
Zaujímavým variantom maliarovho algoritmu je jeho implementácia pomocou stromovej štruktúry BSP (*Binary Space Partitioning tree*).

- Cieľ metódy: uchovať opis priestorových objektov v pohľadovo nezávislom tvare, ktorý pre ľubovoľnú pozíciu pozorovateľa poskytuje údaje o správnom vykresľovaní objektov.
- Strom BSP sa vytvára jednorázovo v rámci inicializácie a potom sa opakovane využíva pre rôzne pohľady.
- Je vhodný pre statické scény, v ktorých sa **nemenia** pozície a veľkosti objektov. V počítačovej grafike sa tento binárny strom využíva hlavne pri urýchľovaní algoritmov prostredníctvom hierarchického delenia priestoru na menšie časti. Každý jeho vnútorný uzol predstavuje rez rozdeľujúci pridelený priestor na dve časti.
- Pri konštrukcii algoritmu postupujeme zhora nadol a rekurzívne delíme trojrozmerný priestor (okno) obsahujúci zobrazované objekty.

Metóda je vhodná len pre objekty ohraničené **rovinnými plôškam**i, ktoré pred konštrukciou stromu zoskupíme do jednej množiny, povedzme *L*.

V nej si zvolíme jednu plôšku napr. s_1 a zostrojíme rez priestoru E^3, vrátane objektov, ktoré chceme zobraziť a ich plôšok - prvkov množiny $L \setminus \{ s_1\}$, rovinou α obsahujúcou plôšku s_1. Rovina α rozdelí priestor E^3 na dva podpriestory povedzme E^3_+ a E^3_-. Tieto plôšky, alebo ich časti, ktoré patria podpriestoru E^3_+ [E^3_-] vytvoria novú množinu plôšok, povedzme L^α_+ [L^α_-]. Miesto množiny plôšok *L* máme teraz dve množiny plôšok L^α_+ a L^α_-. Ak s každou z nich uskutočníme ten istý proces ako s *L* dostaneme štyri množiny plôšok....atď. Tento proces opakujeme ďalej a ďalej, avšak iba s tými množinami plôšok, ktoré obsahujú viac ako jednu položku (plôšku) a ukončíme ho ak žiadna taká množina neexistuje.

- Mnohí autori odporúčajú venovať výberu plôšok určujúcich „rezacie“ roviny zvýšenú pozornosť z hľadiska minimalizácie počtu delených plôšok t.j. počtu prvkov novo vytváraných množín.



Obrázek 11.5: Několik plošek (vlevo) a k nim vytvořený strom BSP (vpravo)

Tak napr. na obrázku 11.5 z učebnice Žára a kol. vidno 5 plôšok a jeden z možných stromov BSP, pri konštrukcii ktorého vznikli štyri fragmenty z plôšok 3 a 4 a autori upozorňujú, že keby sa priestor začal deliť rovinou obsahujúcou plôšku 3, mal by len dva fragmenty.

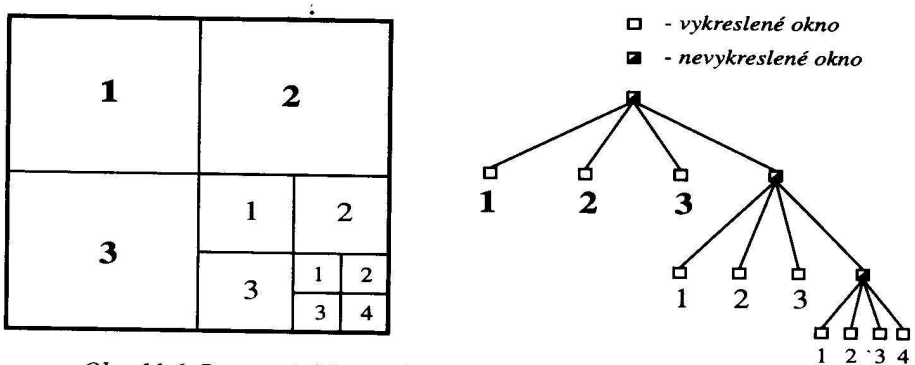
- Pri zobrazovaní objektov prechádzame strom BSP od koreňa a v každom uzle určujeme, ktorý polpriestor je k pozorovateľovi bližší a ktorý je vzdialenejší. Najprv vykresľujeme vzdialenejšiu časť priestoru, potom obsah aktuálneho uzla a nakoniec oblasť bližšiu k pozorovateľovi.

