Računarska Grafika

Skripta sa odgovorima na pitanja

Nikola Lugić, 15193

Novembar 2017. - Februar 2018.

Sadržaj

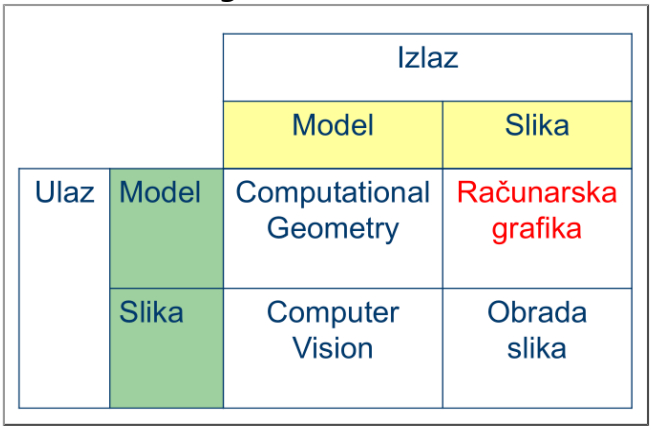
1. Šta je računarska grafika?

Računarska grafika je skup tehnika i sredstava čijom primenom se olakšava obrada grafičkih podataka.

Računarska grafika je grana računarske nauke koja se bavi sintezom računarskih slika stvarnih ili zamišljenih objekata.

Računarska grafika se bavi svim aspektima kreiranja slike pomoću računara: hardverom, softverom i aplikacijama.

2. Računarska grafika i srodne oblasti.



U zavisnosti od toga da li je ulaz/izlaz aplikacije model ili slika objekta, postoje 4 srodne oblasti:

1) Računarska geometrija[1]: transformiše modele objekata, bez rezultata u obliku slike.

2) Računarski vid: vrši prepoznavanje slike, tj. raščlanjivanje slike u upotrebljivi model.

3) Računarska grafika: na osnovu modela prikazuje sliku.

4) Obrada slika: transformiše slike objekata, ne koristi model objekta.

3. Podela aplikacija računarske grafike.

Aplikacije računarske grafike se mogu deliti po više kriterijuma:

1) Po dimenzionalnosti: dele se na 2D i 3D aplikacije.

2) Po tipu interakcije: dele se na aplikacije sa offline crtanjem (iz baze podataka), sa interaktivnim crtanjem (korisnik utiče na iscrtavanje), sa real-time animacijom i sa interaktivnim projektovanjem.

3) Po ulozi slike u aplikaciji: na aplikacije čiji je slika krajnji cilj i one kod kojih je slika samo deo jedne faze projektovanja.

4) Po vezi između objekta i njegovih slika: na aplikacije kod kojih se formira samo jedna slika, više povezanih slika, ili na one koje rade sa hijerarhijskom strukturom objekata.

4. Šta je to grafički sistem?

Grafički sistem predstavlja sponu između grafičke opreme i aplikativnog programa.

Grafički sistem je bilo koji skup hardverskih i softverskih entiteta, takvih da programerima pojednostavljuju upotrebu grafičkih ulaza i izlaza.

Grafički sistem je skup hardverskih i softverskih elemenata i njihovih veza, projektovan tako da zadovolji potrebe određene aplikacije za grafičkom komunikacijom.

Grafički sistem se obično sastoji iz skupa procedura koje odgovaraju različitim primitivama, atributima i ostalim elementima, koje su najčešće organizovane u grafičke biblioteke, takozvane grafičke API-je.

Funkcije grafičkog sistema: mora da obezbedi rad sa podacima, njihovo čuvanje i prikazivanje u obliku teksta i slike, da omogući korisniku da na prihvatljiv način prepozna i interpretira grafički prikazane podatke i da ih po potrebi menja i utiče na njihovo pojavljivanje.

---

1 Termin „*Computational Geometry*“ se može prevesti i kao „Geometrijski algoritmi“, imajući u vidu predmet istraživanja oblasti.

5. Šta je grafički API?

Grafički API (*Application Programming Interface*) je skup grafičkih funkcija organizovanih u jednu ili više grafičkih biblioteka koje predstavljaju interfejs između aplikacionog programa i grafičkog sistema.

Grafički API vrši apstrakciju detalja o hardverskoj i softverskoj implementaciji i daje programeru prikaz sistema kao „crne kutije".

6. Navesti osnovne grupe funkcija koje svaki API mora da ima.

1) Funkcije za primitive: funkcije za iscrtavanje osnovnih grafičkih primitiva (tačka, linija, poligon, piksel, tekst...).

2) Funkcije za kontrolu atributa primitiva: Primitive definišu šta će se nacrtati a njihovi atributi kako će se nacrtati (npr. boja/debljina/tip linije).

3) Funkcije pogleda: Omogućavaju formiranje 2D prikaza na osnovu 3D objekta (projekciju, određivanje koji objekti iz 3D sveta će se naći u 2D prikazu).

4) Funkcije za geometrijske transformacije: Omogućavaju 2D i 3D geometrijske transformacije nad objektima (translacija, rotacija, skaliranje...).

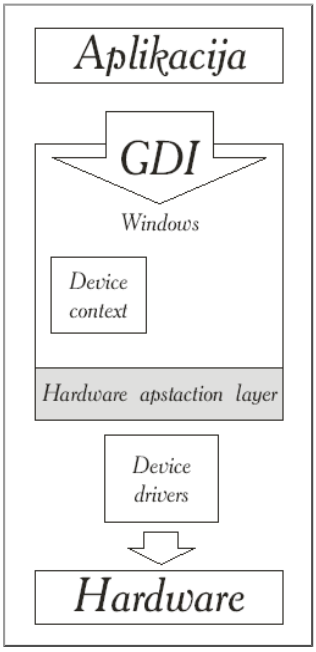
5) Funkcije za ulaz: Omogućavaju korisnicima da vrše unos podataka u aplikaciju putem delova grafičkog sistema (tastatura, miš, tabla, skener...).

6) Kontrolne funkcije: Omogućavaju komunikaciju sa operativnim sistemom pod kojim se grafička aplikacija izvršava (inicijalizacija, kontrola prozora...).

7) Ispitivačke funkcije: Omogućavaju dobijanje informacija o karakteristikama pojedinih komponenata grafičkog sistema (paleta boja, rezolucija...). Na ovaj način programeri dobijaju mogućnost da pišu device independent aplikacije koje ne zavise od grafičkog sistema na kome se izvršavaju.

7. Windows GDI.

GDI (*Graphics Device Interface*) je 2D grafički API koji Microsoft koristi u svojim operativnim sistemima. GDI enkapsulira svoje funkcije i grafičke objekte u MFC klase (CDC - *Device Context*, *CBrush* - četka, *CPen* - olovka, *CBitmap* - bitmapa...). Mesto GDI-ja u *Windows* operativnom sistemu dato je na slici pored.



Potencijalno podpitanje:

-Device Context.

Device Context je apstraktni (virtuelni) uređaj. Kada se programira, pristupa se ovom uređaju i na taj način se ne vodi računa direktno o hardveru i njegovim karakteristikama. O karakteristikama hardvera brigu vode drajveri uređaja.

8. Struktura OpenGL-a.

OpenGL (*Open Graphics Library*) je organizovan u nekoliko biblioteka:

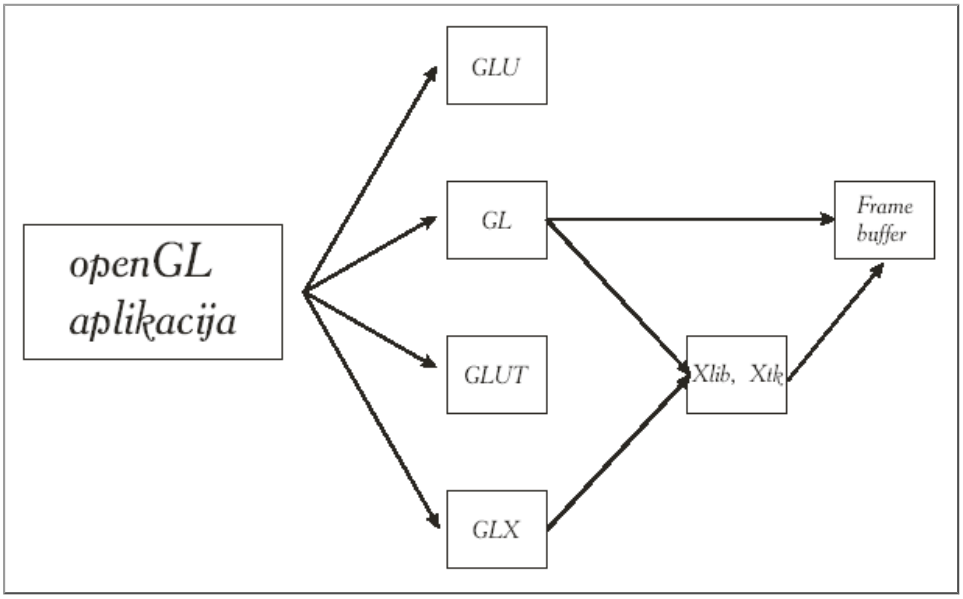
1) GL: jezgro OpenGL-a, osnovna biblioteka koja sadrži imena svih OpenGL funkcija.

2) GLU (*Graphics Utility Library*): biblioteka koja obezbeđuje potrebnu funkcionalnost OpenGL jezgra. Koristi samo GL funkcije i sadrži kod za kreiranje objekata koji se često koriste kao i procedure koje se često koriste.

3) GLUT (*GL Utility Toolkit*): biblioteka koja obezbeđuje minimum funkcionalnosti koja se očekuje od modernih sistema zasnovanih na prozorima.

4) GLX (WGL, AGL): služe za spregu sa operativnim sistemima. Korišćenjem ovih funkcija se gubi potreba da OpenGL aplikacija poziva direktno funkcije operativnog sistema. GLX je sprega sa *X Windows* sistemom, WGL sa *Windows*-om a AGL sa *Apple Mac OS*-om.

Organizacija OpenGL biblioteka prikazana je na slici pored.



9. Razlika između rasterskih i vektorskih podataka.

Rasterski podaci se pamte u obliku matrice čiji je svaki element jedan piksel zapamćene slike, dok se vektorski podaci pamte kao skup tačaka i primitiva na osnovu kojih se dobija zapamćena slika. Rasterski podaci predstavljaju „digitalni pogled" na izvorni dokument, i iz njih se direktno mogu izdvajati informacije korišćenjem neke od metoda za obradu signala. Takođe, oni se mogu kompresovati, što smanjuje potreban prostor za njihovo skladištenje. Nedostatak je to što se mora izvršavati dekompresija, što usporava prikaz. Prednost vektorskih podataka u odnosu na rasterske je u tome što pružaju mnogo više informacija o geometriji i topologiji, kao i to što prilikom uvećanja/umanjenja slika ne gubi kvalitet. Zato se vektorski podaci koriste za pamćenje crteža, dok se rasterski podaci koriste za pamćenje slike, videa itd., tamo gde njihova ograničena preciznost ne dolazi do izražaja.

10. Razlika između rasterskih i vektorskih uređaja.

Displej procesor vektorskih uređaja interpretira komande za iscrtavanje iz bafera (često se zove bafer za osvežavanje - *refresh buffer*) i šalje digitalne koordinate tačke generisanjem vektora koji se konvertuje u odgovarajući analogni signal, koji predstavlja napon potreban da bi se osvetlile tačke na ekranu. Zrak se kreće u onom redosledu i smeru u kome mu diktiraju koordinate iz bafera. Tehnika prikaza na vektorskim uređajima se naziva slučajnim skaniranjem (*random scan*) jer je svaka tačka na ekranu adresibilna.

Rasterski uređaji smeštaju podatke za prikazivanje u bafer za osvežavanje, pri čemu su primitive razložene na piksele. Kompletna slika na rasterskom uređaju se formira iz rastera, koji predstavlja niz horizontalnih rasterskih linija od kojih je svaka red individualnih piksela. Raster se pamti kao matrica piksela koja predstavlja celu površinu ekrana. Slika se skenira sekvencijalno od strane video kontrolera, liniju po liniju i od vrha ka dnu. Intenzitet mlaza reflektuje intenzitet svakog piksela.

11. Princip rada CRT monitora.

Princip rada CRT ekrana je sledeći: elektronski top emituje mlaz elektrona. Tanak mlaz elektrona se postiže sistemom za fokusiranje. Usmeravanje mlaza elektrona se postiže pomoću kalemova za skretanje prema određenoj tački na fosfornoj ploči. Pošto su elektroni dosta ubrzani, oni udaraju u fosfor ogromnom brzinom, pri čemu fosfor svetli. Pošto intenzitet svetlosti opada sa vremenom, tačku je potrebno pogađati više puta u sekundi, da bi se izbeglo treperenje. Osvežavanje cele slike je neophodno vršiti više puta u sekundi. Taj period određuje period osvežavanja slike (*refresh rate*), i on obično iznosi 60Hz (60 frejmova u sekundi). Kontrolni grid u katodnoj cevi služi da odredi broj

elektrona u mlazu. Što je negativniji napon kontrolnog grida, to je broj elektrona u mlazu manji. Na taj način se kontroliše intenzitet osvetljenja. Kod CRT ekrana u boji postoje 3 elektronska topa, po jedan za crvenu, plavu i zelenu boju. Slika se na ekranu iscrtava liniju po liniju, prateći „cik-cak" putanju.

12. Aspect Ratio.

Aspect Ratio (AR) je odnos horizontale i vertikale, tj. odnos između broja piksela po vertikali i horizontali ekrana. Nekada je obično iznosio 4:3, dok je današnji standard 16:9 (*widescreen*).

Računa se po obrascu $AR = AP/AL$, gde je AP broj piksela po dužini, a AL po širini ekrana.

13. Rezolucija.

Rezolucija ekrana jednaka je ukupnom broju piksela koji mogu biti prikazani na ekranu, tj. njegovoj aktivnoj oblasti. Računa se po obrascu $R = AP * AL^2$, gde je AP broj piksela po dužini, a AL po širini ekrana.

14. Princip rada LCD i plazma ekrana.

LCD ekrani prikazuju sliku zahvaljujući osobini dugih molekula kristala da pod uticajem električnog polja ne polarizuju svetlost, tako da ona ne može proći kroz horizontalnu polarizacijsku ploču, usled čega gledalac vidi crnu površinu. U odsustvu električnog polja, molekuli kristala su spiralno raspoređeni i rotiraju polarizaciju ulazne svetlosti za 90°, tako da ona može proći kroz horizontalnu polarizacijsku ploču i samim tim biti vidljiva gledaocu. Tačke na ekranu se adresiraju matrično, pomoču horizontalne i vertikalne rešetke - tačka je tamna ako joj se preko vertikalne rešetke dovede pozitivan, a preko horizontalne negativan napon. Kombinacija pozitivnog napona sa desne, i negativnog napona sa leve strane kristalnog sloja ređa kristale na tom mestu.

Plazma ekrani su sačinjeni od malih neonskih sijalica koje se matričnim adresiranjem preko horizontalne i vertikalne rešetke mogu selektivno paliti i gasiti. Uz dovoljno veliku razliku napona na horizontalnoj i vertikalnoj rešetki, elektroni iz molekula neona se oslobađaju i sijalica svetli. Za održanje ovog stanja dovoljan je i nešto niži napon, a ono se menja kada napon padne ispod nužne granične vrednosti održavanja.

15. Prednosti i mane LED i OLED ekrana.

Prednosti LED ekrana u odnosu na standardne LCD ekrane su sledeće:

1) 45% manja potrošnja električne energije.
2) Znatno tanji ekrani.
3) Svetlija slika uz tamniju crnu boju.
4) Uz korišćenje RGB-LED tehnologije može se dobiti znatno širi gamut boja.
5) Bolja kontrola pojedinačnih piksela na ekranu.

Mane LED ekrana u odnosu na standardne LCD ekrane su sledeće:

1) Veća potrošnja energije u određenim modovima rada (npr. *local dimming*, gde se delovi ekrana zatamnjuju radi boljeg prikaza tamnijih scena).
2) Viša cena.

Prednosti OLED ekrana u odnosu na standardne LCD, kao i LED ekrane su sledeće:

1) Znatno manja potrošnja električne energije, usled nedostatka potrebe za pozadinskim osvetljenjem.
2) Tanji su od drugih vrsta LCD ekrana.
3) Postoji mogućnost izrade savitljivih OLED ekrana.

---

2 U praksi se označava sa „ $AP\ x\ AL$ ", tj. bez prikazivanja ukupnog broja piksela.

Mane OLED ekrana u odnosu na standardne LCD, kao i LED ekrane su sledeće:

1) Slabije osvetljenje nego kod LED ekrana.
2) Znatno viša cena.

16. Prednosti i mane LCD i DLP projektora.

Prednosti LCD projektora su sledeće:

1) Tehnologija je potpuno zrela.
2) Reprodukcija boja je vrlo kvalitetna.
3) Tehnologija izrade je relativno jeftina.

Mane LCD projektora su sledeće:

1) Velike dimenzije.
2) Nemogućnost postizanja visokog kontrasta.
3) Degradacija kvaliteta slike posle nekog vremena (3-4 godine).

