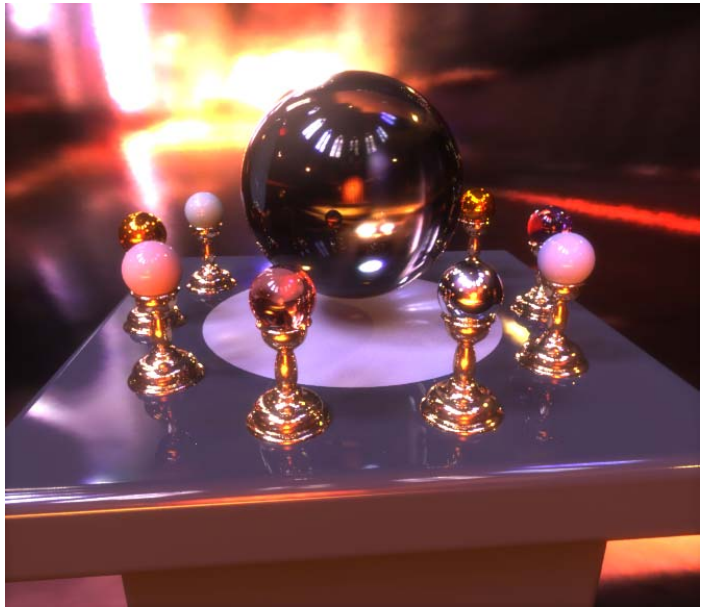
Računarska grafika
(2OER7O02)

Algoritmi za ostvarivanje realnosti prikaza

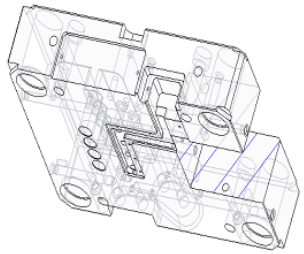
Predavanja

Realnost prikaza

- Uklanjanje sakrivenih ivica i površina
- Boje
- Svetla
- Senke



Uklanjanje sakrivenih ivica i površi



- Za čoveka je uklanjanje sakrivenih linija i površina jednostavan problem.
- Za računar je to znatan problem.
- Uvek kada slika sadrži neprozirne objekte i površi, tada objekti koji su bliže oku prekrivaju objekte koji leže iza njih na istoj liniji pogleda.
- Da bi slika na ekranu bila realna, potrebno je da se izbace svi sakriveni objekti i sve sakrivene površi.
- Da bi se rešio ovaj problem potrebno je da se izračuna **dubina** (*depth*) i **vidljivost** (*visibility*) svih površi slike.

Upoređivanje dubine

- Problem vidljivosti se svodi na utvrđivanje da li dve tačke zaklanjaju jedna drugu. To se može utvrditi sledećim postupkom:

    1. Utvrditi da li tačke P1 i P2 leže na istom projekcionom zraku.

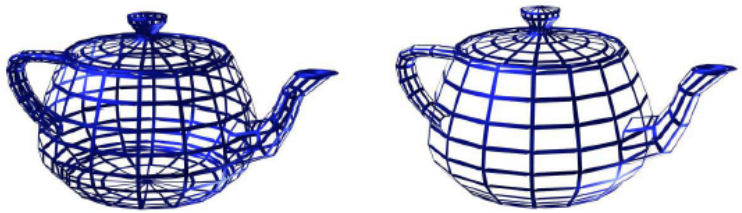
    2. Ako tačke P1 i P2 ne leže na istom projekcionom zraku, onda se one ne zaklanjaju. Međutim, ako se nalaze na istom projekcionom zraku, onda je potrebno utvrditi koja se nalazi bliže posmatraču.

Upoređivanje dubine

- Poređenje dubina se obično vrši posle normalizacije. Tada su projekcioni zraci kod paralelne projekcije paralelni sa Z osom, a kod perspektivne projekcije polaze iz koordinatnog početka.
- Kod paralelne projekcije, tačke su na istom projekcionom zraku ako je **x1 = x2** i **y1 = y2**, a kod perspektivne projekcije, tačke su na istom projekcionom zraku ako je **x1/z1 = x2/z2** i **y1/z1 = y2/z2**.

Algoritmi za uklanjanje sakrivenih ivica i površi

- Do sada je razvijen veliki broj algoritama za rešavanje ovog problema. Ni za jedan od ovih algoritama se ne može reći da je najbolji. Svaki od njih ispunjava neke od zahteva za rešavanje problema sakrivenih površi. Shodno tome ih treba birati u konkretnoj aplikaciji.
- Postoje dva prilaza:
  - Algoritmi **prostora slike**
  - Algoritmi **prostora objekata**



Algoritmi prostora slike

- Prvi pristup za svaki piksel slike dimenzija n x m određuje koji od k objekata je vidljiv.
- Složenost algoritma je $O(n \cdot m \cdot k)$
- *Algoritam:*

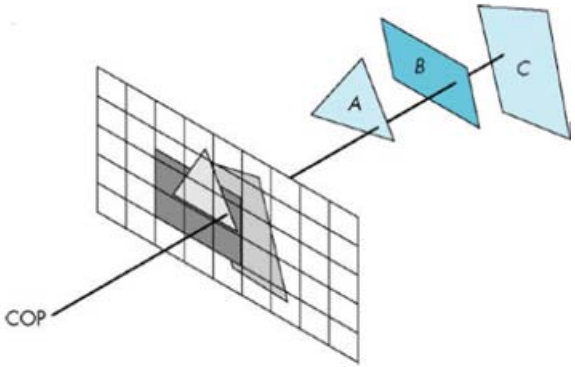
1. Za svaki piksel slike uraditi:

