MARKO ČUPIĆ

SKRIPTA IZ RAČUNALNE GRAFIKE

SADRŽAJ

Nekoliko riječi...

Štovani čitatelji! Osobito mi je zadovoljstvo što Vam mogu ponuditi ovaj tekst na raspolaganje. Računalna grafika je područje koje se u današnje doba razvija strahovitom brzinom, i poznavanje osnova računalne grafike polako se može poistovjećivati sa računalnom pismenošću. U ovoj skripti naglasak je bačen na koncepte i ideje kako bi se što lakše mogli uočiti problemi koji se tu pojavljuju, i načini na koje se oni rješavaju, ukoliko isti postoje. Kroz tekst će se pojaviti i nekoliko primjera napisanih u C programskom jeziku koji će demonstrirati izlaganja i ideje koje se obrađuju. Primjeri su napisani tako da rade, i da se vrlo lagano mogu uočiti pojedini dijelovi koda. Primjeri nisu pisani tako da nude apsolutnu otpornost na pogreške, i oporavak na razna, malo vjerojatna stanja. Ova poboljšanja ostavljaju se Vama da ih implementirate ukoliko mislite da su potrebna...

Da biste mogli pratiti skriptu potrebna je određena podloga iz matematike, i to naročito linearne matematike, iako sam se trudio da, gdje je to god bilo moguće, krenem od samih osnova i objasnim zašto se rade pojedini koraci. Zbog ovih zahtjeva, skripta je namjenjena prvenstveno studentima koji su slušali ove predmete, ili kojima matematika nije velika nepoznanica. Naravno, to ne znači da je nitko drugi ne smije čitati.

Ovim putem posebno bih se zahvalio profesorici Željki Mihajlović za pruženu pomoć, kao i prijateljima koji su mi bili potpora.

1.0 UVOD U OSNOVNE MATEMATIČKE POJMOVE

1.1 NAČINI OZNAČAVANJA

Prije nego što krenemo sa gradivom, potrebno je objasniti na koji će se način u tekstu označavati pojedini elementi.

Točka će se obično označavati kao T_x, pri čemu će slovo *X* biti zamijenjeno u *S* ukoliko se govori o početnoj točci pravca, u *E* ukoliko se govori o završnoj točci pravca, odnosno u *P* ukoliko se govori o proizvoljnoj točci pravca. (Termini početna točka pravca i završna točka pravca biti će jasniji nakon poglavlja 1.4).

Zapis točke T_x po komponentama biti će (T_{x1}, T_{x2},..., T_{xn}) pri čemu je T_{xi} i-ta komponenta točke.

Vektori će se označavati slično kao i točke. Vektor pravca biti će označen kao $\vec{v}_p$, odnosno po komponentama $\overrightarrow{(v_{P1}, v_{P2},..., v_{Pn})}$.

Matrice će se označavati velikim štampanim slovima i podvlakom. Npr. karakteristična matrica pravca nositi će oznaku **L**.

Slike u pojedinim poglavljima označavane su brojem poglavlja iza kojeg slijedi redni broj slike u poglavlju. Npr. oznaka slike 5.1.1.1. označava sliku u poglavlju 5.1.1. pod rednim brojem 1.

Relacije (jednadžbe; formule) označavane su na sličan način: uz svaku relaciju u zagradi se nalazi oznaka koja započinje sa "r:" iza koje slijedi oznaka poglavlja, kosa crta i redni broj relacije u tom poglavlju. Npr. oznaka (r:2.1.1/1) označava relaciju broj 1 u poglavlju 2.1.1. Ovakav način označavanja uveden je da bi se maksimalno olakšalo pronalaženje relacije kada se ona referencira u tekstu.

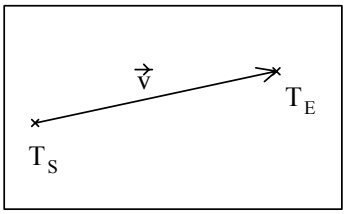
1.2 TOČKA

Točka je matematički pojam koji se u pravilu ne definira. No potrebno je uvesti neke osnovne pojmove. Točka je svojstvo prostora (element prostora; osnovna građevna nedjeljiva cjelina). Zbog toga označavanje točke ovisi o prostoru u kojem se ta točka nalazi. Ono osnovno što određuje označavanje točke je dimenzionalnost prostora. Svaka točka biti će označena svojim koordinatama, i to sa toliko koordinata kolika je dimenzija prostora. Tako će točka u 2D "prostoru" biti označena pomoću dvije koordinate: x i y. U 3D točka će biti označena pomoću tri koordinate: x, y i z. U nastavku ćemo točke označavati kao uređene n-torke koordinata, npr. (x,y,z) ili kao matricu [x y z].

1.3 VEKTOR

Vektor ćemo promatrati kao usmjerenu dužinu, iako riječ "dužina" možda i nije baš najprikladnija. Naime, vektor će nam obično služiti kao gradijent, tj. pokazatelj koji govori za koliko se nešto mijenja. Vektore ćemo označavati pomoću strelice iznad imena što je uobičajena matematička notacija. Zapis vektora, isto kao i točke ovisi o prostoru u kojem taj vektor opisuje, te će imati onoliko komponenti kolika je dimenzionalnost prostora. Zbog toga ćemo vektore također zapisivati kao uređene n-torke, npr. $\overrightarrow{(x,y,z)}$. Kako ćemo u nastavku govoriti o računalnoj grafici, vektore ćemo obično razapinjati između dvije točke, i to na slijedeći način: vektor razapet između točke T_S i T_E kreće iz točke T_S, i završava u točku T_E. Ovime je određen smjer vektora, a njegov iznos se računa pomoću relacije:

$$\vec{v} = T_E - T_S \qquad \text{(r:1.2/1)}$$



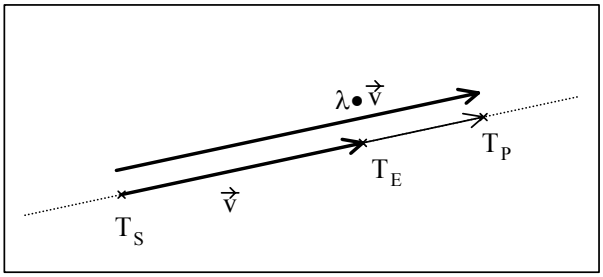
Pri tome se točke oduzimaju na taj način da se odgovarajuće komponente točaka oduzmu.

2.0 PRAVAC

2.1 JEDNADŽBA PRAVCA

2.1.1 PARAMETARSKA JEDNADŽBA

Pravac se također ne definira, no najopćenitije govoreći može se reći da je pravac skup točaka. Ovakva definicija vodi na najopćenitiji zapis pravca: parametarski zapis. Ideja parametarskog zapisa je da se pomoću konačnog broja parametara opišu sve točke koje pripadaju dotičnoj krivulji. Za opis pravca dovoljan je samo jedan parametar, a jednadžbu možemo dobiti iz slike:



$$T_P = (T_E - T_S) \cdot \lambda + T_S = \vec{v} \cdot \lambda + T_S \qquad \text{(r:2.1.1/1)}$$

Formula zapravo kaže: vektor $\vec{v}$ skaliran parametrom λ dodaj točci T_S i dobiti ćeš jednu točku pravca. Ovo napravi za svaki $\lambda \in R$ i dobiti ćeš sve točke pravca.

Pojam "skalirati" parametrom λ znači svaku komponentu vektora $\vec{v}$ pomnožiti realnim brojem λ. Time se norma vektora $\vec{v}$ povećava λ puta, te vrijedi:

$$\|\lambda \cdot \vec{v}\| = \lambda \cdot \|\vec{v}\|$$

Npr. prema gornjoj slici, ukoliko točci T_S dodamo $\lambda\vec{v}$ uz λ=1 doći ćemo do točke T_E. No ukoliko je λ veći od jedan, tada ćemo doći do neke točke koja se krenuvši od točke T_S nalazi iza točke T_E.

U jednadžbi pravca koju smo prethodno napisali krije se zapravo onoliko jednadžbi koliko točke imaju komponenata. Važno je za uočiti da ovakav zapis pravca donekle i usmjerava pravac. Naime, porastom parametra λ točka T_P giba se u smjeru vektora $\vec{v}$ odnosno od točke T_S prema točki T_E i dalje. Ovaj oblik zapisa pravca pogodan je i za određivanje gdje se neka zadana točka pravca nalazi u odnosu na točke T_S i T_E. Vrijedi slijedeće. Ukoliko je za zadanu točku T_P parametar λ:

- negativan, točka T_P je ispred točke T_S.
- nula, točka T_P se upravo poklapa sa točkom T_S.
- pozitivan i manji od 1, točka T_P je između točke T_S i točke T_E.
- jedan, točka T_P se upravo poklapa sa točkom T_E.
- pozitivan i veći od 1, točka T_P je iza točke T_S i iza točke T_E.

Raspišemo li jednadžbu $T_P = (T_E - T_S)\cdot\lambda + T_S = \vec{v}\cdot\lambda + T_S$ po komponentama, dobivamo:

$$(T_{P1}, T_{P2}, ..., T_{Pn}) = \overrightarrow{(v_{P1}, v_{P2}, ..., v_{Pn})}\cdot\lambda + (T_{S1}, T_{S2}, ..., T_{Sn}) \qquad \text{(r:2.1.1/2)}$$

ili u matričnom zapisu:

$$[T_{P1}, T_{P1}, ..., T_{Pn}] = [\lambda \quad 1]\cdot\begin{bmatrix} v_{P1} & v_{P2} & \dots & v_{Pn} \\ T_{S1} & T_{S2} & \dots & T_{Sn} \end{bmatrix} = [\lambda \quad 1]\cdot\mathbf{L} \qquad \text{(r:2.1.1/3)}$$

gdje **L** stoji za karakterističnu matricu pravca. Kako jednadžba mora biti zadovoljena za svaku komponentu zasebno, gornji zapisi jednadžbe raspadaju se na *n* jednadžbi.

2.1.2 RAČUNANJE JEDNADŽBE PRAVCA KROZ DVIJE TOČKE

Neka su zadane dvije točke kroz koje prolazi pravac. Točka T_S neka bude početna točka pravca, točka T_E neka bude završna točka pravca. Jednadžbu pravca kroz te dvije točke možemo izračunati temeljem relacije:

$$T_P = (T_E - T_S)\cdot\lambda + T_S = \vec{v}\cdot\lambda + T_S \qquad \text{(r:2.1.2/1)}$$

tako da izračunamo vektor $\vec{v}$ pomoću relacije (r:1.2/1) koja glasi: $\vec{v} = T_E - T_S$.

No do sličnog se rješenja može doći i ovako. Krenimo iz parametarskog oblika jednadžbe pravca zapisanog matrično (r:2.1.1/3):

$$[T_{P1}, T_{P1}, ..., T_{Pn}] = [\lambda \quad 1]\cdot\mathbf{L}$$

U gornjoj relaciji jedina je nepoznanica matrica **L** koju treba odrediti. Budući da pravac kojeg tražimo za neki parametar λ prolazi kroz točku T_S, i za neki parametar λ prolazi kroz točku T_E, ostavljena nam je sloboda izbora da sami odlučimo za koje će se to vrijednost parametra λ dogoditi. Zbog toga ćemo se odlučiti na najjednostavniji mogući izbor: neka pravac prođe kroz točku T_S za λ=0, a kroz točku T_E za λ=1. Tada možemo pisati:

$$T_S = [0 \quad 1]\cdot\mathbf{L}$$
$$T_E = [1 \quad 1]\cdot\mathbf{L}$$

gdje su matrice T_S i T_E jednoretčane matrice sa onoliko stupaca koliko imamo koordinata. Gornje dvije jednadžbe možemo stopiti u jednu jednadžbu:

$$\begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}\cdot\mathbf{L}$$

Treba uočiti da T_S i T_E na lijevoj strani jednadžbe predstavljaju matrice **1**x**n** pa matrica na lijevoj strani nije **2**x**1** već **2**x**n**! Kako je u jednadžbi sve poznato osim matrice **L**, množeći jednadžbu s lijeva inverznom matricom uz **L** dobiva se:

$$\mathbf{L} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_S \\ T_E \end{bmatrix} = \begin{bmatrix} T_E - T_S \\ T_S \end{bmatrix} \qquad \text{(r:2.1.2/2)}$$

pri čemu je matrica **L** dimenzija **2**x**n**.

2.2 POSEBNI SLUČAJEVI JEDNADŽBE PRAVCA

2.2.1 2D PROSTOR

Svaka točka ima po dvije komponente, pa relacija (r:2.1.1/2):

$$(T_{P1}, T_{P2}, ..., T_{Pn}) = \overrightarrow{(v_{P1}, v_{P2}, ..., v_{Pn})} \cdot \lambda + (T_{S1}, T_{S2}, ..., T_{Sn})$$

prelazi u:

$$(T_{P1}, T_{P2}) = \overrightarrow{(v_{P1}, v_{P2})} \cdot \lambda + (T_{S1}, T_{S2}) \qquad \text{(r:2.2.1/1)}$$

Umjesto oznaka T_{P1} i T_{P2} mogli smo koristi i oznake x_p, y_p, a umjesto oznaka T_{S1} i T_{S2} mogli smo koristi i oznake x_s, y_s. Gornja jednadžba raspada se na dvije jednadžbe po komponentama:

$$T_{P1} = \vec{v}_{P1} \cdot \lambda + T_{S1}$$
$$T_{P2} = \vec{v}_{P2} \cdot \lambda + T_{S2}$$

Eliminacijom parametra λ dobiva se **eksplicitni oblik** jednadžbe pravca:

$$T_{P2} - T_{S2} = \frac{v_{P2}}{v_{P1}} (T_{P1} - T_{S1}) \qquad \text{(r:2.2.1/2)}$$

Budući da je vektor $\vec{v}$ definiran kao $\vec{v} = T_E - T_S$ (r:1.2/1), ili po komponentama:

$$v_{P1} = T_{E1} - T_{S1}$$
$$v_{P2} = T_{E2} - T_{S2}$$

uvrštavanjem u eksplicitni oblik dobiva se:

$$T_{P2} - T_{S2} = \frac{T_{E2} - T_{S2}}{T_{E1} - T_{S1}} (T_{P1} - T_{S1}) \qquad \text{(r:2.2.1/3)}$$

što nakon promjene imena odgovarajućih varijabli u "školska" imena daje:

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}(x - x_1) \qquad \text{(r:2.2.1/3')}$$

Pomnožimo li ovo sa zajedničkim nazivnikom i sredimo, dobiti ćemo **implicitni oblik** jednadžbe pravca:

$$(T_{E2} - T_{S2}) \cdot T_{P1} - (T_{E1} - T_{S1}) \cdot T_{P2} - (T_{E2} - T_{S2}) \cdot T_{S1} + (T_{E1} - T_{S1}) \cdot T_{S2} = 0 \qquad \text{(r:2.2.1/4)}$$

ili:

$$(y_2 - y_1) \cdot x - (x_2 - x_1) \cdot y - (y_2 - y_1) \cdot x_1 + (x_2 - x_1) \cdot y_1 = 0 \qquad \text{(r:2.2.1/4')}$$

Općenito implicitni oblik zapisujemo kao:

$$a \cdot T_{P1} + b \cdot T_{P2} + c = 0 \qquad \text{(r:2.2.1/5)}$$

Od zanimljivijih oblika vrijedno je još spomenuti i **segmentni oblik** jednadžbe pravca, koji direktno daje odsječke pravca na obje osi. Dobije se svođenjem implicitnog oblika na oblik:

$$\frac{T_{P1}}{l_x} + \frac{T_{P2}}{l_y} = 1 \qquad \text{(r:2.2.1/6)}$$

Pri tome vrijednosti l_x i l_y predstavljaju odsječke na obje osi. Iz eksplicitnog se oblika dobije:

$$\frac{T_{P1}}{\frac{T_{S2}v_{P1} - T_{S1}v_{P2}}{-v_{P2}}} + \frac{T_{P2}}{\frac{T_{S2}v_{P1} - T_{S1}v_{P2}}{v_{P1}}} = 1 \qquad \text{(r:2.2.1/7)}$$

2.2.2 3D PROSTOR

U 3D prostoru nije moguće pravac prikazati jednadžbom u eksplicitnom obliku. Naime, parametarski oblik jednadžbe pravca (r:2.1.1/2)

$$(T_{P1}, T_{P2}, ..., T_{Pn}) = \overrightarrow{(v_{P1}, v_{P2}, ..., v_{Pn})} \cdot \lambda + (T_{S1}, T_{S2}, ..., T_{Sn})$$

u tom slučaju prelazi u

$$(T_{P1}, T_{P2}, T_{P3}) = \overrightarrow{(v_{P1}, v_{P2}, v_{P3})} \cdot \lambda + (T_{S1}, T_{S2}, T_{S3}) \qquad \text{(r:2.2.2/1)}$$

pri čemu se svaka točka (ili vektor) opisuje pomoću tri koordinate. Ova jednadžba ekvivalent je tri jednadžbe (po jedna za svaku koordinatu), te je parametar λ nemoguće eliminirati tako da se dobije jedna jednadžba. No eliminacijom parametra λ iz dva para jednadžbi moguće je dobiti implicitni sustav jednadžbe pravca u tri dimenzije:

$$\frac{T_{P1} - T_{S1}}{v_{P1}} = \frac{T_{P2} - T_{S2}}{v_{P2}} = \frac{T_{P3} - T_{S3}}{v_{P3}} \qquad \text{(r:2.2.2/2)}$$

odnosno

$$\frac{T_{P1} - T_{S1}}{T_{E1} - T_{S1}} = \frac{T_{P2} - T_{S2}}{T_{E2} - T_{S2}} = \frac{T_{P3} - T_{S3}}{T_{E3} - T_{S3}} \qquad \text{(r:2.2.2/3)}$$

2.2.3 POTEŠKOĆE

Kada zadajemo jednadžbu pravca, želimo biti u stanju pomoću nje prikazati sve moguće pravce. Pa krenemo li od segmentnog oblika (uz ograničenje razmatranja na zapisa pravca samo u 2D prostoru), vidimo da on ne može prikazati niti jedan pravac paralelan sa bilo kojom osi. Eksplicitni oblik ima poteškoća s pravcima paralelnim s ordinatom. Jedini oblik koji nema problema je implicitni oblik. Međutim, implicitni oblik pravca za primjenu u računalima nije praktičan. Zbog toga se kao relativno dobar oblik pokazuje parametarski oblik. Dodatan "plus" ovom obliku nosi i mogućnost matričnog zapisa, koji je idealan za primjenu u računalima.

3.0 HOMOGENI PROSTOR. HOMOGENE KOORDINATE.

3.1 IDEJA

Dijeljenje sa nulom još je jedan od čestih problema pri geometrijskim proračunima. Npr. traženje sjecišta dva pravca koji su paralelni rezultirati će dijeljenjem sa nulom budući da se sjecište nalazi u beskonačnosti. Da bi se ovakvi problemi izbjegli, uvodi se homogeni prostor i homogene koordinate. Evo razmišljanja. Ukoliko je neka od koordinata jednaka beskonačno, to znači da se može dobiti dijeljenjem sa nulom. Da bi se ovo izbjeglo, potrebno je sve koordinate napisati u obliku razlomka:

$$T_{P1} = \frac{T_{Ph1}}{h} \quad T_{P2} = \frac{T_{Ph2}}{h} \qquad \text{(r:3.1/1)}$$

pri čemu su T_{P1} i T_{P2} standardne (ili radne) koordinate, a T_{Ph1} i T_{Ph2} su homogene koordinate. Uvrštavanjem ovih izraza u implicitni oblik jednadžbe pravca dobivamo:

$$a \cdot \frac{T_{Ph1}}{h} - b \cdot \frac{T_{Ph2}}{h} + c = 0$$

odnosno

$$a \cdot T_{Ph1} - b \cdot T_{Ph2} + c \cdot h = 0 \qquad \text{(r:3.1/2)}$$

U ovom slučaju koordinate T_{Ph1} i T_{Ph2} nikada neće biti jednake beskonačno. Općenito se može reći ovako. Zadana je neka proizvoljna točka T u n-dimenzionalnom prostoru. Ta točka T u homogenom će se prostoru opisivati pomoću n+1 koordinate, i takvih homogenih točaka bit će beskonačno. Npr. točka (12,24,12) u radnom prostoru u tri dimenzije imati će u homogenom prostoru zapise: (12,24,12,1), (6,12,6,0.5), (24,48,24,2), ...

Poseban slučaj je zapis beskonačnosti: ukoliko je homogena koordinata *h* točke jednaka 0, točka leži u beskonačnosti! Ako dobijemo da se dva pravca sijeku u homogenoj točci (6,6,4,2), to znači da se u radnom prostoru sijeku u točci (3,3,2). Ukoliko se sijeku u homogenoj točci (10,10,3,0), tada su ti pravci u radnom prostoru paralelni i nemaju sjecišta.

Ukratko da ponovimo najvažnije. Ukoliko je točka u radnom prostoru zadana kao:

$$T_P = (T_{P1}, T_{P2}, ..., T_{Pn})$$

tada ta točka u homogenom prostoru ima zapis:

$$T_{Ph} = (T_{Ph1}, T_{Ph2}, ..., T_{Phn}, h)$$

pri čemu je definirana veza:

$$T_{Pi} = \frac{T_{Phi}}{h} \text{ odnosno } T_{Phi} = T_{Pi} \cdot h \qquad \text{(r:3.1/3)}$$

3.2 JEDNADŽBA PRAVCA U 2D U HOMOGENOM PROSTORU

U poglavlju 2.2.1 dana je jednadžba pravca u 2D u radnom prostoru (r:2.2.1/1):

$$(T_{P1}, T_{P2}) = \overrightarrow{(v_{P1}, v_{P2})} \cdot \lambda + (T_{S1}, T_{S2})$$

Proširivanjem te jednadžbe uz relaciju (r:3.1/3):

$$T_{Pi} = \frac{T_{Phi}}{h} \text{ odnosno } T_{Phi} = T_{Pi} \cdot h$$

dolazi se do jednadžbe:

$$(T_{P1}, T_{P2}, h_P) = \overrightarrow{(v_{P1}, v_{P2}, v_{Ph})} \cdot \lambda + (T_{S1}, T_{S2}, h_S) \qquad \text{(r:3.2/1)}$$

pri tome treba obratiti pažnju na h_P i h_S komponente točaka. Nepažljiva uporaba gornje relacije može dovesti do nepoželjnih posljedica (vidi zadatak 1).

Comment: Pogledaj je li ovo dobro povezano sa zadatkom 1.1 (homogeni pravac...)

3.3 JEDNADŽBA PRAVCA U 3D U HOMOGENOM PROSTORU

U poglavlju 2.2.2 dana je jednadžba pravca u 3D u radnom prostoru (r:2.2.2/1):

$$(T_{P1}, T_{P2}, T_{P3}) = \overrightarrow{(v_{P1}, v_{P2}, v_{P3})} \cdot \lambda + (T_{S1}, T_{S2}, T_{S3})$$

Proširivanjem te jednadžbe uz relaciju (r:3.1/3):

$$T_{Pi} = \frac{T_{Phi}}{h} \text{ odnosno } T_{Phi} = T_{Pi} \cdot h$$

dolazi se do jednadžbe:

$$(T_{P1}, T_{P2}, T_{P3}, h_P) = \overrightarrow{(v_{P1}, v_{P2}, v_{P3}, v_{Ph})} \cdot \lambda + (T_{S1}, T_{S2}, T_{S3}, h_S) \qquad \text{(r:3.3/1)}$$

pri tome treba obratiti pažnju na h_P i h_S komponente točaka. Nepažljiva uporaba gornje relacije može dovesti do nepoželjnih posljedica (vidi zadatak 1).

Comment: Pogledaj je li ovo dobro povezano sa zadatkom 1.1 (homogeni pravac...)

3.4 ALTERNATIVNI OBLIK JEDNADŽBE PRAVCA U 2D U HOMOGENOM PROSTORU

Krenemo li od implicitnog oblika jednadžbe pravca u 2D (r:2.2.1/5):

$$a \cdot T_{P1} + b \cdot T_{P2} + c = 0$$

i uvrstimo u jednadžbu relaciju za homogene koordinate (r:3.1/3)

$$T_{Pi} = \frac{T_{Phi}}{h} \text{ odnosno } T_{Phi} = T_{Pi} \cdot h$$

dolazimo do jednadžbe

$$a \cdot \frac{T_{Ph1}}{h} + b \cdot \frac{T_{Ph2}}{h} + c = 0$$

odnosno nakon sređivanja

$$a \cdot T_{Ph1} + b \cdot T_{Ph2} + c \cdot h = 0 \qquad \text{(r:3.4/1)}$$

Ova se jednadžba može prikazati i u matričnom obliku, što je dosta čest slučaj. Tada ona glasi:

$$\begin{bmatrix} T_{Ph1} & T_{Ph2} & h \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0 \qquad \text{(r:3.4/2)}$$

Uvođenjem uobičajenih oznaka

$$T_P = \begin{bmatrix} T_{Ph1} & T_{Ph2} & h \end{bmatrix} \qquad \mathbf{G} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \qquad \text{(r:3.4/3)}$$

dolazi se do jednadžbe

$$T_P \cdot \mathbf{G} = 0 \qquad \text{(r:3.4/4)}$$

Ovdje se može razmotriti još jedan interesantan slučaj. Ukoliko odaberemo točku T_P takvu da leži na pravcu, tada će gornja relacija biti zadovoljena. No što ako ubacimo proizvoljnu točku u jednadžbu pravca? Gornja relacija očito neće biti jednaka nuli, i na temelju tog rezultata možemo definirati odnos točke i pravca prema slijedećem kriteriju:

$$T_P \cdot \mathbf{G} \begin{cases} > 0, T_P \text{ je iznad pravca} \\ = 0, T_P \text{ je na pravcu} \\ < 0, T_P \text{ je ispod pravca} \end{cases} \qquad \text{(r:3.4/5)}$$

Poznavanjem dviju točaka T_A i T_B na jednostavan se način može doći do pravca ako se napravi x-produkt tih točaka:

$$T_A = (T_{A1}, T_{A2}, h_A)$$
$$T_B = (T_{B1}, T_{B2}, h_B)$$

$$T_A \times T_B = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{A1} & T_{A2} & h_A \\ T_{B1} & T_{B2} & h_B \end{vmatrix} = \vec{i} \cdot (T_{A2}h_B - T_{B2}h_A) - \vec{j} \cdot (T_{A1}h_B - T_{B1}h_A) + \vec{k} \cdot (T_{A1}T_{B2} - T_{A2}T_{B1})$$

$$= \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{A2}h_B - T_{B2}h_A \\ -(T_{A1}h_B - T_{B1}h_A) \\ T_{A1}T_{B2} - T_{A2}T_{B1} \end{bmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \mathbf{G}$$

Slično se poznavanjem dvaju pravaca može odrediti točka u kojoj se oni sijeku ako se napravi x-produkt:

$$\mathbf{G}_1 = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix} \qquad \mathbf{G}_2 = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$$

$$\mathbf{G}_1^T \times \mathbf{G}_2^T = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{vmatrix} = \vec{i} \cdot (b_1c_2 - b_2c_1) - \vec{j} \cdot (a_1c_2 - a_2c_1) + \vec{k} \cdot (a_1b_2 - a_2b_1) =$$

$$\begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} b_1c_2 - b_2c_1 \\ -(a_1c_2 - a_2c_1) \\ a_1b_2 - a_2b_1 \end{bmatrix} = T_P^T$$

Operacija transponiranja nužna je ukoliko se žele napisati jednadžbe u skladu sa pravilima matematike. Ovdje se može prodiskutirati opravdanost uvođenja homogenih koordinata. Naime, zahvaljujući homogenim koordinatama, matrični račun ne daje niti jedno dijeljenje. Ovo znači da će i paralelni pravci imati sjecište! Doista, ukoliko su pravci paralelni, tada su im koeficijenti smjera jednaki i treća komponenta točke T_P iznositi će nula, što prema definiciji znači točku u beskonačnosti! Evo jednostavnog dokaza. Jednadžba pravca u 2D sa homogenim koordinatama glasi

$$a \cdot T_{Ph,1} + b \cdot T_{Ph,2} + c \cdot h = 0$$

što se može napisati kao:

$$T_{Ph,2} = -\frac{a}{b} \cdot T_{Ph,1} - \frac{c}{b} \cdot h = -k \cdot T_{Ph,1} - \frac{c}{b} \cdot h$$

gdje je *k* koeficijent smjera pravca. Ukoliko pravci G_1 i G_2 imaju jednake koeficijente smjera, to znači prema gornjoj relaciji da vrijedi:

$$-\frac{a_1}{b_1} = -\frac{a_2}{b_2} \Rightarrow a_1 b_2 = a_2 b_1 \Rightarrow a_1 b_2 - a_2 b_1 = 0$$

kako je poslijednja relacija upravo jednadžba zadnje koordinate točke sjecišta, slijedi da je uz paralelne pravce zadnja koordinata (tj. homogeni faktor) točke sjecišta jednaka 0.

4.0 RAVNINA

4.1 JEDNADŽBA RAVNINE

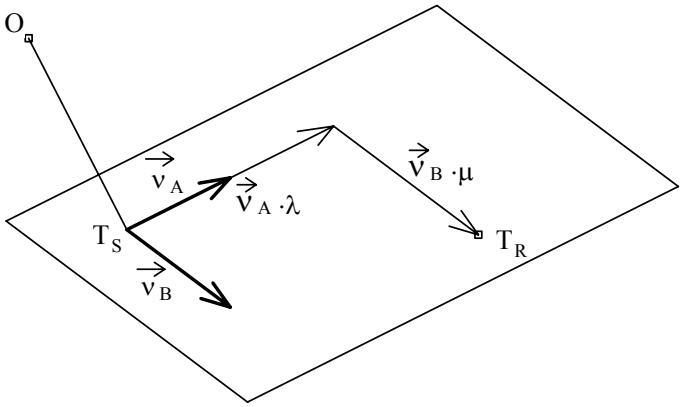
Ravnine opet možemo definirati u n-dimenzijskim prostorima, no kako se u računalnoj grafici koriste jedino ravnine u 3D, razmatranja ćemo ograničiti na taj tip. U 3D prostoru ravnina je određena s dva vektora koji leže u njoj i nisu kolinearni, te jednom točkom ravnine. Krenuvši ovakvom definicijom dolazimo vrlo jednostavno do jednadžbe ravnine u parametarskom obliku:

$$T_R = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu + T_S \qquad \text{(r:4.1/1)}$$

ili riječima:

krenuvši od točke T_S svaka točka ravnine može se dobiti pomakom za $\vec{v}_A \cdot \lambda$ i $\vec{v}_B \cdot \mu$, pri čemu su λ i μ realni parametri.

T_R predstavlja bilo koju točku ravnine. Slijedeća slika jasno ilustrira relaciju (r:4.1/1).



Zapis relacije (r:4.1/1) ekvivalentan je sustava od tri jednadžbe:

$$T_{R1} = v_{A1} \cdot \lambda + v_{B1} \cdot \mu + T_{S1}$$
$$T_{R2} = v_{A2} \cdot \lambda + v_{B2} \cdot \mu + T_{S2}$$
$$T_{R3} = v_{A3} \cdot \lambda + v_{B3} \cdot \mu + T_{S3}$$

Ovo se može prikazati i matričnim zapisom:

$$T_R = [T_{R1} \quad T_{R2} \quad T_{R3}] = [\lambda \quad \mu \quad 1] \cdot \begin{bmatrix} v_{A1} & v_{A2} & v_{A3} \\ v_{B1} & v_{B2} & v_{B3} \\ T_{S1} & T_{S2} & T_{S3} \end{bmatrix}$$

No ovaj se oblik može i pojednostavniti. Ukoliko nađemo vektor $\vec{v}_n$ koji je okomit i na vektor $\vec{v}_A$ i na vektor $\vec{v}_B$, a u parametarskoj jednadžbi T_S prebacimo na lijevu stranu

$$T_R - T_S = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu$$

te razliku točaka na lijevoj strani proglasimo vektorom $\vec{v}_R$

$$\vec{v}_R = \vec{v}_A \cdot \lambda + \vec{v}_B \cdot \mu$$

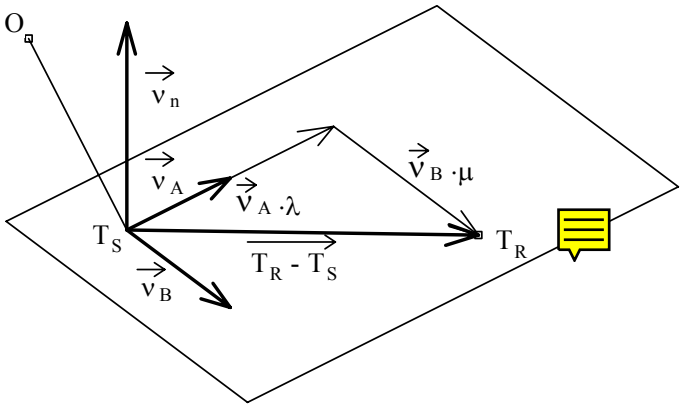
Zatim sve pomnožimo vektorom $\vec{v}_n$, dobiti ćemo

$$\vec{v}_R \cdot \vec{v}_n = 0 \qquad \text{(r:4.1/2)}$$

Cijela desna strana jednadžbe je nestala jer vektori $\vec{v}_A$ i $\vec{v}_B$ pomnoženi sa $\vec{v}_n$ daju nulu (jer smo zahtijevali da vektor $\vec{v}_n$ bude okomit i na vektor $\vec{v}_A$ i na vektor $\vec{v}_B$)! Ovo nas vodi na zapis ravnine pomoću njezine normale:

Comment: Pogledaj je li ovo dobro povezano sa zadatkom 1.1 (homogeni pravac...)

$$(T_R - T_S) \cdot \vec{v}_n = 0 \qquad \text{(r:4.1/3)}$$



Vektor $\vec{v}_n$ naziva se normala ravnine iz očitih razloga. Ukoliko znamo vektore $\vec{v}_A$ i $\vec{v}_B$, vektor $\vec{v}_n$ možemo dobiti na različite načine (i taj vektor nije jednoznačan; postoji beskonačno mnogo vektora koji su okomiti na ravninu; istina, svi su kolinearni i razlika između njih je isključivo u njihovoj duljini, odnosno normi), a jedan od načina je i x-produkt. Možemo odabrati:

$$\vec{v}_n = \vec{v}_A \times \vec{v}_B \qquad \text{(r:4.1/4)}$$

Ukoliko u zapis ravnine pomoću njezine normale (r:4.1/3) $(T_R - T_S) \cdot \vec{v}_n = 0$ uvrstimo vrijednosti po komponentama, dobivamo:

$$(T_{R1} - T_{S1}) \cdot v_{n1} + (T_{R2} - T_{S2}) \cdot v_{n2} + (T_{R3} - T_{S3}) \cdot v_{n3} = 0$$

odnosno nakon množenja i grupiranja:

$$T_{R1} \cdot v_{n1} + T_{R2} \cdot v_{n2} + T_{R3} \cdot v_{n3} - (T_{S1} \cdot v_{n1} + T_{S2} \cdot v_{n2} + T_{S3} \cdot v_{n3}) = 0 \qquad \text{(r:4.1/5)}$$

ili

$$A \cdot T_{R1} + B \cdot T_{R2} + C \cdot T_{R3} + D = 0 \qquad \text{(r:4.1/6)}$$

Ako sada u dobivenu jednadžbu ubacimo homogene koordinate, dobiva se:

$$A \cdot \frac{T_{Rh1}}{h_R} + B \cdot \frac{T_{Rh2}}{h_R} + C \cdot \frac{T_{Rh3}}{h_R} + D = 0$$

ili nakon sređivanja

$$A \cdot T_{Rh1} + B \cdot T_{Rh2} + C \cdot T_{Rh3} + D \cdot h = 0 \qquad \text{(r:4.1/7)}$$

što možemo prikazati i matrično

$$\begin{bmatrix} T_{Rh1} & T_{Rh2} & T_{Rh3} & h_R \end{bmatrix} \cdot \begin{bmatrix} A \\ B \\ C \\ D \end{bmatrix} = 0 \qquad \text{(r:4.1/8)}$$

Ova relacija koristi se vrlo često i korisno ju je poznavati. No ona je bitno iz još jednog razloga. Kod jednadžbe pravca na temelju slične relacije definirali smo odnos točke i pravca. Pomoću ove relacije definirati ćemo odnos točke i ravnine! Jednadžbu (r:4.1/8) simbolički možemo zapisati:

$$T_{Rh} \cdot \mathbf{R} = 0 \qquad \text{(r:4.1/9)}$$

Dakle, za svaku točku koja je na ravnini relacije je zadovoljena. No ako uvrstimo za T_R neku točku koja ne pripada ravnini, gornja relacija neće biti zadovoljena. Na temelju tog rezultata definira se odnos točke i ravnine:

$$T_{Rh} \cdot \mathbf{R} \quad \begin{cases} > 0, T_{Rh} \text{ je iznad ravnine } \mathbf{R} \\ = 0, T_{Rh} \text{ je na ravnini } \mathbf{R} \\ < 0, T_{Rh} \text{ je ispod ravnine } \mathbf{R} \end{cases} \qquad \text{(r:4.1/10)}$$

4.2 JEDNADŽBA RAVNINE KROZ TRI TOČKE

Ravnina je jednoznačno određena sa tri točke. Da bismo odredili jednadžbu ravnine kroz tri točke, krenuti ćemo od parametarskog oblika ravnine u homogenom prostoru:

$$T_{Rh} = \begin{bmatrix} T_{Rh1} & T_{Rh2} & T_{Rh3} & h_R \end{bmatrix} = \begin{bmatrix} \lambda & \mu & 1 \end{bmatrix} \cdot \begin{bmatrix} v_{A1} & v_{A2} & v_{A3} & h_A \\ v_{B1} & v_{B2} & v_{B3} & h_B \\ T_{S1} & T_{S2} & T_{S3} & h_S \end{bmatrix} = \begin{bmatrix} \lambda & \mu & 1 \end{bmatrix} \cdot \mathbf{R}$$

(r:4.2/1)

Ovdje je dakle ravnina određena sa dva vektora i točkom. Pri tome su vektori i točke četverokomponentni jer radimo u tri dimenzije (3 komponente) u homogenom prostoru (još jedna komponenta).

Kao i kod određivanja jednadžbe pravca, i ovdje možemo raditi istom filozofijom. Jednadžba će kroz točku T_A proći za neki λ i neki μ, kroz točku T_B proći za neki drugi λ i neki drugi μ, i kroz točku T_C proći za neki treći λ i neki treći μ. No tada idemo mi odabrati za koje će to λ i μ ravnina proći kroz koje točke. Odabrati ćemo da ravnina prolazi kroz T_A za λ=0 i μ=0, kroz T_B za λ=1 i μ=0 te kroz T_C za λ=1 i μ=1. Tada će vrijediti:

$$\begin{aligned} T_{Ah} &= \begin{bmatrix} T_{Ah1} & T_{Ah2} & T_{Ah3} & h_A \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \underline{R} \\ T_{Bh} &= \begin{bmatrix} T_{Bh1} & T_{Bh2} & T_{Bh3} & h_B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \cdot \underline{R} \\ T_{Ch} &= \begin{bmatrix} T_{Ch1} & T_{Ch2} & T_{Ch3} & h_C \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \cdot \underline{R} \end{aligned}$$

odnosno nakon stapanja u jednu jednadžbu:

$$\begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \cdot \underline{R}$$

odakle slijedi:

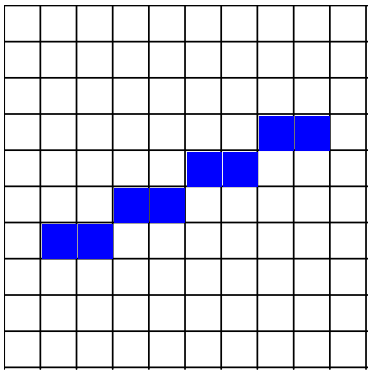
$$\underline{R} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 \\ -1 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} T_{Bh} - T_{Ah} \\ T_{Ch} - T_{Ah} \\ T_{Ah} \end{bmatrix} \qquad \text{(r:4.2/2)}$$

5.0 BRESENHAMOV POSTUPAK CRTANJA LINIJE

5.1 UVOD

Kada na monitoru želimo nacrtati liniju, za to obično pozivamo ugrađenu funkciju jezika u kojem pišemo program, i pri tome zadajemo koordinate početne točke i završne točke. Pri tome se podrazumijeva da će funkcija sama u što kraćem vremenu nacrtati tu liniju. No na koji način se izvodi to brzo crtanje linije? Postupak kojim se to danas radi obično je implementacija Bresenhamovog postupka. Pri tome treba imati u vidu da je monitor dvodimenzionalna prikazna jedinica, odnosno sve se događa u jednoj ravnini.

Monitor je također rasterska prikazna jedinica. Ovo znači da se slika koju prikazuje monitor sastoji od niza elementarnih djelića slike - pixela (picture element). Ovo znači da je slika koju prikazuje monitor diskretna. Moguće je osvijetliti pixel na poziciji npr. (0,0) , no nije moguće osvijetliti pixel na poziciji (1.13,2.52) jer takav pixel ne postoji. Posljedica ovakve diskretizacije slike je da prilikom crtanja npr. linija dolazi do nazubljenosti iste. Više o ovoj pojavi reći ćemo u nastavku.



Slika 5.1.1. Nazubljenost linije uslijed rasterizacije

5.2 JEDNADŽBA PRAVCA

Prilikom crtanja linije poznate su nam koordinate početne i završne točke. Isto tako znamo da liniju crtamo u ravnini. Zaboravimo na tren da liniju crtamo na zaslonu. Zamislimo da imamo idealnu prikaznu jedinicu koja može prikazati sve što poželimo. Prva stvar koju moramo napraviti je izračunati koje sve točke moramo osvijetliti. Da bismo si olakšali razmatranje, zadajmo točke na slijedeći način:

- Početna točka T_S ima koordinate (x_s,y_s).
- Završna točka T_E ima koordinate (x_e,y_e).
- Vrijedi: $x_s<x_e$, $y_s<y_e$.
- Pravac je pod kutom manjim ili jednakim 45°.

Kasnije ćemo se osloboditi ovih ograničenja no za početak neka ograničenja vrijede. Koordinate točaka T_S i T_E u ovom slučaju nisu označene standardnim oznakama ($T_{S,1}$,$T_{S,2}$) i ($T_{E,1}$,$T_{E,2}$) već su iskorištene oznake (x_s,y_s) i (x_e,y_e) budući da ćemo cijeli postupak izvoditi imajući na umu računala gdje su zasloni dvodimenzijski sa standardnim oznakama x i y za pojedine osi.

Jednadžba pravca kroz dvije točke može se napisati prema relaciji (r:2.2.1/3') u obliku:

$$y - y_s = \frac{y_e - y_s}{x_e - x_s}(x - x_s)$$

ili nakon sređivanja i uvođenja oznake *a* za tangens smjera pravca, te oznake *b* za odsječak na osi y:

$$y = a \cdot x + b \text{ pri čemu je } a = \frac{y_e - y_s}{x_e - x_s},\ b = -a \cdot x_s + y_s$$

a i *b* su izdvojeni kao zasebne konstante jer se mogu izračunati samo jednom na početku, obzirom da ovise samo o konstantama. Umjesto oznaka *a* i *b* često su u uporabi i oznake *k* za tangens smjera i *l* za odsječak na osi y.

Uz ovu posljednju formulu pravac se može nacrtati slijedećim trivijalnim algoritmom:

```
a = (ye-ys)/(double)(xe-xs); b = -a*xs+ys;

for( x = xs; x <= xe; x++ ) {
 y = a*x + b;
 y = zaokruži(y);
 osvijetli_pixel(x,y);
}
```

Obzirom na diskretiziranost monitora potrebno je za svaki x pronaći odgovarajući y i tu točku osvijetliti. No ovaj algoritam, iako radi, ima jedan veliki nedostatak: sporost. Osnovni problem ovog algoritma je množenje u petlji. Za svaki x odgovarajući y računa se množenjem i to u aritmetici pomičnog zareza! Algoritam vapi za poboljšanjem!

5.3 BRESENHAMOV POSTUPAK

Pogledamo još jednom što prethodno opisani algoritam radi. Varijabla x mijenja se u petlji od vrijednosti x_s do vrijednosti x_e po jedan i za svaki x se računa odgovarajući y uz uporabu operacije množenja. Pogledajmo malo bolje što to program zapravo računa:

$$y(x = x_s) = a \cdot x_s + b = y_s$$
$$y(x = x_s + 1) = a \cdot (x_s + 1) + b = a \cdot x_s + b + a = y_s + a = y(x = x_s) + a$$
$$y(x = x_s + 2) = a \cdot (x_s + 2) + b = a \cdot x_s + b + 2a = y(x = x_s + 1) + a$$
$$y(x = x_s + 3) = a \cdot (x_s + 3) + b = a \cdot x_s + b + 3a = y(x = x_s + 2) + a$$
...

Pogledamo li desne strane u svakom retku, možemo vidjeti da se svaki redak može dobiti tako da se prethodnom doda vrijednost varijable *a*. To je izvrstan rezultat jer nam ukazuje na to da nam više ne treba vremenski zahtjevno množenje! Možda bi bio problem sa prvim retkom, jer po našoj filozofiji on nema prethodnika pa bismo tu trebali koristiti množenje, no nije tako. Vrijednost prvog retka već nam je zadana i iznosi y_s. Pogledajmo sada što smo dobili.

```
int x,y_pom,;
double a,y;

a = (ye-ys)/(double)(xe-xs);

y=ys;
for( x = xs; x <= xe; x++ ) {
 y_pom = zaokruži(y);
 osvijetli_pixel(x,y_pom);
 y = y + a;
}
```

Poboljšanje je već značajno. Izvan petlje vrijednost varijable y postavlja se na početnu vrijednost. U petlji je uvedena pomoćna varijabla y_pom koja pamti vrijednost varijable y nakon zaokruživanja jer varijablu y ne smijemo dirati budući da nam treba i u nastavku.

Pogledajmo što još ne valja. Prečesto zovemo funkciju zaokruži(y)! Bilo bi jako zgodno kada bismo mogli i bez zaokruživanja znati koja je zaokružena vrijednost y koordinate, ili ako ništa drugo, pokušati funkciju zvati rjeđe. I to se može! Evo kako.

5.4 BRESENHAMOV POSTUPAK

Ideja je slijedeća. Početnu zaokruženu vrijednost y koordinate znamo: to je y_s, tj. $y=y_s$. Trebamo dakle osvijetliti pixel na poziciji (x_s,y_s). Kada se pomaknemo za jedan pixel udesno, vrijednost y koordinate poveća se za koeficijent *a*. Kako smo postavili ograničenje da je pravac pod kutom manjim ili jednakim 45°, to znači da se vrijednost varijable ***a*** kreće između 0 i 1. Uzmimo za primjer da *a* iznosi 0.2. Tada nakon jednog pomaka u desno y se je povećao za 0.2 te iznosi $y=y_s+0.2$. Zaokruživanjem ove vrijednosti dolazi se do vrijednosti y_s te se y koordinata točke koju trebamo osvijetliti ne razlikuje od prethodnog koraka. Znači, trebamo osvijetliti pixel na poziciji (x_s+1,y_s). Pomaknimo se za još jedan pixel u desno. Y koordinatu opet treba uvećati za vrijednost varijable *a*. Sada to iznosi $y=y_s+0.2+0.2= y_s+0.4$. No zaokruživanjem se opet dolazi do vrijednosti $y=y_s$. Znači da trebamo osvijetliti pixel na poziciji (x_s+2,y_s). Idemo za još jedan pixel u desno: $y=y_s+0.4+0.2= y_s+0.6$. I konačno zaokruživanjem ove vrijednosti penjemo se za jedan pixel prema gore: $y=y_s+1$, i osvijetljavamo pixel na poziciji (x_s+2,y_s+1).

Označimo cjelobrojnu vrijednost koordinate y sa y_c, a pridodani decimalni dio sa y_f. Tada gornji postupak možemo opisati ovako: osvjetljavamo točke na poziciji (x_s+k,y_c). Na početku je $y_c=y_s$ a $y_f=0$. Svakim pomakom u desno y_f uvećavamo za koeficijent *a*. Onog trenutka kada y_f pređe (ili dođe na) 0.5, uvećavamo y_c za jedan.

U ovom trenutku možemo nastaviti na dva načina:

1. Možemo nastaviti prilikom pomaka u desno sa dodavanjem vrijednosti varijable *a* varijabli y_f. U tom slučaju slijedeće uvećavanje y_c-a za jedan biti će kada y_f prekorači 1.5, pa slijedeće kada prekorači 2.5 itd. No ova ideja i nije baš najbolja jer opet dozvoljava da y_f postane veći od jedan pa moramo voditi računa da svaki puta cjelobrojni dio "zaboravimo" i sl. što je opet vremenski zahtjevno.
2. Možemo od y_f oduzeti 1, i time poništiti nagomilanu pogrešku te y_f zadržati u opsegu od -0.5 do 0.5 čime nam je usporedba znatno olakšana.

Budući da je drugi postupak bolji, odlučiti ćemo se za njega. No u tom slučaju treba još pojasniti zašto se od y_f oduzima baš 1. Razmislimo malo što nam predstavlja y_f. Crtajmo liniju iz početne točke sa y koordinatom jednakom 0, i ponovimo ukratko gore opisani postupak. Prvo smo bili u y=0, odnosno y_c=0, y_f=0. Zatim smo došli u y=0.2, tj. y_c=0, y_f=0.2. Zatim y=0.4, tj. y_c=0, y_f=0.4, i konačno y=0.6, tj. y_c=1, y_f=0.6. Vrijednost pohranjena u y_f nam zapravo govori koliko smo pobjegli od cjelobrojne koordinate. Npr. za y=0.4, tj. y_c=0, y_f=0.4, osvijetliti ćemo pixel sa y koordinatom 0, dok smo realno gledajući već 0.4 pixela iznad. Onog trena kada od pixela pobjegnemo za 0.5 ili više (npr. za y_f=0.6), prema dogovoru uvećavamo y_c za jedan. No tada više nismo u točci sa y koordinatom 0, već sa y koordinatom 1. A to za pogrešku (y_f) znači da više nismo 0.6 pixela iznad, nego 0.4 pixela ispod trenutnog pixela određenog sa y_c, što znači pogreška sa y_f=0.6 prelazi u y_f=0.6-1=-0.4.

Implementacija ovog algoritma dana je u nastavku.

```
int x,yc;
double a,yf;

a = (ye-ys)/(double)(xe-xs);

yc=ys; yf=0.;
for( x = xs; x <= xe; x++ ) {
 osvijetli_pixel(x,yc);
 yf=yf+a;
 if(yf>=0.5) {
  yf=yf-1.;
  yc=yc+1;
 }
}
```

Uvedemo li još samo jednu minornu modifikaciju, a to je da inicijalno za y_f uzmemo vrijednost -0.5, te poslije usporedbu radimo sa 0.5-0.5=0, dakle nulom, dobili smo Bresenhamov algoritam!

Bresenhamov algoritam

```
int x,yc;
double a,yf;

a = (ye-ys)/(double)(xe-xs);

yc=ys; yf=-0.5;
for( x = xs; x <= xe; x++ ) {
```

```
 osvijetli_pixel(x,yc);
 yf=yf+a;
 if(yf>=0.) {
  yf=yf-1.;
  yc=yc+1;
 }
}
```

Može li bolje? Naravno...

5.5 MODIFIKACIJA BRESENHAMOVOG POSTUPKA

Do osnovnog Bresenhamovog postupka došli smo iz samo jednog zahtjeva: brzina! Pogledom na sam algoritam postavlja se pitanje može li još bolje? Trn u oku u opisanom algoritmu svakako su brojevi sa pomičnim zarezom. Pokazuje se da se cijeli algoritam može prebaciti u cjelobrojnu domenu, a opće je poznato da su operacije sa cijelim brojevima daleko brže od aritmetike u pomičnom zarezu. Stoga ćemo ovo poglavlje posvetiti prilagodbi osnovnog Bresenhamovog algoritma tako da proradi sa cijelim brojevima.

Osnovni element koji je uveo decimalne brojeve u igru bio je koeficijent smjera koji smo računali prema formuli:

$$a = \frac{y_e - y_s}{x_e - x_s}$$

Ovaj koeficijent koristili smo prilikom izračuna pogreške (y_f varijabla). Pogrešku smo računali prema formuli:

$$y_f = y_f + a$$

Idemo to malo raspisati. Uvrštavanjem izraza za *a* dobiva se:

$$y_f = y_f + \frac{y_e - y_s}{x_e - x_s} = \frac{y_f \cdot (x_e - x_s) + y_e - y_s}{x_e - x_s} \Big/ \cdot (x_e - x_s)$$

$$y_f \cdot (x_e - x_s) = y_f \cdot (x_e - x_s) + y_e - y_s$$

Uvođenjem

$$y_f' = y_f \cdot (x_e - x_s)$$

dobivamo

$$y_f' = y_f' + y_e - y_s .$$

Ovaj posljednji redak nam govori da umjesto dosadašnje pogreške možemo pamtiti pogrešku pomnoženu sa koeficijentom x_e-x_s. Kako su to sve cijeli brojevi, ovdje smo se riješili sporih decimalnih brojeva. Slijedeći korak je rješavanje inicijalnih uvjeta. Naime, inicijalno vrijednost pogreške postavljamo na $-\frac{1}{2}$. To opet možemo raspisati:

$$y_f = -\frac{1}{2} \Big/ \cdot (x_e - x_s)$$

$$y_f \cdot (x_e - x_s) = -\frac{x_e - x_s}{2}$$

$$y_f^{'} = -\frac{x_e - x_s}{2}$$

Skoro pa dobro. Još da se riješimo dvojke u nazivniku i svi naši problemi su riješeni. Dvojke ćemo se jednostavno riješiti tako da sve pomnožimo sa 2. No tada na lijevoj strani umjesto nove pogreške stoji njezina dvostruka vrijednost! To će nas još prisiliti da kod izvoda nove pogreške sve pomnožimo sa dvojkom, te će se dobiti redom:

- za pogrešku ćemo koristiti izraz
  $$y_f = y_f + \frac{y_e - y_s}{x_e - x_s} = \frac{y_f \cdot (x_e - x_s) + y_e - y_s}{x_e - x_s} \Big/ \cdot 2(x_e - x_s)$$
  $$2y_f \cdot (x_e - x_s) = 2y_f \cdot (x_e - x_s) + 2 \cdot (y_e - y_s)$$
  $$y_f^{'} = 2y_f \cdot (x_e - x_s)$$
  $$y_f^{'} = y_f^{'} + 2 \cdot (y_e - y_s)$$
- tada inicijalno dodjeljivanje prelazi u
  $$y_f = -\frac{1}{2} \Big/ \cdot 2(x_e - x_s)$$
  $$2y_f \cdot (x_e - x_s) = -(x_e - x_s)$$
  $$y_f^{'} = -(x_e - x_s)$$
- uz ove oznake oduzimanje jedinice od pogreške može se zapisati
  $$y_f = y_f - 1 / \cdot 2(x_e - x_s)$$
  $$2y_f \cdot (x_e - x_s) = 2y_f \cdot (x_e - x_s) - 2(x_e - x_s)$$
  $$y_f^{'} = y_f^{'} - 2(x_e - x_s)$$

Imajući u vidu ove izmjene, može se napisati Bresenhamov algoritam sa cijelim brojevima. Pri tome će se umjesto oznake $y_f^{'}$ koristiti standardna y_f oznaka podrazumijevajući da se pri tome govori o novoj pogreški.

Bresenhamov algoritam sa cijelim brojevima

```
int x,yc;
int a,yf;

a = 2*(ye-ys);

yc=ys; yf=-(xe-xs);
for( x = xs; x <= xe; x++ ) {
 osvijetli_pixel(x,yc);
 yf=yf+a;
 if(yf>=0) {
```

```
  yf=yf-2*(xe-xs);
  yc=yc+1;
 }
}
```

5.6 KUTOVI OD 0° DO 90°

Sada kada smo se riješili decimalnih brojeva, vrijeme da se riješimo i ograničenja vezanih uz kutove. Pa idemo postupno. Razmatranje ćemo raditi za postupak sa cijelim brojevima, ali ćemo imati stalno u vidu kako smo do toga postupka zapravo došli. Pravac pod kutom od 90° za algoritam da decimalnim brojevima (osnovna izvedba) bio je neizvediv. Naime, tangens kuta je tada beskonačno i mi imamo dijeljenje sa nulom (i vjerojatno nasilni prekid programa). No algoritam sa cijelim brojevima nema dijeljenja pa ovo više nije problem. Za kutove od 0° do 45° algoritam smo već izveli. No zašto smo se ograničili na to područje? Odgovor opet treba tražiti u iznosu tangensa kuta. Naime, na tom intervalu on se kreće u granicama od nula do jedan, osiguravajući pri tome da ćemo se pomakom za jedan pixel u desno pomaknuti maksimalno za jedan pixel prema gore. No što ako dopustimo da tangens postane veći od 1? To znači da bismo se jednim pomakom u desno mogli pomaknuti i za više pixela prema gore (vodeći računa o tome da ih sve treba osvijetliti). Međutim, pojavljuje se problem u određivanju koje sve pixele pri tom penjanju treba osvijetliti. Da ne ulazimo dublje u prazne rasprave o ovome, rješenje ćemo potražiti na jedan drugi način. Ako je kut između pravca i apscise veći od 45°, tada je očito kut između pravca i ordinate manji od 45°! Znači, ukoliko sada jednostavno zamijenimo osi prilikom postupka crtanja, umjesto problematičnog višepixelnog pomaka prema gore sa svakim pomakom u desno dobivamo maksimalno jednopixelni pomak u desno sa svakim pomakom prema gore. I to je rješenje! Sve što treba napraviti je ispitati je li tangens kuta veći od 1 (tj. kut veći od 45°). Ako nije, koristimo već poznati algoritam; ako je mijenjamo uloge osima i kopiramo algoritam. Evo implementacije:

```
void crtaj_liniju2(int xs, int ys, int xe, int ye) {
 int x,yc;
 int a,yf;

 if(ye-ys <= xe-xs) {
  a = 2*(ye-ys);
  yc=ys; yf=-(xe-xs);
  for( x = xs; x <= xe; x++ ) {
   osvijetli_pixel(x,yc);
   yf=yf+a;
   if(yf>=0) {
    yf=yf-2*(xe-xs);
    yc=yc+1;
   }
  }
 } else {
  x=xe; xe=ye; ye=x;
  x=xs; xs=ys; ys=x;
  a = 2*(ye-ys);
  yc=ys; yf=-(xe-xs);
```

```
  for( x = xs; x <= xe; x++ ) {
   osvijetli_pixel(yc,x);
   yf=yf+a;
   if(yf>=0) {
    yf=yf-2*(xe-xs);
    yc=yc+1;
   }
  }
 }
}
```

Funkcija krene sa ispitivanjem da li je razlika po y-u manja od razlike po x-u. Ako je, pravac je pod kutom manjim od 45° i koristi se već izvedeni algoritam. Ukoliko je razlika po y-u veća od razlike po x-u, koristeći pomoćnu varijablu ***x*** zamjenjuju se x i y koordinate točaka i nastavlja se sa crtanjem. U petlji se u ovom slučaju može uočiti još jedna razlika: funkciji osvijetli_pixel predajemo na prvi pogled zamijenjene koordinate. No imajući u vidu da smo već prethodno napravili jednu zamjenu, ono što se mijenja u varijabli y_c zapravo je vrijednost x komponente točke i obrnuto.

5.7 KUTOVI OD 0° DO -90°

Osnovna razlika od prethodnog slučaja je okomito gibanje koje sada ide prema dolje, umjesto prema gore. Zbog toga y_c treba umanjivati, a ne uvećavati za jedan. Isto tako postupak pri računanju pogreške se ponešto modificira, no modifikacije su čisto kozmetičke prirode. Evo algoritma:

```
void crtaj_liniju3(int xs, int ys, int xe, int ye) {
 int x,yc;
 int a,yf;

 if(-(ye-ys) <= xe-xs) {
  a = 2*(ye-ys);
  yc=ys; yf=(xe-xs);
  for( x = xs; x <= xe; x++ ) {
   osvijetli_pixel(x,yc);
   yf=yf+a;
   if(yf<=0) {
    yf=yf+2*(xe-xs);
    yc=yc-1;
   }
  }
 } else {
  x=xe; xe=ys; ys=x;
  x=xs; xs=ye; ye=x;
  a = 2*(ye-ys);
  yc=ys; yf=(xe-xs);
  for( x = xs; x <= xe; x++ ) {
   osvijetli_pixel(yc,x);
```

```
    yf=yf+a;
    if(yf<=0) {
     yf=yf+2*(xe-xs);
     yc=yc-1;
    }
   }
  }
}
```

Od izmjena u odnosu na kod koji crta pravce pod kutovima od 0° do 90° mogu se navesti slijedeće:

- inicijalna vrijednost pogreške promijenila je predznak
- y komponenta se ne uvećava za jedan već se smanjuje za jedan
- kako je $y_s>y_e$ ispitivanje $(y_e-y_s)<=(x_e-x_s)$ pretvoreno je $-(y_e-y_s)<=(x_e-x_s)$ da bi se poništio negativan predznak rezultata na lijevoj strani

5.8 SVI KUTOVI

Do sada smo obradili slučajeve kada smo se pri crtanju pravca gibali od lijeva u desno. Ostalo nam je obraditi crtanje pravca kojeg bismo trebali crtati s desna u lijevo. No za ovo nam ne trebaju posebni algoritmi: ukoliko je pravac zadan tako da se crta s desna u lijevo, zamjenom početne i završne koordinate dobivamo pravac koji se crta s lijeva u desno.

```
void crtaj_liniju4(int xs, int ys, int xe, int ye) {
 if( xs <= xe ) {
  if( ys <= ye ) crtaj_liniju2(xs,ys,xe,ye);
  else crtaj_liniju3(xs,ys,xe,ye);
 } else {
  if( ys >= ye ) crtaj_liniju2(xe,ye,xs,ys);
  else crtaj_liniju3(xe,ye,xs,ys);
 }
}
```

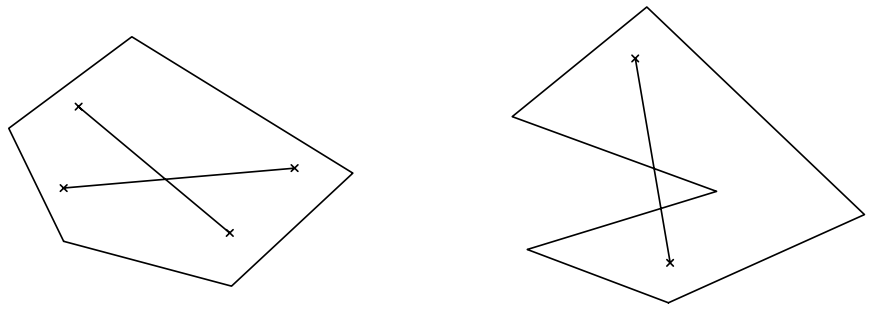
Funkcija ovisno o predanim točkama poziva odgovarajuću funkciju i pri tome po potrebi zamijeni početnu i krajnju točku. Prisjetimo se što smo već implementirali:

- funkcija crtaj_liniju2 crta linije pod kutovima od 0° do 90°, uz $x_s<x_e$
- funkcija crtaj_liniju3 crta linije pod kutovima od 0° do -90°, uz $x_s<x_e$
- funkcija crtaj_liniju4 kombinirajući prethodne dvije crta linije pod svim kutovima, uz proizvoljan odnos koordinata x_s i x_e

6.0 KONVEKSNI POLIGONI

6.1 DEFINICIJA KONVEKSNOG POLIGONA

Poligon je lik koji nastaje slijednim povezivanjem zadanih točaka koje predstavljaju vrhove poligona. Pri tome se povezuje i završna točka sa početnom. Točke koje predstavljaju vrhove poligona označavati ćemo oznakom T_i, gdje je ***i*** indeks vrha. Točke ćemo zadavati u radnom prostoru, i redoslijed zadavanja biti će točno definiran, jer taj redoslijed određuje kako ćemo povezivati točke poligona. Povezivanje vrhova poligona obavlja se linijama (pravcima), koje čine bridove poligona. Bridove ćemo označavati oznakom b_i, gdje je ***i*** indeks brida. Može se uočiti da je broj bridova uvijek jednak broju vrhova poligona. Poligon je konveksan, ukoliko ne postoji spojnica proizvoljnih dviju točaka poligona koja bi prolazila izvan poligona. Jasnije tumačenje daje slika 6.1.1.



a) Konveksni poligon b) Konkavni poligon

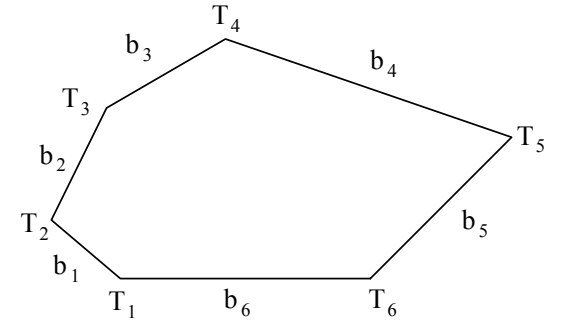
6.1.1.