Prednosti DLP projektora su sledeće:

1) Bolji kontrast.
2) Manje dimenzije projektora.

Mane DLP projektora su sledeće:

1) Visoka cena.
2) Nešto veća buka, zbog neprekidnog okretanja diska sa filterima.

17. Tehnologije projektora.

Osnovne tehnologije za izradu video projektora su LCD i DLP.

LCD projektori koriste 3 odvojena TFT ekrana koji, u kombinaciji sa optičkim elementima, slažu crvenu zelenu i plavu komponentu slike u celinu, koja zatim biva projektovana na platnu.

DLP (*Digital Light Processing*) projektori koji u procesu formiranja slike koriste minijaturna ogledala. Srce DLP projektora čini čip koji za svaki piksel koji projektor može da prikaže ima minijaturno ogledalo u obliku romba, koje može da se okreće oko duže dijagonale. Menjanjem položaja svakog od ogledala postiže se manje ili veće osvetljenje u toj tački. Reprodukcija boja postiže se uz pomoć rotirajućeg diska sa obojenim filterima koji se nalazi između čipa i objektiva i okreće brzo (50-100 obrtaja u sekundi), tako da se dobija vrlo stabilna slika (u rangu sa LCD projektorima).

18. Tehnologije štampača.

Postoji 6 osnovnih vrsta štampača:

1) Matrični štampači: Štampaju pomoću pokretne glave sa iglicama koje ostavljaju trag na papiru udarcima preko trake sa bojom (mastilom).

2) *Ink-jet* štampači („pljuckavci"): Štampaju tako što izbacuju mlaz mastila pod pritiskom kroz tanke cevčice[3], koje se u dodiru sa papirom hladi i lepi za njega.

3) Laserski štampači: Štampaju tako što laserom naelektrišu valjak koji se okreće, pri čemu se toner (prah) zalepi za valjak, koji prelazi preko papira i ostavlja trag.

4) Termalni štampači: Štampaju tako što zagrevaju boju, koja prelazi na papir. Za ovo se mora koristiti specijalni papir.

5) Ploteri: Posebne vrste štampača koje se koriste za štampanje većih formata, i to tako što iscrtavaju sliku pokretanjem pera preko površine papira. Pero se može pokretati u svim pravcima, ili samo duž jedne ose, pri čemu papir rotira po drugoj osi.

6) 3D štampači: Uređaji koji kreiraju 3D model od veštačkog materijala, njegovim sukcesivnim nanošenjem u slojevima.

3 Ove cevčice su više nego duplo tanje od ljudske dlake.

19. VR tehnologije.

VR uređaji su izlazni uređaji koji stimulisanjem ljudskih čula na određen način pokušavaju da stvore utisak prisustva u nekom okruženju koje realno ne postoji. U VR uređaje spadaju VR kacige, rukavice, 3D ekrani i projektori i 3D holografi.

VR kacige su specijalizovane kacige kod kojih se emituje stereo pogled, korišćenjem dva ekrana koji prikazuju dve sinhronizovane slike na malom rastojanju od oka posmatrača. Zahvaljujući tome što ove dve slike zapravo predstavljaju isto okruženje iz dva blago različita ugla, postiže se efekat dubine (3D efekat) kod posmatrača.

VR rukavice su specijalizovane rukavice koje mogu služiti kao ulazni ili izlazni uređaj. Kao izlazni uređaj najčešće se sreću u formi haptičkog *feedback*-a, odnosno uređaja koji stimulišu čulo dodira. Ovime se postiže reakcija virtuelnog okruženja na dodir koja je u velikoj meri verodostojna reakcijama u realnom okruženju.

20. Tehnike prikazivanja i gledanja 3D videa.

1) 3D pomoću naočara/kaciga za svako oko: Jedan ekran prikazuje sliku koju vidi jedno oko, čime se postiže da svako oko vidi sliku „iz svog ugla“, što daje utisak dubine i osećaj 3D prostora.

2) 3D pomoću crveno-plavih naočara: Jedno oko vidi samo plavu boju kroz crveno staklo, dok drugo oko vidi samo crvenu kroz plavo staklo. Dva projektora emituju sliku - jedan u crveno-beloj boji (za oko iza plavog stakla), a drugi u plavo-beloj (za oko iza crvenog stakla).

3) 3D pomoću naočara sa polarizovanim staklima: Slično kao sa crveno-plavim naočarima, samo što je ovde jedno staklo vertikalno polarizovano i kroz njega se vidi samo slika projektora koji emituje vertikalno polarizovanu svetlost, a drugo staklo horizontalno polarizovano, tako da se kroz njega vidi samo horizontalno polarizovana svetlost drugog projektora.

4) *Active Shutter* 3D tehnika: Ekran visoke frekvencije osvežavanja[4] emituje naizmenično sliku za levo i desno oko, a specijalizovane naočare sinhronizovane sa njime zatvaraju pogled onom oku koje u tom trenutku ne bi trebalo da vidi sliku. Naočare zapravo imaju LCD ekrane umesto stakala, i naizmenično prikazuju sliku (ne provide se) ili ne prikazuju ništa (provide se).

5) 3D pomoću TV ekrana koji ne zahtevaju naočare: Nova (i ne sasvim jasna) tehnologija 3D prikaza reklamirana od strane brojnih proizvođača. Ukoliko se slika gleda pod određenim uglom u odnosu na ekran, može se ostvariti 3D efekat i bez korišćenja naočara, pri čemu postoje brojna ograničenja (obično postoji 8 uglova pod kojima je moguće ostvariti 3D prikaz).

21. Šta je skaniranje?

Skaniranje[5] (*scan conversion*) je proces prevođenja kontinualnih opisa grafičkih primitiva u diskretne opise (u skup piksela).

22. „Naivni“ algoritam za skaniranje linije.

„Naivni“ algoritam liniju predstavlja svim onim pikselima koje matematička linija seče. Ne daje tako dobre rezultate i dosta je spor zbog računanja preseka matematičke linije i piksela.

23. Nagibni algoritam za skaniranje linije. [primer/zadatak]

Koristi matematičku definiciju linije - jednačinu prave $y = kx + n$, gde je k koeficijent pravca (nagib). Algoritam je dosta spor zbog operacija množenja i deljenja sa realnim brojevima, a u zavisnosti od nagiba, kao i od toga vršimo li skaniranje po x ili y osi, može se dobiti različito osvetljena linija.

```
void nagibniAlgoritam(int x0, int y0,
    int x1, int y1, COLORREF color)
{
    float k = (y1 - y0) / (x1 - x0);
    float n = y1 - k * x1;
    for (int x = x0; x < x1; ++x)
    {
        float y = k * x + n;
        WritePixel(x, ceil(y), color);
    }
}
```

4 Frekvencija osvežavanja mora biti duplo viša od broja frejmova u sekundi (obično 120+ Hz).

5 Rasterizacija je termin koji se češće sreće u literaturi.

24. Inkrementalni algoritam za skaniranje linije. [primer/zadatak]

```
void DDAAlgoritam(int x0, int y0,
    int x1, int y1, COLORREF color)
{
    float k = (y1 - y0) / (x1 - x0);
    float y = y0;
    for (int x = x0; x <= x1; ++x, y += k)
        WritePixel(x, ceil(y), color);
}
```

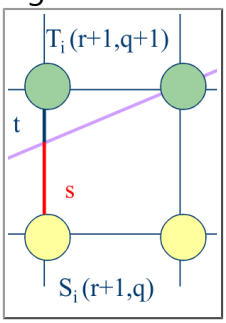
Inkrementalni algoritam (DDA - Digitalni Diferencijalni Analizator) je algoritam za skaniranje linije koji eliminiše množenje tako što iterativno računa vrednosti y koordinata.

$dx = 1 \Rightarrow y_{i+1} = y_i + k$

$y_{i+1} = kx_{i+1} + n = k(x_i + dx) + n = kdx + kx_i + n = y_i + kdx$

25. Bresenhamov algoritam za skaniranje linije. [primer/zadatak]

Tehnika Bresenhamovog algoritma se bazira na odlučivanju treba li svaki put pri inkrementiranju x koordinate inkrementirati i y koordinatu. Kompletna računica svodi se na celobrojnu aritmetiku, što ovaj algoritam čini najbržim.



$$s = \frac{dy}{dx}(r+1) - q \quad t = q + -\frac{dy}{dx}(r+1)$$

$$s - t = 2\left(\frac{dy}{dx}\right) \cdot (r+1) - 2q - 1$$

$$dx(s-t) = 2(rdy - qdx) + 2dy - dx = d_i$$

$r = x_{i-1} \wedge q = y_{i-1} \Rightarrow d_i = 2x_{i-1}dy - 2y_{i-1}dx + 2dy - dx$

$d_{i+1} = 2x_i dy - 2y_i dx + 2dy - dx$

$d_{i+1} - d_i = 2dy(x_i - x_{i-1}) - 2dx(y_i - y_{i-1})$

$x_i - x_{i-1} = 1 \Rightarrow d_{i+1} = d_i + 2dy - 2dx(y_i - y_{i-1})$

```
void bresenhamAlgoritam(int x0, int y0,
    int x1, int y1, COLORREF color)
{
    int dx = abs(x1 - x0);
    int dy = abs(y1 - y0);
    int d = 2 * dy - dx;
    int incr1 = 2 * dy, incr2 = 2 * (dy - dx);

    int x = x0, y = y0, xend = x1;
    if (x0 > x1)
    { x = x1; y = y1; xend = x1; }

    WritePixel(x, y, color);
    while (x < xend)
    {
        ++x;
        if (d < 0)
            d += incr1;
        else
        {
            ++y;
            d += incr2;
        }
        WritePixel(x, y, color);
    }
}
```

26. Trigonometrijski (trivijalni) algoritam za skaniranje kružnice. [primer/zadatak]

```
void trigAlgoritam(int r, COLORREF color)
{
    for (float alfa = 0.0f; alfa < 6.28f;
            alfa += 6.28f / 360)
    {
        int x = ceil(r * cos(alfa));
        int y = ceil(r * sin(alfa));
        WritePixel(x, y, color);
    }
}
```

Trivijalni algoritam radi na osnovu parametarskih jednačina kružnice, tj. na osnovu trigonometrijskih funkcija. Pretpostavka je da se crta kružnica u koordinatnom početku. Glavni problemi algoritma su sporost (usled korišćenja trigonometrijskih funkcija), kao i to što je korak α direktno proporcionalan poluprečniku, tako da se sa povećanjem poluprečnika kruga dobija lošija slika.

Ovaj algoritam je moguće donekle unaprediti zahvaljujući visokoj simetričnosti kruga. Računaju se koordinate tačaka samo za jedan oktant, a svaka tačka se crta 8 puta (uz promenu znaka/zamenu mesta koordinata).

27. Polinomni algoritam za skaniranje kružnice. [primer/zadatak]

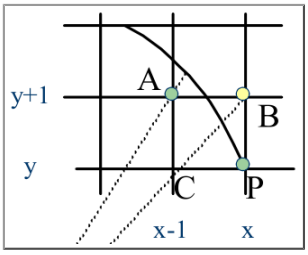
```
void polinomAlgoritam(int r, COLORREF color)
{
    for (int x = 0, int xend = ceil(r / sqrt(2));
            x <= xend; ++x)
    {
        int y = ceil(sqrt(r * r - x * x));
        WritePixel8(x, y, color);
    }
}
```

Polinomni algoritam radi na osnovu jednačine kruga $x^2 + y^2 = R^2$. Veoma je spor zbog aritmetike sa realnim brojevima i računanja korena, a rezultat koji daje nije uvek zadovoljavajući.

28. Bresenhamov algoritam za skaniranje kružnice. [primer/zadatak]

Bresenhamov algoritam se svodi na odlučivanje o izboru između dve kandidat-tačke (A i B) između kojih prolazi kružna linija. Posmatra se luk u prvom oktantu koji počije tačkom $(R, 0)$ i napreduje surotno kretanju kazaljke na satu, do ugla 45°. Algoritam je veoma brz, zahvaljujući tome što koristi samo celobrojnu aritmetiku.

```
void bresenhamAlgoritam(int r, COLORREF color)
{
    int x = r, y = 0, d = 3 - 2 * r;
    while (x > y)
    {
        WritePixel8(x, y, color);
        if (d < 0)
            d += 4 * y++ + 6;
        else
            d += 4 * (y++ - x--) + 10;
    }
    if (x == y)
        WritePixel8(x, y, color);
}
```



Uvodi se pojam kvadratne greške u tački K, $e(K) = x_k^2 + y_k^2 - R^2$, pri čemu važi $e(K) > 0$ za K izvan kruga, $e(K) = 0$ za K na kružnici i $e(K) < 0$ za K unutar kruga.

U našem slučaju algebarski zbir grešaka je $d = e(A) + e(B)$. Za A i B unutar kruga[6] važi $d < 0$ i bira se B. Za A i B izvan kruga[6] važi $d > 0$ i bira se A. Situacija je složenija ako je A unutar, a B izvan kruga.

Za $|e(A)| > |e(B)| \Rightarrow d < 0$ se bira B, a za $|e(A)| < |e(B)| \Rightarrow d > 0$ se bira A.

Za $d_i > 0$ (A): $d_{i+1} = (x-2)^2 + (y+2)^2 - R^2 + (x-1)^2 + (y+2)^2 - R^2$

$d_{i+1} = 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y + 10 - 4x + 4y = d_i + 4(y - x) + 10$

Za $d_i < 0$ (B): $d_i = (x-1)^2 + (y+1)^2 - R^2 + x^2 + (y+1)^2 - R^2 = 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y$

$d_0 = d_i|_{R,0} = 3 - 2R$

$d_{i+1} = (x-1)^2 + (y+2)^2 - R^2 + x^2 + (y+2)^2 - R^2 = 2x^2 + 2y^2 - 2R^2 + 3 - 2x + 4y + 6 + 4y = d_i + 4y + 6$

29. Trigonometrijski algoritam za skaniranje elipse. [primer/zadatak]

```
void trigAlgoritam(int a, int b, COLORREF color)
{
    float t = 0.0f, tend = M_PI_2,
          tstep = 2 * M_PI / 360;
    while (t < tend)
    {
        int x = ceil(a * cos(t));
        int y = ceil(b * sin(t));
        WritePixel4(x, y, color);
        t += tstep;
    }
}
```

Trigonometrijski algoritam radi na osnovu parametarskih jednačina elipse, tj. na osnovu trigonometrijskih funkcija. Pretpostavka je da se crta elipsa sa centrom u koordinatnom početku. Ovaj algoritam je dosta spor zbog trigonometrijskih funkcija, a za veće elipse se može dobiti loš kvalitet prikaza, zbog zavisnosti koordinata od ugla α.

$x = a\cos\alpha \qquad y = b\sin\alpha \qquad \alpha \in (0, 2\pi)$

30. Polinomni algoritam za skaniranje elipse. [primer/zadatak]

Polinomni algoritam radi na osnovu jednačine elipse $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$. Veoma je spor zbog aritmetike sa realnim brojevima i računanja korena, a rezultat koji daje nije uvek zadovoljavajući. $y = \sqrt{b^2 \cdot \left(1 - \frac{x^2}{a^2}\right)}$

```
void polinomAlgoritam(int a, int b, COLORREF color)
{
    for (int x = 0, float a2 = a * a; x < a; ++x)
    {
        int y = ceil(b * sqrt(1 - x * x / a2));
        WritePixel4(x, y, color);
    }
}
```

6 Moguće samo u teoriji.

31. Diferencijalni algoritam za skaniranje elipse I reda. [primer/zadatak]

```
void diffIAlgoritam(float a, float b, COLORREF color)
{
    float ba = M_PI / 180.0f * b / a,
        ab = M_PI / 180.0f * a / b, x0 = a, y0 = 0;
    for (int j = 0; j < 90; ++j)
    {
        float x1 = x0 - ab * y0;
        float y1 = y0 + ba * x0;
        bresenhamLine(x0, y0, x1, y1, color);
        bresenhamLine(-x0, y0, -x1, y1, color);
        bresenhamLine(x0, -y0, x1, -y1, color);
        bresenhamLine(-x0, -y0, -x1, -y1, color);
        x0 = x1;
        y0 = y1;
    }
}
```

Diferencijalni algoritam I reda se bazira na prvom izvodu parametarskih jednačina elipse. Njegov osnovni nedostatak je taj što se ne završava tačno u $(0, b)$, zbog toga što je diferencijalna razlika procenjivana samo na osnovu dve zadnje tačke x_i i x_{i-1}.