2. Odrediti objekat najbliži posmatraču (duž prave određene projekcijom).

3. Obojiti piksel odgovarajućom bojom.

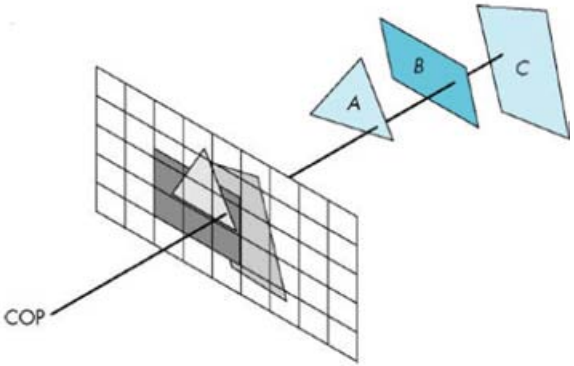
Algoritmi prostora slike

- Ovaj pristup se obično zove ***image-precision***.
- Najpre je korišćen u ranim sistemima sa rasterskom grafikom (koji mogu da adresiraju relativno mali broj tačaka, ali mogu da prikazuju veliki broj objekata).



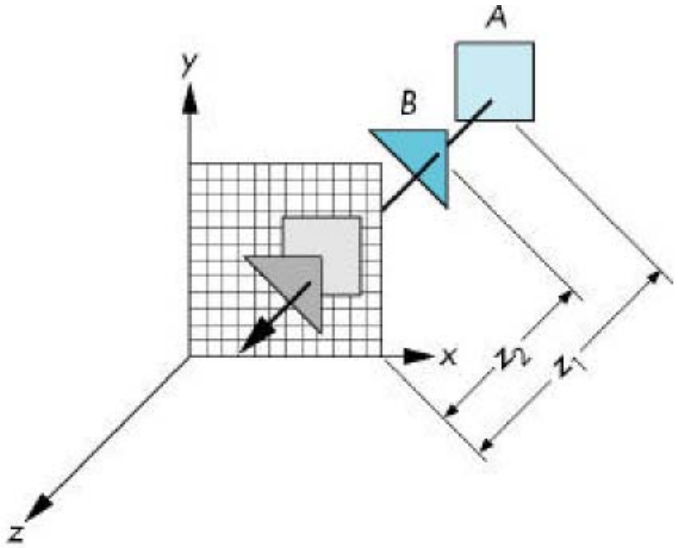
Algoritmi prostora slike

- Algoritmi iz ove grupe su:
  - Z-buffer algoritam (*Catmull* 1974).
  - *Warnock*-ov algoritam (*Warnock* 1969).
  - Scan-line algoritam (*Watkinson*-ov algoritam).
  - Ray-tracing algoritam (algoritam praćenja zraka - *Appel* (1968), *Goldstein i Nagel* (1968, 1971)).



Z-buffer algoritam

- Algoritam sa Z baferom koristi pored bafera uređaja za prikazivanje slike (*frame buffer*) i poseban bafer gde se u poziciji (x,y) pamti najmanja Z koordinata (dubina) tačaka koje se vide iz piksela (x,y).
- Ovaj bafer se zove **Z-buffer**. Dakle, najmanja dubina svih tačaka koje se vide iz piksela (x,y) se pamti u Z-baferu na poziciji (x,y).
- Koristi se u OpenGL-u



Z-buffer algoritam

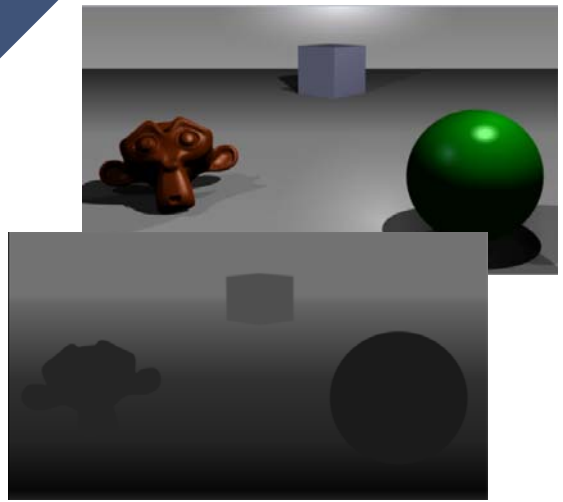
```
void zBuffer(){
int pz;
for (y = 0; y <= YMAX; y++)
 for (x = 0; x <= XMAX; x++){
  WritePixel(x, y, background_value);
  WriteZ(x, y, zmax);
  }
for (each polygon)
 for (each pixel in polygon's projection){
   pz = polygon's z-value at pixel coordinates (x,y);
   if (pz <= ReadZ(x,y)){
     WriteZ(x, y, pz);
     WritePixel(x, y, polygon's color at pixel coordinates (x,y));
   }
 }
```

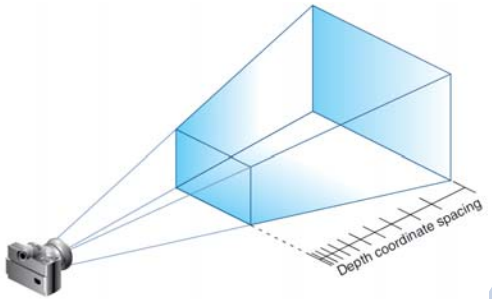
Z-buffer - prednosti

- Jednostavno se implementira i softverski i hardverski.
- Nije potrebno sortiranje objekata pre primene algoritma.
- Poligoni se pojavljuju na slici redom kojim se obrađuju.
- Moguće su optimizacije računanja (izbegavanje množenja i sl. na bazi prirode scan-conversion algoritma).
- Z-bafer algoritam ne zahteva nužno da primitivne površi budu poligoni.
- Z-bafer podaci mogu da budu čuvani zajedno sa generisanom slikom i da se toj sceni *naknadno* doda novi objekat.