Postup:

```
ZobrazBSP (S) {
   pokud (S je list) {
      Vykresli_plochy (S.kořen);
      return;
   }
   pokud (pozorovatel je v předním poloprostoru řezu S) {
      ZobrazBSP (S.zadní_větev);
      Vykresli_plochy (S.kořen);
      ZobrazBSP (S.přední_větev);
   }
   jinak {
      ZobrazBSP (S.přední_větev);
      Vykresli_plochy (S.kořen);
      ZobrazBSP (S.zadní_větev);
   }
}
```

Komentár: Predpokladá sa, že dátová štruktúra S reprezentuje jeden uzol, ktorý v položke koreň obsahuje údaje o plôške, prípadne aj o viacerých plôškach, ktorými prechádza. Ďalej obsahuje smerníky na ľavý a pravý podstrom (položky predná vetva a zadná vetva). Aktualizácia stromu v prípade dynamických objektov je časovo náročná.

Poznámka 1. V predchádzajúcom opise sme sa zaoberali súvislosťou maliarovho algoritmu s binárnym stromom BSP. Teraz sa opäť vrátime k Warnockovmu algoritmu delenia okna. Vieme, že v tomto algoritme sa snažíme vyriešiť problém viditeľnosti v okne naraz. Ak to nie je možné, problém riešime rekurzívne v každom z menších okien, ktoré sme dostali rozdelením pôvodného na štyri časti. Takýmto spôsobom riešenia sa prirodzene vytvára tetrálny strom, ktorého listy reprezentujú tie okná, pri ktorých sme sa vedeli jednoducho presvedčiť, že budú zafarbené homogénne a to buď farbou pozadia, alebo farbou tej plôšky, ktorej priemet okno úplne zakrýva a je najbližšie k pozorovateľovi. Tento prístup ku konštrukcii algoritmu je načrtnutý v učebnici RUFE, na obr. 11.6.



Obr. 11.6 Postupné delenie okna a vytvorenie stromovej štruktúry

V tejto učebnici sa nachádza aj nasledujúci 2-dielny Warnockov algoritmus:

```
Procedure Warnock ( W, L, M, i ) ;
begin
1. if  i = n  then drawmin_z (L, M)
   else
   begin
2. for mnohouholník P ∈ L do
     begin
3.     k:= location (W, P);                    { zistíme polohu mnohouholníka k podoknu }
4.     if k = 0 then L := L - {P};             { mnohouholník P vymažeme zo zoznamu L }
5.     if k = 2 then M:= M + {P}; L:= L - {P}  { mnohouholník P pridáme do zoznamu M }
     end
6. if (number (L) + number (M)) = 0 then draw_box (W, background);     { podmienka 1 }
7. if number (L) = 1 and number (M) = 0 then begin                     { podmienka 2 }
                                                draw_box (W, background);
                                                draw_poly (P, col(P));
                                                end
8. if number (L) = 0 and number (M) = 1 then draw_box (W, col(P));     { podmienka 3}
   else begin
9.       L1:= L; M1:= M;
         Warnock ( W1, L1, M1, i+1);        { rekurzívne zavolanie funkcie pre 4 podokná}
         Warnock ( W2, L1, M1, i+1);
         Warnock ( W3, L1, M1, i+1);
         Warnock ( W4, L1, M1, i+1);
         end
end { návrat z rekurzie }
end.
```

Túto procedúru volá nasledujúci vlastný Warnockov

Algoritmus delenia okna

```
begin
0. Inicializujeme W := celé okno;
     zoznam L := { všetky steny scény };
     zoznam M := ∅;
     i := 1;
     Warnock ( W, L, M,  i );
end
```

Odstraňovanie zakrytých povrchov a BSP-stromy

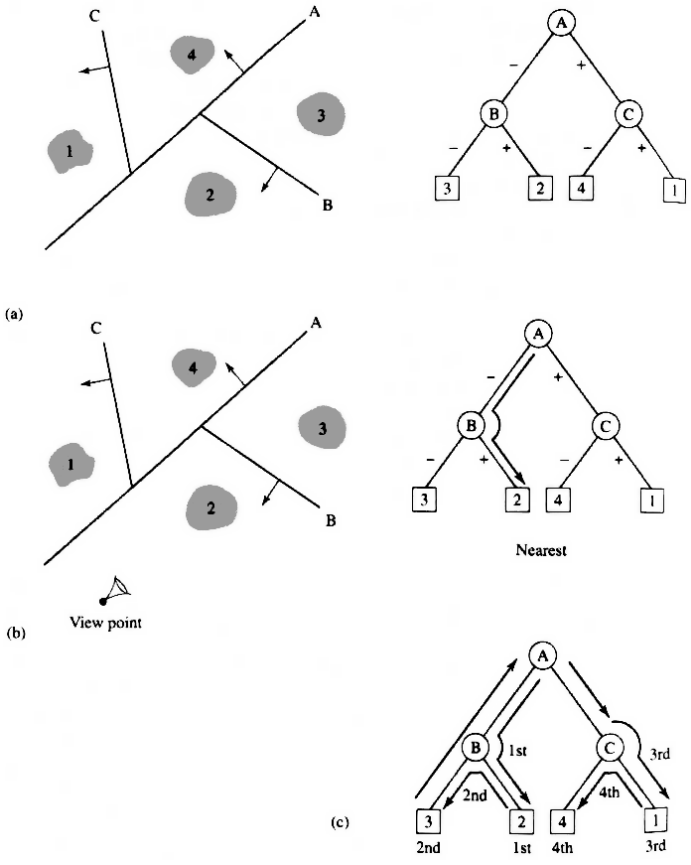
(Alan Watt: 3D CG)