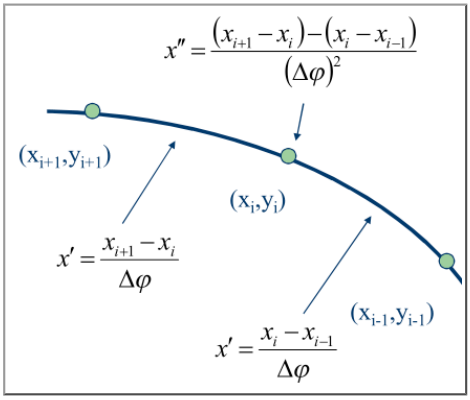
$$\left.\begin{aligned} x &= a\cos\phi \\ y &= b\sin\phi \end{aligned}\right\} \Rightarrow x' = -a\sin\phi = -\frac{a}{b}\cdot y,\ y' = b\cos\phi = \frac{b}{a}\cdot x$$

$$\frac{\Delta x}{\Delta\phi} = x' = \frac{x_i - x_{i-1}}{\Delta\phi} = -\frac{a}{b}\cdot y_{i-1} \Rightarrow \begin{cases} x_i = x_{i-1} - \frac{a}{b}\cdot y_{i-1}\cdot\Delta\phi \\ y_i = y_{i-1} - \frac{b}{a}\cdot x_{i-1}\cdot\Delta\phi \end{cases}$$

32. Diferencijalni algoritam za skaniranje elipse II reda. [primer/zadatak]

```
void diffIIAlgoritam(float a, float b, COLORREF color)
{
    float k = M_PI / 180.0f, x0 = a, x1 = a * cos(k),
        y0 = 0.0f, y1 = b * sin(k);
    k = 2 - k * k;
    for (int j = 0; j < 90; ++j)
    {
        bresenhamLine(x0, y0, x1, y1, color);
        bresenhamLine(-x0, y0, -x1, y1, color);
        bresenhamLine(x0, -y0, x1, -y1, color);
        bresenhamLine(-x0, -y0, -x1, -y1, color);
        x0 = x1;
        y0 = y1;
        x1 = k * x1 - x0;
        y1 = k * y1 - y0;
    }
}
```

Diferencijalni algoritam II reda se bazira na drugom izvodu parametarskih jednačina elipse. Prednost mu je što iterativna izračunavanja ne zavise od oblika elipse (tj. od parametara a i b). Takođe, ovaj algoritam rešava problem diferencijalnog algoritma I reda, zahvaljujući računanju na osnovu tri tačke.



$$x = a\cos\phi \quad y = b\sin\phi \quad x'' = -a\cos\phi = -x$$

$$x'' = \frac{x_{i+1} - 2x_i + x_{i-1}}{(\Delta\phi)^2} = -x_i \Rightarrow \begin{cases} x_{i+1} = (2 - (\Delta\phi)^2)\cdot x_i - x_{i-1} \\ y_{i+1} = (2 - (\Delta\phi)^2)\cdot y_i - y_{i-1} \end{cases}$$

33. Rekurzivni floodfill algoritam za ispunu 4-susednih/8-susednih oblasti. [primer/zadatak]

```
void floodFill4(int x, int y, COLORREF oldColor,
    COLORREF newColor)
{
    if (ReadPixel(x, y) == oldColor)
    {
        WritePixel(x, y, newColor);
        floodFill4(x, y - 1, oldColor, newColor);
        floodFill4(x, y + 1, oldColor, newColor);
        floodFill4(x + 1, y, oldColor, newColor);
        floodFill4(x - 1, y, oldColor, newColor);
    }
}
```

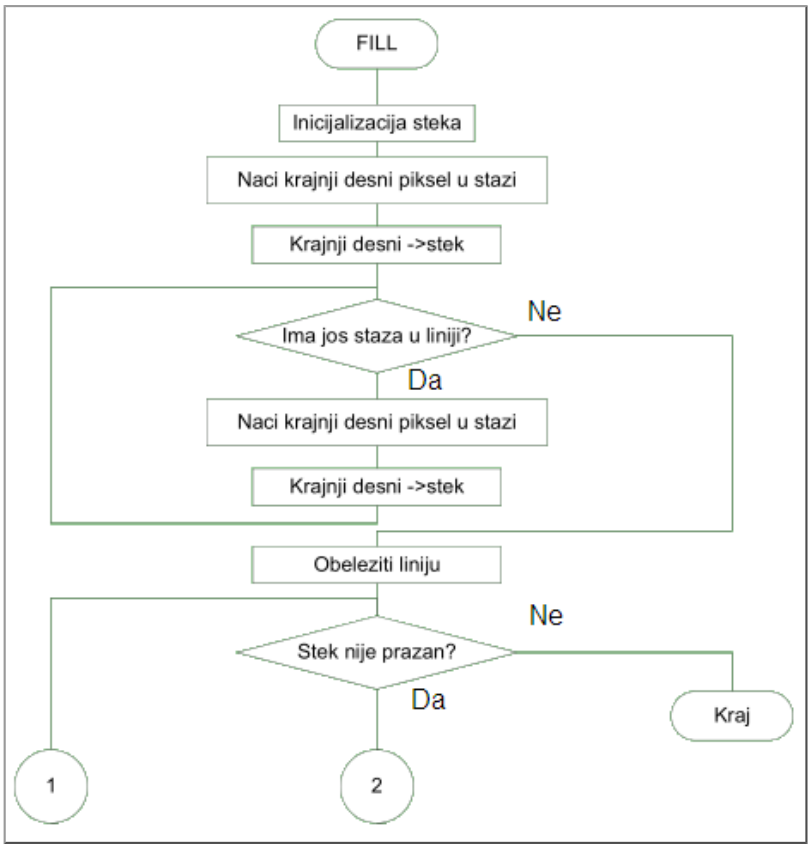
```
void floodFill8(int x, int y, COLORREF oldColor,
    COLORREF newColor)
{
    if (ReadPixel(x, y) == oldColor)
    {
        WritePixel(x, y, newColor);
        floodFill4(x, y - 1, oldColor, newColor);
        floodFill4(x, y + 1, oldColor, newColor);
        floodFill4(x + 1, y, oldColor, newColor);
        floodFill4(x - 1, y, oldColor, newColor);
        floodFill4(x - 1, y - 1, oldColor, newColor);
        floodFill4(x - 1, y + 1, oldColor, newColor);
        floodFill4(x + 1, y - 1, oldColor, newColor);
        floodFill4(x + 1, y + 1, oldColor, newColor);
    }
}
```

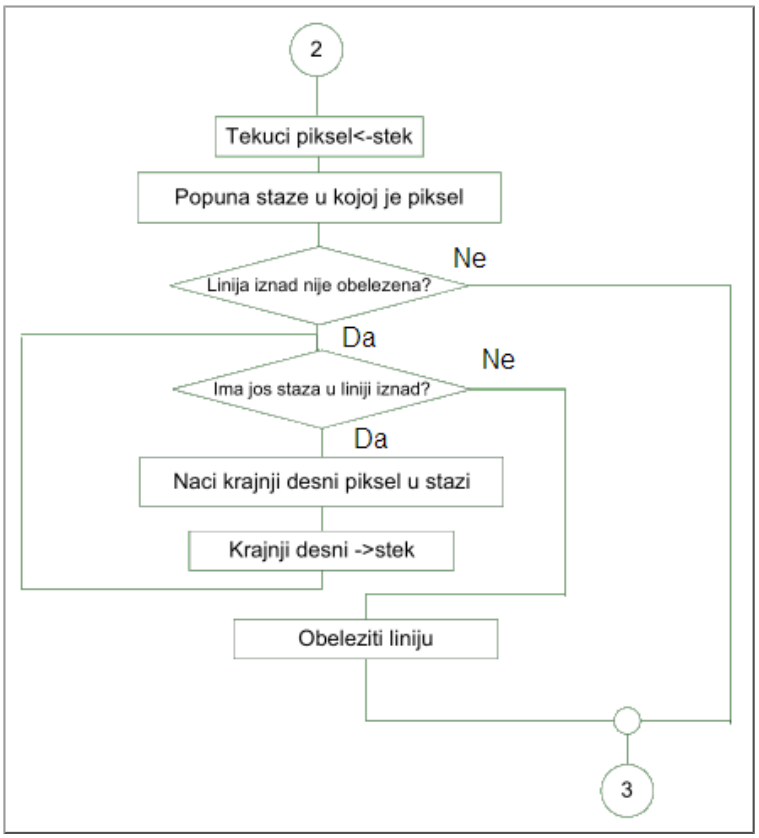
34. Rekurzivni algoritam za ispunu oivičenih 4-susednih/8-susednih oblasti. [primer/zadatak]

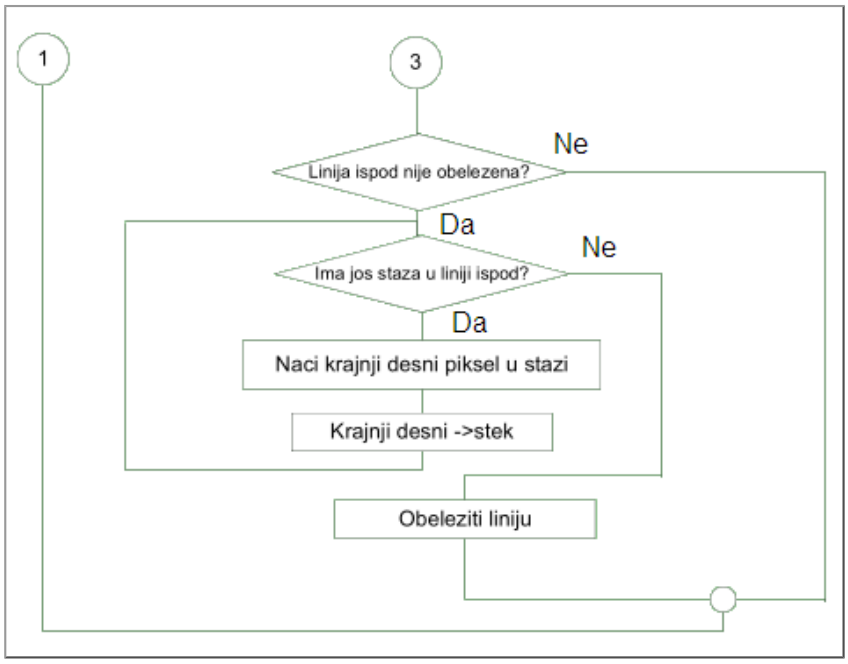
```
void boundaryFill4(int x, int y, COLORREF boundColor,
    COLORREF newColor)
{
    COLORREF temp = ReadPixel(x, y);
    if (temp != boundColor && temp != newColor)
    {
        WritePixel(x, y, newColor);
        boundaryFill4(x, y - 1, boundColor, newColor);
        boundaryFill4(x, y + 1, boundColor, newColor);
        boundaryFill4(x + 1, y, boundColor, newColor);
        boundaryFill4(x - 1, y, boundColor, newColor);
    }
}
```

```
void boundaryFill8(int x, int y, COLORREF boundColor,
    COLORREF newColor)
{
    COLORREF temp = ReadPixel(x, y);
    if (temp != boundColor && temp != newColor)
    {
        WritePixel(x, y, newColor);
        boundaryFill8(x, y - 1, boundColor, newColor);
        boundaryFill8(x, y + 1, boundColor, newColor);
        boundaryFill8(x + 1, y, boundColor, newColor);
        boundaryFill8(x - 1, y, boundColor, newColor);
        boundaryFill8(x - 1, y - 1, boundColor, newColor);
        boundaryFill8(x - 1, y + 1, boundColor, newColor);
        boundaryFill8(x + 1, y - 1, boundColor, newColor);
        boundaryFill8(x + 1, y + 1, boundColor, newColor);
    }
}
```

35. Iterativni algoritam za ispunu oblasti. [primer/zadatak]





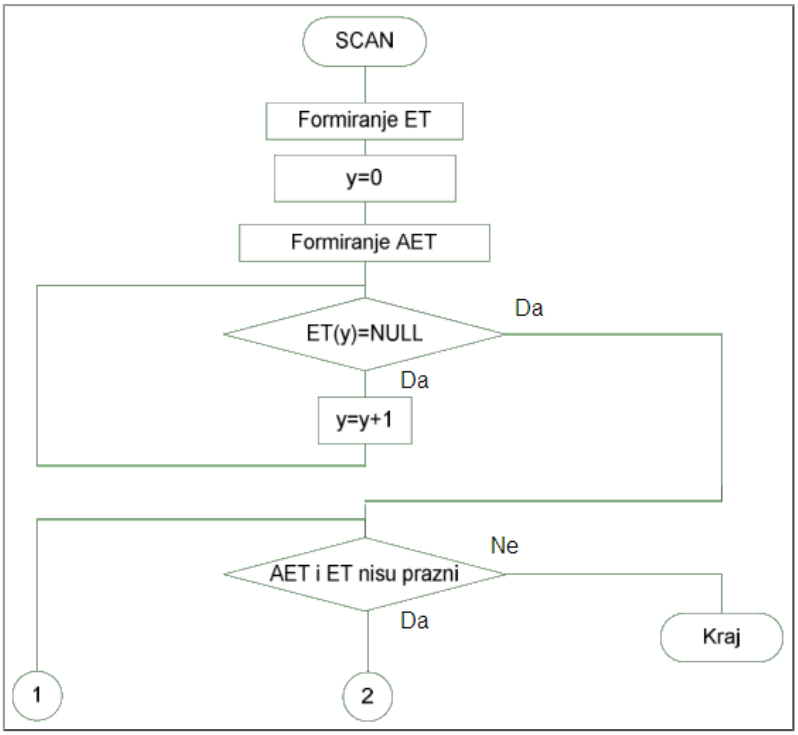


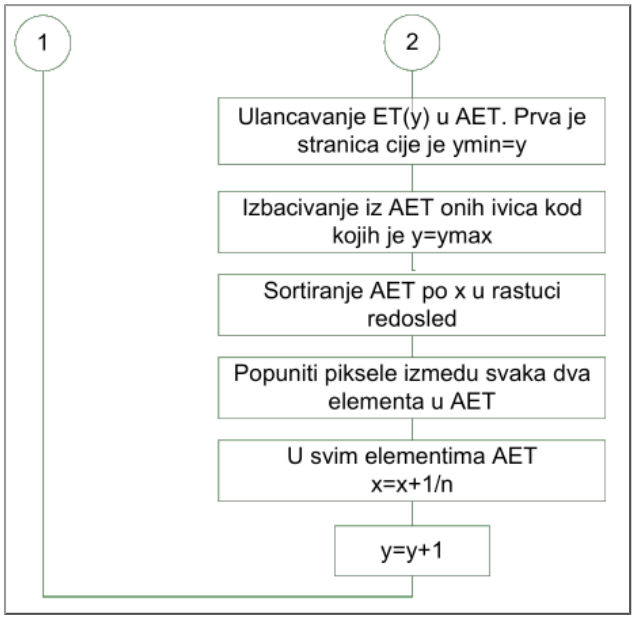
Iterativni algoritam za popunu oblasti koristi se zbog toga što rekruzivni algoritmi brzo popunjavaju stek, što može dovesti do loših performansi ili pak nemogućnosti izvršenja programa.

Algoritam deli oblast za ispunu na linije, a linije na staze, pri čemu je jedna staza niz uzastopnih piksela koje treba obojiti. Za svaku stazu se na steku pamti njen krajnji desni piksel, a po pamćenju svih staza jedne linije, linija se obeležava kao obrađena. Sve dok stek nije prazan, čita se piksel sa steka, vrši se popuna te staze, a zatim se ispituje da li su linije iznad i ispod obeležene - u slučaju da nisu, sve njihove staze se pamte na steku i one se obeležavaju na već opisani način. Ovaj postupak se ponavlja sve dok se ne obrade sve staze (popuni cela oblast).

36. Algoritam za popunu poligona zadatog listom temena. [primer/zadatak]

Algoritam za popunu poligona radi tako što nalazi tačke preseka ivica poligona sa sken linijom, a zatim boji područija između parova tačaka. Koriste se 2 pomoćne strukture: tabela ivica (ET - *Edge Table*), koja za svaku sken liniju pamti duži koje se u toj liniji prvi put presecaju (pamti se najviša tačka duži - y_{max}, x koordinata preseka i inverzni koeficijent pravca - vrednost $1/n$) i tabela aktivnih ivica (AET - *Active Edge Table*), koja se menja u toku izvršenja algoritma uz korišćenje informacija iz ET, koja je zapravo lista ivica koje su presečene u određenoj sken liniji, pri čemu se x koordinata u čvorovima liste ažurira pri svakoj iteraciji za odgovarajuću vrednost inverznog koeficijenta pravca. Dvostruki presek se pamti kao jedna tačka, osim ako se nalazi na lokalnom minimumu ili maksimumu.





Potencijalno podpitanje:

-Modifikacija algoritma za krug i elipsu.

Ne postoji potreba za AET, a u čvorovima ET se pamte samo x_{min} i x_{max}.

37. Cohen-Sutherland algoritam za odsecanje linija. [primer/zadatak]

```
typedef unsigned int outcode;
enum { TOP = 0x8, BOTTOM = 0x4, RIGHT = 0x2, LEFT = 0x1 };

void cohenSutherlandAlgoritam(double x0, double y0,
    double x1, double y1, double xmin, double xmax,
    double ymin, double ymax, COLORREF color)
{
    outcode oc0, oc1, ocOut;
    bool accept = false, not_done = true;
    oc0 = racunajOutCode(x0, y0, xmin, ymin, xmax, ymax);
    oc1 = racunajOutCode(x1, y1, xmin, ymin, xmax, ymax);

    while (not_done) {
        if (!(oc0 | oc1)) {
            accept = true;
            not_done = false;
        }
        else if (oc0 & oc1)
            not_done = false;
        else {
            double x, y;
            ocOut = oc0 ? oc0 : oc1;
            if (ocOut & TOP) {
                x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
                y = ymax;
            } else if (ocOut & BOTTOM) {
                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                y = ymin;
            } else if (ocOut & RIGHT) {
                y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
                x = xmax;
            } else {
                y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
                x = xmin;
            }
            if (ocOut == oc0) {
                x0 = x;
                y0 = y;
                oc0 = racunajOutCode(x0, y0, xmin, xmax, ymin, ymax);
            } else {
                x1 = x;
                y1 = y;
                oc1 = racunajOutCode(x1, y1, xmin, xmax, ymin, ymax);
            }
        }
    }

    if (accept)
        bresenhamLine(x0, y0, x1, y1, color);
}
```

Osnovna ideja *Cohen-Sutherland* algoritma je da se najpre pokuša prihvatanje ili odbacivanje linije u celini. Ako to ne urodi plodom, određuje se presek linije i produžene ivice prozora pa se ponovo pokušava prihvatanje ili odbacivanje ostatka linije. Posmatraju se krajnje tačke linija, pri čemu se razlikuju tri slučaja:

1) Ispunjen je uslov trivijalnog prihvatanja: Obe krajnje tačke su unutar regiona za odsecanje.