Z-buffer - nedostaci

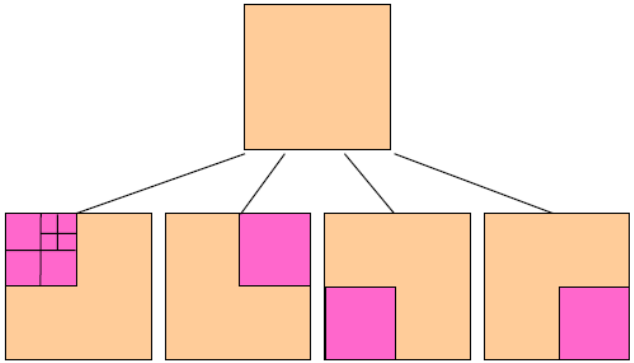
- Zahteva postojanje memorijskog prostora ne samo za vrednost boje za svaki piksel (frejm bafer), već i memorijskog prostora za z-vrednosti (tj. z-koordinate) za svaki piksel (z-bafer).
- Nelinearna preciznost (u osnovnoj verziji, smanjuje se sa rastojanjem)
- Piksel se iscrtava više puta





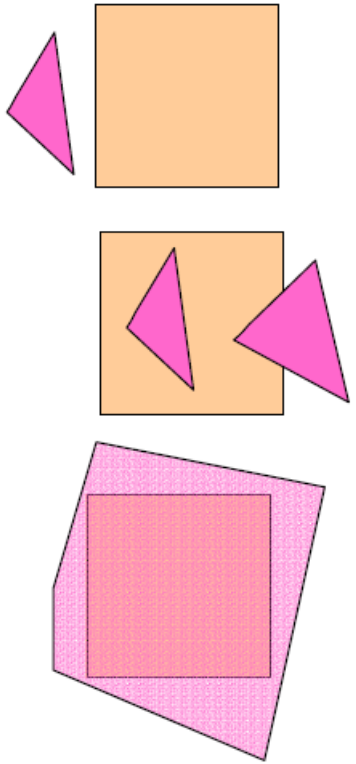
Warnock-ov algoritam

1. Analiziramo sadržaj ispitnog prostora koji je u početku jednak ekranu. Mogući slučajevi:
   - prozor je prazan
   - scena u prozoru je jednostavna i moguće ju je prikazati
   - scena u prozoru je složena, rekurzivno delimo dalje



Warnock-ov algoritam

2. Podela prostora (*quad tree*).
3. Podela poligona u četiri grupe, u zavisnosti od relacije sa prozorom:
   - poligon je izvan prozora (*disjoint*) – eliminiše se
   - poligon prekriva čitav prozor (*surrounding*)
   - poligon je u prozoru (*containing*)
   - poligon seče prozor (*intersecting*) – deli se na:
     - *disjoint* i
     - *containing*

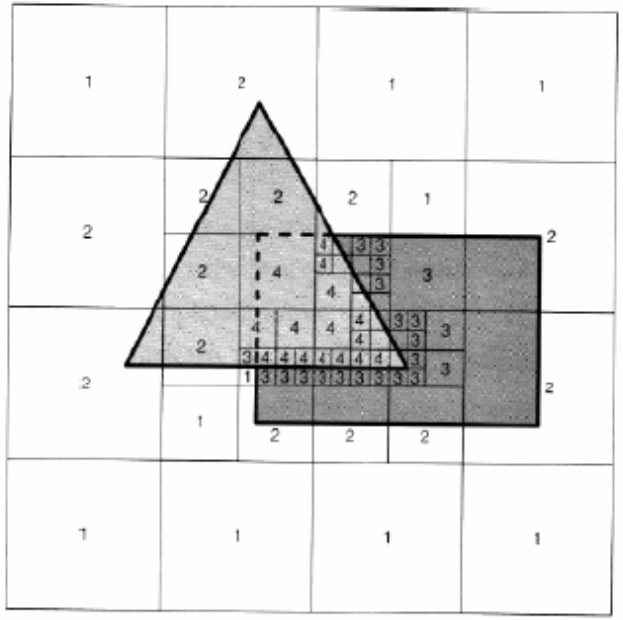


Warnock-ov algoritam

1. Svi poligoni su van tekuće oblasti – boji se pozadinskom bojom
2. Postoji samo jedan poligon koji preseca tekuću oblast ili je potpuno sadržan u njoj – boji se pozadinskom bojom, a zatim rasterizuje dati poligon.
3. Postoji samo jedan obuhvatajući poligon – ispunjuje se bojom datog poligona.
4. Proverava se da li postoji obuhvatajući poligon koji je ispred svih ostalih, ako postoji ispunjuje se njegovom bojom. U protivnom, oblast se deli na 4 podoblasti.

Postupak podele se nastavlja dok oblast ne bude „jednostavna“ ili se ne dođe do 1 piksela (kada se utvrđuje koji je poligon najbliži i njegova boja se koristi za ispunu).

Warnock-ov algoritam



1. Svi poligoni su van tekuće
2. Postoji samo jedan poligon koji preseca tekuću oblast ili je potpuno sadržan u njoj
3. Postoji samo jedan obuhvatajući poligon
4. Postoji obuhvatajući poligon koji je ispred svih ostalih

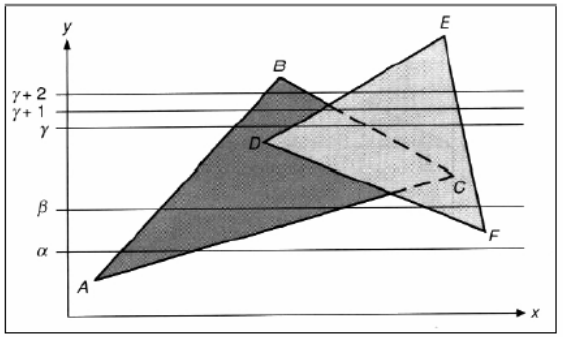
Scan-line algoritam

- Predstavlja proširenje algoritma za ispunu, prikazanog ranije.
- Razlika je u tome što ne radimo sa jednim, već više poligona odjednom.
- Kreira se ***Edge Table*** (**ET**) za sve nehorizontalne ivice svih poligona koji se projektuju na ravan projekcije. (horizontalne se ignorišu)
- Stavke u ET se sortiraju u bakete na osnovu minimalne vrednosti Y koordinate, a u okviru baketa po rastućoj X koordinati niže krajnje tačke.
- Svaka stavka sadrži: **Ymax**, $\mathbf{X_{ymin}}$ koordinatu koja odgovara Ymin, **ΔX** (inkrement X pri prelasku na sledeću sken-liniju) i **ID** poligona kome pripada ivica.