Originálna myšlienka odstraňovania neviditeľných povrchov vychádza z predpokladu nemennej scény, čo do polohy i rozmerov jej objektov, avšak umožňuje zmenu polohy pozorovateľa pri opise scény. Realizácia tejto myšlienky predstavuje dvojfázový proces. V prvej fáze sa konštruuje BSP-strom scény (len raz) a v druhej fáze sa zavádza poloha pozorovateľa a zosúlaďuje sa so skonštruovaným BSP-stromom, s cieľom zistiť viditeľnost. Atraktívnosť tejto metódy pre real-time grafiku je v tom, že mnohé z výpočtov možno realizovať v rámci predspracovania. Najskôr si všimneme najjednoduchšie možnosti, ako týmto spôsobom zisťovať viditeľnosť objektov v scéne a potom si ukážeme, ako sa dajú tieto princípy rozšíriť na určovanie viditeľnosti mnohouholníkových stien telesa.

A: Viditeľnosť objektov v scéne

Ak naša scéna pozostáva z konvexných objektov (ktoré vznikli z konvexných oblastí, čiže ako prieniky polpriestorov), ktoré môžu byť oddelené rovinami, môžeme na delenie priestoru použiť rekurzívnu stratégiu rozdeľuj a panuj. Predpokladáme, že máme nejakú vhodnú stratégiu pre rozmiestňovanie rezových rovín a že strom možno „skompletovať“, ak každá oblasť bude obsahovať len jediný objekt.

Obrázky 1a – 1c demonštrujú túto myšlienku.



Obr.1

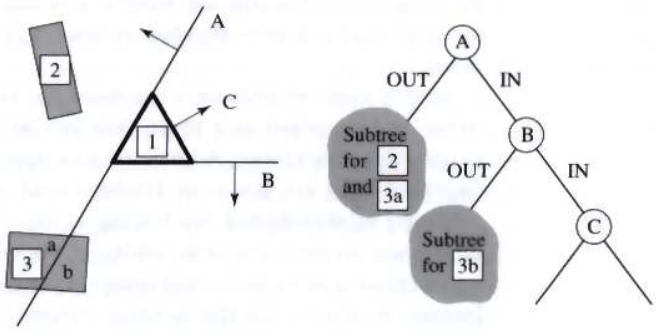
Každý list je označený identifikátorom jemu priradeného objektu a každý uzol reprezentuje deliacu –(rezovú) rovinu. Ak už máme strom zostrojený (obr.1a), môžeme určiť usporiadanie (objektov) podľa ich viditeľnosti z daného pozorovacieho bodu, pričom zostupujeme po strome od koreňa – so súradnicami pozorovacieho bodu pre určenie objektu najbližšieho k polohe pozorovateľa (obr.1b). Začíname v koreni, zostúpime k podstromu na tej strane roviny A, ktorá je bližšia k pozorovaciemu bodu (v tomto prípade je to strana záporná), prejdeme k uzlu asociovanému s rovinou B a odtiaľ k listu 2. Obrázok 1c indikuje smer, ktorým sa máme uberať, keď chceme určiť usporiadanie objektov podľa viditeľnosti. Po dosiahnutí listu 2 sa vrátime do uzla B, prejdeme na vzdialenejšiu stranu roviny B(-) a zostúpime k listu 3. Nasleduje návrat ku koreňu, prechod na vzdialenejšiu stranu roviny A(+), zostúpenie po nej až do listu 1 cez A(+) a C(+); návrat do uzla C, prechod na jeho vzdialenejšiu stranu C(-) a zostúpenie k listu 4. Takto sme získali nasledujúce viditeľnostné usporiadanie objektov od najbližšieho k najvzdialenejšiemu: 2, 3, 1, 4.

V praxi však opísaná schéma nie je až tak veľmi užitočná, lebo mnoho grafických aplikácií sa zaoberá scénami, ktorých objektová zložitosť (počet mnohouholníkov na objekt) je omnoho väčšia ako scénová zložitosť (počet objektov v scéne) a preto je užitočnejšie, keď pracujeme s mnohouholníkmi na objektoch ako s celými objektami. Taktiež tu existuje netriviálny problém rozmiestňovania deliacich rovín. Pravda, ak počet objektov nie je príliš veľký, tak môžeme zvoliť oddeľovaciu rovinu pre každú dvojicu objektov.