2) Ispunjen je uslov trivijalnog odbacivanja: Obe krajnje tačke su levo od x_l ili desno od x_r, ili iznad y_t ili iznad y_b.

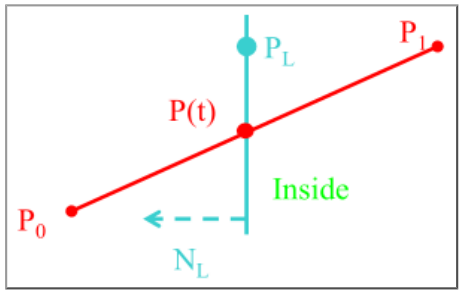
3) Svi ostali slučajevi: Neophodno je izračunavanje preseka.

Prilikom izračunavanja se ivice prozora produžavaju, tako da se cela slika sastoji iz 9 oblasti. Svakoj oblasti se pridružuje 4-bitni položajni kod (*outcode*) $b_3 b_2 b_1 b_0$, pri čemu svaki bit označava jednu oblast (b_3 - iznad, b_2 - ispod, b_1 - desno i b_0 - levo). Zatim se određuju položajni kodovi krajnjih tačaka linije. Uslov trivijalnog prihvatanja ispunjen je ako su oba koda 0. Uslov trivijalnog odbacivanja je da oba koda

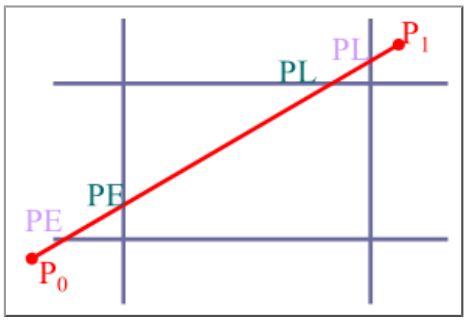
```
outcode racunajOutCode(double x, double y,
    double xmin, double xmax, double ymin, double ymax)
{
    outcode code = 0;
    if (y > ymax)
        code |= TOP;
    else if (y < ymin)
        code |= BOTTOM;

    if (x > xmax)
        code |= RIGHT;
    else if (x < xmin)
        code |= LEFT;

    return code;
}
```

imaju bar jedan zajednički bit. U svim drugim slučajevima se za krajnju tačku linije koja nije u prozoru ispituje da li je levo, desno, iznad ili ispod prozora, te se nalazi presek linije koja se odseca sa odgovarajućom produženom ivicom prozora. Tada se krajnja tačka linije premešta u tačku preseka, i izračunava joj se položajni kod, nakon čega se ponavlja ispitivanje uslova.

38. Cyrus-Beck algoritam za odsecanje linija. [primer/zadatak]



Cyrus-Beck algoritam optimizuje proces nalaženja preseka. Polazi se od parametarske jednačine linije $P(t) = P_0 + (P_1 - P_0) \cdot t$, biraju se tačke koje pripadaju ivicama regiona odsecanja P_L i računa normala N_L na svaku ivicu, a zatim se nalazi t takvo da je $N_L \cdot [P(t) - P_L] = 0$ tako što se $P(t)$ zamenjuje u ovoj jednačini i nalazi njeno rešenje kao $t = N_L \cdot [P_L - P_0] / -N_L \cdot [P_1 - P_0]$. Ovako se t računa za preseke linije sa svim ivicama regiona odsecanja, pri čemu se odbacuju svi preseci kod kojih je $t < 0$ ili $t > 1$, a preostali preseci se kvalifikuju kao potencijalno ulazni (*Potentially Entering* - PE, za $N_L \cdot [P_1 - P_0] < 0$) ili potencijalno izlazni (*Potentially Leaving* - PL, za $N_L \cdot [P_1 - P_0] > 0$). Na kraju, linija se crta između PE sa najvećim t i PL sa najmanjim t.



39. Sutherland-Hodgeman algoritam za odsecanje poligona. [primer/zadatak]

```
typedef struct Vertex { float x, y; } Vertex;
typedef Vertex edge[2];
typedef Vertex outVertexArray[256];

Vertex polygonClipping(Vertex *inputVertexArray)
{
    int inLength = sizeof(inputVertexArray);
    int *outLength = new int;

    edge[0] = clippingWindow[0][0];
    edge[1] = clippingWindow[height][width];
    sutherlandHodgemanAlgoritam(inputVertexArray, outputVertexArray,
        inLength, outLength, edge);
    outputToInput(inLength, inputVertexArray, outLength,
        outputVertexArray);

    edge[0] = clippingWindow[height][0];
    edge[1] = clippingWindow[0][width];
    sutherlandHodgemanAlgoritam(inputVertexArray. outputVertexArray,
        inLength, outLength, edge);
    outputToInput(inLength, inputVertexArray, outLength,
        outputVertexArray);

    edge[0] = clippingWindow[height][width];
    edge[1] = clippingWindow[0][width];
    sutherlandHodgemanAlgoritam(inputVertexArray. outputVertexArray,
        inLength, outLength, edge);
    outputToInput(inLength, inputVertexArray, outLength,
        outputVertexArray);

    edge[0] = clippingWindow[height][0];
    edge[1] = clippingWindow[0][0];
    sutherlandHodgemanAlgoritam(inputVertexArray. outputVertexArray,
        inLength, outLength, edge);
    outputToInput(inLength, inputVertexArray, outLength,
        outputVertexArray);

    delete outLength;
}
```

Kod *Sutherland-Hodgeman* algoritma se vrši odsecanje poligona u odnosu na neku od ivica koje se posmatraju kao granice poligona. Ovaj postupak se ponavlja za svaku od ivica regiona odsecanja.

Ulazna lista algoritma je uređena lista temena poligona, a izlaz lista temena novog poligona koja se najčešće sastoji od nekih starih i nekih novih temena. Ako je P tačka za koju se trenutno proverava da li treba da ide u izlaznu listu, a S tačka iz prethodne iteracije, tada razlikujemo 4 slučaja:

1) S i P su unutar poligona: Dodajemo p u izlaznu listu, a S je već u njoj.

2) S je unutar, a P van poligona: Dodajemo tačku preseka I u izlaznu listu.

3) S i P su van poligona: Ne dodajemo ništa u izlaznu listu.

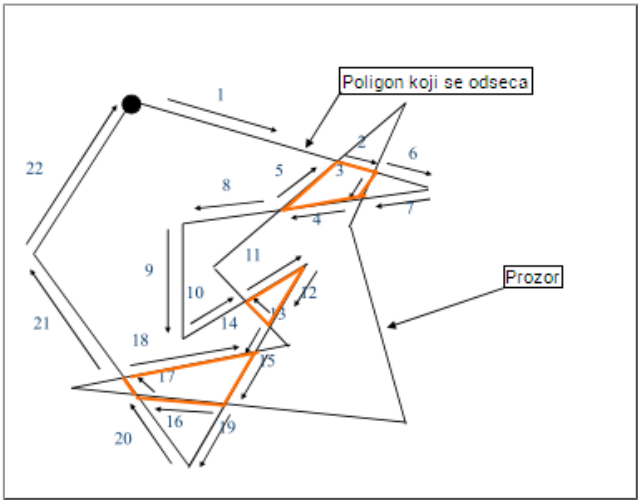
4) S je van, a P unutar poligona: Dodajemo P i tačku preseka I u izlaznu listu.

```
void sutherlandHodgemanAlgoritam(Vertex *inVertexArray,
    Vertex *outVertexArray, int inLength, int *outLength,
    Vertex *clipBoundary)
{
    Vertex i, s = inVertexArray[inLength - 1];
    *outLength = 0;

    for (int j = 0; j < inLength; ++j) {
        Vertex p = inVertexArray[j];
        if (inside(p, clipBoundary)) {
            if (inside(s, clipBoundary))
                output(p, outLength, outVertexArray);
            else {
                intersect(s, p, clipBoundary, i);
                output(i, outLength, outVertexArray);
                output(p, outLength, outVertexArray);
            }
        }
        else {
            if (inside(s, clipBoundary)) {
                intersect(s, p, clipBoundary, i);
                output(i, outLength, outVertexArray);
            }
            s = p;
        }
    }
}
```

```
void outputToInput(int inLength, Vertex *inVertexArray,
    int *outLength, Vertex *outVertexArray)
{
    if (inLength == 2 && *outLength == 3) {
        inVertexArray[0].x = outVertexArray[0].x;
        inVertexArray[0].y = outVertexArray[0].y;
        if (outVertexArray[0].x == outVertexArray[1].x) {
            inVertexArray[1].x = outVertexArray[2].x;
            inVertexArray[1].y = outVertexArray[2].y;
        }
        else {
            inVertexArray[1].x = outVertexArray[1].x;
            inVertexArray[1].y = outVertexArray[1].y;
        }
        inLength = 2;
    }
    else {
        inLength = *outLength;
        for (int i = 0; i < outLength; ++i) {
            inVertexArray[i].x = outVertexArray[i].x;
            inVertexArray[i].y = outVertexArray[i].y;
        }
    }
}
```

```
bool inside(Vertex testVertex, Vertex *clipBoundary)
{
    if (clipBoundary[1].x > clipBoundary[0].x
        && testVertex.y >= clipBoundary[0].y)
        return true;
    if (clipBoundary[1].x < clipBoundary[0].x
        && testVertex.y <= clipBoundary[0].y)
        return true;
    if (clipBoundary[1].y > clipBoundary[0].y
        && testVertex.x <= clipBoundary[1].x)
        return true;
    if (clipBoundary[1].y < clipBoundary[0].y
        && testVertex.x >= clipBoundary[1].x)
        return true;
    return false;
}

void intersect(Vertex first, Vertex second,
    Vertex *clipBoundary, Vertex *intersectPt)
{
    if (clipBoundary[0].y == clipBoundary[1].y) {
        intersectPt->y = clipBoundary[0].y;
        intersectPt->x = first.x + (clipBoundary[0].y - first.y)
            * (second.x - first.x) / (second.y - first.y);
    }
    else {
        intersectPt->x = clipBoundary[0].x;
        intersectPt->y = first.y + (clipBoundary[0].x - first.x)
            * (second.y - first.y) / (second.x - first.x);
    }
}

void output(Vertex newVertex, int *outLength, Vertex outVertexArray)
{
    ++outLength;
    outVertexArray[*outLength - 1].x = newVertex.x;
    outVertexArray[*outLength - 1].y = newVertex.y;
}
```

40. Weiler-Atherton algoritam za odsecanje poligona. [primer/zadatak]



Weiler-Atherton algoritam rešava problem odsecanja proizvoljnog poligona izvan prozora, koji je takođe proizvoljan poligon. Praktično se određuje presek dva proizvoljna poligona. Oba poligona mogu biti konkavna i mogu sadržati rupe. Algoritam kreće od proizvoljnog temena poligona koji se odseca, krećući se u pravcu kazaljke časovnika[7]. Prati se ivica poligona koji se odseca, sve do preseka sa ivicom prozora. Ako ivica „ulazi" u prozor, nastavlja se praćenje ivice poligona koji se odseca. U suprotnom se skreće „udesno" i nastavlja ivicom prozora, kao da je on sada poligon koji se odseca, a originalni poligon za odsecanje prozor. Tačke preseka se pamte da bi se obezbedilo da se svi putevi pređu tačno jednom.

7 Temena u listi su uređena na takav način.

41. Šta su geometrijske transformacije?

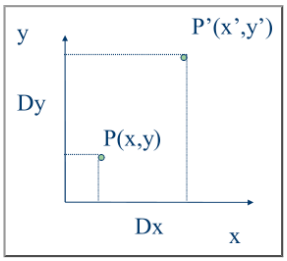
Geometrijske transformacije su matematičke operacije koje preslikavaju originalnu tačku u njenu sliku. Postoje 2D i 3D transformacije.

42. Elementarne geometrijske transformacije. [primer/zadatak]

Elementarne geometrijske transformacije su transformacije čijim se kombinovanjem može konstruisati bilo koja složenija transformacija - bilo koja transformacija se može konstruisati sukcesivnim izvršavanjem određenih elementarnih transformacija.

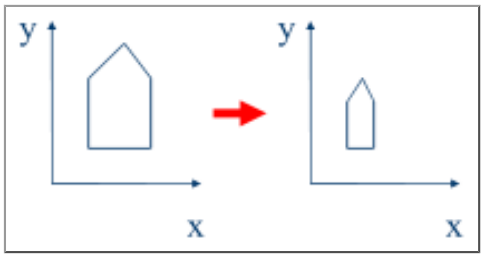
U elementarne geometrijske transformacije spadaju:

1) Translacija: Translacija je pomeranje tačke iz koordinata $[x\ y]$ za D_x po x i D_y po y osi, na koordinate $[x'\ y']$. Očigledno je $x' = x + D_x$ i $y' = y + D_y$. Važi $P' = P + T \Rightarrow [x'\ y']^T = [x\ y]^T + [D_x\ D_y]^T$. Zahvaljujući uniformnim koordinatama, ovo sabiranje vektora se može predstaviti množenjem matricom, što znatno olakšava računicu.



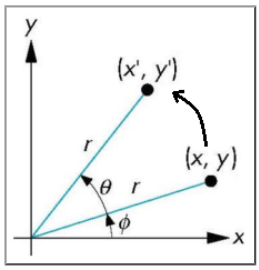
2) Skaliranje: Skaliranje je pomeranje tačaka nekog objekta tako da se objekat širi/skuplja. Tačka $[x\ y]$ se transformiše u tačku $[x'\ y']$, pri čemu je $x' = S_x \cdot x$ i $y' = S_y \cdot y$. Ako je $S_x = S_y$, tada imamo tzv. uniformno skaliranje (objekat se širi/skuplja ravnomerno po obe ose). Takođe, za $S < 0$ imamo umanjenje, a za $S > 0$ uvećanje objekta.

$$P' = P \cdot S \Rightarrow [x'\ y'] = [x\ y]\begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$



3) Rotacija: Tačke nekog objekta se rotiraju za ugao θ oko koordinatnog početka. Tačka $[x\ y]$ se transformiše u tačku $[x'\ y']$, pri čemu važi $x = r\cos\phi$ i $y = r\sin\phi$, odnosno $x' = r\cos(\phi + \theta)$ i $y' = r\sin(\phi + \theta)$.

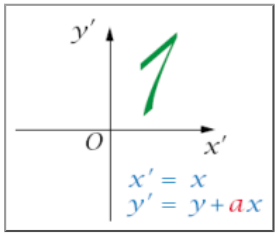
$$\left.\begin{aligned} x' &= r\cos\phi\cos\theta - r\sin\phi\sin\theta = x\cos\theta - y\sin\theta \\ y' &= r\cos\phi\sin\theta - r\sin\phi\cos\theta = x\sin\theta - y\cos\theta \end{aligned}\right\} \Rightarrow P' = P \cdot R \Rightarrow [x'\ y'] = [x\ y]\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

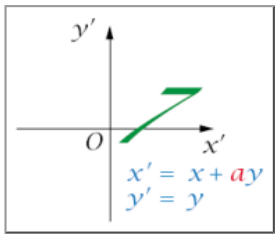


4) Refleksija: Kada je faktor S_x ili S_y kod skaliranja negativan, imamo izvrtanje (refleksiju - *flip*[8]) objekta u odnosu na y ili x osu, respektivno. Ukoliko su oba faktora negativna, objekat se reflektuje u odnosu na obe ose.

5) Smicanje: Smicanje (*Shear*) je rastezanje objekta po jednoj (ili obe) ose. Tačka $[x\ y]$ se transformiše u tačku $[x'\ y']$, pri čemu važi $x' = x + H_x y$ i $y' = y + H_y x$. Ukoliko vršimo smicanje samo u pravcu x (y) ose, vrednost faktora H_y (H_x) je 0.

$$P' = P \cdot H \Rightarrow [x'\ y'] = [x\ y]\begin{bmatrix} 1 & H_y \\ H_x & 1 \end{bmatrix}$$





43. Šta je projekcija?

Projekcija je preslikavanje tačaka iz n-dimenzionalnog prostora u prostor sa manje od n dimenzija. Osnovni parametri projekcije su: centar projekcije (COP), projekcijski zraci i projekcijska ravan.

---

8 U literaturi se koristi termin „*reflection*“, ali se za refleksiju mnogo češće koriste termini „*flip*“ ili „*mirror*“.