6.2 MATEMATIČKI OPIS POLIGONA

Osnovno što će nas kod poligona interesirati, i pomoću čega ćemo donositi razne zaključke jesu jednadžbe bridova poligona. U ovom razmatranju ograničiti ćemo se na poligone u dvije dimenzije. Prema slici 6.2.1, brid b_i određen je sa vrhovima

$$b_i \ldots \begin{cases} T_i, T_{i+1} & \text{za } 0 < i < n \\ T_i, T_1 & \text{za } i = n \end{cases}$$

Pri tome ćemo bridove prikazivati u homogenom prostoru. Sve koordinate poligona prije uporabe proširiti ćemo homogenim parametrom 1, budući da vrhove poligona zadajemo u radnom prostoru.



6.2.1.

Parametarska jednadžba pravca (u radnom prostoru) glasi:

$$T_b = \begin{cases} (T_{i+1} - T_i)\cdot\lambda + T_i & \text{za } 0 < i < n \\ (T_i - T_1)\cdot\lambda + T_i & \text{za } i = n \end{cases}$$

gdje je T_b proizvoljna točka pravca.

Jednadžba pravca u homogenom prostoru može se dobiti kao x-produkt vrhova pravca, kao što je opisano u matematičkom uvodu:

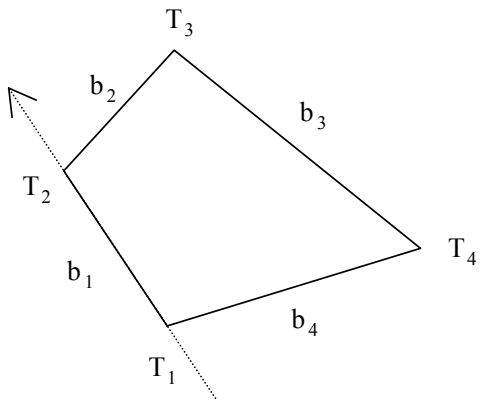
$$T_i \times T_{i+1} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{i,1} & T_{i,2} & 1 \\ T_{i+1,1} & T_{i+1,2} & 1 \end{vmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{i,2} - T_{i+1,2} \\ -(T_{i,1} - T_{i+1,1}) \\ T_{i,1}T_{i+1,2} - T_{i,2}T_{i+1,1} \end{bmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot b_i \quad \text{za } 0 < i < n$$

$$T_i \times T_1 = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ T_{i,1} & T_{i,2} & 1 \\ T_{1,1} & T_{1,2} & 1 \end{vmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot \begin{bmatrix} T_{i,2} - T_{1,2} \\ -(T_{i,1} - T_{1,1}) \\ T_{i,1}T_{1,2} - T_{i,2}T_{1,1} \end{bmatrix} = \begin{bmatrix} \vec{i} & \vec{j} & \vec{k} \end{bmatrix} \cdot b_i \quad \text{za } i = n$$

Oznaka $T_{i,k}$ predstavlja k-tu komponentu točke T_i. Kako smo se ogradili na 2D prostor, umjesto $T_{i,1}$ mogli smo pisati $T_{i,x}$ a umjesto $T_{i,2}$ mogli smo pisati $T_{i,y}$.

6.3 ORIJENTACIJA VRHOVA POLIGONA

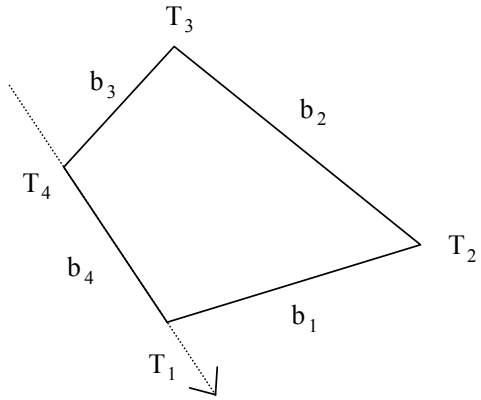
Poligon može imati vrhove zadane tako da se njihovim obilaskom gibamo u smjeru kazaljke na satu, ili u smjeru suprotnom od kazaljke na satu. Orijentacija vrhova biti će nam bitna u daljnjim razmatranjima, pa je nužno objasniti kako se ista može ispitati. U poglavlju 6.2 izračunali smo jednadžbe bridova. Na temelju tih jednadžbi može se ustanoviti u kojem su odnosu točka i zadani brid, odnosno pravac na kojem brid leži (iznad pravca, na pravcu, ispod pravca). Mudrim odabirom ispitne točke jednostavno ćemo doći do spoznaje o orijentaciji vrhova. Pogledajmo sliku 6.3.1.



b_2 je ispod b_1
(uz označeni smjer kretanja nalazi se desno)

6.3.1.

Brid b_1 određen je vrhovima T_1 i T_2. U kakvom je odnosu taj brid s točkom T_3? Točka T_3 nalazi ispod brida! Isto vrijedi i za brid b_2 određen vrhovima T_2 i T_3 i točku T_4. Točka T_4 je ispod brida b_2. Obiđemo li tako sve bridove u krug ispitujući odnos tog brida i prvog slijedećeg vrha poligona, rezultat je isti! Pogledamo li kako su zadani vrhovi našeg poligona, vidimo da su zadani u smjeru kazaljke na satu. Ovo otkriće daje naslutiti kriterij koji bi se mogao koristiti, no prije nego što proglasimo kriterij ispravnim, pogledajmo i drugi slučaj. Na slici 6.3.2. prikazan je poligon sa vrhovima zadanim u smjeru suprotnom od smjera kazaljke na satu.



b_2 je iznad b_1
(uz označeni smjer kretanja nalazi se lijevo)

6.3.2.

Pogledamo sada odnos brida b_1 određenog vrhovima T_1 i T_2 i točke T_3. Točka T_3 nalazi se iznad brida b_1. Slično se opet može vidjeti da ovo vrijedi za svaki brid i prvi slijedeći vrh poligona. Dakle, kriterij je ispravan! Evo načina kako provjeriti orijentaciju vrhova poligona:

- Vrhovi poligona zadani su u smjeru kazaljke na satu ukoliko vrijedi:
  $$(\forall i)\ \ T_j \cdot b_i \le 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$
- Vrhovi poligona zadani su u smjeru suprotnom od smjera kazaljke na satu ukoliko vrijedi:
  $$(\forall i)\ \ T_j \cdot b_i \ge 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$

Ukoliko ustanovimo da je poligon zadan uz jednu orijentaciju vrhova, a nama treba suprotna orijentacija vrhova, tada se jednostavno može zamijeniti redoslijed vrhova poligona i ponovno preračunati jednadžbe bridova.

Imajući u vidu da radimo s konveksnim poligonom, može se jednostavno pokazati slijedeće:

- ukoliko vrijedi da
  $$(\exists i)\ \ T_j \cdot b_i < 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$
  dakle da postoji bar jedan vrh (prvi iza dva koja određuju brid) takav da se nalazi ispod brida, tada mora vrijediti:
  $$(\forall i)\ \ T_j \cdot b_i \le 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$
  odnosno da to vrijedi za svaki vrh. Ovo proizlazi iz činjenice da je poligon konveksan. Isto tako se može pokazati i da
- ukoliko vrijedi
  $$(\exists i)\ \ T_j \cdot b_i > 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$
  dakle da postoji bar jedan vrh (prvi iza dva koja određuju brid) takav da se nalazi iznad brida, tada mora vrijediti:
  $$(\forall i)\ \ T_j \cdot b_i \ge 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$
  odnosno da to vrijedi za svaki vrh.

KRITERIJ ZA ODREĐIVANJE TIPA POLIGONA

Prethodna dva uvjeta mogu se složiti u kriterij koji određuje tip poligona: konveksan ili konkavan. Možemo reći ovako: poligon je konveksan ukoliko vrijedi:

$$(\forall i)\ \ T_j \cdot b_i \le 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$

ili

$$(\forall i)\ \ T_j \cdot b_i \ge 0 \qquad j = \begin{cases} j = i+2, & \text{za} \quad 0 < i < n-1 \\ j = i-n+2, & \text{za} \quad n-1 \le i \le n \end{cases}$$

Konveksnost zahtijeva da ukoliko za neki vrh T_{i+2} utvrdimo da je ispod (iznad) brida b_i (određenog vrhovima T_i i T_{i+1}) tada i svi vrhovi T_{j+2} moraju biti ispod (iznad) svih bridova b_j

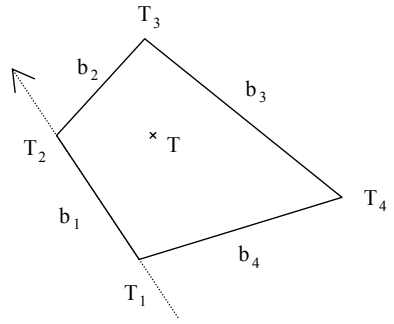
poligona. Ukoliko pak utvrdimo da su neki vrhovi iznad, a neki ispod odgovarajućeg brida, tada je poligon sigurno konkavan!

No ovdje izrečeni kriterij možemo iskoristiti i na drugi način. Ukoliko sigurno znamo da je poligon konveksan, tada se određivanje orijentacije vrhova svodi na jedno jedino ispitivanje! Npr. uzmemo točku T_3 i brid b_1 (određen točkama T_1 i T_2). Ukoliko je T_3 ispod b_1, orijentacija je u smjeru kazaljke na satu; ukoliko je T_3 iznad b_1, orijentacija je u smjeru suprotnom od smjera kazaljke na satu. Jedino što se ovdje nepredviđenoga može dogoditi jest da je poligon zadan "čudno" pa da vrh T_3 leži na bridu b_1. No tada se jednostavno pomaknemo na slijedeći brid i slijedeću točku.

6.4 ODNOS TOČKE I POLIGONA

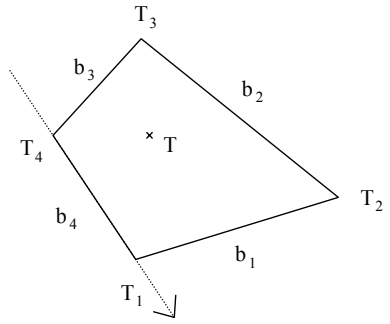
Želimo utvrditi u kakvom je odnosu proizvoljna točka T i poligon (je li točka unutar poligona ili izvan). To možemo napraviti sličnim postupkom kao i kod provjere orijentacije bridova. Potrebno je pogledati odnos svakog brida i zadane točke T. Isto tako je potrebno poznavati orijentaciju bridova poligona. Prema slikama 6.4.1, 6.4.2 i 6.4.3, kriterij glasi:

- Točka T je unutar poligona ukoliko su vrhovi poligona zadani su u smjeru kazaljke na satu, i vrijedi:
  $(\forall i)\ \ T \cdot b_i \le 0 \quad \text{za} \quad 1 \le i \le n$
  odnosno riječima: točka je unutar poligona ako su vrhovi poligona zadani u smjeru kazaljke na satu, i točka je ispod svakog brida.
- Točka T je unutar poligona ukoliko su vrhovi poligona zadani su u smjeru suprotnom od smjera kazaljke na satu, i vrijedi:
  $(\forall i)\ \ T \cdot b_i \ge 0 \quad \text{za} \quad 1 \le i \le n$
  odnosno riječima: točka je unutar poligona ako su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu, i točka je ispod svakog brida.
- I matematički "pametno" za kraj: točka T je izvan poligona ukoliko nije unutra.



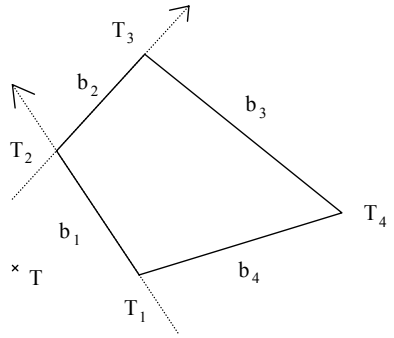
T je ispod b_1
(uz označeni smjer kretanja nalazi se desno)

6.4.1.



T je iznad b_1
(uz označeni smjer kretanja nalazi se lijevo)

6.4.2.



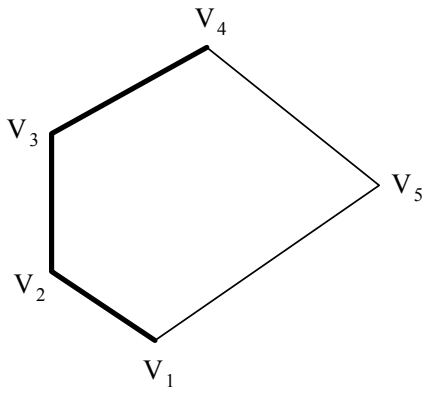
T je iznad b_1 ali je ispod b_2

6.4.3.

6.5 BOJANJE KONVEKSNOG POLIGONA

Postoji nekoliko načina za bojanje poligona. U nastavku će biti opisan postupak sa putujućom vodoravnom zrakom. Ideja je slijedeća: puštati ćemo vodoravne zrake i pratiti gdje se sijeku s poligonom (tj. njegovim bridovima). Dobiti ćemo ili nula ili dva sjecišta. Ukoliko sjecišta postoje, spojiti ćemo ih vodoravnom linijom zadane boje. Da bismo osigurali da za svaku zraku sjecišta uvijek postoje, prije bojanja proći ćemo kroz sve vrhove poligona i zapamtiti najveću i najmanju y-koordinatu. S ovim podatkom vodoravnu zraku ćemo puštati za sve y-e od y_{min} do y_{max} (budući da je zraka zapravo pravac y=konst, traženjem sjecišta sa svim zrakama čiji se y kreće od y_{min} do y_{max} proći ćemo cijeli poligon). Prilikom prolaska kroz točke korisno će biti i zapamtiti najmanju i najveću x koordinatu.
Opisana ideja čini se vrlo jednostavnom, no računski je dosta zahtjevna. Stoga ćemo si uvesti dodatne olakšice i iskoristiti činjenicu da bojimo konveksan poligon. U tom slučaju ideja je slijedeća. Poligon nekako intuitivno možemo podijeliti na lijevi dio (gdje vrhovi rastu prema

većim y vrijednostima) i desni dio (gdje vrhovi padaju prema nižim y vrijednostima) prema slici 6.5.1.



Lijevi bridovi označeni su podebljano,
dok su desni povučeni običnom linijom.

6.5.1.

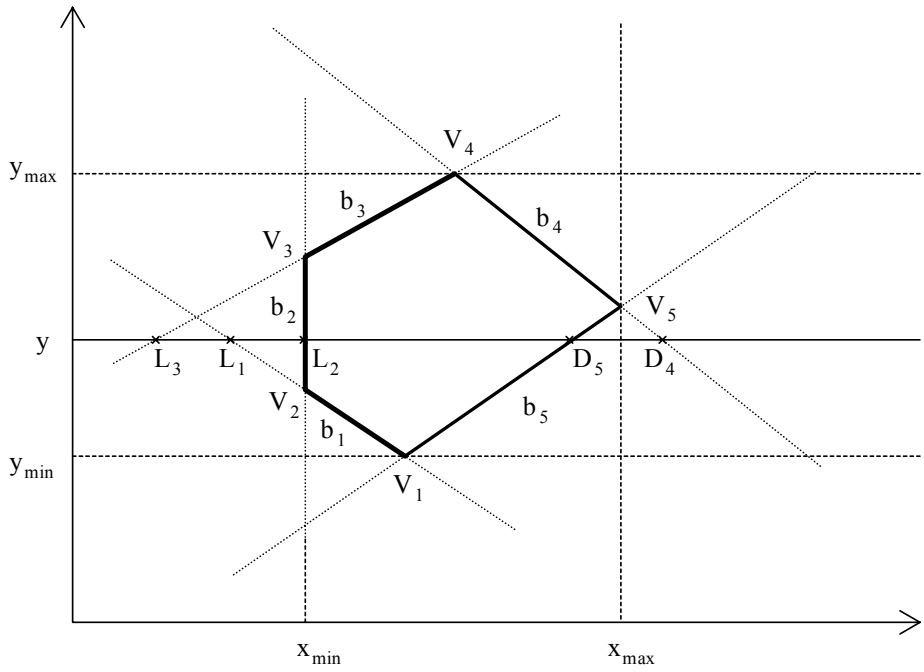
Tražiti ćemo sjecišta sa svim bridovima (slika 6.5.2), i pamtiti njihove x koordinate (y već znamo; to je y koordinata same zrake), i to na slijedeći način:

- za lijeve bridove pamtiti ćemo ono sjecište koje ima najveću x koordinatu, i taj x označiti sa L
- za desne bridove pamtiti ćemo ono sjecište koje ima najmanju x koordinatu, i taj x označiti sa D

Nakon što nađemo sjecište sa svim bridovima, iscrtati ćemo liniju između L i D koordinata na visini y.

Formalno bridove možemo podijeliti na lijeve i desne na slijedeći način:

- brid b_i određen vrhovima T_i i T_{i+1} je lijevi ukoliko je $T_{i,y} < T_{i+1,y}$.
- brid b_i određen vrhovima T_i i T_{i+1} je desni ukoliko je $T_{i,y} > T_{i+1,y}$.



6.5.2.

6.6 STRUKTURA PODATAKA ZA PAMĆENJE KONVEKSNOG POLIGONA

Pri opisu poligona rekli smo da poligon ima onoliko bridova koliko ima vrhova. Stoga za elementarni dio strukture koja pamti podatke o poligonu možemo uzeti pamćenje jednog vrha, i jednog brida. Poligon će se tada sastojati od ***n*** ovakvih struktura. Kako vrhove poligona zadajemo u pixelima, za pamćenje koordinata mogu poslužiti cijeli brojevi. Isto tako obzirom da jednadžbe bridova računamo bez dijeljenja, svi koeficijenti mogu biti cjelobrojni.
Napomena: strukture su pisane za 32-bitni operacijski sustav te oznaka *int* podrazumijeva 32-bitni cijeli broj sa predznakom; za rad pod DOS-om oznake int treba promijeniti u *long*.

Pamćenje točke u 2D sa cjelobrojnim koordinatama:

```
typedef struct {
 int x;
 int y;
} iTocka2D;
```

Pamćenje koeficijenata brida u 2D sa cjelobrojnim koeficijentima:

```
typedef struct {
 int a;
 int b;
 int c;
} iBrid2D;
```

Pamćenje elementarnog dijela poligona:

```
typedef struct {
 iTocka2D Vrh;
 iBrid2D Brid;
 int lijevi;
} iPolyElem;
```

Elementu `lijevi` pridružen je tip `int` u nedostatku prikladnijeg tipa. Tip koji bi najbolje odgovarao bio bi `boolean` koji u C-u ne postoji.

6.7 FUNKCIJE ZA RAD SA POLIGONIMA

Iscrtavanje poligona na zaslonu:

```
void CrtajPoligonKonv(iPolyElem *polel,int n) {
 int i,i0;

 i0 = n-1;
 for( i = 0; i < n; i++ ) {
  crtaj_liniju4(polel[i0].Vrh.x, polel[i0].Vrh.y, polel[i].Vrh.x,
                polel[i].Vrh.y);
  i0 = i;
 }
}
```

Funkcija jednostavno spaja dva po dva vrha u smjeru kako su zadani. Spaja se i završni i početni vrh kako bi se poligon zatvorio. Argumenti funkcije su polje elemenata poligona te broj elemenata polja.

Računanje koeficijenata poligona:

```
void RacunajKoefPoligonKonv(iPolyElem *polel,int n) {
 int i,i0;

 i0 = n-1;
 for( i = 0; i < n; i++ ) {
  polel[i0].Brid.a =polel[i0].Vrh.y-polel[i].Vrh.y;
  polel[i0].Brid.b =-(polel[i0].Vrh.x-polel[i].Vrh.x);
  polel[i0].Brid.c =polel[i0].Vrh.x*polel[i].Vrh.y
                    - polel[i0].Vrh.y*polel[i].Vrh.x;
  polel[i0].lijevi = polel[i0].Vrh.y < polel[i].Vrh.y;
  i0 = i;
 }
}
```

Funkcija računa koeficijente bridova radeći pri tome x-produkt dva po dva vrha. Jednadžba se dobije u homogenom prostoru, a točke se u homogene pretvaraju proširivanjem sa homogenim parametrom iznosa 1. Dodatno se brid klasificira kao lijevi ukoliko je zadan sa vrhovima od kojih je prvi ispod drugoga. Ova klasifikacija biti će ispravna samo ukoliko je poligon zadan vrhovima u smjeru kazaljke na satu. Ukoliko su vrhovi zadani suprotno, tada će postavljena zastavica *lijevi* zapravo označavati da je brid desni.

Bojanje konveksnog poligona:

```
void PopuniPoligonKonv(iPolyElem *polel,int n) {
 int i,i0,y;
```

```
 int xmin, xmax, ymin, ymax;
 double L,D,x;

 /* Traženje minimalnih i maksimalnih koordinata */
 xmin = xmax = polel[0].Vrh.x;
 ymin = ymax = polel[0].Vrh.y;
 for( i = 1; i < n; i++ ) {
  if( xmin > polel[i].Vrh.x ) xmin = polel[i].Vrh.x;
  if( xmax < polel[i].Vrh.x ) xmax = polel[i].Vrh.x;
  if( ymin > polel[i].Vrh.y ) ymin = polel[i].Vrh.y;
  if( ymax < polel[i].Vrh.y ) ymax = polel[i].Vrh.y;
 }

/* Bojanje poligona: za svaki y između ymin i ymax radi... */
 for( y = ymin; y<=ymax; y++ ) {
  /* Pronadi najveće lijevo i najmanje desno sjecište... */
  L = xmin; D = xmax;
  i0=n-1;
  for( i = 0; i < n; i0=i++ ) {
   if(polel[i0].Brid.a==0.) {
    if(polel[i0].Vrh.y == y) {
     if(polel[i0].Vrh.x<polel[i].Vrh.x) {
      L = polel[i0].Vrh.x;
      D = polel[i].Vrh.x;
     } else {
      L = polel[i].Vrh.x;
      D = polel[i0].Vrh.x;
     }
     break;
    }
   } else {
    x = (-polel[i0].Brid.b*y-polel[i0].Brid.c)/(double)polel[i0].Brid.a;
    if(polel[i0].lijevi) {
     if( L < x ) L = x;
    } else {
     if( D > x ) D = x;
    }
   }
  }
  crtaj_liniju4(zaokruzi(L),y,zaokruzi(D),y);
 }
}
```

Funkcija je implementacija opisanog postupka u poglavlju 6.5. Funkcija očekuje da su koeficijenti bridova već izračunati. Pogledajmo trenutak kako se računaju sjecišta sa bridovima. Postoje dva slučaja:

- Brid je vodoravan (i zraka vodoravna!) te postoji opasnost da se brid i zraka sijeku u puno(!!!) točaka. Brid je vodoravan ukoliko mu je koeficijent ***a*** jednak nuli. Tada opet imamo dva slučaja: ili se brid i zraka uopće ne sijeku jer su na različitim y-ima, ili se sijeku u svim točkama jer su na istim y-ima. Ukoliko se ne sijeku, tada jednostavno preskačemo danju analizu i nastavljamo sa obradom slijedećeg brida. Ukoliko se sijeku, tada kao lijevo sjecište uzimamo x koordinatu onog vrha kod kojeg je ona manja, a kao desno sjecište uzimamo x koordinatu onog vrha kod kojeg je ona veća. Zatim prestajemo sa danjom analizom sjecišta (jer kod konveksnog poligona ona ionako ne postoje) i crtamo liniju određenu sa pronađenim lijevim i desnim sjecištem.

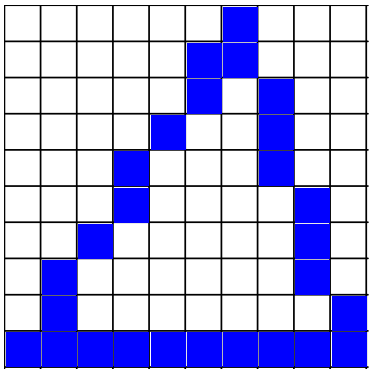
- Brid nije vodoravan te postoji točno jedno sjecište. Tada pronalazimo to sjecište, i ovisno o tome je li brid lijevi ili desni, pamtimo to sjecište kao lijevo ili desno (odnosno pamtimo ga samo ukoliko je lijevo, i veće od trenutno zapamćenog lijevog, odnosno ukoliko je desno i manje od trenutno zapamćenog desnog).

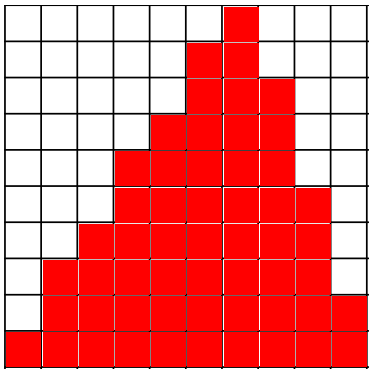
Provjera je li poligon konveksan, te koja je orijentacija vrhova poligona:

```
void ProvjeriPoligonKonv(iPolyElem *polel,int n, int *konv, int *orij) {
 int i,i0,r;
 int iznad, ispod, na;

 ispod = iznad = na = 0;
 i0 = n-2;
 for( i = 0; i < n; i++,i0++ ) {
  if(i0>=n) i0=0;
  r = polel[i0].Brid.a*polel[i].Vrh.x + polel[i0].Brid.b*polel[i].Vrh.y +
      polel[i0].Brid.c;
  if( r == 0 ) na++;
  else if( r > 0 ) iznad++;
  else ispod++;
 }
 *konv = 0; *orij = 0;
 if( ispod == 0 ) {
  *konv = 1;
 } else if( iznad == 0 ) {
  *konv = 1; *orij = 1;
 }
}
```

Funkcija se zasniva na brojanju koliko vrhova leži iznad odgovarajućih bridova, koliko ispod a koliko na njima. Podatak koliko vrhova leži na odgovarajućim bridova redundantan je i trebao bi biti nula. Zaključivanje je slijedeće:

- *ispod = 0*
  Poligon je konveksan jer su tada svi vrhovi iznad odgovarajućih bridova. Orijentacija je u smjeru suprotnom od smjera kazaljke na satu.
- *iznad = 0*
  Poligon je konveksan jer su tada svi vrhovi ispod odgovarajućih bridova. Orijentacija je u smjeru kazaljke na satu.
- *iznad != 0 && ispod != 0*
  Neki su vrhovi iznad a neki ispod odgovarajućih bridova. Poligon je konkavan i informaciju o orijentaciji u varijabli *orij* treba ignorirati.

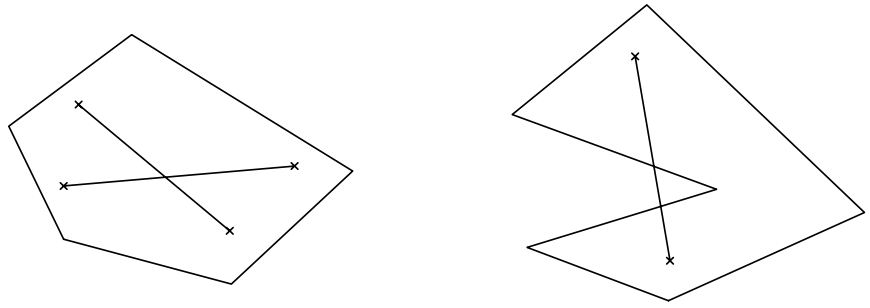

Slika 7.1.a


Slika 7.1.b

7 KONKAVNI POLIGON

7.1 DEFINICIJA KONKAVNOG POLIGONA

Prilikom upoznavanja sa konveksnim poligonima već smo u više navrata spominjali konkavne poligone - kao nešto što komplicira stvari. Da je ovo istina, uvjeriti ćemo se u ovom poglavlju. Konkavni poligon nema nikakvih ograničenja na odnose između svojih bridova i prvog slijedećeg vrha. Konveksni poligon ovdje je postavljao jasne zahtjeve: krenuvši od prvog brida poligona svaki slijedeći brid skreće nas ili ulijevo, ili udesno (ali uvijek isto; npr. svi bridovi nas skreću u lijevo što odmah govori i o orijentaciji vrhova u smjeru kazaljke na satu). Konkavni poligon ovdje nam daje potpunu slobodu. Drugi način definicije konveksnosti zahtijevao je da spojnica bilo koje dvije točke poligona leži isključivo unutar poligona. Konkavni poligon se odriče i ovog zahtjeva. Spojnica bilo koje dvije točke može konačan broj puta izlaziti iz poligona i ponovno ulaziti u njega (pri čemu je broj izlazaka jednak broju ulazaka jer se završna točka spojnice ipak nalazi unutar poligona). Primjer i jasnu razliku između ove dvije vrste poligona daje slika 7.1.1. Pod a) je prikazan konveksni poligon, pod b) konkavni poligon.



a) Konveksni poligon b) Konkavni poligon

7.1.1.

7.2 PROVJERA KONKAVNOSTI POLIGONA I ORIJENTACIJA VRHOVA

Da li je poligon konkavan, može se provjeriti vrlo jednostavno. Naime, dovoljno je provjeriti da poligon nije konveksan prema kriteriju navedenom u poglavlju 7.3. Ukoliko je konveksan, tada nije konkavan; ukoliko nije konveksan, tada jest konkavan. S druge strane, na konkavnost i konveksnost može se gledati kao na realne i prirodne brojeve. Svaki prirodni broj ujedno je i realan. Isto tako svaki konveksni poligon ujedno je i konkavan; konveksnost je samo skup pravila koja ograničavaju konkavnost na ono što zovemo konveksnim likovima (a u ovom slučaju poligonima). No svaki algoritam koji radi za konkavne poligone, raditi će i za konveksne jer su ovi samo specifičan podskup konkavnih.

Comment: Ovo se odnosi na poglavlje o konveksnim poligonima, a ne na ovo! Pazi koji X stavljas!

Orijentaciju vrhova poligona možemo provjeriti na slijedeći način.

- Ukoliko su vrhovi poligona zadani u smjeru kazaljke na satu, tada postoji barem jedan brid takav da su svi vrhovi (a time i bridovi) ispod tog brida, odnosno matematički formulirano:

  $$(\exists i)(\forall j)\ \ b_i \cdot T_j < 0$$

  Prisjetimo li se konveksnih poligona, uočiti ćemo da je tamo uvjet bio gotovo identičan. Razlika je bila samo u činjenici da zbog konveksnosti, ukoliko je postojao barem jedan *i*, tada je to vrijedilo za sve *i*!

- Ukoliko su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu, tada postoji barem jedan brid takav da su svi vrhovi (a time i bridovi) iznad tog brida, odnosno matematički formulirano:

  $$(\exists i)(\forall j)\ \ b_i \cdot T_j > 0$$

  Prisjetimo li se opet konveksnih poligona, uočiti ćemo da je tamo razlika bila samo u činjenici da zbog konveksnosti, ukoliko je postojao barem jedan *i*, tada je to vrijedilo za sve *i*!

7.3 MATEMATIČKI OPIS KONKAVNOG POLIGONA

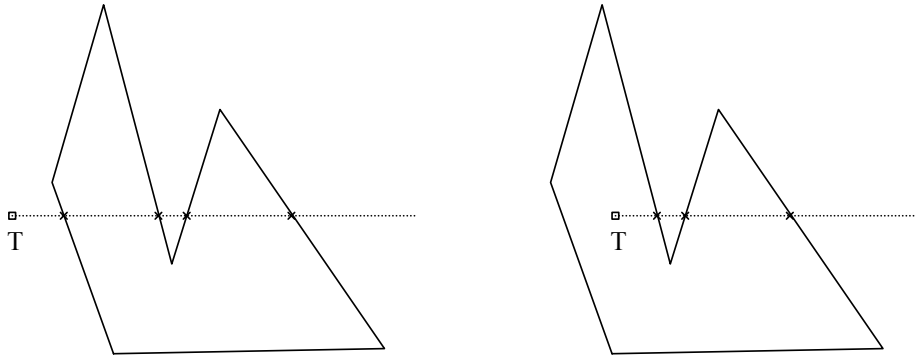
Za konkavne poligone koristiti ćemo identične strukture i jednadžbe kao i za konveksne poligone. I ovdje je svaki brid određen sa dvije točke. Bridova također ima točno toliko koliko i vrhova. Svaki brid određen je relacijom:

$$T_b = \begin{cases} (T_{i+1} - T_i)\cdot\lambda + T_i & \text{za } 0 < i < n \\ (T_i - T_1)\cdot\lambda + T_i & \text{za } i = n \end{cases}$$

gdje je T_b proizvoljna točka pravca na kojem leži i brid, a za $0 \le \lambda \le 1$ upravo proizvoljna točka samog brida.

7.4 ODNOS TOČKE I KONKAVNOG POLIGONA

Da bismo ispitali u leži li proizvoljno zadana točka unutar poligona ili ne leži, kod konveksnih smo poligona dali jednostavan algoritam koji je radio ovu provjeru oslanjajući se na konveksnost i olakšice koje ona donosi. Ovdje ćemo, nažalost, morati posegnuti za drugim kompliciranijim metodama. Jedna od njih zasniva se na ideji da se iz zadane točke pusti zraka (polupravac koji počinje u zadanoj točci) te se izbroji koliko puta zraka sječe poligon (slika 7.4.1).



a) Točka je izvan poligona b) Točka je unutar poligona

7.4.1.

Lako je ustanoviti da ukoliko je taj broj paran, točka je izvan poligona (slika 7.4.1.a), a ukoliko je taj broj neparan, točka je unutar poligona (slika 7.4.1.b). Objašnjenje je jednostavno: krenimo po zraci od početne točke i pratimo sva sjecišta zrake i poligona. Uzmimo npr. da je početna točka zrake unutar poligona. Prvo sjecište na koje naiđemo označavati će izlaz zrake iz poligona. Ukoliko postoji još jedno sjecište, to će označavati da je zraka opet ušla u poligona, no tada će sigurno postojati još jedno sjecište gdje će zraka napustiti poligon, i tako redom. Brojanjem ovih sjecišta ustanoviti ćemo da ih je *neparan broj*! Možemo zaključiti: *točka je unutar poligona ukoliko zraka povučena iz te točke sječe poligon neparan broj puta*. Ponovimo li prethodni postupak ali uz pretpostavku da je zraka izvan poligona, tada će prvo sjecište označavati ulazak u poligon, pa će sigurno postojati još jedno koje će označavati izlazak iz poligona; i opet se sve može ponoviti konačan broj puta. No razlika je očita: sada je sjecišta uvijek paran broj! Dakle, *točka je izvan poligona ukoliko zraka povučena iz te točke sječe poligon paran broj puta*. Iz definicije se može uočiti da se ne postavlja nikakvo ograničenje na smjer zrake; hoćemo li ju povući prema gore, prema dolje, ili kamo već, potpuno je proizvoljno.

Točka je unutar poligona ukoliko zraka povučena iz te točke sječe poligon neparan broj puta.
Točka je izvan poligona ukoliko zraka povučena iz te točke sječe poligon paran broj puta.

Gornja nam definicija daje mogućnost da odredimo odnos točke i poligona, no kako je definicija jedno, a realnost nešto sasvim drugo, treba pogledati nekoliko specifičnih slučajeva koji se mogu pojaviti, te vidjeti što učiniti sa njima. U razmatranjima ćemo vući vodoravne zrake, jer su one najjednostavnije za matematičku obradu.

SPECIFIČNI SLUČAJEVI

1. **Zadana točka se nalazi na bridu.**

   U tom slučaju smatrati ćemo da točka pripada poligonu, odnosno da je unutar poligona, i postupak se prekida.

2. **Zadana točka se poklapa sa vrhom poligona.**

Također ćemo smatrati da je točka unutar poligona, i postupak se prekida.

3. **Sjecište se poklapa sa vrhom poligona, a bridovi iz tog vrha nisu paralelni sa zrakom i ne vrijedi 1. i 2.**

   Mogući slučajevi koji su obuhvaćeni ovim pravilom prikazani su na slici 7.4.2. U slučaju a) i b) zraka ne ulazi u poligon te sjecište treba ignorirati ili brojati kao dvostruko. U slučaju c) i d) zraka ne napušta poligon te ovo sjecište također treba ignorirati (ili brojati kao dvostruko). U slučaju e) i f) zraka iz poligona izlazi van, te ovo sjecište treba brojati (i to kao jednostruko). I konačno u slučaju g) i h) zraka izvana ulazi u poligona, te ovo sjecište također treba brojati (i to kao jednostruko).

4. **Sjecište se poklapa sa vrhom poligona, ali jedan od bridova iz tog vrha je paralelan sa zrakom i ne vrijedi 1. i 2.**

   Slučajevi su prikazani slikom 7.4.3. Pojedini dijelovi slike poklapaju se sa dijelovima slike 7.4.2. Prilikom odlučivanja što učiniti sa sjecištima sada više nije dovoljno promatrati samo dva brida iz tog vrha, već treba pratiti i bridove nakon sjecišta, sve dok su oni paralelni sa zrakom, a nakon toga pogledati prvi slijedeći brid koji nije paralelan sa zrakom. Tada se na temelju početnog brida i pronađenog brida koji nije paralelan sa zrakom donosi odluka.

   U slučajevima a), b), c) i d) sjecišta treba ili brojati kao dvostruka ili ih odbaciti, jer zraka ili je bila izvan poligona te ostaje izvan poligona, ili je bila unutar poligona te ostaje unutra. U slučajevima e) i f) zraka iz poligona izlazi van, a u slučajevima g) i h) zraka izvana ulazi u poligon. U sva četiri slučajeva sjecište treba brojati kao jedno. Jednostavan kriterij za ovaj slučaj može se donijeti na temelju podatka koji su bridovi lijevi a koji su desni (definicija lijevog i desnog brida dana je u 6.5 poglavlju, a ukoliko je brid vodoravan, tada nije niti lijevi, niti desni, već je neutralan). Pravilo je slijedeće: ukoliko se tip brida ne mijenja, sjecište treba brojati; inače ga treba odbaciti. Npr. krećemo sa lijevim bridom, pa slijedi vodoravan brid koji nema utjecaja, i nakon njega opet slijedi lijevi brid; tada sjecište treba brojati.

   Comment: misli se na poglavlje o konveksnim poligonima! Pazi X!

   Može se uočiti da je ovaj slučaj (4) proširenje slučaja 3 pri čemu je vrh zamijenjen vodoravnim pravcem. Zbog toga se slučaj 3 može ignorirati jer će ga slučaj 4 ispravno obraditi.

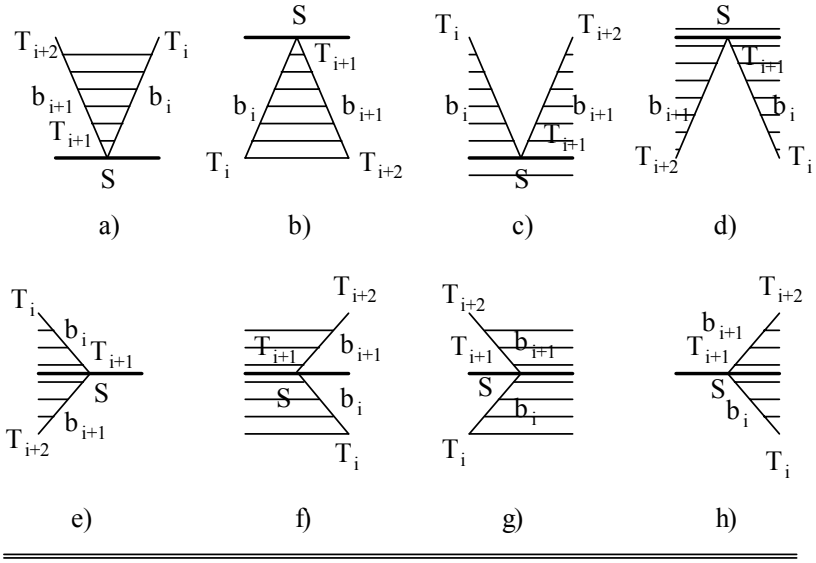
5. **Brid je paralelan sa zrakom.**

   Ukoliko se brid ne poklapa sa zrakom, nema sjecišta i ide se dalje. Ukoliko se brid poklapa sa zrakom, ali zadana točka leži izvan brida, sjecišta se ignoriraju jer će biti obrađena kao dio slučaja 4. Ukoliko se brid poklapa sa zrakom i leži na bridu, točka je unutar poligona i postupak se prekida.

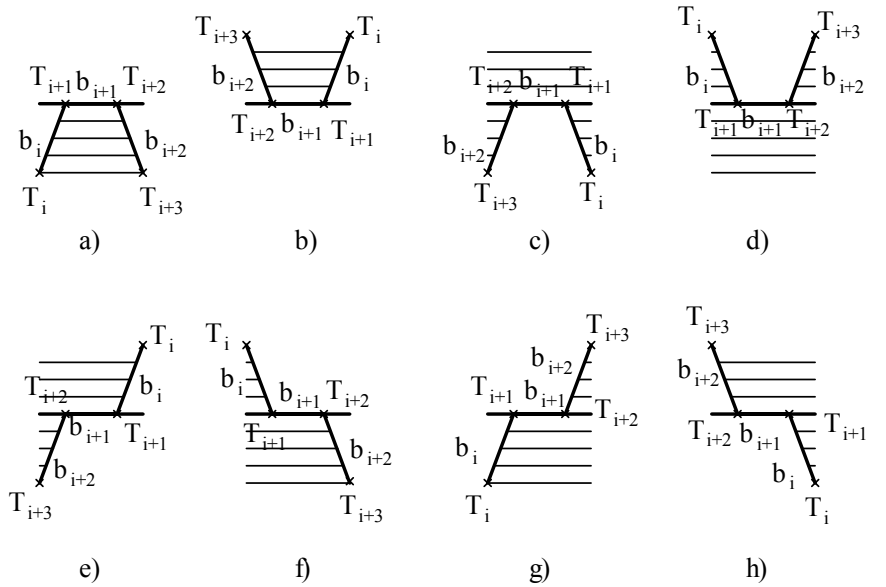
Ovo su više manje svi zanimljivi slučajevi koji se mogli dogoditi. Treba ukazati na još jedan interesantan detalj koji može zbuniti u početku. Kada obilazimo brid za bridom u potrazi za sjecištima, i naletimo na sjecište koje se poklapa sa vrhom poligona, iz tog (kao i iz svakog vrha) izlaze točno dva brida; jednom bridu je to završna točka a drugom bridu je to početna točka. Zanemarivanje ove činjenice dovelo bi nas do dvostrukog brojanja sjecišta i pogrešnog

rezultata. Zato ćemo se u nastavku držati jednostavnog pravila: *ukoliko se sjecište poklopi sa početnom točkom brida, tada sjecište ignoriramo* jer ćemo ga obraditi kod onog brida kojemu će to biti završna točka. Zapravo, tada smo sjecište već obradili jer kojim god smjerom išli prema vrhu poligona (i našem sjecištu), prvo ćemo naići na brid koji počinje u prethodnom vrhu i kojemu je sjecište završna točka, te ćemo ga tu obraditi.

U nastavku ćemo pokazati primjer implementacije ovog algoritma u kod.



Slika 7.4.2.



7.4.3.

7.5 MATEMATIČKI OPIS PROBLEMA

Ne uzimajući u obzir da je poligon zatvoren (radi što jednostavnije indeksacije; poslije se lako sve prilagodi), jednadžbu brida poligona možemo pisati:

$T_b = (T_{i+1} - T_i) \cdot \lambda + T_i, \quad 0 \le \lambda \le 1$

pri čemu je T_b bilo koja točka brida. Za zraku ćemo uzeti vodoravan polupravac koji započinje u zadanoj točci Z, te jednadžbu možemo pisati:

$Z_r = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \mu + Z, \quad \mu \ge 0$

pri čemu je Z_r bilo koja točka zrake.

Brid i zraka sijeku se u nekoj točci, pa vrijedi:

$T_b = Z_r$

odnosno uvrštavanjem izraza za T_b i Z_r:

$(T_{i+1} - T_i) \cdot \lambda + T_i = \begin{pmatrix} 1 & 0 \end{pmatrix} \cdot \mu + Z$

Raspisivanjem po komponentama dobivaju se dvije jednadžbe sa dvije nepoznanice:

$$(T_{i+1,x} - T_{i,x}) \cdot \lambda + T_{i,x} = \mu + Z_x$$
$$(T_{i+1,y} - T_{i,y}) \cdot \lambda + T_{i,y} = Z_y$$

Prodiskutirajmo najprije drugu jednadžbu. Iz nje možemo jednostavno izračunati parametar λ, ukoliko ga jednadžba sadrži. Naime, ukoliko je brid paralelan, tada je $T_{i+1,y} - T_{i,y} = 0$ i jednadžba prelazi u:

$$0 \cdot \lambda + T_{i,y} = Z_y$$

Ukoliko je $T_{i,y} \neq Z_y$, tada jednadžba nema rješenja i brid i zraka se ne sijeku.
Ukoliko je $T_{i,y} = Z_y$, tj. ako se brid i zraka poklapaju, tada je rješenje bilo koji $\lambda \in \Re$, no kako je λ određen jednadžbom brida, kao rješenje se može uzeti bilo koji $0 \leq \lambda \leq 1$. Kako se postupa u ovim slučajevima kada su brid i zraka paralelni, određeno je u 7.4 poglavlju pod "specifičnim slučajevima".

Ukoliko brid i zraka nisu paralelni, tj. $T_{i+1,y} - T_{i,y} \neq 0$, tada se iz druge jednadžbe može izračunati parametar λ kao:

$$\lambda = \frac{Z_y - T_{i,y}}{T_{i+1,y} - T_{i,y}}$$

te se uvrštavanjem u prvu relaciju dobiva i parametar μ:

$$\mu = (T_{i+1,x} - T_{i,x}) \cdot \lambda + T_{i,x} - Z_x$$

Da bi se brid i zraka sjekli, mora biti zadovoljeno: $0 \leq \lambda \leq 1$, i $\mu \geq 0$. Ukoliko nešto od ovoga ne vrijedi, to ukazuje da se pravac na kojem leži brid i pravac na kojem leži zraka sijeku, ali izvan samog brida odnosno same zrake.

7.6 C IMPLEMENTACIJA PROVJERE ODNOSA TOČKE I POLIGONA

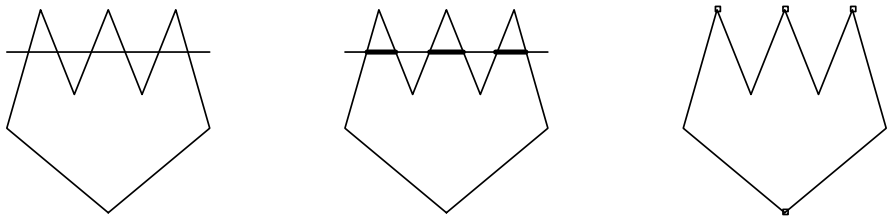
Oslanjajući se na prethodni matematički opis problema algoritam koji radi ispitivanje dade se vrlo jednostavno implementirati. Funkciju koja će obaviti ispitivanje nazvali smo `TockaUKonkPoli`. Funkcija očekuje da su prethodno izračunate jednadžbe bridova i određeni lijevi i desni bridovi (dakle da je pozvana funkcija `RacunajKoefPoligonKonv`). Funkcija vraća **true** ukoliko je točka unutar poligona, odnosno **false** ukoliko nije.

```
bool TockaUKonkPoli(iPolyElem *polel,int n, int Zx, int Zy) {
 int i, j, j0, i0, sjec;
 int i_mi, i_la;
 double mi, la;

 i0 = n-1; sjec = 0;
 for( i = 0; i < n; i++,i0++ ) {
 if( i0 >= n ) i0 = 0;
```

```
  if(polel[i].Vrh.y==polel[i0].Vrh.y) {
   // Brid je vodoravan, pa ako nije na zraci, odbacimo ga
   if(polel[i].Vrh.y != Zy ) continue;
   i_la = polel[i0].Vrh.x - Zx;
   i_mi = polel[i].Vrh.x - Zx;
   if( i_mi == 0 || i_la == 0 ) return true; //slika kop_1
   if( i_mi>0 && i_la<0 || i_mi<0 && i_la>0 ) return true; //slika kop_1
   // inace odbaci brid jer ga obradujemo drugdje
   continue;
  }
  la = (double)(Zy-polel[i0].Vrh.y)/(polel[i].Vrh.y-polel[i0].Vrh.y);
  mi = la*(polel[i].Vrh.x-polel[i0].Vrh.x)+polel[i0].Vrh.x-Zx;
  if( mi<0 ) continue; // ne pripada zraci
  if( la<=0 || la > 1 ) continue; // ne pripada bridu
  if( mi == 0 ) return true; // bas je trazena tocka
  if( la < 1 ) { sjec++; continue; }
  // Ukoliko je la == 1, trebamo pogledati jos nekoliko stvari
  j0 = i; j=j0+1;
  while(1) {
   if( j>=n ) j=0;
   if( j0>=n ) j0=0;
   if(polel[j0].Vrh.y!=polel[j].Vrh.y) break;
   j++; j0++;
  }
  if(polel[i0].lijevi == polel[j0].lijevi) sjec++;
 }
 return ((sjec%2)==1);
}
```

Varijable *mi* i *la*, te njihove cjelobrojne inačice *i_mi* i *i_la* predstavljaju parametre μ i λ. Varijabla *i0* ima ulogu indeksa *i* iz formula, a varijabla *i* ima ulogu indeksa *i+1* u formulama.

7.7 BOJANJE KONKAVNOG POLIGONA



a) b) c)

Oznaka ▫ označava nepopunjeni pixel (slika c).

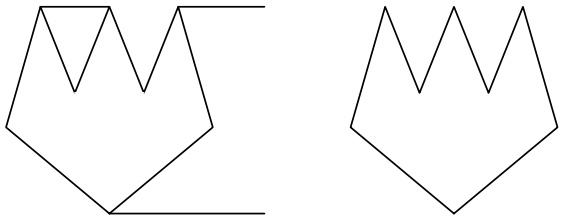
7.7.1.

Problem bojanja konkavnog poligona riješiti ćemo oslanjajući se na već stečena znanja o odnosu točke i poligona. Naime, tamo smo naučili kako pronalaziti sjecišta zrake i poligona, a to će nam u nastavku biti od ključnog značaja. Pogledajmo najprije jedan jednostavan slučaj prikazan na slici 7.7.1. Uz tu sliku, ideja za rješavanje problema bojanja mogla bi glasiti

ovako. Umjesto zrake predstavljene kao polupravac, uzeti ćemo zraku - pravac. Postaviti ćemo zraku na određeni y, i pronaći sva sjecišta zrake i poligona. Ovih će sjecišta naravno biti paran broj! Neka sjecišta ima *2n*. Sortirati ćemo sjecišta po x koordinatama. Zatim ćemo dobivenih *2n* sjecišta promatrati kao *n* segmenata, te ćemo svaki segment obojiti jednostavnim povlačenjem linije između početne i završne točke segmenta, i na visini y. Postupak ćemo ponoviti za svaki y između y_{min} i y_{max}, pri čemu su y_{min} i y_{max} određeni minimalnom i maksimalnom koordinatom vrhova poligona. U nastavku treba pojasniti zašto sjecišta sortiramo i zašto ih grupiramo u segmente po dva sjecišta. Odgovor na prvo pitanje proizlazi iz činjenice da ovisno o tome od kuda krenemo tražiti sjecišta, i ovisno o tome kako je poligon zadan, kretati ćemo se po x osi na nepoznat način. Jednom kada pronađemo sva sjecišta, sortiranjem po x vrijednostima (y-i su svima isti) osiguravamo da prvo sjecište bude ono koje predstavlja ulazak zrake u poligon, slijedeće njezin izlazak, pa slijedeće opet ulazak, pa izlazak, ... Odgovor na drugo pitanje slijedi iz prethodne rečenice: zraku treba bojati samo u onom dijelu gdje prolazi kroz poligon. A kroz poligon prolazi između sjecišta koje označava ulazak, i sjecišta koje označava izlazak. Znači, između prva dva, pa između druga dva (trećeg i četvrtog), pa između treća dva (petog i šestog), itd. Uvesti ćemo si još jednu dodatnu pogodnost: ukoliko sjecište pada u sam šiljak poligona, sjecište ćemo odbaciti ("šiljak" ćemo definirati kao svaki vrh iz kojeg izlaze bridovi suprotnih usmjerenja: lijevi i desni ili desni i lijevi). Slika 7.7.1.a) pokazuje određivanje sjecišta za zraku na visini y, dok 7.7.1.b) pokazuje bojanje segmenata koji se nalaze u poligonu. Slika 7.7.1.c) prikazuje obojani cijeli poligon.

Pogledate li malo pažljivije obojani poligon, uočiti ćete da niti jedan vrh-šiljak koji gleda iz poligona prema van (i to prema gore ili prema dolje) nije obojan, dok vrhovi-šiljci koji gledaju u poligon jesu obojani. Kako to objasniti? Pa to proizlazi iz odbacivanja svih sjecišta koja se poklope sa šiljkom. Tako upravo spomenuti vrhovi-šiljci usmjereni prema gore ili dolje se ignoriraju i ne boja ih se. Ukoliko bismo ove vrhove uzeli u obzir, morali bismo ih uzeti kao dvostruke! U suprotnom, ukoliko ih brojimo ali kao jednostruke, dogoditi će nam se situacija prikazana na slici 7.7.2.a). Svaki šiljak služi kao jedna točka segmenta. Pogledamo li lik na slici 7.7.2., tada prvi šiljak započne segment, drugi šiljak ga završi; treći ga započne, i dalje ga nitko ne završi. No zbog toga bojimo segment koji se uopće ne nalazi u poligonu! Pogledamo li šiljke koji gledaju u tijelo, tada između prvog i drugog šiljka poligon uopće nije obojan. Razlog je opet jednostavan: vanjski brid poligona započeo je segment koji je prvi šiljak završio. Sve do drugog šiljka ništa se ne boja, i tek drugi šiljak započne segment koji se završi izlaskom iz poligona i to se oboji u redu.

Ovdje opisani problem može se ispraviti vrlo jednostavno: svaki šiljak treba uzeti u obzir, i to kao dvostruko sjecište! U tom slučaju, vanjski šiljci prema gore ili dolje biti će proglašeni segmentom duljine 1 pixel; posljedica je bojanje samog vrha. Šiljci prema unutrašnjosti poligona također neće kvariti bojanje: svaki će šiljak zbog svoje dvostrukosti završiti trenutni segment, i odmah iz iste točke započeti novi! Tako ćemo izbjeći sve probleme sa slike 7.7.2.a) i dobiti ispravno obojani poligon (slika 7.7.2.b).



a) b)

7.7.2.

Jednostavan algoritam koji može poslužiti za demonstraciju ovoga naveden je u nastavku.

7.8 C IMPLEMENTACIJA BOJANJA KONKAVNOG POLIGONA (1)

Funkcija koja implementira ideju danu u poglavlju 7.7 prikazana je u nastavku. Funkcija očekuje da su bridovi poligona već izračunati.

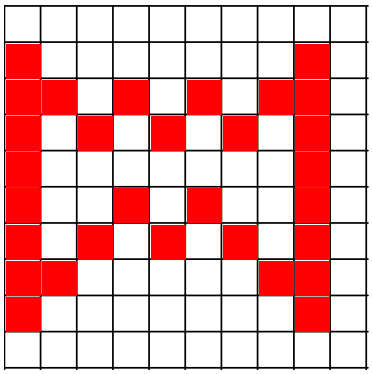
```
void ObojiKonkPoli1(iPolyElem *polel,int n) {
 int i, j, i0, sjec, x, y;
 double mi, la;
 int ymin, ymax;
 int *sjecista;
 int s;

 sjecista = (int*)malloc(sizeof(int)*n);

 /* Trazenje minimalnih i maksimalnih koordinata */
 ymin = ymax = polel[0].Vrh.y;
 for( i = 1; i < n; i++ ) {
  if( ymin > polel[i].Vrh.y ) ymin = polel[i].Vrh.y;
  if( ymax < polel[i].Vrh.y ) ymax = polel[i].Vrh.y;
 }

 for( y = ymin; y<=ymax; y++ ) {

  i0 = n-1; sjec = 0;
  for( i = 0; i < n; i++,i0++ ) {
   if( i0 >= n ) i0 = 0;
   if(polel[i].Vrh.y==polel[i0].Vrh.y) {
    // Brid je vodoravan
    continue;
   }
   la = (double)(y-polel[i0].Vrh.y)/(polel[i].Vrh.y-polel[i0].Vrh.y);
   mi = la*(polel[i].Vrh.x-polel[i0].Vrh.x)+polel[i0].Vrh.x;
   if( la<=0 || la > 1 ) continue; // ne pripada bridu
   if( la < 1 ) {
    sjecista[sjec++]=zaokruzi(mi);
```

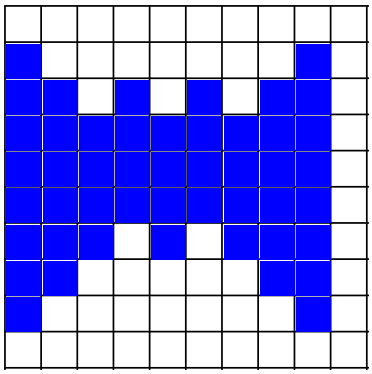
```
     continue;
    }
    // Ukoliko je la == 1, trebamo pogledati je li vrh šiljak
    if(polel[i0].lijevi == polel[i].lijevi) {
     // nije siljak, broji jednostruko
     sjecista[sjec++] = polel[i].Vrh.x;
    } else {
     // je siljak, broji dvostruko
     sjecista[sjec++] = polel[i].Vrh.x;
     sjecista[sjec++] = polel[i].Vrh.x;
    }
   }
   // Sada su poznata sva sjecista;
   // 1) treba ih sortirati
   for( i = 0; i < sjec; i++ ) {
    i0 = i;
    for( j = i+1; j < sjec; j++ ) {
     if(sjecista[j]<sjecista[i0]) i0=j;
    }
    if( i0 != i ) {
     s = sjecista[i0];
     sjecista[i0]=sjecista[i];
     sjecista[i] = s;
    }
   }
   // 2) i vodoravnim linijama pospajati segmente
   for( i = 0; i < sjec; i=i+2 ) {
    for(x=sjecista[i]; x<=sjecista[i+1]; x++ ) {
     osvijetli_pixel(x,y);
    }
   }
  }
  free(sjecista);
}
```

Na slici 7.8.1.a) prikazan je neobojani poligon, a na slici 7.8.1.b) prikazan je taj isti poligon obojan ovom funkcijom. Funkcija je sastavljena od već opisanih osnovnih dijelova:

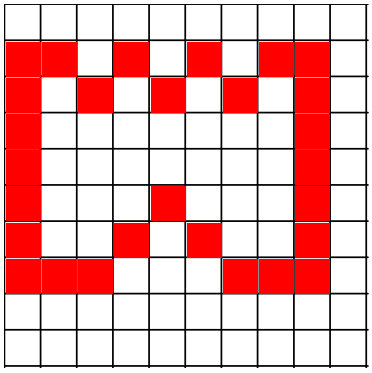
1. traženje minimalne i maksimalne y koordinate
2. presijecanje poligona sa svim zrakama u pronađenom y intervalu
3. traženje sjecišta za svaku zraku, njihovo sortiranje i bojanje segmenata vodoravnim linijama na trenutnom y-u.

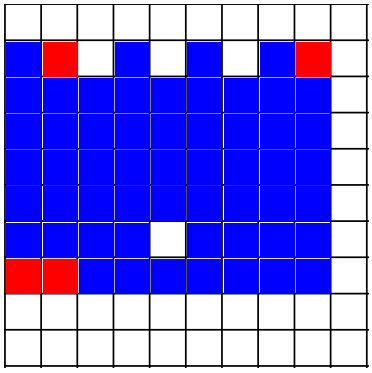
Za potrebe pamćenja sjecišta dinamički se alocira spremnik jer unaprijed nije moguće znati kakav se poligon boja. Što se tiče broja elemenata koje spremnik mora pamtiti, lako je pokazati da poligon i zraka ne mogu imati veći broj sjecišta nego što poligon ima vrhova, pa čak i uz brojanje nekih sjecišta kao dvostruka (šiljci u našem primjeru). Zato se alocira spremnik koji može prihvatiti onoliko sjecišta koliko poligon ima vrhova, čime se izbjegava potreba stalnog dinamičkog realociranja spremnika svakim pronađenim sjecištem.


Slika 7.8.1.a


Slika 7.8.1.b

Slika 7.8.2 također nam dočarava rezultate bojanja jednog poligona. Pod a) je iscrtan neobojani poligon, pod b) njegova obojana verzija... Zapravo, moram se ispričati ali ono pod b) sigurno nije njegova obojana verzija, već nešto... U čemu je problem? Poligon sa slike 7.8.2 donio je našoj funkciji nešto novoga - vodoravne bridove! I to je mjesto gdje se naša funkcija - raspada! Zašto je to tako i što sve vodoravni bridovi donose sa sobom, objasniti ćemo u slijedećem poglavlju.


Slika 7.8.2.a


Slika 7.8.2.b

7.9 BOJANJE KONKAVNOG POLIGONA PO DRUGI PUTA

U prethodnim poglavljima dali smo ideju za bojanje konkavnog poligona, i proanalizirali njezinu realizaciju. Vidjeli smo da algoritam radi, tako dugo dok nemamo u igri vodoravne bridove. Ukoliko ste već zaboravili zašto su nam problematični baš vodoravni bridovi, evo malog podsjetnika: već pri analizi odnosa točke i poligona imali smo problema sa istima! Razlog tome bila je paralelnost zrake i brida, jer smo tada imali ili niti jedno, ili beskonačno mnogo sjecišta! Taj se je problem prenio i ovdje. Da smo koristiti zraku pod kutom od 45°, tada nam vodoravni bridovi ne bi bili problem - problem bi bili bridovi pod kutom od 45°. Kakvu god zraku odabrali, ovo ne možemo izbjeći. Zato smo uzeli najjednostavniju moguću zraku - i završili sa problematičnim vodoravnim bridovima.

Pogledajmo sada sve slučajeve presijecanja zrake i brida.

