

# Računarska grafika

*Beleške za predavanja*

Vesna Marinković

email: [vesnap@matf.bg.ac.rs](mailto:vesnap@matf.bg.ac.rs)

URL: [www.matf.bg.ac.rs/~vesnap](http://www.matf.bg.ac.rs/~vesnap)

Predrag Janičić

email: [janicic@matf.bg.ac.rs](mailto:janicic@matf.bg.ac.rs)

URL: [www.matf.bg.ac.rs/~janicic](http://www.matf.bg.ac.rs/~janicic)

Matematički fakultet, Beograd

©2017

Autori:

dr Vesna Marinković, docent Matematičkog fakulteta u Beogradu  
dr Predrag Janičić, redovni profesor Matematičkog fakulteta u Beogradu

**RAČUNARSKA GRAFIKA**

Sva prava zadržana. Nijedan deo ovog materijala ne može biti reprodukovani niti smešten u sistem za pretraživanje ili transmitovanje u bilo kom obliku, elektronski, mehanički, fotokopiranjem, smanjenjem ili na drugi način, bez prethodne pismene dozvole autora.

---

## Predgovor

---

Ovaj tekst predstavlja prateći materijal za kurs "Računarska grafika" na Matematičkom fakultetu Univerziteta u Beogradu. Nastao je kao proširenje skripte prof. dr Predraga Janičića koji je ovaj kurs držao tokom od 2002/03 do 2004/05 i od 2007/08 do 2013/14. Tekst je prvenstveno zasnovan na narednim materijalima:

- John F. Hughes et al: *Computer Graphics: principles and practice* (3rd. ed),
- Peter Shirley et al, *Fundamentals of Computer Graphics* (3rd ed.),
- slajdovi za kurs "Uvod u računarsku grafiku" Univerziteta Brown

ali i na mnogim drugim izvorima.

Materijal je namenjen studentima da im olakša praćenje nastave i pripremu ispita, ali ni na koji način ne može zameniti pohađanje nastave.

Veliku pomoć u izradi trenutne verzije skripte dobila sam od prof. dr Predraga Janičića. Zahvaljujem se takođe i studentima na mnogobrojnim korisnim komentarima, posebno Lazaru Vasoviću, Gorani Vučić i Jeleni Simović. S obzirom na to da je materijal i dalje u fazi dorade i ima grešaka, molim čitaoce da ako uoče neku grešku, jave autoru skripte.

Vesna Marinković,  
Beograd, septembar 2017.

---

# Sadržaj

---

<b>Sadržaj</b>	<b>4</b>
<b>1 Uvod</b>	<b>7</b>
1.1 Logistika kursa . . . . .	7
1.2 Šta je računarska grafika? . . . . .	8
1.3 Hjерархијско моделиовање . . . . .	13
1.4 Различите парадигме у рачунарској графици . . . . .	15
1.5 Интерактивна рачунарска графика – од Sketchpad-a до данас . . . . .	16
1.6 Архитектура растер система за приказ са процесором за приказ . . . . .	24
1.7 Примене рачунарске графике . . . . .	26
1.8 Интересантни линкови: . . . . .	28
1.9 Питанја . . . . .	28
<b>2 2D algoritmi</b>	<b>29</b>
2.1 Алгоритми за цртање 2D прimitива . . . . .	29
2.2 Цртање дузи (eng. scan converting line) . . . . .	31
2.3 Цртање круга . . . . .	38
2.4 Попunjавање полигона . . . . .	43
2.5 Клипинг/сеckanje линија (eng. clipping) . . . . .	50
2.6 Питанја . . . . .	57
<b>3 Geometrijske osnove</b>	<b>59</b>
3.1 Једначине праве и равни . . . . .	59
3.2 Вектори . . . . .	60
3.3 Linearne transformacije . . . . .	60
3.4 2D transformacije . . . . .	61
3.5 3D transformacije . . . . .	73
3.6 Пресликавање тачака, правих и равни . . . . .	77
3.7 Трансформације као промена координатног система . . . . .	78
3.8 Трансформације и graf scene . . . . .	79
3.9 Питанја . . . . .	81
<b>4 Projektovanje</b>	<b>83</b>
4.1 Tipovi projektovanja . . . . .	83
4.2 Perspektivna projekcija . . . . .	85

4.3	Paralelna projekcija . . . . .	89
4.4	Tipovi planarnih projekcija . . . . .	94
4.5	Primer izračunavanja projekcija tačaka . . . . .	94
4.6	Pitanja . . . . .	97
<b>5</b>	<b>Zadavanje sintetičkog modela kamere</b>	<b>99</b>
5.1	Zadavanje kamere kod perspektivnog projektovanja . . . . .	100
5.2	Zadavanje kamere kod paralelnog projektovanja . . . . .	103
5.3	Načini modelovanja kamere . . . . .	104
5.4	Izgradnja matrica transformacija na osnovu zadatih parametara kamere . . . . .	104
5.5	Proces renderovanja . . . . .	110
5.6	Pitanja . . . . .	111
<b>6</b>	<b>Načini opisivanja figura u 2D i 3D</b>	<b>113</b>
6.1	“Mreže” u 2D . . . . .	116
6.2	Mreže u 3D . . . . .	118
6.3	Pitanja . . . . .	123
<b>7</b>	<b>Opisivanje krivih i površi u 3D</b>	<b>125</b>
7.1	Osnovne polinomske krive . . . . .	126
7.2	Nadovezivanje krivih i Catmull-Rom splajn . . . . .	128
7.3	Kubni B-splajn . . . . .	131
7.4	Crtanje krivih . . . . .	131
7.5	Pitanja . . . . .	131
<b>8</b>	<b>Vidljivost</b>	<b>133</b>
8.1	Uvod (“To render or not to render, that is the question...”) . . . . .	133
8.2	Rej kasting . . . . .	137
8.3	Bafer dubine . . . . .	140
8.4	z-bafer algoritam . . . . .	141
8.5	Algoritmi sa listama prioriteta . . . . .	142
8.6	Odsecanje i odbacivanje u odnosu na zarubljenu piramidu pogleda	145
8.7	Pitanja . . . . .	145
<b>9</b>	<b>Prostorne strukture podataka</b>	<b>147</b>
9.1	Stabla . . . . .	149
9.2	Mreža (rešetka) . . . . .	154
9.3	Pitanja . . . . .	158
<b>10</b>	<b>Monohromatska i hromatska svetlost</b>	<b>159</b>
10.1	Monohromatska svetlost . . . . .	159
10.2	Izbor intenziteta . . . . .	160
10.3	Polutoniranje . . . . .	161
10.4	Hromatska (obojena) svetlost . . . . .	164
10.5	Izbor i korišćenje boja . . . . .	174
10.6	Pitanja . . . . .	175
<b>11</b>	<b>Osvetljenje i senčenje</b>	<b>177</b>
11.1	Modeli osvetljenja . . . . .	178
11.2	Hromatska svetlost . . . . .	185

---

11.3 Senčenje poligona . . . . .	185
11.4 Senke . . . . .	186
11.5 Transparentnost . . . . .	186
11.6 Međuobjektne refleksije i globalno osvetljenje . . . . .	187
<b>12 Teksture . . . . .</b>	<b>191</b>
12.1 Popločavanje i istezanje . . . . .	193
12.2 Pridruživanje teksturnih koordinata . . . . .	194
<b>13 Aliasing . . . . .</b>	<b>199</b>
13.1 Antialiasing . . . . .	199
13.2 Teksturni aliasing . . . . .	201

## *Glava 1*

---

# **Uvod**

---

### **1.1 Logistika kursa**

#### **Cilj kursa:**

- *praktični deo:* pisanje programa u OpenGL-u koji rade sa 2D i 3D računarskom grafikom
- *teorijski deo:* razumevanje matematičkih aspekata i algoritama koji su u osnovi modernih grafičkih sistema
- cilj kursa NIJE učenje specifičnosti nekog konkretnog grafičkog programa kao što je Maya, već savladavanje koncepata na kojima se grafički programi baziraju

#### **Literatura:**

- John F. Hughes et al, “Computer Graphics: Principles and Practice (3rd ed.)”, 2013.
- Peter Shirley et al, “Fundamentals of Computer Graphics (3rd ed.)”, 2009.

## 1.2 Šta je računarska grafika?

“Jedna slika vredi hiljadu reči”  
kineska poslovica

“Možda najbolji način da definišemo računarsku grafiku jeste taj da utvrdimo šta ona u stvari nije. Ona nije mašina. Nije ni računar niti grupa računarskih programa. Nije ni znanje i iskustvo grafičkog dizajnera, programera, pisca, filmskog stručnjaka, niti stručnjaka za pravljenje reprodukcija. Računarska grafika je sve ovo – tehnologija kojom se upravlja na promišljen način i koja je usmerena ka preciznom i opisnom prenošenju informacija.”

William A. Fetter, 1960, “tvorac” termina računarska grafika

**Računarska grafika** je nauka i umetnost vizuelnog komuniciranja kroz ekran računara i uređaje za interakciju<sup>1</sup>. Do nedavno se pod vizuelnim aspektom uglavnom podrazumevao smer komunikacije od računara ka korisniku, dok se smer od korisnika ka računaru uobičajeno realizovao uređajima kao što su miš i tastatura. Međutim, poslednjih godina je smer komunikacije od korisnika ka računaru počeo da se menja: moguće je računaru vratiti vizuelne podatke, posredstvom novih vidova interfejsa primenjenih na video ulaz.