44. Podela projekcija.

Planarne geometrjiske projekcije se dele na:

1) Paralelne projekcije: Projekcijski zraci polaze iz tačke (centra projekcije) koja se nalazi u beskonačnosti, što znači da su svi projekcijski zraci paralelni. Paralelne projekcije se karakterišu projekcijskom ravni i pravcem projekcije (DOP), i dele se na:

a) Normalne (ortogonalne) projekcije: projekcijski zraci su normalni na projekcijsku ravan, koje se dalje dele na:

α) Ortografske projekcije: Projekcijska ravan je normalna na neku od koordinatnih osa - može biti pogled odozgo, spreda ili sa strane.

β) Aksonometrijske projekcije: Projekcijska ravan zaklapa proizvoljan ugao sa koordinatnim osama - može biti izometrijska (ugao je isti za sve ose), dijametrijska (ugao je isti za dve ose) ili trijametrijska (ugao je različit za sve ose) projekcija.

b) Kose (klingonalne) projekcije:

α) Kavaljerska projekcija: Projekcijski zraci zaklapaju ugao od 45° sa projekcijskom ravni.

β) Kabinetska projekcija: Projekcijski zraci zaklapaju ugao od $arctg(2)$ sa projekcijskom ravni.

γ) Ostale kose projekcije: Projekcijski zraci zaklapaju proizvoljni ugao sa projekcijskom ravni.

2) Perspektivne (centralne) projekcije: Centar projekcije nije u beskonačnosti. Perspektivne projekcije se karakterišu projekcionom ravni, pravcem projekcije (DOP) i centrom projekcije (COP) i dele se na:

a) Perspektivne projekcije sa jednom tačkom.

b) Perspektivne projekcije sa dve tačke.

c) Perspektivne projekcije sa tri tačke.

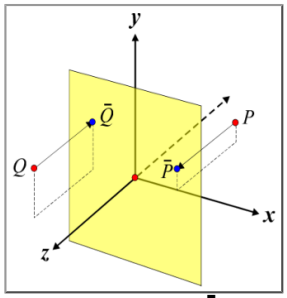
45. Koje se projekcije najčešće koriste u grafici?

Najčešće se koriste projekcije koje preslikavaju 3D objekte u 2D slike objekata, uglavnom su to planarne projekcije (projekcije na ravan).

46. Izvođenje matrica projekcije.

Kao i sve druge geometrijske transformacije, i projekcija se može svesti na množenje matrica.

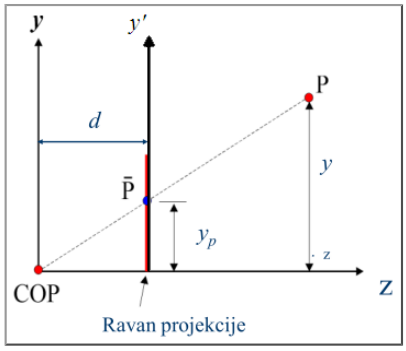
1) Ortografska projekcija: 2) Perspektivna projekcija (y' je y osa u i M'_{per}):



$$\frac{y}{z}=\frac{y_p}{d}\Rightarrow y_p=\frac{y\cdot d}{z}\quad \frac{y}{z+d}=\frac{y_p}{d}\Rightarrow y_p=\frac{y\cdot d}{z+d}$$

$$\frac{x}{z}=\frac{x_p}{d}\Rightarrow x_p=\frac{x\cdot d}{z}\quad \frac{x}{z+d}=\frac{x_p}{d}\Rightarrow x_p=\frac{x\cdot d}{z/d}$$

$$z_p=d \qquad z_p=0$$



$$\left.\begin{aligned}x_p&=x\\y_p&=y\\z_p&=z\end{aligned}\right\}\Rightarrow M_{ort}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}$$

$$P'=P\cdot M_{ort}$$

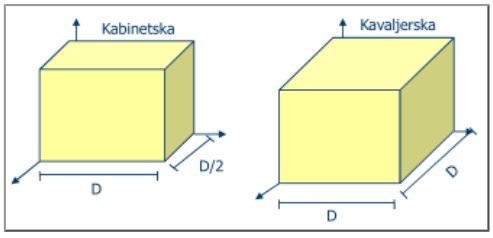
$$M'_{per}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&1/d\\0&0&0&0\end{bmatrix}$$

$$P'=P\cdot M'_{per}$$

$$M_{per}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1/d\\0&0&0&1\end{bmatrix}$$

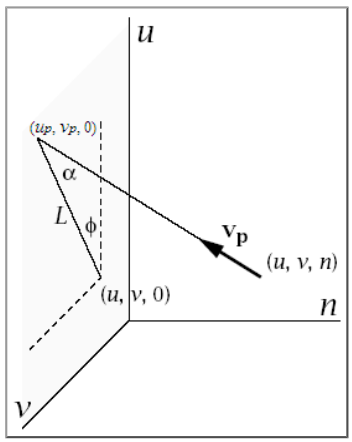
$$P'=P\cdot M_{per}$$

3) Kosa projekcija (za $\alpha=45°$ i $L_1=1$ je kavaljerska, a za $\alpha=arctg(2)$ i $L_1=1/2$ kabinetska):



$$\left.\begin{aligned}u_p&=u+L\cos\phi\\v_p&=v+L\sin\phi\\L&=n/\tan\alpha\\L_1&=1/\tan\alpha\end{aligned}\right\}\Rightarrow M_{kos}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\L_1\cos\phi&L_1\sin\phi&0&0\\0&0&0&1\end{bmatrix}$$

$$P'=P\cdot M_{kos}$$



47. Šta je formiranje 2D pogleda?

Formiranje 2D pogleda predstavlja proces računarskog predstavljanja objekata iz 2D sveta na nekom od izlaznih grafičkih uređaja (štampač, ekran, ploter...).

Potrebno je lokalne koordinate (koordinate objekta) transformisati u svetske (prirodne) koordinate (svetski koordinatni sistem opisuje relativnu poziciju i orijentaciju svih objekata na sceni). Za transformaciju lokalnog u svetski koordinatni sistem koriste se elementarne 2D transformacije. Prozor u svetskom koordinatnom sistemu određuje pravougaonu oblast koja će se prikazati na izlaznom uređaju. Koordinate pogleda opisuju poziciju i orijentaciju svakog objekta u odnosu na zadati prozor, pa se vrši isecanje (*clipping*) objekata koji su van prozora. Prozor se zatim preslikava u zaslon (*viewport*) - pravougaonu oblast u normalizovanim koordinatama (normalizovane koordinate opisuju virtuelni uređaj i postoje zbog kompatibilnosti sa različitim izlaznim uređajima). Koordinate uređaja predstavljaju koordinate konkretnog izlaznog uređaja i zavise od aspekta i rezolucije tog uređaja. Na kraju procesa formiranja pogleda se zaslon preslikava u koordinate uređaja i vrši prikaz.

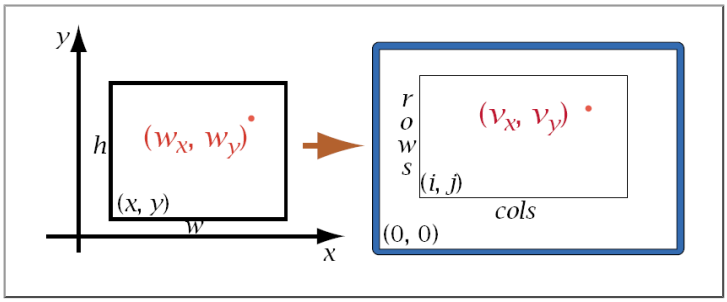
48. Šta je aspekt?

Aspekt (*aspect ratio*) definiše odnos širine i visine slike ($aspect = w/h$).

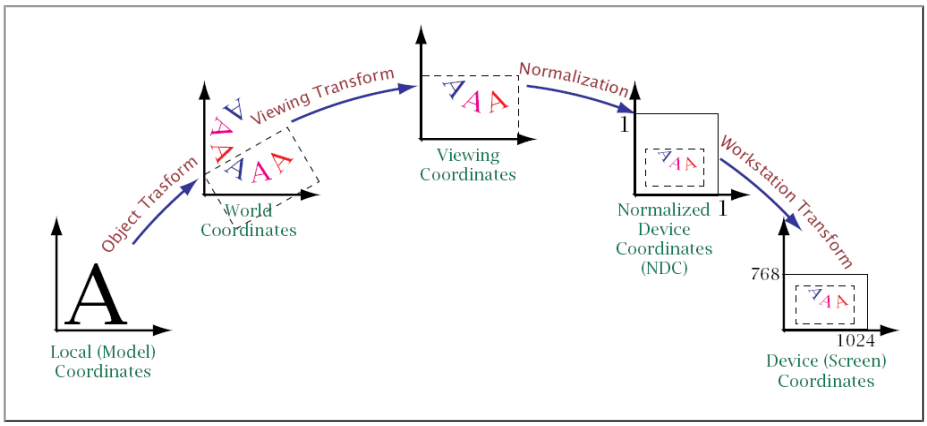
49. Transformacija prozora u zaslon.

Zaslon (*viewport*) predstavlja pravougaonu oblast u normalizovanim koordinatama u koju se preslikava prozor iz svetskog koordinatnog sistema. Obično grafički API obezbeđuju funkciju za transformaciju prozora u zaslon (*widow-to-viewport transformation*).

$$P' = P \cdot T(-x_0, -y_0) \cdot S(w/W, h/H) \cdot T(u_0, v_0)$$



50. Implementacija 2D pogleda.



$$P' = P \cdot T(-x_0, -y_0) \cdot R(-\theta) \cdot S(w/W, h/H) \cdot T(u_0, v_0) \cdot S(c, r)$$

51. Definicija 3D pogleda, uz primer sa fotografijom.

3D pogled predstavlja proces računarskog predstavljanja objekata iz 3D sveta na nekom od izlaznih grafičkih uređaja (štampač, ekran, ploter...). Osnovni elementi kod klasičnog i kompjuterskog pristupa posmatranju objekata su isti - postoje objekti, posmatrač, projektori i ravan projekcije. Ovo se najbolje vidi iz analogije između računarskog posmatranja objekata i fotografije - pozicioniranju fotoaparata odgovara pozicioniranju pogleda u 3D prostoru, pozicioniranje modela koji će biti fotografisan pozicioniranju modela u 3D prostoru, sočivu fotoaparata odgovara projekcija (određivanje oblika pogleda), a fotografiji konačni prikaz na zaslonu.

52. FoV i njegov uticaj na prikaz.

FoV (*Field of View* - ugao pogleda) označava ugao kojim se izražavaju dimenzije pogleda - pomoću vrednosti aspekta i FoV-a, kao i rastojanja posmatrača od ravni pogleda, određuje se volumen pogleda. Veći FoV rezultuje umanjenim, a manji uvećanim prikazom, usled distorzije koja se dešava pri perspektivnoj projekciji. Da ne bi došlo do distorzije, i samim tim manjka realnosti prikaza, preporučuje se vrednost FoV u opsegu 30-60°.

53. Proširenje Cohen-Sutherland algoritma za odsecanje u kanoničkom volumenu.

Svakoj oblasti se pridružuje 6-bitni položajni kod (*outcode*): $b_5 b_4 b_3 b_2 b_1 b_0$, gde svaki bit označava jednu oblast.

Za paralelnu projekciju imamo: b_5 - iznad volumena, $y > 1$, b_4 - ispod volumena, $y < -1$, b_3 - desno od volumena, $x > 1$, b_2 - levo od volumena, $x < -1$, b_1 - ispred volumena, $z > 1$, b_0 - iza volumena, $z < -1$. Za perspektivnu projekciju imamo: b_5 - iznad volumena, $y > z$, b_4 - ispod volumena, $y < -z$, b_3 - desno od volumena, $x > z$, b_2 - levo od volumena, $x < -z$, b_1 - ispred volumena, $z > 1$, b_0 - iza volumena, $z < -z_{max}$.

Uslov trivijalnog prihvatanja ispunjen je ako su oba koda 0. Uslov trivijalnog odbacivanja je da oba koda imaju bar jedan zajednički bit. U svim drugim slučajevima se traži presek linije i volumena pogleda.

54. Koji su metodi za ostvarivanje realnosti prikaza?

Metodi za ostvarivanje realnosti prikaza su uklanjanje sakrivenih ivica i površi, kao i korišćenje boje, svetla i senke.

55. Podela algoritama za uklanjanje sakrivenih ivica i površi.

Algoritmi za uklanjanje sakrivenih ivica i površi dele se na sledeće 2 grupe:

1) Algoritmi prostora slike (*image-precision*): Za svaki piksel slike dimenzija $m \times n$ određuju koji je od k objekata vidljiv - složenost je $O(m \cdot n \cdot k)$ i u njih spadaju:
   a) *Z-buffer* algoritam.
   b) *Warnock*-ov algoritam.
   c) *Scan-line* (*Watkinson*-ov) algoritam.
   d) *Ray-tracing* algoritam.

2) Algoritmi prostora objekata (*object-precision*): Za svaki od k objekata u sceni određuje koji je vidljiv - složenost je $O(k^2)$ i u njih spadaju:
   a) Slikarev algoritam.
   b) *Back face culling* algoritam.
   c) Algoritam koji koristi BSP stabla.
   d) Algoritam koji koristi *octree* stabla.

56. Upoređivanje dubine.

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

1) Utvrditi da li tačke P_1 i P_2 leže na istom projekcionom zraku.

2) Ukoliko tačke P_1 i P_2 ne leže na istom projekcionom zraku, onda se one ne zaklanjaju. U suprotnom je potrebno utvrditi koja se nalazi bliže posmatraču.

Upoređivanje dubine se obično vrši posle normalizacije, kada su 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 $x_1 = x_2$ i $y_1 = y_2$, a kod perspektivne projekcije ako je $x_1/z_1 = x_2/z_2$ i $y_1/z_1 = y_2/z_2$.

57. Z-buffer algoritam za uklanjanje sakrivenih ivica i površi.

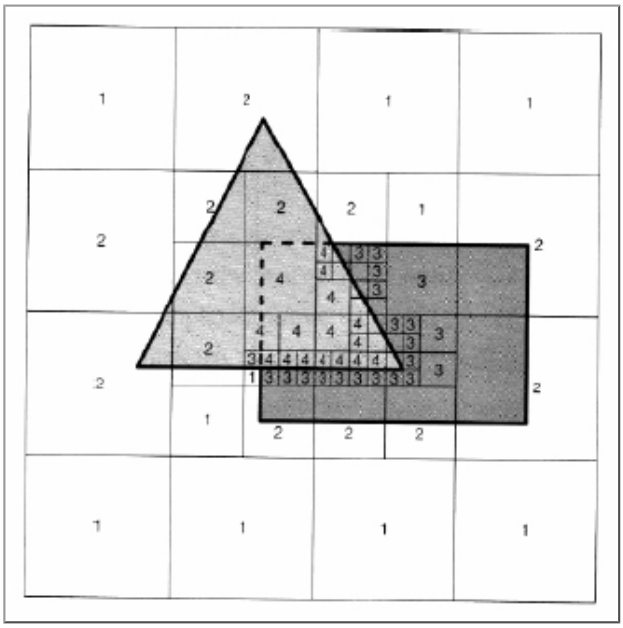
Z-buffer algoritam pored bafera uređaja za prikaz slike (*frame buffer*) koristi još jedan bafer gde se na poziciji (x, y) pamti najmanja z koordinata (dubina) tačaka koje se vide iz piksela (x, y) - taj bafer se zove *Z-buffer*. Njegove prednosti su jednostavna kako softverska, tako i hardverska implementacija, jednostavan princip rada koji ne zahteva sortiranje objekata pre primene algoritma, to što se poligoni na slici pojavljuju u redosledu obrađivanja, to što ne zahteva da primitivne površi budu poligoni, mogućnost optimizacije[9] i mogućnost čuvanja *Z-buffer* podataka zajedno sa generisanom slikom, uz naknadno dodavanje novih objekata. Osnovni nedostatak je to što zahteva mnogo dodatnog memorijskog prostora. Ovaj algoritam se koristi u OpenGL-u.

```
void zBufferAlgoritam()
{
    for (int y = 0; y <= YMAX; ++y)
        for (int x = 0; x <= XMAX; ++x)
        {
            WritePixel(x, y, bk_color);
            WriteZ(x, y, zmax);
        }
    for (int i = 0; i < pgons.size(); ++i)
        for (int j = 0; j < pgons[i].pPix.size(); ++j)
        {
            int x = pgons[i].pPix[j].x;
            int y = pgons[i].pPix[j].y;
            int pZ = pgons[i].zValue(x, y);
            if (pZ <= ReadZ(x, y))
            {
                WriteZ(x, y, pZ);
                WritePixel(x, y, pgons[i].colValue(x, y));
            }
        }
}
```

58. Warnock-ov algoritam za uklanjanje sakrivenih ivica i površi. [primer/zadatak]

Warnock-ov algoritam se može predstaviti u tri koraka:

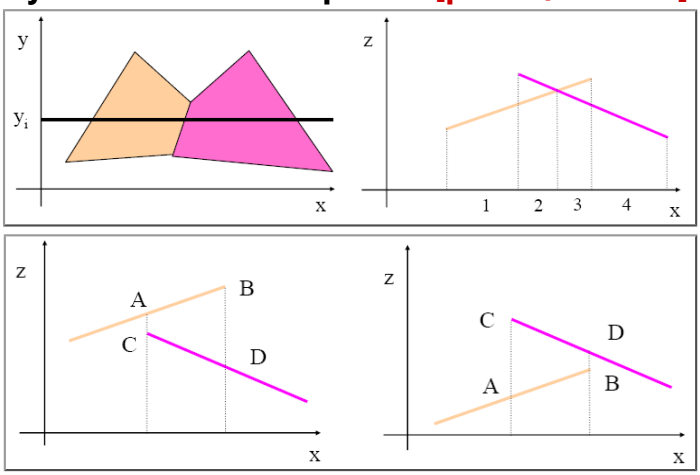
1) Analizira se sadržaj prozora koji se ispituje (u početku je taj prozor jednak zaslonu), pri čemu se mogu javiti sledeći slučajevi:
   a) Prozor je prazan.
   b) Scena u prozoru je prosta i moguće ju je prikazati.
   c) Scena u prozoru je složena, pa je potrebno rekurzivno podeliti prozor i primeniti algoritam.
2) Deli se prozor po *quadtree* principu.
3) Dele se poligoni u tri grupe, u zavisnosti od relacije sa prozorom:
   a) Poligon je izvan prozora, pa se uklanja iz liste.
   b) Poligon seče prozor ili je u njemu, pa se čuva delom ili u celosti.
   c) Poligon prekriva prozor, pa se ceo prozor boji bojom poligona.