B. **Určovanie viditeľnosti mnohouholníkových stien na telese**

Pri hľadaní usporiadania ($f \rightarrow n$ alebo $n \rightarrow f$) stien priestorového objektu, ktorý má všetky steny rovinné, je výhodné voliť za deliace roviny práve roviny obsahujúce stenové mnohouholníky. Mnohouholník sa potom odstráni zo zoznamu a použije sa ako koreňový uzol. Všetky ostatné mnohouholníky sa potom testujú voči rovine obsahujúcej tento mnohouholník a umiestňujú na vhodnú zostupnú vetvu stromu. Každý mnohouholník, ktorý pretína rovinu koreňového mnohouholníka sa rozdelí na dve časti. Tento proces rekurzívne pokračuje, pokiaľ nie sú roviny všetkých mnohouholníkov použité ako deliace. Obyčajne procedúra vytvára viac mnohouholníkov ako bolo pôvodne v scéne, avšak nie toľko, aby to v praxi spôsobovalo vážnejšie problémy.

Tento proces je na jednoduchom príklade ilustrovaný na obr.2.



Obr.2

Ako prvá je zvolená rovina A obsahujúca mnohouholník z objektu 1, ktorá delí objekt 3 na dve časti 3a a 3b. Strom sa vytvára ako predtým, ale teraz sme použili IN/OUT konvenciu na

vyjadrenie, že nejaká strana rozkladu určitú entitu obsahuje, pretože toto má teraz zmysel pre mnohouholníkové objekty.

Usporiadanie od najvzdialenejších k najbližším sa uskutočnilo ako u BSP-stromov. Vykresľovanie mnohouholníkov vo frame-buffri v tomto poradí je dôsledkom maliarovho algoritmu – bližšie mnohouholníky prekresľujú vzdialenejšie, už vykreslené mnohouholníky.

Pre generovanie viditeľnostného usporiadania scény potrebujeme:

- zostúpiť po strome so súradnicami pozície pozorovateľa (v koreni)
- v každom uzle určiť, či pozorovateľ je na prednej, alebo zadnej strane roviny obsahujúcej rovinu tohto uzla
- zostúpiť najskôr po podstrome na vzdialenej strane a dať na výstup mnohouholníky
- zostúpiť po bližšej strane podstromu a dať na výstup mnohouholníky.

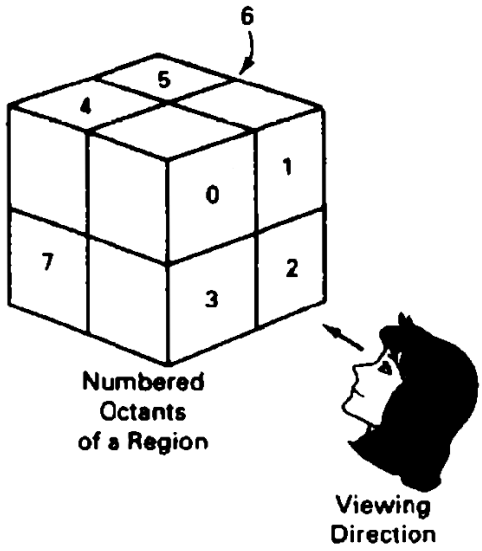
Tento postup dáva usporiadanie mnohouholníkov odzadu dopredu vzhľadom na aktuálnu polohu pozorovateľa a sú v tomto poradí aj vykresľované do frame buffra.

Usporiadanie mnohouholníkov od najbližšieho k najvzdialenejšiemu sa po istých formálnych úpravách v počítačovej grafike tiež úspešne využíva, vrátane riešenia niektorých extrémne komplikovaných prípadov.

Problematika usporiadania je podrobnejšie opísaná v učebnici Theory of three-dimensional Computer Graphics, ktorú editoval L. Szirmay-Kalos.

Poznámka k viditeľnostným OCTREE metódam (podľa učebnice Hearn – Baker).

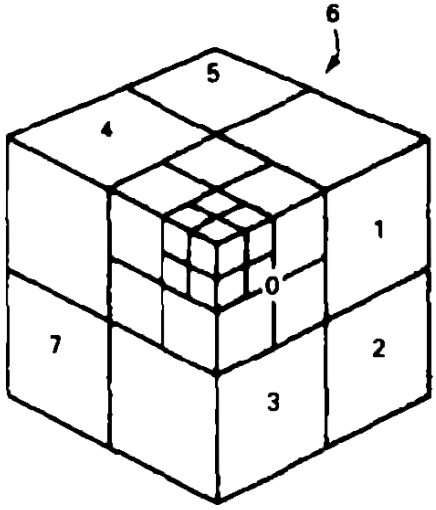
- Ak je použitá stromová reprezentácia pre vizualizáciu objemu, tak odstránenie zakrytých plôch sa vykoná premietnutím octree uzlov na pozorovaciu rovinu v usporiadaní odpredu dozadu. Na obrázku 13.24 je predná strana priestorovej oblasti (strana k pozorovateľovi) vytvorená oktantami 0, 1, 2 a 3.