Scan-line algoritam

- Postoji i ***Polygon Table*** **(PT)**, koja sadrži za svaki poligon: **ID**, **koeficijente jednačine ravni**, **način senčenja** ili **boju** i „**in-out**“ *bool* fleg (inicijalno FALSE).
- Kada sken-linja preseče ivicu nekog poligona, počinje njegova rasterizacija i „in-out“ fleg se postavlja na TRUE (trenutno je IN).
- Kada je za dva ili više poligona setovan na TRUE, došlo je do preklapanja. Zbog efikasnosti, obično se pravi posebna lista koja sadrži sve poligone u stanju IN.

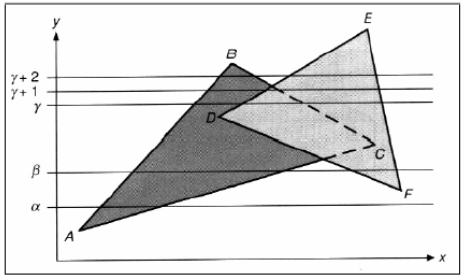
Za y=α, sken linija preseca samo jedan poligon
Za y=β, preseca dva, ali ne istovremeno
Za y=γ, dva poligona su u stanju IN, kada se naiđe na ivicu DE, pa se tada određuje koji je bliži



Scan-line algoritam

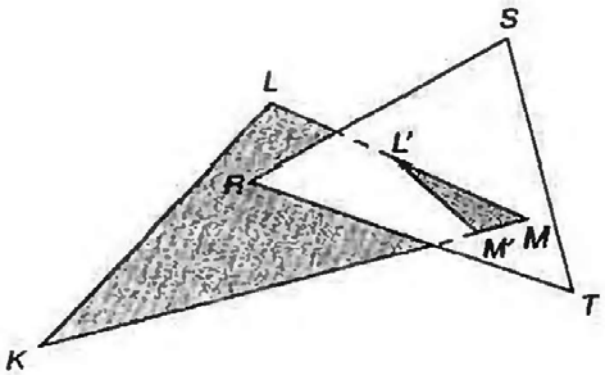
- Koji je poligon bliži, određuje sa na osnovu jednačina ravni, zamenom vrednosti za Y (tekuća sken-linija) i X (na osnovu AET).
- Ako se ne presecaju linije preseka ravni poligona sa Y=Yi ravni (sadrži sken-liniju), onda se „isključuje“ dalji poligon („in-out“ postaje FALSE), a nastavlja rasterizacija bližeg (IN).
- Dubinska koherencija se može iskoristiti ako se poligoni ne presecaju uzajamno. Kada jednom odredimo koji je ispred drugog, ne mora se sa sledećom sken-linijom ponovo testirati preklapanje, već se nasleđuje od prethodne linije.

$$Ax + By + Cz + D = 0$$



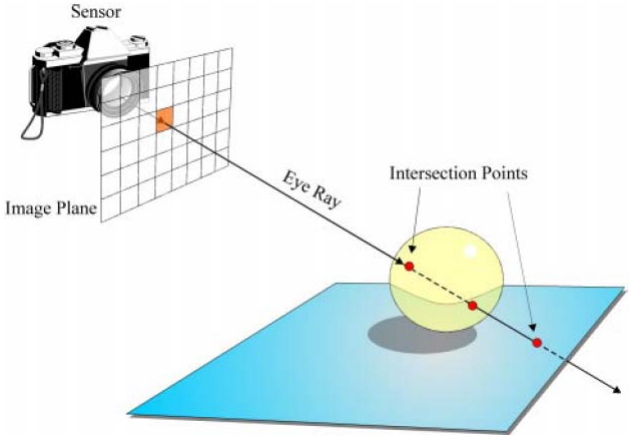
Scan-line algoritam

- Ako se poligoni uzajamno presecaju, to se može rešiti na više načina:
  - Uvođenjem lažnih ivica (L'M') na mestima gde jedan poligon probija drugi i uvrstiti ih u ET
  - Na nivou svake sken-linije, odrediti X-koordinatu preseka nakon koje se menja „vidljivost poligona“



Ray-tracing algoritam (algoritam praćenja zraka)

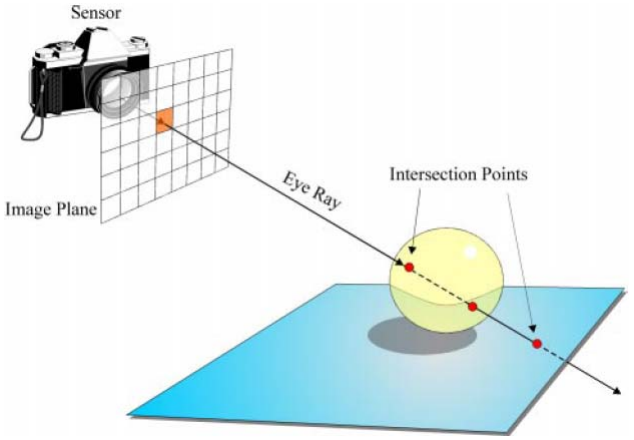
- Ovaj algoritam se još zove i **Ray-casting** algoritam, mada se ovo ime obično koristi samo za varijantu za određivanje vidljivosti (a ray-tracing za puni rekurzivni algoritam).
- Ideja je sledeća: Za svaki piksel na slici se emituje imaginarni zrak od oka posmatrača ka sceni, a da pritom prođe kroz zadati piksel. Nakon toga se detektuje prvi objekat koji je pogođen emitovanim zrakom i piksel se boji u boji pogođenog objekta.