59. Scan-line (Watkinson-ov) algoritam za uklanjanje sakrivenih ivica i površi. [primer/zadatak]

Scan-line algoritam[10] se može predstaviti u 4 koraka:

1) Postavlja se ispitna linija u projekcionoj ravni.
2) Određuju se tzv. „rasponi uzorka" - delovi linije na kojima se ne može dogoditi promena vidljivosti. Oni zadovoljavaju sledeće uslove:
   a) Broj segmenata u rasponu uzorka je konstantan i veći od nule.
   b) Projekcije preseka za $y = y_i$ unutar raspona uzorka u ravni xz se ne seku unutar raspona uzorka - svaki presek u projekciji označava novi raspon uzorka.



9 Moguće je izbegavanje množenja i ostale optimizacije na bazi *scan-conversion* algoritma.
10 *3D Studio Max* koristi ovaj algoritam prilikom renderovanja.

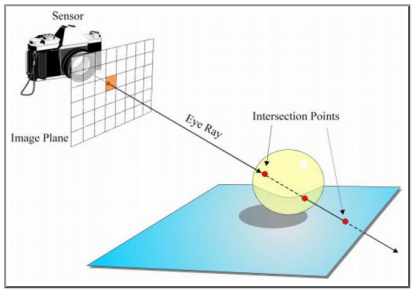
3) Određuje se vidljivost - raspon uzorka se testira u odnosu na vidljivost u dva slučaja:

a) Ako je broj segmenata na tekućoj ispitnoj liniji različit od broja segmenata na prethodnoj ispitnoj liniji.

b) Ako se krajnje tačke dva segmenta zamene, po veličini z koordinate (npr. tačke A i C, B i D) kad prelazimo iz tekuće u susednu ispitnu liniju - tada se obe površine seku u prostoru između dve ispitne ravni.

4) Ispitivanjem z koordinate možemo odrediti koji segment u rasponu uzorka prekriva druge segmente.

60. Ray-tracing algoritam za uklanjanje sakrivenih ivica i površi.



Ideja algoritma je sledeća: Za svaki piksel na slici se emituje imaginarni zrak od oka posmatrača ka sceni koji prolazi kroz dati piksel. Nakon toga se detektuje prvi objekat koji je pogođen emitovanim zrakom i piksel se boji njegovom bojom.

```
void rayTraceAlgoritam()
{
    Ray ray;
    Point intersection;
    for (int i = 0; i < viewport.height; ++i)
        for (int j = 0; j < viewport.width; ++j)
        {
            findEyeRay(viewport, i, j. &ray);
            traceRay(ray, scene.objects, &intersection);
            shade(ray, intersection);
        }
}
```

Algoritam je originalno razvijen za simuliranje putanja balističkih projektila, ali sada ima najznačajniju primenu u grafici. Koristi se za određivanje vidljivosti, za Bulove operacije nad telima, određivanje refleksije i refrakcije itd.

61. Slikarev algoritam za uklanjanje sakrivenih ivica i površi.

Slikarev algoritam radi analogno slikarstvu - prvo se iscrtava najudaljeniji poligon koji ne sakriva druge, a zatim se preko njega iscrtavaju poligoni koji ga (delimično) zaklanjaju. Može se opisati u 4 koraka:

1) Sortiraju se poligoni po rastućoj z koordinati najudaljenijeg temena poligona (z_{max}).

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.

Problem ovog algoritma je što u slučaju uzajamnog preklapanja dolazi do beskonačne petlje. Detektovanje uzajamnog preklapanja i rešenje problema sastoji se iz 4 koraka:

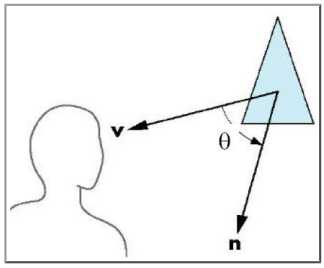
1) Pri prebacivanju Q ispred P u listi, Q se obeleži.

2) Pri svakom prebacivanju vrši se provera da li je poligon Q već bio obeležen.

3) Ako se otkrije da je Q već bio obeležen detektovano je uzajamno preklapanje.

4) U slučaju detektovanog preklapanja poligon Q se deli na dva dela Q_1 i Q_2 pomoću ravni poligona P.

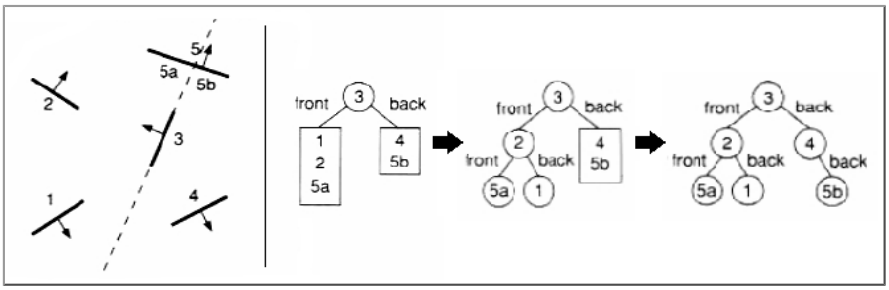
62. Back face culling algoritam za uklanjanje sakrivenih ivica i površi.



Ideja ovog algoritma je da se kod neprozirnih zatvorenih objekata eliminišu zadnje stranice koje ne mogu da se vide direktno. Stranica je vidljiva ako i samo ako je $90° \geq \theta \geq -90°$, što je ekvivalentno sa $\cos\theta \geq 0$, ili $v \cdot n \geq 0$.

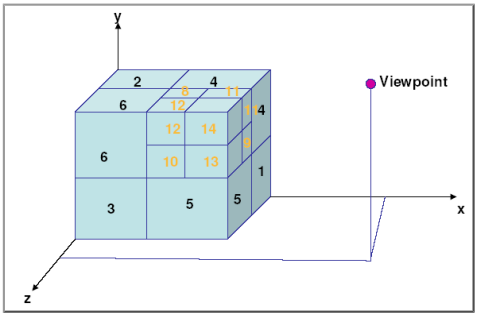
Ovim postupkom se obično uklanja oko 50% stranica, i ovaj algoritam se koristi u OpenGL-u.

63. Algoritam koji koristi BSP stabla za uklanjanje sakrivenih ivica i površi. [primer/zadatak]



BSP (*Binary Space Partitioning*) stabla se formiraju na osnovu podele prostora poligonima. Naime, prostor poligona se deli ravnima kojima poligoni pripadaju, bira se jedan poligon koji će biti koren stabla, a zatim se formira stablo u zavisnosti od toga da li su poligoni ispred ili iza ravni kojoj pripada poligon koji je odabran za koren stabla. Kada se formira kompletno stablo, vrši se njegov inorder obilazak u suprotnom smeru, kako bi se prvo iscrtali svi poligoni iza korena, koren, a zatim i svi poligoni ispred njega.

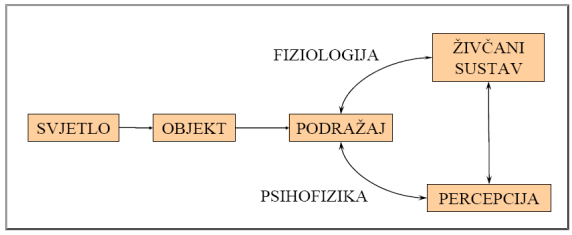
64. Algoritam koji koristi octree stabla za uklanjanje sakrivenih ivica i površi.



Kod ovog algoritma se prostor deli u 8 oktanata i formira se stablo koje sadrži poligone u svakom oktantu. Ako jedan oktant sadrži više od minimalnog broja poligona, on se rekurzivno deli na još 8 oktanata.

Prvo se iscrtavaju poligoni koji se nalaze u oktantu koji je najudaljeniji od posmatrača, a zatim se iscrtavanje rekurzivno ponavlja za oktante koji se graniče sa iscrtanim oktantom.

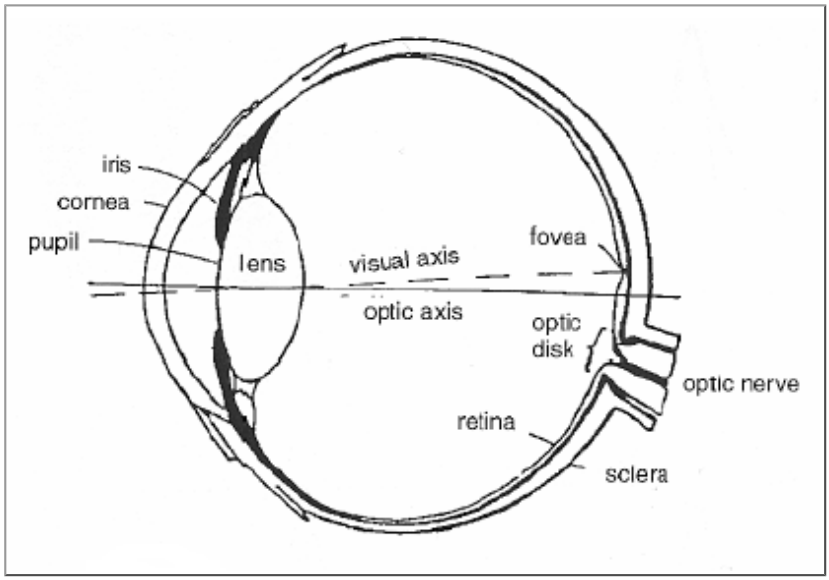
65. Definicija boje.



Boja je osećaj koji se stvara kada se svetlost detektovana pomoću retine (mrežnjače) u oku interpretira u mozgu. Osećaj boje je biološki osećaj.

66. Oko i njegove nesavršenosti u percepciji boja.

Slika se formira na mrežnjači oka, gde se nalaze dve vrste fotoreceptora: štapići, kojih ima oko 10 miliona i čepići, kojih je manje - oko 6.5 miliona. Postoje 3 vrste čepića koje su osetljive na različite talasne dužine svetlosti: dugi, srednji i kratki. Ovo odgovara osnovnim bojama: crvenoj, zelenoj i plavoj. Nedostatak jedne vrste čepića je uzrok daltonizma. Nesavršenost oka ogleda se u tome što pod uticajem određenih boja/oblika na slici naš mozak proizvodi sliku koja ne odgovara realnosti (primeri optičkih varki odnose se na paralelnost linija, boje predmeta, percepciju dubine itd.).



67. Podela i karakteristike svetlosti.

Svetlost se deli na bezbojnu (monohromatsku) i obojenu (hromatsku) svetlost. Najvažnija karakteristika svetlosti je njena talasna dužina. Vidljiva svetlost ima talasnu dužinu između 400 i 700 nanometara (tzv. vidljivi spektar).

Za obojenu svetlost su najvažnije sledeće veličine:

1) Nijansa (*hue*): Pravi razliku između boja (crvene, zelene, ljubičaste itd.).

2) Zasićenost (*saturation*): Meri koliko je boja daleko od sive istog intenziteta. Crvena je visoko zasićena, a roze relativno nezasićena boja.

3) Svetlina (*lightness*): Uključuje monohromatsku notaciju za osećaj intenziteta reflektujućeg objekta.

4) Sjajnost (*brightness*): Koristi se umesto svetline da bi se označio osećaj intenziteta predmeta koji emituje svetlost (npr. sunce ili sijalica).

68. Merenje boja.

U računarskoj grafici je neophodno specificirati i meriti boje, ako se želi njihovo precizno korišćenje. Za reflektovanu svetlost se merenje može obaviti vizuelnim poređenjem uzorka nepoznate boje sa setom standardnih uzoraka na dnevnoj svetlosti, kako bi se izbegao uticaj ambijentalne svetlosti na naš osećaj za boju. Nijansa-svetlina-zasićenost (HLS) je široko prihvaćen model koji predstavlja boje u 3D prostoru. U štamparstvu i grafičkoj industriji boja se obično specificira poklapanjem sa već odštampanim uzorcima. Umetnici boju specificiraju preko tinti, senki, tonova i jako ili slabo zasićenih pigmenata.

Da bi se boje specificirale objektivno, koristi se grana fizike koja se zove kolorimetrija. Ova disciplina se zasniva na sledećim ključnim pojmovima:

1) Dominantna talasna dužina: Talasna dužina boje koju vidimo kada gledamo u svetlo, koja odgovara percepciji nijanse.

2) Čistoća eksitacije: Odgovara zasićenju boje. Čistoća eksitacije obojene svetlosti je proporcionalna čistoći svetla dominantne talasne dužine i belom svetlu potrebnom za definisanje konkretne boje.

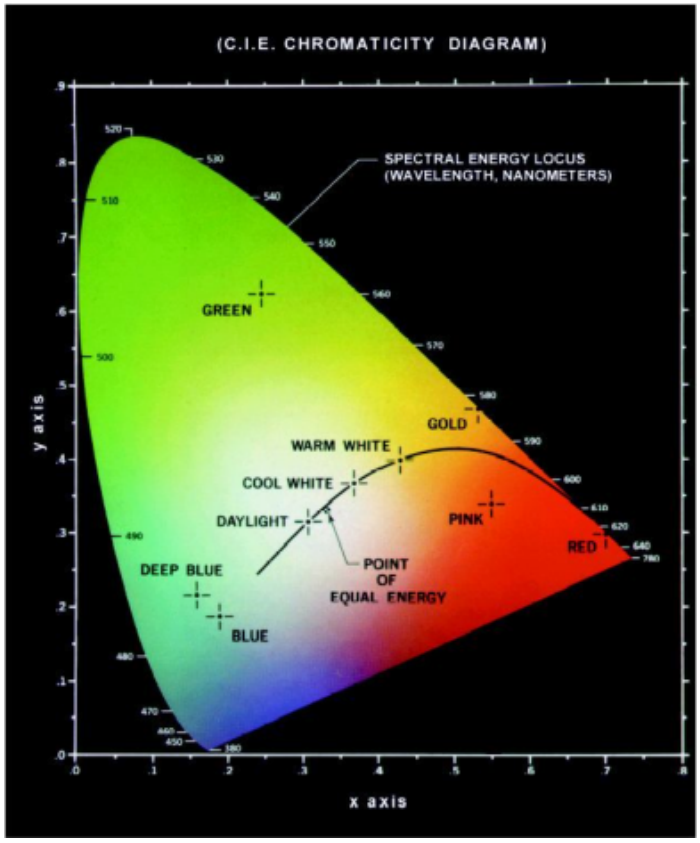
3) Svetlenje: Količina ili intenzitet svetlosti, odgovara svetlini/sjajnosti boje. Potpuno čista boja je 100% zasićena i stoga ne sadrži belu svetlost. Sve ostale boje koje imaju zasićenje između 0 i 100% imaju i belu komponentu svetlosti. Bela boja i sve nijanse sive su 100% nezasićene. Sive boje ne sadrže boju bilo koje dominantne talasne dužine.

69. Modeli boja.

Model boja je specifikacija 3D koordinatnog sistema boja, čiji je cilj da omogući pogodnu specifikaciju boja unutar neke skale boja. Skala koja je od interesa ovde je skala za kolor CRT monitor.

Postoji više modela boja: CIE, EGB, CMY, CMYK, YIQ, HSV (HSB) i HLS model.

70. CIE model boja.



CIE (*Commission Internazionale de l'Eclairage*) je komisija koja je 1931. godine definisala 3 standardne komponente boje: X, Y i Z - hipotetičke boje koje ne postoje u stvarnosti. Mešanjem odgovarajućih količina ovih boja može se dobiti bilo koja druga boja.