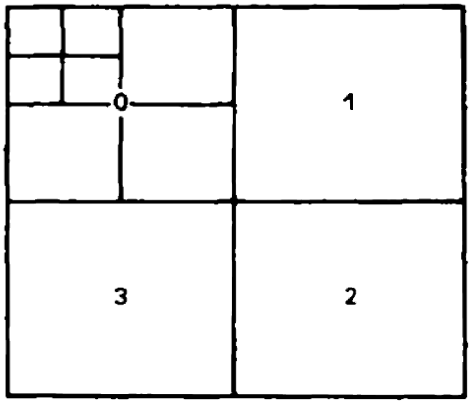
Obr.13.24

Predné povrchy (čelné steny) týchto oktantov sú pre pozorovateľa (z hľadiska pozorovateľa) viditeľné. Všetky povrchy (steny) smerom k zadným častiam predných oktantov, alebo na zadných oktantoch (4, 5, 6, 7) môžu byť zakryté prednými stenami (povrchmi).

- Zadné steny sú z hľadiska smeru pozorovania vyznačeného na obrázku eliminované spracovaním dátových prvkov uložených v uzloch stromu octree v poradí 0, 1, 2, 3, 4, 5, 6, 7. Toto vyústi v prvom prehliadaní octree-stromu do toho, že uzly reprezentujúce oktanty 0, 1, 2 a 3 v celej oblasti sú prechádzané pred uzlami reprezentujúcimi oktanty 4, 5, 6 a 7. Podobne uzly pre predné štyri podoktanty oktantu 0 sú prehliadavané pred uzlami pre štyri zadné podoktanty. Prieskum octree pokračuje v tomto poradí pre každý oktant podrozdelenia.
- Ak sa v uzle octree objaví farba pozadia, tak pixlovej oblasti vo frame-buffri zodpovedajúcej tomuto uzlu je táto farba priradená len vtedy, ak predtým žiadna hodnota nebola do tejto oblasti uložená. Vďaka tomu sú uložené v buffri len farby popredia. Do prázdnej oblasti nič nie je uložené. Každý uzol, o ktorom sa zistí, že je úplne obklopený, je vylúčený z ďalšieho spracovania, takže jeho podstromy nie sú sprístupnené.
- Iný prístup k objektom reprezentovaných pomocou octrees možno získať aplikovaním takých transformácií na octrees reprezentácie objektov, ktoré preorientovávajú objekt do súladu so zvoleným pohľadom. Predpokladajme, že octree reprezentácia je vždy nastavená tak, že oktanty 0, 1, 2 a 3 oblasti tvoria čelnú stenu, ako na obr. 13-24.
- Jedna metóda pre zobrazenie octree je taká, že najskôr sa zobrazí octree na quadtree viditeľných oblastí traverzovaním (prehliadaním) uzlov octree-stromu spredu dozadu v rekurzívnej procedúre. Potom je quadtree reprezentácia pre viditeľné plochy „natiahnutá-zavedená“ do frame buffra. Obrázok 13.25 znázorňuje oktanty v priestorovej oblasti a korešpondujúce kvadranty v pozorovacej rovine.



Octants in Space



Quadrants for
the View Plane

Obr.13.25

Príspevky kvadrantu 0 prichádzajú od 0 a 4. Hodnoty farieb v kvadrante 1 sú získané z plôch v oktantoch 1 a 5 a hodnoty v každom z ďalších dvoch kvadrantov sú generované z dvojice oktantov súosích (vyrovnaných do priamky) s každým z týchto kvadrantov.

- Rekurzívny výpočet (processing) octree uzlov je demonštrovaný v procedúre:

„typedef enum (SOLID, MIXED) Status“

v učebnici CG Hearn-Baker str.486; ktorá vychádza z octree-opisu a vytvára quadtree reprezentáciu pre viditeľné plochy v oblasti. V mnohých prípadoch sa oba – predný aj zadný oktant - musia preskúmať (vyšetriť) na korektnosť určenia hodnoty farby pre kvadrant.

Ale ak predný oktant je homogénne vyplnený určitou farbou, nemusíme už vyšetrovať zadný oktant. Pre heterogénne oblasti sa rekurzívne volá procedúra, ktorá používa ako nové argumenty deti (potomkov) heterogénnych oktantov a novovytvorené quadtree - uzly. Ak predok (front) je prázdny, tak sa spracúva zadný oktant. V opačnom prípade sa urobia dve rekurzívne volania, jedno pre zadný oktant a jedno pre predný oktant.

O tejto problematike sa možno dočítať aj v učebnici RUFE na stranách 172-174.