1. **Zraka i brid su paralelni (tj. brid je vodoravan).**
   Brid se odbacuje jer će biti analiziran u jednom od slijedećih koraka.
2. **Zraka i brid se ne sijeku.**
   Brid se odbacuje.
3. **Zraka i brid se sijeku po unutrašnjosti brida, i brid nije vodoravan.**
   Sjecište se broji kao jednostruko.
4. **Zraka i brid se sijeku u početnoj točki brida, i brid nije vodoravan.**
   Sjecište se odbacuje jer će biti obrađeno u jednom od slijedećih koraka.
5. **Zraka i brid se sijeku u završnoj točki brida i brid nije vodoravan.**
   Za analizu ovog slučaja pogledati slijedeću listu.

Definirajmo prije nastavka proširen pojam šiljka. Vodoravne bridove smatrati ćemo neutralnima, a u razmatranje ćemo uzeti ostale bridove. Vrh je šiljak, ukoliko dolazni brid i odlazni (prvi koji nije vodoravan) brid imaju suprotna usmjerenja, tj. lijevi i desni ili desni i lijevi. Primjeri ovako shvaćenih šiljaka prikazani su na slici 7.4.3. a) do d).

Ukoliko se zraka i brid sijeku prema točki 5 iz gornje liste, tada su mogući slijedeći slučajevi:

a) brid nakon vrha-sjecišta nije vodoravan, tj. imamo šiljak u užem smislu

SJECIŠTE JE ŠILJAK	ZRAKA SE PRIJE SJECIŠTA NALAZI	ZRAKA SE POSLIJE SJECIŠTA NALAZI	SLIKA	AKCIJA
DA	IZVAN	IZVAN	7.4.2.a 7.4.2.b	broji sjecište dvostruko
DA	UNUTAR	UNUTAR	7.4.2.c 7.4.2.d	odbaci sjecišta
NE	UNUTAR	IZVAN	7.4.2.e 7.4.2.f	broji sjecište jednostruko
NE	IZVAN	UNUTAR	7.4.2.g 7.4.2.h	broji sjecište jednostruko

b) brid nakon vrha-sjecišta je vodoravan, tj. imamo šiljak u proširenom smislu

SJECIŠTE JE ŠILJAK	ZRAKA SE PRIJE SJECIŠTA NALAZI	ZRAKA SE POSLIJE SJECIŠTA NALAZI	SLIKA	AKCIJA
DA	IZVAN	IZVAN	7.4.3.a 7.4.3.b	pamti početno i završno sjecište
DA	UNUTAR	UNUTAR	7.4.3.c 7.4.3.d	odbaci sjecišta
NE	UNUTAR	IZVAN	7.4.3.e 7.4.3.f	pamti završno sjecište
NE	IZVAN	UNUTAR	7.4.3.g 7.4.3.h	pamti početno sjecište

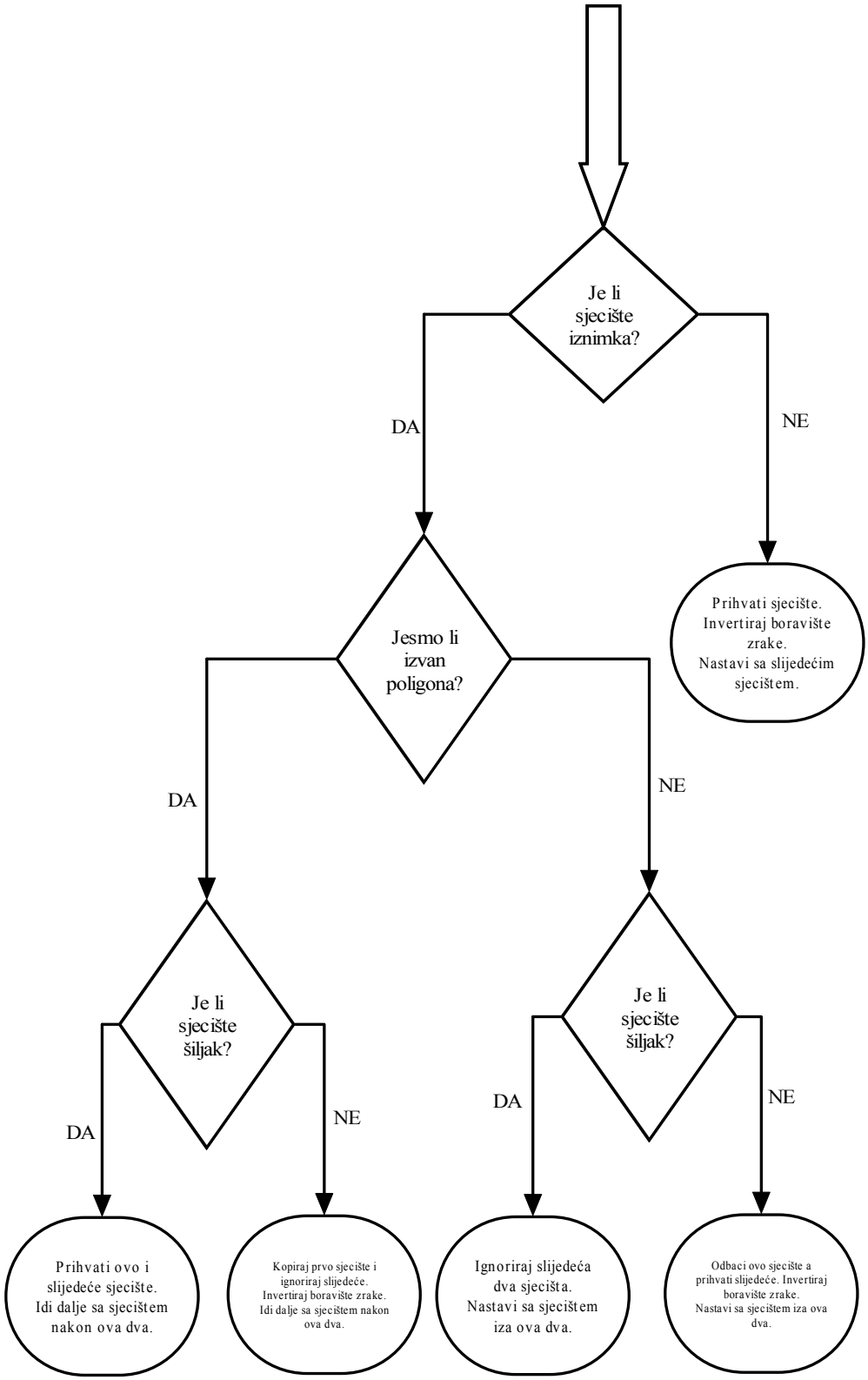
Pogledom na obje tablice lako se uoči razlika: svi su odgovarajući elementi tablica isti, osim što je šiljak iz a) dijela u b) dijelu vodoravnim bridovima razvučen u dvije točke: početnu točku šiljka i završnu točku šiljka.

Analizom prva tri stupca tablice može se doći do slijedećeg zaključka: šiljak ne mijenja "boravište" zrake, tj. ako je zraka prije dolaska na šiljak bila u poligonu, i nakon odlaska ostati će u njemu.

Iz tablica se također vidi da je za određivanje načina postupanja sa bridovima potrebno poznavati gdje se je zraka nalazila prije dolaska u sjecište, i kamo ide nakon toga. Ove podatke za vrijeme traženja sjecišta, nažalost, ne znamo!

Situacija na prvi pogled izgleda beznadno. No samo na prvi pogled. Ključno za odlučivanje što učiniti sa sjecištima je poznavanje "boravišta" zrake u tom trenutku. Prisjetimo li se kako tražimo sjecišta, očito je da u tom trenutku ne raspolažemo tom informacijom. No umjesto toga možemo napraviti nešto drugo: možemo pronaći sva sjecišta, bez da išta ignoriramo. Naime, prilikom traženja sjecišta znamo da li je slijedeći brid vodoravan ili nije; to je najmanji problem. Ukoliko naiđemo na šiljak u užem smislu (dakle niti jedan brid nije vodoravan) pamtiti ćemo sjecište kao dvostruko. Ukoliko naiđemo na vrh iz kojega odlazi vodoravan brid, pamtiti ćemo početnu točku (sjecište) i završnu točku (početak prvog ne-vodoravnog brida). Uz to, za svako sjecište ćemo pamtiti pripada li šiljku ili ne, te pripada li "običnom" sjecištu (točka 3 iz prethodne liste), ili iznimci (sjecišta prema točki 5 iz prethodne liste). Primjer ovakvog određivanja bridova prikazan je na slici 7.9.1. Oznaka **s** uz oznaku sjecišta označava pripadnost šiljku, dok oznaka **!** označava sjecište-iznimku. Sa slike je jasno vidljivo da se sjecišta-iznimke određene na ovaj način uvijek pojavljuju u parovima. Nakon što na ovaj način pronađemo sva sjecišta, u skladu sa algoritmom ćemo ih i sortirati. I sada se postavlja pitanje: "što sada znamo o zraci"?

Gdje je zraka bila prije dolaska u prvo sjecište? Budući da zraka dolazi iz negativne beskonačnosti a poligon počinje u nekoj konačnoj točki, zraka je očito bila izvan poligona! I sa ovom informacijom, i tipizacijom sjecišta provedenih u prethodnom koraku znamo sve! Naime, ako sada naiđe sjecište šiljak, zraka ostaje vani! Ako naiđe sjecište koje nije šiljak, zraka će ući u poligon. Na ovaj način možemo jednostavno pratiti što se događa sa zrakom i tu možemo provesti filtriranje nepotrebnih sjecišta. Dijagram odlučivanja prikazan je u nastavku.


Je li sjecište iznimka?
DA
NE
Prihvati sjecište. Invertiraj boravište zrake. Nastavi sa slijedećim sjecištem.
Jesmo li izvan poligona?
DA
NE
Je li sjecište šiljak?
Je li sjecište šiljak?
DA
NE
DA
NE
Prihvati ovo i slijedeće sjecište. Idi dalje sa sjecištem nakon ova dva.
Kopiraj prvo sjecište i ignoriraj slijedeće. Invertiraj boravište zrake. Idi dalje sa sjecištem nakon ova dva.
Ignoriraj slijedeća dva sjecišta. Nastavi sa sjecištem iza ova dva.
Odbaci ovo sjecište a prihvati slijedeće. Invertiraj boravište zrake. Nastavi sa sjecištem iza ova dva.

Nakon što smo proveli sortiranje sjecišta, prema prethodnom dijagramu potrebno je provjeriti svako sjecište i po potrebi neka odbaciti. Ovo se može izvesti kao dodatni korak prije samog spajanja segmenata. Isto tako, kako postupkom ne umećemo nova sjecišta, cijeli postupak možemo izvesti koristeći samo polje koje smo već alocirali za pamćenje sjecišta.

Evo i nekoliko modifikacija koje treba provesti da bi se omogućio rad ovog algoritma. O sjecištu smo do sada pamtili samo x koordinatu. Zbog potrebe tipizacije sjecišta moramo uvesti dodatnu zastavicu koja će čuvati opis sjecišta. Zbog toga ćemo uvesti jednu jednostavnu strukturu koja će opisivati svako sjecište:

```
typedef struct {
 int x;
 unsigned char Vrsta;
} Sjecista;
```

Pri tome će element *Vrsta* čuvati informaciju o sjecištu:

- **VRSTA_OBICAN** Sjecište nije iznimka
- **VRSTA_SIJECE** Sjecište je iznimka, i nije šiljak.
- **VRSTA_DIRA** Sjecište je iznimka, i jest šiljak.

Ovakvom organizacijom izbjegavamo potrebu za dvije zastavice: 1) je li iznimka i 2) je li šiljak. Naime, prema dijagramu odlučivanja, ukoliko sjecište nije iznimka, druga nam informacija i ne treba. Tek ukoliko jest iznimka, koristimo "drugu" zastavicu. Ovo se u kodu može ispitati vrlo jednostavno; neka je *sjec* varijabla tipa *Sjecista*. Tada vrijedi:

```
if( sjec.Vrsta == VRSTA_OBICAN ) {
  // sjecište nije iznimka
} else {
 if( sjec.Vrsta == VRSTA_SIJECE ) {
   // Sjeciste je iznimka i nije šiljak
 } else {
   // Sjeciste je iznimka i jest šiljak
 }
}
```

Da bi se izbjeglo provođenje ovog koraka, koji ma kako bio jednostavan, ipak unosi dodatno usporenje algoritma, uvesti ćemo dodatnu zastavicu koju ćemo resetirati na nulu za svaku novu zraku, a uvećavati svakim pronalaženjem sjecišta-iznimke. Tako ćemo moći provjeriti treba li izvoditi dodatni korak ili ne.

7.10 C IMPLEMENTACIJA BOJANJA KONKAVNOG POLIGONA (1)

Funkcija koja implementira ideju danu u poglavlju 7.9 prikazana je u nastavku. Funkcija očekuje da su bridovi poligona već izračunati.

```
#define VRSTA_OBICAN 0
#define VRSTA_SIJECE 1
#define VRSTA_DIRA 2

void __fastcall TForm1::ObojiKonkPoli(iPolyElem *polel,int n) {
 int i, j, j0, i0, sjec, x, y;
 double mi, la;
 int ymin, ymax;
 Sjecista *sjecista;
 Sjecista s;
 int izvan;
 int TrebaProvjeru;

 sjecista = (Sjecista*)malloc(sizeof(Sjecista)*n);

 /* Trazenje minimalnih i maksimalnih koordinata */
 ymin = ymax = polel[0].Vrh.y;
 for( i = 1; i < n; i++ ) {
  if( ymin > polel[i].Vrh.y ) ymin = polel[i].Vrh.y;
  if( ymax < polel[i].Vrh.y ) ymax = polel[i].Vrh.y;
 }

 for( y = ymin; y<=ymax; y++ ) {

  TrebaProvjeru = 0;
  i0 = n-1; sjec = 0;
  for( i = 0; i < n; i++,i0++ ) {
   if( i0 >= n ) i0 = 0;
   if(polel[i].Vrh.y==polel[i0].Vrh.y) {
    // Brid je vodoravan
    continue;
   }
   la = (double)(y-polel[i0].Vrh.y)/(polel[i].Vrh.y-polel[i0].Vrh.y);
   mi = la*(polel[i].Vrh.x-polel[i0].Vrh.x)+polel[i0].Vrh.x;
   if( la<=0 || la > 1 ) continue; // ne pripada bridu
   if( la < 1 ) {
    sjecista[sjec].x=zaokruzi(mi);
    sjecista[sjec++].Vrsta=VRSTA_OBICAN;
    continue;
   }
   // Ukoliko je la == 1, trebamo pogledati jos nekoliko stvari
   j0 = i; j=j0+1;
   while(1) {
    if( j>=n ) j=0;
    if( j0>=n ) j0=0;
    if(polel[j0].Vrh.y!=polel[j].Vrh.y) break;
    j++; j0++;
   }
   if(polel[i0].lijevi == polel[j0].lijevi) {
    sjecista[sjec].x = polel[i].Vrh.x;
    if(polel[i].Vrh.x!=polel[j0].Vrh.x) {
     sjecista[sjec++].Vrsta=VRSTA_SIJECE;
     sjecista[sjec].x = polel[j0].Vrh.x;
     sjecista[sjec++].Vrsta=VRSTA_SIJECE;
     TrebaProvjeru++;
    } else {
     sjecista[sjec++].Vrsta=VRSTA_OBICAN;
    }
   } else {
    sjecista[sjec].x = polel[i].Vrh.x;
    sjecista[sjec++].Vrsta=VRSTA_DIRA;
    sjecista[sjec].x = polel[j0].Vrh.x;
    sjecista[sjec++].Vrsta=VRSTA_DIRA;
    TrebaProvjeru++;
```

```
    }
   }
   // Sada su poznata sva sjecista;
   // 1) treba ih sortirati
   for( i = 0; i < sjec; i++ ) {
    i0 = i;
    for( j = i+1; j < sjec; j++ ) {
     if(sjecista[j].x<sjecista[i0].x) i0=j;
    }
    if( i0 != i ) {
     s = sjecista[i0];
     sjecista[i0]=sjecista[i];
     sjecista[i] = s;
    }
   }
   // 2) i treba provesti algoritam za odredivanje koja
   //    sjecista ostaju

   if( TrebaProvjeru != 0 ) {
    izvan = 1;
    i = 0; j=0;
    do {
     if( sjecista[i].Vrsta == VRSTA_OBICAN ) {
      izvan = !izvan;
      sjecista[j++] = sjecista[i++];
      continue;
     }
     if( izvan ) {
      if(sjecista[i].Vrsta == VRSTA_DIRA ) {
       sjecista[j++] = sjecista[i++];
       sjecista[j++] = sjecista[i++];
       continue;
      } else {
       sjecista[j++] = sjecista[i++]; i++;
       izvan = !izvan;
       continue;
      }
     } else {
      if(sjecista[i].Vrsta == VRSTA_DIRA ) {
       i++; i++;
       continue;
      } else {
       i++; sjecista[j++] = sjecista[i++];
       izvan = !izvan;
       continue;
      }
     }
    } while( i < sjec );
   }
   for( i = 0; i < j; i=i+2 ) {
    for(x=sjecista[i].x; x<=sjecista[i+1].x; x++ ) {
     osvijetli_pixel(x,y);
    }
   }
  }
  free(sjecista);
 }
```

Funkcija se od prethodne razlikuje samo po implementaciji dijagrama odlučivanja, kako bi odredila koja su sjecišta nepotrebna.

7.11 KORAK DALJE U PRAVOM SMJERU

Osvrnimo se malo unazad i pogledajmo kako smo tražili sjecišta? Za svaku zraku koristili smo izvedenu formulu i za svako sjecište radili nekoliko oduzimanja, dijeljenje, množenje pa čak i zbrajanje i sve to nad realnim brojevima! I tako za svaki brid, i za svaku zraku. No idemo malo razmisliti... Ako zraka *y* sječe brid b_i, da li će i zraka *y+1* sječi brid b_i, i ako hoće, u kojoj točki? Odgovor na prvo pitanje je: vjerojatno hoće. Odgovor na drugo pitanje idemo izračunati!

Brid bi određen je:

$$T_b = (T_{i+1} - T_i) \cdot \lambda + T_i, \qquad 0 \le \lambda \le 1$$

Zraka na visini *y* određena je relacijom:

$$T_z = (1 \quad 0) \cdot \mu + (0 \quad y), \qquad \mu \in \Re$$

Sjecište su one točke za koje vrijedi $T_b=T_z$, te se dobije da je za te točke x-koordinata jednaka:

$$T_{b,x}(y) = T_{z,x}(y) = \frac{T_{i+1,x} - T_{i,x}}{T_{i+1,y} - T_{i,y}}(y - T_{i,y}) + T_{i,x}$$

Gdje se sijeku taj isti brid b_i i zraka *y+1*?

$$T_{b,x}(y+1) = T_{z,x}(y+1) = \frac{T_{i+1,x} - T_{i,x}}{T_{i+1,y} - T_{i,y}}(y + 1 - T_{i,y}) + T_{i,x}$$

$$= \frac{T_{i+1,x} - T_{i,x}}{T_{i+1,y} - T_{i,y}}(y - T_{i,y}) + T_{i,x} + \frac{T_{i+1,x} - T_{i,x}}{T_{i+1,y} - T_{i,y}} = T_{b,x}(y) + \frac{T_{i+1,x} - T_{i,x}}{T_{i+1,y} - T_{i,y}} = T_{b,x}(y) + \xi$$

Sjecištu sa prethodnom zrakom potrebno je dodati jedan jedini konstantni realni broj koji se za svaki brid može izračunati samo jednom!!! Ovime smo sve one računske operacije sveli na jednostavno zbrajanje dva realna broja. Postupak Vas podsjeća na nešto? Ne? Ime Bresenham ne zvuči poznato?

Baš kao kod Bresenhamovog algoritma za crtanje linije sjecišta zapravo predstavljaju pixele linije koje smo tamo vrlo uspješno i vrlo brzo znali izračunati... Dakle, i ovaj se postupak dade vrlo jednostavno svesti na cjelobrojno računanje.

Krenimo dalje. Kako ćemo znati kada se uopće zraka i brid sijeku, da možemo početi primjenjivati gornji postupak? Evo ideje. Sve bridove podijeliti ćemo u tri kategorije:

1. brid koji će biti aktivan
2. brid koji je aktivan
3. brid koji nije aktivan

Stanje brida ovisiti će o napredovanju zrake. Zraka kreće od visine y_{min}. Prije početka rada algoritma svi su bridovi tipa 1 (brid koji će biti aktivan). Algoritam krene. Onog trena kada zraka stigne do donjeg kraja brida b_i, tip brida se mijenja u tip 2 (brid koji je aktivan), postavlja se početno sjecište zrake i brida na x koordinatu donjeg kraja brida (ovo se obavi u internom spremniku pridruženom svakom bridu), i u ovom koraku se to sjecište ignorira

(analogijom sa prethodnim algoritmima sjecište odgovara onom gdje je $\lambda=0$). Ukoliko je zraka naišla na brid bi tipa 2, tada se računa sjecište (pomoću zapamćenoga u internom spremniku, i pomoću faktora ξ koji se također pamti u tom spremniku) i sjecište se prihvaća. Ukoliko je zraka prošla brid, tip brida je postavlja u tip 3 (neaktivan brid).

Pomoću ove kategorizacije odmah se vidi da prilikom prolaska kroz popis bridova:

- neaktivne bridove preskačemo (tip 3)
- za aktivne bridove računamo sjecišta i provjeravamo treba li brid postati neaktivan
- za bridove koji će biti aktivni provjeravamo samo treba li ih aktivirati

Na ovdje opisani način pronalaženje sjecišta dade se dosta ubrzati. Ukoliko se još i bridovi u popisu bridova sortiraju prema y koordinatama, napredovanjem algoritma može se postići da određen dio popisa uopće ne treba provjeravati jer su u njemu sigurno svi bridovi neaktivni, te se mogu dobiti i daljnja ubrzanja. Kako ovo izgleda na primjeru, može se vidjeti na slici 7.11.1.

7.12 ZAKLJUČAK

U ovom poglavlju bavili smo se općenitim postupcima popunjavanja poligona. Krenuli smo od ideje, i došli do algoritma koji se temelji na *scan-line* algoritmu za popunjavanje poligona. Posljednja prikazana funkcija čak i popunjava poligon bez greške. No prikazani algoritam ima relativno veliku manu - pronalaženje sjecišta je zbog operacija sa brojevima s pomičnim zarezom dosta spor postupak. Danas su razvijeni i drugi algoritmi koji ovaj problem minimiziraju (primjer je dan u poglavlju 7.11).

8. KRIVULJE - OPĆENITO

8.1 UVOD

Pojam krivulje promatrati ćemo u najširem obliku: krivulja je niz točaka (uz dodatne uvjete koji pobliže opisuju specifični tip krivulje). U ovom dijelu teksta pozabaviti ćemo se načinima zadavanja krivulja, klasifikacijama krivulja te poželjnim svojstvima krivulja.

8.2 ZADAVANJE KRIVULJA

Krivulje obično opisujemo analitički, formulom (ili formulama). Tako imamo tri osnovne kategorije zapisa krivulje:

- Eksplicitnom jednadžbom y=f(x). Problemi koji se pri tome javljaju svakom su poznati. Npr. jednadžba pravca u eksplicitnom obliku glasila je $y=ax+b$. No ovaj oblik zapisivanja imao je i ozbiljnih problema. Jednadžbu kružnice oduvijek smo pisali u obliku: $y = \pm\sqrt{R^2 - x^2}$, ne iz razonode, već jednostavno iz činjenice *da se eksplicitnim oblikom ne mogu prikazati funkcije sa višestrukim vrijednostima*! Kod kružnice je to još nekako prolazilo dodavanjem znaka ± ispred korijena i prešutnim prihvaćanjem da tako zapravo radimo sa dvije krivulje a ne sa jednom.
- Implicitnom jednadžbom f(x,y)=0. Ovakav oblik jednadžbe može prikazati višestruke vrijednosti, no ne može dati djelomičan prikaz. Npr. implicitni oblik jednadžbe kružnice je $x^2 + y^2 = R^2$. Ovime su obuhvaćene sve točke koje pripadaju kružnici, no sada riječ sve počinje predstavljati problem. Naime, ponekad želimo prikazati samo npr. četvrt kružnice!
- Parametarskim zapisom. Pri tome se uvede jedan (ili više) parametar, te se sve koordinate opišu kao eksplicitne funkcije parametra (ili parametara). Npr. za kružnicu možemo pisati ovako: parametar je t, koordinate su $x=cos(t)$, $y=sin(t)$, $0 \le t < 2\pi$. Ovaj oblik nudi i mogućnost djelomičnog prikaza objekta. Tako za prikaz samo prve četvrtine kružnice parametar t umjesto u intervalu $0 \le t < 2\pi$ treba birati u intervalu $0 \le t < \pi/2$.

Neovisno o analitičkom prikazu krivulje, krivulje možemo zadavati prema slijedećim kriterijima:

- tako da prolaze kroz zadane točke
- tako da im definiramo vrijednosti prve derivacije u pojedinim točkama (tangente)
- tako da im definiramo vrijednosti viših derivacija

Pri tome se mogu ravnopravno koristiti i kombinacije navedenih kriterija; npr. možemo tražiti da krivulja prolazi kroz n zadanih točaka, te da tangenta u prvoj točci bude pod kutom $\pi/4$ i sl. No prilikom ovakvog zadavanja treba imati u vidu da ponekad nije jednostavno dobiti sve potrebne parametre krivulje koja će ispunjavati ovakve miješane kriterije.

8.3. KLASIFIKACIJA KRIVULJA

Krivulje možemo klasificirati prema različitim kriterijima. Najčešće su klasifikacije slijedeće:

- periodičke ⇔ neperiodičke
- racionalne ⇔ neracionalne
- otvorene ⇔ zatvorene
- interpolacijske ⇔ aproksimacijske

8.4. POŽELJNA SVOJSTVA KRIVULJA

U nastavku će biti nabrojano što sve želimo kod krivulja.

- Mogućnost prikaza višestrukih vrijednosti (znači, za jedan x više mogućih y vrijednosti).
- Nezavisnost od koordinatnog sustava.
- Svojstvo lokalnog nadzora (promjena jedne točke uzrokuje promjenu oblika krivulje samo u okolini te točke).
- Smanjenje varijacije (zbog visokog stupnja polinoma krivulje ponekad se umjesto glatke krivulje pojavljuje nepoželjno istitravanje).
- Red neprekinutosti što veći (više o ovome u nastavku).
- Mogućnost opisa osnovnih geometrijskih oblika poput kružnice, elipse i sl.

8.5. RED NEPREKINUTOSTI

Red neprekinutosti dolazi iz matematičke analize i odnosi se na funkcije. Kako se krivulje zadaju također preko funkcija, treba pogledati na što se odnose redovi neprekinutosti u ovom slučaju.

8.5.1. C KONTINUITETI

- C^0 kontinuitet (čita se "ce nula", ne "ce na nultu"). Zahtjeva se neprekinutost u koordinatama. Ovo je, funkcijski gledano, zahtjev na funkciju da nema diskontinuitete. Npr. funkcija apsolutno(x) je funkcija C^0 kontinuiteta.
- C^1 kontinuitet. Zahtjeva se neprekinutost tangente u svim točkama. Drugim riječima, ne smije biti šiljaka, već krivulja mora biti glatka. Npr. funkcija apsolutno(x) nije funkcija C^1 kontinuiteta, iako ima C^0 kontinuitet. Razlog je šiljak u ishodištu!
- C^2 kontinuitet. Zahtjeva se neprekinutost druge derivacije u svim točkama.
- C^3 kontinuitet. Zahtjeva se neprekinutost treće derivacije u svim točkama. Može se opisati kao neprekinutost u izvijanju.

8.5.2. G KONTINUITETI

G kontinuiteti postavljaju nešto blaže zahtjeve. Da bi vrijedio G kontinuitet, zahtjeva se da derivacije budu proporcionalne (dok C kontinuiteti zahtjevaju da budu iste).

Traži se da vrijedi:

$$f_1'\big|_T = k \cdot f_2'\big|_T$$

9. KRIVULJE ZADANE PARAMETARSKIM OBLIKOM

9.1. UVOD

U prethodnom poglavlju opisani su načini zapisivanja funkcija. U ovom poglavlju posvetiti ćemo se najzanimljivijem obliku – parametarskom. Ovaj oblik specifičan je po tome što uvodi dodatni parametar, zahvaljujući kojemu omogućava opisivanje i pravih funkcija, i funkcija sa višestrukim vrijednostima. Ideja je da se sve koordinate izraze kao funkcija tog novog parametra. Parametar ćemo označavati oznakom *t*. U tom se slučaju ovisnost koordinata o parametru *t* može izraziti funkcijski:

$$x = f(t)$$
$$y = g(t)$$

Funkcije *f* i *g* mogu biti različitih oblika. Neke od oblika upoznati ćemo u slijedećem poglavlju.

Parametarski oblik za svaku vrijednost parametra određuje po jednu točku krivulje. Zamislimo li da parametar *t* predstavlja vrijeme, a krivulja putanju sićušne čestice, tada parametarski oblik za svaki trenutak određuje položaj čestice. Pustimo li vrijeme da teče kontinuirano, tada će se čestica gibati po zadanoj krivulji. Dakako, pri tome se moramo osloboditi fizikalnih zakonitosti koje idu uz ovu sliku; naime, kako će izgledati ta putanja ovisi jedino o funkcijama koje određuju ovisnosti koordinata o parametru. Tako je moguće da se čestica u jednom trenutku giba ulijevo, pa već u slijedećem zakrene za 90° i nastavi gibanje, ili pak prekine gibanje na tom mjestu i nastavi na nekom sasvim drugom.

Mijenjamo li parametar *t* po svim dozvoljenim vrijednostima (za koje su funkcije definirane), dobiti ćemo cijelu krivulju. No mijenjamo li taj parametar unutar manjeg intervala $[t_0,t_1]$, čestica će opisati samo segment krivulje.

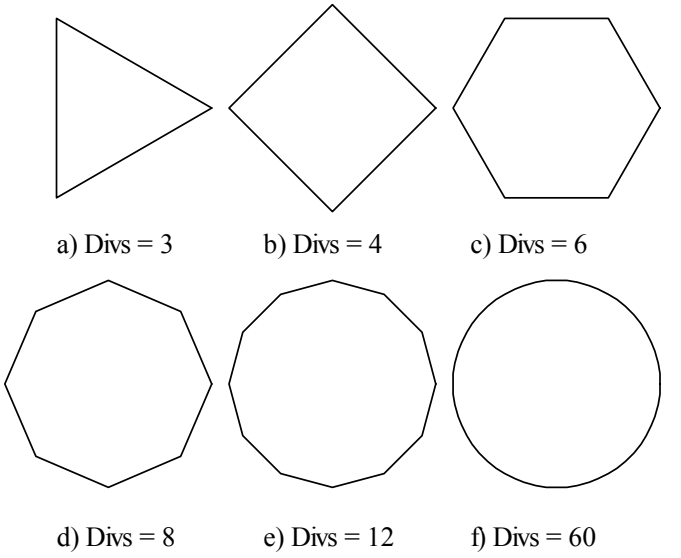
9.2. UPORABA PARAMETARSKOG OBLIKA

Krivulja je po definiciji skup točaka. Beskonačan skup točaka. Parametarski oblik nam ovisno o parametru *t* daje za svaki *t* po jednu točku krivulje. Ukoliko želimo nacrtati cijelu krivulju (dakle pokupiti sve točke), posao nećemo obaviti nikada. Krivulje obično prikazujemo na zaslonu, koji je rasterska jedinica. Zaslon se sastoji od niza elemenata koji mogu biti ili upaljeni, ili ugašeni. Elementi zaslona adresiraju se cjelobrojnom adresom. Proizlazi da zaslon ima konačno mnogo elemenata, i krivulja će na zaslonu biti aproksimirana konačnim brojem elemenata. No kako ćemo odrediti te elemente? Jedan od načina je određivanje konačnog broja točaka krivulje, te njihovo povezivanje pravcima. Za primjer ćemo uzeti prikaz kružnice. Kružnica se u parametarskom obliku može napisati:

$$x = R \cdot \cos(t \cdot 2\pi) + CenterX$$
$$y = R \cdot \sin(t \cdot 2\pi) + CenterY$$

Sve točke kružnice pokupiti ćemo ukoliko mijenjamo parametar t u intervalu [0,1>. Središte kružnice nalazi se u točki (CenterX,CenterY), a radijus kružnice je R.

Prema navedenom receptu, da bismo nacrtali krivulju (kružnicu), idemo odrediti *Divs* točaka kružnice, te susjedne točke spojiti linijama. No postavlja se pitanje koliko točaka treba odrediti? Pokušajmo to utvrditi eksperimentom. Na slikama 9.2.1a do 9.2.1f prikazani su rezultati za različite vrijednosti broja točaka.



9.2.1.

Iz slika je jasno vidljivo kada je prikaz bolji; što više točaka odredimo računski, prikaz na zaslonu je uvjerljiviji. Što nas opet vodi na zaključak da treba uzeti beskonačno finu podjelu. No ovo baš i nije istina! Neka je kružnica radijusa 100 elemenata. I pretpostavimo da smo izračunali upravo onoliko točaka koliko je potrebno da bi svaka izračunata točka imala i susjednu koja je isto izračunata (drugim riječima, linije kojima spajamo te točke dugačke su točno jedan element – tj. svaka spojnica degenerirala je u točku). U tom slučaju finijom podjelom nećemo ništa postići osim da na zaslonu više puta osvijetlimo iste točke. Koliko smo točaka osvijetlili? Pa gruba računica kaže da smo osvijetlili cijeli opseg kružnice, dakle $2\pi R = 628$ elemenata! No u praksi ćemo ipak ići na malo više točaka. Evo obrazloženja. Mijenjamo li parametar t od 0 do 0.25, opisati ćemo prvu četvrtinu kružnice. Pogledajmo kako naše funkcije raspodjeljuju točke u toj četvrtini. Prvih 45° prolazimo sa t između 0 i 0.125. Drugih 45° prolazimo sa t između 0.125 i 0. 25. Oba ova segmenta imaju isti broj izračunatih točaka. Da smo umjesto kružnice crtali elipsu dosta izduženu po X osi, tada bi u prvih 45° segment bio puno duži od segmenta u drugih 45°, a oba bi dobila isti broj točaka! Ovo vodi na zaključak da funkcije već inherentno neravnomjerno raspoređuju točke, te se može dogoditi da na jednom segmentu imamo puno izračunatih točaka a na drugom malo. Na segmentu na kojem ima malo točaka jasnije bi se vidjelo spajanje linijama a to ne želimo! Zato je potrebno napraviti već u startu gušću podjelu da se ovi efekti učine zanemarivim.

9.3. PRIMJER CRTANJA KRUŽNICE

U nastavku ćemo dati implementaciju funkcije koja crta kružnicu na zaslonu, oslanjajući se na parametarski zapis krivulje. Funkciji se predaju koordinate gornjeg lijevog vrha i donjeg desnog vrha kvadrata kojemu se upisuje kružnica. Prototip funkcije biti će:

```
void Circle(T2DRealPoint Point0, T2DRealPoint Point1, int Divs );
```

Struktura `T2DRealPoint` opisuje jednu točku i definirana je kao:

```
typedef struct {
 double x;
 double y;
} T2DRealPoint;
```

Ukoliko će se funkcija koristiti isključivo za prikaz kružnice na zaslonu, komponente točke mogu biti i cjelobrojne; inače je opravdano koristiti realne koordinate (npr. ukoliko su dimenzije platna na koje se crta normirane tako da su širina i visina jednake 1).

Funkcija osim vrhova kvadrata prima još jedan parametar : *Divs* kojim se određuje sa koliko će se segmenata nacrtati krivulje.

```
void Circle( T2DRealPoint Point0, T2DRealPoint Point1, int Divs ) {
 double t, t0, CenterX, CenterY, Rad;
 T2DRealPoint p0, p;
 bool Prvi;
 int n;

 if(Point0.x > Point1.x ) {
  t0 = Point0.x;
  Point0.x = Point1.x;
  Point1.x = t0;
 }
 if(Point0.y > Point1.y ) {
  t0 = Point0.y;
  Point0.y = Point1.y;
  Point1.y = t0;
 }

 CenterX = (Point0.x+Point1.x)/2.;
 CenterY = (Point0.y+Point1.y)/2.;
 Rad = Point1.x - CenterX;

 t0 = 2.*M_PI/Divs; Prvi = true;
 for( n=0, t=0.; n<=Divs; n++,t=t+t0 ) {
  p.x = Rad*cos(t)+CenterX;
  p.y = Rad*sin(t)+CenterY;
  if( Prvi ) {
   p0 = p;
   Prvi = false;
  } else {
   MoveTo(p0.x, p0.y);
   LineTo(p.x, p.y);
   p0 = p;
  }
 }
```

```
}
```

U funkciji se najprije osigura da je gornji lijevi kut doista gornji lijevi, te se ista provjera napravi i sa donjim desnim. Zatim se računa centar kružnice, i njezin radijus. Konačno se u petlji računaju koordinate točaka i međusobno se spajaju. Kontrola broja ponavljanja petlje vrši se pomoću cjelobrojne varijable n. Iako se na prvi pogled čini kao nepotrebno, potrebno je zbog toga što je aritmetika pomičnog zareza netočna i zna dovesti do toga da se crtanje zadnjeg segmenta preskoči. Funkcije *MoveTo* i *LineTo* zadužene su za crtanje pojedinih segmenata. Ukoliko se koriste funkcije koje kao argumente zahtijevaju cijele brojeve, tada koordinate treba zaokružiti (inače će decimalni dio biti odsječen pa će npr. 4.95 biti pretvoreno u 4 što nije dobro).

Ovdje primijenjen postupak za izračunavanje točaka kružnice temelji se direktno na jednadžbi kružnice (parametarskom obliku), te kao takav ne spada u grupu optimalnih. Za iscrtavanje kružnice postoje i brži postupci, a jedan od njih je i Bresenhamov postupak za kružnice!

9.4. PRIMJER CRTANJA ELIPISE

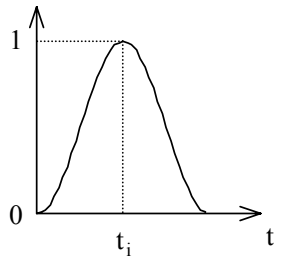
Prethodni primjer može se vrlo modificirati tako da se umjesto kružnice crta elipsa. Elipsa za razliku od kružnice ima dva radijusa; jedan određuje širinu elipse a drugi visinu. Funkcija koju ćemo napisati prihvatiti će koordinate pravokutnika kojemu treba upisati elipsu.

```
void Ellipse(T2DRealPoint Point0, T2DRealPoint Point1, int Divs ) {
 double t, t0, CenterX, CenterY, RadX, RadY;
 T2DRealPoint p0, p;
 bool Prvi;
 int n;

 if(Point0.x > Point1.x ) {
  t0 = Point0.x;
  Point0.x = Point1.x;
  Point1.x = t0;
 }
 if(Point0.y > Point1.y ) {
  t0 = Point0.y;
  Point0.y = Point1.y;
  Point1.y = t0;
 }

 CenterX = (Point0.x+Point1.x)/2.;
 CenterY = (Point0.y+Point1.y)/2.;
 RadX = Point1.x - CenterX;
 RadY = Point1.y - CenterY;

 t0 = 2.*M_PI/Divs; Prvi = true;
 for( n=0, t=0.; n<=Divs; n++,t=t+t0 ) {
  p.x = RadX*cos(t)+CenterX;
  p.y = RadY*sin(t)+CenterY;
  if( Prvi ) {
   p0 = p;
   Prvi = false;
  } else {
   MoveTo(p0.x, p0.y);
   LineTo(p.x, p.y);
   p0 = p;
```

```
    }
  }
}
```

9.5. KONSTRUKCIJA KRIVULJE OBZIROM NA ZADANE TOČKE

Jedan od najčešćih problema koji je vezan uz područje krivulja može se iskazati slijedećim zadatkom: "*zadano je n+1 točka; potrebno je povući krivulju obzirom na te točke*". Što točno znači "*obzirom*", ovisi o potrebama; to može značiti "kroz točke", ali i ne mora. Također se mogu postavljati različiti zahtjevi na oblik krivulje; npr. možemo tražiti da krivulja bude glatka i sl.

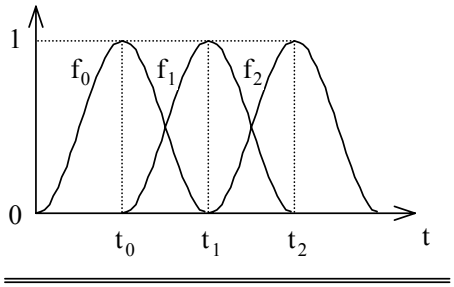


9.5.1.

Jedno od mogućih rješenja je uporaba težinskih funkcija. Ideja je slijedeća. Želimo konstruirati krivulju čiji se početak dobije za vrijednost parametra $t=t_s$ a kraj za $t=t_e$. Pri tome krivulja prolazi kraj točke T_i za vrijednost parametra $t=t_i$. Iskoristiti ćemo težinske funkcije zvonolikog oblika koje imaju maksimum za $t=t_i$, kao na slici 9.5.1. Težinskih funkcija biti će onoliko koliko je zadano točaka. Svaka točka obliku krivulje doprinositi će svojim koordinatama pomnoženim sa težinskom funkcijom. Jednadžbu krivulje tada ćemo zapisati kao sumu svake točke pomnožene težinskom funkcijom. Težinske funkcije označavati ćemo oznakom f_i, pri čemu indeks *i* govori kojoj točci pripada ta težinska funkcija. Tako jednadžbu krivulje možemo zapisati:

$$T_K = \sum_{i=0}^{n} f_i(t) \cdot T_i$$

gdje je T_K točka krivulje a T_i i-ta zadana točka (*kako je zadano n+1 točka, indeks ide od 0 do n*). Uzmimo kao primjer zadane tri točke. Težinske funkcije mogle bi izgledati kao na slici 9.5.2.



9.5.2.

Ukoliko krivulju crtamo od $t=t_0$ do $t=t_2$, što možemo reći o krivulji gledajući ove težinske funkcije?
Za $t=t_0$ je $f_0=1$, $f_1=0$, $f_2=0$, pa je:

$$T_K(t=t_0) = \sum_{i=0}^{3} f_i(t) \cdot T_i = f_0(t_0) \cdot T_0 + f_1(t_0) \cdot T_1 + f_2(t_0) \cdot T_2 = 1 \cdot T_0 + 0 \cdot T_1 + 0 \cdot T_2 = T_0$$

Za $t=t_1$ je $f_0=0$, $f_1=1$, $f_2=0$, pa je:

$$T_K(t=t_1)=\sum_{i=0}^{3} f_i(t)\cdot T_i = f_0(t_1)\cdot T_0 + f_1(t_1)\cdot T_1 + f_2(t_1)\cdot T_2 = 0\cdot T_0 + 1\cdot T_1 + 0\cdot T_2 = T_1$$

Za $t=t_2$ je $f_0=0$, $f_1=0$, $f_2=1$, pa je:

$$T_K(t=t_2)=\sum_{i=0}^{3} f_i(t)\cdot T_i = f_0(t_2)\cdot T_0 + f_1(t_2)\cdot T_1 + f_2(t_2)\cdot T_2 = 0\cdot T_0 + 0\cdot T_1 + 1\cdot T_2 = T_2$$

Krivulja dakle prolazi kroz sve zadane točke, a međutočke se aproksimiraju. Možemo li pogledom na sliku 9.5.2. reći kako krivulja ovisi o promjeni pojedine točke? Na slici je jasno vidljivo da istovremeno na položaj točke djeluju najviše dvije težinske funkcije dok su ostale nula; naime, na intervalu $[t_0,t_1]$ djeluju f_0 i f_1 dok je f_2 jednaka nuli. Na intervalu $[t_1,t_2]$ djeluju f_1 i f_2 dok je f_0 jednaka nuli. To znači da oblik krivulje na intervalu $[t_0,t_1]$ određuju isključivo točke T_0 i T_1 dok točku T_2 možemo pomicati kamo god želimo bez da utječemo na ovaj segment krivulje. Oblik krivulje na intervalu $[t_1,t_2]$ određuju pak točke T_1 i T_2 dok točku T_0 možemo pomicati kamo god želimo bez da utječemo na ovaj segment krivulje. Ovakvo svojstvo krivulje kada promjena položaja jedne točke utječe na promjenu oblika krivulju isključivo u okolini te točke naziva se **svojstvo lokalnog nadzora**. Naime, neke krivulje nemaju ovo svojstvo, pa pomaknemo li jednu točku krivulje, mijenja se cijela krivulja! Svojstvo lokalnog nadzora poželjno je svojstvo.

Pogledajmo još malo sliku 9.5.2. Na slici su sve težinske funkcije jednake. Takve funkcije nazivamo **uniformnima**. Kod težinskih funkcija možemo mijenjati dva parametra: visinu težinske funkcije, te širinu težinske funkcije. Pogledajmo kakvog to utjecaja ima na oblik krivulje.

Ukoliko mijenjamo visinu težinske funkcije, fizikalno to možemo protumačiti kao promjenu privlačenja dotične točke i krivulje. Što je težinska funkcija viša, to će odgovarajuća točka više utjecati na oblik krivulje; što je težinska funkcija niža, to će točka manje mijenjati oblik krivulje. Ovo nam omogućava da osim samih točaka odredimo i njihove "težine" te na temelju toga mijenjamo oblik krivulje.

Promjena širine težinske funkcije može se tumačiti kao promjena dosega privlačne sile između točke i krivulje. Što je težinska funkcija šira, to će odgovarajuća točka imati utjecaj na veći komad krivulje; što je težinska funkcija uža, to će točka utjecati na krivulju na manjem segmentu. Na slici 9.5.2. odabrane su težinske funkcije tako da svaka točka utječe na krivulju na segmentu od prethodnog susjeda točke pa do slijedećeg susjeda točke. Ovime smo postigli svojstvo lokalnog nadzora jer utjecaj jedne točke nije dopirao dalje od njezinih susjeda. Ukoliko bismo širine težinskih funkcija udvostručili, tada bi utjecaj točke T_0 dopirao sve do točke T_2; a kako krivulju crtamo upravo od točke T_0 do točke T_2, tada bi točka T_0 utjecala na cijelu krivulju! A time bismo izgubili svojstvo lokalnog nadzora.

Ukoliko se dozvoljava da se pojedine težinske funkcije razlikuju po širini i po visini, tada za takve težinske funkcije kažemo da su **neuniformne**.

9.6. ZAKLJUČAK

U ovom poglavlju upoznali smo se sa značajem parametarskog oblika, načinima crtanja funkcija zadanih na taj način, te idejom konstrukcije krivulje obzirom na zadane točke. U dijelu 9.5. pokazali smo kako se krivulje mogu konstruirati koristeći težinske funkcije. Pri tome su kao primjer uzete zvonolike težinske funkcije. Primjer nam je poslužio i za upoznavanje sa utjecajima pojedinih parametara samih težinskih funkcija na oblik krivulje. Situacija u praksi, što se tiče ovog dijela je vrlo šarolika. Ne samo da se koriste svakakvi oblici težinskih funkcija, nego se i izvode novi, ovisno o zahtjevima koji se postavljaju na krivulje. Kao primjer krivulja koje nude dosta slobode a koriste zvonolike funkcije navesti ću NURBS (**neu**niformni **r**acionalni **b**-**s**pline). Primjer često korištenih krivulja koje ne koriste zvonolike težinske funkcije su Bezierove krivulje.

U nastavku ove skripte upoznati ćemo se nekim od čestih oblika krivulja.

10. BEZIEROVE KRIVULJE

10.1. UVOD

U računalnoj grafici jedan od čestih zadataka je provlačenje glatke krivulje između zadanog niza točaka (aproksimacija krivulje). Jedno od rješenja problema nude nam Bezierove krivulje. Do krivulja su nezavisno došli 1962. Bezier koji je radio za tvrtku Renault, te 1959. De Casteljau koji je radio za tvrtku Citroen. Da je riječ o istim krivuljama, 1970. godine dokazao je Robert Forest. Za matematički opis Bezierovih krivulja postoje dvije metode: *gibanje vrha sastavljenog otvorenog poligona* koja vodi na Bezierove težinske funkcije, te *parametarsko dijeljenje vektora* koje vodi na Bernsteinove težinske funkcije. Za praktičnu primjenu u računalima Bernsteinove težinske funkcije su daleko pogodnije. Isto tako ćemo razlikovati dva tipa Bezierovih krivulja: *aproksimacijska* i *interpolacijska*. Aproksimacijska Bezierova krivulja prolazi kroz početnu i završnu točku, dok kroz ostale točke ne prolazi. Zato se i kaže da aproksimira zadanu krivulju. Interpolacijska Bezierova krivulja prolazi kroz sve zadane točke; ona dakle interpolira krivulju po segmentima između zadanih točaka, a prolazi kroz zadane točke.

No aproksimacijsku Bezierovu krivulju lagano je odrediti, kako ćemo vidjeti u nastavku. S druge strane, interpolacijsku Bezierovu krivulju dobivamo trikom. Naime, budući da jednostavno provlačimo aproksimacijsku krivulju, a zadane su nam točke kroz koje krivulja mora proći, te točke ćemo iskoristiti tako da izračunamo nove točke između kojih ćemo provući aproksimacijsku krivulju, uz uvjet da ta krivulja prolazi kroz stare zadane točke. Vidimo dakle da osnovno što moramo naučiti je kako računati aproksimacijsku krivulju.

10.2 APROKSIMACIJSKA BEZIEROVA KRIVULJA

10.2.1 OZNAKE

Prilikom rada s Bezierovim krivuljama te njihovog matematičkog tretmana koristiti ćemo slijedeće oznake:

- Točke - vrhovi poligona biti će označeni oznakom $\mathbf{T_i}$, gdje je **i** indeks točke. Bezierova krivulja biti će zadana točkama T_0, T_1, ..., T_n. Pri tome **n** označava stupanj krivulje. Tako će krivulja trećeg stupnja (n=3) biti zadana sa četiri točke: T_0, T_1, T_2 i T_3.
- Oznakom $\vec{r}_i$ označavati će se radij-vektori točaka vrhova poligona. Radij vektor je vektor koji spaja ishodište i zadanu točku T_i. Zbog toga su mu sve komponente jednake kao i kod same točke T_i. Dakle, radij-vektor koji spaja točku (1 2) je (1 2).
- Oznakom $\vec{a}_i$ označavati će se vektori između točaka T_{i-1} i T_i; kako točka T0 nema prethodne, vektor $\vec{a}_0$ biti će jednak radij-vektoru te točke, tj $\vec{a}_0 = \vec{r}_0$. Općenito se može zapisati: $$\vec{a}_i = \begin{cases} T_i - T_{i-1}, & i > 0 \\ T_0, & i = 0 \end{cases}$$
- Težinske funkcije bit će označavane oznakom $\psi_{i,n}$, pri čemu će kod Bezierovih težinskih funkcija umjesto ψ stajati **f**, a kod Bernsteinovih težinskih funkcija **b**. Pri

tome **i** označava indeks funkcije, i taj parametar će se mijenjati između **0** i **n**, dok n označava stupanj krivulje.

10.2.2 BEZIEROVE TEŽINSKE FUNKCIJE

Do ovih se funkcija dolazi metodom gibanja vrha sastavljenog otvorenog poligona, kao što je već spomenuto u uvodu. Sastavljeni otvoreni poligon označava poligon koji se dobije kada se svi vektori izvornog poligona $\vec{a}_i$ pomnože sa određenim težinskim funkcijama i zatim zbroje. Kod nas će, dakako, vektori biti pomnoženi Bezierovim težinskim funkcijama. Slijedi da se može pisati:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t)$$

pri čemu je $\vec{p}(t)$ radij-vektor točke koja pripada krivulji (vrh sastavljenog otvorenog poligona), a ***t*** je parametar kojim određujemo sve točke krivulje. Za t=0 dobiva se početna točka krivulje, a za t=1 dobiva se završna točka krivulje. Sve ostale točke dobivaju se za $t \in \langle 0,1 \rangle$.

Do analitičkih izraza za funkcije dolazi se iz slijedećih zahtjeva:

1. Krivulja za t=0 prolazi kroz prvu zadanu točku:
   $\vec{p}(0) = \vec{a}_0$
   odakle slijedi:
   $f_{0,n}(0) = 1$
   $f_{i,n}(0) = 0, \quad i = 1,2,...n$
2. Krivulja za t=1 prolazi kroz zadnju zadanu točku:
   $\vec{p}(1) = \sum_{i=0}^{n} \vec{a}_i$
   odakle slijedi:
   $f_{i,n}(1) = 1, \quad i = 0,1,...n$
3. U početnoj točki nagib krivulje jednak je nagibu vektora $\vec{a}_1$:
   $f'_{1,n}(0) = 1, \quad f'_{i,n}(0) = 0, \quad i \neq 1$
   Za izvod pogledati okvir .
4. U završnoj točki nagib krivulje jednak je nagibu vektora $\vec{a}_n$:
   $f'_{n,n}(1) = 1, \quad f'_{i,n}(1) = 0, \quad i \neq n$
5. Postavlja se zahtjev na druge derivacije u početnoj točki:
   $f''_{1,n}(0) \neq 0, \quad f''_{2,n}(0) \neq 0, \quad f''_{i,n}(0) = 0, \quad i \neq 1,2$
6. Postavlja se zahtjev na druge derivacije u završnoj točki:
   $f''_{n-1,n}(1) \neq 0, \quad f''_{n,n}(1) \neq 0, \quad f''_{i,n}(1) = 0, \quad i \neq n-1, n$

7. Zahtjeva se simetričnost, odnosno traži se da redoslijed točaka ne utječe na izgled krivulje (zamjena početne i krajnje točke nema utjecaja), što vodi na zahtjev:
   $f_{i,n}(t) = 1 - f_{n-i+1,n}(1-t), \qquad i = 1,2,...,n$

Vektor $\vec{a}_1$ je razlika radij vektora $\vec{a}_1 = \vec{r}_1 - \vec{r}_0$, te svojim komponentama određuje nagib: $tg\delta = \frac{a_{1,y}}{a_{1,x}}$. Krivulja je zadana u parametarskom obliku sumom $\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t)$. Ova jednadžba vrijedi i za svaku komponentu zasebno. Nagib krivulje određen je derivacijom $\frac{dy}{dx}$, što iz dane jednadžbe ne možemo dobiti direktno, već malim trikom. Pomnožimo traženu derivaciju sa prikladno napisanom jedinicom:

$$\frac{dy}{dx} = \frac{dy}{dx}\cdot\frac{dt}{dt} = \frac{\frac{dy}{dt}}{\frac{dx}{dt}} = \frac{\frac{d}{dt}\left(\sum_{i=0}^{n} a_{i,y} f_{i,n}(t)\right)}{\frac{d}{dt}\left(\sum_{i=0}^{n} a_{i,x} f_{i,n}(t)\right)} = \frac{\sum_{i=0}^{n} a_{i,y} f'_{i,n}(t)}{\sum_{i=0}^{n} a_{i,x} f'_{i,n}(t)} =$$

$$= \frac{a_{0,y} f'_{0,n}(t) + a_{1,y} f'_{1,n}(t) + ... + a_{n,y} f'_{n,n}(t)}{a_{0,x} f'_{0,n}(t) + a_{1,x} f'_{1,n}(t) + ... + a_{n,x} f'_{n,n}(t)}$$

Zahtjev je da u početnoj točki (t=0) nagib bude $\frac{a_{1,y}}{a_{1,x}}$, odakle slijedi:

$$\frac{dy}{dx}(t=0) = \frac{a_{0,y} f'_{0,n}(0) + a_{1,y} f'_{1,n}(0) + ... + a_{n,y} f'_{n,n}(0)}{a_{0,x} f'_{0,n}(0) + a_{1,x} f'_{1,n}(0) + ... + a_{n,x} f'_{n,n}(0)} = \frac{a_{1,y}}{a_{1,x}}$$

što je moguće samo ako su derivacije svih težinskih funkcija u toj točci jednake nuli, osim derivacije funkcije $f_{0,n}$ jer ona množi tražene komponente vektora $\vec{a}_1$.

Slično se dobije i za zahtjev 4, gdje se traži jednakost nagiba u završnoj točci krivulje (t=1), i vektora $\vec{a}_n$, ukoliko se u gornju formulu uvrsti t=1, i $tg\delta = \frac{a_{n,y}}{a_{n,x}}$. Tada se izjednačavanjem dobije da derivacije svih težinskih funkcija osim $f_{n,n}$ moraju biti nula.

Okvir 10.2.1.1.

Težinske funkcije koje proizlaze iz ovih zahtjeva nazivaju se Bezierove težinske funkcije i zadane su izrazom:

$$f_{i,n}(t) = \frac{(-t)^i}{(i-1)!}\frac{d^{(i-1)}\Phi_n(t)}{dt^{(i-1)}}, \quad i = 1,2,...,n$$

$$f_{0,n}(t) = 1$$

pri čemu je funkcija $\Phi_n(t)$ zadana jednadžbom:

$$\Phi_n(t) = \frac{1-(1-t)^n}{-t}$$

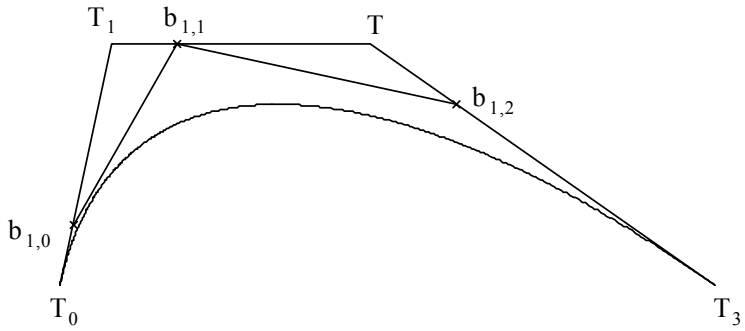
Zbog potrebe za deriviranjem i vrlo složenih izraza vidljivo je da ovakav zapis nije baš prikladan za uporabu u računalima. No ovi se izrazi mogu napisati u, za računala, puno prihvatljivijem obliku: kao rekurzivne funkcije.

$$f_{i,n}(t) = (1-t) f_{i,n-1}(t) + t \cdot f_{i-1,n-1}(t)$$
$$f_{0,0}(t) = 1, \qquad f_{-1,k}(t) = 1, \qquad f_{k+1,k}(t) = 0$$

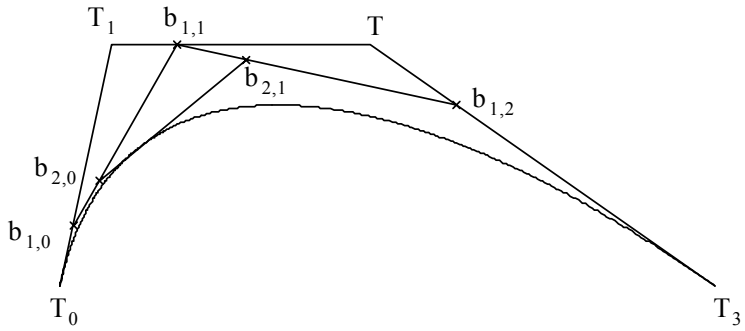
10.2.3. BERNSTEINOVE TEŽINSKE FUNKCIJE

Do funkcija se dolazi na slijedeći način. Potrebno je odrediti točku krivulje za fiksni t. Npr. za t=1/3. Sve stranice izvornog poligona podijele se novom točkom koja odgovara zadanom parametru t (duljina cijele stranice je 1). Novodobivene točke spoje se spojnicama, i zatim se na spojnicama određuju nove točke, zadane parametrom t. Postupak se ponavlja sve dok ne ostane samo jedna spojnica i na njoj određena točka. Ta točka ujedno je i točka krivulje.

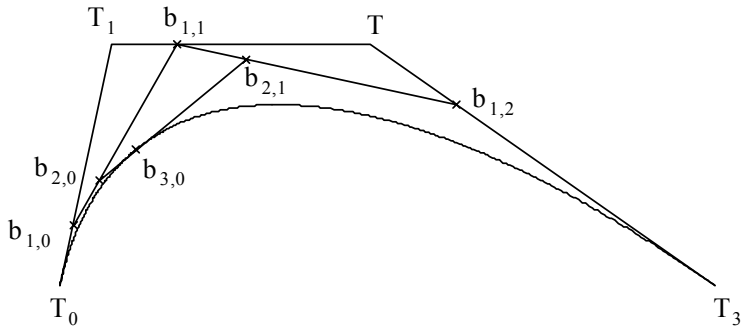
Postupak ćemo najbolje ilustrirati na primjeru. Neka su zadane točke prema slici 10.2.3.1. Na slici 10.2.3.1.a točke su odmah spojene u otvoreni poligon. Odredimo točku za t=1/4. Točka $b_{1,0}$ označava točku na četvrtini između točaka T_1 i T_0. Točka $b_{1,1}$ je četvrtina između točaka T_2 i T_1. Točka $b_{1,2}$ je četvrtina između točaka T_3 i T_2. Točaka $b_{1,0}$, $b_{1,1}$ i $b_{1,2}$ spojene su u dvije nove spojnice. Na slici 10.2.3.1.b te su spojnice podijeljene tako da se točka $b_{2,0}$ nalazi na četvrtini između $b_{1,1}$ i $b_{1,0}$, a točka $b_{2,1}$ nalazi na četvrtini između $b_{1,2}$ i $b_{1,1}$. Ove dvije točke opet su spojene spojnicom. Na slici 10.2.3.1.c ta je spojnice podijeljena tako da se točka $b_{3,0}$ nalazi na četvrtini između $b_{2,1}$ i $b_{2,0}$. Time je postupak gotov jer više nemamo novih spojnica i točka $b_{3,0}$ pripada krivulji.


T1
b1,1
T
b1,2
b1,0
T0
T3

a)


T1
b1,1
T
b2,1
b1,2
b2,0
b1,0
T0
T3

b)


T1
b1,1
T
b2,1
b1,2
b2,0
b3,0
b1,0
T0
T3

c)

10.2.3.1.

Matematički se postupak može opisati na slijedeći način. Krenuli smo od točaka poligona:

$$b_{1,0} = (T_1 - T_0) \cdot t + T_0$$
$$b_{1,1} = (T_2 - T_1) \cdot t + T_1$$
$$b_{1,2} = (T_3 - T_2) \cdot t + T_2$$

Zatim smo spojnice dijelili:

$$b_{2,0} = (b_{1,1} - b_{1,0}) \cdot t + b_{1,0}$$
$$b_{2,1} = (b_{1,2} - b_{1,1}) \cdot t + b_{1,1}$$

I konačno smo i te spojnice podijelili:

$$b_{3,0} = (b_{2,1} - b_{2,0}) \cdot t + b_{2,0}$$

Ako sada $b_{3,0}$ izrazimo samo preko točaka, dobiti ćemo:

$$b_{3,0} = (1-t)^3 \cdot T_0 + 3t(1-t)^2 \cdot T_1 + 3t^2(1-t) \cdot T_2 + t^3 \cdot T_3$$

ili da to napišemo korektno preko radij-vektora točaka:

$$b_{3,0} = (1-t)^3 \cdot \vec{r}_0 + 3t(1-t)^2 \cdot \vec{r}_1 + 3t^2(1-t) \cdot \vec{r}_2 + t^3 \cdot \vec{r}_3$$

Ova točka pripada krivulji, pa konačno možemo pisati:

$$\vec{p}(t) = (1-t)^3 \cdot \vec{r}_0 + 3t(1-t)^2 \cdot \vec{r}_1 + 3t^2(1-t) \cdot \vec{r}_2 + t^3 \cdot \vec{r}_3$$

Dobivena je formula koja opisuje sve točke Bezierove krivulje zadane preko četiri točke, dakle krivulje trećeg reda. Pogleda li se formula malo bolje, vidi se da faktori ispred radij-vektora neobično podsjećaju na binomnu formulu. Da ima istine u tome, govori nam slijedeći zapis ovdje prikazanih težinskih funkcija poznat kao Bernsteinove težinske funkcije:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{r}_i \cdot b_{i,n}(t)$$

pri čemu su težinske funkcije dane relacijom:

$$b_{i,n}(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} = \frac{n!}{i!(n-i)!} \cdot t^i \cdot (1-t)^{n-i}$$

Ovo je najjednostavniji oblik za uporabu u računalima, i mi ćemo ga koristiti u nastavku. Pojava faktorijela također ne komplicira stvari.