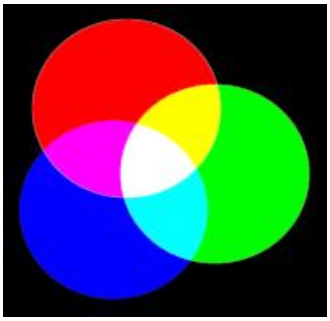
Količine X, Y i Z komponenti potrebne da se mešanjem dobije neka boja čija je distribucija spektralne energije data sa $P(\lambda)$ su $X = k \cdot \int P(\lambda) \cdot \bar{x}_\lambda \, d\lambda$, $Y = k \cdot \int P(\lambda) \cdot \bar{y}_\lambda \, d\lambda$, $Z = k \cdot \int P(\lambda) \cdot \bar{z}_\lambda \, d\lambda$, gde je λ talasna dužina svetlosti. Za objekte koji svetle (npr. sijalica ili monitor), $k = 680 \; lm/W$. Za reflektujuće objekte se uglavnom uzima da bela boja ima Y vrednost jednaku 100, dok Y komponenta ostalih boja ima vrednost između 0 i 100, pa je $k = \frac{100}{\int P_w(\lambda) \cdot y_\lambda \, d\lambda}$.

U ovom obrascu je P_w distribucija spektralne energije za bilo koji izvor svetla izabran kao standard za belo.

Vrednosti hromatičnosti se dobijaju kao $x = \frac{X}{X} + Y + Z$, $y = \frac{Y}{X + Y + Z}$ i $z = \frac{Z}{X + Y + Z}$.

Hromatičnost zavisi samo od dominantne talasne dužine, ne i od količine energije svetla. CIE dijagram hromatičnosti dobija se projekcijom 3D modela na xy ravan. CIE model je manje prirodan od RGB modela, ali je zgodan za transformaciju iz jednog prostora modela boja u drugi.

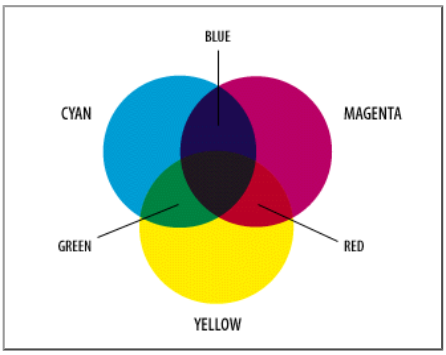
71. RGB model boja.



RGB (*Red-Green-Blue*) model se koristi u rasterskoj grafici i kod CRT monitora u boji. Baziran je na Dekartovom koordinatnom sistemu - $(x, y, z) = (r, g, b)$. Ovaj model pripada grupi aditivnih modela - boje se dobijaju dodavanjem crnoj boji. Boja koju korisnik vidi zavisi od fosfora konkretnog CRT-a. Za prevođenje (korigovanje) boje jednog CRT-a u boju drugog koristi se jednačina sa desne strane.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

72. CMY model boja.



CMY (*Cyan-Magenta-Yellow*) model boja je, za razliku od RGB modela, subtraktivni model boja - njegove boje su komplementi crvene, zelene i plave boje. U subtraktivnim modelima boja, boje se dobijaju tako što se oduzimaju od bele boje. Ovaj model se koristi za uređaje koji štampaju po beloj podlozi.

Potencijalno podpitanje:

-Konverzija iz RGB - CMY i obratno.

Ako *cyan* oduzima crvenu boju od bele svetlosti, a imajući u vidu da je bela svetlost jednaka $R + G + B$ možemo definisati *cyan* kao $R + G + B - R = G + B$. Magenta oduzima zelenu boju, a žuta plavu. Konverzija između RGB i CMY modela (i obratno) se obavlja jednačinama sa desne strane.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

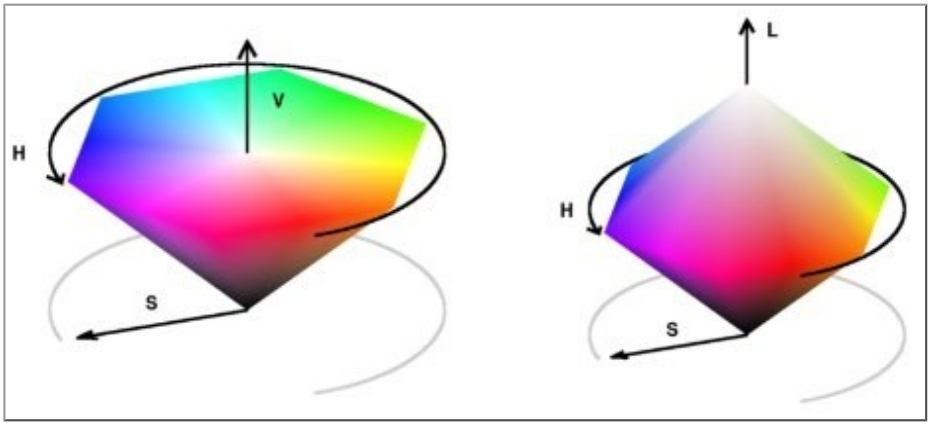
73. CMYK model boja.

CMYK (*Cyan-Magenta-Yellow-Black*) model je izveden iz CMY modela. Iz CMY modela se može preći na CMYK sledećim transformacijama: $K = min(C, M, Y) \Rightarrow C = C - K \wedge M = M - K \wedge Y = Y - K$

74. HSV model boja.

HSV (*Hue-Saturation-Value*) / HSB (*Sue-Saturation-Brightness*) model je korisnički orijentisan, za razilku od RGB, CMY, CMYK i YIQ modela koji su hardverski orijentisani. Zasnovan je na intuitivnom osećaju umetnika za tintu, senku i ton. Koordinatni sistem je cilindričan tj. model je definisan u heksakonusu. Vrh heksakonusa odgovara $V = 1$, i tu se nalaze relativno svetle boje, dok se boje u ravni $V - 1$ ne osećaju sa istom sjajnošću. Nijansa (Hue) se meri uglom u odnosu na vertikalnu osu, pri čemu je crvena na 0, zelena na 120 itd... Komplementarne boje su pomerene za 180°. Vrednost S je između 0 na centralnoj liniji (V osi) i 1 na stranama heksakonusa. Zasićenje se meri kao relativno u odnosu na gamu boje predstavljene ovim modelom, koji je ustvari podskup celog dijagrama CIE hromatičnosti - 100% zasićenje u ovom modelu je manje od 100% eksitacione čistoće.

Heksagon je visine 1 po V osi, sa vrhom u koordinatnom početku. Vrh je crne boje i ima V komponentu jednaku 0. U ovoj tački su vrednosti H i S irelevantne. Tačka sa $S = 0$ i $V = 1$ je bela. Vrednost V između 0 i 1 kada je $S = 0$ daje sivu boju. Kada je $S = 0$, vrednost H je irelevantna. Bilo koja boja sa $V = 1$ i $S = 1$ odgovara situaciji kada umetnik koristi malo pigmenata. Dodavanje belog pigmenta odgovara menjanju S (bez promene V). Senke se kreiraju smanjivanjem V (pri $S = 1$). Tonovi se kreiraju smanjivanjem S i V, a promena H odgovara izboru siromašnog pigmenta sa kojim se startuje.



75. HLS model boja.

HLS (*Hue-Lightness-Saturation*) model boja je definisan u cilindričnom prostoru duple šestougaone piramide. Ustvari, mi HLS predstavljamo kao deformaciju HSV, gde je bela boja podignuta da bi se formirao gornji heksagon od ravni $V = 1$. Svetlina se kreće od 0 za crno (vrh donje piramide) do 1 za belo (vrh gornje piramide).

76. Šta su modeli osvetljenja i senčenja?

Model osvetljenja (*illumination/lighting model*) predstavlja metodu (algoritam) koja se koristi za realan prikaz svetla u sceni. Model senčenja (*shading model*) predstavlja metodu (algoritam) koja se koristi za realan prikaz interakcije objekata sa svetlom (senke i zatamnjenja). Postoji više modela osvetljenja i modela senčenja. Model senčenja je širi okvir i on koristi[11] model osvetljenja. Neka od rešenja za problem osvetljenja zasnovana su na iskustvu i eksperimentima i nisu utemeljena u fizici, ali daju dobre rezultate.

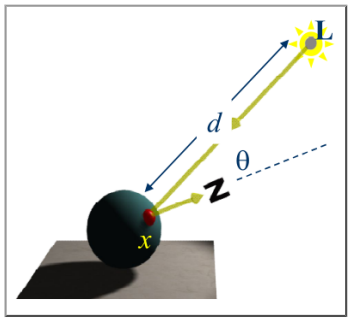
77. Modeli osvetljenja.

Postoje sledeći modeli osvetljenja:

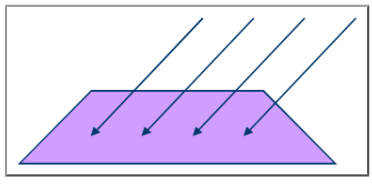
1) Samo-osvetljenost: Najjednostavniji model osvetljenja kod koga je svakoj tački objekta pridružen isti intenzitet svetlosti. Model nije realističan, jer u njemu svako telo izgleda kao da emituje svetlost, tj. da samo sebe osvetljava (otuda i naziv). Jednačina osvetljenja za ovaj model je $I = k_i$, gde je I rezultujući intenzitet svetla, a k_i intenzitet svetla pridružen objektu.

2) Ambijentalno svetlo: Ambijentalno svetlo je difuzno svetlo bez usmerenog izvora, koje je proizvod višestrukog odbijanja svetlosti od svih površina prisutnih u okruženju. Pod pretpostavkom da se svetlost ravnomerno rasprostire u svim smerovima, jednačina osvetljenosti za svaki objekat biće $I = I_a \cdot k_a$, gde je I_a konstantni intenzitet ambijentalnog svetla, a k_a koeficijent ambijentalne refleksije objekta, za koji važi $0 \le k_a \le 1$.

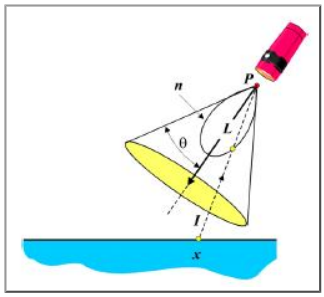
3) Tačkasti izvori svetla: Tačkasti izvor svetlosti ravnomerno širi zrake iz jedne tačke u svim smerovima. Javlja se difuzla ili lambertovska refleksija - odbijanje svetlosti od mat/hrapavih površina. Ovakve površine izgledaju jednako osvetljene iz svih uglova posmatranja i osvetljenost za jednu površinu zavisi samo od ugla θ između pravca svetla $\vec{L}$ i pravca normale na površinu objekta $\vec{N}$. Jednačina osvetljenosti u ovom modelu je $I = I_p \cdot k_p \cdot \cos\theta$, gde je I_p jačina izvora svetlosti, k_p koeficijent difuzne refleksije ($0 \le k_p \le 1$) materijala od kojeg je sačinjen objekat, a θ ugao između pravca svetla i pravca normale. Ako su $\vec{L}$ i $\vec{N}$ normalizovani, tada je $I = I_p \cdot k_p \cdot (\vec{L} \cdot \vec{N})$.



4) Direkciono svetlo: Ukoliko je izvor svetlosti dovoljno udaljen od svih objekata, možemo da smatramo da je vektor $\vec{L}$ konstantan za sve objekte i takav izvor svetlosti zovemo direkcioni izvor svetlosti. Ukoliko se osvetljenost računa po formuli $I = I_p \cdot k_p \cdot (\vec{L} \cdot \vec{N})$ dobija se prikaz u kome svi objekti izgledaju kao da su osvetljeni samo sa jedne strane i da se nalaze u mračnoj prostoriji (zanemaruje se difuzna komponenta svetla). Da bi se ublažio taj efekat, često se koristi jednačina $I = I_a \cdot k_a + I_p \cdot k_p \cdot (\vec{L} \cdot \vec{N})$, koja uključuje ambijentalnu osvetljenost.



5) *Spot* svetlo: *Spot* svetlo predstavlja svetlo koje se emituje od izvora u vrlo uskom ugaonom opsegu (poput reflektora).



6) Prošireni izvori svetla: Prošireni (distribuirani) izvori svetlosti imaju površinu i, kao posledicu toga, daju mekše senke.

11 Model senčenja može pozivati model osvetljenja za svaki piksel slike, ili samo za neke piksele, dok za druge koristi interpolaciju.

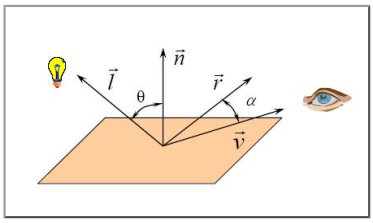
78. Šta je rendering?

Rendering je postupak kojim se određuje odgovarajuća boja piksela koji je pridružen nekom objektu u sceni. Ovaj postupak nije jednostavan i zavisi od geometrije objekta u toj tački (vektora normale a površinu objekta), tipa, pozicije i boje svetlosnog izvora, pozicije posmatrača, materijala od koga je objekat sačinjen (refleksije svetlosti), kao i od uticaja drugih faktora (magla, dim...). Model osvetljenja i model senčenja su osnovni modeli kod renderinga.

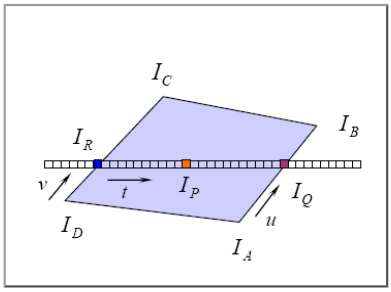
79. Modeli lokalnog i globalnog pristupa osvetljenja i senčenja.

Modeli osvetljenja i senčenja se dele na dve grupe:

1) Modeli lokalnog pristupa: Razmatraju samo osvetljenje od lokalnih izvora svetlosti i ne uzimaju u obzir refleksiju od ostalih objekata u sceni. U njih spadaju:



a) Konstantno senčenje (*flat shading*): Sve tačke površine unutar poligona imaju isti intenzitet. To znači da je za neki poligon $\vec{l} \cdot \vec{n} = const$ i $\vec{r} \cdot \vec{v} = const$. Dobra strana ove metode je velika brzina, ali je prevelika istaknutost ivica veliki nedostatak.



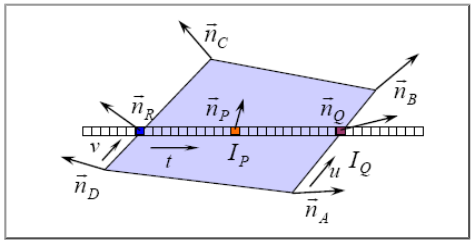
b) *Gouraud*-ovo senčenje: Ova metoda eliminiše vidljivost ivica interpolacijom boja i intenziteta osvetljenosti duž svakog poligona u 3D površi. Boja i intenzitet se računaju za svako teme poligona, a zatim se vrši interpolacija duž ivica poligona. Interpolacija se vrši još jednom duž sken-linija prilikom skeniranja svakog od poligona. Rezultat ovakvog postupka su vrlo glatki prelazi. Postupak je sporiji od prethodnog, ali je i dalje dovoljno brz za praktičnu upotrebu.

$$\textit{Interpolacija duž ivica}: \quad I_Q = (1-u)\cdot I_A + u \cdot I_B,\ u = \frac{\overline{AQ}}{\overline{AB}}$$

$$I_R = (1-v)\cdot I_D + v \cdot I_C,\ v = \frac{\overline{DR}}{\overline{DC}}$$

$$\textit{Interpolacija duž sken-linije}: \quad I_P = (1-t)\cdot I_R + t \cdot I_Q,\ t = \frac{\overline{RP}}{\overline{RQ}}$$

c) *Phong*-ovo senčenje: Ova metoda vrši interpolaciju normala, tako da se izračunavanje boje i intenziteta obavlja za svaki piksel ponaosob, što za rezultat ima veliki broj izračunavanja i odličan kvalitet prikaza, uz malu mogućnost primene za senčenje u realnom vremenu.

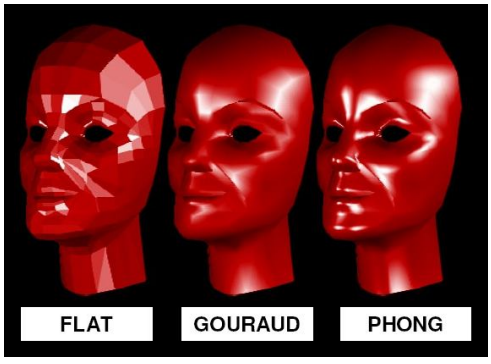


$$\textit{Interpolacija duž ivica}: \quad \vec{n}_Q = (1-u)\cdot \vec{n}_A + u \cdot \vec{n}_B,\ u = \frac{\overline{AQ}}{\overline{AB}}$$

$$\vec{n}_R = (1-v)\cdot \vec{n}_D + v \cdot \vec{n}_C,\ v = \frac{\overline{DR}}{\overline{DC}}$$

$$\textit{Interpolacija duž sken-linije}: \quad \vec{n}_P = (1-t)\cdot \vec{n}_R + t \cdot \vec{n}_Q,\ t = \frac{\overline{RP}}{\overline{RQ}}$$

2) Modeli globalnog pristupa: Razmatraju i refleksiju od ostalih objekata u sceni. U njih spadaju:



a) Metod praćenja zraka (*Ray-tracing*): Uzima u obzir senke, refleksiju i refrakciju (prelamanje zraka kroz transparentne objekte). Ova metoda podrazumeva tačkaste izvore svetlosti i zavisi od položaja posmatrača.

b) Metod isijavanja (*Radiosity method*): Uzima u obzir senke, refleksiju i refrakciju, ali za razliku od prethodne metode svetlosni izvori mogu biti proizvoljnih dimenzija i geometrije. Ova metoda ne zavisi od položaja posmatrača.

c) *Path-tracing* metod: Koristi *Monte-Carlo* metodu za upravljanje geometrijom, refleksijom i osvetljajem. Ovo je najbolja, ali najzahtevnija metoda.