Ray-tracing algoritam (algoritam praćenja zraka)

- Originalno razvijen za simuliranje putanja balističkih projektila, sada ima najznačajnije primene u grafici. Koristi se za određivanje vidljivosti, bulove operacije nad telima, određivanje refleksije i refrakcije (prelamanja) itd.

```
for each pixel in viewport
{
    determine eye ray for pixel
    intersection = trace(ray, objects)
    colour = shade(ray, intersection)
}
```



Algoritmi prostora objekata

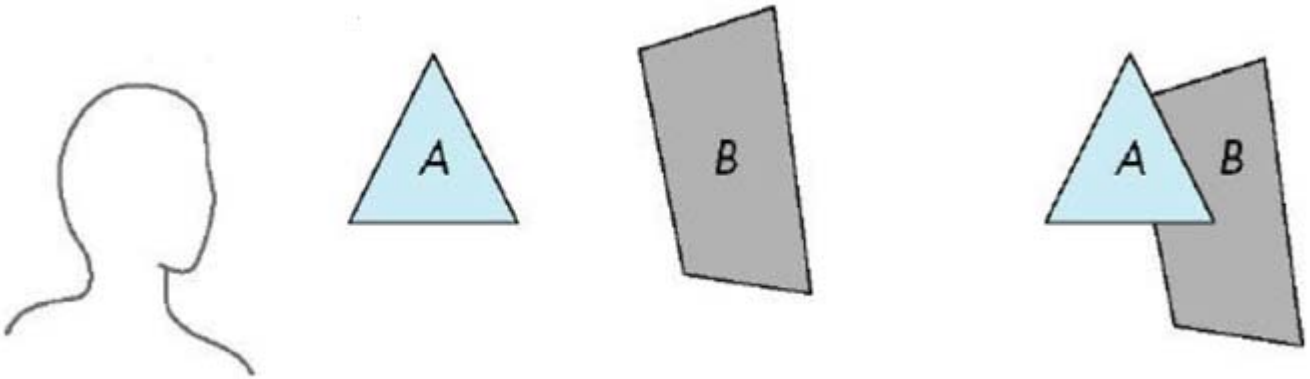
- Drugi pristup za svaki od *n* objekata u sceni određuje koji je vidljiv.
- Složenost algoritama je $O(n^2)$
- Ovaj pristup se obično zove ***object-precision***.

Algoritmi prostora objekata

- Algoritmi iz ove grupe su:
  - Slikarev algoritam (*Newell & Sancha, 1972*).
  - *Back face culling* algoritam.
  - Algoritam koji koristi *BSP* stabla (*Fuchs, Kedem, Naylor*, 1980).
  - Algoritam koji koristi *octree* stabla.

Slikarev algoritam (*depth-sort*)

- Suština: prvo se "slika" najudaljeniji poligon koji ne sakriva druge, a zatim se preko njega slikaju poligoni koji ga (delimično) zaklanjaju.



Slikarev algoritam (*depth-sort*)

Algoritam:

1. Sortiraju se poligoni po rastućoj z-koordinati najudaljenijeg temena (z_{max}) poligona.
2. Kreće se od najudaljenijeg poligona i za svaki poligon se proverava da li on zaklanja neki od narednih poligona u listi. Ako zaklanja – zaklonjeni poligon se prevezuje ispred zaklanjajućeg: sprovodi se 5 testova koji seređaju od jednostavnijih ka složenijim proverama.
3. Rešavaju se uzajamna preklapanja – detektuje se potencijalna beskonačna petlja.
4. Crtaju se popunjeni poligoni počevši od najudaljenijih od posmatrača.

Slikarev algoritam (*depth-sort*)

- 5 testova preklapanja*
  1. Da li se poligoni ne preklapaju po x koordinati?
  2. Da li se poligoni ne preklapaju po y koordinati?
  3. Da li je P potpuno (u celosti) iza ravni kojoj pripada Q u odnosu na posmatrača?
  4. Da li je Q potpuno sa iste strane ravni koja sadrži P, kao i posmatrač?
  5. Da li se projekcije poligona na XY-ravan ne preklapaju? (ovo se utvrđuje poređenjem njihovih ivica)

* Razmatramo poligon P (tekući za rasterizaciju) i sve poligone Q (potencijalno ispred P, tj bliži posmatraču) koji se po Z koordinatama preklapaju sa P.

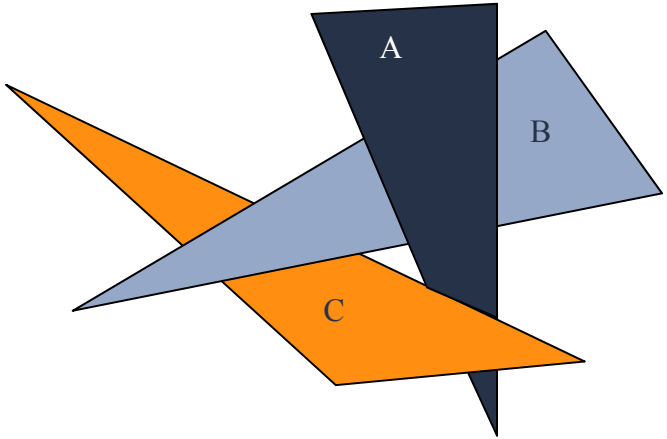
Slikarev algoritam (*depth-sort*)

- Ako bilo koji od prethodnih testova uspe, P može da se iscrta.
- Ako svih 5 testova ne uspe, potencijalno P prekriva Q, pa treba proveriti da li se Q može crtati pre P.
- Testovi 3 i 4 se ponavljaju, ali P i Q menjaju uloge.
- Ako je neki od prethodnih testova zadovoljen, P i Q menjaju mesta u listi i Q se prvi iscrtava.
- Ako ni ovi testovi ne budu zadovoljeni, jedan od poligona mora biti podeljen na dva dela, na osnovu ravni koja sadrži drugi poligon (izvorni poligon se uklanja iz liste, a dva novoformirana dodaju).