10.2.4. VEZA BEZIEROVIH I BERNSTEINOVIH TEŽINSKIH FUNKCIJA

U uvodu smo već rekli da se i pomoću Bezierovih i pomoću Bernsteinovih težinskih funkcija opisuje ista krivulja; to je dokazao Robert Forest. Mi ćemo u nastavku samo dati tu vezu:

$$f_{i,n}(t) = \sum_{j=i}^{n} b_{j,n}(t)$$

10.2.5. CRTANJE APROKSIMACIJSKE KRIVULJE POMOĆU BERNSTEINOVIH POLINOMA

Uvesti ćemo strukturu sVector2D koja će biti zadužena za pamćenje radij-vektora zadane točke. Struktura je oblika:

```
typedef struct {
 double x;
 double y;
} sVector2D;
```

Struktura nam dopušta pamćenje točaka čije koordinate nisu cijeli brojevi (ovo se često pojavljuje pri radu sa grafičkim objektima prije završne faze u kojoj se objekt prikazuje na rasterskoj jedinici).

Definirati ćemo tri varijeble: *Vectors* kao polje koje pamti zadane vektore, *Factors* kao polje pamti binomne koeficijente (n "povrh" i), te *n* kao broj koji pokazuje stupanj krivulje (i polinoma). Deklaracije varijabli su:

```
int n;
int *Factors;
sVector2D *Vectors;
```

Slijedeći korak je zadavanje broja n, te alociranje i popunjavanje polja *Vectors* i *Factors*; npr.

```
n = 3;

Factors = new int[n+1];
Vectors = new sVector2D[n+1];

Vectors[0].x = 1; Vectors[0].y = 1;
Vectors[1].x = 10; Vectors[1].y = 40;
Vectors[2].x = 35; Vectors[2].y = 10;
Vectors[3].x = 45; Vectors[3].y = 40;

CalcFactors();
```

Operator new dolazi iz jezika C++, i za naše potrebe može se poistovjetiti funkciji malloc; alokacije polja Factors i Vectors mogle su se u klasičnom C-u napisati ovako:

```
Factors = (int*)malloc(sizeof(int)*(n+1));
Vectors = (sVector2D*)malloc(sizeof(sVector2D)*(n+1));
```

Ovdje se lagano može uočiti veza između stupnja krivulje i broja točaka: stupanj krivulje za jedan je manji od broja točaka kojim je krivulja zadana. Gornji programski isječak poziva funkciju *CalcFactors()*. Funkcija računa binomne koeficijente i popunjava polje Factors. Evo tijela funkcije:

```
void CalcFactors() {
 int i, a;

 a = 1;
 for( i = 1; i <= n+1; i++ ) {
  Factors[i-1] = a;
  a = a * (n-i+1) / i;
 }
}
```

Kao što se vidi iz tijela funkcije, faktorijele se nigdje ne spominju! Objašnjenje je trivijalno. Neka je zadan stupanj n. Tada su faktori redom:

$$Factors[0] = 1$$

$$Factors[1] = \frac{n}{1} = Factors[0] \cdot \frac{n}{1}$$

$$Factors[2] = \frac{n \cdot (n-1)}{1 \cdot 2} = Factors[1] \cdot \frac{n-1}{2}$$

$$Factors[3] = \frac{n \cdot (n-1) \cdot (n-2)}{1 \cdot 2 \cdot 3} = Factors[2] \cdot \frac{n-2}{3}$$

...

$$Factors[n] = \frac{n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 1}{1 \cdot 2 \cdot 3 \cdot \ldots \cdot n} = Factors[n-1] \cdot \frac{n-(n-1)}{n}$$

odnosno svaki se faktor može dobiti poznavanjem samo prethodnog faktora; također, svi su faktori cijeli brojevi. Ostalo je još jedino nacrtati krivulju! Funkcija koja to radi zove se *DrawBezier*, a tijelo funkcije slijedi:

```
void DrawBezier() {
 int i, j;
 double t, b;
 sVector2D p;

 // Iscrtavanje kontrolnog poligona...
 for( i = 0; i < n; i++ ) {
  Line(Vectors[i].x,Vectors[i].y,Vectors[i+1].x,Vectors[i+1].y, clRed);
 }

 // Iscrtavanje krivulje...
 t = 0;
 for( i = 0; i < 1000; i++, t=t+1./1000. ) {
  p.x = p.y = 0.;
  for( j = 0; j <= n; j++ ) {
```

```
   if( j != 0 ) {
    b = Factors[j]*pow(t,j)*pow(1-t,n-j);
   } else {
    b = Factors[j]*pow(1-t,n-j);
   }
   p.x += Vectors[j].x * b;
   p.y += Vectors[j].y * b;
  }
  PutPixel( p.x, p.y, clBlack );
 }
}
```

Iscrtavanje krivulje radi se za tisuću vrijednosti parametra t u intervalu od 0 do 1 (ovo je određeno petljom po *i*). Radij-vektor svake točke računa se sumom svih radij-vektora zadanih točaka pomnoženih sa odgovarajućim Bernsteinovim koeficijentom (petlja po *j*). Bernsteinov koeficijent pohranjuje se u varijablu *b*, dok je izračunavanje diktirano varijablom *j*. Naime, ukoliko je *j=0*, i nalazimo se u prvom koraku pa je i *t=0*, opći izraz za izračunavanje koeficijenta nam daje nula na nultu i tu program radi krivo. No mi znamo da u tom slučaju rezultat iznosi 1, jer je taj dio izraza proizašao iz *t* na nultu, što je jedan. Sve ostalo bi trebalo biti dovoljno jasno. Za iscrtavanje linije korištena je funkcija koja prima početnu i završnu točku, te boju linije. Funkcija za iscrtavanje pixela prima koordinate i boju. U nedostatku ovih funkcija lagano se napišu funkcije koje ovo mogu oponašati.

10.2.6. SVOJSTVA APROKSIMACIJSKIH BEZIEROVIH KRIVULJA

1. Postoji konveksna ljuska i krivulja je unutar ljuske.
2. Krivulja nema više valova od kontrolnog poligona (ili drugim riječima, ravnina kojom presiječemo krivulju nema više sjecišta od kontrolnog poligona).
3. Krivulja nema svojstvo lokalnog nadzora. Naime, ukoliko pomaknemo samo jednu točku kojom smo zadali krivulju, cijela će krivulja promijeniti oblik. Poželjno bi bilo kada bi se krivulja promijenila samo u okolici pomaknute točke, no ovo kod Bezierovih krivulja ne vrijedi.
4. Broj točaka u direktnoj je vezi sa stupnjem krivulje. Npr. krivulja petog stupnja zadana je pomoću četiri kontrolne točke.
5. Neovisnost o afinim transformacijama (translacije, rotacije, skaliranja). Perspektivne transformacije nisu afine transformacije pa za njih ovo ne vrijedi!
6. Simetričnost. Krivulja ima isti izgled ukoliko kontrolne točke zamijenimo tako da točka koja je bila prva postane zadnja (i obratno).

10.3 INTERPOLACIJSKA BEZIEROVA KRIVULJA

Interpolacijska krivulja prolazi svim zadanim točkama. Način crtanja interpolacijskih krivulja već je opisan u uvodu, no neće škoditi da ponovimo ideju. Zadati ćemo točke (odnosno radij-vektore točaka) kroz koje želimo da krivulja prođe za neki parametar t. Zatim ćemo na temelju tih točaka izračunati kontrolne točke aproksimacijske krivulje, koja će proći kroz naše tražene točke.

Problem se općenito može zapisati ovako. Zadani su radij-vektori:

$$\vec{p}_0(t_0), \vec{p}_1(t_1), ..., \vec{p}_i(t_i), ..., \vec{p}_n(t_n)$$

Kako je krivulja zadana sa n+1 radij-vektorom, krivulja je n-tog stupnja. Za opis krivulje koristiti ćemo Bezierove težinske funkcije. Neka je kontrolni poligon aproksimacijske krivulje zadan vektorima $\vec{a}_0, ..., \vec{a}_n$. Tada je svaka točka aproksimacijske krivulje dana sumom:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t)$$

U tom slučaju postaviti ćemo jednostavan zahtjev: obzirom na zadane radij-vektore zahtijevati ćemo da vrijedi:

$$\vec{p}(t_i) = \sum_{j=0}^{n} \vec{a}_j f_{j,n}(t_i) = \vec{p}_i(t_i)$$

Odnosno tražimo da krivulja za zadane t_i prođe kroz zadane $\vec{p}_i$. Ovo se može zapisati i matrično:

$$\begin{bmatrix} \vec{p}_0 \\ \vdots \\ \vec{p}_n \end{bmatrix} = \begin{bmatrix} f_{0,n}(t_0) & f_{1,n}(t_0) & \cdots & f_{n,n}(t_0) \\ f_{0,n}(t_1) & f_{1,n}(t_1) & \cdots & f_{n,n}(t_1) \\ \vdots & \vdots & \ddots & \vdots \\ f_{0,n}(t_n) & f_{1,n}(t_n) & \cdots & f_{n,n}(t_n) \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vdots \\ \vec{a}_n \end{bmatrix}$$

$$P = F \cdot A \quad \Rightarrow \quad \mathrm{A} = F^{-1} \cdot P$$

Iz ovih jednadžbi potrebno je odrediti matricu **A**. Nakon toga su nam poznati svi vektori $\vec{a}_i$ te se iz njih može direktno crtati krivulja, ili se mogu izračunati radij-vektori točaka kontrolnog poligona aproksimacijske krivulje i zatim crtati krivulju pomoću Bernsteinovih težinskih funkcija.

Vrlo često se u praksi zadaju samo točke kroz koje želimo provući krivulju, ali se pri tome ne specificira za koju vrijednost parametra *t* krivulja mora proći kroz koju točku. Tada se za parametar može odabrati i vrlo jednostavan oblik:

$$t_i = \frac{i}{n}$$

pa se prethodni matrični račun pojednostavljuje:

$$\begin{bmatrix} \vec{p}_0 \\ \vdots \\ \vec{p}_n \end{bmatrix} = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 1 & f_{1,n}(\frac{1}{n}) & \cdots & f_{n,n}(\frac{1}{n}) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & f_{1,n}(\frac{n}{n}) & \cdots & f_{n,n}(\frac{n}{n}) \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vdots \\ \vec{a}_n \end{bmatrix}$$

$$P = F \cdot A \quad \Rightarrow \quad \mathrm{A} = F^{-1} \cdot P$$

Kao primjer možemo uzeti krivulju trećeg stupnja koju želimo provući kroz četiri točke: $\vec{p}_0, \vec{p}_1, \vec{p}_2, \vec{p}_3$. Budući da vrijednosti parametara nisu zadane, uzeti ćemo parametre prema relaciji:

$$t_i = \frac{i}{n}$$

te dobivamo:

$$\vec{p}_0 = \vec{p}\left(\frac{0}{3}\right), \quad \vec{p}_1 = \vec{p}\left(\frac{1}{3}\right), \quad \vec{p}_2 = \vec{p}\left(\frac{2}{3}\right), \quad \vec{p}_3 = \vec{p}\left(\frac{3}{3}\right)$$

Bezierove težinske funkcije za krivulju trećeg reda glase:

$$f_{0,3}(t) = 1$$
$$f_{1,3}(t) = 3 \cdot t - 3 \cdot t^2 + t^3$$
$$f_{2,3}(t) = 3 \cdot t^2 - 2 \cdot t^3$$
$$f_{3,3}(t) = t^3$$

Uvrstimo li ovo u matricu, dobivamo:

$$\begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & \frac{19}{27} & \frac{7}{27} & \frac{1}{27} \\ 1 & \frac{26}{27} & \frac{20}{27} & \frac{8}{27} \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \end{bmatrix}$$

$$P = F \cdot A \quad \Rightarrow \quad \mathrm{A} = F^{-1} \cdot P$$

Rješenje za matricu A glasi:

$$A = \frac{1}{18} \cdot \begin{bmatrix} 18 & 0 & 0 & 0 \\ -33 & 54 & -27 & 6 \\ 21 & -81 & 81 & -21 \\ -6 & 27 & -54 & 33 \end{bmatrix} \cdot P$$

odnosno po komponentama:

$$\begin{bmatrix} \vec{a}_0 \\ \vec{a}_1 \\ \vec{a}_2 \\ \vec{a}_3 \end{bmatrix} = \frac{1}{18} \cdot \begin{bmatrix} 18 & 0 & 0 & 0 \\ -33 & 54 & -27 & 6 \\ 21 & -81 & 81 & -21 \\ -6 & 27 & -54 & 33 \end{bmatrix} \cdot \begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_2 \\ \vec{p}_3 \end{bmatrix}$$

Sada kada smo izračunali tražene vektore, jednadžba aproksimacijske Bezierove krivulje (koja je interpolacijska obzirom na zadane točke) glasi:

$$\vec{p}(t) = \sum_{i=0}^{n} \vec{a}_i f_{i,n}(t) == \sum_{i=0}^{3} \vec{a}_i f_{i,3}(t) = 1 \cdot \vec{a}_0 + \left(3 \cdot t - 3 \cdot t^2 + t^3\right) \cdot \vec{a}_1 + \left(3 \cdot t^2 - 2 \cdot t^3\right) \cdot \vec{a}_2 + t^3 \cdot \vec{a}_3$$

U ovom primjeru bile su zadane n+1 točka. No interpolacijska se krivulja može provlačiti i na temelju drugih podataka. Za izračun aproksimacijske krivulje potrebno nam je n+1 uvjet. Neki od načina zadavanja su slijedeći:

- n+1 poznata točka (kao u primjeru)
- n poznatih točaka i poznata tangenta u nekoj od točaka (obično prva ili zadnja točka)

Moguće je računati krivulju i na temelju poznavanja viših derivacija i sl.

11. PRIKAZ KRIVULJA POMOĆU RAZLOMLJENIH FUNKCIJA

11.1. UVOD

Jedan od načina zapisivanja krivulja jest pomoću razlomljenih funkcija. Pri tome se krivulje zapravo opisuju parametarski uz jedan parametar *t*, u homogenom prostoru, a svaka koordinata točke polinomna funkcija parametra *t*. Kako ćemo krivulje opisivati općenito u 3D prostoru, svaka točka T_K imati će svoje tri koordinate $T_{K,1}$ ili x, $T_{K,2}$ ili y i $T_{K,3}$ ili z u radnom prostoru, odnosno naziv T_{Kh} i četiri koordinate $T_{Kh,1}$, $T_{Kh,2}$, $T_{Kh,3}$ i $T_{Kh,h}$ u homogenom prostoru.

11.2. PRIKAZ POMOĆU KVADRATNIH RAZLOMLJENIH FUNKCIJA

Ove funkcije omogućavaju jednostavan prikaz koničnih krivulja (krivulje koje nastaju kao presjecište stošca i ravnine, npr. kružnice, elipse i sl.). Kako je riječ o kvadratnih funkcijama, funkcijske ovisnosti svih koordinata u homogenom prostoru biti će izražene kvadratnim polinomom:

$$T_{Kh,1} = a_1 \cdot t^2 + b_1 \cdot t + c_1$$
$$T_{Kh,2} = a_2 \cdot t^2 + b_2 \cdot t + c_2$$
$$T_{Kh,3} = a_3 \cdot t^2 + b_3 \cdot t + c_3$$
$$T_{Kh,h} = a \cdot t^2 + b \cdot t + c$$

Povratkom u radni prostor dobiva se:

$$T_{K,1} = x = \frac{a_1 \cdot t^2 + b_1 \cdot t + c_1}{a \cdot t^2 + b \cdot t + c}$$
$$T_{K,2} = y = \frac{a_2 \cdot t^2 + b_2 \cdot t + c_2}{a \cdot t^2 + b \cdot t + c}$$
$$T_{K,3} = z = \frac{a_3 \cdot t^2 + b_3 \cdot t + c_3}{a \cdot t^2 + b \cdot t + c}$$

Razlog za naziv "razlomljene kvadratne" funkcije trebao bi biti jasan iz prethodnih jednadžbi. Zadržimo li se u homogenom prostoru, tada se jednadžbe po koordinatama mogu spojiti u jedan matrični zapis:

$$T_{Kh} = \begin{bmatrix} T_{Kh,1} & T_{Kh,2} & T_{Kh,3} & T_{Kh,h} \end{bmatrix} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K}$$

Matrica **K** naziva se **karakteristična matrica krivulje**.

Za određivanje matrice **K** dovoljno je poznavati tri točke kroz koje krivulja mora proći. Npr. neka prođe kroz točku T_A za $t=t_1=0$, kroz točku T_B za $t=t_2=0.5$, i kroz točku T_C za $t=t_3=1$. Tada vrijedi:

$$T_{Ah} = \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \end{bmatrix} \cdot \mathbf{K}$$
$$T_{Bh} = \begin{bmatrix} T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \end{bmatrix} = \begin{bmatrix} t_2^2 & t_2 & 1 \end{bmatrix} \cdot \mathbf{K}$$
$$T_{Ch} = \begin{bmatrix} T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix} = \begin{bmatrix} t_3^2 & t_3 & 1 \end{bmatrix} \cdot \mathbf{K}$$

Zapišemo li ovo pomoću jedne matrice, dobivamo:

$$\begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \\ T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \\ T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix} \cdot \mathbf{K}$$

Da bismo odredili matricu **K**, cijelu jednadžbu potrebno je pomnožiti sa inverznom matricom koeficijenata i to sa lijeve strane:

$$\mathbf{K} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} t_1^2 & t_1 & 1 \\ t_2^2 & t_2 & 1 \\ t_3^2 & t_3 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah,1} & T_{Ah,2} & T_{Ah,3} & T_{Ah,h} \\ T_{Bh,1} & T_{Bh,2} & T_{Bh,3} & T_{Bh,h} \\ T_{Ch,1} & T_{Ch,2} & T_{Ch,3} & T_{Ch,h} \end{bmatrix}$$

Uvrstimo li zadane t-ove u izraz, dobiva se:

$$\mathbf{K} = \begin{bmatrix} 0^2 & 0 & 1 \\ 0.5^2 & 0.5 & 1 \\ 1^2 & 1 & 1 \end{bmatrix}^{-1} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} T_{Ah} \\ T_{Bh} \\ T_{Ch} \end{bmatrix}$$

Zadamo li sada i točke T_A, T_B i T_C, može se matrica **K** odrediti u potpunosti.

11.3. PARAMETARSKE DERIVACIJE U HOMOGENOM PROSTORU

Svim koordinatama definirali smo funkcijsku ovisnost o parametru *t*. U ovom slučaju ta je funkcijska ovisnost definirana kvadratnim polinomom. No to znači da se te funkcije daju i derivirati! Pa pogledajmo kako bi izgledala prva derivacija po parametru *t*. Funkcijske ovisnost koordinata definirane su relacijama:

$$T_{Kh,1} = a_1 \cdot t^2 + b_1 \cdot t + c_1$$
$$T_{Kh,2} = a_2 \cdot t^2 + b_2 \cdot t + c_2$$
$$T_{Kh,3} = a_3 \cdot t^2 + b_3 \cdot t + c_3$$
$$T_{Kh,h} = a \cdot t^2 + b \cdot t + c$$

Deriviranjem svake relacije po *t* dobiva se:

$$\frac{dT_{Kh,1}}{dt} = a_1 \cdot 2t + b_1$$
$$\frac{dT_{Kh,2}}{dt} = a_2 \cdot 2t + b_2$$
$$\frac{dT_{Kh,3}}{dt} = a_3 \cdot 2t + b_3$$
$$\frac{dT_{Kh,h}}{dt} = a \cdot 2t + b$$

Označimo li derivaciju u točki T_{Kh} po parametru *t* oznakom T'_{Kh} tada možemo derivaciju raspisati po komponentama u matričnom obliku:

$$T'_{Kh} = \left[T'_{Kh,1} \quad T'_{Kh,2} \quad T'_{Kh,3} \quad T'_{Kh,h}\right] = \left[2t \quad 1 \quad 0\right] \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix} = \left[2t \quad 1 \quad 0\right] \cdot \mathbf{K}$$

Možda da još jednom naglasim što predstavlja oznaka T'_{Kh}. To nije derivacija točke, jer točku (očito) ne možemo derivirati (barem se nadam da je svima jasno da se uređeni par od četiri fiksna zadana broja ne može derivirati...). Oznakom T'_{Kh} označili smo derivaciju funkcija kojima su definirane ovisnosti o parametru *t* u nekoj proizvoljnoj točki T_{Kh}. Kako tih funkcija ima četiri, tako je i struktura T'_{Kh} četverokomponentni vektor.

Drugu derivaciju dobivamo deriviranjem prve derivacije; dobiva se:

$$\frac{d^2T_{Kh,1}}{dt^2} = a_1 \cdot 2$$
$$\frac{d^2T_{Kh,2}}{dt^2} = a_2 \cdot 2$$
$$\frac{d^2T_{Kh,3}}{dt^2} = a_3 \cdot 2$$
$$\frac{d^2T_{Kh,h}}{dt^2} = a \cdot 2$$

ili matrično zapisano:

$$T''_{Kh} = \left[T''_{Kh,1} \quad T''_{Kh,2} \quad T''_{Kh,3} \quad T''_{Kh,h}\right] = \left[2 \quad 0 \quad 0\right] \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix} = \left[2 \quad 0 \quad 0\right] \cdot \mathbf{K}$$

Sve više derivacije daju:

$$\left.\begin{aligned}
&\frac{d^n T_{Kh,1}}{dt^n} = 0 \\
&\frac{d^n T_{Kh,2}}{dt^n} = 0 \\
&\frac{d^n T_{Kh,3}}{dt^n} = 0 \\
&\frac{d^n T_{Kh,h}}{dt^n} = 0
\end{aligned}\right\} \quad n > 2$$

odnosno matrično:

$$T_{Kh}^{(n)} = \begin{bmatrix} T_{Kh,1}^{(n)} & T_{Kh,2}^{(n)} & T_{Kh,3}^{(n)} & T_{Kh,h}^{(n)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}, \; n > 2$$

Iz ovog jednostavnog izvoda jasno se vidi da su sve derivacije određene upravo karakterističnom matricom krivulje **K**. Također se vidi da se sve derivacije mogu direktno dobiti samo deriviranjem elemenata matrice parametra *t*. Tako smo krenuli od:

$$T_{Kh} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \mathbf{K}$$

Prva derivacija je bila:
$T_{Kh}^{'} = \begin{bmatrix} 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{K}$

Druga derivacija:
$T_{Kh}^{''} = \begin{bmatrix} 2 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}$

I sve ostale više:
$T_{Kh}^{(n)} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{K}$

11.4. PRIKAZ POMOĆU KUBNIH RAZLOMLJENIH FUNKCIJA

Ove funkcije omogućavaju jednostavan prikaz koničnih krivulja (krivulje koje nastaju kao presjecište stošca i ravnine, npr. kružnice, elipse i sl.), no može dati i infleksije što kvadratne razlomljene funkcije nisu mogle. Kako je riječ o kubnim funkcijama, funkcijske ovisnosti svih koordinata u homogenom prostoru biti će izražene kubnim polinomom:

$$T_{Kh,1} = a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1$$
$$T_{Kh,2} = a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2$$
$$T_{Kh,3} = a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3$$
$$T_{Kh,h} = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$$

Povratkom u radni prostor dobiva se:

$$T_{K,1} = x = \frac{a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$

$$T_{K,2} = y = \frac{a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$

$$T_{K,3} = z = \frac{a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3}{a \cdot t^3 + b \cdot t^2 + c \cdot t + d}$$

Jednadžbe dane za homogenih prostor opet vode na matrični zapis:

$$T_{Kh} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & a_3 & a \\ b_1 & b_2 & b_3 & b \\ c_1 & c_2 & c_3 & c \\ d_1 & d_2 & d_3 & d \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \mathbf{A}$$

gdje je **A** karakteristična matrica krivulje.

11.5. PARAMETARSKE DERIVACIJE U HOMOGENOM PROSTORU

U ovom su slučaju sve funkcijske ovisnosti kubne:

$$T_{Kh,1} = a_1 \cdot t^3 + b_1 \cdot t^2 + c_1 \cdot t + d_1$$

$$T_{Kh,2} = a_2 \cdot t^3 + b_2 \cdot t^2 + c_2 \cdot t + d_2$$

$$T_{Kh,3} = a_3 \cdot t^3 + b_3 \cdot t^2 + c_3 \cdot t + d_3$$

$$T_{Kh,h} = a \cdot t^3 + b \cdot t^2 + c \cdot t + d$$

Deriviranjem po parametru t dobiva se:

$$\frac{dT_{Kh,1}}{dt} = a_1 \cdot 3t^2 + b_1 \cdot 2t + c_1$$

$$\frac{dT_{Kh,2}}{dt} = a_2 \cdot 3t^2 + b_2 \cdot 2t + c_2$$

$$\frac{dT_{Kh,3}}{dt} = a_3 \cdot 3t^2 + b_3 \cdot 2t + c_3$$

$$\frac{dT_{Kh,h}}{dt} = a \cdot 3t^2 + b \cdot 2t + c$$

što se matrično može zapisati kao:

$$T'_{Kh} = \begin{bmatrix} T'_{Kh,1} & T'_{Kh,2} & T'_{Kh,3} & T'_{Kh,h} \end{bmatrix} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \cdot \mathbf{A}$$

Više derivacije iznose:

$$T''_{Kh} = \begin{bmatrix} T''_{Kh,1} & T''_{Kh,2} & T''_{Kh,3} & T''_{Kh,h} \end{bmatrix} = \begin{bmatrix} 6t & 2 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}$$
$$T'''_{Kh} = \begin{bmatrix} T'''_{Kh,1} & T'''_{Kh,2} & T'''_{Kh,3} & T'''_{Kh,h} \end{bmatrix} = \begin{bmatrix} 6 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}$$
$$T^{(n)}_{Kh} = \begin{bmatrix} T^{(n)}_{Kh,1} & T^{(n)}_{Kh,2} & T^{(n)}_{Kh,3} & T^{(n)}_{Kh,h} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix} \cdot \mathbf{A}, \quad n > 3$$

11.6. VEZA IZMEĐU PARAMETARSKIH DERIVACIJA U RADNOM I HOMOGENOM PROSTORU

Veza između radnih i homogenih koordinata već nam je poznata:

$$T_{K,1} = \frac{T_{Kh,1}}{T_{Kh,h}} \qquad T_{K,2} = \frac{T_{Kh,2}}{T_{Kh,h}} \qquad T_{K,3} = \frac{T_{Kh,3}}{T_{Kh,h}}$$

gdje je T_K točka u radnom prostoru, a T_{Kh} točka u homogenom prostoru. Ovu ovisnost možemo i kraće zapisati matrično:

$$T_{Kh} = T_{Kh,h} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \end{bmatrix} = T_{Kh,h} \cdot \begin{bmatrix} T_K & 1 \end{bmatrix}$$

Pretpostavimo da su nam poznate parametarske derivacije koordinata u radnom prostoru, odnosno da znamo:

$$T'_{K,1} \equiv \frac{dT_{K,1}}{dt} \qquad T'_{K,2} \equiv \frac{dT_{K,2}}{dt} \qquad T'_{K,3} \equiv \frac{dT_{K,3}}{dt}$$

te da znamo funkciju kojom ćemo vršiti prebacivanje u homogeni prostor $T_{Kh,h}$ i njezinu derivaciju $T'_{Kh,h}$.

Kako vrijede veze:

$$T_{Kh,1} = T_{K,1} \cdot T_{Kh,h} \qquad T_{Kh,2} = T_{K,2} \cdot T_{Kh,h} \qquad T_{Kh,3} = T_{K,3} \cdot T_{Kh,h}$$

parametarske derivacije u homogenom prostoru mogu se prikazati:

$$T'_{Kh,1} = (T_{K,1} \cdot T_{Kh,h})' = T'_{K,1} \cdot T_{Kh,h} + T_{K,1} \cdot T'_{Kh,h}$$
$$T'_{Kh,2} = (T_{K,2} \cdot T_{Kh,h})' = T'_{K,2} \cdot T_{Kh,h} + T_{K,2} \cdot T'_{Kh,h}$$
$$T'_{Kh,3} = (T_{K,3} \cdot T_{Kh,h})' = T'_{K,3} \cdot T_{Kh,h} + T_{K,3} \cdot T'_{Kh,h}$$

Zapisano matrično dobije se:

$$T'_{Kh} = \begin{bmatrix} T'_{Kh,1} & T'_{Kh,2} & T'_{Kh,3} & T'_{Kh,h} \end{bmatrix} =$$

$$= \begin{bmatrix} T'_{K,1} \cdot T_{Kh,h} + T_{K,1} \cdot T'_{Kh,h} & T'_{K,2} \cdot T_{Kh,h} + T_{K,2} \cdot T'_{Kh,h} & T'_{K,3} \cdot T_{Kh,h} + T_{K,3} \cdot T'_{Kh,h} & T'_{Kh,h} \end{bmatrix} =$$

$$= \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \\ T'_{K,1} & T'_{K,2} & T'_{K,3} & 0 \end{bmatrix} = \begin{bmatrix} T'_{Kh,h} & T_{Kh,h} \end{bmatrix} \cdot \begin{bmatrix} T_K & 1 \\ T'_K & 0 \end{bmatrix}$$

Na ovaj način dobili smo direktnu vezu između traženih parametarskih derivacija u homogenom prostoru i poznatih parametarskih derivacija u radnom prostoru.

Druga parametarska derivacija u homogenom prostoru dobije se deriviranjem prve parametarske derivacije u homogenom prostoru; nakon što se deriviraju izrazi i nakon ubacivanja u matricu dobiva se:

$$T''_{Kh} = \left[T''_{Kh,1} \quad T''_{Kh,2} \quad T''_{Kh,3} \quad T''_{Kh,h}\right] =$$

$$= \left[(T'_{K,1} \cdot T_{Kh,h} + T_{K,1} \cdot T'_{Kh,h})' \quad (T'_{K,2} \cdot T_{Kh,h} + T_{K,2} \cdot T'_{Kh,h})' \quad (T'_{K,3} \cdot T_{Kh,h} + T_{K,3} \cdot T'_{Kh,h})' \quad T''_{Kh,h}\right] =$$

$$= \left[T''_{Kh,h} \quad 2 \cdot T'_{Kh,h} \quad T_{Kh,h}\right] \cdot \begin{bmatrix} T_{K,1} & T_{K,2} & T_{K,3} & 1 \\ T'_{K,1} & T'_{K,2} & T'_{K,3} & 0 \\ T''_{K,1} & T''_{K,2} & T''_{K,3} & 0 \end{bmatrix} = \left[T''_{Kh,h} \quad 2 \cdot T'_{Kh,h} \quad T_{Kh,h}\right] \cdot \begin{bmatrix} T_K & 1 \\ T'_K & 0 \\ T''_K & 0 \end{bmatrix}$$

I više derivacije mogu se izvesti na sličan način.

11.7. PRIMJERI

Pomoću razlomljene kvadratne funkcije želimo odrediti krivulju koja će prolaziti slijedećim točkama:

- za $t=t_1=0$ kroz točku [R 0 0 1]
- za $t=t_2=0.5$ kroz točku [0 R 0 1]
- za $t=t_2=1$ kroz točku [-R 0 0 1]

Na prvi pogled popis točaka odgovara kružnici, no pogledajmo što ćemo dobiti. U poglavlju 11.2. pokazano je kako se na temelju ovih podataka računa matrica **K**. Dobiva se:

$$\mathbf{K} = \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} R & 0 & 0 & 1 \\ 0 & R & 0 & 1 \\ -R & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & -4R & 0 & 0 \\ -2R & 4R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}$$

Posljednji stupac $(0\ 0\ 1)^T$ nam govori da je riječ o paraboli, a ne o koničnoj krivulji. Sve konične krivulje imaju posljednji stupac jednak $(1\ 0\ 1)^T$.

Svaka točka ove naše krivulje određena je jednostavnom relacijom:

$$T_{Kh} = \left[t^2 \quad t \quad 1\right] \cdot \mathbf{K} = \left[t^2 \quad t \quad 1\right] \cdot \begin{bmatrix} 0 & -4R & 0 & 0 \\ -2R & 4R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}$$

Ukoliko želimo prikazati kružnicu, morati ćemo se poslužiti trikom. Znamo da kod kružnice vrijedi veza:

$$x = R\cos\varphi$$
$$y = R\sin\varphi$$

Iz matematike znamo i vezu između funkcija *sin* i *cos* sa funkcijom *tg*. Primjenimo li tu vezu, dobivamo:

$$t = tg\frac{\varphi}{2} \qquad x = R\frac{1-t^2}{1+t^2} \qquad y = R\frac{2t}{1+t^2}$$

Uzmemo li nazivnik razlomaka za funkciju homogenog parametra, dobiva se:

$$T_{Kh} = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -R & 0 & 0 & 1 \\ 0 & 2R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}$$

odnosno matrica **K** iznosi:

$$\mathbf{K} = \begin{bmatrix} -R & 0 & 0 & 1 \\ 0 & 2R & 0 & 0 \\ R & 0 & 0 & 1 \end{bmatrix}$$

Zadnji stupac nam govori da smo dobili koničnu krivulju.

11.8. ODREĐIVANJE MATRICE A

U nastavku ćemo odrediti matricu **A** za krivulju koja je zadana pomoću početne i završne točke u radnom prostoru, te prve derivacije u tim točkama; početna točka dobiva se za parametar $t=t_0=0$ dok se završna točka dobiva se za parametar $t=t_1=1$. Dakle, poznato nam je slijedeće:

$$t = t_0 = 0 \Rightarrow T_0 = \begin{bmatrix} T_{0,1} & T_{0,2} & T_{0,3} \end{bmatrix}$$
$$t = t_1 = 1 \Rightarrow T_1 = \begin{bmatrix} T_{1,1} & T_{1,2} & T_{1,3} \end{bmatrix}$$
$$t = t_0 = 0 \Rightarrow T_0' = \begin{bmatrix} T_{0,1}' & T_{0,2}' & T_{0,3}' \end{bmatrix}$$
$$t = t_1 = 1 \Rightarrow T_1' = \begin{bmatrix} T_{1,1}' & T_{1,2}' & T_{1,3}' \end{bmatrix}$$

Kako prva točka leži na krivulji, tada zadovoljava jednadžbu krivulje. Zato možemo pisati:

$$T_{0h} = \begin{bmatrix} T_{0h,1} \cdot T_{0h,h} & T_{0h,2} \cdot T_{0h,h} & T_{0h,3} \cdot T_{0h,h} & T_{0h,h} \end{bmatrix} = \begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \end{bmatrix} \cdot \mathbf{A}$$

Ovo se kraće može zapisati sažimanjem prva tri stupa u jedan, u kojem ćemo tada pisati trokomponentnu točku radnog prostora (čime dimenzija matrice ostaje očuvana):

$$T_{0h} = \begin{bmatrix} T_0 \cdot T_{0h,h} & T_{0h,h} \end{bmatrix} = \begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \end{bmatrix} \cdot \mathbf{A}$$

Za prvu derivaciju u homogenom prostoru u poglavlju 11.6. smo izveli vezu sa derivacijom u radnom prostoru:

$$T'_{0h} = \left[T'_0 \cdot T_{0h,h} + T_0 \cdot T'_{0h,h} \quad T'_{0h,h}\right] = \left[3t_0^2 \quad 2t_0 \quad 1 \quad 0\right]$$

Slično se može pisati i za drugu točku, te se dobiju četiri jednadžbe:

$$T_{0h} = \left[T_0 \cdot T_{0h,h} \quad T_{0h,h}\right] = \left[t_0^3 \quad t_0^2 \quad t_0 \quad 1\right] \cdot \mathbf{A}$$
$$T_{1h} = \left[T_1 \cdot T_{1h,h} \quad T_{1h,h}\right] = \left[t_1^3 \quad t_1^2 \quad t_1 \quad 1\right] \cdot \mathbf{A}$$
$$T'_{0h} = \left[T'_0 \cdot T_{0h,h} + T_0 \cdot T'_{0h,h} \quad T'_{0h,h}\right] = \left[3t_0^2 \quad 2t_0 \quad 1 \quad 0\right]$$
$$T'_{1h} = \left[T'_1 \cdot T_{1h,h} + T_1 \cdot T'_{1h,h} \quad T'_{1h,h}\right] = \left[3t_1^2 \quad 2t_1 \quad 1 \quad 0\right]$$

Sustav se može zapisati kao jedna poveća matrica, te se dobiva:

$$\begin{bmatrix} T_0 \cdot T_{0h,h} & T_{0h,h} \\ T_1 \cdot T_{1h,h} & T_{1h,h} \\ T'_0 \cdot T_{0h,h} + T_0 \cdot T'_{0h,h} & T'_{0h,h} \\ T'_1 \cdot T_{1h,h} + T_1 \cdot T'_{1h,h} & T'_{1h,h} \end{bmatrix} = \begin{bmatrix} t_0^3 & t_0^2 & t_0 & 1 \\ t_1^3 & t_1^2 & t_1 & 1 \\ 3t_0^2 & 2t_0 & 1 & 0 \\ 3t_1^2 & 2t_1 & 1 & 0 \end{bmatrix} \cdot \mathbf{A}$$

Matricu na desnoj strani označiti ćemo oznakom **B** (matrica parametara). Uvrštavanjem vrijednosti za parametre dobiva se:

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}$$

Matricu na lijevoj strani prethodne jednadžbe možemo malo preurediti tako da je napišemo kao umnožak dviju matrica, kako slijedi:

$$\begin{bmatrix} T_0 \cdot T_{0h,h} & T_{0h,h} \\ T_1 \cdot T_{1h,h} & T_{1h,h} \\ T'_0 \cdot T_{0h,h} + T_0 \cdot T'_{0h,h} & T'_{0h,h} \\ T'_1 \cdot T_{1h,h} + T_1 \cdot T'_{1h,h} & T'_{1h,h} \end{bmatrix} = \begin{bmatrix} T_{0h,h} & 0 & 0 & 0 \\ 0 & T_{1h,h} & 0 & 0 \\ T'_{0h,h} & 0 & T_{0h,h} & 0 \\ 0 & T'_{1h,h} & 0 & T_{1h,h} \end{bmatrix} \cdot \begin{bmatrix} T_0 & 1 \\ T_1 & 1 \\ T'_0 & 0 \\ T'_1 & 0 \end{bmatrix} = \mathbf{H} \cdot \mathbf{V}$$

Prva matrica u umnošku naziva se matrica **H** (matrica ovisna samo o homogenom parametru i funkciji po kojoj se on mijenja), dok se druga matrica naziva matrica **V** (i ona je ovisna samo o zadanim točkama i derivacijama u njima). Sada se početna jednadžba može pisati:

$$\mathbf{H} \cdot \mathbf{V} = \mathbf{B} \cdot \mathbf{A}$$

Za matricu **A** se dobiva:

$$\mathbf{A} = \mathbf{B}^{-1} \cdot \mathbf{H} \cdot \mathbf{V} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}$$

Matrica **M** naziva se **univerzalna transformacijska matrica** i jednaka je inverzu matrice **B**. Kako smo matricu **B** već odredili, matrica **M** iznosi:

$$\mathbf{M} = \mathbf{B}^{-1} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Ukoliko su nam poznate točke krivulje i derivacije u njima (**V**), te homogeni parametri i njihove derivacije za obje točke (**H**), na temelju izvedene relacije možemo odrediti matricu **A** kao umnožak univerzalne transformacijske matrice, matrice homogenog parametra i matrice točaka.

11.9. HERMITOVA KRIVULJA

Hermitova krivulja specijalan je slučaj kubne razlomljene krivulje, po tome što funkcija po kojoj se mijenja homogeni parametar nije kubna, već je konstanta i iznosi 1. Drugim riječima, za točku krivulje T_{Kh} u homogenom prostoru vrijedi:

$$T_{Kh,1} = P_3(t) \quad T_{Kh,2} = Q_3(t) \quad T_{Kh,3} = R_3(t) \quad T_{Kh,h} = 1$$

gdje su P, Q i R polinomi po varijabli *t*.

Iskoristimo sada izraz za matricu A:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}$$

Matrica **M** je poznata, matricu **V** lako odredimo ako znamo dvije točke i derivacije u njima. Ostaje nam još matrica **H**. Ona u sebi sadrži homogene parametre (koji su u ovom slučaju za sve točke jednaki 1 - Hermitova krivulja!), i njihove derivacije. Kako su funkcije homogenih parametara konstante, njihova je derivacije 0! Time matrica **H** postaje jedinična!

$$\mathbf{H} = \begin{bmatrix} T_{0h,h} & 0 & 0 & 0 \\ 0 & T_{1h,h} & 0 & 0 \\ T'_{0h,h} & 0 & T_{0h,h} & 0 \\ 0 & T'_{1h,h} & 0 & T_{1h,h} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \mathbf{I}$$

Između Hermitove krivulje i Bezierove krivulje za kubni slučaj postoji veza! Već smo rekli da Hermitova krivulje definira da je svim točkama homogeni parametar jednak jedan. To znači da su prve tri homogene koordinate jednake radnim koordinatama. U tom je slučaju za prelazak iz homogenih u radne koordinate dovoljno u izrazu

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{V}$$

matricu **V** zamijeniti samo s njena prva tri stupca. Dobije se:

$$\vec{p}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \underline{M} \cdot \begin{bmatrix} \vec{p}_0 \\ \vec{p}_1 \\ \vec{p}_0^{'} \\ \vec{p}_1^{'} \end{bmatrix} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{r}_0 \\ \vec{r}_1 \\ \vec{r}_2 \\ \vec{r}_3 \end{bmatrix}$$

Lijeva strana jednadžbe dolazi od Hermitove krivulje; desna strana jednadžbe je Bezierova krivulja! Zaključak je da to mogu biti iste krivulje! Pri tome se Hermitova krivulja zadaje početnom i krajnjom točkom i derivacijama u njima, dok se Bezierova krivulja zadaje preko četiri točke: početnu, dvije kontrolne i završnu.

11.10. ODREĐIVANJE MATRICE A - PRIMJER

Zadano je slijedeće:

$$T_0 = \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \quad t = t_0 = 0$$
$$T_1 = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \quad t = t_1 = 1$$
$$T_0^{'} = \begin{bmatrix} 1 & 1 & 0 & 0 \end{bmatrix} \quad t = t_0 = 0$$
$$T_1^{'} = \begin{bmatrix} 1 & -1 & 0 & 0 \end{bmatrix} \quad t = t_1 = 1$$

Treba odrediti matricu **A**. Za matricu **A** smo izveli izraz:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V}$$

M je poznato:

$$\mathbf{M} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

H možemo odrediti:

$$\mathbf{H} = \begin{bmatrix} T_{0h,h} & 0 & 0 & 0 \\ 0 & T_{1h,h} & 0 & 0 \\ T_{0h,h}^{'} & 0 & T_{0h,h} & 0 \\ 0 & T_{1h,h}^{'} & 0 & T_{1h,h} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & 0 & 1 & 0 \\ 0 & b & 0 & 1 \end{bmatrix}$$

Otkuda sada *a* i *b*? Poznavanje iznosa homogenog parametra u nekoj točci ne govori nam ništa o derivaciji u toj točci! Da bismo znali derivaciju, moramo znati funkciju po kojoj se mijenja homogeni parametar! Zato ostavljamo dvije nepoznanice: *a* i *b* koje ćemo izračunati kasnije.

Matrica **V** nam je također poznata:

$$\mathbf{V} = \begin{bmatrix} T_0 & 1 \\ T_1 & 1 \\ T_0' & 0 \\ T_1' & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Množenjem ove tri matrice dobiva se:

$$\mathbf{A} = \mathbf{M} \cdot \mathbf{H} \cdot \mathbf{V} = \begin{bmatrix} b & 0 & 0 & a+b \\ -b & -1 & 0 & -2a-b \\ 1 & 1 & 0 & a \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

No da bismo odredili matricu **A** jednoznačno, vidimo da nam nedostaje još jedan uvjet. Tako možemo tražiti slijedeće: neka krivulja za $t=t_2=1/2$ prođe kroz točku radnog prostora (1/2 1/2 0). Ovaj podatak pomoći će nam da odredimo traženu matricu. Ali računu treba pristupiti oprezno! Zadali smo si točku u radnom prostoru i tražimo da krivulja prođe kroz nju. No jednadžba krivulje daje sve točke u homogenom prostoru! I to za svaku točku radnog prostora daje samo jednu točku homogenog prostora (iako tih točaka za svaku točku radnog prostora ima beskonačno). Ukoliko ovu činjenicu ignoriramo, mogli bismo reći slijedeće: točka leži na krivulji, pa vrijedi:

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix} = \begin{bmatrix} t_2^3 & t_2^2 & t_2 & 1 \end{bmatrix} \cdot \mathbf{A}$$

tj.

$$\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{8} & \frac{1}{4} & \frac{1}{2} & 1 \end{bmatrix} \cdot \begin{bmatrix} b & 0 & 0 & a+b \\ -b & -1 & 0 & -2a-b \\ 1 & 1 & 0 & a \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ovaj sustav predstavlja četiri jednadžbe sa četiri nepoznanice, i pri tome je nerješiv (zapravo, rješiv je: rješenje ne postoji). Naime, već izjednačavanjem po drugoj komponenti dobiva se:

$$\frac{1}{2} = -\frac{1}{4} + \frac{1}{2} = \frac{1}{4}$$

što je očito besmisleno. Jedno od loših tumačenja ovog rješenja je da krivulja jednostavno ne može proći kroz tu točku uz zadane parametre. I ovo je mjesto na kojem treba razmisliti. Krivulja (očito) ne može proći kroz tu točku homogenog prostora, no može li možda proći kroz neku drugu točku homogenog prostora a da pri tome prolazi kroz istu točku radnog prostora? Odgovor na ovo je potvrdan! Naime, sustav je ispravno napisati i rješavati po komponentama samo ukoliko su sve komponente u potpunosti nezavisne - što ovdje nisu. Da bismo dobili naše rješenje, potrebno je primijeniti zavisnosti koje znamo i raditi jednačenje po stvarno-nezavisnim komponentama: komponentama radnog prostora! Tada ćemo dobiti sustave:

$$\frac{\frac{1}{8}b-\frac{1}{4}b+\frac{1}{2}}{\frac{1}{8}(a+b)+\frac{1}{4}(-2a-b)+\frac{1}{2}a+1}=\frac{\frac{1}{2}}{1}$$

$$\frac{-\frac{1}{4}+\frac{1}{2}}{\frac{1}{8}(a+b)+\frac{1}{4}(-2a-b)+\frac{1}{2}a+1}=\frac{\frac{1}{2}}{1}$$

$$\frac{0}{\frac{1}{8}(a+b)+\frac{1}{4}(-2a-b)+\frac{1}{2}a+1}=\frac{0}{1}$$

Sada umjesto četiri "nezavisne" jednadžbe imamo samo tri, i to rješive. Treća jednadžba je identitet pa otpada. Prve dvije daju:

$$\frac{-b+4}{a-b+8}=\frac{1}{2} \qquad \frac{2}{a-b+8}=\frac{1}{2}$$

$$\Rightarrow a=-2$$
$$\Rightarrow b=2$$

Tražena matrica **A** glasi:

$$\mathbf{A}=\begin{bmatrix} 2 & 0 & 0 & 0 \\ -2 & -1 & 0 & 2 \\ 1 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ostalo nam je još da pogledamo kroz koju je točku homogenog prostora krivulja prošla za traženu točku radnog prostora:

$$T_{Kh}=\begin{bmatrix} \left(\frac{1}{2}\right)^3 & \left(\frac{1}{2}\right)^2 & \frac{1}{2} & 1 \end{bmatrix}\cdot\begin{bmatrix} 2 & 0 & 0 & 0 \\ -2 & -1 & 0 & 2 \\ 1 & 1 & 0 & -2 \\ 0 & 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} \frac{1}{4} & \frac{1}{4} & 0 & \frac{1}{2} \end{bmatrix}$$

a odgovarajuća točka radnog prostora je doista tražena:

$$T_K=\begin{bmatrix} \frac{\frac{1}{4}}{\frac{1}{2}} & \frac{\frac{1}{4}}{\frac{1}{2}} & \frac{0}{\frac{1}{2}} \end{bmatrix}=\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}$$

12. 2D TRANSFORMACIJE

12.1. UVOD

Postoji nekoliko elementarnih transformacija koje djeluju nad točkom. Ukoliko se ta točka nalazi u 2D prostoru, tada govorimo o 2D transformacijama. U nastavku ćemo obraditi slijedeće elementarne transformacije:

- Translacija
- Rotacija
- Skaliranje
- Smik

Pri tome ćemo uvijek koristiti točke u homogenom prostoru, te u matričnom obliku. Svaku elementarnu transformaciju predstavlja operator koji ćemo predstaviti kvadratnom matricom. U 2D prostoru ovi operatori imaju kvadratne matrice reda 3, budući da se točke zapisuju kao jednoretčane matrice sa tri stupca (x, y i h). Djelovanje operatora na točku dobiva se množenjem matrice točke i matrice operatora:

$T_h^{'} = T_h \cdot \underline{\Psi}$

gdje je $\underline{\Psi}$ matrica operatora.

Budući da se djelovanje operatora opisuje matričnim množenjem, lako se može doći do slijedećeg zaključka: **ukoliko na jednu točku djeluje više transformacija, tada je bitan redoslijed djelovanja transformacija**! Dokaz ove tvrdnje je trivijalan. Naime, matrično množenje nije komutativno pa se za različite redoslijede množenja matrica dobivaju različiti rezultati.

Djelovanje više transformacija može se zapisati:

$T_h^{'} = T_h \cdot \underline{\Psi_1} \cdot \underline{\Psi_2} \cdot \ldots \cdot \underline{\Psi_n} = T_h \cdot \underline{\Psi'}$

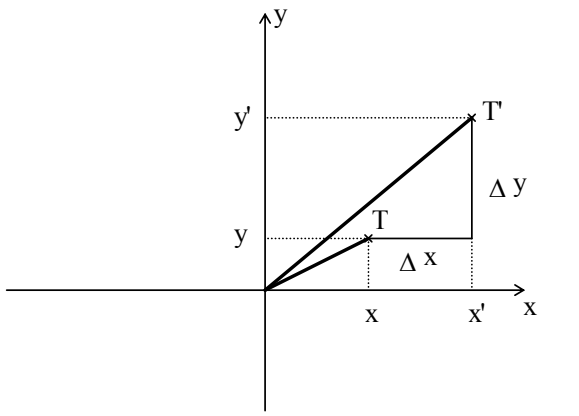
$\underline{\Psi'} = \underline{\Psi_1} \cdot \underline{\Psi_2} \cdot \ldots \cdot \underline{\Psi_n}$

pri čemu na točku prvo djeluje operator ψ_1, pa ψ_2 itd.

Postoji i jedna iznimka za gornje pravilo: **unutar jedne vrste transformacija, redoslijed djelovanja pojedine transformacije nije bitan**! Ovo se može i matematički dokazati, no ostanimo na logičkom dokazu: rotiramo li točku za kut α pa za kut β, dobiti ćemo isti rezultat kao i da rotiramo točku najprije za kut β pa onda za kut α.

12.2. TRANSLACIJA

Translacija je transformacija koja svakoj komponenti točke u radnom prostoru dodaje određeni pomak. Primjer translacije prikazan je na slici 12.2.1. Točka T translatira se u točku T'.



12.2.1.

Postupak translacije može se opisati slijedećim jednadžbama u radnom prostoru:

$T_1^{'} = T_1 + \Delta_1$

$T_2^{'} = T_2 + \Delta_2$

gdje su T_1 i T_2 komponente točke T a T_1' i T_2' komponente točke T'. Iskoristimo li vezu između radnih i homogenih koordinata

$$T_1 = \frac{T_{h,1}}{T_{h,h}} \qquad T_2 = \frac{T_{h,2}}{T_{h,h}}$$

dobiva se:

$T_{h,1}^{'} = T_{h,1} + \Delta_1 \cdot T_{h,h}$

$T_{h,2}^{'} = T_{h,2} + \Delta_2 \cdot T_{h,h}$

pri čemu su $T_{h,i}$ komponente homogenog zapisa točke T (odnosno tada T_h), dok je $T_{h,h}$ homogeni parametar.

Prethodne relacije pokazuju da se točka T'_h može dobiti matričnim množenjem točke T_h sa operatorom translacije:

$$\left[T_{h,1}^{'} \quad T_{h,2}^{'} \quad T_{h,h}^{'}\right] = \left[T_{h,1} \quad T_{h,2} \quad T_{h,h}\right] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta_1 & \Delta_2 & 1 \end{bmatrix}$$

pa se operator translacije zapisuje:

$$\underline{\Psi_{tr}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ \Delta_1 & \Delta_2 & 1 \end{bmatrix}$$

Inverzni operator ovom operatoru je operator translacije za negativan pomak:

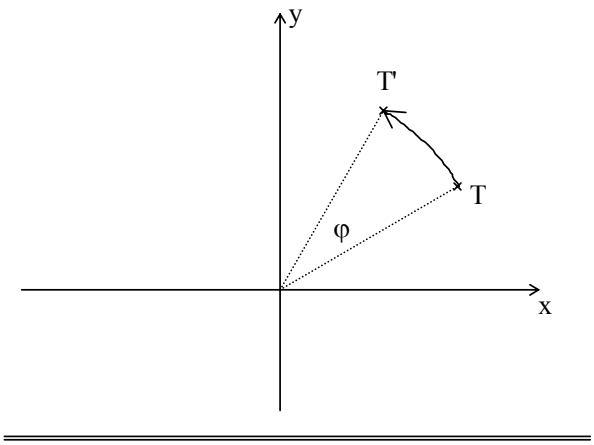
$$\underline{\Psi_{tr}^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -\Delta_1 & -\Delta_2 & 1 \end{bmatrix}$$

jer vrijedi:

$$\underline{\Psi_{tr}} \cdot \underline{\Psi_{tr}^{-1}} = \underline{\Psi_{tr}^{-1}} \cdot \underline{\Psi_{tr}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

12.3. ROTACIJA

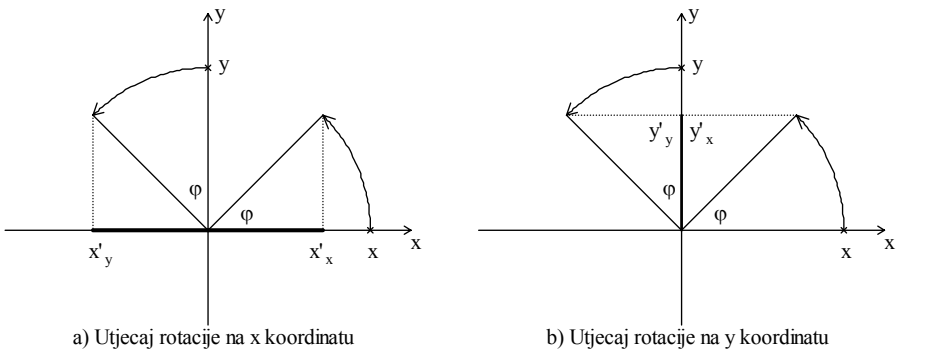
Rotacija je translacija koja točku rotira oko ishodišta za zadani kut φ. Smjer rotacije može biti podudaran sa smjerom kazaljke na satu, ili suprotan od smjera kazaljke na satu. Kako je matematički "pozitivan" smjer rotacije definiran kao smjer suprotan od smjera kazaljke na satu, tako će u nastavku biti izveden operator koji rotira točku u smjeru suprotnom od kazaljke na satu. Ukoliko se želi rotirati u smjeru kazaljke na satu, dovoljno je umjesto kuta φ rotirati za smjer -φ. Primjer rotacije prikazan je na slici 12.3.1.



12.3.1.

Da bismo izveli matricu operatora rotacije, potrebno se je prisjetiti linearne algebre, gdje smo naučili da se djelovanje operatora može zapisati kao zbroj djelovanja operatora na svaku komponentu baze prostora pojedinačno (naravno, ako operator zadovoljava određene uvjete

koji su ovdje zadovoljeni). Budući da izvodimo rotaciju u ravnini, imamo dvije komponente baze: komponentu u smjeru osi x, i komponentu u smjeru osi y. Djelovanje operatora na te dvije komponente prikazano je na slici 12.3.2.



a) Utjecaj rotacije na x koordinatu b) Utjecaj rotacije na y koordinatu

12.3.2.

Pogledajmo najprije sliku 12.3.2.a. Ako točku koja leži na osi x zarotiramo za kut φ, dobiti ćemo točku čija je x'x koordinata jednaka $x_x^{'} = x \cdot \cos\varphi$. Ako točku koja leži na osi y zarotiramo za kut φ, dobiti ćemo točku čija je x'y koordinata jednaka: $x_y^{'} = -y \cdot \sin\varphi$. Tada je djelovanje operatora na točku daje x' komponentu jednaku sumi ova dva djelovanja:

$$x^{'} = x \cdot \cos\varphi - y \cdot \sin\varphi$$

Da bismo utvrdili kako se tvori y' komponenta točke, pogledajmo djelovanje operatora opet na točke (x,0) i (0,y) samo sada pratimo što se događa sa y-projekcijama nastalih točaka, kao što je prikazano na slici 12.3.2.b. Djelovanjem na (x,0) dobiva se $y_x^{'} = x \cdot \sin\varphi$ a djelovanjem na (0,y) dobiva se $y_y^{'} = y \cdot \cos\varphi$. Ukupno djelovanje daje:

$$y^{'} = x \cdot \sin\varphi + y \cdot \cos\varphi$$

Zapisano u matričnom obliku u homogenim koordinatama dobiva se:

$$\left[T_{h,1}^{'} \quad T_{h,2}^{'} \quad T_{h,h}^{'}\right] = \left[T_{h,1} \quad T_{h,2} \quad T_{h,h}\right] \cdot \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

pa se operator rotacije zapisuje:

$$\underline{\Psi_{ro}} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ukoliko želimo operator koji rotira u smjeru kazaljke na satu, dovoljno je umjesto kuta φ rotirati za smjer -φ, pa se uvrštavanjem u matricu operatora i uzimanjem u obzir parnosti funkcije cos i neparnosti funkcije sin dobiva operator:

$$\underline{\Psi'_{ro}} = \begin{bmatrix} \cos\varphi & -\sin\varphi & 0 \\ \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Korištenjem ovog operatora uz pozitivne vrijednosti kuta φ dobiti ćemo rotaciju u smjeru kazaljke na satu.

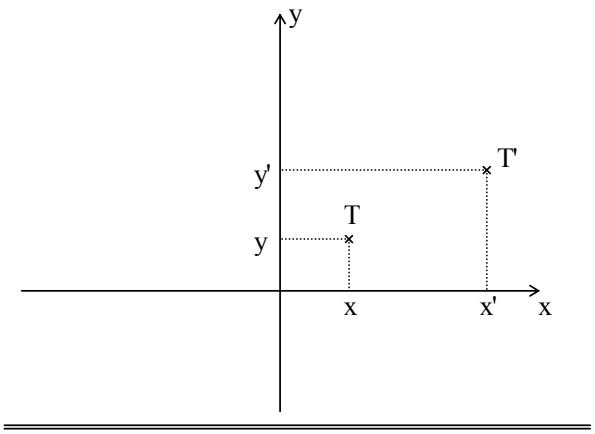
Odmah se može uočiti da se operatori $\underline{\Psi_{ro}}$ i $\underline{\Psi'_{ro}}$ međusobno poništavaju u djelovanju te su inverzi jedan drugome. Naime, ako pogledamo operator koji opisuje djelovanje oba operatora, dobiva se:

$$\underline{\Psi_{ro}} \cdot \underline{\Psi'_{ro}} = \underline{\Psi'_{ro}} \cdot \underline{\Psi_{ro}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

iz čega je vidljivo da slijedno djelovanje jednog pa drugog operatora vraća točku u prvobitni položaj.

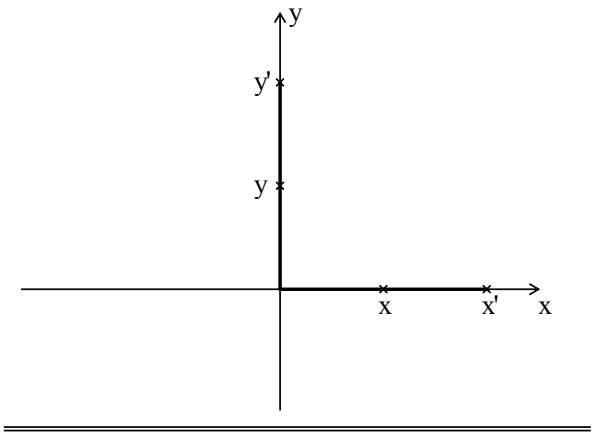
12.4. SKALIRANJE

Skaliranje je transformacija koja "skalira" (rasteže ili steže) svaku komponentu točke. Pri tome je skaliranje svake komponente određeno faktorom skaliranja, pri čemu faktori ne moraju biti isti za sve komponente. Ukoliko je to slučaj, tada govorimo o neproporcionalnom skaliranju. Ukoliko su faktori skaliranja jednaki za sve komponente, tada govorimo o proporcionalnom skaliranju. Primjer skaliranja prikazan je na slici 12.4.1.



12.4.1.

Djelovanje operatora po komponentama prikazano je na slici 12.4.2.



12.4.2.

Vrijedi:

$$x' = k_1 \cdot x$$
$$y' = k_2 \cdot y$$

odnosno prelaskom na homogene koordinate:

$$\begin{bmatrix} T'_{h,1} & T'_{h,2} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,1} & T_{h,2} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

pa se operator skaliranja zapisuje:

$$\underline{\Psi_{sk}} = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Ukoliko želimo proporcionalno skaliranje, tada će k_1 biti jednak k_2 što možemo nazvati k, pa operator poprima oblik:

$$\underline{\Psi_{psk}} = \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

što se još može zapisati i u obliku:

$$\underline{\Psi_{psk}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \frac{1}{k} \end{bmatrix}$$

Inverzni operator operatoru skaliranja je skaliranje recipročnim koeficijentima, što za neproporcionalno skaliranje daje:

$$\underline{\Psi_{sk}^{-1}} = \begin{bmatrix} \frac{1}{k_1} & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

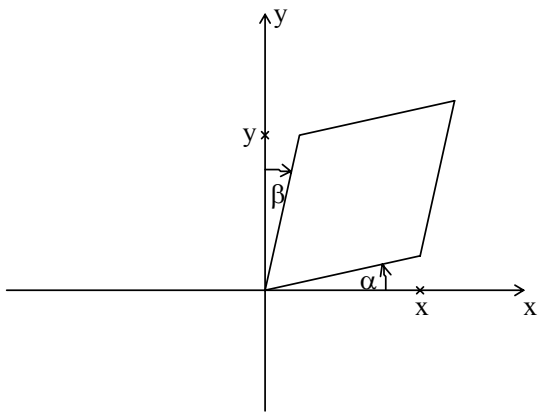
a za proporcionalno skaliranje:

$$\underline{\Psi_{psk}^{-1}} = \begin{bmatrix} \frac{1}{k} & 0 & 0 \\ 0 & \frac{1}{k} & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ ili } \underline{\Psi_{psk}^{-1}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & k \end{bmatrix}$$

ovisno o matrici kojom je vršeno proporcionalno skaliranje.