Računarska grafika je međudisciplinarna oblast u kojoj važnu ulogu igraju fizika (za modelovanje svetlosti i izvođenje simulacija potrebnih za animacije), matematika (za opis geometrije figura), čovekova moć opažanja (bitna je za alokaciju potrebnih resursa jer ne želimo da trošimo vreme da renderujemo stvari koje neće biti vidljive), interakcija između ljudi i računara, grafički dizajn, umetnost. Sve ovo je potrebno da bi smer komunikacije od računara ka korisniku bio što efikasniji.

Uža definicija računarske grafike bi glasila da se računarska grafika bavi pravljenjem modela objekata na sceni (geometrijskog opisa objekata na sceni i opisa kako objekti reflektuju svetlost) i modela osvetljenja scene (matematički opis izvora svetlosne energije, pravaca u kojima se ona emituje, raspodele talasnih dužina svetlosti) i, na osnovu njih, pravljenjem reprezentacija određenog pogleda na scenu (svetlost koja dolazi od nekog imaginarnog oka ili kamere na sceni).

Dakle, rad u okviru računarske grafike podrazumeva stvaranje, skladišteњe i rukovanje modelima i slikama. Pritom to mogu biti realistični vizuelni efekti, informativne tehničke ilustracije ili interesantne računarske animacije. Modeli dolaze iz raznolikog skupa oblasti kao što su fizika, biologija, matematika, umetnost i ovaj skup se stalno i iznova širi. Modeli mogu da prikazuju stvarne objekte (uključujući stvarne objekte koji se ne mogu videti u stvarnom svetu - na primer, atome) ili zamišljene objekte. Dakle, i dobijene slike mogu biti u potpunosti sintetičke ili dobijene obradom fotografija. Dakle, računarska grafika bavi se prikazima i na osnovu dvodimenzionalnih i na osnovu trodimenzionalnih modela.

Rad u oblasti računarske grafike zahteva dobro poznavanje određenog hardvera i grafičkog aplikacionog programskog interfejsa (API), kao što su npr. GTK ili QT. Međutim, računarska grafika je oblast koja se jako brzo

<sup>1</sup>Ova definicija preuzeta je iz knjige “Computer Graphics: principles and practice”.

razvija, te čemo se mi na časovima predavanja truditi da izbegnemo zavisnost od bilo kog određenog hardvera i API-ja, već će nam akcenat biti na konceptima koji su zajednički za sve implementacije.

Veliki ideo današnjeg istraživanja u oblasti računarske grafike je u metodama za pravljenje geometrijskih modela, metodama za predstavljanje refleksije površi, metodama za animaciju scena prema zakonima fizike i aproksimacijama ovih zakona, interakciju sa virtuelnim objektima i sl.

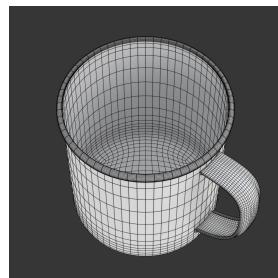
Često se postavlja pitanje *zašto izučavati računarsku grafiku?* Razlozi su brojni i oni uključuju:

- *intelektualne izazove*: razumevanje nekih aspekata fizičkog sveta; primena novih metoda izračunavanja i prikaza i novih tehnologija
- *estetske/umetničke izazove*: pravljenje realističnih virtuelnih svetova i interagovanje sa njima
- *tehničke izazove*: matematika projekcija, krivih, površi; fizika osvetljenja i senčenja; programiranje grafičkih softvera

Računarska grafika je danas uz fotografiju i televiziju treći dominantan način proizvodnje slika. Prednost računarske grafike je predstavljanje apstraktnih objekata, kao što su npr. podaci koji nemaju ugrađenu geometriju (npr. numerički rezultati nekog eksperimenta).

Osnovne poddiscipline računarske grafike su:

**modelovanje** – bavi se pravljenjem matematičke specifikacije tela i njegovih vizuelnih svojstava na način koji je moguće sačuvati na računaru

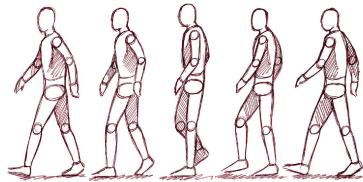


Slika 1.1: Mrežni model šolje za kafu

**renderovanje** – to je proces kreiranja realistične dvodimenzione digitalne slike na osnovu (dvodimenzionog ili trodimenzionog) modela i (realističnog ili nerealističnog) modela ponašanja svetlosti;

**animacija** – to je proces kreiranja nizova slika koje, kada se prikažu brzo jedna za drugom, daju utisak glatkog kretanja.

Postoje, takođe, i mnoge druge oblasti koje uključuju računarsku grafiku, kao što su:



Slika 1.2: Niz slika ljudskog tela u pokretu kojima se stiče utisak kretanja

**obrada slika** – bavi se zapisivanjem i obradom slika (kao što su isecanje dela slike, skaliranje slike, kombinovanje više slika, ...), kao i rekonstrukcijom dvodimenzionalih ili trodimenzionalnih objekata na osnovu njihovih slika. Obrada slika ima široku primenu: u satelitskim snimanjima, medicini, kosmičkim istraživanjima, robotici, prepoznavanju slova (OCR) itd.



Slika 1.3: Primer isecanja i skaliranja slike

**virtuelna realnost** – pokušava da korisnika „ubaci“ u 3D virtuelni svet, korišćenjem napredne 3D grafike i naprednih uređaja za prikaz

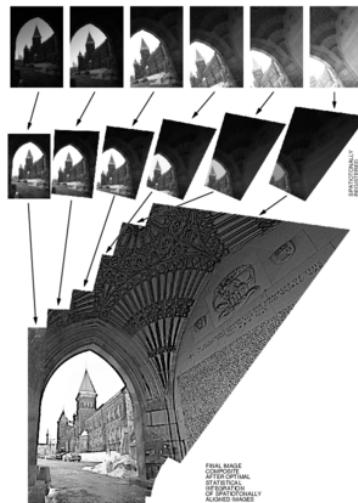


Slika 1.4: Veći i manji 3D skener (vlasništvo MI SANU)

**3D skeniranje** – koristi tehnologiju baziranu na pronalaženju opsega za pravljenje merljivih 3D modela; oni su korisni za pravljenje bogatih vizuel-

nih prikaza i obrada ovakvih modela često zahteva algoritme iz oblasti grafike

**računarska fotografija** – korišćenje metoda iz oblasti računarske grafike i obrade slika za omogućavanja novih načina fotografskog pamćenja objekata, scena i okruženja



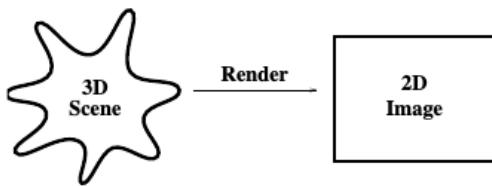
Slika 1.5: Računarska (“nedigitalna”) fotografija pruža brojne nove mogućnosti – ovde je prikazano kako se na osnovu niza slika, optimalnim kombinovanjem informacija iz više različito eksponiranih fotografija koje se preklapaju, može napraviti panoramska slika

Neke od ovih poddisciplina se nužno prožimaju, te na primer kreiranje slika (koje spada u renderovanje) i njihova obrada imaju sve više zajedničkih tačaka i često je teško povući jasnu granicu između njih (na primer, programi kao PhotoShop omogućavaju izdvajanje dela slike po nekom kriterijumu, a zatim obradu tog dela na neki zadati način).

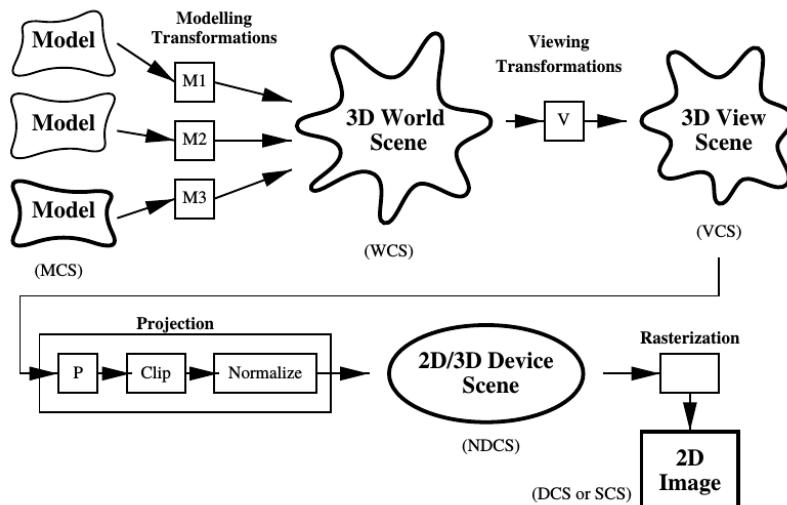
U okviru ovog kursa akcenat će biti na procesima modelovanja i renderovanja.

Krenućemo od opisa *renderovanja unapred* koje predstavlja proces renderovanja koji je najčešće podržan u hardveru i koji koristi OpenGL. U ovoj vrsti renderovanja primitive se transformišu od modela ka uređaju za prikaz. S druge strane, rejtrejsing algoritam (koji ćemo izložiti naknadno u okviru kursa) predstavlja primer koncepta *renderovanja unazad* kod koga se kreće od tačke na slici i onda se utvrđuje koje se primitive projektuju na nju. Oba pristupa imaju i svoje prednosti i mane.