Algoritmus: Newel-Newel –Sancha

Algoritmus N-N-Sancha je jeden z prístupov, ako rozvinúť myšlienky načrtnuté v maliarovom algoritme.

Prvým krokom je výpočet východzieho hĺbkového usporiadania. Toto je dané roztriedením mnohouholníkov podľa ich maximálnej z-ovej súradnice z_{max} do zoznamu L. Ak neexistujú žiadne dva mnohouholníky, ktorých rozsahy z-ových súradníc by sa prekrývali, tak zoznam L považujeme za korektne hĺbokovo usporiadaný. V opačnom prípade, a to je všeobecný prípad (vyjmúc veľmi špeciálne scény napr. také, ktoré pozostávajú iba z mnohouholníkov ležiacich v rovinách kolmých na os z), je nevyhnutné uskutočniť ďalšie výpočty.

- Zvoľme si najskôr mnohouholník P, ktorý je poslednou položkou v zozname L.

Ak rozsah z-ových súradníc jeho bodov sa neprekrýva s rozsahmi z-ových súradníc všetkých predchádzajúcich mnohouholníkov, tak mnohouholník P je korektne umiestnený a možno vziať mnohouholník nachádzajúci sa v zozname L pred P a uskutočniť s ním tie isté úvahy ako s mnohouholníkom P.

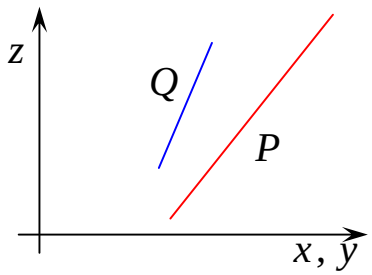
V opačnom prípade (a to je všeobecný prípad), mnohouholník P prekrýva z-rozsahom mnohouholníky z množiny $\{Q_1, Q_2,...,Q_m\}$ t.j. jeho z-ový rozsah sa prekrýva so z-ovým rozsahom každého z nich. Túto množinu možno nájsť prehľadávaním zoznamu L od mnohouholníka P dopredu a postupným zaraďovaním všetkých mnohouholníkov Q takých, že $z_{max}(Q) > z_{min}(P)$.

- V nasledujúcom kroku sa pokúsime zistiť, či mnohouholník P nezakrýva niektorý z mnohouholníkov množiny $\{Q_1, Q_2,...,Q_m\}$.

Mnohouholník P nezakrýva mnohouholník Q , ak $Q \succ P$, t.j. ak mnohouholník P nezakrýva žiadny bod mnohouholníka Q, čiže ak je splnená niektorá z nasledujúcich podmienok:

1. $z_{min}(P) > z_{max}(Q)$ (nesmú sa prekrývať ich z-rozsahy (tzv. z minimax check));
2. Ohraničujúci pravouholník P v rovine (x,y) sa nesmie prekrývať s ohraničujúcim pravouhoulníkom Q v rovine (x,y) (tzv. x,y minimax check);
3. Každý vrchol mnohouholníka P je vzdialenejší od pozorovacieho bodu (view-point) než rovina obsahujúca Q;
4. Každý vrchol mnohouholníka Q je bližšie k pozorovaciemu bodu (view-point) než rovina obsahujúca mnohouholník P;
5. Priemety mnohouholníkov P, Q do roviny (x,y) sa nesmú pretínať.

Usporiadanie týchto podmienok odráža zložitosť ich overovania, čo je dôležité z hľadiska praxe. Ak sme takto vyvrátili, že mnohouholník P zakrýva mnohouholník Q pre $Q \in \{Q_i\}$, tak mnohouholník Q možno presunúť za mnohouholník P v zozname L. Táto situácia je ilustrovaná na obrázku:



Prirodzene, ak mnohouholník P pretína mnohouholník Q, potom sa jeden z nich musí rozdeliť na dve časti rovinou obsahujúcou druhý mnohouholník. Cykly tiež možno vyriešiť rozdelením. Aby sme to zabezpečili, ihneď ako je mnohouholník presúvaný na inú pozíciu v zozname L, označíme ho. Ak sa s označeným mnohouholníkom ide opäť hýbať napr. preto, že $Q \not\succ P$, tak sa automaticky predpokladá, že mnohouholník Q je príčinou cyklu a preto sa rozdelí na dve časti Q_1, Q_2 tak, že $Q_1 \not\succ P$ a $Q_2 \succ P$, vďaka čomu len Q_1 sa presunie za mnohouholník P. Vhodnou rezovou rovinou je rovina mnohouholníka P, ako je ilustrované na známom obrázku o vzájomnom prekrývaní sa dvoch a troch mnohouholníkov.

Pre pokračovanie v algoritme N-N-Sancha je vhodné pripomenúť si niektoré zrejmé a užitočné skutočnosti.

Najskôr si všimnime, že pre ľubovoľný mnohouholník P možno vytvoriť dva polpriestory H_P^+ a H_P^- určené rovinou π obsahujúcou P.