Slikarev algoritam (*depth-sort*)

Problem:

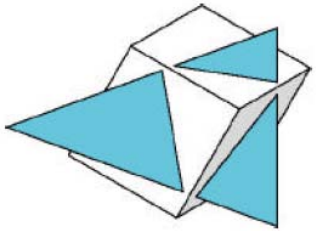
U slučaju uzajamnog preklapanja (npr. A,B i C na slici) – dolazi do beskonačne petlje.



Slikarev algoritam (*depth-sort*)

Rešenje:

Slučaj uzajamnog preklapanja se otkriva tako što se:



- Pri prebacivanju Q ispred P u listi, Q se obeleži.
- Pri svakom prebacivanju vrši se provera da li je poligon Q već bio obeležen (prebačen).
- Ako se otkrije da je Q već bio obeležen - detektuje se uzajamno preklapanje.
- U slučaju detektovanog preklapanja poligon Q se deli na 2 (Q1 i Q2) pomoću ravni poligona P.

Back face culling algoritam

- Ideja: kod neprozirnih zatvorenih objekata eliminišu se zadnje stranice koje ne mogu da se vide direktno.
- Test: stranica je vidljiva ako i samo ako je:

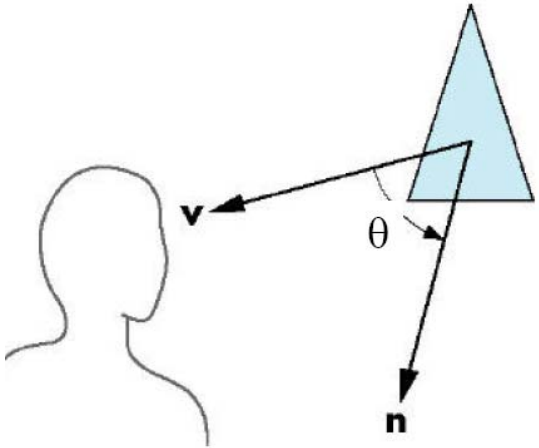
$$90 \geq \theta \geq -90$$

- Ovaj uslov je ekvivalentan sa:

$$\cos\theta \geq 0$$

- Ili:

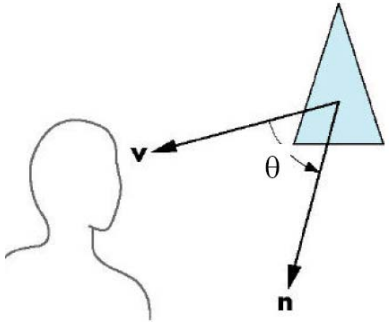
$$\mathbf{v} \bullet \mathbf{n} \geq 0$$



Back face culling algoritam

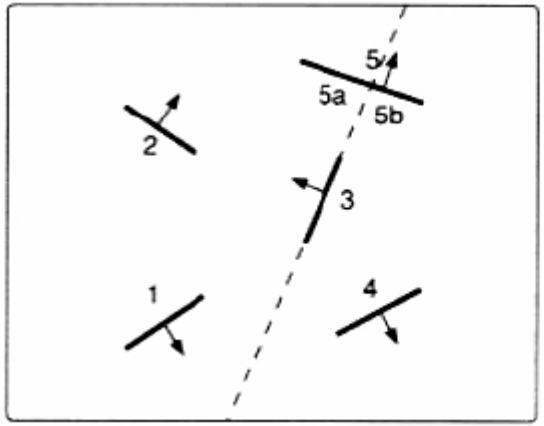
- Na ovaj način se obično uklanja oko 50% stranica.
- Koristi se u OpenGL-u:

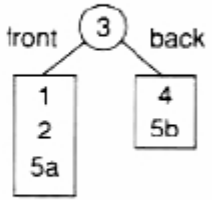
```
glEnable(GL_CULL_FACE)
glCullFace( GL_FRONT / GL_BACK / GL_FRONT_AND_BACK )
```



Algoritam koji koristi BSP stabla

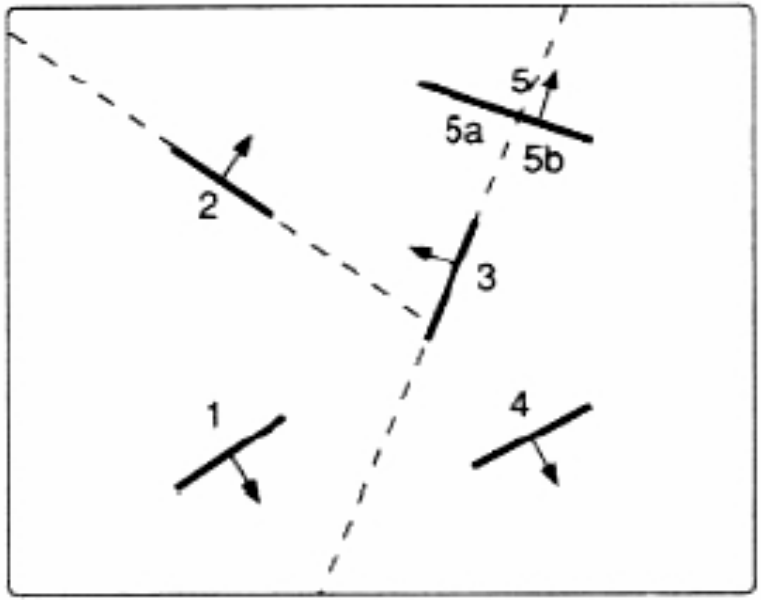
- **BSP** – **B**inary **S**pace **P**artitioning trees.
- Prostor poligona se deli ravnima kojima poligoni pripadaju. Izabere se jedan poligon koji će biti koren stabla, a zatim se formira stablo u zavisnosti da li su poligoni ispred ili iza ravni kojoj pripada startni poligon.