Renderovanje predstavlja transformisanje scene u sliku. Pritom se scena sastoji od modela u trodimenzionom prostoru. Modeli su sastavljeni od primitiva koje su podržane u sistemu za renderovanje. Tradicionalno, transformacija iz modela preko scenu pa sve do slike se “razbija” na manje korake, koji se nazivaju **grafičkom protočnom obradom** (engl. graphics pipeline). Osnovna varijanta grafičke protočne obrade kod renderovanja unapred prikazana je na slici 1.6. Najpre se geometrijski modeli postavljaju na 3D scenu (primenom



različitih geometrijskih transformacija na osnovna geometrijska tela jedinične veličine), zatim se ovi modeli posmatraju kroz kameru čime se njihove svetske koordinate transformišu u koordinate standardne zapremine pogleda. Nakon toga se oni projektuju u 2D sliku i ova slika se transformiše u prozor za prikaz.



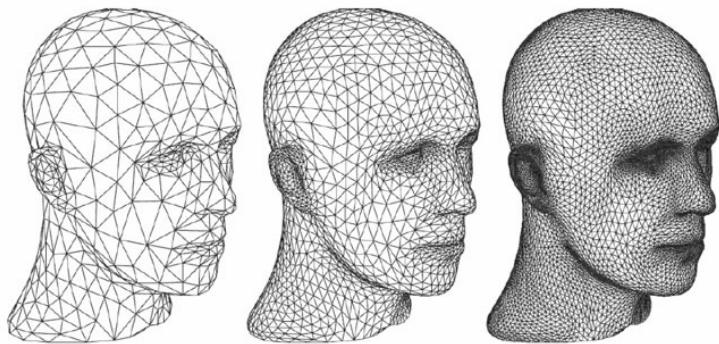
Slika 1.6: Osnovni model protočne obrade kod renderovanja unapred

Modeli se sastoje iz velikog broja geometrijskih primitiva (ili se mogu transformisati u njih). Neke od primitiva koje su direktno podržane u hardveru jesu tačke, duži, poligoni (trouglovi ili konveksni poligoni)... Modelovanje primitiva takođe uključuje i deo po deo polinomijalne krive, deo po deo polinomijalne površi i slično. Takođe, potrebni su brojni osnovni algoritmi kao što su:

- *transformisanje*: transformisanje reprezentacije modela/primitiva iz jednog u drugi koordinatni sistem
- *odsecanje/odbacivanje skrivenih površi*: odbacivanje primitiva i delova primitiva koje nisu vidljive na ekranu
- *rasterizacija*: konvertovanje projektovane primitive u skup piksela

### 1.3 Hijerarhijsko modelovanje

Prilikom modeliranja potrebno je da model “uhvati” glavne karakteristike (podatke, ponašanje) objekata/pojava koja se modeluju: ti podaci uključuju geometriju objekta, njegov izgled, atributi, . . . Možemo praviti model nekog jednostavnog objekta kao što je npr. čajnik ili pak mrežu od 2 milijarde poligona skulpture Mikelanđelovog Davida <sup>2</sup>.



Slika 1.7: Tri različita modela istog objekta sa različitim nivoom detalja (različitim brojem trouglova u mreži)

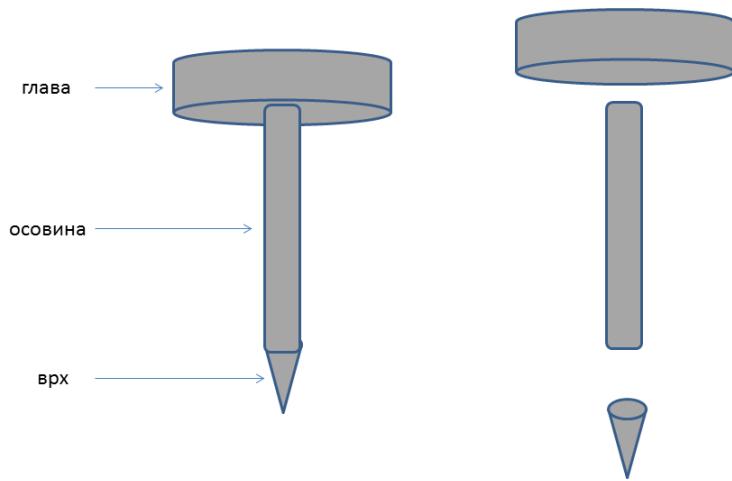
**Modelovanje** obuhvata pravljenje modela, primenu materijala na modele, postavljanje modela na scenu, pozicioniranje svetla na sceni, postavljanje kamere, dok je **renderovanje** pravljenje slike uz pomoć kamere.

U modelovanju se objekat koji treba modelovati prvo (vizuelno) analizira i onda se *razlaže* (*dekomponuje*) na jednostavnije komponente. Važni principi prilikom modelovanja su sledeći:

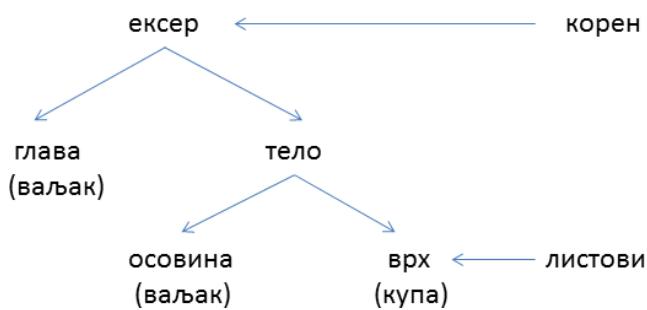
- divide-and-conquer strategija (podeli-pa-vladaj ili tehnika razlaganja)
- hijerarhija geometrijskih komponenti
- redukcija na *primitive* (sfere, kocke, . . .)
- razlikujemo jednostavne elemente od onih koji to nisu (primer: ekser i šraf)

---

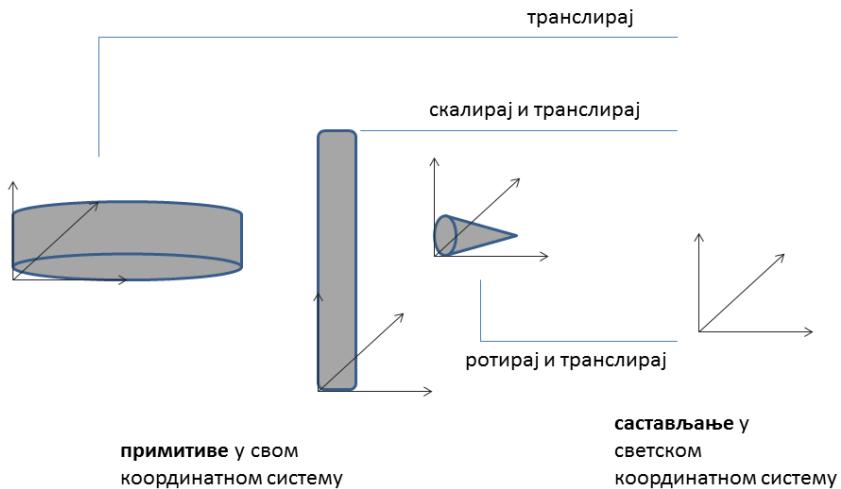
<sup>2</sup>Na adresi <http://youtu.be/TAZIvyAJfeM> dat je fotorealistični prikaz ljudskog oka.



**Dijagram stabla** obezbeđuje hijerarhijski, vizuelni metod za izražavanje odnosa "sastavljen od". Ovakvi dijagrami su deo 3D progamskog interfejsa (npr 3D Studio MAX).

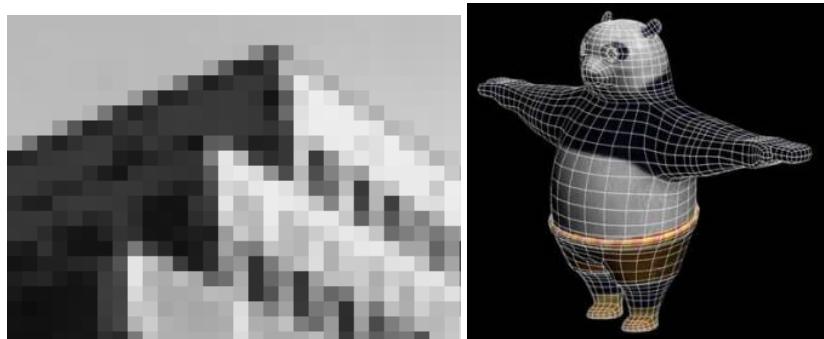


Struktura podataka koja treba da se renderuje naziva se **граф scene**. Primitive kreirane u procesu dekompozicije se trebaju *sastaviti* da bi se kreirao finalni objekat. Ovo se postiže korišćenjem afinih transformacija: translacije, rotacije, skaliranja. Bitan je redosled ovih transformacija jer nisu komutativne.



## 1.4 Različite paradigmе у računarskoj grafici

Postoje dve osnovne paradigmе koje koriste grafičke aplikacije su: **графика заснована на узорку** и **графика заснована на геометрији**.



Slika 1.8: Графика заснована на узорку и на геометрији (прузето са курса U.Brown)

### 1.4.1 Графика заснована на узорку

**Графика заснована на узорку** користи дискретне узорке за описивање визуелне информације. Пиксели се могу креирати дигитализацијом слике (често се неки аспекти физичког света узоркују за визуелизацију, нпр. температуре у Србији).