Potom:

a) ak pozícia pozorovateľa je v H_P^+, tak pre každý bod $p \in H_P^+$ platí, že mnohouholník P nemôže zakrývať bod p a
b) pre ľubovoľný bod $p \in H_P^-$, bod p nemôže zakrývať mnohouholník P.

Na druhej strane, ak sa pozícia pozorovateľa nachádza v H_P^-, tak získame podobné pozorovacie výsledky, len úlohy H_P^+ a H_P^- treba vymeniť.

Úplný algoritmus pre určenie hĺbkového usporiadania množiny mnohouholníkov $S = \{P_1, P_2,...,P_n\}$ možno získať konštrukciou založenou na nasledujúcej myšlienke, ako to urobili Fuchs a kol.:

- Najskôr si zvoľme jeden z mnohouholníkov P_i, $i \in \{1, 2,...,n\}$. Pomocou neho možno vytvoriť nasledujúce dve množiny:

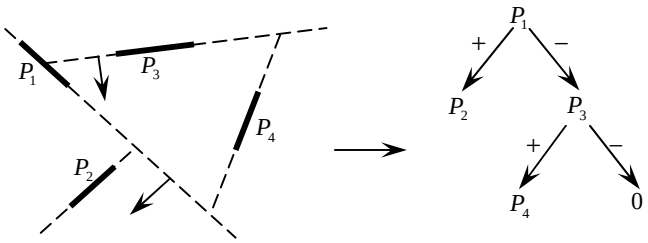
(*) $S_i^+ = (S \setminus P_i) \cap H_i^+$ a $S_i^- = (S \setminus P_i) \cap H_i^-$ kde mohutnosť množín $\left|S_i^+\right|$, $\left|S_i^-\right| \leq |S| - 1 = n - 1$.

Poznamenávame, že niektoré (ak nie všetky) mnohouholníky môžu byť v procese konštrukcie rozdelené na dve časti.

- Ak pozorovací bod (view-point) je v H_i^+, tak mnohouholník P_i nemôže zakryť žiadny z mnohouholníkov v S_i^+ a žiadny mnohouholník z S_i^- nemôže zakryť mnohouholník P_i. Ak pozorovací bod je v H_i^-, tak výsledok je analogický s tým, že úlohy S_i^+ a S_i^- sa zamenia. Teda pozícia mnohouholníka P_i v hĺbkovom usporiadaní je taká, že pre každý mnohouholník $P \in S_i^+$ a $Q \in S_i^-$ je P_i medzi P a Q. Formálne P_i je medzi S_i^+ a S_i^-. Stačí nám teda hĺbkovo usporiadať mnohouholníky v množinách S_i^+ a S_i^-. Urobíme to rekurzívne: vyberie sa mnohouholník P_j z S_i^+ a vytvoria sa dve množiny S_j^+, S_j^- a tak isto sa z množiny S_i^- vyberie mnohouholník P_k a vytvoria sa množiny S_k^+, S_k^-,..., atď. V každom kroku tohto prerozdeľovacieho procesu sa pokúšame hĺbkovo usporiadať mnohouholníky v množinách typu $S_\bullet^+$ resp. $S_\bullet^-$, kde bodky reprezentujú nejaké indexy. Po každom kroku v procese pokračujeme ďalej len s tými množinami vyššie uvedeného typu, ktoré obsahujú viac ako jeden mnohouholník.

Pretože podľa (*) je mohutnosť množín $(S_\bullet^+)^+ ..., (S_\bullet^+)^-$ $[(S_\bullet^-)^+ ..., (S_\bullet^-)^-]$ aspoň o 1 menšia ako mohutnosť množiny $S_\bullet^+$ $[S_\bullet^-]$ je jasné, že táto stop podmienka rekurziu korektne ukončí. Výsledkom bude množina $S_\bullet^\bullet$, v ktorej keď jednoprvkové podmnožiny mnohouholníkov nahradíme mnohouholníkmi samotnými, dostaneme hĺbkovo usporiadanú množinu *S*.

Predchádzajúca konštrukcia bola založená na prerozdeľovaní objektového priestoru na dva polpriestory. Ako už vieme, takéto konštrukcie sa zvyknú reprezentovať binárnymi stromami. Naša konštrukcia výslednej množiny je založená na binárnom prerozdeľovaní objektového priestoru, ktoré je ilustrované na obr.1.



Obr.1.

∗ Prvá rovina nám rozdelí priestor na dva polpriestory, ďalšia rovina delí výsledné polpriestory, konkrétne druhá rovina rozdelila prvý polpriestor, tretia rozdelila druhý polpriestor, atď.

∗ Takéto prerozdeľovanie sa dá dobre reprezentovať binárnym stromom, tzv. BSP-stromom, ktorý je tiež ilustrovaný na obr.1. Prvá rovina je tu asociovaná s koreňom, druhá a tretia sú asociované s potomkami koreňa, atď.