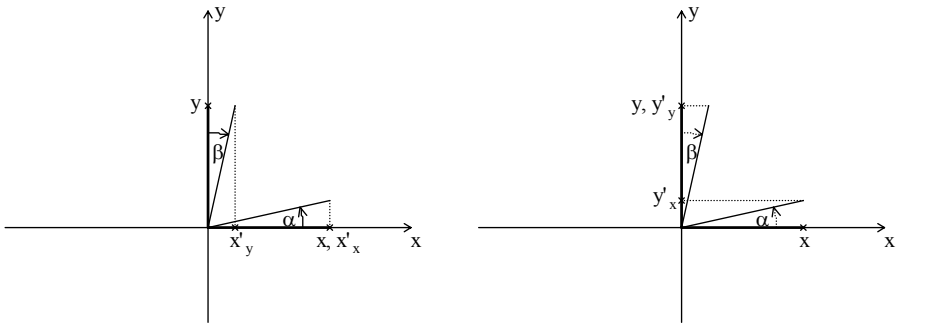
12.5. SMIK

Smik je "uzdužna" deformacija čije se djelovanje najbolje vidi sa slike 12.5.1. Sastoji se od deformacije uzduž osi x i deformacije uzduž osi y te se opisuje kutovima β i α.



12.5.1.

Djelovanje operatora na pojedine osi prikazano je na slici 12.5.2.



12.5.2.

Za pojedine komponente proizlazi:

$x' = x'_x + x'_y = x + y \cdot tg\beta$

$y' = y'_y + y'_x = y + x \cdot tg\alpha$

pa se prelaskom na homogene koordinate dobiva:

$$\left[T'_{h,1} \quad T'_{h,2} \quad T'_{h,h}\right] = \left[T_{h,1} \quad T_{h,2} \quad T_{h,h}\right] \cdot \begin{bmatrix} 1 & tg\alpha & 0 \\ tg\beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

pa se operator smika zapisuje:

$$\underline{\Psi_{sm}} = \begin{bmatrix} 1 & tg\alpha & 0 \\ tg\beta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

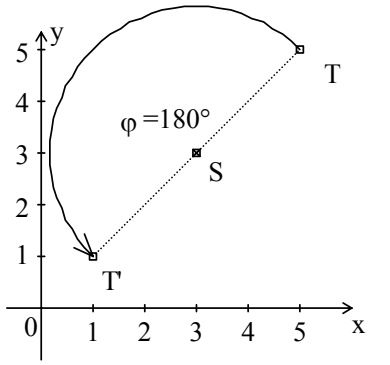
Potražimo li inverz ovog operatora iz uvjeta da umnožak operatora smika i inverznog operatora daje jediničnu matricu dobiva se:

$$\underline{\Psi_{sm}^{-1}} = \begin{bmatrix} \dfrac{1}{1 - tg\alpha \cdot tg\beta} & \dfrac{-tg\alpha}{1 - tg\alpha \cdot tg\beta} & 0 \\ \dfrac{-tg\beta}{1 - tg\alpha \cdot tg\beta} & \dfrac{1}{1 - tg\alpha \cdot tg\beta} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

što nije definirano ukoliko je umnožak tangensa jednak 1.

12.6. PRIMJENA TRANSFORMACIJA

U nastavku ćemo pokazati jedan jednostavan primjer na kojem se mogu uočiti neki interesantni detalji. Potrebno je rotirati točku T(5,5) u smjeru suprotnom od smjera kazaljke na satu za kut φ oko točke S(3,3). Slika 12.6.1. pokazuje što želimo učiniti. Na slici je prikazan primjer rotacije za 180°.

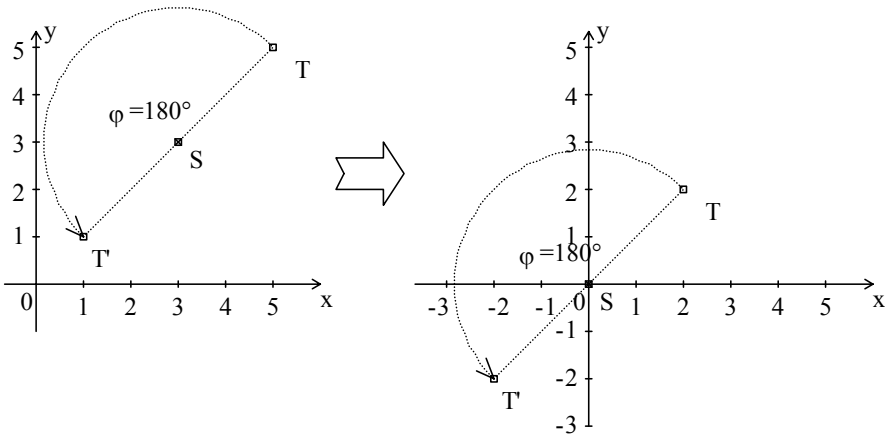


12.6.1.

Pokušamo li direktno primijeniti operator rotacije, rezultat neće dati očekivani rezultat (zapravo, ovisi što ste očekivali). Naime, operator rotacije izveden je tako da rotira točku oko ishodišta! Želimo li rotirati točku oko nekog drugog središta, morati ćemo primijeniti i operator translacije! Evo koraka koje treba napraviti:

1. Točku T potrebno je translatirati translacijom koja bi točku S (željeno središte rotacije) dovela u ishodište koordinatnog sustava. To će biti translacija za $\Delta_1 = -S_1$ i $\Delta_2 = -S_2$ ako je točka S dana komponentama (S_1 S_2 1). Nazovimo ovu tranformaciju ψ_1: $\underline{\Psi_1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -S_1 & -S_2 & 1 \end{bmatrix}$
2. Na ovako dobivenu točku možemo primijeniti operator rotacije jer se sada središte rotacije nalazi u ishodištu! Tada je ψ_2: $\underline{\Psi_2} = \begin{bmatrix} \cos\varphi & \sin\varphi & 0 \\ -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 1 \end{bmatrix}$
3. I konačno, točku koju smo rotirali moramo natrag translatirati inverznom translacijom od one iz prvog koraka: $\underline{\Psi_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ S_1 & S_2 & 1 \end{bmatrix}$

Što se je dogodilo nakon prvog koraka opisuje slika 12.6.2.



12.6.2.

U svakom koraku dobili smo po jedan operator. Zbirni operator ψ koji će napraviti zadanu operaciju dobije se kao umnožak sva tri operatora i to upravo redoslijedom kojim su djelovali:

$$\underline{\Psi} = \underline{\Psi_1} \cdot \underline{\Psi_2} \cdot \underline{\Psi_3}$$

Da je ovo ispravno, možemo se lako uvjeriti pokusom. Ako rotiramo točku T za 180° oko točke S, trebali bismo dobiti točku T'(1,1). Ako izračunamo vrijednost operatora uz zadani kut dobiti ćemo:

$$\underline{\Psi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -3 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 3 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 6 & 6 & 1 \end{bmatrix}$$

Primijenimo li operator na točku T(5,5) dobivamo:

$$\begin{bmatrix} T'_{h,1} & T'_{h,2} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,1} & T_{h,2} & T_{h,h} \end{bmatrix} \cdot \underline{\Psi}$$

$$\begin{bmatrix} T'_{h,1} & T'_{h,2} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} 5 & 5 & 1 \end{bmatrix} \cdot \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 6 & 6 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

Dakle, dobili smo točku koju smo i očekivali.

Sličan postupak koji je naveden u prethodna tri koraka često se provodi jer izvedene elementarne transformacije djeluju obzirom na ishodište, pa treba dobro paziti što se želi postići, a što transformacije zapravo daju.

13. 3D TRANSFORMACIJE

13.1. UVOD

3D transformacije su transformacije nad točkom u 3D prostoru. U nastavku ćemo obraditi iste transformacije koje smo obradili u 2D prostoru. Jedina novost koju donosi 3D prostor su tri operatora rotacije umjesto jednog u 2D prostoru. Tako ćemo obraditi:

1. Translaciju
2. Rotaciju oko osi x
3. Rotaciju oko osi y
4. Rotaciju oko osi z
5. Skaliranje
6. Smik

I podsjetimo se još jednom: redoslijed djelovanja transformacija bitan je za krajnji rezultat, osim ako se ne radi o uzastopnom djelovanju iste transformacije kada je redoslijed nebitan. Naravno, rotacija oko osi x i rotacija oko osi y nisu iste transformacije!

U nastavku će biti dati izrazi za ove transformacije, budući da se oni izvode identično kao i izrazi za 2D transformacije koje smo već obradili u prethodnom poglavlju.

13.2. TRANSLACIJA

Translacija pomiče točku tako da svakoj koordinati točke doda određeni pomak. Vrijedi:

$$x' = x + \Delta_1$$

$$y' = y + \Delta_2$$

$$z' = z + \Delta_3$$

ili nakon prelaska u homogene koordinate:

$$\begin{bmatrix} T'_{h,1} & T'_{h,2} & T'_{h,3} & T'_{h,h} \end{bmatrix} = \begin{bmatrix} T_{h,1} & T_{h,2} & T_{h,3} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta_1 & \Delta_2 & \Delta_3 & 1 \end{bmatrix}$$

što daje operator translacije u 3D:

$$\underline{\Psi_{tr}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \Delta_1 & \Delta_2 & \Delta_3 & 1 \end{bmatrix}$$

Inverz je translacija za negativne pomake:

$$\underline{\Psi_{tr}^{-1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -\Delta_1 & -\Delta_2 & -\Delta_3 & 1 \end{bmatrix}$$

13.3. ROTACIJA

Razlikujemo tri rotacije: rotaciju oko osi x, rotaciju oko osi y te rotaciju oko osi z.

13.3.1. ROTACIJA OKO OSI X

Rotacija oko osi x rotira točku u yz ravnini, pri čemu x-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi x u smjeru suprotnom od smjera kazaljke na satu glasi:

$$\underline{\Psi_{rotx}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Njemu inverzni operator je operator rotacije oko osi x u smjeru kazaljke na satu:

$$\underline{\Psi'_{rotx}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

13.3.2. ROTACIJA OKO OSI Y

Rotacija oko osi y rotira točku u xz ravnini, pri čemu y-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi y u smjeru suprotnom od smjera kazaljke na satu glasi:

$$\underline{\Psi_{roty}} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Njemu inverzni operator je operator rotacije oko osi y u smjeru kazaljke na satu:

$$\underline{\Psi'_{roty}} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

13.3.3. ROTACIJA OKO OSI Z

Rotacija oko osi z rotira točku u xy ravnini, pri čemu z-koordinata točke ostaje nepromijenjena.

Operator rotacije oko osi z u smjeru suprotnom od smjera kazaljke na satu glasi:

$$\underline{\Psi_{rotz}} = \begin{bmatrix} \cos\chi & \sin\chi & 0 & 0 \\ -\sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Njemu inverzni operator je operator rotacije oko osi z u smjeru kazaljke na satu:

$$\underline{\Psi'_{rotz}} = \begin{bmatrix} \cos\chi & -\sin\chi & 0 & 0 \\ \sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Usporedimo li ovu matricu sa matricom dobivenom za 2D rotaciju, vidjeti ćemo da je matrica građena identično! Ovo je posljedica činjenice da smo rotaciju u 2D upravo izveli kao rotaciju u xy ravnini gdje smo rekli da točku rotiramo oko ishodišta; ovo bi se slobodno moglo proširiti pa reći da rotaciju izvodimo oko osi okomite na xy ravninu koja prolazi ishodištem ➔ z-osi!

13.4. SKALIRANJE

Skaliranje svaku koordinatu "skalira" (rasteže ili steže) množeći je sa odgovarajućim koeficijentom. Kako imamo tri koordinate radnog prostora, imati ćemo i tri koeficijenta. Ukoliko su ti koeficijenti međusobno različiti, govorimo o neproporcionalnom skaliranju; inače govorimo o proporcionalnom skaliranju.

Operator skaliranja glasi:

$$\underline{\Psi_{sk}} = \begin{bmatrix} k_1 & 0 & 0 & 0 \\ 0 & k_2 & 0 & 0 \\ 0 & 0 & k_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Inverzni operator je skaliranje recipročnim koeficijentima:

$$\underline{\Psi_{sk}^{-1}} = \begin{bmatrix} \frac{1}{k_1} & 0 & 0 & 0 \\ 0 & \frac{1}{k_2} & 0 & 0 \\ 0 & 0 & \frac{1}{k_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proporcionalno skaliranje dobije se za $k_1=k_2=k_3=k$, no tada se operator proporcionalnog skaliranja može zapisati na još jedan način (osim već prikazanog općeg, pa uvrštavanjem k za sve koeficijente):

$$\underline{\Psi_{psk}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{k} \end{bmatrix}$$

U tom slučaju inverz ovom operatoru je operator:

$$\underline{\Psi_{psk}^{-1}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & k \end{bmatrix}$$

13.5. SMIK

Smik je uzdužna deformacija koja deformira položaj točke i opisuje se kutom deformacije prema svakoj koordinatnoj osi. U 3D prostoru imamo 3 koordinatne osi i tri kuta: α, β i χ.

Operator smika u 3D glasi:

$$\underline{\Psi_{sm}} = \begin{bmatrix} 1 & tg\alpha & tg\alpha & 0 \\ tg\beta & 1 & tg\beta & 0 \\ tg\chi & tg\chi & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Izvođenje inverznog operatora i pronalaženje uvjeta njegove egzistencije ostavljam čitateljima za zabavu.

13.7. PRIMJER

U nastavku ove skripte trebati će nam rotacija oko zadanog središta, pa ću Vas ovdje podsjetiti na primjer koji smo već pokazali kod 2D transformacija. Potrebno je točku T rotirati oko točke S. Naravno, kako u 3D prostoru imamo 3 vrste rotacija, potrebno je zadati i koju rotaciju želimo primijeniti. No jednom kada imamo sve podatke, postupak je slijedeći:

1. translatirati točku S u ishodište koordinatnog sustava
2. izvršiti rotaciju
3. primijeniti inverznu translaciju od translacije iz prvog koraka

14. PROJEKCIJE I TRANSFORMACIJE POGLEDA

14.1. PROJEKCIJE

Područje računalne grafike osim rada u 2D prostoru obuhvaća i rad u 3D prostoru. Štoviše, ljudima je koncepcija 3D prostora daleko bliža nego 2D prostor. Razloga tome je mnogo, a jedan je i taj što živimo u 3D prostoru, i pojmovi poput iznad, ispod, lijevo, desno, ispred ili iza sasvim su nam jasni. No u 2D prostoru ograničeni smo na samo dva smjera. U 2D prostoru ne možemo prikazati tijela; moguć je isključivo prikaz likova. S druge strane, prikazne jedinice danas su još uvijek dvodimenzionalne! Zaslon monitora nudi nam mogućnost prikazivanja u jednoj ravnini. S druge strane, čovjek nastoji i u svijet računala uvesti 3D prostor. I dakako, to je jednim dijelom moguće. Sjetimo se samo fotoaparata. Kada slikamo, slikamo objekte u 3D prostoru. Kao rezultat slikanja dobivamo sliku, dakle komad papira na kojemu je "slika" smještena u ravninu – u 2D prostor. Ipak, pogledom na sliku dobivamo jasnu predodžbu o onome što je slikano; imamo privid 3D prostora. Dakako, u tom 3D prostoru ne možemo pogledati kako bi objekti izgledali kada bismo ih pogledali malo desnije, ili pak kada bismo otišli iza njih. Dobivena slika jednostavno je zamrznut prikaz onoga što smo vidjeli točno sa mjesta s kojeg smo gledali i točno u smjeru u kojem smo gledali. I to je mjesto na koje uskače računalna grafika. Tu se dakle pruža mogućnost da i na računalu kreiramo takve "snimke" koje nam pokazuju što bismo vidjeli od 3D prostora kada bismo stajali u jednoj točci tog prostora i gledali prema nekoj drugoj točci.

Metode koje opisuju na koji način "gledamo" i što bismo zapravo vidjeli zovu se – **projekcije**. Naziv "projekcije" dolazi od činjenice da objekte 3D prostora "projiciramo" na ravninu – u 2D prostor. Projekcije su matematički modeli koji nam govore na koji način treba točke 3D prostora preslikati u ravninu u točke 2D prostora. Postoji više vrsta projekcija, a mi ćemo u nastavku obraditi dvije:

- Paralelna projekcija
- Perspektivna projekcija

Evo jednostavnog primjera što znači projicirati objekte 3D prostora u 2D prostor. Uzmimo list papira i stavimo ga na stol. Iznad njega (ali ne na njega) postavimo nekakav objekt, npr. olovku, i iznad postavimo uključenu svjetiljku. Rezultat je sjena olovke na papiru; 3D objekt preslikao se je u ravninu.

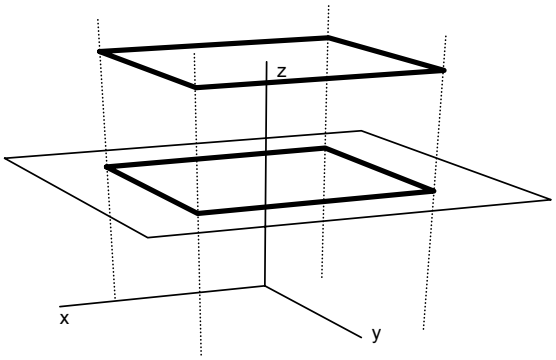
Ovaj jednostavan pokus pokazao nam je još nešto – projekcije su destruktivne! U 3D prostoru znamo i duljinu olovke, i debljinu olovke, promjer i sl. Projiciranjem u 2D prostor dobili smo sliku iz koje više ne možemo doznati sve te informacije.

14.1.1. PARALELNA PROJEKCIJA

Jedan od najjednostavnijih modela projekcija jest model paralelne projekcije. Taj model podrazumijeva da su sve zrake svijetlosti koje obasjavaju objekt okomite na ravninu projiciranja i samo takve zrake tvore sliku. Drugim riječima, projekcija daje sliku koju bi dao točkasti izvor iznad ravnine projiciranja i to na beskonačnoj udaljenosti; tada bi sve zrake koje

dolaze do objekata iznad ravnine bile paralelne, i okomite na samu ravninu. Npr. projiciranjem objekta širokog 10 cm dobili bismo i sliku široku točno 10 cm.

Za opisivanje projekcije poslužiti ćemo se slijedećim primjerom. Želimo dobiti paralelnu projekciju točaka na ravninu z=H pri čemu je H je proizvoljan broj (to je dakle projekcija u xy ravninu na visini H). Slika 14.1.1.1. prikazuje problem.



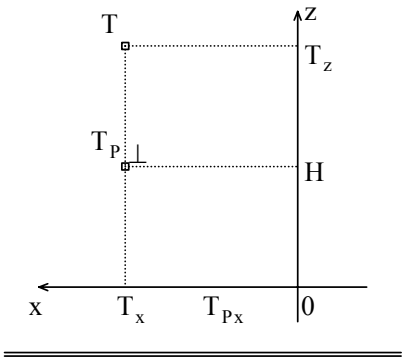
Slika 14.1.1.1.

Na slici je prikazan lik koji se projicira, ispod njega ravnina projekcije sa likom koji se dobije projiciranjem, te karakteristične zrake projiciranja kroz sva četiri vrha lika. Zrake su okomite na ravninu projekcije.

Ovaj model uzet je zbog jednostavnosti, da bismo se lakšu upoznali sa idejom paralelne projekcije. Rješenje zadanog problema je jednostavno. Ako proizvoljnu točku T projiciramo, dobiti ćemo točku T_P i pri tome će za komponente točke T_P vrijediti:

$$T_{Px} = T_x$$
$$T_{Py} = T_y$$
$$T_{Pz} = H$$

To se jasno vidi sa slike 14.1.1.2. Na slici je prikazana situacija za x-komponentu, no isto vrijedi i za y-komponentu. Projicira se točka T, a projekcija je točka T_P.



14.1.1.2.

Sve točke koje ćemo na ovaj način projicirati, imati će z koordinatu jednaku H; dakle, sve će ležati u ravnini z=H. Ovdje prikazano paralelno projiciranje može se opisati matricom paralelne projekcije:

$$\boldsymbol{\pi}_{\mathbf{par}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & H & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Tada se projekcija opisuje umnoškom točke i matrice projekcije:

$$T_{Ph} = \begin{bmatrix} T_{Ph,x} & T_{Ph,y} & T_{Ph,z} & T_{Ph,h} \end{bmatrix} = T_h \cdot \boldsymbol{\pi}_{\mathbf{par}} =$$

$$= \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,z} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & H & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & H & 1 \end{bmatrix}$$

Kada dobivene točke prikazujemo u ravnini, tada kao x i y koordinatu koristimo prve dvije komponente točke, dok ostale komponente zanemarujemo.

Prilikom crtanja točaka u ravnini javlja se i jedan jednostavan problem: dvije točke 3D prostora preslikavaju se u istu točku ravnine. Koju točku tada prikazati u ravnini? Npr. ako se želi prikazati projekcija prve točke, tada ćemo je prikazati crvenom bojom, a ako se želi prikazati projekcija druge točke, tada ćemo je prikazati plavom bojom. Kojom bojom prikazati točku? Da bismo odgovorili na ovo pitanje, treba se prisjetiti koja je zapravo svrha projekcije? Želimo "vidjeti" 3D prostor na zaslonu! Ako je to tako, tada se dvije različite točke 3D prostora koje se preslikaju u istu točku 2D prostora mogu tumačiti kao dva objekta jedan iza drugoga! Pa koji ćemo objekt vidjeti? Naravno, onaj koji je bliži samoj ravnini projekcije! U tom slučaju ćemo boju odabrati prema onoj točki koja je bliža ravnini projekcije. No kako modificirati matricu projekcije tako da sačuva udaljenost od ravnine projekcije? Iskoristiti ćemo z-koordinatu projicirane točke za pohranu udaljenosti točke koju smo projicirali od same ravnine. Ako to učinimo, tada vrijedi:

$$T_{Px} = T_x$$
$$T_{Py} = T_y$$
$$T_{Pz} = T_z - H$$

dok se matrica paralelne projekcije koja čuva udaljenost točke od ravnine modificira u:

$$\boldsymbol{\pi}'_{\mathbf{par}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1-H & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

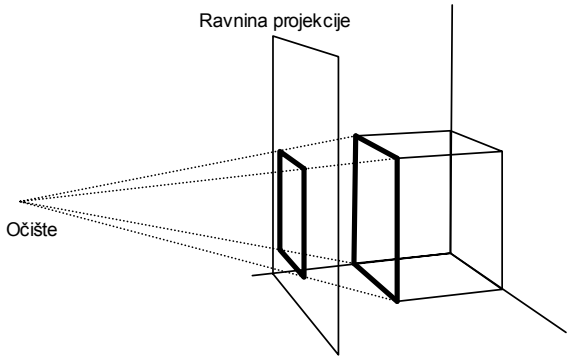
Rezultat koji se dobije množenjem točke koju projiciramo i ove matrice treba tumačiti ovako:

- Prve dvije komponente odgovaraju komponentama točke u ravnini
- Z koordinata točke, budući da točka pripada ravnini z=H iznosi upravo H
- Treća komponenta ravnine odgovara udaljenosti točke koju smo projicirali od ravnine projekcije, i može poslužiti kao kriterij za određivanje koju točku treba prikazati u slučaju da se više točaka projicira u istu točku ravnine

Matrica paralelne projekcije ispala je ovako jednostavna zbog toga što smo odabrali vrlo jednostavan primjer projekcije: projekciju na ravninu z=H. Razumno bi bilo pitati se a kako bi izgledala matrica paralelne projekcije na proizvoljnu ravninu u 3D prostoru. No odgovor na ovo glasi: takvu matricu (na svu sreću) ne moramo tražiti jer bi ispala krajnje komplicirana i nepregledna. Umjesto toga, primijeniti ćemo postupak transformacije pogleda o kojem će biti više riječi u nastavku, i zatim iskoristiti upravo ovu jednostavnu matricu paralelne projekcije. No prije no što se upustimo u postupak transformacija pogleda, pogledajmo još i drugi tip projekcije.

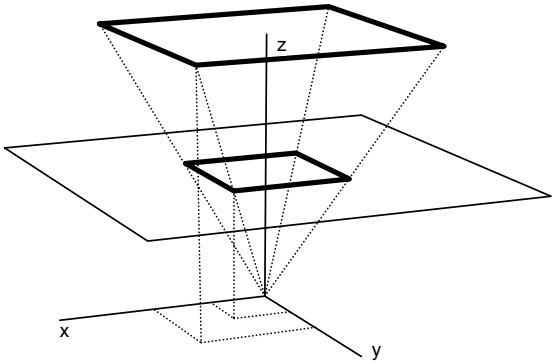
14.1.2. PERSPEKTIVNA PROJEKCIJA

Perspektivna projekcija je model koji je čovjeku bliži. Naime, model uvodi dvije točke: očište i gledište. Očište je točka u koju smještamo oko, dok gledište predstavlja točku oko koje se stvara ravnina projekcije (gledište pripada toj ravnini). Ravnina projekcije koja se stvara ujedno je okomita na vektor očište-gledište. Projekcija proizvoljne točke T dobije se tako da se očište spoji pravce sa zadanom točkom T, i kao projekcija se uzima točka u kojoj dobiveni pravac siječe (odnosno probada) ravninu projekcije. Jedan općeniti primjer perspektivne projekcije prikazan je na slici 14.1.2.1.



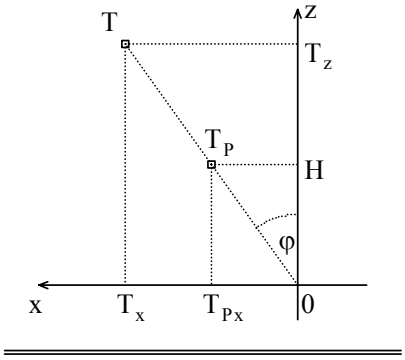
Slika 14.1.2.1.

Za matematičku analizu problema pokušajmo odrediti perspektivnu projekciju proizvoljne točke T, ukoliko očište stavimo u ishodište koordinatnog sustava, a gledište na z-os na visini z=H. Ovakvim odabirom očišta i gledišta postiže se da je ravnina projekcije upravo ravnina z=H (dakle, xy ravnina podignuta od ishodišta za H). Slika 14.1.2.2. pokazuje perspektivnu projekciju lika uz zadano očište i gledište:



Slika 14.1.2.2.

Pogledajmo što se događa sa pojedinim točkama. Jedan vrh lika i njegova projekcija sa slike 14.1.2.2. spojnicama su spojeni te su na x i y koordinatnim osima označene njihove koordinate (ili ako se osvrnemo na prethodno poglavlje, učinjene su njihove paralelne projekcije na z=0 ravninu). Analizu treba provesti za svaku koordinatu zasebno, no kako je rezultat identičan, u nastavku je na slici 14.1.2.3. prikazan slučaj za x-koordinatu.



14.1.2.3.

Trokuti 0-H-T_P i 0-T_z-T su slični trokuti jer dijele jednak kut φ. Tada vrijedi:

$$\frac{T_{Px}}{H} = \frac{T_x}{T_z}$$

odakle slijedi:

$$T_{Px} = T_x \cdot \frac{H}{T_z}$$

Slična analiza daje i y koordinatu projekcije, te se dobije:

$$T_{Px} = T_x \cdot \frac{H}{T_z} \qquad T_{Py} = T_y \cdot \frac{H}{T_z}$$

Ovdje izvedeni odnosi mogu se zapisati i matrično, ukoliko za sve točke iskoristimo njihove homogene inačice. Dobije se:

$$\boldsymbol{\pi}_{\mathbf{per.proj}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

U točnost se možemo uvjeriti pomnožimo li točku i matricu projekcije:

$$T_{Ph} = T_h \cdot \boldsymbol{\pi}_{\mathbf{per.proj}} = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,z} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} T_{h,x} & T_{h,y} & 0 & \frac{T_{h,z}}{H} \end{bmatrix}$$

Prelaskom u radni prostor nakon dijeljenja sa homogenim parametrom dobiju se upravo izrazi od kojih smo krenuli.

Z koordinata točke koja se dobije perspektivnom projekcijom preko prethodno izvedene matrice iznosi 0. Naime, kako točka leži u ravnini projekcije logično je za z-koordinatu uzeti vrijednost nula. No ponekad projekciju nećemo koristiti kao prijelaz iz 3D u 2D sustav. U tom slučaju potrebno je projekciji svake točke ostaviti sve tri koordinate ispravnima: znači ako projiciramo na ravninu z=H, tada sve projicirane točke imaju z koordinatu jednaku H. Matrica koja će vršiti ovakvo projiciranje dobije se modifikacijom prethodne matrice:

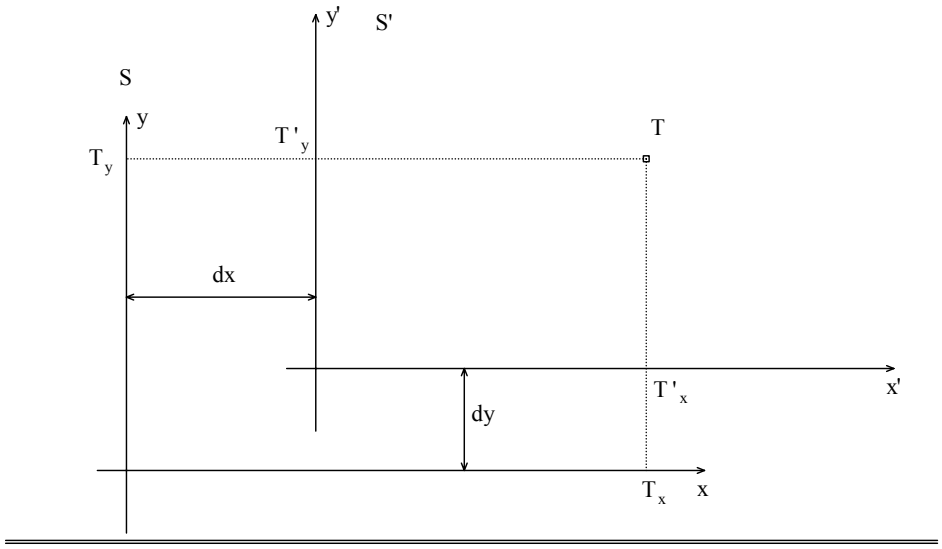
$$\boldsymbol{\pi}'_{\mathbf{per.proj}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Kao i kod paralelne projekcije, i ovdje ćemo se zaustaviti i nećemo ići u kompliciranije slučajeve, odnosno u opći slučaj (što bismo vidjeli kada bi očište bilo negdje, a gledište negdje drugdje). Razlog tome je kompliciranost cijelog postupka ukoliko bismo išli ovako direktno, grubom silom. Umjesto toga, dovoljno je reći da se takav opći slučaj može postupkom transformacije pogleda svesti upravo na ovaj jednostavan! I stoga, pogledajmo što nam nudi transformacija pogleda…

14.2. TRANSFORMACIJE POGLEDA

Transformacije pogleda su postupci kojima se točke iz jednog koordinatnog sustava preslikavaju u drugi koordinatni sustav. Ideja je, kao i uvijek, krajnje jednostavna. Treba pronaći matricu (4x4) takvu da kada svoju proizvoljnu točku pomnožite tom matricom, dobijete točku sa koordinatama koje bi vaša točka imala kada biste je promatrali iz nekog drugog sustava. Postoji nekoliko koraka koje treba proći na putu do te matrice. Pa krenimo kroz primjere.

U koordinatnom sustavu S imamo točku T. Koje bi koordinate ta točka imala u sustavu S' koji bismo dobili kada bismo sustav S lagano trknuli tako da mu ishodište otklizi u točku O'? Problem je prikazan na slici:



14.2.1.

Problem je radi jednostavnost prikazan u 2D prostoru. Sa slike 14.2.1. jasno se vidi veza između koordinata u oba sustava:

$$\left.\begin{aligned} T_x &= T_x^{'} + dx \\ T_y &= T_y^{'} + dy \end{aligned}\right\} \begin{aligned} T_x^{'} &= T_x - dx \\ T_y^{'} &= T_y - dy \end{aligned}$$

Proširenjem na koordinatni sustav u 3D prostoru dobije se:

$T_x^{'} = T_x - dx$

$T_y^{'} = T_y - dy$

$T_z^{'} = T_z - dz$

Upotrijebimo li homogeni zapis točaka, gornju transformaciju možemo prikazati matričnim umnoškom točke i matrice:

$$\mathbf{\Theta_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -dx & -dy & -dz & 1 \end{bmatrix}$$

Ova matrica naziva se **matricom translacije** i čini prvu od mogućih transformacija pogleda. Dokaz da ova matrica obavlja željenu transformaciju je jednostavan:

$$T_h^{'} = T_h \cdot \Theta_1 = \begin{bmatrix} T_{h,x} & T_{h,y} & T_{h,z} & T_{h,h} \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -dx & -dy & -dz & 1 \end{bmatrix} =$$

$$\begin{bmatrix} T_{h,x} - dx \cdot T_{h,h} & T_{h,y} - dy \cdot T_{h,h} & T_{h,z} - dz \cdot T_{h,h} & T_{h,h} \end{bmatrix}$$

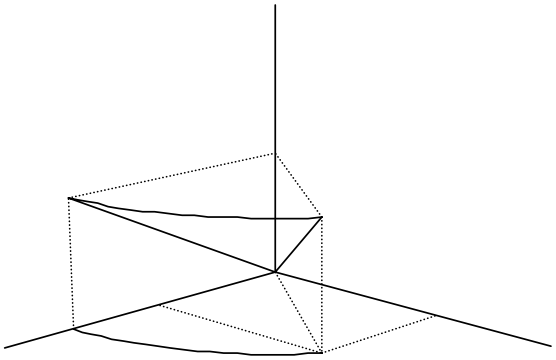
Prelaskom u radni prostor dobije se:

$$T_x^{'} = \frac{T_{h,x}^{'}}{T_{h,h}^{'}} = \frac{T_{h,x} - dx \cdot T_{h,h}}{T_{h,h}} = \frac{T_{h,x}}{T_{h,h}} - dx = T_x - dx$$

$$T_y^{'} = \frac{T_{h,y}^{'}}{T_{h,h}^{'}} = \frac{T_{h,y} - dy \cdot T_{h,h}}{T_{h,h}} = \frac{T_{h,y}}{T_{h,h}} - dy = T_y - dy$$

$$T_z^{'} = \frac{T_{h,z}^{'}}{T_{h,h}^{'}} = \frac{T_{h,z} - dz \cdot T_{h,h}}{T_{h,h}} = \frac{T_{h,z}}{T_{h,h}} - dz = T_z - dz$$

Ovim razmatranjem naučili smo kako preslikati koordinate iz jednog koordinatnog sustava u drugi, ukoliko je ishodište drugog samo pomaknuto za neki vektor (dx dy dz). No što ukoliko je drugi koordinatni sustav umjesto pomaka uslijed našeg udarca iz gornjeg primjera doživio rotaciju? Radi jednostavnosti pretpostaviti ćemo da mu je ishodište bilo učvršćeno u referentni sustav pa nije moglo doći do pomaka, već samo do rotacije (ovaj rotirani sustav označiti ćemo sa S'). Kako će tada izgledati koordinate zadane točke T u tom sustavu? U 2D sustavu svaka rotacija jednoznačno je određena kutom rotacije – jednim kutom. U 3D prostoru opis rotacije zahtjeva uporabu prostornog kuta – kuta koji se može u pravokutnom koordinatnom sustavu opisati pomoću dvije komponente kuta: kutom koji projekcija rotirane točke u xy ravninu čini sa x osi koordinatnog sustava, te kutom koji projekcija rotirane točke u xz ravninu čini sa z osi koordinatnog sustava (zapravo, ovo nije jedina mogućnost; mogu se koristi bilo koja dva para ravnina i u njima odgovarajući kutovi). No unatoč ovako složenom opisu prostornog kuta, ideja rješenja je ista kao i kod prethodnog problema. U prethodnom slučaju rješenje smo dobili tako da smo pogledali što trebamo učiniti da bismo oba koordinatna sustava ponovno poklopili jedan preko drugoga; rješenje je bilo ishodištu novog sustava oduzeti vektor pomaka i time su se sustavi poklopili. No u ovom primjeru ishodišta se već poklapaju. Ono što se ne poklapa su osi. Ono što ćemo pokušati da bismo poklopili naše sustave jest slijedeće: uhvatiti ćemo z-os novog sustava, i zarotirati je tako da se poklopi sa z-osi starog sustava. No kako je za ovo potrebno primijeniti prostornu rotaciju, razložiti ćemo problem na dva dijela. Najprije ćemo z-os zarotirati kutom u xy-ravnini u smjeru kazaljke na satu (dakle u matematički negativnom smjeru) tako da padne u xz ravninu. Slijedeća slika prikazuje postupak:



Uzmimo da se točka $G=(G_x,G_y,G_z)$ nalazi na z-osi zarotiranog S' sustava. Kut α što ga projekcija te točke u xy ravninu zatvara sa x osi određen je:

$$\cos\alpha = \frac{G_x}{\sqrt{G_x^2 + G_y^2}} \qquad \sin\alpha = \frac{G_y}{\sqrt{G_x^2 + G_y^2}}$$

Rotiranjem za taj kut α točka G preslikala se je u točku G' koja leži u xz ravnini. Pri tome su komponente točke G' određene izrazima:

$$G_x^{'} = \sqrt{G_x^2 + G_y^2}$$

$$G_y^{'} = 0$$

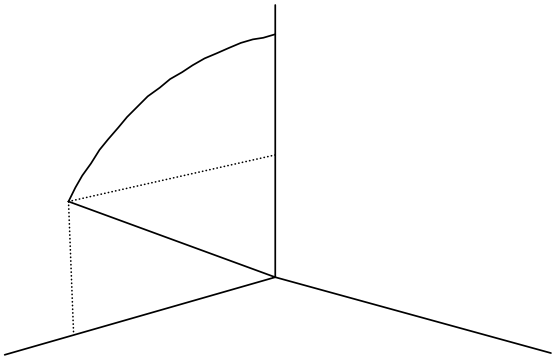
$$G_z^{'} = G_z$$

Rotacijom u xy ravnini z-koordinata točke ostala je očuvana. Y-koordinata pala je na nulu jer je točka rotacijom stigla u xz razninu. Ova transformacija opisuje se matricom:

$$\mathbf{\Theta_2} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ovdje treba spomenuti i specijalni slučaj koji može nastupiti. Ako je $G_x=0$ i $G_y=0$, tada se ova rotacija preskače, jer su osi obaju sustava već kolinearne, te kut α nije definiran! No krenimo dalje.

Zajedno sa točkom G koja se je preslikala u G', z-os našeg rotiranog sustava također je pala u xz ravninu referentnog sustava. Nova situacija prikazana je na slici 14.2.3.



Još nam je preostalo da točku G' odvučemo na z-os referentnog koordinatnog sustava rotacijom u xz ravnini opet u smjeru kazaljke na satu. Kut β koji predstavlja kut u xz ravnini što ga točka G' (odnosno precizno govoreći njezina projekcija na xz ravninu) zatvara sa z-osi određen je:

$$\cos\beta = \frac{G_z'}{\sqrt{(G_x')^2 + (G_z')^2}} \qquad \sin\beta = \frac{G_x'}{\sqrt{(G_x')^2 + (G_z')^2}}$$

Rotacijom točke G' dobiva se točka G" koja leži i na z-osi referentnog sustava, te za njezine komponente vrijedi:

$$G_x'' = 0$$
$$G_y'' = 0$$
$$G_z'' = \sqrt{(G_x')^2 + (G_z')^2} = \sqrt{G_x^2 + G_y^2 + G_z^2}$$

I ova se transformacija može opisati matrično:

$$\mathbf{\Theta_3} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ovim postupkom postigli smo poklapanje z osi referentnog sustava sa rotiranim sustavom. Pri izvođenju izraza za sin(β) i cos(β) specijalnog slučaja nema. Naime, pogreška bi nastupila ukoliko bi G'_x i G'_z bili istodobno jednaki nuli, no to se ne može dogoditi osim u slučaju da se je polazna točka G poklapala sa ishodištem (što ne smije jer tada nismo definirali z-os).

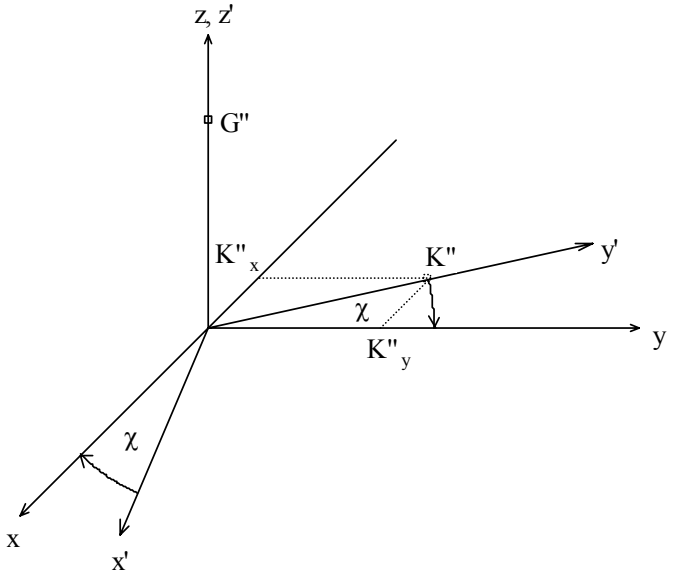
Rotacijama $\mathbf{\Theta_2}$ i $\mathbf{\Theta_3}$ postigli smo ispravno poklapanje z-osi referentnog sustava i z-osi zarotiranog sustava. No da li smo tim rotacijama uspjeli poklopiti i x i y osi? Generalno govoreći, i jesmo i nismo. Odgovor "jesmo" možemo pravdati ovako: rotirani sustav zadali smo samo jednom točkom – točkom na z-osi. Zbog toga smo položaj x i y osi rotiranog sustava ostavili nedefinirane! Zatim smo rotacijama postigli poklapanje z-osi oba sustava. I što je sa x i y osima? Pa x i y osi rotiranog sustava bile su baš takve, da se sada upravo poklapaju sa x i y osima referentnog sustava! Naravno, ovo obrazloženje nije baš najprihvatljivije, no istinito je tako dugo dok rotirani sustav ne definiramo malo bolje… Tu ulazi u igru odgovor "nismo". Trodimenzionalni sustav jednoznačno je zadan ukoliko je poznato npr. ishodište, točka na z-osi i točka na y-osi, te je sustav desni. Točku ishodišta

znamo – to je ista točka koja je i ishodište referentnog sustava. Točku na z-osi isto znamo: zadali smo je kao točku G. Da bismo sustav definirali do kraja, zadajmo još i točku K koja leži na y-osi sustava, i riješimo problem do kraja.

Na sustav S' do ovog trena već smo djelovali rotacijama Θ_2 i Θ_3. Točka K uslijed tih rotacija prešla je u točku K":

$$K_h'' = K_h \cdot \Theta_2 \cdot \Theta_3$$

pri čemu je K_h homogena inačica točke K. Uslijed obavljenih rotacija z-komponenta točke K" postala je nulom, dok su x i y komponente općenito različite od nule (barem jedna od njih). Situaciju prikazuje slijedeća slika:



14.2.4.

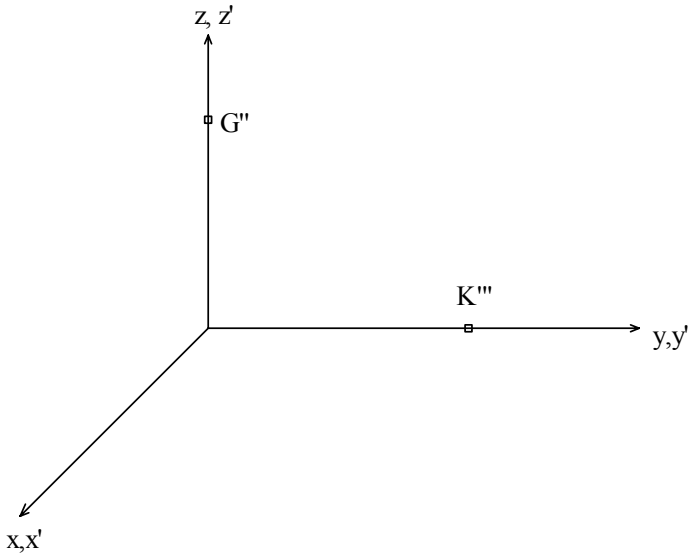
Kut χ koji određuje nepoklapanje između y i y' osi definiran je:

$$\cos\chi = \frac{K_y''}{\sqrt{(K_y'')^2 + (K_x'')^2}} \qquad \sin\chi = \frac{-K_x''}{\sqrt{(K_y'')^2 + (K_x'')^2}}$$

Rotiranjem sustava S' oko osi z za kut χ postići ćemo potpuno poklapanje referentnog sustava i sustava S'. Matrica kojom se izvodi rotacija oko z osi za kut χ glasi:

$$\Theta_4 = \begin{bmatrix} \cos\chi & -\sin\chi & 0 & 0 \\ \sin\chi & \cos\chi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rezultat ove rotacije prikazan je na slici 14.2.5.



14.2.5.

Ukoliko se ipak odlučimo na nezadavanje točke K, može se uzeti jednostavna pretpostavka: kut χ iznosi 90°. Tada će matrica rotacije oko z-osi izgledati:

$$\Theta'_4 = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pogledamo li koje smo sve transformacije primijenili, vidjeti ćemo da smo iskoristili transformacije Θ_2, Θ_3 i Θ_4 ili Θ'_4. To znači da se cijeli posao može obaviti jednom transformacijom $\Theta_2*\Theta_3*\Theta_4$ (odnosno $\Theta_2*\Theta_3*\Theta'_4$). Dakle, ukoliko imamo zadane koordinate točke T u referentnom sustavu, i imamo zadanu jednu točku G koja leži na z-osi našeg rotiranog sustava, te točku K koja se nalazi na y-osi našeg rotiranog sustava, tada ćemo koordinate točke T u rotiranom sustavu S' dobiti množenjem točke T i transformacijskih matrica.

Još nam je preostalo odgovoriti na najopćenitiji slučaj: kako bi glasile koordinate točke T u sustavu koji je uslijed "udarca" pretrpio i rotaciju, i translaciju? Tada mu se niti ishodišta, niti osi više ne poklapaju. Odgovor na ovo pitanje dati će nam kompozicija prethodnih slučajeva: prvo treba ishodište rotiranog sustava vratiti u ishodište referentnog sustava (matricom Θ_1), a zatim sustav treba još dodatno zarotirati matricom $\Theta_2*\Theta_3*\Theta_4$. Pri tome treba obratiti pažnju na točke koje će se uzeti kao G točka i K točka za određivanje Θ_2, Θ_3 i Θ_4. Naime, kada matricom Θ_1 ishodište rotiranog sustava vratimo u ishodište referentnog sustava, tada i izvorne točke G i K treba translatirati istom matricom Θ_1 i tako dobivene točke treba uzeti za određivanje matrica Θ_2, Θ_3 i Θ_4.

Sa slike 14.2.5. vidljivo je da se orijentacija sustava prilikom tranformacija ne mijenja: polazni sustav bio je desni – završni sustav također je desni. Postoji transformacija pogleda koja nam omogućava i tu promjenu. Ukoliko je referentni sustav lijevi, a odredišni sustav desni (ili obrnuto), tada će se koordinate između sustava transformirati jednostavnim okretanjem predznaka na x osi što čini matrica:

$$\Theta_5 = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ovime završavamo pregled transformacija pogleda.

14.3. PRIMJENA NA POOPĆENJE PERSPEKTIVNE PROJEKCIJE

U poglavlju 14.1.2. pokazali smo osnove perspektivne projekcije. Uveli smo točku očišta kao onu točku gdje stavljamo naše oko, te točku gledišta kao onu točku koja predstavlja točku koja pripada ravnini projekcije, i koja ujedno tvori ishodište 2D koordinatnog sustava u toj ravnini. Dodatno smo rekli da je ravnina projekcije određena jednom pripadnom točkom (gledištem) i normalom (vektor očište-gledište). No tu smo stali. Kao primjer perspektivne projekcije uzeli smo najjednostavniji mogući: očište je u ishodištu koordinatnog sustava, a gledište je na z os i to točka (0 0 H) gdje je H udaljenost očišta od gledišta.

Da bismo poopćili problem, sada ćemo razmotriti situaciju kada su očište i gledište proizvoljne točke u prostoru. Kako tada naći perspektivnu projekciju? Pogledajmo još jednom što znamo. Znamo naći perspektivnu projekciju ako se očište nalazi u ishodištu, a gledište na z osi. Hm… A koje bi bile koordinate proizvoljne točke T ako mjerenje vršimo iz koordinatnog sustava koji ima ishodište u očištu a gledište mu se nalazi na z-osi? Da, riječ je o transformaciji pogleda… Pa krenimo redom.

Očište ćemo u nastavku označavati sa Q, gledište sa R.

14.3.1. KORAK 1

Prisjetimo li se transformacije pogleda, tada je prvi korak vraćanje novog ishodišta u ishodište referentnog sustava. Dakle, koristimo matricu Θ_1, uz pomake koji odgovaraju upravo koordinatama očišta. Transformacije ćemo raditi nad proizvoljnom točkom T.

$$\mathbf{\Theta_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Q_x & -Q_y & -Q_z & 1 \end{bmatrix}$$

Ova transformacija očište će preslikati u ishodište, dok će gledište preslikati u:

$$R_h^{'} = R_h \cdot \mathbf{\Theta_1} = \begin{bmatrix} R_{h,x} & R_{h,y} & R_{h,z} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Q_x & -Q_y & -Q_z & 1 \end{bmatrix} =$$

$$\begin{bmatrix} R_{h,x} - Q_x & R_{h,y} - Q_y & R_{h,z} - Q_z & 1 \end{bmatrix}$$

Prema tome, za pojedine komponente vrijedi:

$R_x^{'} = R_x - Q_x$

$R_y^{'} = R_y - Q_y$

$R_z^{'} = R_z - Q_z$

14.3.2. KORAK 2

Sada kada se ishodišta obaju sustava poklapaju, treba izvršiti rotaciju sustava tako da im se z osi poklope. Prvi korak je bio rotacija u xy ravnini matricom $\mathbf{\Theta_2}$.

$$\mathbf{\Theta_2} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

pri čemu vrijedi:

$$\cos\alpha = \frac{G_x}{\sqrt{G_x^2 + G_y^2}} \qquad \sin\alpha = \frac{G_y}{\sqrt{G_x^2 + G_y^2}}$$

dok je G točka upravo jednaka točci gledišta nakon prve transformacije: G=R', pa se može pisati:

$$\cos\alpha = \frac{R'_x}{\sqrt{R'^2_x + R'^2_y}} \qquad \sin\alpha = \frac{R'_y}{\sqrt{R'^2_x + R'^2_y}}$$

Transformacija $\mathbf{\Theta_2}$ djelovanjem na točku R' daje točku R":

$$R''_h = R'_h \cdot \Theta_2 = \begin{bmatrix} R'_{h,x} & R'_{h,y} & R'_{h,z} & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} R'_{h,x} \cdot \cos\alpha + R'_{h,y} \cdot \sin\alpha & -R'_{h,x} \cdot \sin\alpha + R'_{h,y} \cdot \cos\alpha & R'_{h,z} & 1 \end{bmatrix}$$

Uvrštavanjem izraza za sin(α) i cos(α) dobiva se:

$$R''_x = \sqrt{R'^2_x + R'^2_y}$$

$$R''_y = 0$$

$$R''_z = R'_z$$

14.3.3. KORAK 3

Sada je potrebno primijeniti i rotaciju u xz ravnini transformacijom Θ_3.

$$\Theta_3 = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Pri čemu su:

$$\cos\beta = \frac{R''_z}{\sqrt{\left(R''_x\right)^2 + \left(R''_z\right)^2}} \qquad sin\beta = \frac{R''_x}{\sqrt{\left(R''_x\right)^2 + \left(R''_z\right)^2}}$$

Transformacija Θ_3 djelovanjem na točku R" daje točku R'", što po komponentama daje:

$$R'''_x = 0$$

$$R'''_y = 0$$

$$R'''_z = \sqrt{\left(R''_x\right)^2 + \left(R''_z\right)^2} = \sqrt{R'^2_x + R'^2_y + R'^2_z}$$

Primijenimo li još i transformaciju Θ'_4 (a po potrebi i Θ_5 ako su sustavi različito orijentirani), učinili smo sve potrebne transformacije kako bi sustav sa ishodištem u očištu i točkom gledišta na njegovoj z osi preveli u sustav kojemu se ishodište i z os poklapaju sa referentnim sustavom. Što smo time dobili? Ako znamo točku T u referentnom sustavu, tada su njezine koordinate u sustavu sa ishodištem u očištu jednake:

$$T' = T \cdot \Theta_1 \cdot \Theta_2 \cdot \Theta_3 \cdot \Theta'_4 \cdot \Theta_5$$

Budući da su nam time poznate koordinate u sustavu u kojem je očište jednako ishodištu, a gledište se nalazi na z-osi, tada točku T' znamo onom jednostavnom matricom perspektivno projicirati, te možemo pisati:

$$T_p^{'} = T'\cdot\boldsymbol{\pi}_{\mathbf{per.proj}} = T\cdot\boldsymbol{\Theta}_1\cdot\boldsymbol{\Theta}_2\cdot\boldsymbol{\Theta}_3\cdot\boldsymbol{\Theta}_4^{'}\cdot\boldsymbol{\Theta}_5\cdot\boldsymbol{\pi}_{\mathbf{per.proj}}$$

pri čemu je matrica perspektivne projekcije dana sa:

$$\boldsymbol{\pi}_{\mathbf{per.proj}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{H} \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

H je udaljenost od očišta do gledišta pa se može izračunati iz tih podataka, ili se može primijetiti da smo tu udaljenost već izračunali prilikom transformacije pogleda, te se udaljenost nalazi kao z komponenta točke R'''$_z$. Dakle, za H vrijedi:

$$H = \sqrt{(Q_x - R_x)^2 + (Q_y - R_y)^2 + (Q_z - R_z)^2}$$

$$H = \sqrt{(R_x^{'})^2 + (R_y^{'})^2 + (R_z^{'})^2}$$

$$H = \sqrt{(R_x^{''})^2 + (R_z^{''})^2}$$

$$H = \sqrt{(R_z^{'''})^2} = R_z^{'''}$$

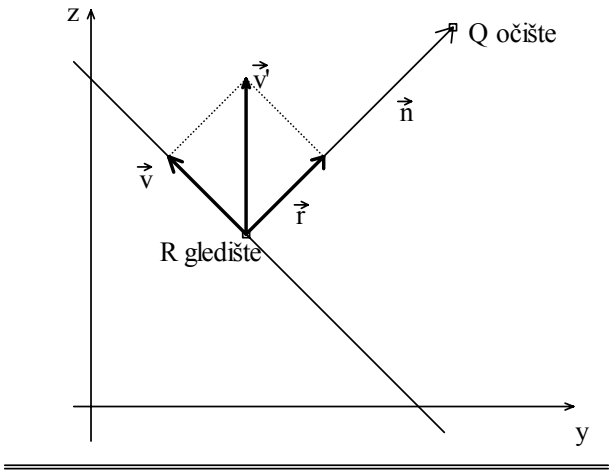
Da smo željeli dobiti opću paralelnu projekciju, postupak bi bio identičan, samo bismo umjesto posljednje matrice perspektivne projekcije uzeli matricu paralelne projekcije.

U postupku se koristi matrica $\boldsymbol{\Theta'_4}$ budući da prilikom zadavanja parametara za perspektivnu projekciju nismo zadali niti jedan podatak koji bi nam omogućio identificiranje položaja osi y. Ukoliko se želi potpuna kontrola nad slikom koju generira perspektivna projekcija, tada je potrebno zadati još i jednu točku K koja se nalazi na y-osi sustava u kojem radimo projekciju. Podsjetimo se još jednom zašto je to tako.

Kod opće perspektivne projekcije zadajemo dvije točke: očište Q i gledište R. Pri tome se stvara novi sustav sa ishodištem u točci Q, i sa pozitivnom z'-osi kroz točku R. Okomito na spojnicu R-Q u kroz točku R stvara se ravnina projekcije. U toj ravnini leže i x' i y' osi ovog novog sustava. Jedini problem u svemu tome je što specificiranjem samo točaka Q i R mi nemamo kontrolu nad x' i y' osima, tj. ne možemo nikako zadati: "ja hoću da y'-os gleda baš ovako…". Jedna od mogućnosti da se dobije kontrola nad tim osima objašnjena je u poglavlju o transformacijama pogleda pomoću točke K. Dakle, ukoliko prilikom zadavanja parametara perspektivne projekcije zadamo i točku K, tada ćemo umjesto matrice $\boldsymbol{\Theta'_4}$ koristiti matricu $\boldsymbol{\Theta_4}$ čime ćemo dobiti kontrolu nad svim osima sustava. Implementacija ove jednostavne modifikacije ostavlja se čitateljima za vježbu. No postoji i drugi pristup.

14.3.4. VIEW-UP VEKTOR

View-up vektor popularan je naziv za jedan od načina "kroćenja" svih osi sustava (dakle osi x' i y'). Evo ideje. Korisnik će prilikom zadavanja parametara npr. perspektivne projekcije zadati i vektor koji će pokazivati smjer y' osi. Ovo može biti dosta problematično jer jedinični vektor y' osi leži u ravnini projekcije, a korisnik prilikom zadavanja "što želi dobiti" ne mora sam računati sve – tome služe računala. Stoga je problem slijedeći: korisnik programa može reći npr. "y' os neka gleda prema gore". U prirodi "prema gore" obično označava u smjeru pozitivne z-osi, čiji je reprezentant npr. vektor $\vec{v}' = [0 \quad 0 \quad 1]$. No (npr.) očište i gledište zadani su tako da ravnina projekcije leži pod kutem od 45° prema ravnini xz, i okomita je na yz ravninu. Vektor $\vec{v}$ koji će u tom slučaju gledati "prema gore" biti će npr. $\vec{v} = [0 \quad -1 \quad 1]$ jer on leži u ravnini projekcije (dok vektor $\vec{v}'$ očito ne leži). Postavlja se pitanje kako uz približni vektor $\vec{v}'$ pronaći pravi vektor $\vec{v}$. Problem je ilustriran na slici 14.3.4.1.



14.3.4.1.

Rješenje za view-up vektor $\vec{v}$ dobiva se jednostavnim računom. Treba uočiti da vrijedi:

$\vec{v}' = \vec{r} + \vec{v}$ (rel: 14.3.4/1)

Vektor $\vec{n}$ definiran je kao razlika točaka:

$\vec{n} = Q - R$ (rel: 14.3.4/2)

Kut između vektora $\vec{n}$ i vektora $\vec{v}'$ definiran je sa:

$$\cos(\vec{n}, \vec{v}') = \frac{\vec{n} \cdot \vec{v}'}{\|\vec{n}\| \cdot \|\vec{v}'\|}$$ (rel: 14.3.4/3)

Za vektor $\vec{r}$ normu možemo dobiti iz pravokutnog trokuta:

$$\|\vec{r}\| = \|\vec{v}'\| \cdot \cos(\vec{r}, \vec{v}') = \frac{\vec{n} \cdot \vec{v}'}{\|\vec{n}\|}$$ (rel: 14.3.4/4)

dok je sam vektor jednak svojoj normi puta jedinični vektor u smjeru $\vec{n}$:

$$\vec{r} = \|\vec{r}\| \cdot \frac{\vec{n}}{\|\vec{n}\|} = \frac{\vec{n} \cdot \vec{v}'}{\|\vec{n}\|^2} \cdot \vec{n} \qquad \text{(rel: 14.3.4/5)}$$

Ako iz relacije (rel: 14.3.4/1) izvučemo vektor $\vec{v}'$ i uvrstimo ono što smo dobili za vektor $\vec{r}$, dobiva se:

$$\vec{v} = \vec{v}' - \vec{r} = \vec{v}' - \frac{\vec{n} \cdot \vec{v}'}{\|\vec{n}\|^2} \cdot \vec{n} \qquad \text{(rel: 14.3.4/6)}$$

Izraz za view-up vektor da se još pojednostavniti ako se vektor $\vec{n}$ prije uporabe normira, pa se umjesto izraza (rel: 14.3.4/2) koristi izraz:

$$\vec{n} = \frac{Q - R}{\|Q - R\|} \qquad \text{(rel: 14.3.4/7)}$$

Tada izraz za view-up vektor daje:

$$\vec{v} = \vec{v}' - \vec{r} = \vec{v}' - \left(\vec{n} \cdot \vec{v}'\right) \cdot \vec{n} \qquad \text{(rel: 14.3.4/8)}$$

Koristeći dobiveni izraz točka K može se izračunati trivijalno:

$$K = R + \vec{v} \qquad \text{(rel: 14.3.4/9)}$$

Nakon izračunavanja točke K može se krenuti u izgradnju matrica za opću perspektivnu projekciju koristeći pri tome matricu Θ_4.

14.4. TRANSFORMACIJE POGLEDA NA DRUGI NAČIN

14.4.1. MALI IZVOD

U poglavlju 14.2. vidjeli smo jedan način prelaska između dva sustava. U nastavku ćemo dati prikaz još jedne mogućnosti. Zanima nas koje će koordinate imati točka T u sustavu S' ako znamo koordinate točke T u referentnom sustavu S. Neka je referentni sustav S naš poznati osnovni sustav sa osima x, y i z koje su u smjeru vektora $\vec{i}$, $\vec{j}$ i $\vec{k}$. Sustav S' možemo zadati na slijedeći način: ishodište mu je u točci C (čije koordinate znamo u sustavu S), pozitivna z-os zadana je vektorom $\vec{N}$, pozitivna y-os zadana je vektorom $\vec{V}$ a pozitivna x-os zadana je vektorom $\vec{U}$. Dakle, imamo situaciju prikazanu na slici 14.4.1.

slika 14.4.1.