У овом приступу пиксели су локације тачака са приђућим вредностима узорка, највеће интензитета светlosti, transparentnosti и другим контролним

informacijama. Kada se slika definiše kao niz piksela, nju je moguće jednostavno:

- izmeniti: to su izmene kreirane od strane korisnika, kao što su sečenje i lepljenje sekcija, alati tipa četkice i slično,
- obraditi: tu spadaju algoritamske operacije koje se izvode na slici bez intervencije korisnika, kao što su zamućivanje, oštrenje, balansiranje boja, rotiranje i sl.

Primer grafičke aplikacije koja je zasnovana na uzorku je Adobe Photoshop GIMP™

Prednosti ovog pristupa su da kada se slika jednom definiše u terminima boje na  $(x, y)$  poziciji mreže, ona se lako može modifikovati izmenom lokacije ili vrednosti boje, informacije o pikselima jedne slike se mogu iskopirati u drugu, zamenom ili kombinovanjem sa prethodnim pikselima. Mane ovog pristupa su što ne postoje dodatne informacije već važi paradigma WYSIAYG (What You See Is All You Get). Dakle, ne postoje informacije o dubini, niti se scena može istraživati iz druge tačke pogleda.

#### 1.4.2 Grafika zasnovana na geometriji

Grafika zasnovana na geometriji se naziva i *skalabilna vektorska grafika* ili *objektno-orientisana grafika*. U ovoj paradigmi kreiraju se i čuvaju matematički opisi ili modeli geometrijskih elemenata (pravih, poligona, ...) i pridruženih atributa (npr. boje, svojstva materijala) i onda se oni uzorkuju za vizuelizaciju (vrši se rasterizacija). Korisnik najčešće ne može da direktno radi nad individualnim pikselima u geometrijski zasnovanim programima, već dok korisnik radi sa geometrijskim elementima, program iznova uzorkuje i prikazuje elemente.

Primeri 2D grafičkih aplikacija koje su zasnovane na ovoj paradigmi su Adobe Illustrator™, CorelDRAW™, ..., dok su Autodesk's AutoCAD™, Autodesk's Maya™, Autodesk's 3D Studio Max™ primeri 3D grafičkih aplikacija.

Renderovanje sve više kombinuje geometrijski zasnovanu grafiku i grafiku zasnovanu na uzorku, da bi se povećao kvalitet finalnog proizvoda.

## 1.5 Interaktivna računarska grafika – od Sketchpad-a do danas

Interaktivna računarska grafika podrazumeva dinamički način prikaza slike na računaru uz aktivno učešće čoveka u stvaranju i izmeni slike, gde su rezultati odmah vidljivi (na primer, igrice). Nasuprot tome, pod **neinteraktivnom računarskom grafikom** podrazumeva svako generisanje ili prezentovanje slikevnih informacija koje ne zadovoljava prethodne uslove; dakle, radi se o statičnim informacijama, koje su prezentovane bojom i oblikom, bez interakcije čoveka.

U interaktivnoj računarskoj grafici korisnik kontroliše sadržaj, strukturu i izgled objekata i njihovih slika koje se prikazuju korišćenjem brze vizuelne povratne informacije. To se postiže korišćenjem ulaznih uređaja, kao što su recimo miš, ekran osetljiv na dodir, (sve ređe tastatura)...

Osnovne komponente interaktivnog grafičkog sistema su:

- **ulaz** (realizuje se posredstvom nekog ulaznog uređaja, kao što su recimo miš ili olovka)
- **obrada** (i skladištenje modela u pozadini)
- **prikaz/izlaz** (realizuje se posredstvom nekog izlaznog uredjaja, kao što je ekran, štampač, ...)

### 1.5.1 Sketchpad

“Sistem Sketchpad koristi crtanje kao novi vid komunikacije sa računarom. Sistem sadrži ulaz, izlaz i računarske programe koji omogućavaju interpretaciju informacija koje se direktno iscrtavaju na računarskom ekranu. Sketchpad se pokazao korisnim kao pomoć u razumevanju procesa, kao što je kretanje veza, koje se može opisati slikama. Sketchpad, takođe, olakšava iscrtavanje slika koje sadrže veliki broj ponavljenih elemenata i slika koje treba da budu jako precizne, kao i izmenu slika prethodno iscrtanih pomoći njega.”

Ivan Sutherland, prvi pasus doktorske teze “Sketchpad, A Man-Machine Graphical Communication System”, 1963, MIT.

**Sketchpad** je bio prvi u potpunosti interaktivni grafički sistem, koji sadrži skoro sve ključne elemente interaktivne računarske grafike. Za rad je korišćen mejnfrejm računar u MIT-ovim laboratorijama. Sam Sketchpad je uveo mnoge koncepte koji postoje i u današnjim sistemima, kao što su: kontekstni meniji, hijerarhijsko modelovanje i slično. Doktorska teza Ivana Sutherlenda predstavlja prvu tezu iz oblasti računarske grafike.

Interaktivna računarska grafika je dugo ostala iznad mogućnosti svih osim tehnološki najintenzivnijih organizacija. Neki od razloga za to su visoke cene grafičkog hardvera (koji se proizvodio u malim serijama), nepotabilan softver koji je najčešće pisan za poseban proizvođačev uređaj za prikaz, nepostojanje grafičkih standarda i sl.

### 1.5.2 Savremena interaktivna računarska grafika

Danas se za upravljanje računarom koriste ulazni uređaji koji prihvataju podatke i instrukcije od korisnika i konvertuju ih u formu razumljivu računaru. Ulazni uređaji se grubo mogu podeliti na:

- *tekstualne ulazne uređaje*: npr. tastatura,
- *pokazivačke uređaje*: miš, trackball, ekran osetljiv na dodir (touchscreen), light-pen, olovka (stajlus),
- *force-feedback (haptic) uređaje*: džojstik, volan za upravljanje i drugi kontroleri igara (kao što je Wii) - njima se simuliraju određeni fizički atributi, čime se omogućava da korisnik dodirom direktno komunicira sa virtuelnim objektima),
- *slikovne i video uređaje*: skener, digitalni fotoaparat, digitalna kamera, ...



Slika 1.9: Multi-touch – funkcionalnost pomoću koje ekrani osjetljivi na dodir registruju istovremeni višestruki dodir na površini (preuzeto sa: <https://www.engadget.com/2010/04/20/synaptics-extends-multitouch-gesture-suite-to-linux-chrome-os-i/>)



Slika 1.10: Force-feedback radna stanica koja predstavlja integrisani sistem simulacija kompletne šake leve i desne ruke (preuzeto sa: [http://www.diytrade.com/china/pd/5554912/Immersion\\_Haptic\\_Force\\_Feedback\\_Workstation.html](http://www.diytrade.com/china/pd/5554912/Immersion_Haptic_Force_Feedback_Workstation.html))

U današnje vreme razvijaju se i *senzorski uređaji*. Primer je Leap Motion – američka kompanija koja se bavi proizvodnjom i prodajom senzorskih uređaja koji pružaju podršku pokretima prstiju i ruke i predstavljaju ulaznu tehnologiju koja radi bez fizičkog kontakta<sup>3</sup>.

*Pametni uređaji* su elektronski uređaji koji se mogu povezati sa drugim uređajima ili mrežama korišćenjem nekog bežicnog protokola i koji mogu da u određenoj meri rade autonomno i interaktivno. U poslednje vreme primećuje se ubrzani razvoj “pametnih” uređaja, kao što su pametni telefoni, laptopovi, tableti, pametni satovi, i slično.



Slika 1.11: Pametni satovi – uključuju praćenje GPS lokacije tokom trčanja, vožnje biciklom, plivanja (vodootporne varijante), izračunavanje raznih statistika, potrošnju kalorija, ... (preuzeto sa: <http://switch2reality.com/smart-watch/>)

<sup>3</sup>Prikaz senzorskog uređaja je dostupan na adresi: <https://www.youtube.com/watch?v=zXghYjh6Gro&feature=youtu.be>

Takođe, primećuje se nagli trend razvoja tehnologije sa namjerom una-predjenja doživljaja virtuelne stvarnosti. Jedan od takvih uređaja jesu i HMD (Head-Mounted Displays) uređaji za prikaz koji se nose na glavi i imaju mali optički uređaj ispred jednog ili oba oka. Najčešće se njima prikazuju samo računarski generisane slike, a poneke varijante mogu da kombinuju i pogled na realni svet.



Slika 1.12: HMD uređaji (preuzeto sa: [https://en.wikipedia.org/wiki/Head-mounted\\_display](https://en.wikipedia.org/wiki/Head-mounted_display))

Razvijaju se i 3D prostori virtuelne stvarnosti, npr. 3D virtuelne pećine (engl. cave)<sup>4</sup>. Virtuelna pećina je okruženje virtuelne stvarnosti u kojoj su projektori usmereni na 3 do 6 zidova kocke veličine sobe. Zidovi su uobičajeno napravljeni od projektorskih ekrana, a pod, takođe, može da bude projektorski ekran ili tanki panel za prikaz. Sistemi za projektovanje su veoma vi-soke rezolucije da bi obezbedili uverljivu iluziju stvarnosti. Korisnici nose 3D naočare i mogu da vide objekte koji lebde u vazduhu, mogu da se kreću oko njih i slično. Prostor je opremljen i velikim brojem zvučnika postavljenih pod različitim uglovima kojima se 3D video doživljaj dopunjuje 3D zvučnim efektima.