∗ Pre naše účely nie sú až tak dôležité roviny samotné, ale ich určujúce mnohouholníky, ktoré preto priradíme uzlom stromu, vďaka čomu bude aj množina $S_\bullet^\bullet$ všetkých mnohouholníkov obsiahnutých v objeme asociovaná s každým uzlom. Každý listový uzol potom nebude obsahovať žiadny mnohouholník (bude prázdny), alebo bude obsahovať mnohouholník z asociovanej množiny $S_\bullet^\bullet$.

Algoritmus pre vytvorenie BSP-stromu pre množinu *S* môže byť nasledovný:

BSP Tree (*S*)
vytvorenie nového uzla *N*;
***S*(*N*) = *S*;**
if $|S| \le 1$ **then**
$P(N) = \text{null}; L(N) = \text{null}; R(N) = \text{null};$
else
$P = \text{Select}(S); P(N) = P;$
vytvor množiny S_P^+ a S_P^-**;**
$L(N) = \text{BSP Tree}(S_P^+)$**;**
$R(N) = \text{BSP Tree}(S_P^-)$**;**
end if
return *N*;
end.

Kde $S(N)$, $P(N)$, $L(N)$ a $R(N)$ označujú postupne množinu všetkých mnohouholníkov, rezový mnohouholník (jeho rovinu), ľavý a pravý potomok asociovaný s uzlom N.

Veľkosť BSP-stromu, čiže počet v ňom uložených mnohouholníkov ,je na jednej strane vysoko závislá na tvare objektov scény a na druhej strane na voľbe stratégie použitej v procedúre Select.

Tvorcovia algoritmu navrhli aj heuristické (nedokázané, ale praxou overené) kritérium na minimalizáciu počtu mnohouholníkov v BSP-strome. Ich stratégia je založená na snahe minimalizovať počet mnohouholníkov, ktoré sa majú rozdeliť a počty dvojíc mnohouholníkov, ktoré sú v konflikte (t.j. patriacich tej istej množine mnohouholníkov pričom rovina jedného pretína druhý).

Procedúra Select(S) vráti $P \in S$ práve vtedy, keď je pre P výraz $I_{13} + I_{31} + w|S_2|$ maximálny, kde w je experimentálne určená váha, $I_{ij} = \sum_{P \in S_i} \sum_{Q \in S_j} f(P,Q)$; pričom platí, že $f(P,Q) = 1$, ak rovina obsahujúca P pretína Q; v opačnom prípade $f(P,Q) = 0$. Potom $S_1 = \{Q \in S; Q$ je celý v $H_P^+\}$, $S_2 = \{Q \in S; Q$ je preťatý rovinou $\alpha \supset P\}$ a $S_3 = \{Q \in S; Q$ je celý v $H_P^-\}$ sú podmnožiny S.

Zdôrazňujeme, že BSP-strom určený týmto algoritmom je pohľadovo nezávislý t.j. obsahuje vlastné hĺbkové usporiadanie pre každú polohu pozorovateľa. Rozdiely spôsobené rôznymi polohami pozorovateľa sa prejavia v spôsobe prehľadania (traverzovania) stromu pre opravu (vylepšenie) aktuálneho usporiadania. Podľa charakteristických čŕt BSP-stromu jeho prehľadávanie bude vždy typu „inorder prehľadávaním“. Ak predpokladáme, že určitá akcia sa má previesť na každom uzle binárneho stromu, tak inorder prehľadávanie (inorder traversal) znamená, že pre každý uzol binárneho stromu sa najskôr prezrie (rekurzívne preskúma) jeden z jeho potomkov (synov), potom sa akcia prevedie na samotnom uzle a nakoniec sa preskúma druhý potomok. Akciou pre každý uzol N sa tu rozumie vykreslenie mnohouholníka $P(N)$ asociovaného s uzlom N. Ak poloha pozorovateľa je v $H_{P(N)}^+$, tak sa najskôr vykreslí pravý podstrom, potom mnohouholník $P(N)$ a nakoniec ľavý podstrom; usporiadanie ľavého i pravého potomka je odzadu dopredu.

Nasledujúci algoritmus kreslí mnohouholníky stromu BSP-tree N v ich vlastnom hĺbkovom usporiadaní:

```
BSP Draw (N)
    if N je prázdne then return;
    if poloha pozorovateľa je v H+P(N) then
       BSP Draw(R(N)); Draw(P(N)); BSP Draw(L(N));
    else
       BSP Draw(L(N)); Draw(P(N)); BSP Draw(R(N));
     endif
end.
```

Keď už BSP-strom bol vytvorený procedúrou BSP Tree, nasledujúce obrazy pre ďalšie polohy pozorovateľa sa dajú vygenerovať postupnými volaniami procedúry BSP Draw.