80. Senke i algoritmi za njihovo određivanje.

Senke doprinose realističnosti scene tako što omogućavaju dobijanje dopunskih informacija o položaju objekta u sceni, dubinskoj udaljenosti objekta, obliku objekta, kao i položaju izvora svetlosti. Delovi senke su umbra, koja nastaje pod uticajem glavnog izvora svetlosti i čini najtamniji deo senke objekta i penumbra, koja nastaje pod uticajem više izvora svetlosti ili izvora svetlosti koji nisu tačkasti i čini svetliji deo na krajevima senke objekta.

Od algoritama za određivanje senki postoje:

1) Lažne senke: Kao senke se koriste jednostavni objekti (poligoni pored/ispod objekta).

2) Projektovane senke: Senke su projekcije objekata iz pozicije izvora svetlosti.

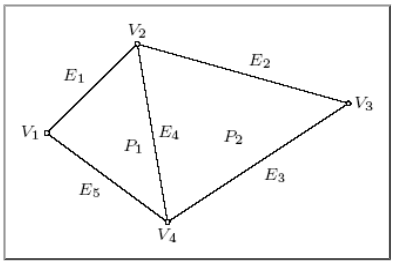
3) Mape senki: Vrše se dva renderinga - prvi se vrši iz ugla izvora svetlosti, pri čemu se pamti dubina svake površine koju svetlo „vidi", a drugi renderuje pogled posmatrača, uz poređenje svake tačke sa zapamćenom dubinom (mapom senki), radi utvrđivanja da li je površina osenčena ili ne.

4) Zapremina senke: Za svaki objekat se određuje poluotvorena zapremina senke koja ima oblik najsličniji obliku zarubljene piramide. Ulazak zraka u zapreminu svake senke povećava dubinu senke, dok je izlazak iz zapremine smanjuje.

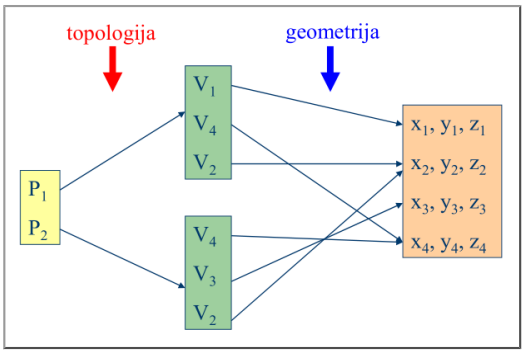
81. Modeli poligonalne mreže.

Modeli predstavljanja poligonalne mreže su:

1) Eksplicitna lista temena: Najjednostavniji model predstavljanja poligonalne mreže, u kome se tačke memorišu prema redosledu obilaska. Model nije struktuiran i nije efikasan jer se temena ponavljaju, pa izmena jednog temena predstavlja veliki posao.



2) Lista poligona: Svako teme se memoriše samo jednom, a svaki poligon se predstavlja listom pokazivača na temena. U memoriji se skup tačaka pamti kao $V = (v_0, v_1, v_2, \ldots)$, a poligon se definiše listom pokazivača $P(Pv_i, Pv_j, Pv_k)$. Orijentacija poligona određena je redosledom temena (koristi se pravilo desne ruke). Ovim modelom se postiže razdvajanje geometrije od topologije - geometrija određuje lokaciju temena, a topologija njihovu organizaciju u ivice i organizaciju ivica u poligone. Ukoliko se u ovom modelu geometrija promeni, topologija ostaje ista.

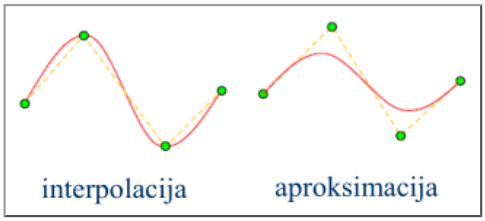


3) Eksplicitna lista ivica: Pored svakog temena, i svaka ivica se memoriše samo jednom (pamti se lista ivica) i sadrži pokazivače na temena koja je sačinjavaju i na poligone kojima pripada. Poligon se zadaje listom pokazivača na listu ivica. Kod ovog načina se svaka ivica iscrtava samo jednom, ali i dalje je prisutan problem teškog nalaženja zajedničkih temena i ivica susednih temenu.

$$V = (V_1, V_2, V_3, V_4) = ((x_1, y_1, z_1), \ldots, (x_4, y_4, z_4))$$
$$E_1 = (V_1, V_2, P_1, null)$$
$$E_2 = (V_2, V_3, P_2, null)$$
$$E_3 = (V_3, V_4, P_2, null)$$
$$E_4 = (V_4, V_2, P_1, P2)$$
$$E_5 = (V_4, V_1, P_1, null)$$
$$P_1 = (E_1, E_4, E_5)$$
$$P_2 = (E_2, E_3, E_4)$$

82. Interpolacija i aproksimacija.

Interpolacija se koristi kada se rekonstruiše originalna kriva koja mora proći kroz zadate tačke, a aproksimacija radi postizanja estetskog efekta tako što se kriva aproksimira nekom pravilnijom krivom (sa blažim promenama). U tom slučaju kriva treba da prođe u blizini zadatih tačaka.



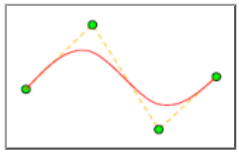
83. Osnovni tipovi krivih.

Krive mogu biti otvorene ili zatvorene, razlomljene ili nerazlomljene, kao i periodiče ili neperiodične.

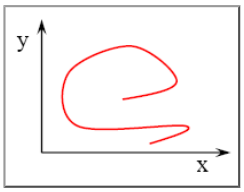
84. Osnovni principi prikaza krivih.

Sledeće karakteristike su zajedničke za sve načine predstavljanja krivih:

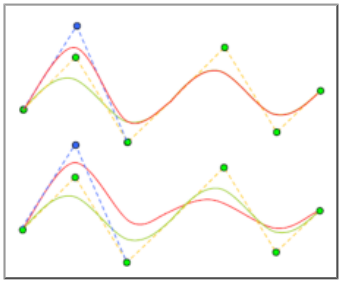
1) Kontrolne tačke: Uobičajeni način za interaktivnu kontrolu delova krive - ili se definišu sve tačke kroz koje mora da prođe kriva, ili samo neke tačke, koje po određenoj zavisnosti utiču na njen oblik.



2) Višeznačne vrednosti: U opštem slučaju oblik krive ne predstavlja grafik jednoznačno određene funkcije jedne promenljive, već može imati višeznačne vrednosti.

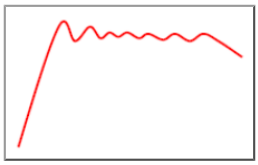


3) Nezavisnost od koordinatnog sistema: Oblik krive ne bi smeo da se promeni kada se kriva prebaci iz jednog u drugi koordinatni sistem.



4) Globalna i lokalna kontrola: Kada se promeni položaj jedne od kontrolnih tačaka, kriva sa lokalnom kontrolom menja oblik samo u blizini te kontrolne tačke, dok kriva sa globalnom kontrolom menja oblik u celini.

5) Varijacije između kontrolnih tačaka: Algoritam za crtanje krive mora obezbediti da izračunata kriva nema više prevojnih tačaka od izvorne krive - treba smanjiti „titranje“ krive.

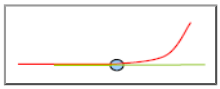


6) Mogućnost dodavanja i brisanja kontrolnih tačaka: Omogućava preciziranje krive, kao i interaktivnost algoritma za crtanje krive.

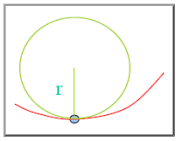
7) Granični uslovi (kontrola reda neprekidnosti): Često se kod crtanja krivih koristi spajanje većeg broja segmenata krivih. Od algoritama za crtanje se može zahtevati da zadovolji granične uslove u tačkama spoja segmenata krive. Postoji nekoliko tipova neprekidnosti:

a) C^0 - Neprekidnost nultog reda: Početna tačka narednog segmenta je krajnja tačka prethodnog segmenta - $f(t) = g(t)$.

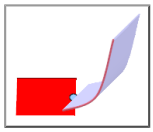
b) C^1 - Neprekidnost prvog reda: Rezultujuća kriva ima jedinstvenu tangentu u tački spajanja - $f'(t) = g'(t)$.



c) C^2 - Neprekidnost drugog reda: Rezultujuća kriva ima jedinstvenu krivinu u tački spajanja (oskulatorna kružnica se poklapa sa krivom) - $f'(t) = g'(t) \wedge f''(t) = g''(t)$.



d) C^3 - Neprekidnost trećeg reda: Dobija se kontinuitet u razvijanju - $f'''(t) = g'''(t)$.



8) Pored C kontinuiteta postoje i tzv. G kontinuiteti koji zahtevaju proporcionalnost:

a) G^1 - $f'(t) = k \cdot g'(t),\ k > 0$.

b) G^2 - $f''(t) = k \cdot g''(t),\ k > 0$.

c) G^3 - $f'''(t) = k \cdot g'''(t),\ k > 0$.

Važno je napomenuti da C^1 direktno implicira G^1, osim kada je vektor tangente $[0\ 0\ 0]$. Kod C^1 je moguća promena smera krive, dok kod G^1 nije.

85. Analitičko predstavljanje krivih. [primer/zadatak]

Kriva se može predstaviti analitički u nekom od sledećih oblika:

1) Implicitni oblik.
2) Eksplicitni oblik.
3) Parametarski oblik.

Primer implicitnog predstavljanja krive je:

$$C: \begin{matrix} F(x, y, z) = 0 \\ G(x, y, z) = 0 \end{matrix}$$

$$Jednačina\ prave: Ax + By + Cz = 0$$

Primer eksplicitnog predstavljanja krive je:

$$C: \begin{matrix} y = f(x) \\ z = g(x) \end{matrix} \quad Jednačina\ prave: \begin{matrix} y = m_1 \cdot x + b_1 \\ z = m_2 \cdot x + b_2 \end{matrix}$$

$$Tačka: P[x, f(x), g(x)]$$

Primer parametarskog predstavljanja krive je:

$$C: \begin{matrix} x = f(t) \\ y = g(t) \\ z = h(t) \\ a \le t < b \end{matrix} \quad Jednačina\ prave: \begin{matrix} x = x_0 + (x_1 - x_0) \cdot t \\ y = y_0 + (y_1 - y_0) \cdot t \\ z = z_0 + (z_1 - z_0) \cdot t \end{matrix} \quad \begin{matrix} Tačka\ u\ vektorskom\ obliku: \\ V(t) = [x(t)\ y(t)\ z(t)] \\ V'(t) = [x'(t)\ y'(t)\ z'(t)] \end{matrix}$$

86. Prednosti parametarskog predstavljanja krivih.

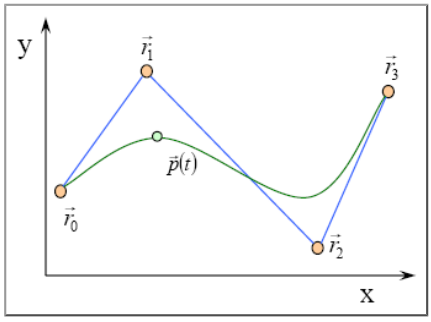
Prednosti parametarskog predstavljanja krivih naspram predstavljanja krivih u implicitnom ili eksplicitnom obliku su:

1) Mogućnost predstavljanja krivih koje nemaju poznatu matematičku definiciju.
2) Sve koordinate se tretiraju na isti način.
3) Mogućnost izračunavanja višeznačnih vrednosti krivih.
4) Razlika pri generisanju krive u 3D u odnosu na 2D se svodi na dodatno izračunavanje $z(t)$.

87. Krive koje se najčešće koriste.

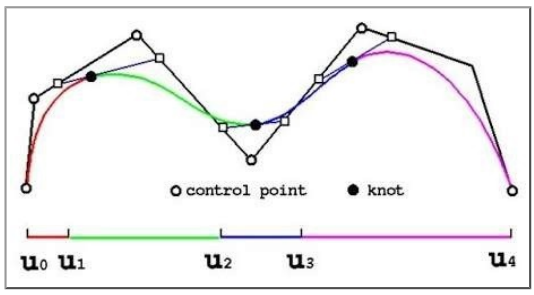
Krive koje se najčešće koriste su:

1) *Bezier*-ove krive: *Bezier*-ova kriva se generiše za skup kontrolnih tačaka, od kojih prolazi samo kroz početnu i krajnju, dok ostale utiču na njen oblik. *Bezier*-ove krive imaju sledeće karakteristike:



   a) Postojanje tzv. konveksne ljuske (convex hull) koju obrazuju kontrolne tačke.
   b) Kriva nema više krivina od kontrolnog poligona. Broj preseka ravni sa krivom je uvek veći ili jednak od broja preseka ravni sa kontrolnim poligonom.
   c) Ne postoji lokalna kontrola.
   d) Broj kontrolnih tačaka je direktno u vezi sa stepenom krive.
   e) Nezavisnost od transformacija (translacije, rotacije, skaliranja...).
   f) Simetričnost - simetričnim zamenjivanjem redosleda kontrolnih tačaka se ne menja oblik krive.

2) *B-spline* krive: B-elastična kriva ima osobine elastične letve - otuda naziv „*spline*". *B-spline* se generiše za skup kontrolnih tačaka i monotoni skup parametara koji se zovu „knotovi". Razlikujemo uniformne i neuniformne *B-spline* krive, u zavisnosti od toga da li su knot intervali ravnomerno raspoređeni. *B-spline* krive imaju sledeće karakteristike:



   a) Kriva leži unutar konveksne ljuske.
   b) i -ti segment se nalazi unutar konveksne ljuske pripadajućih kontrolnih tačaka.
   c) Postoji lokalna kontrola. Pomeranje jedne tačke utiče na najviše dva segmenta.

3) NURBS krive: NURBS (*Non-Uniform Rational B-Spline*) krive se dobijaju iz neuniformnih *B-spline* krivih. Kod njih pored skupa kontrolnih tačaka i knotova imamo i skup težina koje su uvek pozitivni brojevi.

88. Analitičko predstavljanje površi. [primer/zadatak]

Površ se može predstaviti analitički na neki od sledećih načina:

1) Implicitni oblik.
2) Eksplicitni oblik.
3) Parametarski oblik.

Primer implicitnog predstavljanja površi je:

$$P\colon F(x, z, z) = 0 \qquad \begin{matrix} \textit{Jednačina ravni}: \\ A \cdot x + B \cdot y + C \cdot z + D = 0 \end{matrix}$$

Primer eksplicitnog predstavljanja površi je:

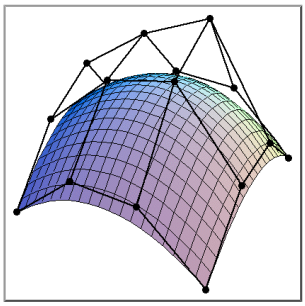
$$z = f(x, y) \quad P[x, y, f(x, y)] \qquad \begin{matrix} \textit{Jednačina ravni}: \\ z = a \cdot x + b \cdot y + c \end{matrix}$$

Primer parametarskog predstavljanja površi je:

$$\begin{matrix} x = f(s, t) \\ y = g(s, t) \\ z = h(s, t) \end{matrix} \qquad \begin{matrix} 0 \le s < 1 \\ 0 \le t < 1 \end{matrix}$$

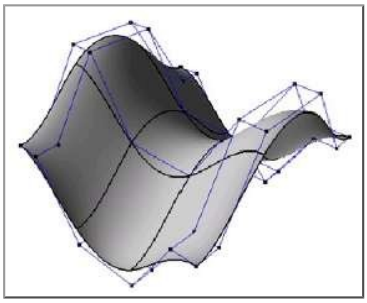
89. Površi koje se najčešće koriste.

Površi koje se najčešće koriste su:



1) *Bezier*-ove površi: Formiraju se od *Bezier*-ovih patch-eva.

2) *B-spline* površi: Formiraju se od *B-spline patch*-eva. Kao i *B-spline* krive, mogu biti uniformne ili neuniformne, u zavisnosti od toga da li je raspodela parametara s i t uniformna ili ne. Prednost *B-spline* površi u odnosu na *Bezier*-ove je postojanje lokalne kontrole - pomeranje jedne kontrolne tačke utiče samo na one patch-eve u njenoj okolini.



3) NURBS površi: Dobijaju se od neuniformnih *B-spline* površi u četvorodimenzionalnom prostoru. Prednost NURBS površi u odnosu na ostale je postojanje težina za svaku kontrolnu tačku, čime se definiše njen uticaj na oblik površi.

Karakteristika svih navedenih površi je mogućnost da se njima predstavljaju vrlo glatke površine u realnom vremenu.