Slika 1.13: 3D virtuelne pećine (preuzeto sa: <http://147.91.14.147/Blog.aspx?id=252>)

Još jedan primer razvoja podrške virtuelnoj realnosti predstavlja i onlajn viruelni svet Second Life lansiran 2003. godine, koji je u 2013. godini imao preko milion korisnika. U njemu korisnici kreiraju virtuelne reprezentacije sebe - avatare i mogu da interaguju sa drugim avatarima, objektima i mestima. Mogu da se socijalizuju, učestvuju u aktivnostima, grade, kupuju i raspolažu

---

<sup>4</sup>Simulacija pećine dosupna je na adresi <https://www.youtube.com/watch?v=j59JxfbvxGg>

virtuelnim nekretninama. 2007. godine joj se pridružila i Srbija i mnoge kompanije su videle interes da budu deo ovog virtuelnog sveta.

### 1.5.3 Prikazni uređaji

U izlazne uređaje spadaju ekrani, štampači (2D i 3D), ploteri, video projektori (2D i 3D), itd.

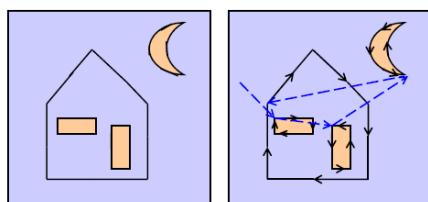
Pre nego što krenemo sa pregledom razvoja izlaznih tehnologija potrebno je definisati neke osnovne termine.

**Adresivost** (engl. addressability) je broj pojedinačnih (ne nužno razlučivih) tačaka po inču koje mogu biti kreirane. Može da se razlikuje horizontalna adresivost i vertikalna adresivost.

**Rezolucija** je broj razlučivih od strane posmatrača ili uređaja različitih linija po inču (na primer, naizmenično crnih i belih) koje uređaj može da kreira. Rezolucija ne može biti veća od adresivosti.

#### Ekrani (monitori)

U početku su korišćeni **vektorski sistemi** (od 1950-ih do sredine 1970-ih). Kod njih se linija dobija tako što se digitalne koordinate krajnjih tačaka transformišu u analogni napon za elektronski zrak koji pada na površinu ekrana. Ova metodologija se zove i *random scan* metodologija (zato što linija može da spaja „bilo koje dve tačke na ekranu“). Slika se osvežava obično 30 do 60 puta u sekundi (30-60 Hz). Na vektorskim monitorima moguće je iscrtati i do 100000 linija. Pritom su linije glatke, ali se obojene površine teško prikazuju. Pogodni su samo za mrežne modele. Vektorski ekrani su imali veoma visoku rezoluciju (neki ekrani su mogli da adresiraju i do 4000x4000 tačaka).



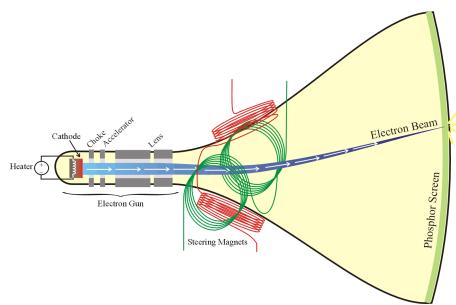
Slika 1.14: Iscrtavanje slike na vektorskem monitoru

1960-ih se javlja nova generacija vektorskih sistema, pod nazivom **DVST sistemi** (engl. direct view storage tube). Kod njih se ne zahteva velika učestalost osvežavanja slike (jer je slika reprezentovana raspodelom naelektrisanja na unutrašnjoj površini ekrana). Međutim, prilikom izmene i najmanjeg detalja na slici, kompletna slika (raspodela naelektrisanja) mora ponovo biti kreirana.

1970-ih se javljaju **CRT sistemi** (engl. cathode ray tube) koji predstavljaju **rastersku grafiku** zasnovanu na televizijskoj tehnologiji i katodnim cewima. Kod ovih sistema slika je reprezentovana pikselima. Oni funkcionišu po sledećem principu: elektronski top emituje zrak elektrona koji se ubrzava pozitivnim naponom do fosforom presvučenog ekrana. Na putu do ekrana, mehanizmom za fokusiranje elektroni se usmeravaju ka konkretnoj tački na ekranu. Obzirom da fosfor emituje svetlost koja opada eksponencijalno sa vremenom, sliku je potrebno osvežavati, najčešće 60 puta u sekundi (60Hz).



Slika 1.15: Tektronix računar sa DVST monitorom (1970-e)



Slika 1.16: Ilustracija rada katodne cevi

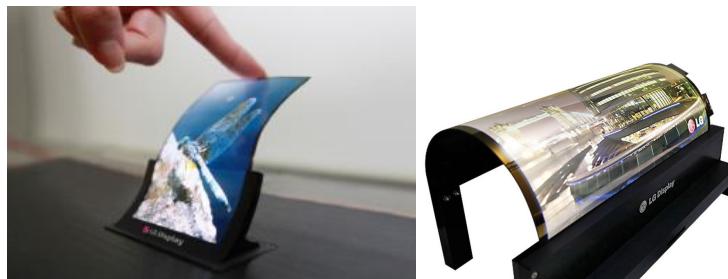
U ovim sistemima piksel nema jasno određene ivice već se za njegovu veličinu uzima prečnik oblasti gde je intenzitet emitovanja veći od 50% intenziteta u središtu oblasti. Kod CRT sistema sadržaj slike ne utiče na brzinu prikazivanja, te kreiranje slike može da bude zahtevnije nego na vektorskim sistemima: na primer, u vektorskome sistemu za telo koje rotira dovoljno je preračunavati koordinate kontrolnih tačaka.

Za razliku od vektorskih sistema kod kojih su linije glatke, u rasterskim sistemima kose linije su „stopenaste“, ali se zato obojene površine lakše prikazuju (problem popunjavanja oblasti nije težak).

CRT monitori se i danas neretko koriste u štamparskoj industriji, medijskim kompanijama, oblasti profesionalne fotografije – jer daju pouzданije boje i kontrast, kao i širi ugao posmatranja.

**LCD ekrani** (eng. liquid-crystal display) postaju široko rasprostranjeni od sredine 1990-ih. Oni predstavljaju tanke, ravne prikazne uređaje. Zasnovani su na korišćenju dugačkih molekula kristala, koji zahvaljujući specifičnom korišćenju elektriciteta i polarizacije svetlosti stvaraju osvetljene ili tamne delove ekranu. LCD paneli ne proizvode svetlost, te im je neophodan izvor svetla (tzv. "pozadinsko svetlo"). Glavna prednost im je što troše veoma malo električne energije i što nema treperenja slike. Slika se osvežava i 200 puta u sekundi (200Hz). Za razliku od CRT ekrana, LCD ekrani imaju nativnu fiksnu, rezoluciju, sa fiksnim rasterom.

Kao podtip LCD ekrana postoje i **active matrix LCD ekrani** kod kojih je u svaki piksel ugrađen po jedan tranzistor (koji opisuje taj piksel), te kod njih nema treperenja. Jedan od podtipova ovog tipa ekrana su **TFT ekrani** (eng. Thin Film Transistor). Nekoliko puta u sekundi se vrši kontrola individualnih piksela da bi se utvrdilo da li ima promena.



Slika 1.17: Fleksibilni OLED ekrani

U novije vreme javljaju se i **OLED ekrani** (engl. organic light-emitting diode). Oni se prave od organskih materijala koji emituju svetlost kada kroz njih prolazi elektricitet. Dok je kod LCD ekrana potrebno da postoji pozadinsko svetlo, pikseli kod OLED ekrana proizvode svoju svetlost, te su oni efikasniji i dosta tanji. Takođe na OLED ekranima moguće je postići veću kontrast nego kod LCD ekrana. Međutim ova tehnologija je jako komplikovana i zbog toga su OLED ekrani mnogo skuplji od LCD ekrana.

### Štampači (2D)

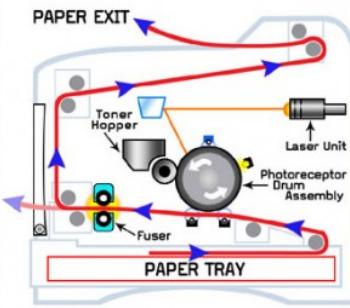
**Matrični štampači** (engl. dot-matrix printers) imaju od 7 do 24 iglice (ili pina) koje udaraju u ribon (traku) i ostavljaju trag na papiru. Adresivost može da bude i manja nego rastojanje između iglica (jer može da se štampa u dva prolaza, a pri tome je moguće pomeranje papira za dužinu manju od rastojanja između dve susedne iglice ili su iglice raspoređene pogodno u dve kolone). Postoji i varijanta u boji (sa četiri ribona).



Slika 1.18: Matrični štampač Epson LQ-570

Kod **laserskih štampača** laserski zrak prelazi preko pozitivno nanelektrisanog rotirajućeg doboša i oblast pogođena zrakom gubi nanelektrisanje i pozitivno nanelektrisanje ostaje samo tamo gde kopija treba da bude crna. Nakon toga se negativno nanelektrisani prah tonera privlači od strane pozitivno nanelektrisanih oblasti. U kolor verziji ovaj postupak se ponavlja tri puta.

**Ink-jet štampači** rade na principu usmeravanja tankih mlazeva tečnog mastila na papir. Oni su najčešće u boji i imaju zasebne komore za tri boje i crnu.



Slika 1.19: Ilustracija rada laserskog štampača

Ink-jet štampači su najčešće korišćeni štampači u širokoj upotrebi zbog svoje niske cene, visokog (fotorealsitičnog) kvaliteta otiska, mogućnosti za štampanje u živim bojama i jednostavne upotrebe. Nedostaci ink-jet štampača su cena štampanja po strani (mastilo je prilično skupo) i osetljivost otiska (mastilo je najčešće rastvorivo u vodi).