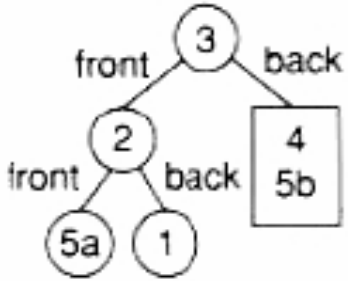




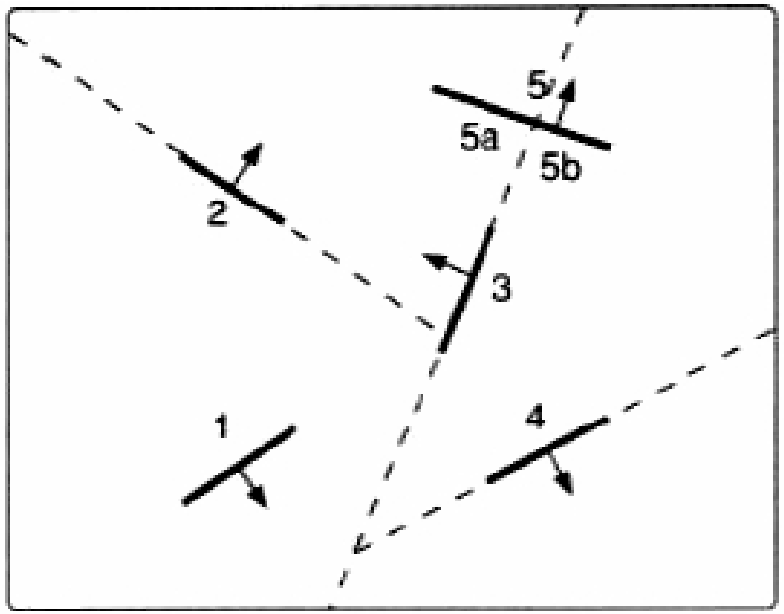
Poligon 3 je koren stabla

Algoritam koji koristi BSP stabla

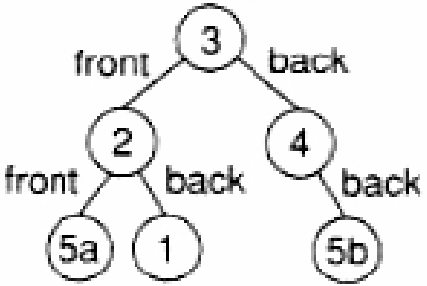




Algoritam koji koristi BSP stabla

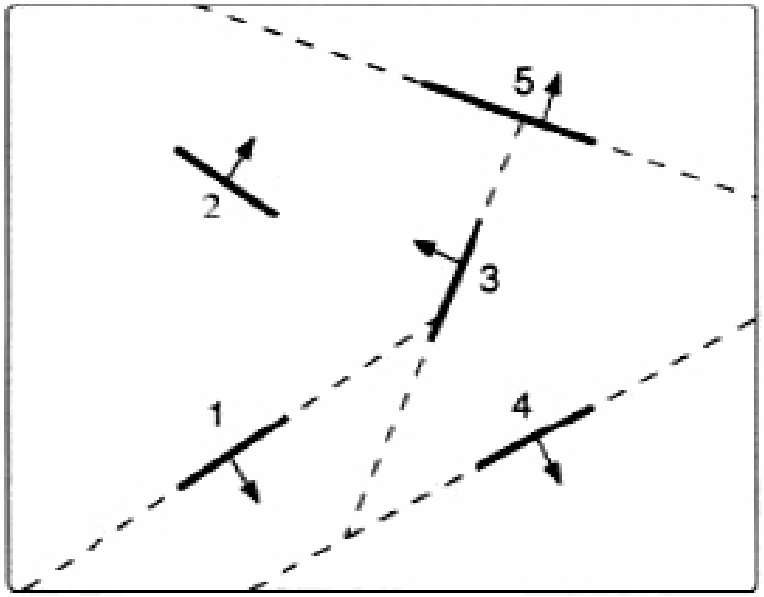


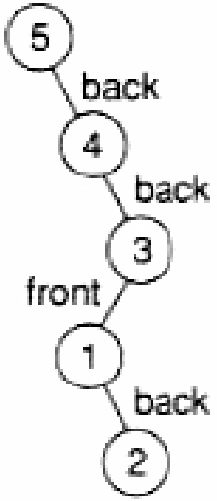
Kompletno stablo



Algoritam koji koristi BSP stabla

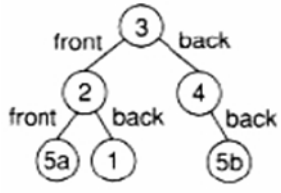
Alternativno stablo – izbegava se podela poligona, ali stablo nije balansirano