Prilikom zadavanja vektora $\vec{N}, \vec{V}$ i $\vec{U}$ treba paziti da su vektori međusobno okomiti, i jedinični (dakle, norma im je 1). Ukoliko zadani vektori nisu jedinični, treba ih prije uporabe normirati! Kako sada doći do koordinata točke T u sustavu S'? Prema već navedenom receptu, najprije treba ishodište sustava S' dovesti u ishodište sustava S. To ćemo postići matricom Θ_1 uz pomak koji odgovara koordinatama ishodišta sustava S' mjereno iz sustava S:

$$\mathbf{\Theta_1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -C_x & -C_y & -C_z & 1 \end{bmatrix}$$

Ovime smo postigli poklapanje ishodišta obaju sustava. Sada još treba očitati koordinate. No prije nego što se upustimo u to, prisjetimo se, što su zapravo koordinate? Hm… Koordinatni sustav zadan je svojim baznim vektorima. Ako od ishodišta krenemo malo po jednom vektoru, pa nakon toga krenemo po drugom vektoru, i tako po svim vektorima, došli smo u neku točku prostora. Koordinate te točke su upravo one "malo po jednom vektoru"-komponente. Točka u koju smo stigli može se predočiti vektorom: projekcije tog vektora na svaki bazni vektor daje upravo odgovor "koliko malo" smo se pomaknuli po tom vektoru! Dakle, svaku koordinatu točke možemo dobiti tako da vektor točke projiciramo na odgovarajući bazni vektor. A projekcija jednog vektora na drugi je upravo njihov skalarni produkt (ukoliko je vektor na koji projiciramo jediničan; inače treba rezultat podijeliti sa njegovom normom). Pogledajmo to na primjeru. U referentnom sustavu zadana je točka T=(5,2,1). Pretvorbom u vektor dobivamo $5\vec{i}+2\vec{j}+1\vec{k}$ jer je referentni sustav upravo zadan vektorima $\vec{i}$, $\vec{j}$ i $\vec{k}$. Koliko iznosi x komponenta točke? Množimo $(5\vec{i}+2\vec{j}+1\vec{k})$ skalarno sa $(1\vec{i}+0\vec{j}+0\vec{k})$ i rezultat je 5! Y komponenta dobije se množenjem $(5\vec{i}+2\vec{j}+1\vec{k})$ sa $(0\vec{i}+1\vec{j}+0\vec{k})$ i rezultat je 2! I z-koordinata dobije se množenjem $(5\vec{i}+2\vec{j}+1\vec{k})$ sa $(0\vec{i}+0\vec{j}+1\vec{k})$ i rezultat je 1! Po ovoj analogiji može se pokazati da komponente točke T u sustavu S' odgovaraju upravo projekcijama vektora T na bazne vektore sustava S' (*matematički dokaz tvrdnje da koordinate u novom sustavu predstavljaju upravo projekcije radij-vektora točke na pojedine bazne vektore novog sustava možete pogledati u dodatku A*). Kako je sustav S' zadan sa tri bazna vektora: $\vec{U}$, $\vec{V}$ i $\vec{N}$ (koji su jedinični), tada vrijedi:

$$T_x' = \vec{T}\cdot\vec{U} = T_x\cdot U_x + T_y\cdot U_y + T_z\cdot U_z$$
$$T_y' = \vec{T}\cdot\vec{V} = T_x\cdot V_x + T_y\cdot V_y + T_z\cdot V_z$$
$$T_z' = \vec{T}\cdot\vec{N} = T_x\cdot N_x + T_y\cdot N_y + T_z\cdot N_z$$

Ovo se može i matrično zapisati, pa matrica $\mathbf{\Theta_2}$ glasi:

$$\mathbf{\Theta_2} = \begin{bmatrix} U_x & V_x & N_x & 0 \\ U_y & V_y & N_y & 0 \\ U_z & V_z & N_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ukupna matrica transformacije pogleda tada glasi: $\mathbf{\Theta}=\mathbf{\Theta_1}\mathbf{\Theta_2}$.

U postupku smo pretpostavili da su zadana sva tri vektora: $\vec{U}$, $\vec{V}$ i $\vec{N}$. No lako je pokazati da ne moraju biti zadani baš svi vektori. Npr. ukoliko su zadani vektori $\vec{V}$ i $\vec{N}$, vektor $\vec{U}$ može se dobiti kao x-produkt vektora $\vec{V}$ i $\vec{N}$ (što će rezultirati desnom orijentacijom sustava S') ili kao x-produkt vektora $\vec{N}$ i $\vec{V}$ (što će rezultirati lijevom orijentacijom sustava S').

Nadalje, vektor $\vec{V}$ mora biti zadan tako da bude okomit na vektore $\vec{U}$ i $\vec{N}$. Ukoliko se korisniku dopusti da zada vektor koji je približno okomit, tada se pravi okomit vektor može izračunati pomoću relacije za view-up vektor (rel: 14.3.4/6) odnosno (rel: 14.3.4/8).

Najslobodniji pristup dopustiti će korisniku zadavanje trodimenzionalne točke C, vektora $\vec{N}$ kao pokazivača pozitivnog smjera osi z', vektor smjera $\vec{V}$' koji i ne mora biti baš okomit na vektor $\vec{N}$ te podatak da li se želi lijevi ili desni sustav S'. U tom slučaju prikladan postupak računanja je slijedeći: prvo treba normirati vektor $\vec{N}$, zatim izračunati pravi vektor $\vec{V}$, te iz podatka o orijentaciji sustava treba izračunati vektor $\vec{U}$ kao x-produkt. Zatim se slože matrice Θ_1 i Θ_2 te ukupna matrica transformacije glasi: $\Theta=\Theta_1\Theta_2$.

14.4.2. PRIMJENA NA PERSPEKTIVNU PROJEKCIJU

Točka C biti će očište. Gledište možemo zadati na dva načina:

1. Eksplicitnim zadavanjem točke gledišta R; tada se vektor $\vec{N}$ računa kao R-C uz naknadno normiranje. Tada se za H može uzeti norma od R-C, ili se točka R može koristiti samo za određivanje vektora $\vec{N}$, dok se H i dalje zadaje proizvoljno.
2. Zadavanjem vektora $\vec{N}$ (koji po potrebi treba normirati), te zadavanjem udaljenosti H (u ovom slučaju H se mora zadati)

Uz ove podatke potrebno je zadati i view-up vektor, te željenu orijentaciju sustava.

Poglavlje 14.4.1. sadrži sve formule potrebne za dobivanje matrice transformacije pogleda ($\Theta=\Theta_1\Theta_2$). Jedino što preostaje je pomnožiti tu matricu sa matricom perspektivne projekcije!

15. OSVJETLJAVANJE

15.1 UVOD

U prethodnim poglavljima naučili smo dovoljno da znamo kako iscrtati žičani model objekta na zaslonu. Sada je došlo vrijeme da se upoznamo sa načinima kako u scenu uvesti svjetlosne izvore i analizirati njihov utjecaj na objekte.

Svjetlost je samo jedan od oblika zračenja, te i za nju vrijede općenite fizikalne zakonitosti:

$$\text{UPADNA SVJETLOST} = \sum \begin{cases} REFLEKTIRANA \\ RASPRŠENA \\ ABSORBIRANA \\ TRANSMITIRANA \end{cases}$$

Svaka od ovih komponenti ovisna je o samim fizikalnim svojstvima materijala, o grubоći površine, o valnoj duljini same svijetlosti i još mnoštvu faktora. Zbog toga je općenita analiza vrlo složena i u praksi neizvediva. Dodatne poteškoće stvara i međusobna interakcija objekata (ukoliko u sceni imamo više od jednog objekta). Naime, dio svjetlosti koji osvjetljava objekt A dolazi do objekta B (bilo refleksijom, bilo raspršenjem i sl.) i osvjetljava ga. Objekt B opet dio te svjetlosti odbija prema objektu A; objekt A opet dio šalje objektu B, i mislim da ste shvatili. Zbog ove kompleksnosti uvode se modeli koji u određenoj mjeri uvažavaju pojedine fizikalne zakone i interakcije objekata, te daju koliko-toliko zadovoljavajuće rezultate. Među najjednostavnije modele spada i Phongov reflekcijski model koji ćemo obraditi u nastavku.

15.2. PHONGOV REFLEKSIJSKI MODEL

15.2.1. OPĆENITO O MODELU

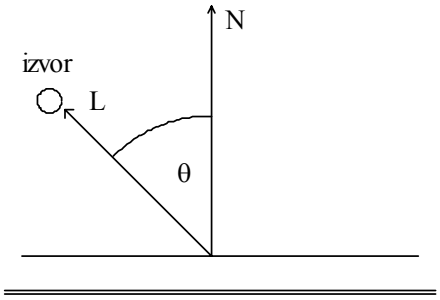
Model pretpostavlja da se cjelokupno osvjetljenje objekta (i svakog njegovog djelića) može opisati kao linearna kombinacija triju komponenata:

- Difuzna komponenta
- Zrcalna komponenta
- Ambijentna komponenta

Uz to, svjetlosni izvori sa kojima ovaj model barata mogu biti isključivo točkasti (dakle, nema rasvjetnih tijela; sva svijetlost dopire iz jedne ili više točaka, pa govorimo o jednom ili više točkastih izvora).

15.2.2. DIFUZNA KOMPONENTA

Iz fizike je poznato da intenzitet svjetlosti koji dopire do bilo koje elementarne površine tijela ovisi o kutu pod kojim upada svjetlost u odnosu na normalu te elementarne površine. Pri tome, prema slici 15.2.2.1. vrijedi:



15.2.2.1.

$$I_d = I_i \cdot k_d \cdot \cos(\theta) \qquad \text{(rel:15.2.2.1/1)}$$

Pri tome je I_i je intenzitet točkastog izvora, a k_d empirijski koeficijent refleksije ovisan o valnoj duljini upadne svijetlosti ($0 \le k_d \le 1$). θ je kut između normale površine i linije od promatrane točke do izvora.

Pogledajmo sliku 15.2.2.1. i na njoj označene vektore. Vektor $\vec{L}$ predstavlja jedinični vektor iz točke koju promatramo i usmjeren je prema izvoru. Vektor $\vec{N}$ jedinični je vektor koji predstavlja normalu u promatranoj točci. Sada kosinus kuta θ možemo izraziti skalarnim produktom tih vektora, pa relacija (rel:15.2.2.1/1) prelazi u:

$$I_d = I_i \cdot k_d \cdot \left(\vec{L} \cdot \vec{N}\right) \qquad \text{(rel:15.2.2.1/2)}$$

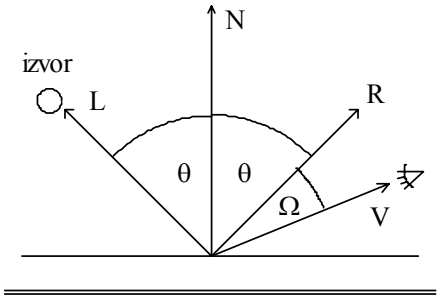
Ako I_d ispadne negativan zbog $\vec{L} \cdot \vec{N} < 0$, tada se za I_d uzima 0.

Ukoliko u sceni ima više točkastih izvora, tada se ukupan intenzitet u promatranoj točci može pisati:

$$I_d = k_d \cdot \sum_n I_{i,n} \cdot \left(\vec{L}_n \cdot \vec{N}\right) \qquad \text{(rel:15.2.2.1/3)}$$

15.2.3. ZRCALNA KOMPONENTA

Zrcalna komponenta, prema slici 15.2.3.1., funkcija je kuta Ω.



15.2.3.1.

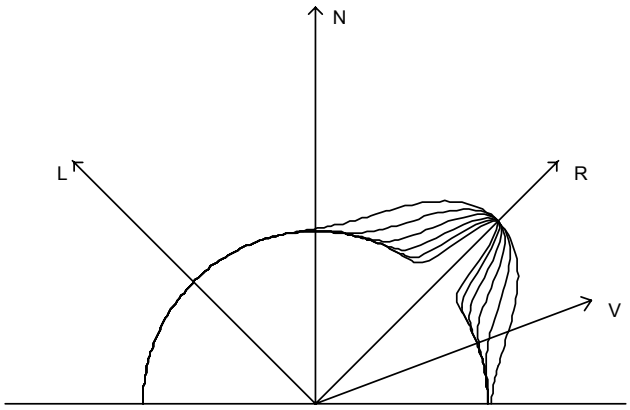
$$I_s = I_i \cdot k_s \cdot \cos^n \Omega \qquad \text{(rel:15.2.3/1)}$$

pri čemu je I_i intenzitet izvora, ks koeficijent ovisan o materijalu ($0 \le k_s \le 1$). n je indeks koji opisuje grubоću površine i njezina reflektirajuća svojstva.

Prema slici 15.2.3.1. vektor $\vec{R}$ predstavlja vektor reflektirane zrake i nalazi se u ravnini koju tvore vektori $\vec{L}$ i $\vec{N}$, pod istim kutem što ga zatvara vektor $\vec{L}$ sa vektorom $\vec{N}$. Vektor $\vec{V}$ predstavlja vektor usmjeren iz promatrane točke prema promatraču (oku). Vektori $\vec{R}$ i $\vec{V}$ također su jedinični vektori. U tom slučaju kut Ω može se opisati skalarnim produktom $\vec{R}\,\vec{V}$, pa relacija (rel:15.2.3/1) prelazi u:

$$I_s = I_i \cdot k_s \cdot \left(\vec{R} \cdot \vec{V}\right)^n \qquad \text{(rel:15.2.3/2)}$$

Ova komponenta omogućava nam postizanje efekta blještavila. Naime, ovisno o faktoru n kut između promatrača i reflektirane zrake imati će različiti utjecaj. Ukoliko n teži u beskonačnost, površina se ponaša kao zrcalo! Naime, tada će ova komponenta postojati samo u smjeru reflektirane zrake – a to je upravo karakteristika idealnog zrcala. Ukoliko je pak površina nešto grublja, tada ćemo imati rasipanje svjetlosti i u malim kutevima oko reflektirane zrake. To opisujemo konačnim n-om. Slika 15.2.3.2. pokazuje utjecaj indeksa n na intenzitet ove komponente uzevši u obzir položaj vektora $\vec{V}$.



15.2.3.2.

Slika je generirana za n=10,20,40,80,160 i 320. Uz n=10 dobivena je vanjska krivulja što pokazuje da je za male vrijednosti indeksa n ova komponenta prisutna u širokom spektru kuteva oko reflektirane zrake. Najveći n (=320) generirao je najužu krivulju. Iz ovoga proizlazi da se spektar kuteva u kojima ova komponenta ima utjecaja smanjuje i teži prema kutu Ω=0, u kojem bi slučaju komponenta postojala samo u smjeru reflektirane zrake.

15.2.4. AMBIJENTNA KOMPONENTA

Ambijentna komponenta rezultat je interakcija među objektima opisanih u uvodu. U ovom modelu ambijentna komponenta uzima se konstantnom:

$$I_g = I_a \cdot k_a \qquad \text{(rel:15.2.4/1)}$$

Ambijenta komponenta osigurava da površine koje su stražnje u odnosu na izvor svjetlosti ne budu potpuno crne.

15.2.5. UKUPAN UTJECAJ

Ukupan utjecaj iskazuje se kao linearna kombinacija svih triju komponenata:

$$I = I_a \cdot k_a + I_i \cdot \left(k_d \cdot \left(\vec{L} \cdot \vec{N} \right) + k_s \cdot \left(\vec{R} \cdot \vec{V} \right)^n \right) \qquad \text{(rel:15.2.5/1)}$$

Radi jednostavnosti, model pretpostavlja da se svjetlosni izvor, kao i promatrač, nalaze u beskonačnosti. Ova pretpostavka rezultira konstantnim vrijednostima vektora $\vec{L}$ i $\vec{V}$ kroz čitavu scenu, čime se izbjegava potreba za računanjem istih u svakoj promatranoj točci, pa se cijeli postupak značajno ubrzava. No ovo povlači za posljedicu da će proizvoljno velikoj površini biti u svakoj točci pridjeljen isti intenzitet. Da bi se ovo izbjeglo, u model se ipak uvodi ovisnost intenziteta o udaljenosti. Naravno, budući da se promatrač nalazi u

beskonačnosti, kao mjerodavna se uzima udaljenost od točke pogleda (kojom se definira vektor $\vec{V}$) i promatrane točke. Tada se relacija (rel:15.2.5/1) modificira u:

$$I = I_a \cdot k_a + I_i \cdot \left(k_d \cdot \left(\vec{L} \cdot \vec{N}\right) + k_s \cdot \left(\vec{R} \cdot \vec{V}\right)^n\right)/(r+k) \qquad \text{(rel:15.2.5/2)}$$

pri čemu je r udaljenost promatrane točke od točke pogleda, a k konstanta veća od nule koja sprječava dijeljenje sa nulom u slučaju r=0.

Ukoliko se za opis svjetlosti koriste R, G i B komponente, tada relaciju (rel:15.2.5/2) treba primijeniti za sve tri komponente, pri čemu treba uočiti da k_s nije ovisan o svjetlosti; stoga vrijedi:

$$\begin{aligned} I_r &= I_a \cdot k_{a,r} + I_i \cdot \left(k_{d,r} \cdot \left(\vec{L} \cdot \vec{N}\right) + k_s \cdot \left(\vec{R} \cdot \vec{V}\right)^n\right)/(r+k) \\ I_g &= I_a \cdot k_{a,g} + I_i \cdot \left(k_{d,g} \cdot \left(\vec{L} \cdot \vec{N}\right) + k_s \cdot \left(\vec{R} \cdot \vec{V}\right)^n\right)/(r+k) \\ I_b &= I_a \cdot k_{a,b} + I_i \cdot \left(k_{d,b} \cdot \left(\vec{L} \cdot \vec{N}\right) + k_s \cdot \left(\vec{R} \cdot \vec{V}\right)^n\right)/(r+k) \end{aligned} \qquad \text{(rel:15.2.5/2)}$$

15.2.6. ZBIRNI PREGLED MODELA

- Svjetlosni izvori su točkasti.
- Svjetlosni izvori i promatrač nalaze se u beskonačnosti.
- Difuzna i reflektirajuća komponenta modeliraju se lokalno.
- Koristi se empirijski model simulacije rasipanja svjetlosti oko reflekcijskog vektora čime se modelira blještavilo.
- Za boju rasipane svjetlosti uzima se boja izvora.
- Ambijent je modeliran konstantom.

15.2.7. PRIMJER

Pogledajmo kako se ovaj model ponaša u najjednostavnijim slučajevima. U scenu ćemo postaviti kuglu u ishodište 3D sustava. Za prijelaz u 2D sustav koristiti ćemo paralelnu projekciju, uz promatrača koji se nalazi u $+\infty$ na z-osi i gleda prema ishodištu. Koristiti ćemo jedan svjetlosni izvor, koji će se također nalaziti u beskonačnosti, ali ne na z osi, već u ravnini y=0, te u odnosu na ishodište pod kutem od 45° između osi x i z; matematički rečeno, pod vektorom [1 0 1]). Budući da gledamo odozgo, crtati ćemo samo gornji plašt kugle ($z \geq 0$).

Uz paralelnu projekciju preslikavanje iz 3D u 2D sustav vrlo je jednostavno: x'=x, y'=y, z'=0. Crtanje kugle nije riješeno najoptimalnije, no poslužiti će za demonstraciju. Kako se cijela kugla projicirana u z=0 ravninu svede na krug upisan kvadratu $-R \leq x \leq R$ i $-R \leq y \leq R$, funkcija provjerava za svaku točku te površine da li pripada krugu; ako ne, ide se na slijedeću točku, a ako pripada, računa se intenzitet. Funkcija koja ovo računa dana je u nastavku:

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
  int x,y;
  double z, I, Id, Is;
  const int R = 100;
```

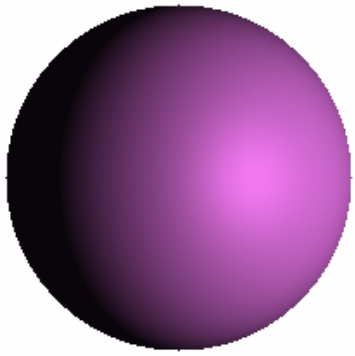
```
  double nx, ny, nz;
  double lx, ly, lz, rx, ry, rz, l_n_2, vx, vy, vz, naz;

  lx = 1; ly = 0; lz = 1;
  naz = sqrt(lx*lx + ly*ly + lz*lz);
  lx = lx / naz; ly = ly / naz; lz = lz / naz;

  vx = 0; vy = 0; vz = 1;
  vz = sqrt(vx*vx + vy*vy + vz*vz);
  vx = vx / vz; vy = vy / vz; vz = vz / vz;

  for( x=-R; x<=R; x++ ) {
   for( y=-R; y<=R; y++ ) {
    z = R*R - (x*x + y*y);
    if( z < 0 ) continue;
    z = sqrt(z);
    nz = sqrt(x*x + y*y + z*z);
    nx = x / nz; ny = y / nz; nz = z / nz;
    Id = lx*nx + ly*ny + lz*nz;
    l_n_2 = 2*Id;
    if( Id > 0. ) Id = 200*Id; else Id = 0.;
    rx=l_n_2*nx-lx; ry=l_n_2*ny-ly; rz=l_n_2*nz-lz;
    naz = sqrt(rx*rx + ry*ry + rz*rz);
    rx = rx / naz; ry = ry / naz; rz = rz / naz;
    Is = rx*vx + ry*vy + rz*vz;
    if( Is > 0. ) {
     Is = 45*pow(Is,80);
    } else Is=0.;
    I = 10. + Id + Is;
    Image1->Canvas->Pixels[x+150][-y+150]=RGB(I,I/2,I);
   }
  }
}
```

Svaki izračunati pixel crta se na zaslonu, a budući da se dio kruga dobiva uz negativne vrijednosti x i y koordinata, ishodište je pomaknuto u točku (150,150). Dodatno se mijenja i orijentacija y osi prema gore (na zaslonima se pozitivni y proteže prema dolje).

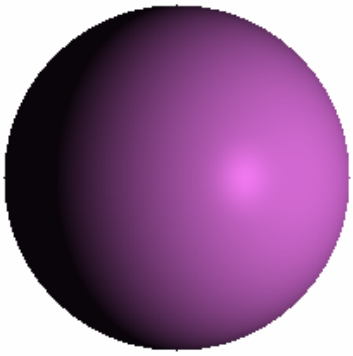
Funkcija pretpostavlja slijedeće vrijednosti:

- I_i=256
- k_d=0.78125 $\Rightarrow$ I_d= $I_i k_d$=200
- k_s= 0.17578125 $\Rightarrow$ I_s= $I_i k_s$=45
- I_g=$I_a k_a$=10.
- n=80

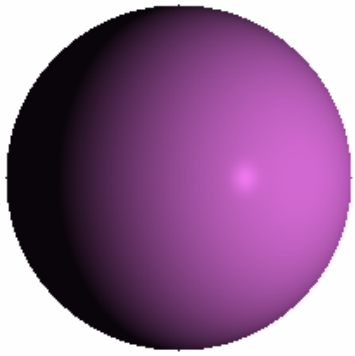
Četiri slike uz različite vrijednosti faktora n prikazane su u nastavku.



Slika 15.2.7.1.
n = 5



Slika 15.2.7.2.
n = 20



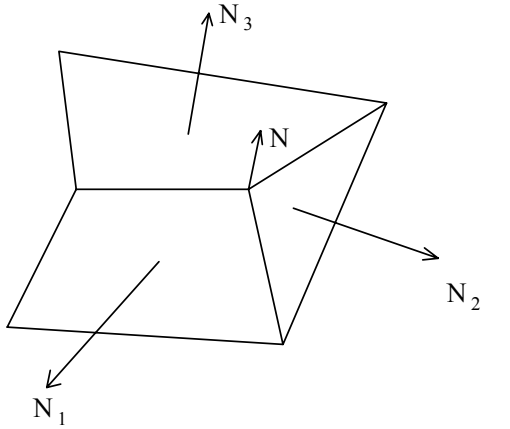
Slika 15.2.7.3.
n = 80

Slika 15.2.7.4.
n = 360

15.3. GOURAUDOVO SJENČANJE POLIGONA

U poglavlju 15.2. opisan je Phongov model. Jedan od postupaka koji sjenčanje poligona temelje na tom modelu je Gouraudov postupak. Prema relaciji (rel:15.2.2.1/2) difuzna komponenta u svakoj točci ovisi o skalarnom produktu vektora $\vec{L}$ i vektora $\vec{N}$. Budući da je izvor smješten u beskonačnosti, vektor $\vec{L}$ konstantnog je iznosa u cijeloj sceni. Kako sjenčamo poligon, a poligon je dio ravnine, tada je po cijeloj površini poligona vektor $\vec{N}$ konstantan. Slijedi da je skalarni produkt tih vektora, a time i difuzna komponenta, po cijeloj površini poligona konstantna.

Tijela (zapravo njihovo oplošje) u 3D prostoru obično modeliramo nizom poligona. Sjenčanje tijela tada se svodi na sjenčanje poligona. Svaki poligon dijeli pojedine vrhove sa drugim poligonima. Potrebno je u svakom vrhu pronaći srednju normalu, i pomoću nje izračunati intenzitet u tom vrhu. Srednja normala računa se kao aritmetička sredina normala svih poligona koji dijele taj vrh. Slika 15.3.1. prikazuje primjer.



15.3.1.

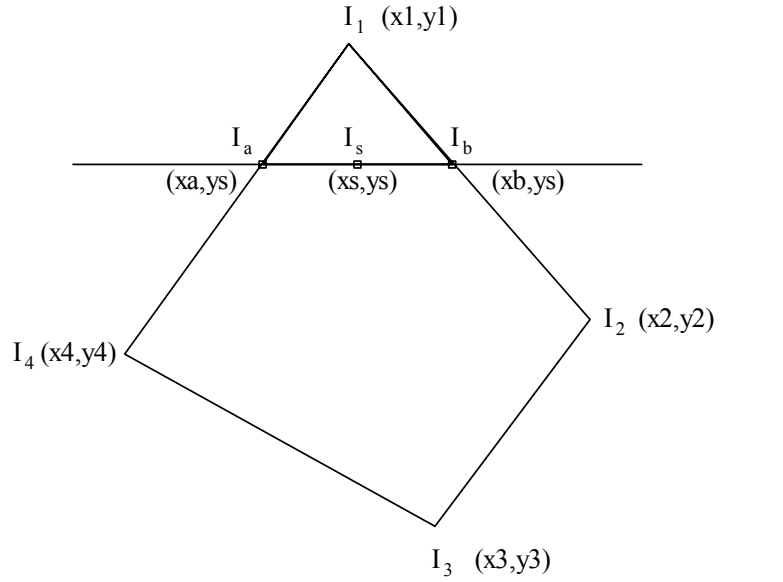
Uz sliku vrijedi relacija $\vec{N} = \frac{1}{3}\left(\vec{N}_1 + \vec{N}_2 + \vec{N}_3\right)$, dok općenito vrijedi:

$$\vec{N} = \frac{1}{n}\sum_{i=1}^{n} \vec{N}_i \qquad \text{(rel:15.3/1)}$$

Intenzitet u vrhu računa se prema relaciji (rel:15.2.5/1), pri čemu zanemarujemo I_s komponentu, pa slijedi:

$$I = I_a \cdot k_a + I_i \cdot k_d \cdot \left(\vec{L} \cdot \vec{N}\right) \qquad \text{(rel:15.3/2)}$$

Nakon što ovo izračunamo za sve vrhove, krećemo na sjenčanje poligona. Sada svaki poligon, općenito govoreći, ima u svojim vrhovima različite intenzitete. Ove intenzitete treba najprije interpolirati uzduž svih bridova (npr. DDA postupkom). Nakon ovoga poznati su intenziteti u svakoj točci svakog brida poligona. Sada još treba te intenzitete interpolirati od lijevih bridova poligona do desnih bridova poligona (opet DDA postupkom). Ideja je pokazana na slici 15.3.2.



15.3.2.

Za točku (xa,ys) intenzitet se dobije interpolacijom između I_1 i I_4:

$$I_a = \frac{1}{y_4 - y_1}\left(I_1\left(y_4 - y_s\right) + I_4\left(y_s - y_1\right)\right)$$

Za točku (xb,ys) intenzitet se dobije interpolacijom između I_1 i I_2:

$$I_b = \frac{1}{y_2 - y_1}\left(I_1\left(y_2 - y_s\right) + I_2\left(y_s - y_1\right)\right)$$

Intenzitet točke (xs,ys) dobije se interpolacijom intenziteta I_a i I_b:

$$I_s = \frac{1}{x_b - x_a}\left(I_a\left(x_b - x_s\right) + I_b\left(x_s - x_a\right)\right)$$

Ovakav postupak računanja naziva se bilinearna interpolacija (*naime, najprije se linearno interpoliraju intenziteti uzduž y osi, a potom se vrši interpolacija tih interpoliranih vrijednosti uzduž x osi*).

Budući da se intenzitet I_s računa za svaki pixel scan-linije koji se nalazi unutar poligona, postupak se može malo modificirati da bi se dobilo na brzini. Uvede se prirast ΔI_s:

$$\Delta I_s = \frac{\Delta x}{x_b - x_a}\left(I_b - I_a\right)$$

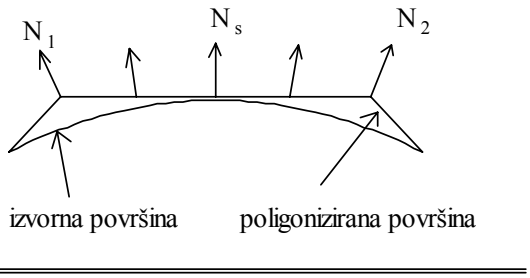
pa se intenzitet može računati prema:

$$I_{s,n} = I_{s,n-1} + \Delta I_s$$

Pri tome Δx označava korak po x-osi. Ukoliko popunjavamo svaki pixel, što je uobičajeno, Δx iznosi 1.

15.4. PHONGOVO SJENČANJE

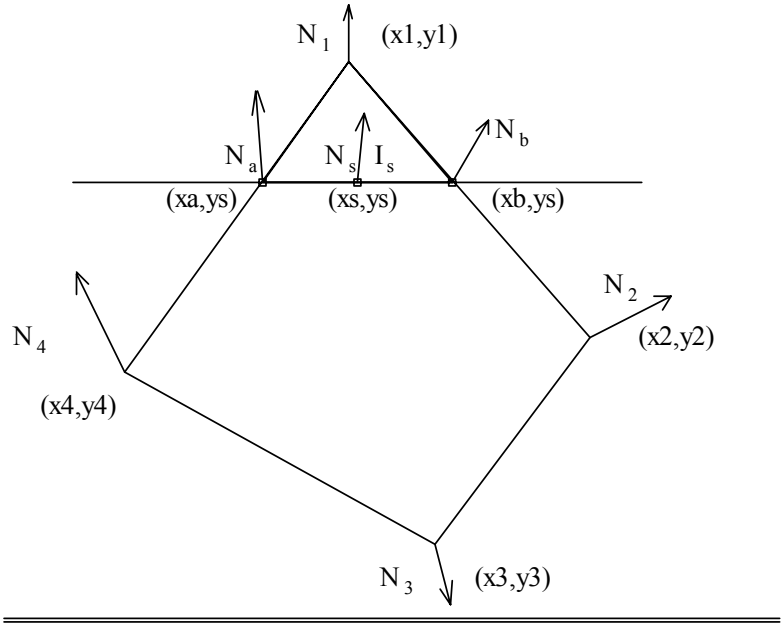
Ovo je još jedan postupak koji se zasniva na Phongovom reflekcijskom modelu. Postupak je za računalo zahtjevniji, ali daje bolje rezultate od Gouraudovog sjenčanja. Postupak zadržava bilinearnu interpolaciju, ali više se ne interpoliraju intenziteti, već same normale, da bi se na kraju za svaki pixel na temelju dobivene vrijednosti za normalu izvelo računanje intenziteta na temelju relacije (rel:15.3/2). Sada je vidljivo zašto je postupak puno zahtjevniji. Međutim, dobra strana postupka jest privid da interpolirana normala slijedi zakrivljenosti površine. Slika 15.4.1. pokazuje primjer.



15.4.1.

Zakrivljenu površinu pokušali smo prikazati poligonima. U tu svrhu generirano je nekoliko poligona tako da "slijede" zakrivljenost površine. Naravno, mi smo uzeli svega tri poligona (dok bi za koliko toliko realnu i glatku zakrivljenost trebalo uzeti puno više).Normale u zajedničkim vrhovima su N_1 i N_2. Ukoliko normalu N_s dobijemo (bi)linearnom interpolacijom, dobiva se dojam da normala slijedi zakrivljenost izvorne površine, pa očekujemo i bolje rezultate od sjenčanja.

U postupak krećemo kao i kod Gouraudovog sjenčanja: potrebno je izračunati normale u svim vrhovima prema relaciji (rel:15.3/1). Zatim se izvodi bilinearna interpolacija normala, tako da u točci (xs,ys) dobijemo normalu N_s. Posljednji korak je izračun intenziteta u toj točci prema relaciji (rel:15.3/2) gdje za N uzimamo N_s. Slika 15.4.2. pokazuje postupak.



15.4.2.

Za točku (xa,ys) normala se dobije interpolacijom između N_1 i N_4:

$$\vec{N}_a = \frac{1}{y_4 - y_1}\left(\vec{N}_1(y_4 - y_s) + \vec{N}_4(y_s - y_1)\right)$$

Za točku (xb,ys) normala se dobije interpolacijom između N_1 i N_2:

$$\vec{N}_b = \frac{1}{y_2 - y_1}\left(\vec{N}_1(y_2 - y_s) + \vec{N}_2(y_s - y_1)\right)$$

Normala točke (xs,ys) dobije se interpolacijom normala N_a i N_b:

$$\vec{N}_s = \frac{1}{x_b - x_a}\left(\vec{N}_a(x_b - x_s) + \vec{N}_b(x_s - x_a)\right)$$

Budući da se normala N_s računa za svaki pixel scan-linije koji se nalazi unutar poligona, postupak se može malo modificirati da bi se dobilo na brzini. Uvede se prirast ΔN_s:

$$\Delta\vec{N}_s = \frac{\Delta x}{x_b - x_a}\left(\vec{N}_b - \vec{N}_a\right)$$

pa se intenzitet može računati prema:

$$\vec{N}_{s,n} = \vec{N}_{s,n-1} + \Delta\vec{N}_s$$

Pri tome Δx označava korak po x-osi. Ukoliko popunjavamo svaki pixel, što je uobičajeno, Δx iznosi 1. Treba uočiti da ovdje sve relacije opisuju **vektore**, što znači da se svaka relacija pri računanju raspada na tri relacije (jer se računanje provodi za svaku komponentu vektora

posebno). To je još jedan razlog daleko veće računske zahtjevnosti na sklopovlje koje izvodi ove kalkulacije.

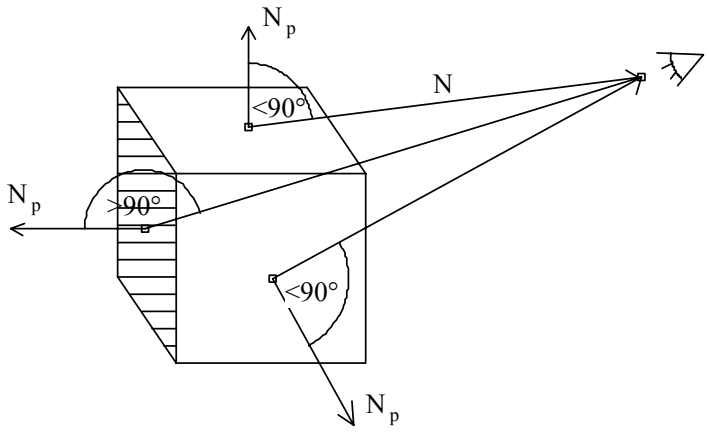
16. UKLANJANJE NEVIDLJIVIH ELEMENATA

16.1. UVOD

Objekte scene opisujemo matematičkim modelima: poligonima, funkcijama i sl. Rad sa ovakvim opisom (doduše, jedinim mogućim) vrlo je spor i računski zahtjevan. Dokaz ove tvrdnje vidi se već iz postupka sjenčanja. Da bi se iscrtavanje scene maksimalno ubrzalo, treba iz postupka iscrtavanja izbaciti sve radnje koje su nepotrebne (a takvih ima). Npr. ukoliko u scenu stavimo kocku, gdjegod da se nalazio promatrač, nikada mu neće biti vidljive sve stranice kocke. Te "nevidljive" stranice zapravo su stražnje stranice kocke gledano iz smjera promatrača. Te stranice ne treba niti sjenčati jer će ionako biti uklonjene. Pročitate li pažljivo prethodnu rečenicu, uočiti ćete da se u njoj nalaze dvije različite tvrdnje. Prva tvrdnja je da stranice ne treba sjenčati. Na ovom mjestu otvara se prostor za algoritme koji unaprijed odlučuju koje radnje ne treba činiti. Razvijen je niz algoritama u ove svrhe, no većina ih radi sa površinama (dakle, svim vrstama poligona i sl.). Ovakvim algoritmima problem predstavljaju same točke. Naime, točka nema niti površinu, niti orijentaciju i vrlo je nezgodna za 3D scenu. Druga spomenuta tvrdnja je "ionako će biti uklonjene". Ovo je mjesto na kojem su nastali različiti algoritmi koji vidljivost rješavaju na razini točke. Dakako, ovi algoritmi su daleko moćniji od prethodno spomenutih jer rade sa najmanjim djelićem prostora (sve ostalo poput linija, površina i sl. sastoji se upravo od točaka). Nedostatak ovih algoritama je u izuzetnoj sporosti i velikim memorijskim zahtjevima. U algoritme iz prvog dijela spadaju provjere u sustavu scene (uklanjanje stražnjih poligona i sl) i provjere u sustavu prikaza (minimax provjere). Drugi dio odnosi se na Z-buffer algoritam.

16.2. UKLANJANJE STRAŽNJIH POLIGONA – PROVJERA NORMALE

Ovaj postupak pomoći će nam da otkrijemo koji su poligoni tijela stražnji u odnosu na promatrača. Pretpostavka je da je tijelo opisano nizom poligona, i da je tijelo konveksno. Nadalje, pretpostavlja se da su vrhovi poligona zadani u smjeru suprotnom od smjera kazaljke na satu ako gledamo poligon iz točke koja se nalazi izvan tijela. Ovo će za posljedicu imati da sve normale poligona gledaju iz tijela prema van. Tada se odluka da li je poligon prednji ili stražnji može donijeti na temelju kuta što ga normala poligona zatvara na vektorom usmjerenim prema gledatelju. Slika 16.2.1. demonstrira zaključivanje:



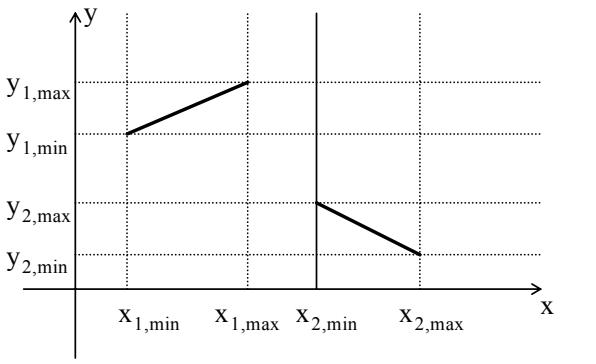
16.2.1.

Potrebno je promatrati kut što ga zatvara vektor normale poligona $\vec{N}_p$ i vektor iz središta poligona prema promatraču $\vec{N}$. Tada vrijedi:

- Poligon je stražnji, ukoliko je $\vec{N}_p \cdot \vec{N} < 0$
- Poligon je prednji (vidljiv), ukoliko je $\vec{N}_p \cdot \vec{N} > 0$
- Poligon je degenerirao u liniju ukoliko je $\vec{N}_p \cdot \vec{N} = 0$

16.3. MINIMAX PROVJERE

Minimax provjere služe nam brza provjera da li se dva objekta ne preklapaju. Prethodna rečenica zvuči malo čudno, ali ispravno karakterizira postupak. Naime, minimax provjerama možemo utvrditi da li je sigurno da se dva objekta ne preklapaju. Ukoliko pak utvrdimo da postoji mogućnost da se objekti preklapaju, tada daljnje provjere treba vršiti drugim algoritmima. Evo ideje. Svaki objekt u sustavu prikaza ima svoje minimalne i maksimalne koordinate kroz koje se proteže. Usporedbom tih vrijednosti možemo donijeti neke zaključke. Primjer je prikazan na slici 16.3.1. Provjerava se preklapanja dvaju segmenata linije. Linije su uzete radi jednostavnosti, no postupak je identičan za bilo kakve druge objekte.

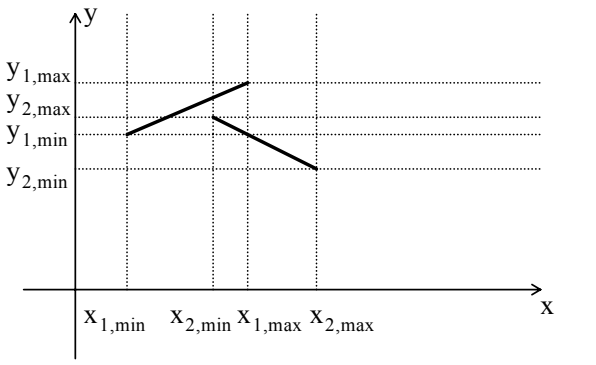


16.3.1.

Postavlja se pitanje, kada možemo biti sigurni da se objekti ne preklapaju?

- Ukoliko je $x_{1,max} < x_{2,min}$
- Ukoliko je $x_{1,max} < x_{1,min}$
- Ukoliko je $y_{1,max} < y_{2,min}$
- Ukoliko je $y_{2,max} < y_{1,min}$

Ukoliko je zadovoljen bilo koji od ova četiri uvjeta, tada smo sigurni da se objekti ne preklapaju. Ukoliko niti jedan od ta četiri uvjeta nije ispunjen, tada treba izvršiti daljnje provjere. Naime, i dalje je moguće da se objekti ne preklapaju, a niti jedan gornji uvjeti nije ispunjen (primjer na slici 16.3.2.).



16.3.2.

Na slici 16.3.2. objekti se i dalje ne preklapaju, no minimax provjerom to ne možemo utvrditi. Minimax provjera u ovom slučaju rezultirati će potrebom za dodatnim provjerama, jer se kvadratna područja u kojima leže objekti preklapaju.

16.4. Z BUFFER

Z buffer je postupak koji provjeru preklapanja radi na razini točaka. Ideja je slijedeća. Slika se iscrtava na zaslon konačnih dimenzija. Tada se prilikom iscrtavanja točke na zaslon u z bufferu pamti udaljenost izvorne točke od promatrača (z koordinata točke nakon projekcije). Npr. potrebno je iscrtati točku (x,y,z). Vrši se projekcija točke i dobije se točka (x',y',z'). Na zaslon treba iscrtati točku na poziciji (x',y'). No prije nego što nacrtamo tu točku, potrebno je u z bufferu pogledati da li je na tu poziciju već nacrtana neka druga točka, i ako je, koliko je ona daleko bila od promatrača. Ukoliko je naša točka bliža, tada ona skriva prethodno iscrtanu točku (jer je bliže promatraču), pa je crtamo i u z buffer upisujemo na mjesto (x',y') udaljenost naše točke. Ukoliko je naša točka dalja, tada je ne crtamo.

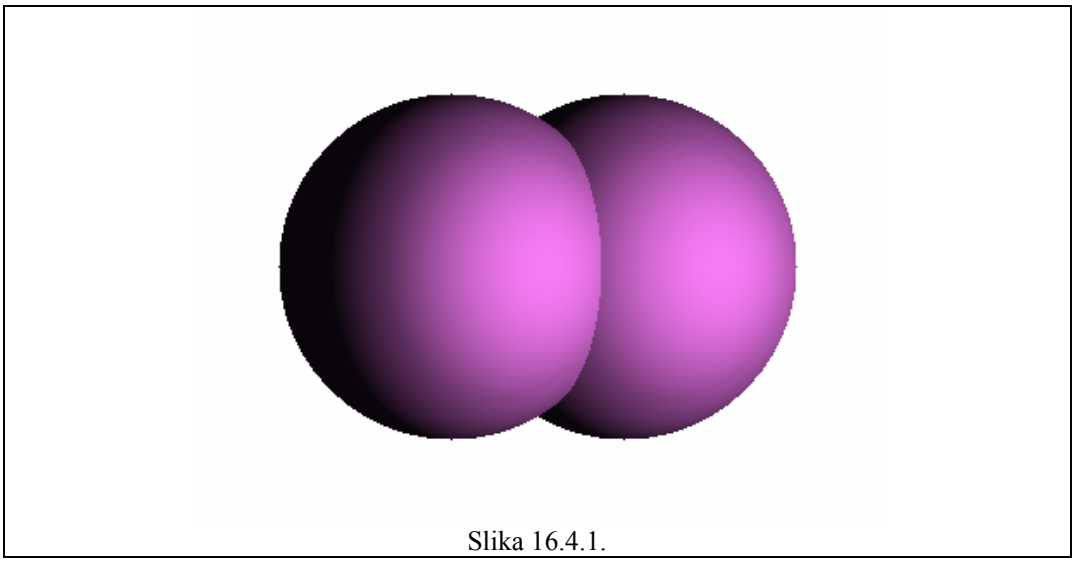
U praktičnim izvedbama z buffer implementira se kao polje dimenzija rezolucija_x * rezolucija_y. Da li će to biti polje tipa short, int ili nešto treće, ovisi o dozvoljenoj dubini scene. Inicijalno se svi elementi polja postave na najveće vrijednosti (*dakle, do sada smo "iscrtavali" jedino točke u beskonačnosti koje su iza svih*).

Najveći nedostatak ovakvog pristupa je u zahtjevima za memorijom. Npr. odlučimo li se na prikaz u rezoluciji 1024x768 pixela, u maksimalnu dublinu scene takvu da možemo koristiti short-tip (2 bajta), potrebna količina memorije iznosi 1024*768*2=1572864 bajtova ili 1.5MB! Zahtjevi za memorijom mogu još porasti ako u z buffer odlučimo upisivati više informacija. Naime, ukoliko ga koristimo za spremanje dubine, a točke iscrtavamo na zaslonu, tada ćemo kod vrlo kompleksnih scena kojima treba dosta vremena za iscrtavanje uočiti kako se neki pixeli pojavljuju, pa ih zamjenjuju drugi pixeli dobiveni od bližih objekata. Da bi se ovo izbjeglo, z buffer možemo koristiti tako da pamti i dubinu pojedine točke, i njezinu boju (dakle iscrtavanje se u ovom koraku ne radi na zaslon). Jednom kada smo čitavu scenu iscrtali, možemo cijelu sliku u jednom prolazu iz z buffera prebaciti na zaslon. No ovakav postupak za svaku točku z buffera zahtjeva 2 bajta za dubinu + 3 bajta za boju što daje 5 bajtova po točki, ili uz rezoluciju 1024x768 iznos od 3932160 bajtova ili 3.75MB.

Ove ogromne memorijske zahtjeve možemo ublažiti višeprolaznim iscrtavanjem scene: npr. u prvom prolazu iscrtati ćemo gornju polovicu zaslona, a u drugom prolazu donju polovicu. Tada nam treba z buffer veličine za pola zaslona. Općenito, n prolazno iscrtavanje zahtjeva samo n-ti dio z buffera koji bi trebalo jednoprolazno iscrtavanje, ali zato postupak traje n puta dulje!

Razvijen je i scan line z buffer algoritam koji iscrtava liniju po liniju zaslona; no efikasna primjena ovog algoritma zahtjeva velike izmjene u načinu pamćenja objekata.

Kod Phongovog modela osvjetljavanja prikazali smo funkciju koja je nacrtala kuglu i osjenčala je. U nastavku ćemo pokazati funkciju koja je osjenčati dvije kugle koje se međusobno probadaju koristeći Phongov model i z buffer. Rezultat rada funkcije prikazan je na slici 16.4.1. a funkcija je navedena u nastavku.



Slika 16.4.1.

Centar desne kugle nalazi se nešto ispod centra lijeve kugle.

```
void DvijeKugle(void)
{
  int x,y;
  double z, I, Id, Is;
  const int R = 100;
  double nx, ny, nz;
  short *buffer;
  double lx, ly, lz, rx, ry, rz, l_n_2, vx, vy, vz, naz;

  lx = 1; ly = 0; lz = 1;
  naz = sqrt(lx*lx + ly*ly + lz*lz);
  lx = lx / naz; ly = ly / naz; lz = lz / naz;

  vx = 0; vy = 0; vz = 1;
  naz = sqrt(vx*vx + vy*vy + vz*vz);
  vx = vx / naz; vy = vy / naz; vz = vz / naz;

  buffer = (short*)malloc(300*400*sizeof(short));
  if( buffer==NULL ) {
   Application->MessageBox("Nema dovoljno memorije","Poruka",MB_APPLMODAL|MB_ICONERROR|MB_OK);
   return;
  }

  for( y=0; y<300; y++ ) {
   for( x=0; x<400; x++ ) {
    buffer[y*400+x] = -32000;
   }
  }

  for( x=-R; x<=R; x++ ) {
   for( y=-R; y<=R; y++ ) {
    z = R*R - (x*x + y*y);
    if( z < 0 ) continue;
    z = sqrt(z);
    nz = sqrt(x*x + y*y + z*z);
    nx = x / nz; ny = y / nz; nz = z / nz;
    Id = lx*nx + ly*ny + lz*nz;
    l_n_2 = 2*Id;
    if( Id > 0. ) Id = 200*Id; else Id = 0.;
    rx=l_n_2*nx-lx; ry=l_n_2*ny-ly; rz=l_n_2*nz-lz;
    naz = sqrt(rx*rx + ry*ry + rz*rz);
    rx = rx / naz; ry = ry / naz; rz = rz / naz;
```

```
    Is = rx*vx + ry*vy + rz*vz;
    if( Is > 0. ) {
     Is = 45*pow(Is,1.1);
    } else Is=0.;
    I = 10. + Id + Is;
    buffer[(y+150)*400+(x+150)] = z;
    Image1->Canvas->Pixels[x+150][y+150]=RGB(I,I/2,I);
   }
  }

  for( x=-R; x<=R; x++ ) {
   for( y=-R; y<=R; y++ ) {
    z = R*R - (x*x + y*y);
    if( z < 0 ) continue;
    z = sqrt(z);
    nz = sqrt(x*x + y*y + z*z);
    nx = x / nz; ny = y / nz; nz = z / nz;
    Id = lx*nx + ly*ny + lz*nz;
    l_n_2 = 2*Id;
    if( Id > 0. ) Id = 200*Id; else Id = 0.;
    rx=l_n_2*nx-lx; ry=l_n_2*ny-ly; rz=l_n_2*nz-lz;
    naz = sqrt(rx*rx + ry*ry + rz*rz);
    rx = rx / naz; ry = ry / naz; rz = rz / naz;
    Is = rx*vx + ry*vy + rz*vz;
    if( Is > 0. ) {
     Is = 45*pow(Is,1.1);
    } else Is=0.;
    I = 10. + Id + Is;
    if(buffer[(y+150)*400+(x+150+100)]<z-50) {
     buffer[(y+150)*400+(x+150+100)] = z-50;
     Image1->Canvas->Pixels[x+150+100][y+150]=RGB(I,I/2,I);
    }
   }
  }
  free(buffer);
}
```

Lijeva kugla ima centar u ishodištu, dok desna kugla ima centar u točci (100,0,-50). U petlji koja crta lijevu kuglu z-buffer samo se popunjava vrijednostima, budući da je to prvi objekt. Pri crtanju druge kugle postupak računanja je isti, samo što se z buffer najprije provjerava. Svaka točka desne kugle računa se kao da je u ishodištu, a umjesto dobivenih x i y i z koordinata koriste se koordinate (x+100,y,z-50). Pri implementaciji z buffera u ovoj funkciji točka je bliža i promatraču što joj je z koordinata veća (jer je promatrač na z osi u plus beskonačnosti). Zbog toga je inicijalno z buffer popunjen sa negativnim vrijednostima.

17. FRAKTALI

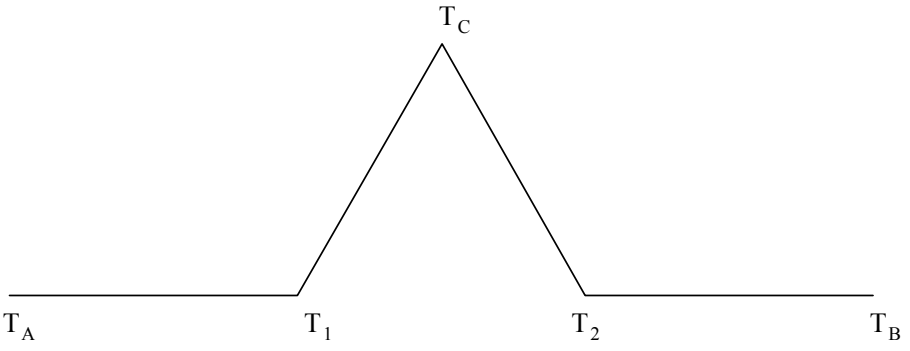
17.1. UVOD

Jedno od ljepših područja računalne grafike pokriva i matematičke umotvorine – fraktale. Mi ćemo u nastavku dati prikaz nekoliko karakterističnih fraktala i pojasniti postupak kako se do njih dolazi. Treba napomenuti da ovdje prikazani fraktali čine vrlo mali dio do sada otkrivenih fraktala, a generalno govoreći, skup fraktala je beskonačan skup tako da ih nikada niti nećemo upoznati sve. Inače, moram se korigirati glede uvodne rečenice; fraktali zapravo i nisu matematičke već prirodne tvorevine. Matematičari su samo našli način kao baratati sa njima. Pa krenimo od najjednostavnije vrste.

17.2. SAMOPONAVLJAJUĆI FRAKTALI

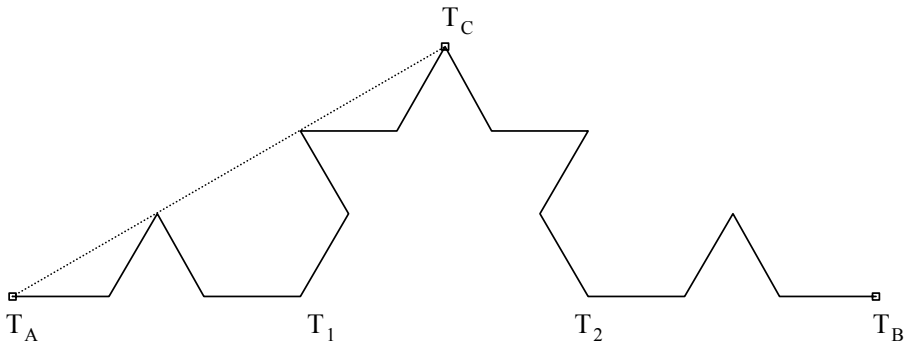
Samoponavljajući fraktali su tvorevine koje na svim skalama umanjenja zadržavaju isti karakteristični oblik. Kao primjer ćemo uzeti Kochinu krivulju. Kochina krivulja dobije se slijedećim jednostavnim algoritmom.

1. Krenimo od osnovnog oblika prikazanog na slijedećoj slici.



Konstrukcija fraktala

2. Svaki od četiri segmenta prepravi tako da umjesto linije umetneš cijeli lik iz točke 1. Dobiti će se slika:

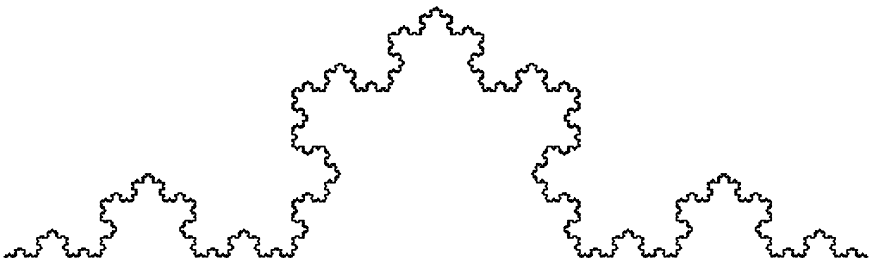


Konstrukcija fraktala

Točke T_A i T_C spojene su spojnicom da se pokaže kako novi vrhovi leže na njoj; inače spojnica ne spada u proces generiranja fraktala.

3. Svaki segment nastao u prethodnom koraku opet zamijeni oblikom iz koraka 1.; i proces ponavljaj u beskonačnost…

Praktična implementacija gornjeg algoritma neće naravno ići u beskonačnost, već do neke zadane dubine. Npr. ukoliko se ide do dubine 6, dobije se slika:



Primjer fraktala

Kochina krivulja dobije se kada rekurziju pustimo da ide u beskonačnost. Dakako, na računalima obično prikazujemo aproksimaciju Kochine krivulje. Nitko još nije uspio nacrtati sasvim točnu krivulju…

Funkcija koja iscrtava krivulju prema prethodnom algoritmu može se napisati kao vrlo jednostavna rekurzivna funkcija:

```
void DrawFractal1Rek( T2DRealPoint A, T2DRealPoint B, T2DRealPoint C, int depth )
{
 T2DRealPoint Ap,Bp,Cp;

 if( depth > 5 ) {
  MoveTo(fround(A.x),fround(A.y));
  LineTo(fround(A.x+(B.x-A.x)/3.),fround(A.y+(B.y-A.y)/3.));
  LineTo(fround(C.x),fround(C.y));
  LineTo(fround(A.x+2.*(B.x-A.x)/3.),fround(A.y+2.*(B.y-A.y)/3.));
  LineTo(fround(B.x),fround(B.y));
  return;
 }

 depth++;
```

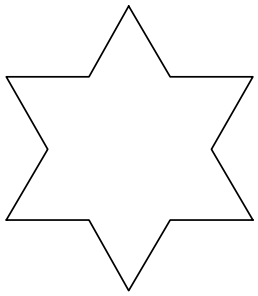
```
 Ap.x = A.x; Ap.y = A.y;
 Bp.x =A.x+(B.x-A.x)/3.; Bp.y = A.y+(B.y-A.y)/3.;
 Cp.x =A.x+(C.x-A.x)/3.; Cp.y =A.y+(C.y-A.y)/3.;
 DrawFractal1Rek( Ap, Bp, Cp, depth );

 Ap.x = Bp.x; Ap.y = Bp.y;
 Bp.x =C.x; Bp.y = C.y;
 Cp.x =A.x+2.*(C.x-A.x)/3.; Cp.y =A.y+2.*(C.y-A.y)/3.;
 DrawFractal1Rek( Ap, Bp, Cp, depth );

 Ap.x = Bp.x; Ap.y = Bp.y;
 Bp.x =A.x+2.*(B.x-A.x)/3.; Bp.y = A.y+2.*(B.y-A.y)/3.;
 Cp.x =B.x+2.*(C.x-B.x)/3.; Cp.y =B.y+2.*(C.y-B.y)/3.;
 DrawFractal1Rek( Ap, Bp, Cp, depth );

 Ap.x = Bp.x; Ap.y = Bp.y;
 Bp.x =B.x; Bp.y = B.y;
 Cp.x =B.x+(C.x-B.x)/3.; Cp.y =B.y+(C.y-B.y)/3.;
 DrawFractal1Rek( Ap, Bp, Cp, depth );
}
```

Funkcija prima točke A, B i C koje na prethodnim slikama odgovaraju točkama T_A, T_B i T_C, te uz njih i malo elementarne geometrije računa točke T'_A, T'_B i T'_C za svoj svaki segment te izračunate točke predaje u novi rekurzivni poziv. Maksimalna dubina rekurzije određena je prvim ispitivanjem i postavljena je na prvu veću od 5 (dakle 6). Uz ovu funkciju, još je potrebna i nerekurzivna funkcija koja će pozvati svoju rekurzivnu inačicu:

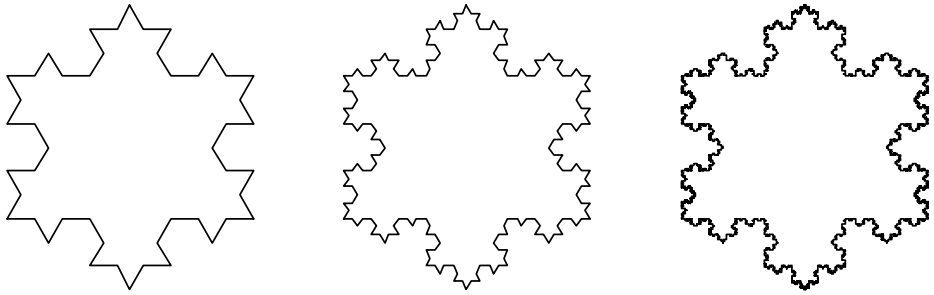
```
void DrawFractal1()
{
  T2DRealPoint A,B,C;

  A.x = 10; A.y = 200 - 10;
  B.x = 320 - 10; B.y = 200 - 10;
  C.x = ( A.x + B.x )/2.;
  C.y = A.y - sqrt(3)/2.*( B.x - A.x )/3.;
  DrawFractal1Rek(A, B, C, 1);
}
```

Funkcija pretpostavlja zaslon razlučivosti 320x200 i ostavlja po 10 praznih pixela lijevo, desno i ispod krivulje.

Jednostavnim proširenjem gornjeg algoritma dobiti ćemo poznatu Kochinu pahuljicu (Koch's Snowflake). Umjesto od jedne linije krenuti ćemo od trokuta čije su stranice segmenti prikazani u prethodnom algoritmu pod brojem 1. Dobiti ćemo lik:



Sada ćemo svaku liniju iterativno zamjenjivati tim segmentima. Rezultat je slijedeći:



pa …

Iz opisanog postupka može se vidjeti da se Kochina pahuljica može dobiti kao jednakostranični trokut čije su stranice Kochine krivulje! Matematički gledano, ovaj fraktal zanimljiv je po još nečemu: duljina njegova opsega je beskonačna, iako je površina koju lik zauzima konačna! Algoritam za crtanje pahuljice također se sastoji od dva dijela: jedne rekurzivne funkcije koja je već navedena kod Kochine krivulje (`DrawFractal1Rek`), i nerekurzivne funkcije koja poziva istu:

```
void DrawFractalSnowflake()
{
  T2DRealPoint A,B,C;
  double visina,d;

  visina = 200 - 20;
  d = sqrt(3.)*visina/2.;

  A.x = 10; A.y = 200 - 10 - visina/4.;
  B.x = A.x + d; B.y = A.y;
  C.x = ( A.x + B.x )/2.;
  C.y = A.y + sqrt(3)/2.*d/3.;
  DrawFractal1Rek(A, B, C, 1);

  B.x = A.x + d/2.;
  B.y = A.y - d*sqrt(3.)/2.;
  C.x = (A.x+B.x)/2.-visina/4.*sqrt(3)/2.;
  C.y = (A.y+B.y)/2.-visina/4./2.;
  DrawFractal1Rek(A, B, C, 1);

  A.x = A.x + d;
  C.x = (A.x+B.x)/2.+visina/4.*sqrt(3)/2.;
  C.y = (A.y+B.y)/2.-visina/4./2.;
  DrawFractal1Rek(A, B, C, 1);
}
```

I ova funkcija podrazumijeva ekran 320x200 pixela. Funkcija se sastoji od tri cjeline: poziva za iscrtavanje donje Kochine krivulje, poziva za iscrtavanje lijeve Kochine krivulje te poziva za iscrtavanje desne Kochine krivulje.

17.3. MANDELBROTOV FRAKTAL

Mandelbrotov fraktal nastaje provjeravanjem konvergencije iterativnog niza u kompleksoj ravnini. Evo o čemu se radi. Neka je zadano iterativno preslikavanje:

$$z_{n+1} = z_n^2 + c \qquad z_0 = 0 + 0i$$

Varijable z i c su kompleksni brojevi. Odabrati ćemo jedan kompleksni broj (i dodijeliti ga varijabli c). Početi ćemo postupak iteracije i dobivati redom brojeve:

$$z_0, z_1, z_2, z_3, z_4, \ldots$$

Postoje tri mogućnosti za ovaj niz:

- Niz konvergira prema nekom broju
- Niz oscilira
- Niz divergira u beskonačnost

Za potrebe dobivanja Mandelbrotovog fraktala ove tri mogućnosti promatrati ćemo kao samo dvije:

- Niz divergira
- Niz ne divergira

Da bismo ispitali ponašanje niza, računati ćemo redom određeni broj koeficijenata. Koliko će to koeficijenata biti, ovisiti će o dobivenom nizu. Naime, odrediti ćemo dva uvjeta zaustavljanja.

1. Maksimalni iznos modula koeficijenta z koji ćemo označiti sa ε. Naime, ukoliko niz divergira, tada će brojevi u nizu postajati sve veći i veći. Kada ustanovimo da je modul broja veći od ε, reći ćemo da niz divergira i prekinuti ćemo iteraciju.
2. Maksimalan broj iteracija m. Da li niz divergira ili ne, provjeriti ćemo u maksimalno m iteracija. Ako do tada niz pokaže tendenciju divergiranja, iteriranje će se prekinuti zbog uvjeta 1. Inače ćemo iteriranje prekinuti nakon m iteracija.

Mandelbrotov fraktal dobiva se tako da se redom provjeravaju točke u kompleksnoj ravnini, i svakoj točci pridruži se boja koja odgovara broju iteracija koje smo izvršili pri provjeravanju divergencije niza u toj točci. Sada treba još samo odgovoriti na pitanje kako ćemo to crtati! Naši zasloni nisu kompleksne ravnine. No analogije ima. Zaslon ima x i y os; kompleksna ravnina ima realnu i imaginarnu os. Zbog toga ćemo izvršiti "mapiranje" zaslona u kompleksnu ravninu. Može se vrlo jednostavno pokazati da ukoliko se želi mapirati interval od a_{min} do a_{max} u interval b_{min} do b_{max}, odgovarajuća formula glasi:

$$b = \frac{(a - a_{\min})}{(a_{\max} - a_{\min})} \cdot (b_{\max} - b_{\min}) + b_{\min}$$

Pri iscrtavanju Mandelbrotovog fraktala provjeravati ćemo dio kompleksne ravnine omeđen realnim komponentama u_{min} i u_{max}, te imaginarnim komponentama v_{min} i v_{max}. Kako želimo iskoristiti cijeli zaslon dimenzija WxH, x koordinate točaka zaslona kretati će se u intervalu od 0 do W-1; dakle $x_{min}=0$, $x_{max}=W-1$. Y-koordinate točaka kretati će se u intervalu od 0 do H-1 pa je $y_{min}=0$, $y_{max}=H-1$. U tom slučaju gornja se formula malo pojednostavljuje, pa se točka (x,y) preslikava u kompleksni broj (u,v) pri čemu vrijedi:

$$u = \frac{x}{x_{\max}} \cdot (u_{\max} - u_{\min}) + u_{\min}$$
$$v = \frac{y}{y_{\max}} \cdot (v_{\max} - v_{\min}) + v_{\min} \qquad \text{(rel:17.3/1)}$$

Uz ove sve podatke, algoritam za iscrtavanje Mandelbrotovog fraktala glasi:

1. $\forall x \in [x_{min}, x_{max}]$ i $\forall y \in [y_{min}, y_{max}]$
2. izvrši mapiranje točke (x,y) u kompleksni broj (u,v) prema relaciji (rel:17.3/1).
3. postavi k=-1; c=(u,v); $z_n=(0,0)$;
4. k=k+1; $z_{n+1}=z_n^2+c$; ako je modul(z_{n+1}) veći od ε ili ako je k>m, pređi na korak 5. Inače ponovi korak 4.
5. Na koordinate (x,y) ispiši pixel u boji k.

Primjer implementacije ovog algoritma dan je u nastavku.