Postoje i **termalni štampači** (eng. thermal-transfer printers) zasnovani na korišćenju zagrevanja posebnih vrsta papira, slično kao kod faks mašina.

### Ploteri

Kao što matrični štampači odgovaraju raster grafici, tako ploteri odgovaraju vektorskoj grafici. Međutim, poslednjih godina u velikoj meri su potisnuti od strane štampača velikih formata.

Među ostale izlazne uređaje spadaju i projektori (2D i 3D), 3D štampači i sl.

#### 1.5.4 Uslovi za razvoj moderne računarske grafike

Neki od uslova koji su morali da budu ispunjeni da bi se razvila moderna računarska grafika su sledeći:

- revolucija u hardveru
  - Murov zakon: snaga računara se udvostručuje približno svakih 12-18 meseci, dok se veličina računara smanjuje;
  - novi procesori su 64-bitni sa 2, 4, 6, 8 jezgara;
  - svakih 6 meseci postoji značajni napredak u razvoju grafičkih čipova (u odnosu na period od nekoliko godina kod procesora opšte namene).
- razvoj grafičkog podsistema
  - grafička obrada se prebacuje sa procesora na čip koji je napravljen da brzo izvršava grafičke operacije;
  - grafički procesori (engl. GPU, Graphics Processing Unit) su inicijalno razvijeni da daju podršku grafičkim izračunavanjima posebne

namene; međutim, postali su toliko moćni da su počeli da se koriste kao koprocesori i sve više se koriste za paralelizaciju druge vrste izračunavanja (GPGPU – General-Purpose Computing on Graphics Processing Unit).

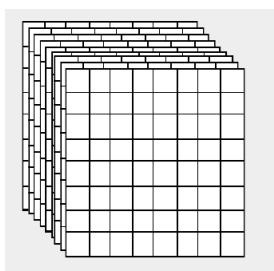
- napredak u razvoju softvera
  - razvoj algoritama i struktura podataka (modelovanje materijala, renderovanje prirodnih pojava, ubrzavajuće strukture podataka za rejtrejsing i sl.);
  - paralelizacija operacija (većina operacija se neometano izvršava paralelno): npr. menjanje vrednosti jednog piksela je najčešće nezavisno od ostalih;
  - distribuirano izračunavanje i računanje u “oblaku” (operacije se šalju na “oblak” i dobijaju se rezultati, nije bitno kako).

## 1.6 Arhitektura raster sistema za prikaz sa procesorom za prikaz

Osnovne komponente rasterskih sistema za prikaz su:

- procesor za prikaz, odnosno grafički procesor (engl. display processor, GPU)
- video kontroler (engl. video controller)
- frejm bafer (engl. frame buffer)
- ekran

U rasterskim sistemima slika se formira u **frejm baferu** procesom rasterizacije. Na frejm bafer se može gledati kao na računarsku memoriju organizovanu u vidu dvodimenzionog niza tako da svaka adresibilna lokacija ( $x, y$ ) odgovara jednom pikselu. U njemu se čuva sadržaj koji se zatim prikazuje na ekranu. **Bitska dubina** (engl. bit depth) je broj bitova koji odgovara svakom pikselu. Tipične rezolucije frejm bafera su  $640 \times 480 \times 8$ ,  $1280 \times 1024 \times 8$ ,  $1280 \times 1024 \times 24 \dots$ .

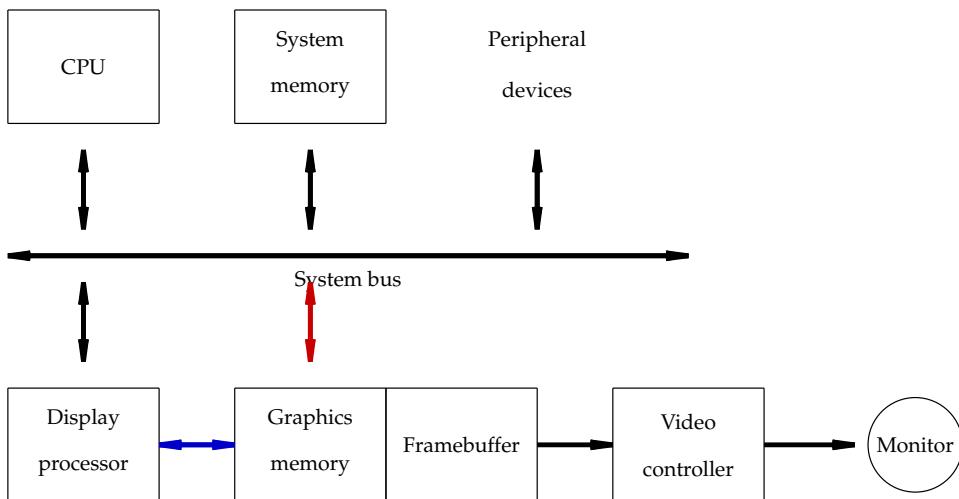


Slika 1.20: Primer predstavljanja frejm bafera

**Procesor za prikaz** (odnosno grafički procesor) je hardver specijalne namene razvijen da pomogne u rasterizovanju grafičkih primitiva koje se smeštaju u frejm bafer<sup>5</sup>.

**Video kontroler** pristupa frejm baferu i prikazuje liniju po liniju na ekranu. On je zadužen da stalno osvežava sadržaj ekrana. Postoje dva tipa kontrolera:

- **sa preplitanjem** (engl. interlaced) – osvežavaju se parne, a zatim neparne linije na npr. 30Hz; najčešće se koristi kod televizora da umanji efekat treperenja
- **bez preplitanja** (engl. noninterlaced) – koristi se ako cela slika može da se osvežava na više od 60Hz; najčešće se koristi kod ekrana



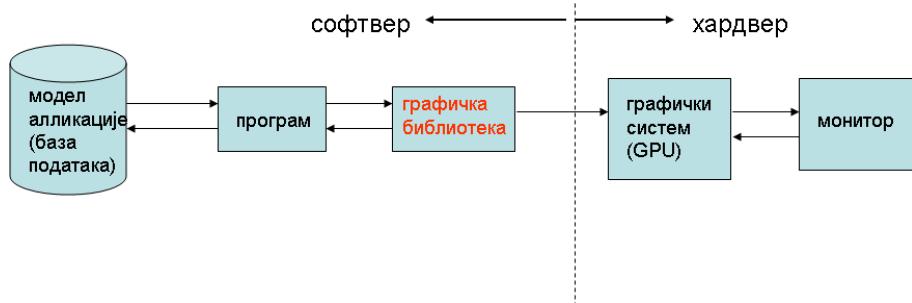
Slika 1.21: Jedna arhitektura raster sistema za prikaz (strelice crvene i plave boje ilustruju dve varijante)

Konceptualno okruženje za interaktivnu grafiku se sastoji iz narednih komponenti:

- grafička biblioteka se nalazi između aplikacije i hardvera za prikaz (grafičkog sistema)
- program aplikacije preslikava objekte aplikacije u njihove slike pozivom funkcija grafičke biblioteke

<sup>5</sup>Moderni grafički procesori, zahvaljujući svojoj paralelnoj strukturi, mogu biti efikasniji od opštenamenskih centralnih procesora i često se koriste kada je potrebno obraditi velike količine podataka paralelno (popularni jezici/sistemi: OpenGL, Nvidia CUDA).

- interakcija sa korisnikom rezultuje u izmeni slike i/ili modela



Ovo hardversko i softversko okruženje je staro 5 decenija, ali se i dalje koristi.

Primeri grafičkih biblioteka su OpenGL™, DirectX™, HTML5+WebGL™ ... Grafičke biblioteke imaju podršku za zadavanje primitiva (karakteri, prave, poligoni, ...), atributa (boja, stil linije, svojstva materijala za 3D, ...), svetlosti, transformacija. Razlikujemo **direktni** i **odloženi režim**. U direktnom režimu nema sačuvane reprezentacije, paket čuva samo stanje atributa i aplikacija mora da u potpunosti iscrta svaki frejm. U odloženom režimu se reprezentacija scene koju treba prikazati čuva u vidu **grafa scene** (datog u obliku direktnog acikličkog grafa) te biblioteka kompajlira i prikazuje sadržaj grafa scene.

## 1.7 Primene računarske grafike

Računarska grafika je iz noviteta prerasla u svakodnevni fenomen. Kako je grafika sve više i više postajala deo svakodnevice, tako su rasla i očekivanja korisnika. Video igre danas prikazuju više miliona poligona u sekundi, a specijalni efekti su toliko dobri da ih više nije lako razaznati od materijala koji nije kompjuterski generisan. Digitalne kamere i fotoaparati nam daju ogromne nizove piksela, a takođe i alati za njihovu obradu brzo evoluiraju. U isto vreme sve veća snaga računara omogućila je bogatiju formu grafike.

Računarska grafika ima veliki uticaj na sve oblasti filma, televizije, oglašavanja i igara. Takođe, promenila je način na koji posmatramo informacije iz oblasti medicine, arhitekture, kontrole procesa u industriji, rada mreže, kao i informacije iz svakodnevnog života, pa tako na primer vidimo mape vremenske prognoze i slične vizuelizacije informacija. Možda je najznačajniji uticaj računarske grafike u oblasti grafičkih korisničkih interfejsa na telefonima, računarima, kontrolnim tablama automobila i mnogim drugim elektronskim kućnim uređajima.