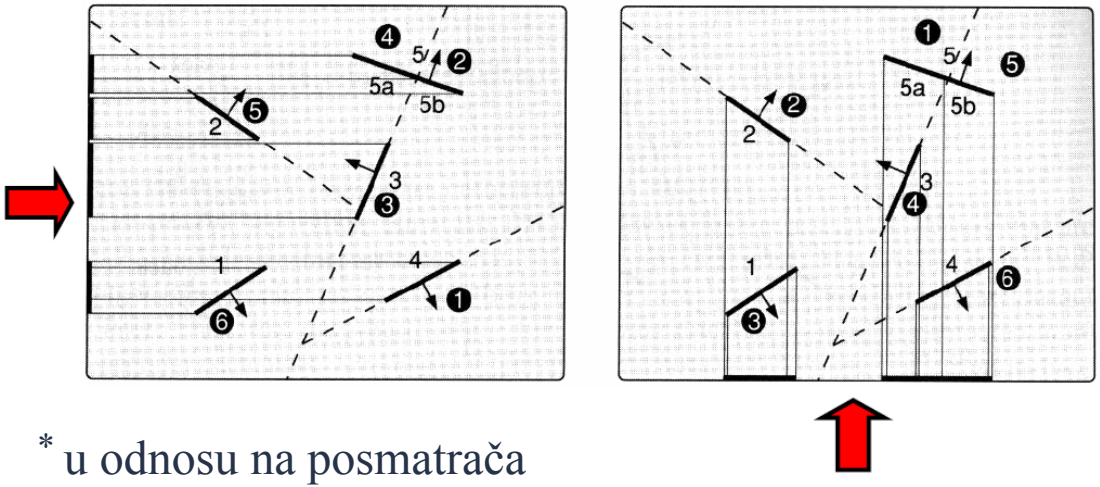




Algoritam koji koristi BSP stabla

- Obilazak stabla (rekurzija):
  - Prvo se crtaju poligoni iza* ravni korena stabla.
  - Zatim se crta poligon koji je koren stabla.
  - Onda se crtaju poligoni ispred* ravni korena stabla.

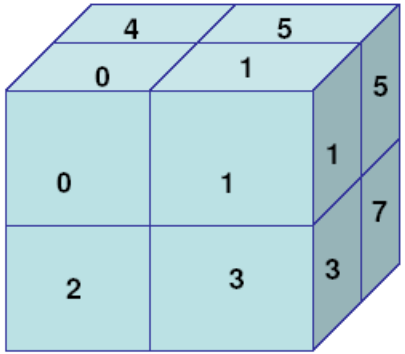


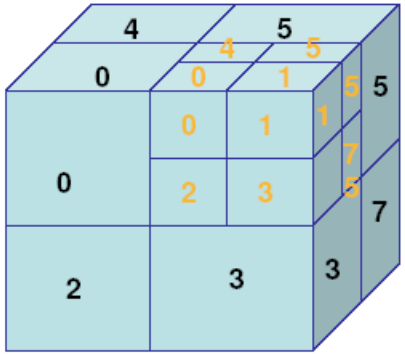


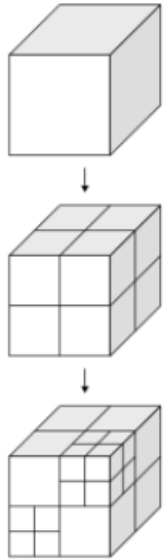
* u odnosu na posmatrača

Algoritam koji koristi *octree* stabla

- Prostor se deli u 8 oktanata.
- Pravi se stablo koje sadrži poligone u svakom oktantu.
- Ako jedan oktant sadrži više od maksimalnog broja poligona, on se rekurzivno deli na još 8 oktanata.

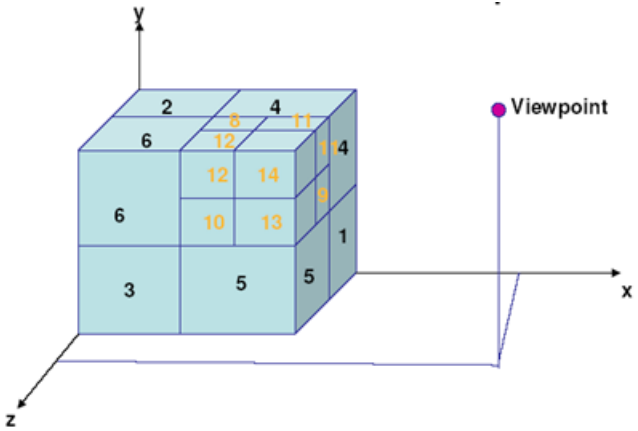






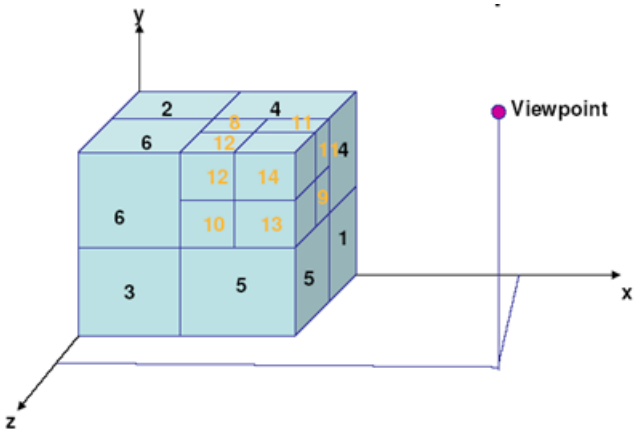
Algoritam koji koristi *octree* stabla

- Prvo se iscrtavaju poligoni koji se nalaze u oktantu koji je najudaljeniji od posmatrača (na slici 0).
- Zatim se iscrtavanje rekurzivno ponavlja za tri oktanta koji se graniče sa iscrtanim oktantom (imaju zajedničku stranica, na slici 1, 2 i 3).



Algoritam koji koristi *octree* stabla

- Zatim se iscrtavanje rekurzivno ponavlja za naredna tri oktanta koji se graniče sa prethodnim (na slici 4, 5 i 6).
- Na kraju se rekurzivno iscrtava najbliži oktant (na slici 7).



PITANJA