```
void Mandelbrot(double eps, int m,
                double umin, double umax,
                double vmin,  double vmax,
                int xmin, int xmax,
                int ymin, int ymax ) {
  int x, y, k;
  double u,v;
  double z_re, z_im;
  double c_re, c_im;
  double a,b;
  double deltau,deltav;

  eps = eps*eps;
  deltau=(double)(umax-umin)/(double)xmax;
  deltav=(double)(vmax-vmin)/(double)ymax;
  v = (double)ymin*(vmax-vmin)/ymax+vmin;
  for( y = ymin; y <= ymax; y++ ) {
   u = (double)xmin*(umax-umin)/xmax+umin;
   for( x = xmin; x <= xmax; x++ ) {
    c_re = u; c_im = v;
    z_re = z_im = 0.;
    k = -1;
    do {
     k++;
     a = z_re * z_re - z_im * z_im + c_re;
     b = 2. * z_re * z_im + c_im;
     z_re = a; z_im = b;
    } while(a*a+b*b<eps && k<m);
    k = 255-k;
    Pixels[x][y] = RGB(      1.30727E-3*k*k-0.8313931*k+255,
                             4.25231E-3*k*k-1.83336*k+255,
                             7.4591E-4*k*k+6.07737E-2*k
                      );
    u+=deltau;
   }
   v+=deltav;
  }
}
```

Funkcija je napisana uz slijedeće pretpostavke:

- Rad bez palete boja (32-bitna boja) tako da se boje mogu direktno sintetizirati koristeći funkciju RGB(…).
- Maksimalni broj iteracija 255 (ovaj uvjet dolazi samo zbog načina na koji se računaju pojedine komponente boje). Ograničenje se može maknuti ukoliko se promjeni način određivanja boje.
- Provjera da li broj konvergira ne koristi modul već kvadrat modula; time je izbjegnuto vađenje korijena za svaku točku ravnine i za svaku iteraciju čime se dobiva na brzini. No zbog toga je na početku programa i sama pogreška kvadrirana.
- Koriste se opće formule za mapiranje $(x,y) \rightarrow (u,v)$ tako da se ne mora fraktal iscrtavati na cijelom zaslonu. Dodatno su funkcije razložene na dijelove i umjesto neprestanog

množenja i dijeljenja sve je svedeno na inkrementiranje varijabli za unaprije izračunate vrijednosti, koje se računaju samo kada za to ima potrebe.

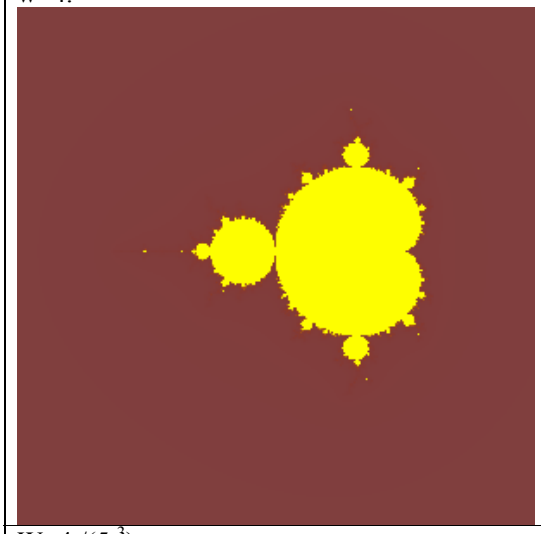
Neki primjeri prikazani su u nastavku.

Mandelbrotov fraktal pokazuje interesantne oblike u kompleksnoj ravnini i to u dijelu od približno –2 do 2 po realnoj i isto tako –2 do 2 po imaginarnoj osi. Ukoliko se odlučimo za kvadratnu površinu s centrom u broju (Cx,Cy) i širinom (i visinom) w, tada se prethodna funkcija može pozvati:

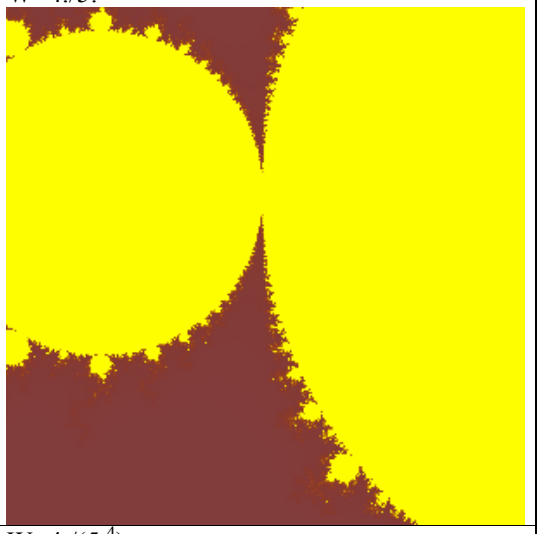
```
Mandelbrot(100, 255, Cx-w/.2, Cx+w/.2, Cy-w/.2, Cy+w/.2, 0, 319, 0, 199 );
```

U nastavku ćemo središte pogleda postaviti u točku (Cx,Cy)=(-0.7454265, 0.1130090). Vrijednost parametra w biti će navedena uz svaku sliku.

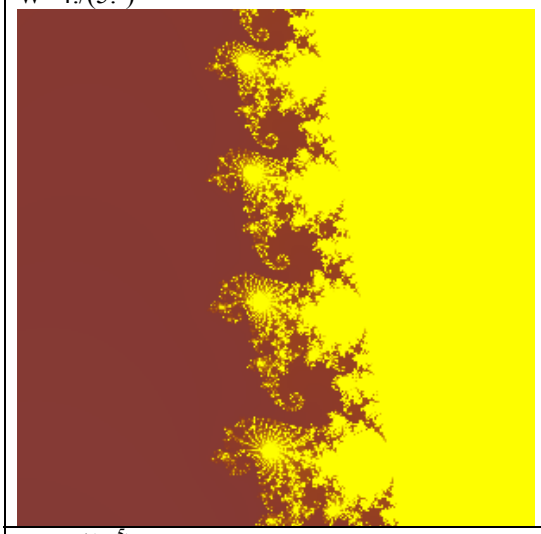
w=4.



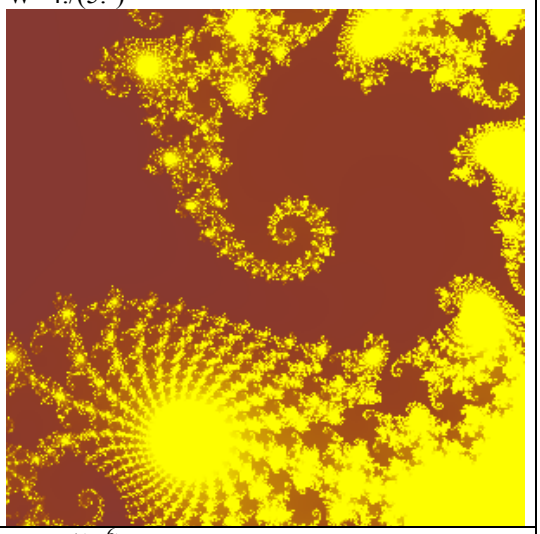
W=4./5.



W=4./(5.^{3})

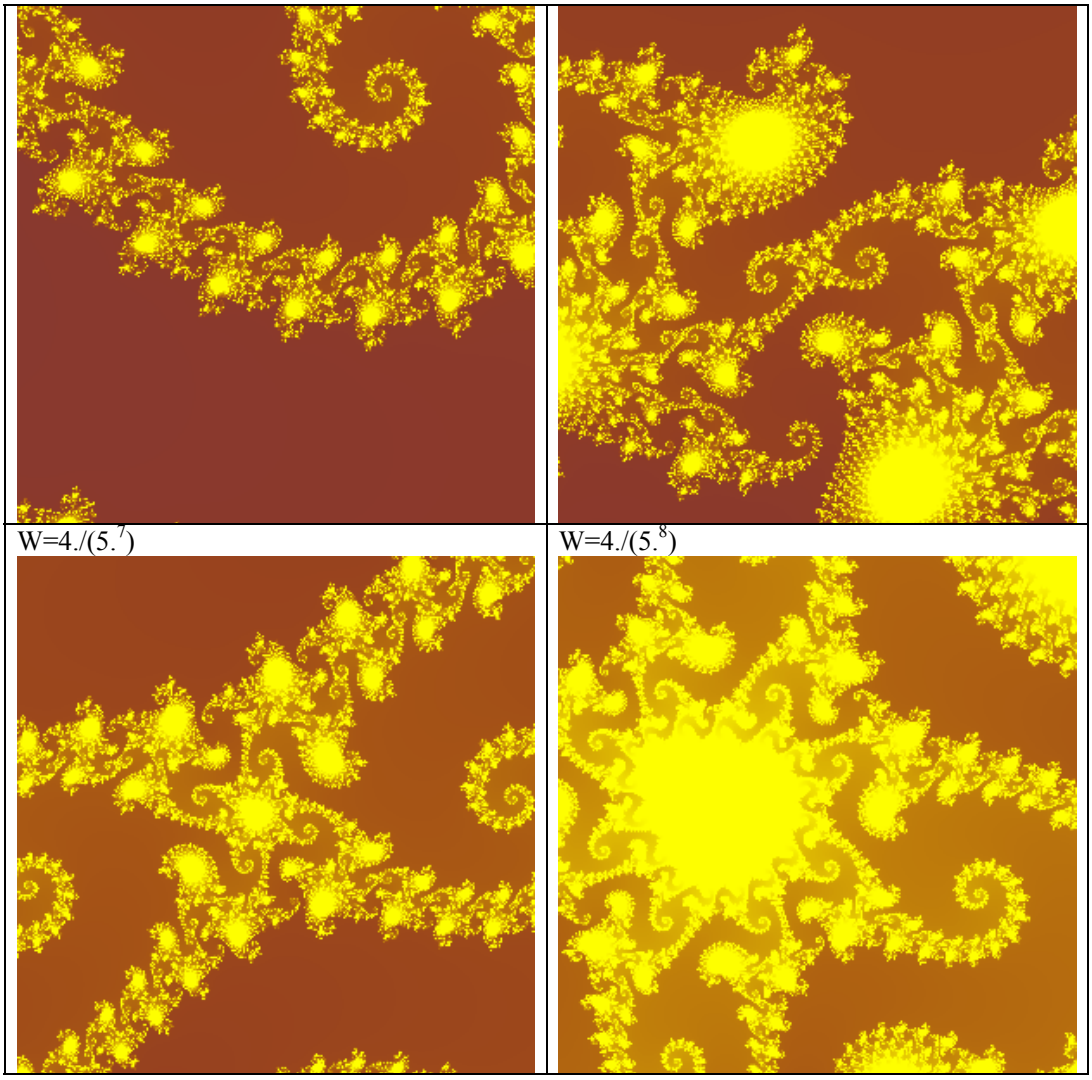


W=4./(5.^{4})



W=4./(5.^{5})

W=4./(5.^{6})



17.4. JULIJEVA KRIVULJA

Da, krivulja. Zašto krivulja a ne fraktal nije mi poznato ali na svim mjestima gdje sam se susreo sa ovime obično je bilo navedeno nešto poput "Mandelbrot set and Julias Curve". Postoji i razlog zašto se ova dva pojma spominju zajedno: Julijevu krivulju dobivamo minornom izmjenom algoritma za Mandelbrotov fraktal. Prisjetimo se još jednom. Mandelbrotov fraktal smo dobili kao poslijedicu provjere konvergencije niza koji nastaje iterativnim preslikavanjem:

$$z_{n+1} = z_n^2 + c \qquad z_0 = 0 + 0i\,.$$

Pri tome smo za konstantu c uzeli upravo točku u kompleksnoj ravnini koju smo ispitivali. Julijevu krivulju dobiti ćemo uz isto preslikavanje:

$$z_{n+1} = z_n^2 + c \qquad z_0 = u + vi$$

ali uz dvije razlike.

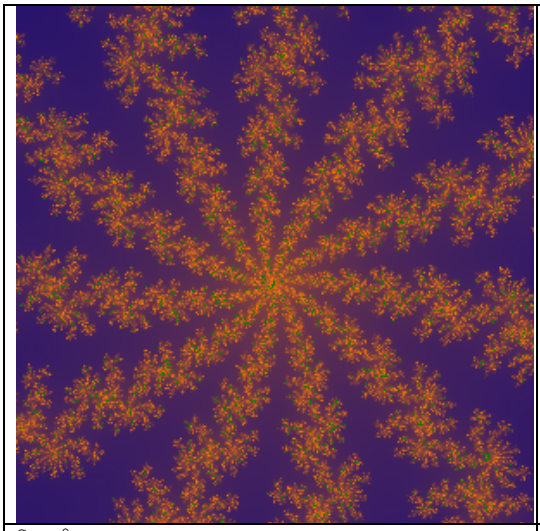
- Prvi član niza nije (0+0i) već upravo točka za koju ispitujemo konvergenciju (u+vi).
- Konstanta je unaprijed čvrsto definiran kompleksni broj koji ne ovisi o točci za koju ispitujemo konvergenciju.

Uz ove dvije izmjene funkcija koja će iscrtavati Julijevu krivulju mogla bi izgledati ovako:

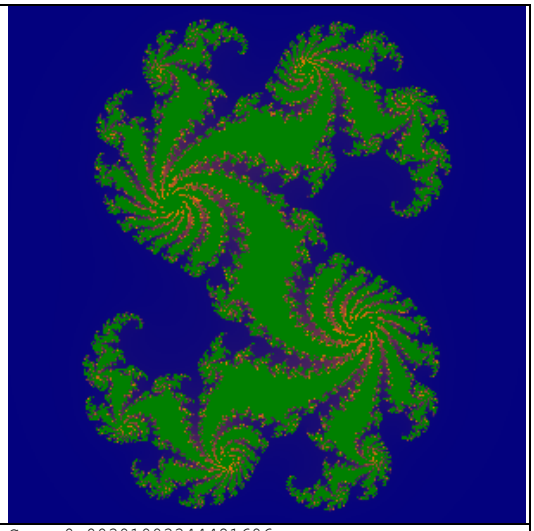
```
Void Julij(double eps, int m, double c_re, double c_im, double umin, double umax,
                  double vmin,  double vmax, int xmin, int xmax, int ymin, int ymax ) {
  int x, y, k;
  double u,v;
  double z_re, z_im;
  double a,b;
  double deltau,deltav;

  eps = eps*eps;
  deltau=(double)(umax-umin)/(double)xmax;
  deltav=(double)(vmax-vmin)/(double)ymax;
  v = (double)ymin*(vmax-vmin)/ymax+vmin;
  for( y = ymin; y <= ymax; y++ ) {
   u = (double)xmin*(umax-umin)/xmax+umin;
   for( x = xmin; x <= xmax; x++ ) {
    z_re = u; z_im = v;
    k = -1;
    do {
     k++;
     a = z_re * z_re - z_im * z_im + c_re;
     b = 2. * z_re * z_im + c_im;
     z_re = a; z_im = b;
    } while(a*a+b*b<eps && k<m);
    k = 255-k;
    Pixels[x][y] = RGB(       1.30727E-3*k*k-0.8313931*k+255,
                              4.25231E-3*k*k-1.83336*k+255,
                              7.4591E-4*k*k+6.07737E-2*k
                      );
    u+=deltau;
   }
   v+=deltav;
  }
}
```

Za prikaz nekih karakterističnih oblika poslužiti ćemo se istom taktikom kao i kod Mandelbrotovog fraktala. Dakle, u kompleksnoj ravnini odabrati ćemo prozor za koji ćemo ispitivati konvergencije točaka. Centar prozora je u točci (Cx,Cy) a širina (i visina) prozora je w. Dodatno, konstanta koja se koristi u postupku biti će označena sa ConstX i ConstY.

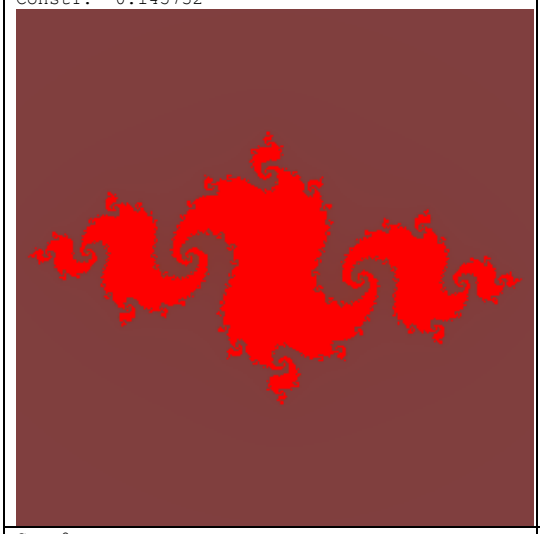
Cx: 0.0769230769230769 Cy: -0.518394648829431 W: 0.2 ConstX: 0.179180 ConstY: 0.588843	Cx: 0 Cy: 0 W: 2.3 ConstX: 0.32 ConstY: 0.043



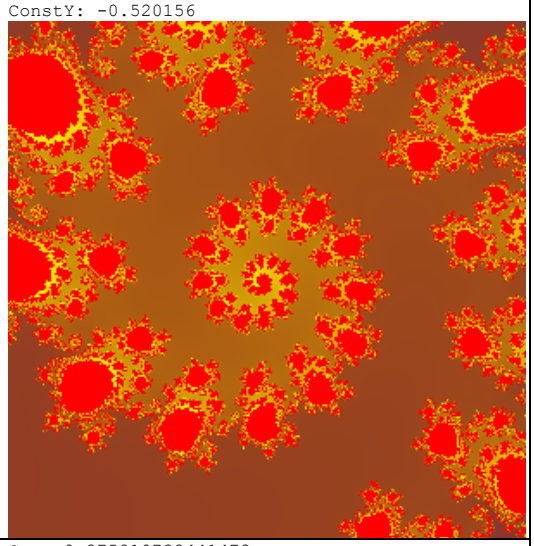
Cx: 0
Cy: 0
W: 3.2
ConstX: -0.807984
ConstY: -0.145732



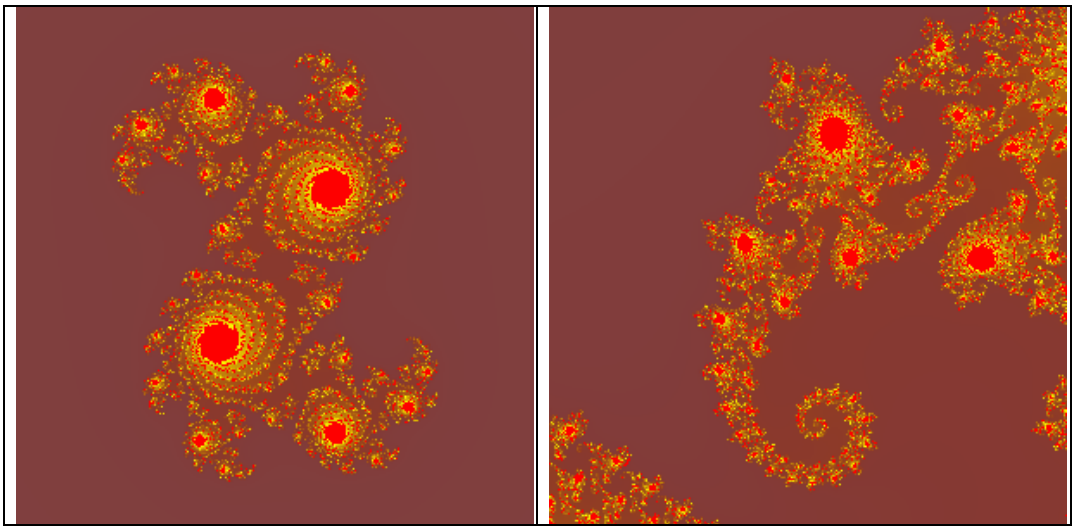
Cx: -0.00301003344481606
Cy: 0.198996655518395
W: 0.08
ConstX: -0.500036
ConstY: -0.520156



Cx: 0
Cy: 0
W: 2.7
ConstX: 0.371588
ConstY: -0.155767



Cx: -0.275919732441472
Cy: -0.396321070234114
W: 0.5
ConstX: -.74543
ConstY: .11301



U nedostatku prostora u nastavku ću navesti samo neke od parametara koji vode do zanimljivih slika.

Svima su zajednički parametri:

- Centar prozora u točci (Cx,Cy)=(0,0)
- Širina (i visina) prozora w=4

ConstX	ConstY
0.155114	-0.470914
-0.391936	0.453160
0.179180	0.588843
-0.906652	-0.063846
-0.502346	0.558921
-0.708605	0.053607
-0.549722	0.550616
0.325578	0.545938
-0.762463	-0.027366
-1.233091	-0.053551
-0.162022	-0.643480
-0.807984	-0.145732
-1.009018	0.200253
0.280671	-0.535311
-1.087530	-0.042409
0.260043	-0.005810
-0.022023	0.705886
0.240183	-0.580091
0.130282	0.613403
0.416111	-0.239016
-0.212523	-0.751017

ConstX	ConstY
-0.859367	0.203302
0.386646	-0.229820
-0.586441	0.483202
0.108418	0.610887
-0.500036	-0.520156
-0.106036	-0.713234
-0.495141	-0.583987
-0.827186	-0.042244
-0.964169	-0.014027
0.275751	0.591888
0.371588	-0.155767
0.337128	-0.412187
-0.193388	-0.703894
0.290019	0.502282
-0.613422	-0.415656
-0.115536	-0.813656
-0.431275	0.578900
-0.753062	-0.229746
-0.124655	0.842925
-1.164108	-0.195863
0.382244	0.370021

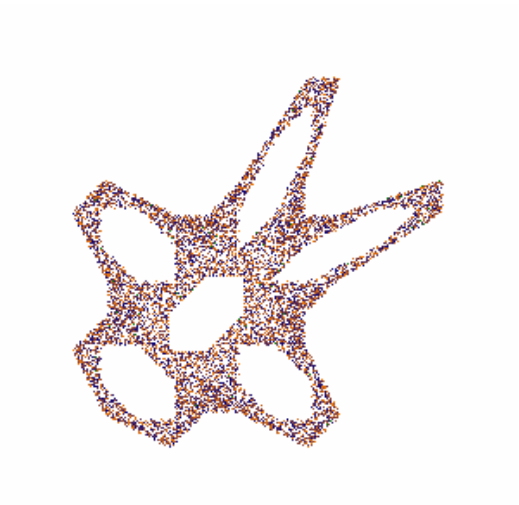
Ovih četrdesetak točaka dobiveno iz petstotinjak iscrtavanja uz slučajno odabrane točke pa su probrane najinteresantnije slike. Uz parametar w=4 dobiti će se prikaz cijelog fraktala, no to bi trebala biti tek polazna točka. Za svaku sliku nakon što se iscrta, centar prozora može se pomaknuti u neku zanimljivu točku i parametar w se smanji na npr. w/10. Zatim se to iscrta. U slijedećem koraku opet se može još zumirati (smanjivanjem w) itd.

17.5. FRAKTAL "by" GINGER

Nažalost, precizniji od ovoga ne mogu biti jer sam do fraktala došao sasvim slučajno preko interneta, i sve što je tamo bilo navedeno je ovo ime. No ono to me se dojmilo je trivijalan generator fraktala: sve što treba učiniti jest odabrati proizvoljne početne vrijednosti za x i y koordinate, te na zaslonu iscrtavati točke koje se dobiju slijedećim iterativnim postupkom:

$$x_{n+1} = 1 - y_n - |x_n|$$
$$y_{n+1} = x_n$$

Ovo je primjer iterativnog preslikavanja koje za neke početne vrijednosti ne divergira. Npr. uz početne vrijednosti: x_0=-0.1, y_0=0.8 dobije se:



Boja svake točke može se odrediti proizvoljno. U ovom primjeru boja se svakom iteracijom uveća za 2, te kada stigne do 256, opet se resetira na nulu. Slika je dobivena uz 10000 iteracija. Funkcija koja je iscrtava:

```
void Ginger2(int m, double x, double y, int xg_min, int xg_max, int yg_min, int yg_max,
                int x_outmin, int x_outmax, int y_outmin, int y_outmax ) {
  int k, c;
  double xn, yn;
  int xp, yp;

  c=0;
  for( k = 0; k < m; k++ ) {
   if(x>=0) xn = 1.-y+x;
   else xn = 1.-y-x;
   yn = x;

   xp = (xn - xg_min)/(double)(xg_max-xg_min)*(double)(x_outmax-x_outmin)+x_outmin;
   yp = (yn - yg_min)/(double)(yg_max-yg_min)*(double)(y_outmax-y_outmin)+y_outmin;
   if(xp>=x_outmin && xp<=x_outmax &&
```

```
      yp>=y_outmin && yp<=y_outmax ) {
     c = c+2;
     if( c > 256 ) c = 0;
     Image1->Canvas->Pixels[xp][yp] = RGB(c,c/2,(256-c)/2);
   }
   x=xn; y=yn;
  }
}
```

Najveći broj parametara funkcije služi opisu prozora kroz koji ćemo gledati. Naime, točke koje se dobiju postupkom leže relativno blizu ishodištu, pa bismo direktnim iscrtavanjem dobili samo mrlju na zaslonu oko ishodišta. Zbog toga vršimo skaliranje očekivanih raspona (xg_min, xg_max) i (yg_min, yg_max) koje daje algoritam i raspona koje predstavlja zaslon (x_outmin, x_outmax) i (y_outmin, y_outmax). Obično su rasponi algoritma (xg_min, xg_max)=(-5,10), (yg_min, yg_max)=(-5,10) a rasponi definirani zaslonom (x_outmin, x_outmax)=(0,319) i (y_outmin, y_outmax)=(0,199) pri rezoluciji 320x200.

Osnovna razlika između generiranja ovog fraktala i npr. Mandelbrotovog je u tome što ovdje svakom iteracijom dobijemo jednu točku koju bojimo, dok smo kod Mandelbrota ispitivali da li niz divergira i tek nakon što smo "odvrtili" određen broj koraka, dobili smo informaciju o točci. U našem primjeru jednom kada "odvrtimo" iteraciju, fraktal je gotov! To za sobom povlači i daleko veću brzinu generiranja ovog fraktala.

18. PREGLED OSNOVNIH UREĐAJA VEZANIH UZ RAČUNALA I SLIKU

18.1. UVOD

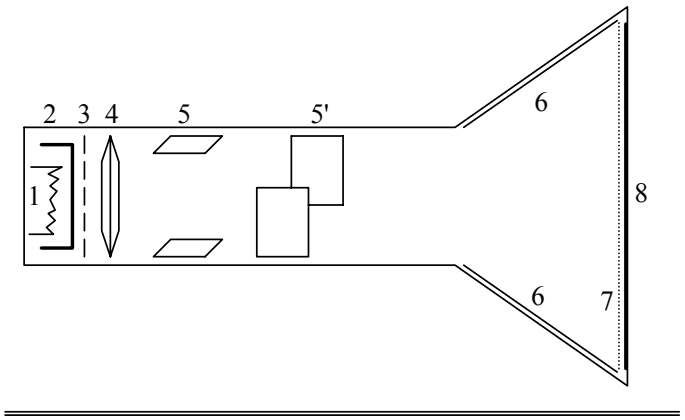
Budući da ova skripta govori o računalnoj grafici, skripta ne bi bila potpuna kada ne bismo spomenuli i osnovne uređaje vezane uz prikaz i unos slike u računalo. Pri tome mislim prvenstveno na monitore kao uređaje na kojima se prikazuje slika, na printere koji sliku ispisuju na papir te na skenere koji omogućavaju unos slike pohranjene na papiru u računalo. U ovom poglavlju osvrnuti ćemo se na vrste tih uređaja i osnovne principe rada.

18.2. MONITORI

18.2.1. OSNOVE RADA KATODNE CIJEVI (CRT)

Osnova rada prvih monitora, a danas i velike većine monitora jest upravo katodna cijev. Katodna cijev je uređaj koji se sastoji od nekoliko osnovnih dijelova:

1. Žarna nit
2. Katoda
3. Kontrolna mrežica
4. Sustav za fokusiranje
5. Sustav za vodoravni i okomiti otklon
6. Anoda (metalizacija)
7. Maska
8. Fosfor naparen na staklo



Katodna cijev

Žarna nit se utjecajem struje užari, te užari katodu. Pod utjecajem visoke temperature dolazi do termičke emisije elektrona iz katode. Ovi emitirani elektroni se propuštaju kroz kontrolnu mrežicu te se sustavom za fokusiranje oblikuju u zraku elektrona. Zatim na zraku djeluje sustav za vodoravni i okomiti otklon čime se precizno određuje smjer zrake, i kao posljedica toga i točka gdje će zraka udarati u staklo zaslona katodne cijevi. Nakon što je tako određen smjer zrake, zraka prolazi pokraj anode koja se nalazi na visokom pozitivnom naponu (15 do 20 kV), i koja dodatno ubrzava elektrone u zraci (povećava ime kinetičku energiju). Konačno zraka prolazi kroz masku koja se nalazi ispred samog zaslona i udara u sloj fosfora naparenog na unutrašnju stranu stakla zaslona. Prilikom ovog sudara elektroni predaju svoju kinetičku energiju fosforu koji ulazi u pobuđeno stanje i tu energiju isijava putem emisije fotona; time se generira svjetlost u točci pogođenoj zrakom. Koliki će biti intenzitet ovog zračenja ovisi o kinetičkoj energiji koju elektroni iz mlaza predaju fosforu – što je veća kinetička energija, veći će biti intenzitet emitirane svjetlosti. Mrežica koja se nalazi pred staklom osigurava da zraka tijekom svog putovanja ne pogađa bilo gdje, već isključivo u ona mjesta gdje je naparen fosfor; naime, ukoliko zraka pogodi tijelo mrežice, mrežica je od vodljivog materijala i svi elektroni biti će apsorbirani, te neće pogađati u staklo iza mrežice. Kada zraka pogodi fosfor, fosfor će u pobuđenom stanju ostati neko određeno vrijeme, i tijekom tog vremena emitirati će svjetlost. Ovo vrijeme naziva se **perzistencija**, i poželjno je da je što kraće, kako bi se na monitoru mogla prikazivati i slika koja se brzo mijenja.

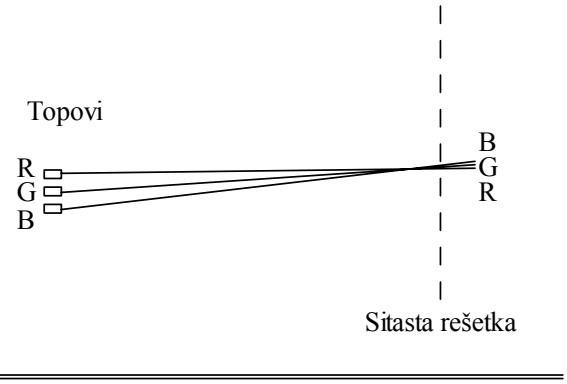
Jedan tip fosfora nakon uzbude emitira svjetlost točno određene valne duljine – odnosno boje. Ukoliko želimo prikaz u boji, tada se umjesto jednog tipa fosfora koriste tri različita tipa, koji se nanose ili u obliku tri točkice (dots) ili u obliku tri linije (stripes), koja emitiraju tri različite boje – crvenu, zelenu i plavu. Isto tako postoje tri topa koja generiraju tri elektronske zrake.

Različite implementacije katodnih cijevi i različiti tipovi mrežica rezultiraju različitim "kvalitetama" prikaza slike. Tako su se razvila dva osnovna tipa: CRT sa FST i CRT sa TRINITRON cijevima, o kojima ćemo bitnije detalje izložiti u nastavku.

18.2.2. CRT SA FST

Kratica FST dolazi od engleskog Flat Square Tube. Ovaj tip cijevi poznat je još i pod nazivom shadow mask cijev. Radi se o cijevima kod kojih je prednja površina dio plašta velike kugle. Naravno, da bi zaslon djelovao ravno, bitno je da radijus zakrivljenosti te kugle bude što veći. Za današnje cijevi ovaj radijus iznosi desetke metara. Prednja maska izvedena je kao sito, dakle sa nizom rupica koje osiguravaju precizno pogađanje fosforne točkice u fosforescentnom premazu stakla. Nepovoljno kod ovakve izvedbe maske je vrlo velika apsorpcija elektrona i kada to nije poželjno. Naime, maska uvijek apsorbira i do 80% elektrona iz mlaza, što dovodi do toga da svega 20% elektrona pogađa fosfor i uzrokuje emisiju svjetlosti. Kod ovog tipa cijevi topovi se izvode u dvije inačice: prva je delta raspored, a druga on-line raspored (raspored u liniji).

Raspored topova

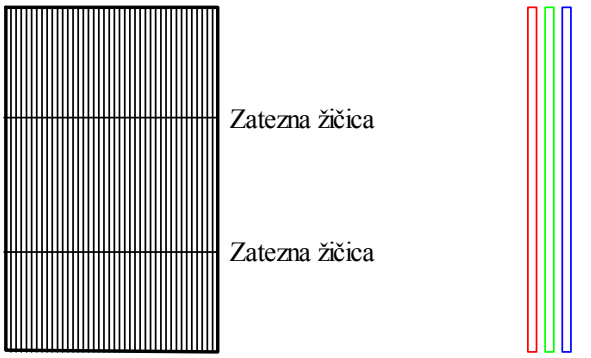


Prikaz topova i sitaste rešetke kod FST-a

18.2.3. CRT SA TRINITRON CIJEVI

Ovaj tip cijevi razvila je tvrtka Sony i u prvo vrijeme bila je korištena isključivo u njihovim monitorima, no s vremenom prihvatili su je i drugi proizvođači (i to u boljim modelima monitora). Karakteristika ovih cijevi je prednja površina koja nije izvedena kao dio kugle već kao dio plašta valjka. Na ovaj način automatski je uklonjena zakrivljenost u vertikalnom smjeru, a preostala zakrivljenost u horizontalnom smjeru savladava se velikim radijusima zakrivljenosti. Kao rezultat ovoga nastale su FD (flat display) trinitron cijevi kod kojih radijus horizontalne zakrivljenosti iznosi čak 50 metara.

Trinitron cijevi ne koriste sitastu masku kao što je to slučaj sa FST cijevima, već kao masku koriste velik broj vertikalno postavljenih tankih žičica koje su učvršćene samo na dva kraja: na vrhu i na dnu cijevi. Uslijed zagrijavanja i sudara sa elektronskom zrakom ove žičice mogu početi titrati, pa da bi se to spriječilo, žičice su dodatno učvršćene sa još jednom ili dvije horizontalne niti (damper wires). Nažalost, te zatezne žičice onemogučavaju prolaz zrake na tim mjestima, pa se prilikom prikaza velikih površina intenzivne boje na zaslonu može primjetiti lagano zatamnjenje na tim mjestima. Fosforni premaz kod ovih je cijevi nanešen u obliku okomitih linija (stripes) budući da i maska ostavlja okomite proreze. Ovom konstrukcijom maske dopušta se daleko manja apsorpcija elektrona od strane maske nego što je to bio slučaj kod FST-a, te je slika na monitorima jačeg intenziteta, svjetlija i kontrastnija.



Maska i raspodjela fosfora kod TRINITRON cijevi

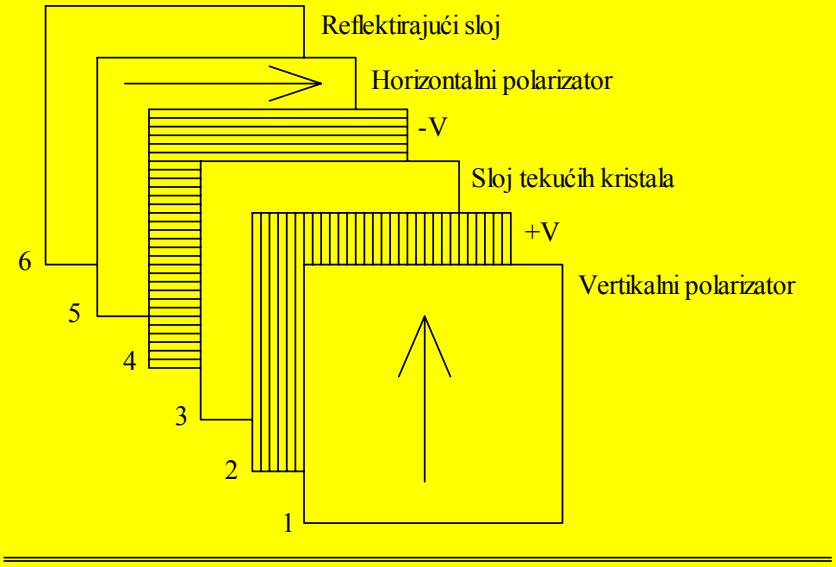
18.2.4. CROMACLEAR

CromaClear još je jedna nova tehnologija o kojoj ćemo reći samo nekoliko osnovnih stvari. Tehnologija je osmišljena u tvrtci NEC (Nippon Electric Company), a sama ideja nije zapravo ništa novo. Radi se o spajanju FST i TRINITRON tehnologija te pokušaju da se iz obje tehnologije izvuče samo najbolje. Pri tome se koristi sitasta maska kao što je to slučaj kod FST-a, međutim otvori nisu jednoliki već su izduženog heksagonalnog oblika, čime se osigurava veća površina kroz koju može proći zraka. Fosforni premaz izveden je prema TRINITRON receptu, dakle kao niz okomitih linija (stripes). Rezultat je vrlo kontrastna i svijetla slika uz dodatno bolje fokusiranje elektronske zrake koje je posljedica uporabe sitaste maske.

18.2.5. LCD ZASLONI

Kratica LCD označava zaslon sa tekućim kristalima (Liquid Crystal Display). Ovdje se koristi svojstvo tekućih kristala koje omogućava promjenu polarizirajućih svojstava pod utjecajem električnog polja. Naime, ukoliko nema električnog polja, tekući kristali zakreću ravninu polarizacije svjetlosti za 90°, dok pod utjecajem električnog polja ovaj efekt nestaje. Presjek jednog LCD-a prikazan je na slici 18.2.5.1. To je struktura koja se sastoji od 6 slojeva. Prvi sloj koji se nalazi na putu svjetlosti je vertikalni polarizator. Ambijentna svjetlost (svjetlost iz okoline) titra u svim smjerovima okomito na pravac svog širenja; kažemo da takva svjetlost nije polarizirana. Prolaskom kroz vertikalni polarizator titranja u svim pravcima osim u okomitom se guše. Svjetlost koja prođe kroz taj polarizator titra isključivo, slikovito govoreći, gore – dolje. Slijedi prozirni sloj na kojem se nalaze vertikalno položene žičice koje se spajaju na pozitivni potencijal. Treći sloj je sloj tekućih kristala koji se izvodi kao vrlo tanak. Ovaj sloj zakreće ravninu polarizacije svjetlosti za devedeset stupnjeva, ili ne, ovisno o prisustvu električnog polja. Četvrti sloj je proziran sloj na kojem se nalaze horizontalne žičice koje se spajaju na negativan potencijal. Peti sloj čini horizontalni polarizator koji će propustiti samo one komponente svjetlosti koje titraju u horizontalnom

smjeru, dakle, lijevo – desno. Konačno posljednji šesti sloj je reflektirajući sloj koji svjetlost reflektira natrag.



Slika 18.2.5.1. Prikaz rada LCD-a

Vertikalne i horizontalne žičice omogućavaju da se za svaku točku zaslona na njihovom sjecištu stvori električno polje, ili ne, ovisno o tome kakva ta točka mora biti. Ukoliko u toj točki nema polja, a do kristala na tom mjestu dopire svjetlost koja je prethodno vertikalno polarizirana, kristali će okrenuti smjer titranja za 90°; time će svjetlost sloj kristala napustiti titrajući lijevo – desno. Takva svjetlost biti će propuštena kroz horizontalni polarizator, odbiti će se od reflektirajućeg sloja, ponovno će biti propuštena kroz horizontalni polarizator, zakrenuti će se u okomito titranje prolaskom kroz kristale, proći će kroz vertikalni polarizator i izaći će van. Točku ćemo doživjeti kao svjetlu. Ukoliko je u točci prisutno polje, efekt zakretanja neće biti prisutan i na horizontalni polarizator doći će vertikalno polarizirana svjetlost. Naravno, ovo će rezultirati gašenjem te svjetlosne zrake, i iz točke natrag neće izaći ništa – dobiti ćemo crnu točku.

LCD ima mnoge prednosti pred klasičnim CRT izvedbama zaslona: izuzetno su lagani i tanki i troše vrlo malo energije. Nažalost, ukoliko ste dobro shvatili opisani mehanizam rada, jasno Vam je da za prikaz slike LCD-i trebaju vanjski izvor svjetlost – oni ne stvaraju svjetlost kao što to rade CRT-ovi. Ovo je dakako mana, a ima ih još: brzina promjene slike ne može biti velika jer kristali sporo prelaze iz jednog moda rada u drugi, ograničen je kut pod kojim se vidi slika i zasloni su vrlo osjetljivi na pritisak.

18.2.6. ZASLONI S PLAZMOM

Ovaj tip zaslona opisati ćemo u samo nekoliko riječi. Na mjestu ukrštavanja elektroda, dakle na mjestu na kojem postoji mogućnost prikazivanja točke nalazi se plin XeNe. Pod utjecajem električnog polja (dakle kada želimo prikazati točku) dolazi do ionizacije plina i

dolazi do UV zračenja. Ovo UV zračenje pogađa atome fosfora koji prelaze u pobuđeno stanje i sami počinju zračiti vidljivu svjetlost.

Dobre strane ove tehnologije su mogućnost izrade zaslona velikih dijagonala (čak do 100 inča), te aktivan prikaz slike (za razliku od LCD-a koji je pasivan jer treba vanjsku svjetlost za svoj rad). Loša strana leži u velikoj težini ovi zaslona (do 100 kg).

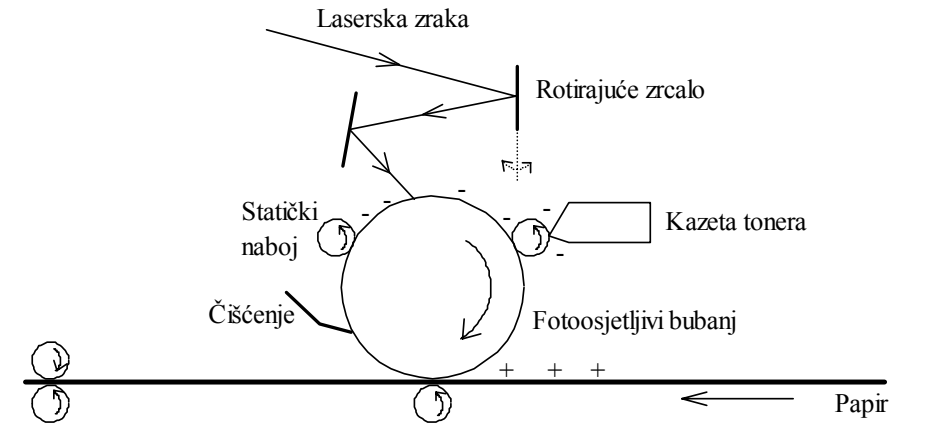
18.3. PISAČI

18.3.1. OPĆENITO

Danas postoji nekoliko vrsta pisača, od kojih ću spomenuti matrične, laserske, tintne i termo pisače. Danas su najpopularniji upravo laserski i tintni pisači, pa ćemo o njima reći nekoliko riječi i upoznati se sa osnovnim načelima rada. Pregled će biti više informativan nego detaljan jer na ovom nivou nema potrebe ulaziti u sve pojedinosti.

18.3.2. LASERSKI PISAČI

Osnovni mehanizam rada laserskog pisača prikazan je na slici 18.3.2.1. Bubanj mehanizma nabijen je negativnim nabojem. Mjesto na kojem laserska zraka pogodi bubanj pod utjecajem svjetlosti se izbije, i taj dio površine bubnja postane neutralno nabijen. Laserska zraka pri tome po bubnju iscrtava redak po redak slike koju treba ispisati. Za horizontalni hod zrake brine se rotirajuće zrcalo. Nakon što zraka iscrta dio slike na bubnju, taj dio dolazi do valjka na kojem se nalazi toner. Toner je također negativno nabijen, što znači da se neće prihvaćati na bubanj, osim na ona mjesta koja su neutralizirana laserskom zrakom. U ovom koraku toner tvori sliku na bubnju. Daljnjom rotacijom bubnja toner dolazi do papira koji je prethodno nabijen pozitivnim nabojem. Budući da je naboj tonera negativan, sav toner sa bubnja prilijepiti će se za papir i time će na papiru nastati slika. Međutim, ovako nastala slika vrlo je nepostojana. Zbog toga se papir naknadno pritišče i zagrijava što dovodi do fuzije tonera u papir. Na ovaj način dobiva se kvalitetna i postojana slika.



Slika 18.3.2.1. Princip rada laserskog pisača

18.3.3. Tintni pisači

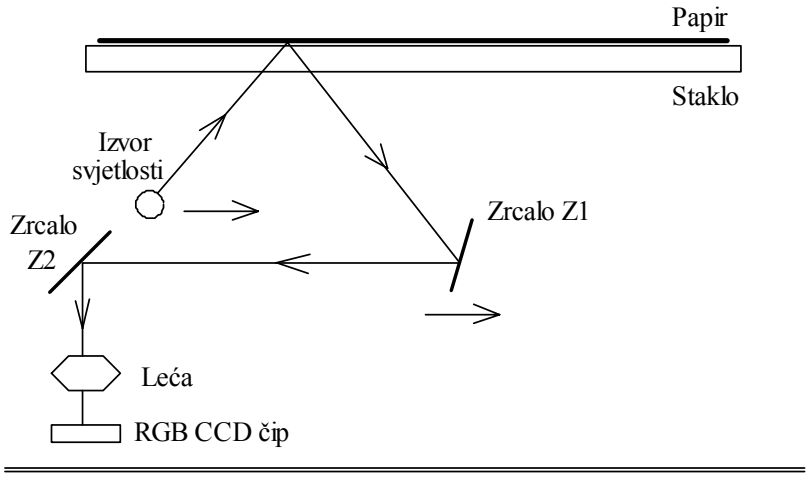
Tintni pisači zasnovani su na ideji izbacivanja što većeg broja što manjih kapljica tinte što većom brzinom. Razlikujemo tri osnovne tehnologije kojima se ovo ostvaruje: BubbleJet tvrtke Canon, Piezo tvrtke Epson te Thermal Ink Jet tvrtke HP. Iako su ovo tri različite tehnologije, načini rada BubbleJet tehnologije i Thermal Ink Jet dosta su slični pa ćemo ih opisati zajedno.

Po nekim pričama najstarija tehnologija – termička ink-jet – otkrivena je sasvim slučajno kada je neki znanstvenik proučavao princip rada aparata za kavu. Ideja je da se tinta sustavom kanala dovodi u komoru u kojoj je izveden grijač velike snage. Kratkotrajni strujni impuls izazvati će trenutno zagrijavanje grijača i dovesti do vrenja tinte koja će ispuniti cijelu komoru i stvoriti nadtlak koji će istisnuti malu količinu tinte kroz mlaznicu (ovisno o izvedbi ova količina iznosi između 3 i 6 pikolitara). U međuvremenu kratkotrajni impuls koji je izazvao zagrijavanje grijača već je prestao pa se grijač a time i tinta brzo hlade i ponovno prelaze u tekuće stanje. U tom trenutku u komori nastaje podtlak (jer je dio tinte istisnut) i taj podtlak uvlači u komoru novu količine tinte iz spremnika, nakon čega se postupak može ponoviti. U cjeloj priči postoji samo jedan bitan nedostatak – ograničenje brzine rada zbog vremena koje je potrebno da se grijač i tinta ohlade.

Za razliku od termalne tehnologije koja koristi termičku ekspanziju tinte za postizanje nadtlaka koji izbacuje kapljicu tinte kroz mlaznicu, Piezo tehnologija koristi svojstvo određenih kristala da se pod utjecajem električnog polja šire ili stežu čime mijenjaju svoj volumen. Tinta se dovodi u komoru u kojoj postoji ovakav kristal, i zatim se na kristal dovodi strujni impuls. Rezultat je trenutna deformacija kristala i nastajanje nadtlaka koji izbacuje kapljicu tinte kroz mlaznicu. Nestankom strujnog impulsa kristal se vraća u prvobitno stanje, a budući da je dio tinte izbačen iz komore, u komori nastaje podtlak koji uvlači novu količinu tinte iz spremnika, i proces se opet može ponavljati. Ova tehnologija omogućava veće brzine izbacivanja tinte i regulaciju količine tinte koja se izbacuje u kapljici.

18.4. SKENERI

Skener je uređaj koji nam omogućava da sliku sa papira prenesemo na računalo. Osnovni mehanizam skenera prikazan je na slici 18.4.1.



Slika 18.4.1. Princip rada skenera

Izvor svjetlosti fiksiran je sa zrcalom Z1, i oni zajedno putuju korak po korak sa lijeve strane prema desnoj strani. Zraka svjetlosti od izvora se širi prema papiru kojega "skeniramo", reflektira se od njegove površine, zatim se reflektira od zrcala Z2, prolazi kroz leću i dolazi do CCD čipa (CCD je kratica od Charge Coupled Device). CCD čip sadrži niz fotoelektričnih elemenata koji ovisno o osvjetljenju generiraju analogne signale. Ovi analogni signali zatim se pretvaraju u digitalne A/D pretvornicima. Preciznost A/D pretvornika uobičajeno može biti 8, 10, 12 ili 16 bitova. Za svaku točku papira koja se skenira dobiju se tri analogna signala (tri kanala) koji govore o količini crvene, zelene i plave komponente. Sva tri kanala pretvaraju se na A/D pretvornicima pa za jednu točku dobijemo 24, 30, 36 ili 48 bitnu informaciju o boji (tri kanala puta broj bitova jednog A/D pretvornika).
Prilikom skeniranja slika koje su nastale kao rezultat tiska, printanja ili nekog sličnog postupka, dolazi do nekih nepoželjnih pojava na koje treba obratiti pažnju. Jedna od njih je Moiré efekt. Radi se o tome da je svaka slika nastala ovim postupcima zapravo sastavljena od niza sitnih točkica koje su dovoljno blizu da ih golim okom ne primijetimo. Međutim, raspored tih točkica je pravilan, a rezultat je načina na koji je slika ispisana. Oblik ovih točkica je najčešće eliptičan, i organiziran u vjenčiće sastavljene od četiri osnovne boje koje se koriste pri ispisu CMYK. Činjenica da golim okom ovo ne vidimo ne znači ništa, jer skener ove točkice jasno raspoznaje. I tu je problem. Naime, slika koje se dobije izgleda grozno, jer umjesto boja kakve mi vidimo imamo cijelo čudo šarenila. Ovaj problem može se ukloniti odgovarajućim programima nakon što se slika skenira.

19. BOJE

19.1. UVOD

Što su boje? Kako čovjek vidi boje? Kako možemo izmjeriti "boje"? Ovo su samo neka od pitanja na koje ćemo pokušati dati odgovor kroz ovaj tekst. Boje su prvenstveno vezane uz pojam svjetlosti. Čovjek vidi predmete oko sebe zahvaljujući razvijenim senzorima za elektromagnetske valove u području valnih duljina koje nazivamo vidljiva svjetlost. Ovo područje obuhvaća valne duljine od 390 do 760 nm, odnosno frekvencijski spektar između 394.7 i 769.2 THz. Kod čovjeka postoje dvije vrste senzora koji su se specijalizirali za dvije različite namjene, vjerojatno kao prilagodba na čovjekovu okolinu. Naime, čovjek živi, kada o svjetlosti govorimo, u dva različita okruženja: danu i noći.

Noć je karakteristična po tome što svjetlosti ima vrlo malo, odnosno vrlo je slabog intenziteta. U takvom okruženju bitno je raspoznavati osnovne elemente okoline; dakle, bitno je isključivo razaznavati različite intenzitete svjetlosti, i na temelju toga raspoznavati objekte koji ga okružuju. U ovakvoj okolini nema dovoljno informacija, a niti potrebe, za raspoznavanjem boje. I upravo za ovakvo okruženje razvila se je prva vrsta osjetila – štapići. To su osjetila koja raspoznaju različite intenzitete svjetlosti, i mogu "raditi" već pri vrlo malim količinama svjetlosti.

Za razliku od noći, tijekom dana raspoloživa količina svjetlosti je vrlo velika. Toliko velika, i na toliko različitih valnih duljina, da se na temelju te količine svjetlosti može dobiti puno više informacije od običnog intenziteta – možemo raspoznavati boju! Za ovaj režim rada razvijen je drugi tip osjetila – čunjići. Pomoću tih osjetila čovjek je sposoban primati informaciju o boji predmeta iz okoline. U čovjekovom oku postoje tri tipa ovih osjetila: osjetilo za crvenu boju, osjetilo za zelenu boju te osjetilo za plavu boju. Naravno, odmah se postavlja pitanje kako onda vidimo npr. žutu boju, kada za nju nemamo osjetila. Odgovor na ovo pitanje dati ćemo u nastavku.

19.2. SHEME ZA PRIKAZ BOJA – PROSTORI BOJA

19.2.1. MODELI

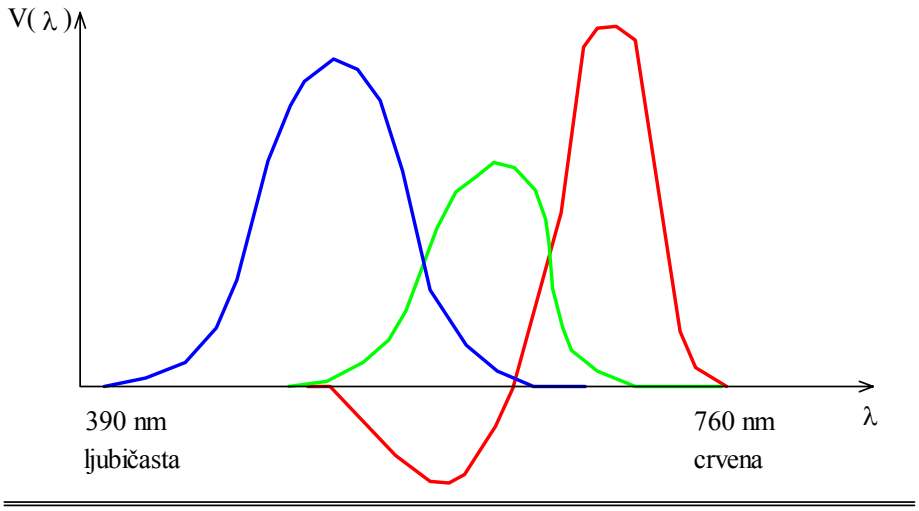
Tijekom proučavanja svjetlosti i problematike vezane uz boje razvijeno je nekoliko modela kojima se pokušava dobiti kontrola nad bojama. Prvi i osnovni model proizašao je iz otkrića kako čovjek vidi boje. Tako je nastao RGB model (**R**ed, **G**reen, **B**lue). Teorijskim razmatranjima razvijen je XYZ model. Iz tehničkih razloga razvijen je CMY (**C**yan, **M**agenta, **Y**ellow), a potom i CMYK (**C**yan, **M**agenta, **Y**ellow, blac**K**). Radi lakoće odabiranja boja nastao je HSV model (**H**ue, **S**aturation, **V**alue). Razvijeni su još i modeli razreda Y (class Y models) kao rezultat razvitka TV i video tehnike, i drugi.

19.2.2. RGB MODEL

RGB model proizašao je iz pokušaja da se imitira način na koji čovjek vidi boje. Naime, čovjek ima osjetila za tri osnovne (primarne) boje: crvenu, zelenu i plavu. Uzevši tu činjenicu u obzir, pretpostavilo se je da se sve boje mogu prikazati kao linearna kombinacija ovih triju osnovnih boja. Na ovaj način, da bismo pamtili boju jedne točke, trebamo pamtiti tri broja: koliko imamo crvene, koliko imamo zelene, te koliko imamo plave. Ako sve tri komponente iznose nula, dobiti ćemo crno. Ako sve tri komponente iznose maksimum, dobiti ćemo bijelo. Ukoliko su pojedine komponente prisutne sa različitim udjelima, dobiti ćemo različite boje. Nijanse sivih boja dobivati ćemo ukoliko sve tri osnovne boje držimo jednake po iznosu, i taj iznos variramo. Fizikalno si ovaj proces možemo zamisliti na slijedeći način: imamo na raspolaganju tri svjetlosna izvora: izvore crvene, zelene i plave. Sva tri izvora tuku u istu točku. Ukoliko su sva tri izvora ugašena, točka je crna jer nema nikakve svjetlosti koja bi je obasjala. Počnemo li zatim paliti pojedine izvode, točka će poprimati različite boje jer će doći do miješanja boja. Zbog ovog svojstva da se pojedine komponente svjetlosti zbrajaju, ovo miješanje zovemo aditivno miješanje. Proces je prikazan na slici 19.2.2.1.

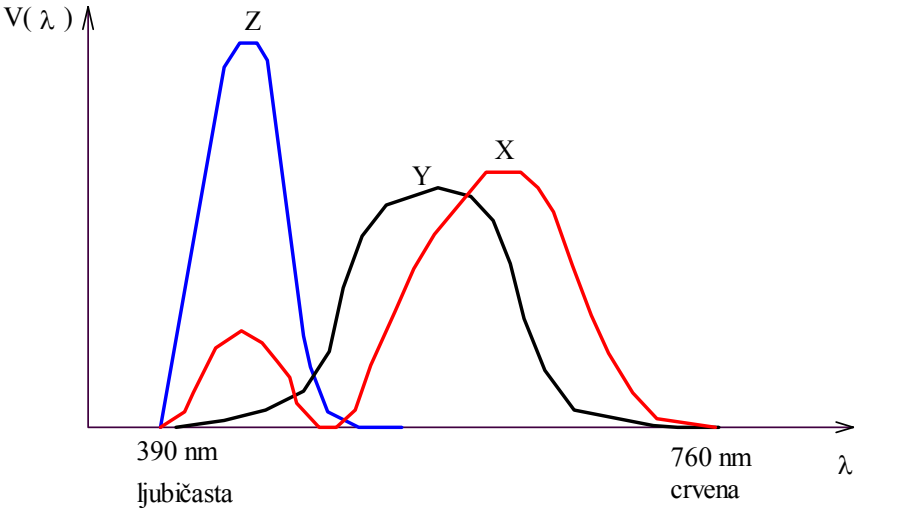
Slika 19.2.2.1.

Opisani model vrlo je jednostavan. Međutim, ima jednu manu. Sve boje ne mogu se opisati pomoću ovih triju primarnih boja, barem ne ako imamo u vidu fizikalnu sliku miješanja boja. Naime, pokazuje se da bi se sve boje mogle prikazati kada bi crveni izvor, osim emitiranja crvene svjetlosti mogao i oduzimati crvenu svjetlost – iz ničega. Naime, za prikaz svih boja, koeficijent crvene boje potezao bi se i u pozitivnom području, i u negativnom području, kao što to prikazuje slika 19.2.2.2.



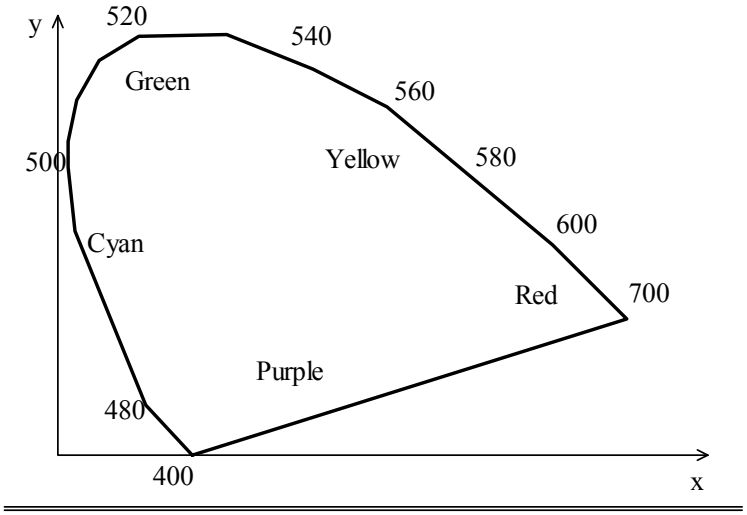
Slika 19.2.2.2.

Ovaj nedostatak rezultirao je sastankom CIÉ (Commission Internationale de l'Éclairage) gdje se je pokušao definirati model koji bi također imao samo tri primarne boje X, Y i Z, i koji bi opisivao sve boje uz pozitivne koeficijente. Rezultat je prikazan na slici 19.2.2.3. :



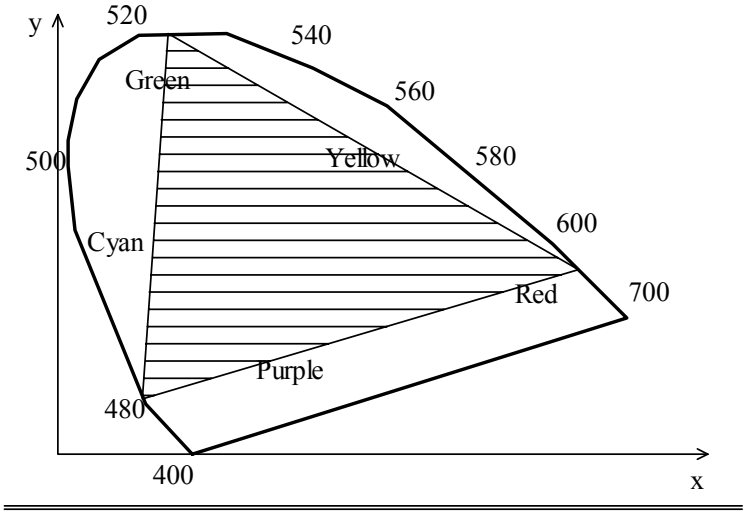
Slika 19.2.2.3.

CIÉ je također definirala kromatski dijagram koji opisuje sve boje. Dijagram je prikazan na slici 19.2.2.4.



Slika 19.2.2.4.

Ovaj dijagram ima veliku važnost, utoliko što omogućava da bolje shvatimo našu ograničenost u prikazivanju boja. Naime, ukoliko imamo na raspolaganju tri izvora boja (npr. izvore crvene, zelene i plave boje), boje koje možemo prikazati u kromatskom se dijagramu nalaze unutar trokuta koji zatvaraju te tri boje (slika 19.2.2.5.). Ovaj trokut naziva se GAMUT.



Slika 19.2.2.5.

Boje koje se nalaze izvan tog trokuta, ne mogu se prikazati kombinacijom tih triju boja! Ako pokušate umjesto sa tri izvora uporabiti četiri – ništa se ne mijenja; lik sada neće

biti trokut, već četverokut, i površina će biti nešto veća. No opet će biti boja koje ne možemo prikazati!

Model RGB koristi se u uređajima u kojima je moguće ostvariti tri izvora boja i njihovo miješanje, na način kako smo to opisali. To u praksi znači da će se model koristiti prvenstveno u monitorima. Općenito vrijedi: trokut koji razapinju boje topova u monitoru naziva se GAMUT uređaja.

19.2.3. CMY/CMYK MODEL

Model CMY slijedi upravo obratnu filozofiju od RGB modela. Pretpostavka je da bez ikakvih utjecaja na točku boja točke mora biti bijela. Boje ćemo dobiti tako da od bijele oduzimamo pojedine komponente u različitim iznosima. Osnovne boje koje se ovdje koriste su **C**yan (cijan), **M**agenta (neka verzija ljubičaste) i **Y**ellow (žuta). Ovo je primjer subtraktivnog modela, i to je komplementarni model RGB modelu. Štoviše, ukoliko normiramo pojedine komponente RGB modela (dakle, svake boje ima u iznosu od 0 do maksimalno 1), tada vrijedi relacija:

```
C=1-R
M=1-G
Y=1-B
```

Slika 19.2.3.1. prikazuje miješanje ovih osnovnih boja.

Slika 19.2.3.1.

Uvjet da je točka sama po sebi bijela ukoliko na nju ne djeluje ništa nalazimo upravo kod bijelog papira – zbog toga se ovaj model koristi upravo kod printera. Zapravo, kod printera se koristi CMYK model, koji je ekonomiziran CMY model. Naime, teorijske pretpostavke i praksa obično se baš i ne slažu najbolje, pa tako maksimalnim miješanjem boja CMY dobivamo crnu koja baš i nije onako crna kako bismo to željeli. S druge strane, većina dokumenata koja se danas još uvijek ispisuje obilato koristi upravo crnu boju, što za ovaj

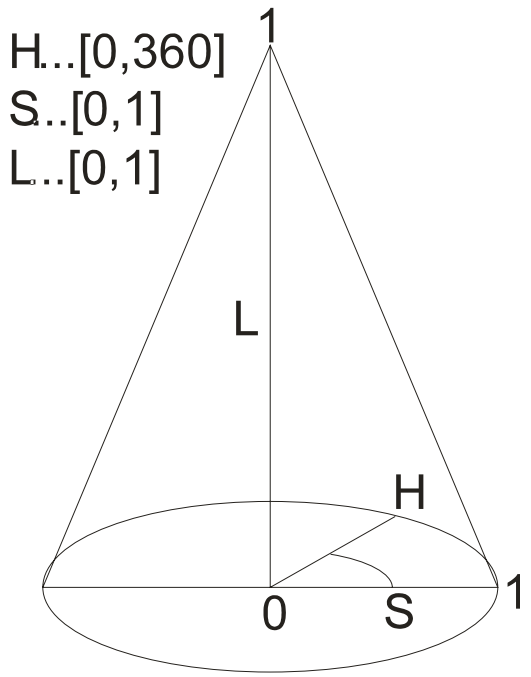
model znači maksimalnu potrošnju svih boja – a uopće ne ispisujemo u boji. Zbog tih razloga, u model je uvedena još i crna boja, te je model dobio u oznaci slovo k od riječi blac**K**. Tako se uvode slijedeće relacije:

```
K=min(C,M,Y)
C=C-K
M=M-K
Y=Y-K
```

19.2.4. HLS MODEL

HLS model (Hue, Lightnes, Saturation) nastao je iz potrebe da se olakša odabir boje. Naime, prethodno opisani modeli vrlo su nezgrapni kada je riječ o odabiru boje. Zamislite da ste našli kombinaciju koja vam daje neku nijansu narančaste boje, i sada želite dobiti samo malo zasićeniju boju, ili pak malo svjetliju. Što učiniti da bi se to postiglo? Problem je u tome što treba mijenjati sve tri komponente RGB modela, a to znači smrt jednostavnoj uporabi. Ideja je da promjenom vrijednosti H obilazimo sve boje. Kada pronađemo odgovarajuću boju, sa L odredimo željenu svjetlinu, te sa S njezinu zasićenost.

Komponente H, L i S čine stožac, prema slici 19.2.4.1.



Slika 19.2.4.1.

H je predstavljen kutom u odnosu na pozitivnu x os, i njime se biraju boje. Vrijedi slijedeće:

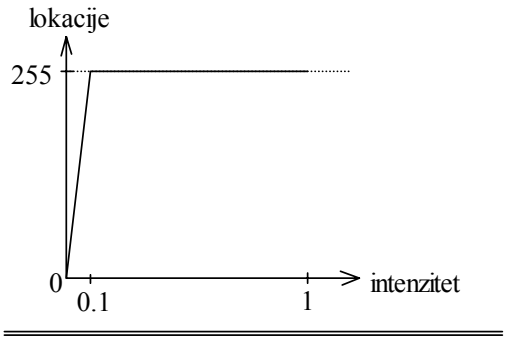
H vrijednost	Odgovarajuća boja
0°	Plava
60°	Magenta
120°	Crvena

180°	Žuta
240°	Zelena
300°	Cijan

Ukoliko zasićenost stavimo na nulu (S=0), tada promjenom svjetline (L) od 0 do 1 dobivamo sve nijanse sive boje, uključujući crnu (L=0), i bijelu (L=1).

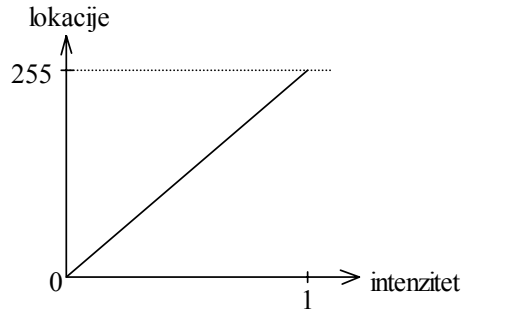
19.3. GAMMA KOREKCIJA

Zamislimo da se nalazimo pred slijedećim problemom: imamo na raspolaganju 256 lokacija na kojima možemo pamtiti intenzitet boje. Intenzitet je u rasponu od 0 do 1 (0 je minimalni, 1 maksimalni). Potrebno je odrediti koliki intenzitet ćemo pohraniti u pojedinu lokaciju, a da pri tome dobijemo takvu raspodjelu koja će biti uočljiva ljudskom oku. Npr. intenzitete možemo raspodijeliti prema slici 19.3.1.



Slika 19.3.1

Lokacija 0 sadržavati će intenzitet 0, lokacija 255 intenzitet 0.1, dok će intenziteti između 0 i 0.1 biti linearno pridjeljeni lokacijama 1 do 254. Međutim, ovakva raspodjela nije dobra jer ćemo cijelu sliku nacrtanu pomoću ovakvih intenziteta jedva vidjeti – naime, intenziteti su raspodijeljeni kao da je na snazi opća opasnost i metode zamračivanja. To nije dobro. Možda je bolja raspodjela prikazana na slici 19.3.2.



Slika 19.3.2

Tu je intenzitet također linearno raspodijeljen, i to tako da se u pojedinim lokacijama nalaze i najmanji, a u pojedinim i najveći intenziteti. To je, dakako, bolje. Međutim... Ljudsko oko intenzitete ne raspoznaje baš na ovaj način. Naime, da bi oko primijetilo da se je intenzitet povećao, to povećanje mora biti za potenciju veće! Naime, karakteristika ljudskog oka je logaritamska! To znači da oko primijeti intenzitet koji je veći za 1 tek kada logaritam tog intenziteta poraste za jedan, a logaritam poraste za jedan samo ako potencija poraste za jedan! Dakle, umjesto linearne raspodjele intenziteta sa slike 19.3.2:

$$
\begin{aligned}
&I_0 = I_0 \\
&I_1 = r \cdot I_0 \\
&I_2 = (r + r) \cdot I_0 \\
&\ldots \\
&I_n = (n \cdot r) \cdot I_0
\end{aligned}
$$

treba napraviti eksponencijalnu raspodjelu intenziteta:

$$
\begin{aligned}
&I_0 = I_0 \\
&I_1 = r \cdot I_0 \\
&I_2 = r^2 \cdot I_0 \\
&\ldots \\
&I_n = r^n \cdot I_0
\end{aligned}
\qquad \text{(rel. 19.3/1)}
$$

Ujedno znamo da je I_n upravo jednak 1, pa možemo izračunati faktor r:

$$
I_n = r^n \cdot I_0 = 1 \Rightarrow r = \left(\frac{1}{I_0} \right)^{\frac{1}{n}} \qquad \text{(rel. 19.3/2)}
$$

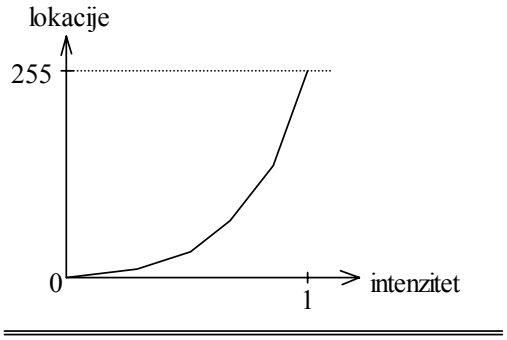
U općem slučaju se, dakle, može napisati formula:

$$I_j = r^j \cdot I_0 = \left(\left(\frac{1}{I_0} \right)^{\frac{1}{n}} \right)^j \cdot I_0 = (I_0)^{1-\frac{j}{n}} \qquad \text{(rel 19.3/3)}$$

Kako u našem slučaju imamo 256 razina, odnosno razine od 0 do 255, vrijedi:

$$r = \left(\frac{1}{I_0} \right)^{\frac{1}{255}}, \qquad I_j = (I_0)^{1-\frac{j}{255}}, \qquad j = 0,1,...,255$$

Ovakva raspodjela prikazana je na slici 19.3.3.



Slika 19.3.3

Pogledajmo sada situaciju sa klasičnim CRT monitorom. Osvjetljenje neke točke na zaslonu monitora rezultat je pogađanja te točke (odnosno fosfora u toj točci) zrakom elektrona. Intenzitet tako dobivene svjetlosti to je veći, što više elektrona pogađa tu točku u jedinici vremena. Matematički se ovisnost intenziteta može opisati relacijom:

$$I = k \cdot N^{\gamma} \qquad \text{(rel 19.3/4)}$$

I...intenzitet koji emitira fosfor
k...konstanta
γ...konstanta

Broj elektrona proporcionalan je naponu koji uzrokuje pojavu, te vrijedi:

$$I = k \cdot (k_1 \cdot V)^{\gamma} = k \cdot k_1^{\gamma} \cdot V^{\gamma} = K \cdot V^{\gamma} \qquad \text{(rel 19.3/4)}$$

I...intenzitet koji emitira fosfor
k,k_1,K...konstanta
γ...konstanta

Ukoliko je poznat intenzitet koji želimo dobiti, napon koji je potreban za njegov prikaz dobiti ćemo izjednačavanjem relacija 19.3/1 i 19.3/4:

$$K \cdot V_j^{\gamma} = r^j \cdot I_0 \Rightarrow V_j = \left(r^j \cdot \frac{I_0}{K} \right)^{\frac{1}{\gamma}}$$

Treba uočiti da je γ karakteristika samog monitora, pa će zbog toga na različitim monitorima odgovarajući naponi biti različiti – ili pak isti, ali tada će prikazane slike biti različite, a to ipak ne želimo.

19.4. PAMĆENJE BOJA NA RAČUNALU

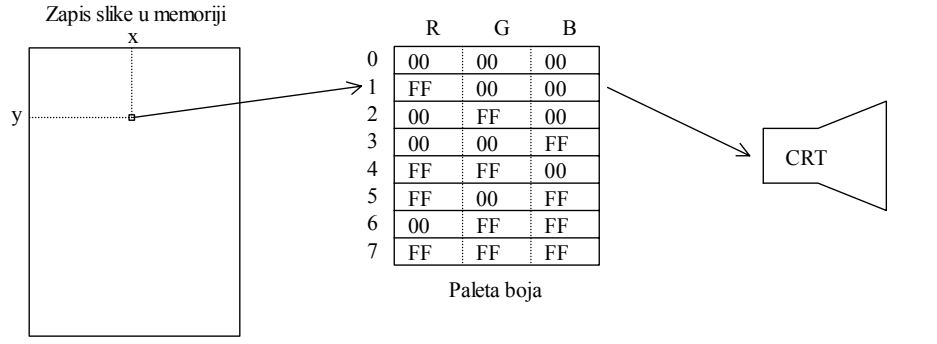
Rad sa bojama na računalima izveden tako da se može prilagoditi potrebama i memorijskim zahtjevima, odnosno raspoloživim memorijskim resursima samog računala. Osnovni način prezentacije boje na računalu je putem RGB sustava. Dakle, za svaku boju pamte se tri komponente: crvena, zelena i plava. Međutim, tu postoje dva moda rada:

- Uporaba paleta boja
- Direktna reprezentacija boja

Prva metoda razvijena je zbog uštede memorije, i time naravno nameće određena ograničenja. Glavni faktor koji određuje koliko ćemo različitih boja moći prikazati naziva se dubina. Ovisnost dubine boja i broja mogućih različitih boja dana je relacijom 19.4/1:

$n = 2^d$ n...broj raspoloživih boja
d...dubina

Dubina se mjeri u broju bitova. Ideja je da se formira jedna tablica koja ima onoliko elemenata koliko mi to odredimo preko dubine. Svaki taj element sadržavati će tri komponente: količinu crvene, količinu zelene te količinu plave boje. Boja pojedinog piksela (npr. boja *j*) određivati će se brojem od 0 do n-1, i odgovarati će onoj boji čija se definicija nalazi u odgovarajućem elementu tablice (dakle, *j*-ti element). Slika 19.4.1. prikazuje kako to radi.



Slika 19.4.1.

Paleta boja definira slijedeće boje:

Indeks u paleti boja	**Boja**
0	Crna
1	Crvena
2	Zelena
3	Plava

4	Žuta
5	Magenta
6	Cijan
7	Bijela

U navedeno primjeru, na koordinata (x,y) nalazi se boja 1. Prije iscrtavanja na zaslonu, gleda se u paletu boja i pronalazi pod brojem 1 definicija crvene boje. Na zaslonu se crta crvena točka. U ovom primjeru tablica je imala 8 elemenata jer je zadana dubina iznosila 3 bita. Ukoliko odaberemo dubinu od 8 bitova, imati ćemo na raspolaganju 256 mogućih boja. Ova metoda dobra je kada nemamo puno memorije na raspolaganju, a niti ne želimo prikazivati true-color slike.

Povećavamo li dubinu do veličine od 24 bita, paleta boja nam se više ne isplati jer paletom boja možemo definirati jednaki iznos boja kao i da direktno definiramo boju. Naime, uz 24 bita po jednom pikselu, bitove možemo grupirati u grupe po 8: 8 bitova za crvenu, 8 bitova za zelenu i 8 bitova za plavu. U tom slučaju više ne koristimo palete, već se za svaki piksel direktno definira boja. Mana ove metode je što zahtjeva enormne količine memorije za pohranu malo veće slike. Metoda je prikazana na slici 19.4.2.

Slika 19.4.2.

Sada na lokaciji (x,y) piše kompletan zapis boje: FF 00 00 što odgovara crvenoj boji.

20. DODATAK A.

PROBLEM I.

Zadan je trodimenzionalni prostor S pomoću tri bazna vektora tog prostora: $\vec{\mathbf{e}}, \vec{\mathbf{f}}$ i $\vec{\mathbf{g}}$. Vektori su jedinični (norma im je 1) te međusobno okomiti. Zadana je točka T sa koordinatama (T_x,T_y,T_z) u tom prostoru. Umjesto prostora S želi se koristiti prostor S' zadan baznim vektorima $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$. Pri tome je poznat zapis vektora $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$ pomoću komponenti u prostoru S. Zapis glasi:

$$\vec{\mathbf{a}} = a_x \cdot \vec{\mathbf{e}} + a_y \cdot \vec{\mathbf{f}} + a_z \cdot \vec{\mathbf{g}}$$

$$\vec{\mathbf{b}} = b_x \cdot \vec{\mathbf{e}} + b_y \cdot \vec{\mathbf{f}} + b_z \cdot \vec{\mathbf{g}} \quad \text{(rel:20/1)}$$

$$\vec{\mathbf{c}} = c_x \cdot \vec{\mathbf{e}} + c_y \cdot \vec{\mathbf{f}} + c_z \cdot \vec{\mathbf{g}}$$

Bazni vektori $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$ također su jedinični i međusobno okomiti.

Potrebno je pronaći koje će koordinate točka T imati gledano iz prostora zadanog baznim vektorima $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$?

Rješenje.

Točka T u prostoru S' imati će zapis pomoću baznih vektora prostora S'. To znači da će proizvoljna točka T imati u prostoru S' zapis:

$$T = \begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} = \lambda \cdot \vec{\mathbf{a}} + \mu \cdot \vec{\mathbf{b}} + \gamma \cdot \vec{\mathbf{c}} \quad \text{(rel:20/2)}$$

Ukoliko u prethodni izraz uvrstimo (rel:20/1), dobiti ćemo:

$$T = \lambda \cdot \left(a_x \cdot \vec{\mathbf{e}} + a_y \cdot \vec{\mathbf{f}} + a_z \cdot \vec{\mathbf{g}}\right) + \mu \cdot \left(b_x \cdot \vec{\mathbf{e}} + b_y \cdot \vec{\mathbf{f}} + b_z \cdot \vec{\mathbf{g}}\right) + \gamma \cdot \left(c_x \cdot \vec{\mathbf{e}} + c_y \cdot \vec{\mathbf{f}} + c_z \cdot \vec{\mathbf{g}}\right)$$

odnosno nakon grupiranja:

$$T = \left(\lambda \cdot a_x + \mu \cdot b_x + \gamma \cdot c_x\right) \cdot \vec{\mathbf{e}} + \left(\lambda \cdot a_y + \mu \cdot b_y + \gamma \cdot c_y\right) \cdot \vec{\mathbf{f}} + \left(\lambda \cdot a_z + \mu \cdot b_z + \gamma \cdot c_z\right) \cdot \vec{\mathbf{g}}$$

(rel:20/3)

Posljednji izraz daje zapis točke T preko baznih vektora $\vec{\mathbf{e}}, \vec{\mathbf{f}}$ i $\vec{\mathbf{g}}$ (dakle zapis točke u prostoru S). No mi već znamo zapis točke T preko baznih vektora $\vec{\mathbf{e}}, \vec{\mathbf{f}}$ i $\vec{\mathbf{g}}$; on glasi:

$$T = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix} = T_x \cdot \vec{\mathbf{e}} + T_y \cdot \vec{\mathbf{f}} + T_z \cdot \vec{\mathbf{g}} \qquad \text{(rel:20/4)}$$

Izjednačavanjem zapisa danih u (rel:20/3) i (rel:20/4) dobije se sustav:

$$\begin{aligned}\lambda \cdot a_x + \mu \cdot b_x + \gamma \cdot c_x &= T_x \\ \lambda \cdot a_y + \mu \cdot b_y + \gamma \cdot c_y &= T_y \\ \lambda \cdot a_z + \mu \cdot b_z + \gamma \cdot c_z &= T_z\end{aligned} \quad \text{(rel:20/5)}$$

U ovom sustavu nepoznanice su jedino koordinate točke T u prostoru S': λ, μ i γ. Riješimo najprije nepoznanicu λ. Prema pravilima iz linearne algebre, možemo pisati:

$$\lambda = \frac{\begin{vmatrix} T_x & b_x & c_x \\ T_y & b_y & c_y \\ T_z & b_z & c_z \end{vmatrix}}{\begin{vmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{vmatrix}}$$

$$\lambda = \frac{T_x \cdot (b_y c_z - c_y b_z) - T_y \cdot (b_x c_z - c_x b_z) + T_z \cdot (b_x c_y - c_x b_y)}{a_x \cdot (b_y c_z - c_y b_z) - a_y \cdot (b_x c_z - c_x b_z) + a_z \cdot (b_x c_y - c_x b_y)} \quad \text{(rel:20/6)}$$

Da bismo razriješili što nam posljednji izraz zapravo govori, treba se nečega prisjetiti. Ako imamo zadane vektore $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$, kako pronaći neki vektor koji bi bio okomit na oba? Jedan od načina je x-produktom.

$$\vec{\boldsymbol{\omega}} = \begin{vmatrix} \vec{\mathbf{e}} & \vec{\mathbf{f}} & \vec{\mathbf{g}} \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{vmatrix} = (b_y c_z - b_z c_y) \cdot \vec{\mathbf{e}} - (b_x c_z - b_z c_x) \cdot \vec{\mathbf{f}} + (b_x c_y - b_y c_x) \cdot \vec{\mathbf{g}} \quad \text{(rel:20/7)}$$

Komponente tog vektora koji je okomit na polazne vektore $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$ definirane su ptrehodnim izrazom (rel:20/7). Ukoliko su vektori $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$ jedinični, tada je i vektor određen x-produktom također jedinični. I sada treba dobro razmisliti. Ukoliko je vektor $\vec{\boldsymbol{\omega}}$ okomit i na vektor $\vec{\mathbf{b}}$, i na vektor $\vec{\mathbf{c}}$, i pri tome je jedinični, tada je vektor $\vec{\boldsymbol{\omega}}$ ili jednak vektoru $\vec{\mathbf{a}}$, ili je jednak minus vektoru $\vec{\mathbf{a}}$! Naime, budući da se nalazimo u 3D prostoru, tada su dva vektora koja su okomita na iste međusobno okomite vektore uvijek kolinearni! Kako su i vektor $\vec{\boldsymbol{\omega}}$, i vektor $\vec{\mathbf{a}}$ istovremeno i jedinični, tada se mogu razlikovati samo po smjeru! Dakle vrijedi:

$\vec{\boldsymbol{\omega}} = \vec{\mathbf{a}}$ (rel:20/8)

ili

$\vec{\boldsymbol{\omega}} = -\vec{\mathbf{a}}$ (rel:20/9)

Prema relaciji (rel:20/7) komponente vektora $\vec{\boldsymbol{\omega}}$ iznose:

$$\omega_x = b_y c_z - b_z c_y$$
$$\omega_y = -\left(b_x c_z - b_z c_x\right) \quad \text{(rel:20/10)}$$
$$\omega_z = b_x c_y - b_y c_x$$

Uvrštavanjem (rel:20/10) u (rel:20/6) dobiva se:

$$\lambda = \frac{T_x \cdot \omega_x + T_y \cdot \omega_y + T_z \cdot \omega_z}{a_x \cdot \omega_x + a_y \cdot \omega_y + a_z \cdot \omega_z} \qquad \text{(rel:20/11)}$$

Ukoliko vrijedi (rel:20/8), tada (rel:20/11) prelazi u:

$$\lambda = \frac{T_x \cdot a_x + T_y \cdot a_y + T_z \cdot a_z}{a_x \cdot a_x + a_y \cdot a_y + a_z \cdot a_z} = \frac{\vec{\mathbf{T}} \cdot \vec{\mathbf{a}}}{\vec{\mathbf{a}} \cdot \vec{\mathbf{a}}} = \frac{\vec{\mathbf{T}} \cdot \vec{\mathbf{a}}}{\left\|\vec{\mathbf{a}}\right\|^2} = \vec{\mathbf{T}} \cdot \vec{\mathbf{a}} \quad \text{(rel:20/12)}$$

Nazivnik u (rel:20/12) nestaje jer je norma vektora $\vec{\mathbf{a}}$ jednaka 1.

Ukoliko vrijedi (rel:20/9), tada (rel:20/11) prelazi u:

$$\lambda = \frac{-T_x \cdot a_x - T_y \cdot a_y - T_z \cdot a_z}{-a_x \cdot a_x - a_y \cdot a_y - a_z \cdot a_z} = \frac{-\left(\vec{\mathbf{T}} \cdot \vec{\mathbf{a}}\right)}{-\left(\vec{\mathbf{a}} \cdot \vec{\mathbf{a}}\right)} = \frac{\vec{\mathbf{T}} \cdot \vec{\mathbf{a}}}{\left\|\vec{\mathbf{a}}\right\|^2} = \vec{\mathbf{T}} \cdot \vec{\mathbf{a}} \quad \text{(rel:20/13)}$$

Obje pretpostavke (rel:20/8 i rel:20/9) vode na isti rezultat: koordinata λ jednaka je skalarnom produktu radij vektora točke T (zapisane u prostoru S) i vektora $\vec{\mathbf{a}}$ (zapisanog u prostoru S).

Identičnim se postupkom može pokazati da općenito vrijedi za sve koordinate:

$$\lambda = \vec{\mathbf{T}} \cdot \vec{\mathbf{a}}$$
$$\mu = \vec{\mathbf{T}} \cdot \vec{\mathbf{b}} \quad \text{(rel:20/14)}$$
$$\gamma = \vec{\mathbf{T}} \cdot \vec{\mathbf{c}}$$

Ovi izrazi vrijede tako dugo dok se poštuju pretpostavke uz koje su izvedeni, a to su:

- Svi bazni vektori su jedinični.
- Svi bazni vektori pojedinog prostora su međusobno okomiti.

Relaciju (rel:20/14) možemo pisati i u matričnom obliku:

$$T = \begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix} \cdot \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{T}} \cdot \vec{\mathbf{a}} & \vec{\mathbf{T}} \cdot \vec{\mathbf{b}} & \vec{\mathbf{T}} \cdot \vec{\mathbf{c}} \end{bmatrix} \quad \text{(rel:20/15)}$$

PROBLEM II.

Zadani su sustavi S i S' kao i u prethodnom razmatranju, no poznate su koordinate točke T u sustavu S': $T=[\lambda\ \mu\ \gamma]$ a traži se zapis točke u sustavu S: $T=[T_x\ T_y\ T_z]$.

Rješenje:

U prethodnom razmatranju pokazali smo da vrijedi relacija (rel:20/5). Ona direktno daje vrijednosti pojedinih komponenata u prostoru S. Matrično relacija može zapisati:

$$\begin{bmatrix} T_x & T_y & T_z \end{bmatrix} = \begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix} \quad \text{(rel:20/16)}$$

Relaciju (rel:20/15) mogli smo izvesti direktno iz relacije (rel:20/16) množenjem cijele jednadžbe sa desne strane inverzom matrice baznih vektora za koju se pokazuje da je jednaka upravo transponiranoj matrici:

$$\begin{bmatrix} T_x & T_y & T_z \end{bmatrix} = \begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix} \Big/ \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix}^{-1}$$

što daje:

$$\begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix}^{-1} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix}^{T}$$

iz čega slijedi:

$$\begin{bmatrix} \lambda & \mu & \gamma \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix} \cdot \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix}$$

što je upravo relacija (rel:20/15)! Naravno, sada će "matematičari" odmah graknuti da kako se za inverz matrice može uzeti transponirana matrica. Ovo dakako općenito ne vrijedi, no uzevši u obzir od čega je matrica građena, lako je pokazati da to u ovom slučaju vrijedi:

$$\begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix} \cdot \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix}^{T} = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \end{bmatrix} \cdot \begin{bmatrix} a_x & b_x & c_x \\ a_y & b_y & c_y \\ a_z & b_z & c_z \end{bmatrix} = \begin{bmatrix} \vec{\mathbf{a}} \cdot \vec{\mathbf{a}} & \vec{\mathbf{a}} \cdot \vec{\mathbf{b}} & \vec{\mathbf{a}} \cdot \vec{\mathbf{c}} \\ \vec{\mathbf{b}} \cdot \vec{\mathbf{a}} & \vec{\mathbf{b}} \cdot \vec{\mathbf{b}} & \vec{\mathbf{b}} \cdot \vec{\mathbf{c}} \\ \vec{\mathbf{c}} \cdot \vec{\mathbf{a}} & \vec{\mathbf{c}} \cdot \vec{\mathbf{b}} & \vec{\mathbf{c}} \cdot \vec{\mathbf{c}} \end{bmatrix} = \mathbf{I}$$

jer su vektori $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ i $\vec{\mathbf{c}}$ međusobno okomiti pa su im skalarni produkti jednaki nuli, odnosno budući da su vektori jedinični, tada je skalarni produkt vektora sa samim sobom jednak 1.

RJEŠENI ZADACI

ZADATAK 1.

Odredi jednadžbu pravca u 2D koji prolazi

a) kroz točke radnog prostora T_S=(1,2) i T_E=(5,10) i diskutiraj položaj točaka (3,6), (3,7) i (7,14) u odnosu na dvije poznate točke pravca.

b) kroz točke homogenog prostora T_S=(1,2,1) i T_E=(10,20,2) i provjeri leži li točka T_Q=(6,12,2) na pravcu.

Rješenje:

a) Pravac ćemo zapisati u parametarskom obliku

$$T_P = \overrightarrow{(T_E - T_S)} \cdot \lambda + T_S$$

što uvrštavanjem zadanih koordinata daje

$$\begin{aligned}
&\begin{bmatrix} T_{P1} & T_{P2} \end{bmatrix} = \overrightarrow{(\begin{bmatrix} T_{E1} & T_{E2} \end{bmatrix} - \begin{bmatrix} T_{S1} & T_{S2} \end{bmatrix})} \cdot \lambda + \begin{bmatrix} T_{S1} & T_{S2} \end{bmatrix} \\
&= \overrightarrow{(\begin{bmatrix} 5 & 10 \end{bmatrix} - \begin{bmatrix} 1 & 2 \end{bmatrix})} \cdot \lambda + \begin{bmatrix} 1 & 2 \end{bmatrix} \\
&= \overrightarrow{\begin{bmatrix} 4 & 8 \end{bmatrix}} \cdot \lambda + \begin{bmatrix} 1 & 2 \end{bmatrix}
\end{aligned}$$

Točka (3,6)

$$\left.\begin{aligned} (3,6) &= (4,8) \cdot \lambda + (1,2) \\ 3 &= 4 \cdot \lambda + 1 \Rightarrow \lambda = \frac{1}{2} \\ 6 &= 8 \cdot \lambda + 2 \Rightarrow \lambda = \frac{1}{2} \end{aligned}\right\}$$

Točka (3,6) je na pravcu jer je λ jednak za sve komponente. Dodatno, kako je 0<λ<1 zaključujemo da se točka nalazi između točaka T_S i T_E.

Točka (3,7)

$$\left.\begin{aligned} (3,7) &= (4,8) \cdot \lambda + (1,2) \\ 3 &= 4 \cdot \lambda + 1 \Rightarrow \lambda = \frac{1}{2} \\ 7 &= 8 \cdot \lambda + 2 \Rightarrow \lambda = \frac{5}{8} \end{aligned}\right\}$$

Točka (3,7) ne nalazi se na pravcu jer λ ne može istovremeno poprimiti dvije različite vrijednosti.

Točka (7,14)

$$(7,14) = (4,8)\cdot\lambda + (1,2)$$
$$\left.\begin{aligned} 7 &= 4\cdot\lambda + 1 \Rightarrow \lambda = \frac{3}{2} \\ 14 &= 8\cdot\lambda + 2 \Rightarrow \lambda = \frac{3}{2} \end{aligned}\right\}$$

Točka (7,14) nalazi se na pravcu. Kako je $\lambda>1$, zaključujemo da se točka nalazi iza točke T_E, tj. točka nije dio segmenta pravca određenog točkama T_S i T_E.

Drugi način rješavanja zadatka temelji se na implicitnom zapisu jednadžbe pravca i to u homogenom prostoru. Točke T_S i T_E prebaciti ćemo u homogeni prostor:

$T_{Sh} = (1,2,1)$ $T_{Eh} = (5,10,1)$

U poglavlju 2.2.1 izvedena je jednadžba za implicitni oblik pravca u radnom prostoru:

$$(T_{E2} - T_{S2})\cdot T_{P1} - (T_{E1} - T_{S1})\cdot T_{P2} - (T_{E2} - T_{S2})\cdot T_{S1} + (T_{E1} - T_{S1})\cdot T_{S2} = 0$$

Uvrštavanjem koordinata T_S i T_E dobiva se:

$$(10-2)\cdot T_{P1} - (5-1)\cdot T_{P2} - (10-2)\cdot 1 + (5-1)\cdot 2 = 0$$
$$8\cdot T_{P1} - 4\cdot T_{P2} + 0 = 0$$

Uvrštavanjem homogenih koordinata za točku T_{Ph} dobiva se:

$$8\cdot T_{Ph1} - 4\cdot T_{Ph2} + 0\cdot h_P = 0$$

ili matrično:

$$\begin{bmatrix} T_{Ph1} & T_{Ph2} & h_P \end{bmatrix}\cdot\begin{bmatrix} 8 \\ -4 \\ 0 \end{bmatrix} = 0$$

Sada možemo pomoću ovog oblika diskutirati položaj pojedinih točaka u ovisnosti o pravcu.

Točka (3,6).
Proširujemo točku u homogeni prostor: (3,6,1)

$$\begin{bmatrix} 3 & 6 & 1 \end{bmatrix}\cdot\begin{bmatrix} 8 \\ -4 \\ 0 \end{bmatrix} = 3\cdot 8 - 6\cdot 4 + 1\cdot 0 = 24 - 24 + 0 = 0$$

Zaključak: točka (3,6) nalazi se na pravcu. Gdje se nalazi u odnosu na točke T_S i T_E iz ovog se oblika ne može odrediti.

Točka (3,7).
Proširujemo točku u homogeni prostor: (3,7,1)

$$\begin{bmatrix} 3 & 7 & 1 \end{bmatrix}\cdot\begin{bmatrix} 8 \\ -4 \\ 0 \end{bmatrix} = 3\cdot 8 - 7\cdot 4 + 1\cdot 0 = 24 - 28 + 0 = -4$$

Zaključak: točka (3,7) nalazi se ispod pravca. Gdje se nalazi u odnosu na točke T_S i T_E iz ovog se oblika ne može odrediti.

Točka (7,14).
Proširujemo točku u homogeni prostor: (7,14,1)

$$\begin{bmatrix}7 & 14 & 1\end{bmatrix}\cdot\begin{bmatrix}8\\-4\\0\end{bmatrix} = 7\cdot 8 - 14\cdot 4 + 1\cdot 0 = 56 - 56 + 0 = 0$$

Zaključak: točka (7,14) nalazi se na pravcu. Gdje se nalazi u odnosu na točke T_S i T_E iz ovog se oblika ne može odrediti.

b) Idemo odmah primijeniti parametarsku jednadžbu, smatrajući homogenu koordinati "normalnom" koordinatom.

$$\begin{bmatrix}T_{P1} & T_{P2} & h_P\end{bmatrix} = \overrightarrow{(\begin{bmatrix}T_{E1} & T_{E2} & h_E\end{bmatrix} - \begin{bmatrix}T_{S1} & T_{S2} & h_S\end{bmatrix})}\cdot\lambda + \begin{bmatrix}T_{S1} & T_{S2} & h_S\end{bmatrix}$$
$$= \overrightarrow{(\begin{bmatrix}10 & 20 & 2\end{bmatrix} - \begin{bmatrix}1 & 2 & 1\end{bmatrix})}\cdot\lambda + \begin{bmatrix}1 & 2 & 1\end{bmatrix}$$
$$= \overrightarrow{\begin{bmatrix}9 & 18 & 1\end{bmatrix}}\cdot\lambda + \begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

Kada kao T_P uvrstimo točku (3,6,1)≡(6,12,2) dobiti ćemo:

$$\begin{bmatrix}3 & 6 & 1\end{bmatrix} = \overrightarrow{\begin{bmatrix}9 & 18 & 1\end{bmatrix}}\cdot\lambda + \begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

što je ekvivalentno sustavu od tri jednadžbe sa jednom nepoznanicom:

$$3 = 9\cdot\lambda + 1 \Rightarrow \lambda = \frac{2}{9}$$
$$6 = 18\cdot\lambda + 2 \Rightarrow \lambda = \frac{2}{9}$$
$$1 = 1\cdot\lambda + 1 \Rightarrow \lambda = 0$$

Izgleda kao da točka (3,6,1) ne leži na pravcu! No pogledate li malo bolje pravac i sve točke, vidjeti ćete da pravac prolazi kroz točke u radnom prostoru (1,2) i (5,10) te da smo upravo ispitivali je li točka (3,6) na tom pravcu. Maloprije smo vidjeli da jest. A sada tvrdimo da nije? Tu nešto ne valja!

U čemu je štos? Pa zapravo je dosta jednostavno vidjeti o čemu se radi. Mi smo primjenom parametarskog oblika jednadžbe pravca pretpostavili da se sve koordinate nekako mijenjaju ovisno o vektoru pravca $\vec{v}_P$ i parametru λ. I doista, kada govorimo o pravcu u radnom prostoru, ovo je razumna pretpostavka. No primjenjujući ovu pretpostavku na pravac zapisan preko homogenih koordinata, pretpostavka ne vrijedi! Naime, homogeni parametar točke ne mora slijediti apsolutno nikakav zakon! Pa rekli smo već da jedna točka u radnom prostoru ima beskonačno mnogo prikaza u homogenom prostoru. Tako pravac koji prolazi točkom (1,2) u radnom prostoru, prolazi točakama (1,2,1),(2,4,2),(3,6,3)... u homogenom prostoru.
Da bismo mogli koristiti parametarski oblik jednadžbe pravca, potrebno se je pridržavati jednog malog dogovora: *sve točke treba prikazati pomoću istog homogenog parametra*. Ako to učinimo, tada će vektor pravca $\vec{v}_P$ kao komponentu homogenog parametra imati vrijednost 0, te će jednadžba pravca direktno preslikavati homogeni parametar početne točke pravca u svaku drugu točku pravca. Dakle, pravilo kaže: sve točke treba prikazati pomoću istog homogenog parametra. Idemo odabrati najjednostavnije: h=1. Tada naše točke glase:

$T_S = (1,2,1)$, $T_E = (5,10,1)$, $T_Q = (3,6,1)$

$$[T_{P1} \quad T_{P2} \quad h_P] = (\overrightarrow{[T_{E1} \quad T_{E2} \quad h_E] - [T_{S1} \quad T_{S2} \quad h_S]}) \cdot \lambda + [T_{S1} \quad T_{S2} \quad h_S]$$
$$= (\overrightarrow{[5 \quad 10 \quad 1] - [1 \quad 2 \quad 1]}) \cdot \lambda + [1 \quad 2 \quad 1]$$
$$= \overrightarrow{[4 \quad 8 \quad 0]} \cdot \lambda + [1 \quad 2 \quad 1]$$

$$3 = 4 \cdot \lambda + 1 \Rightarrow \lambda = \frac{1}{2}$$
$$6 = 8 \cdot \lambda + 2 \Rightarrow \lambda = \frac{1}{2}$$
$$1 = 0 \cdot \lambda + 1 \Rightarrow \lambda \in R, \text{npr. } \lambda = \frac{1}{2}$$

Sada je sve u redu. Točka (3,6,1) doista leži na pravcu. Treba uočiti što se je sada dogodilo. Kako sve točke imaju isti homogeni parametar, homogeni parametar vektora pravca iznosi nula i dobije se jednadžba: $0 = 0 \cdot \lambda$

Kako je ova jednadžba ispunjena za svaki parametar λ, odabiremo baš onaj koji nam treba da pokažemo kako je točka na pravcu!

ZADATAK 2.

Odredi najmanju udaljenost između dva pravca koji nisu paralelni. Pravci su zadani u parametarskom obliku:

$T_P = \vec{v}_P \cdot \lambda + T_S$

$T_P^{'} = \vec{v}_P^{'} \cdot \mu + T_S^{'}$

Pronaći rješenje za slučaj:

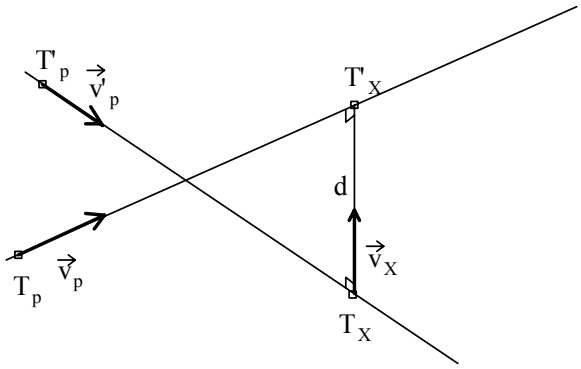
$\vec{v}_p = [1 \quad 0 \quad 0]$

$\vec{v}_P^{'} = [0 \quad 1 \quad 0]$

$T_P = [1 \quad 1 \quad 0]$

$T_P^{'} = [1 \quad 0 \quad 2]$

Rješenje.



Najmanja udaljenost između dva pravca dobiva se na onom mjestu gdje je spojnica ta dva pravca okomita i na jedan i na drugi pravac. Neka se ta spojnica proteže od točke T_X na jednom pravcu do točke $T_X^{'}$ na drugom pravcu. Spojnica također leži na pravcu. Taj pravac opisati ćemo vektorom pravca $\vec{v}_X$ i početnom točkom T_X. Kako je spojnica okomita na oba pravca, slijedi da je vektor pravca $\vec{v}_X$ okomit na vektore pravaca oba pravca:

$\vec{v}_P \cdot \vec{v}_X = 0$

$\vec{v}_P^{'} \cdot \vec{v}_X = 0$

Također točka T_X leži na pravcu pa vrijedi:

$T_X = \vec{v}_P \cdot \lambda + T_S$

Točka $T_X^{'}$ leži na drugom pravcu pa vrijedi:

$T_X^{'} = \vec{v}_P^{'} \cdot \mu + T_S^{'}$

Točke T_X i $T_X^{'}$ leže na spojnici pa vrijedi:

$T_X^{'} = \vec{v}_X \cdot \rho + T_X$

Ovih sedam jednadžbi raspada se na 11 jednadžbi sa 12 nepoznanica (nakon raspisivanja po komponentama) te je očito da rješenje nije jedinstveno (kritični su $\vec{v}_X$ i ρ jer o njima ništa nismo rekli). Sustav možemo upotpuniti zahtjevom da vektor $\vec{v}_X$ nastane baš kao razlika točaka $T_X^{'}$ i T_X, odnosno: $\vec{v}_X = T_X^{'} - T_X$, i tada vrijedi ρ=1. No unatoč tome sustav je preglomazan da bi ga išli računati ručno. Zbog toga ćemo si malo olakšati problem i krenuti u rješavanje postupno. Već je rečeno da zadani pravci nisu paralelni. To znači da im vektori pravaca nisu kolinearni. Zbog toga možemo jednostavno naći vektor koji je okomit na oba pravca. Ovakvih vektora ima beskonačno (i svi su međusobno kolinearni, tj. leže na istom pravcu, samo su različito "dugački"). No nama treba bilo koji od njih. Pa jednog takvog možemo dobiti kao npr. x-produkt. Ili možemo riješiti sustav koji nastaje iz uvjeta okomitosti:

$\vec{v}_P \cdot \vec{v}_X = 0$

$\vec{v}_P^{'} \cdot \vec{v}_X = 0$

Ovo je sustav dvije jednadžbe sa tri nepoznanice. Sustav očito nije jednoznačan baš zbog činjenice da nismo ništa rekli o željenoj dužini (odnosno normi) vektora. Uz zadane vrijednosti za vektore, i odabirom $v_{X3} = 1$ slijedi: $v_{X1} = 0, \quad v_{X2} = 0$, tj. $v_X = [0 \quad 0 \quad 1]$. Sada

kada smo pronašli okomiti vektor, iskoristimo činjenicu da T_X leži na jednom pravcu, te da drugi kraj spojnice leži na drugom pravcu i na samoj spojnici:

$T_X = \vec{v}_P \cdot \lambda + T_S$

$T_X + \vec{v}_X \cdot \rho = \vec{v}_P^{'} \cdot \mu + T_S^{'}$

Ovo se raspada u sustav od 6 jednadžbi sa 6 nepoznanica, no uz zadane točke rješenje se dobije trivijalno:

$\lambda = 0$

$\mu = 1$

$\rho = 2$

Udaljenost između dva zadana pravca jednako je duljini vektora $\rho \vec{v}_X$, tj.

$$d = \left\| \rho \cdot \vec{v}_X \right\| = \rho \cdot \sqrt{(v_{X1})^2 + (v_{X2})^2 + (v_{X3})^2} = 2 \cdot 1 = 2$$

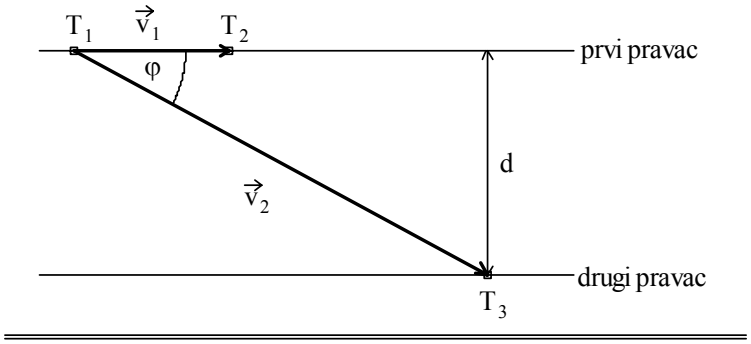
ZADATAK 3.

Odredi najmanju udaljenost između dva paralelna pravca. Pravci su zadani u parametarskom obliku:

$T_P = \vec{v}_P \cdot \lambda + T_S$

$T_P^{'} = \vec{v}_P \cdot \mu + T_S^{'}$

Rješenje.



Slika uz 3. zadatak

Budući da su pravci paralelni, pronađemo li dvije točke na jednom pravcu i jednu točku na drugom pravcu, te točke leže u ravnini i čine trokut. Točke ćemo odrediti sasvim proizvoljno. Npr.

$T_1 = T_p(\lambda = 0) = \vec{v}_P \cdot 0 + T_S = T_S$

$T_2 = T_p(\lambda = 1) = \vec{v}_P \cdot 1 + T_S = \vec{v}_P + T_S$

$T_3 = T_P^{'}(\mu = 0) = \vec{v}_P \cdot 0 + T_S^{'} = T_S^{'}$

Poznavajući točke T_1 i T_2, imamo vektor $\vec{v}_1 = T_2 - T_1$. Poznavajući točke T_1 i T_3, imamo vektor $\vec{v}_2 = T_3 - T_1$. Kut što ga čine $\vec{v}_1$ i $\vec{v}_2$ može se izračunati:

$$\cos \angle(\vec{v}_1, \vec{v}_2) = \frac{\vec{v}_1 \cdot \vec{v}_2}{\|\vec{v}_1\| \cdot \|\vec{v}_2\|}$$

No taj isti kut određen je odnosom katete koja predstavlja visinu trokuta, i hipotenuze koju predstavlja udaljenost između točaka T_3 i T_1, a to je upravo norma vektora $\vec{v}_2$. Slijedi:

$$\sin^2 \angle(\vec{v}_1, \vec{v}_2) = 1 - \cos^2 \angle(\vec{v}_1, \vec{v}_2) = \left(\frac{d}{\|\vec{v}_2\|} \right)^2$$

slijedi da je

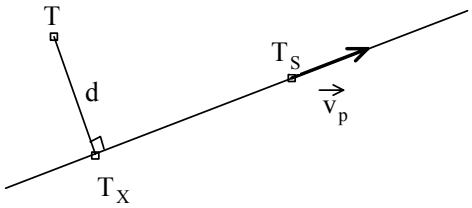
$$d^2 = \left(1 - \cos^2 \angle(\vec{v}_1, \vec{v}_2)\right) \cdot \|\vec{v}_2\|^2$$

Kosinus se može odmah izračunati, te uvrštavanjem u posljednju relaciju dobiva se i kvadrat udaljenosti.

ZADATAK 4.

Odredi najmanju udaljenost između točke T=(T_1,T_2,T_3) i pravca zadanog u parametarskom obliku $T_P = \vec{v}_P \cdot \lambda + T_S$.

Rješenje.



Slika uz 4. zadatak

Na pravcu treba odrediti onu točku T_X koja je najbliža točki T. To će biti ona točka za koju je spojnica povučena iz T prema T_X okomita na pravac (odnosno vektor pravca). Dobiju se jednadžbe:

$(T - T_X) \cdot \vec{v}_P = 0$ ... uvijet okomitosti

$T_X = \vec{v}_P \cdot \lambda + T_S$ ... T_X pripada pravcu

Sustav se raspada u četiri jednadžbe sa četiri nepoznanice. Dobije se:

$$(T_1 - T_{X1}) \cdot v_{P1} + (T_2 - T_{X2}) \cdot v_{P2} + (T_3 - T_{X3}) \cdot v_{P3} = 0$$
$$T_{X1} = v_{P1} \cdot \lambda + T_{S1}$$
$$T_{X2} = v_{P2} \cdot \lambda + T_{S2}$$
$$T_{X3} = v_{P3} \cdot \lambda + T_{S3}$$

Ako se tri posljednje jednadžbe uvrste u prvu, dobiti će se jednadžba sa nepoznanicom λ:

$$\lambda = \frac{v_{P1} \cdot (T_1 - T_{S1}) + v_{P2} \cdot (T_2 - T_{S2}) + v_{P3} \cdot (T_3 - T_{S3})}{v_{P1}^2 + v_{P2}^2 + v_{P3}^2}$$

Sada se iz jednadžbi za T_{xi} mogu izračunati pojedine komponente točke T_X jednostavnim uvrštavanjem parametra λ. Zadana udaljenost naposljetku je jednaka udaljenosti točaka T i T_X, odnosno normi vektora $T\text{-}T_X$.

$$d = \sqrt{(T_1 - T_{X1})^2 + (T_2 - T_{X2})^2 + (T_3 - T_{X3})^2}$$

ZADATAK 5.

Odredi sjecište dva pravca. Pravci su zadani u parametarskom obliku:
$T_P = \vec{v}_P \cdot \lambda + T_S$
$T_P' = \vec{v}_P' \cdot \mu + T_S'$

Rješenje.

Pravci se mogu sjeći, i ne moraju. Ukoliko se sijeku, tada postoje takvi λ i μ da $T_P = T_P'$. Ako se ovo napiše po komponentama, slijedi:

$$v_{P1} \cdot \lambda + T_{S1} = v_{P1}' \cdot \mu + T_{S1}'$$
$$v_{P2} \cdot \lambda + T_{S2} = v_{P2}' \cdot \mu + T_{S2}'$$
$$v_{P3} \cdot \lambda + T_{S3} = v_{P3}' \cdot \mu + T_{S3}'$$

Ovo je sustav tri jednadžbe sa dvije nepoznanice: λ i μ. Za rješavanje je dovoljno riješiti sustav po proizvoljne dvije jednadžbe, no tada treba provjeriti je li i ona treća zadovoljena! Ako nije, pravci se ne sijeku! Ako je, pravci se sijeku.

ZADATAK 6.

Odredi najmanju udaljenost između točke $T=(T_1,T_2,T_3)$ i ravnine zadane preko normale $\vec{v}_N$ i točke T_S koja pripada ravnini: $(T_R - T_S) \cdot \vec{v}_N = 0$.

Rješenje.

Na ravnini je potrebno pronaći takvu točku T_R za koju je spojnica te točke sa točkom T okomita na ravninu, odnosno kako znamo vektor $\vec{v}_N$ koji je okomit na ravninu, tražiti ćemo da spojnica bude paralelna (kolinearna) sa vektorom $\vec{v}_N$ što je identičan zahtjev zahtjevu okomitosti. Upravo izgovoren tekst pretočen u jednadžbe glasi:

$(T_R - T_S)\cdot \vec{v}_N = 0$ ... T_R je na ravnini
$T - T_R = \lambda \cdot \vec{v}_N$ ..... Uvjet kolinearnosti

Sustav se raspada na četiri jednadžbe sa četiri nepoznanice: tri komponente točke T_R i parametar λ.

$$(T_{R1} - T_{S1})\cdot v_{N1} + (T_{R2} - T_{S2})\cdot v_{N2} + (T_{R3} - T_{S3})\cdot v_{N3} = 0$$
$$T_1 - T_{R1} = \lambda \cdot v_{N1}$$
$$T_2 - T_{R2} = \lambda \cdot v_{N2}$$
$$T_3 - T_{R3} = \lambda \cdot v_{N3}$$

Uvrštavanjem donje tri jednadžbe u prvu, dobiva se jednadžba po parametru λ:

$$\lambda = \frac{(T_1 - T_{S1})\cdot v_{N1} + (T_2 - T_{S2})\cdot v_{N2} + (T_3 - T_{S3})\cdot v_{N3}}{v_{N1}^2 + v_{N2}^2 + v_{N3}^2}$$

gdje se λ može izračunati. Zatim se izračunati λ uvrsti u gornje tri jednadžbe i izračuna se točka T_R za koju se dobije:

$$T_{Ri} = T_i - \lambda \cdot v_{Ni}$$

Računanje točke T_R i nije nužno za određivanje udaljenosti točke od ravnine. Naime, udaljenost točke od ravnine je udaljenost između točaka T_R i T, odnosno norma vektora $T\text{-}T_R$. No ova norma se može dobiti direktno iz gornje tri jednadžbe koje opisuju T_i - T_{Ri}:

$$d = \|(T - T_R)\| = \sqrt{(T_1 - T_{R1})^2 + (T_2 - T_{R2})^2 + (T_3 - T_{R3})^2} =$$
$$= \sqrt{(\lambda \cdot v_{N1})^2 + (\lambda \cdot v_{N2})^2 + (\lambda \cdot v_{N3})^2} = \lambda \cdot \sqrt{{v_{N1}}^2 + {v_{N2}}^2 + {v_{N3}}^2} =$$
$$= \lambda \cdot \|\vec{v}_N\|$$

Iz posljednje relacije direktnim uvrštavanjem izračunatog parametra λ dobiva se tražena udaljenost.

ZADATAK 7.

Odredi sjecište dviju ravnina: $(T_R - T_S)\cdot \vec{v}_N = 0$ i $(T_R' - T_S')\cdot \vec{v}_N' = 0$.

Rješenje.

Ukoliko se dvije ravnine sijeku, tada postoje zajedničke točke. To znači da je zahtjev da točka $T_R = T_R^{'}$ što rezultira sustavom:

$(T_R - T_S) \cdot \vec{v}_N = 0$

$(T_R - T_S^{'}) \cdot \vec{v}_N^{'} = 0$

Ovo je sustav od dvije jednadžbe sa tri nepoznanice. Ukoliko rješenje postoji, tada ćemo imati jednadžbu sa jednim parametrom. Zaključujemo da ukoliko se ravnine sijeku, sijeku se po pravcu. Kada raspišemo jednadžbe, dobije se:

$(T_{R1} - T_{S1}) \cdot v_{N1} + (T_{R2} - T_{S2}) \cdot v_{N2} + (T_{R3} - T_{S3}) \cdot v_{N3} = 0$

$(T_{R1} - T_{S1}^{'}) \cdot v_{N1}^{'} + (T_{R2} - T_{S2}^{'}) \cdot v_{N2}^{'} + (T_{R3} - T_{S3}^{'}) \cdot v_{N3}^{'} = 0$

Sustav treba riješiti nekom od metoda za rješavanje sustava linearnih jednadžbi. No da ne kompliciramo, pretpostaviti ćemo da su koeficijenti uz T_{R1} različiti od 0, odnosno da se T_{R1} pojavljuje u obje jednadžbe. Tada se T_{R1} može izlučiti iz gornje jednadžbe i uvrstiti u donju (ukoliko pretpostavka o koeficijentima ne vrijedi, umjesto T_{R1} može se uzeti T_{R2} ili T_{R3}; nešto će uvijek postojati!). Dobiti će se jednadžba oblika:

$a \cdot T_{R2} + b \cdot T_{R3} + c = 0$ tj.

$T_{R2} = -\frac{b}{a} \cdot T_{R3} - \frac{c}{a}$ ili

$T_{R2} = E \cdot T_{R3} + F$

Uvrštenjem ovog rješenja u npr. prvu jednadžbu dobiti će se i T_{R1} kao funkcija od T_{R3}, što će biti oblika:

$T_{R1} = G \cdot T_{R3} + H$

Proglasimo li sada T_{R3} parametrom λ dobiti ćemo:

$T_{R1} = E \cdot \lambda + F$

$T_{R2} = G \cdot \lambda + H$

$T_{R3} = 1 \cdot \lambda + 0$

što je parametarski oblik jednadžbe pravca!

ZADATAK 8.

Odredi sjecište triju ravnina: $(T_R - T_S) \cdot \vec{v}_N = 0$, $(T_R^{'} - T_S^{'}) \cdot \vec{v}_N^{'} = 0$ i $(T_R^{''} - T_S^{''}) \cdot \vec{v}_N^{''} = 0$.

Rješenje.

Ukoliko se tri ravnine sijeku, tada postoje zajedničke točke (praviti ćemo se da ne znamo koliko ih je pa ćemo o broju raspravljati nakon što napišemo sustav). To znači da je zahtjev da točka $T_R = T_R^{'} = T_R^{''}$ što rezultira sustavom:

$(T_R - T_S) \cdot \vec{v}_N = 0$

$(T_R - T_S^{'}) \cdot \vec{v}_N^{'} = 0$

$(T_R - T_S^{''}) \cdot \vec{v}_N^{''} = 0$

Ovo je sustav od tri jednadžbe sa tri nepoznanice. Ukoliko rješenje postoji, tada ćemo imati jedno jedinstveno rješenje. Zaključujemo da ukoliko se ravnine sijeku, sijeku se u točci. Kada raspišemo jednadžbe, dobije se:

$$(T_{R1} - T_{S1}) \cdot v_{N1} + (T_{R2} - T_{S2}) \cdot v_{N2} + (T_{R3} - T_{S3}) \cdot v_{N3} = 0$$
$$(T_{R1} - T_{S1}^{'}) \cdot v_{N1}^{'} + (T_{R2} - T_{S2}^{'}) \cdot v_{N2}^{'} + (T_{R3} - T_{S3}^{'}) \cdot v_{N3}^{'} = 0$$
$$(T_{R1} - T_{S1}^{''}) \cdot v_{N1}^{''} + (T_{R2} - T_{S2}^{''}) \cdot v_{N2}^{''} + (T_{R3} - T_{S3}^{''}) \cdot v_{N3}^{''} = 0$$

Sustav treba riješiti nekom od metoda za rješavanje sustava linearnih jednadžbi. Kao rješenje će se dobiti točka $T_R = (T_{R1}, T_{R2}, T_{R3})$

ZADATAK 9.

Zadana je ravnina $R = \begin{bmatrix} 2 \\ -7 \\ -2 \\ -13 \end{bmatrix}$. Zapisati ovu ravninu pomoću njezine normale!

Rješenje.

Ravninu treba prikazati u obliku $(T_R - T_S) \cdot \vec{v}_N = 0$. Da bismo ovo mogli, treba nam vektor normale, te jedna (bilo koja) točka ravnine.
Ukoliko se za prikaz točaka ravnine koristi predloženi matrični oblik, možemo pisati:

$$T_R \cdot R = 0 \quad \text{odnosno} \quad T_R \cdot \begin{bmatrix} 2 \\ -7 \\ -2 \\ -13 \end{bmatrix} = 0$$

Slijedi :

$$\begin{bmatrix} T_{R1} & T_{R2} & T_{R3} & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ -7 \\ -2 \\ -13 \end{bmatrix} = 0$$

odnosno :

$$2T_{R1} - 7T_{R2} - 2T_{R3} - 13 = 0$$

Ukoliko krenemo iz zapisa preko normale $(T_R - T_S) \cdot \vec{v}_N = 0$, raspisivanjem izraza dobiva se da su faktori uz komponente točke T_R upravo komponente vektora $\vec{v}_N$. Znači da direktnim očitavanjem bilo iz matrice R, bilo iz zapisa koji se dobije množenjem točke T_R sa matricom R, slijedi da vektor $\vec{v}_N$ glasi:

$$\vec{v}_N = \begin{bmatrix} 2 & -7 & -2 \end{bmatrix}$$

Još nam preostaje da odredimo bilo koju točku ravnine koju ćemo proglasiti točkom T_S. No kako sve točke ravnine zadovoljavaju jednadžbu:

$$2T_{S1} - 7T_{S2} - 2T_{S3} - 13 = 0$$

odabirom npr. $T_{S3}=0$, $T_{S2}=1$ slijedi $T_{S1}=10$, tj. $T_S=(10, 1, 0)$, te zapis pomoću normale glasi:

$$\left(T_R - \begin{bmatrix}10 & 1 & 0\end{bmatrix}\right) \cdot \begin{bmatrix} 2 \\ -7 \\ 2 \end{bmatrix} = 0$$

ZADATAK 10.

Zadane su ravnina $R_A = \begin{bmatrix} 2 \\ -7 \\ -2 \\ -13 \end{bmatrix}$ i $R_B = \begin{bmatrix} -1 \\ 12 \\ 18 \\ -35 \end{bmatrix}$. Naći sjecište ovih ravnina!

Rješenje.

Ravnine R_A i R_B možemo kao u zadatku 9 napisati pomoću normale, te sjecište naći kao u zadatku 7. No može i kraće. Matrice R_A i R_B određuju dvije implicitne jednadžbe ravnina. Zahtjevom da točke pripadaju istovremeno obim ravninama proizlazi sustav:

$$2T_{R1} - 7T_{R2} - 2T_{R3} - 13 = 0$$
$$-T_{R1} + 12T_{R2} + 18T_{R3} - 35 = 0$$

Ovo je sustav dviju jednadžbi sa tri nepoznanice. Eliminiranjem nepoznanice T_{R1} dobiva se:

$$17T_{R2} + 34T_{R3} - 84 = 0 \quad \text{pa je}$$
$$T_{R2} = -2T_{R3} + \frac{84}{17}$$

Uvrštavanjem izraza za T_{R2} u npr. prvu jednadžbu dobiva se:

$$T_{R1} = -6T_{R3} + \frac{413}{17}$$

Vidimo da izrazi za T_{R1} i T_{R2} ovise o T_{R3}. Zato ćemo T_{R3} proglasiti parametrom λ, i imamo parametarski oblik jednadžbe pravca koji je ujedno i jednadžba sjecišta:

$$T_{R1} = -6 \cdot \lambda + \frac{413}{17}$$
$$T_{R2} = -2 \cdot \lambda + \frac{84}{17}$$
$$T_{R3} = 1 \cdot \lambda + 0$$

Pravac možemo zapisati preko karakteristične matrice pravca L:

$$L = \begin{bmatrix} -6 & -2 & 1 & 0 \\ \frac{413}{17} & \frac{84}{17} & 0 & 1 \end{bmatrix}$$

U ovom zapisu u prvom se retku nalazi vektor pravca u homogenom prostoru (zato četiri komponente i to je razlog što je homogena komponenta jednaka nuli), a u drugom retku se nalazi bilo koja točka pravca opet u homogenom izdanju (zato je uzeto h=1 i prepisana je točka T_S iz parametarskog oblika jednadžbe pravca).