Primeri korišćenja računarske grafike su brojni:

- grafički korisnički interfejsi
  - prirodnija i lakša komunikacija sa računarom korišćenjem paradigm *point and click*
  - grafička manipulacija sve više zamenjuje kucanje komandi
  - prisutni su u svim tipovima aplikacija – telefonima, računarima, automobilskim kontrolnim tablama, kućnim električnim uređajima

- interaktivna izrada crteža u biznisu, nauci (npr. medicini) i tehnologiji
  - vizuelizacija neophodna za razumevanje i interpretiranje podataka velikog obima i kompleksnih podataka koji nemaju nužno prirodni vizuelni prikaz. Na primer, vremenski trend cena deset različitih akcija može se vizuelizovati pametnim algoritmima za izradu grafikona tako da ljudi mogu da uoče obrasce u ovim podacima, iako ih na prvi pogled ne vide
  - u medicini se prave smislene slike podataka skeniranih pacijenata. Na primer, na osnovu skupa podataka o vrednosti dubine dobijenih kompjuterskom tomografijom (CT), mogu se kreirati osenčene slike koje pomažu doktorima da iz ovakvih podataka izdvoje glavne informacije
- elektronsko izdavaštvo
- CAD i CAM (eng. Computer Aided Design and Manufacturing) – ove oblasti koriste računarsku tehnologiju za dizajniranje delova i proizvoda na računarima, a zatim, korišćenjem ovih virtuelnih dizajna, za vođenje procesa proizvodnje. Na primer, mnogi mehanički delovi se dizajniraju u paketu za 3D računarsko modelovanje i onda automatski proizvode na uređaju za glodanje kontrolisanim od strane računara
- naučne simulacije i vizualizacije, kartografija i sl.
- simulacije u cilju treniranja – ekstremno korisne za inicijalne treninge u oblastima u kojima je kritična sigurnost, npr simulacije voznje, simulacije letenja ili u svrhe treninga iskusnih korisnika kao što su specifične situacije borbe sa vatrom koje bi bilo ili suviše skupo ili opasno fizički napraviti
- simulacije i animacije u zabavne svrhe: filmovi (“Priča o igračkama”, 1995, prvi film koji je u potpunosti napravljen korišćenjem računarske grafike), računarske igre (koje sve više koriste sofisticirane 3D modele i algoritme renderovanja), vizuelni efekti (skoro svi novi filmovi koriste digitalni kompozit da bi sastavili pozadinu sa posebno snimljeno prvim planom filma; takođe, mnogi filmovi koriste 3D modelovanje i animaciju da bi stvorili sintetička okruženja, objekte i čak i karaktere za koje većina gledalaca nikada ne bi posumnjali da nisu stvarni),
- industrijski dizajn, umetnost i sl.
- 3D štampa – tehnologija proizvodnje trodimenzionalnih objekata na osnovu prototipova
- virtuelna realnost (Google Earth i VR)<sup>6</sup>.

<sup>6</sup><https://dp8hsntg6do36.cloudfront.net/558dcce661646d15d60a0000/410bf733-f844-4920-afa0-4dd27b6f6a87high.webm>

**1.8 Interesantni linkovi:**

- SIGGRAPH - vodeća konferencija u oblasti računarske grafike<sup>7</sup>; prikazuju se novi akademski rezultati iz oblasti računarske grafike, kao i komercijalni alati i postoji nekoliko konferencija iz bliskih oblasti koje se paralelno održavaju; zbornik radova sa ove konferencije, koji objavljuje ACM, smatra se najvažnijom referencom koju neko ko se praktično bavi ovom oblašću može da ima

**1.9 Pitana**

- 1.1 Čime se sve bavi računarska grafika?
- 1.2 Koje su osnovne poddiscipline računarske grafike?
- 1.3 Koja je razlika između renderovanja unapred i renderovanja unazad?
- 1.4 Šta se podrazumeva pod tim da je proces modelovanja hijerarhijski?
- 1.5 Koje dve paradigmе razlikujemo u računarskoj grafici? Koje su njihove osnovne prednosti, a šta su njihovi nedostaci?
- 1.6 Šta označava pojam interaktivne računarske grafike?
- 1.7 Koja je razlika između adresivosti i rezolucije uređaja za prikaz?
- 1.8 Kako se generiše slika kod vektorskih sistema?
- 1.9 Kakav izgled imaju kose linije u vektorskim, a kakav u rasterskim sistemima?
- 1.10 Čemu služi frejm bafer? Čemu služi video kontroler?
- 1.11 Navesti bar četiri oblasti u kojima se koristi računarska grafika.

---

<sup>7</sup>Link konferencije je <http://s2016.siggraph.org/live-streaming-sessions>.

## *Glava 2*

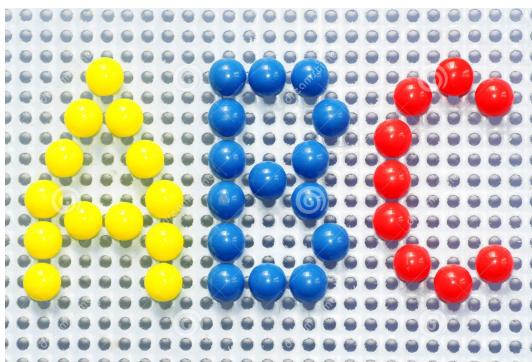
---

# 2D algoritmi

---

### **2.1 Algoritmi za crtanje 2D primitiva**

U **rasterskim sistemima** ekran je zadat matricom lokacija koje nazivamo **pikseli** (pixel = picture element). Pikseli su predstavljeni ili kao krugovi sa središtim koja su čvorovi celobrojne mreže ili kao kvadrati određeni celobrojnom mrežom. U narednim algoritmima, smatraće se da su pikseli reprezentovani kao krugovi sa središtim koja su čvorovi celobrojne mreže.



Takođe, dogovor je da se koordinatni početak nalazi u donjem levom ugлу ekrana i da se odgovarajući piksel nalazi na poziciji  $(0, 0)$ . Ovo je u skladu sa praksom u OpenGL-u. Jedna od  $2^N$  vrednosti intenziteta boje pridružena je svakom pikselu, gde je  $N$  broj bitova koji se koristi po pikselu. Crno-bele slike obično koriste jedan bajt po pikselu, dok slike u boji koriste jedan bajt po kanalu, a svakom pikselu su pridružena tri kanala: za crvenu, zelenu i plavu boju. Podaci o boji se čuvaju u frejm baferu (koji se često naziva i **mapa slike** ili **bitmapa**).

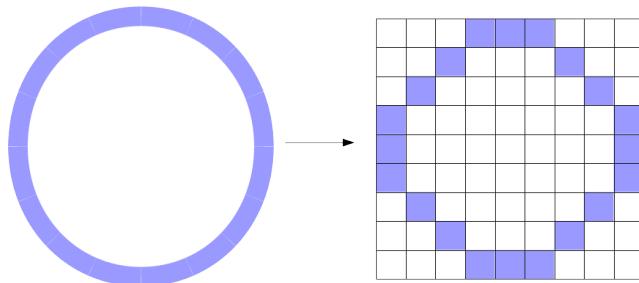
Rastersku sliku je moguće skalirati. Međutim, smanjenjem slike u odnosu na izvornu dimenziju gube se informacije o slici (npr. ako sliku smanjimo na 50% originalne dimenzije biće prikazan svaki drugi piksel) te slika postaje nazubljena. Povećavanjem slike u odnosu na izvornu dimenziju gube se detalji, te slika postaje mutna.

U **vektorskim sistemima** slika se predstavlja pomoću geometrijskih oblika kao što su tačke, linije, krive i poligoni, a oni se temelje na matematičkim jednačinama. Slika se čuva kao kolekcija figura, zajedno sa parametrima koji definišu kako će figura biti iscrtana i gde će se nalaziti na slici. Na primer, za čuvanje vektorske slike kruga potrebno je zapamtiti njegov poluprečnik, koordinate centra kruga, stil i boju linije i stil i boju unutrašnjosti kruga. Dakle, u ovom slučaju potrebno je čuvati dosta manje informacija nego u rasterskoj grafici, te je veličina odgovarajuće datoteke manja. Međutim, memorija koju slika zauzima zavisi od njene kompleksnosti. Takođe, moguće je skalirati (povećati, smanjiti) sliku bez gubitka na kvalitetu slike. Pomeranje figure i njen popunjavanje nema negativan uticaj na kvalitet slike kao kod rasterske grafike. Osnovni nedostatak vektorske grafike je to što nije pogodan za predstavljanje fotografija i fotorealističnih slika.

Problem kojim se bavimo u ovom poglavlju jeste crtanje osnovnih geometrijskih figura – duži i kruga na rasterskim sistemima. Proces transformisanja osnovnih objekata niskog nivoa u njihovu odgovarajuću reprezentaciju pikselima naziva se **sken konverzija** (eng. scan conversion) ili **rasterizacija**. Ovo je često aproksimacija objekta jer je frejm bafer predstavljen diskretnom mrežom. Na vektorskim sistemima i u vektorskim formatima – ovaj problem ne postoji.

Primitivne operacije koje su nam na raspolaganju su:

- `setpixel(x, y, color)` - postavlja boju piksela na poziciji  $(x, y)$  na datu boju *color*
- `getpixel(x,y)` - vraća boju piksela na poziciji  $(x, y)$



Slika 2.1: Sken konverzija kruga

Osnovne rasterske operacije nad pravom u 2D obuhvataju:

1. **sken konverziju** – za dati par piksela koji definiše kraj duži i datu boju, obojiti sve piksele koji leže na toj duži i
2. **odsecanje** (eng. clipping) – ako je jedna od krajnjih tačaka van granica regiona koji nam je od interesa, obojiti samo segment koji se nalazi unutar granica.

Postoji više načina na koje se može vršiti odsecanje. Jedna od tehnika jeste da se izvrši odsecanje primitive izračunavanjem preseka sa granicama regiona, pre nego što se izvrši sken konverzija. Prednost ovakvog prisupa jeste

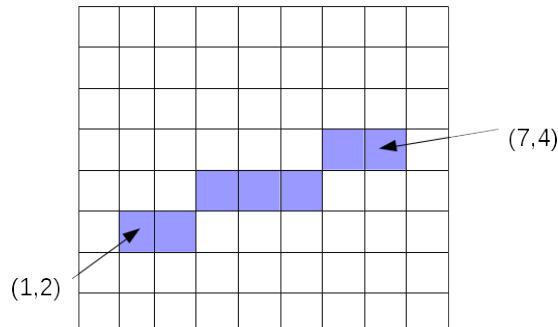
što se sken konverzija vrši samo nad odsečenom verzijom primitive, umesto sa (potencijalno mnogo složenijom) originalnom primitivom. Ova tehnika se uglavnom koristi prilikom odsecanja duži, pravougaonika i poligona za koje su algoritmi odsecanja relativno jednostavniji.

Najjednostavnija verzija odsecanja (grubom silom) podrazumeva da se uradi sken konverzija kompletne primitive, ali da se upišu samo vidljivi pikseli. U principu, ovo se radi tako što se za svaki piksel proverava da li njegove koordinate pripadaju regionu od interesa. Prednost ovog pristupa je u tome što je primenljiv na proizvoljni region odsecanja.

S obzirom na to da rasterski ekrani pozivaju algoritme za odsecanje i sken konverziju svaki put kada se slika izmeni, ovi algoritmi moraju ne samo proizvesti vizuelno zadovoljavajuće slike, već, takođe, moraju raditi što je brže moguće. U nastavku ćemo videti na koje je sve načine moguće efikasno izvesti ove dve osnovne operacije.

## 2.2 Crtanje duži (eng. scan converting line)

Zadatak: postaviti boje piksela tako da aproksimiraju duž od tačke  $(x_0, y_0)$  do tačke  $(x_1, y_1)$ .



Slika 2.2: Primer sken konverzije duži od tačke  $(1,2)$  do tačke  $(7,4)$  – na slici je svaki piksel predstavljen kvadratom određenim celobrojnom mrežom

Zahtevi:

- Niz piksela treba da bude što bliži idealnoj liniji i treba da prolazi kroz krajnje tačke.
- Slika duži treba da bude nezavisna od redosleda kojim su date njene krajnje tačke.
- Crtanje treba da bude što je moguće brže.
- Sve linije treba da budu iste osvetljenosti i sve tačke svake linije treba da budu iste osvetljenosti (ili da tako izgledaju) bez obzira na njihov nagib i dužinu.
- Dodatni kriterijumi:
  - Omogućiti crtanje debljih linija i linija nacrtanih određenim stilom.

- Definisati varijante za crtanje linije u tehnički antialiasing.

Duž koju je potrebno nacrtati data je koordinatama svojih krajnjih tačaka  $(x_0, y_0)$  i  $(x_1, y_1)$ . Eksplisitna jednačina prave je:

$$y = mx + B.$$

Ukoliko je prava zadata dvema tačkama  $(x_0, y_0)$  i  $(x_1, y_1)$ , zamenom ovih vrednosti u jednačinu prave dobijamo vrednosti koeficijenata  $m$  i  $B$ . Tačnije, iz jednačina  $y_0 = mx_0 + B$  i  $y_1 = mx_1 + B$  dobijamo:

$$m = \frac{y_1 - y_0}{x_1 - x_0},$$

a koeficijent  $B$  onda računamo kao:

$$B = y_0 - mx_0.$$

Ako zamenimo ovu vrednost za  $B$  u opštu jednačinu prave, dobijamo jednačinu prave kroz jednu zadatu tačku  $(x_0, y_0)$ :

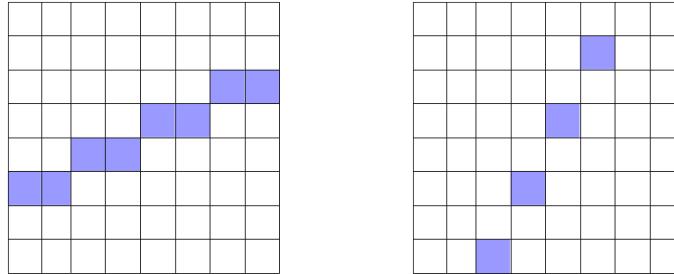
$$y = m(x - x_0) + y_0.$$

### 2.2.1 Osnovni zadatak

Za početak razmatraćemo samo jednobitne slike kod kojih svaki piksel može biti samo uključen ili isključen. Želimo da nacrtamo duž debljine 1.

Najčešće ćemo se susresti sa zadatkom da nacrtamo horizontalnu, vertikalnu i duž koeficijenta 1 ili -1. Međutim, ovakve duži se crtaju trivijalno. Problem je kako nacrtati ostale duži.

Razmatraćemo za početak samo duži koeficijenta  $m$  takvih da je  $|m| < 1$  (tj.  $-1 < m < 1$ ); ostali slučajevi rešavaju se simetrično, pogodnim izmenama u izloženim algoritmima.



Slika 2.3: Prikaz rasterizacije duži čiji je koeficijent pravca (a)  $m < 1$ , (b)  $m > 1$  ukoliko bi se u svakoj koloni označavao po jedan piksel

U slučajevima kada je  $m < 1$  i kada je  $m > 1$  crta se različit broj piksela pa su prave različitog nivoa osvetljenosti. Stoga za duži koeficijenta između -1 i 1, u svakoj koloni treba da bude označen tačno jedan piksel, dok za duži van tog opsega, u svakoj vrsti treba da bude označen tačno jedan piksel.

Moguće je razmatrati različite kriterijume ocene kvaliteta dobijene rasterizovane duži. Mi ćemo ovde birati piksele čija su rastojanja od idealne duži najmanja.

### 2.2.2 Algoritam grube sile

Najjednostavnija strategija za crtanje duži određene tačkama  $(x_0, y_0)$  i  $(x_1, y_1)$  sastoji se od narednih koraka:

- izračuna se koeficijent pravca  $m = \frac{y_1 - y_0}{x_1 - x_0}$
- $x$  se povećava za 1 počev od  $x$  koordinate početne tačke  $(x_0, y_0)$  duži,
- za svako  $x_i$  računa se vrednost  $y_i = mx_i + B = m(x_i - x_0) + y_0$
- osvetljava se piksel sa koordinatama  $(x_i, round(y_i))$ , pri čemu je  $round(y_i) = \lfloor y_i + 0.5 \rfloor$

Nedostaci ovog pristupa su sledeći:

- ako je  $x_1 < x_0$  ništa se ne crta  
(rešenje: promeniti poređak tačaka ukoliko je  $x_1 < x_0$ )
- algoritam je neefikasan zbog broja operacija i korišćenja realne aritmetike  
- svaki korak zahteva oduzimanje, sabiranje, množenje i zaokruživanje  
jer je  $y$  promenljiva realnog tipa  
(rešenje: pogledati naredne algoritme)

```
procedure Line_brute_force(x0, y0, x1, y1 : integer);
var
    x : integer;
    dx, dy, y, m : real;

begin
    dx := x1 - x0;
    dy := y1 - y0;
    m := dy/dx;
    for x := x0 to x1 do
    begin
        y := m * (x - x0) + y0;
        setpixel(x, round(y));
    end
end.
```

Slika 2.4: Algoritam grube sile za crtanje duži

### 2.2.3 Osnovni inkrementalni algoritam

Gornja veza se može izraziti i na sledeći način:

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = mx_i + m\Delta x + B = y_i + m\Delta x$$

Ako je  $\Delta x = 1$ , onda je, prema gornjoj jednačini,  $y_{i+1} = y_i + m$ , tj. umesto da svaki put računamo iznova vrednost za  $y$ , prilikom povećanja vrednosti

```

procedure Line_increment_basic(x0, y0, x1, y1 : integer);
var
    x : integer;
    dx, dy, y, m : real;

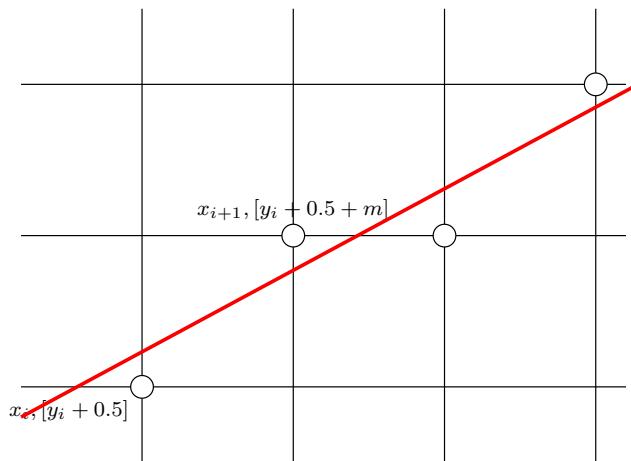
begin
    dx := x1 - x0;
    dy := y1 - y0;
    m := dy/dx;
    y := y0;
    for x := x0 to x1 do
        begin
            setpixel(x, round(y));
            y := y + m
        end
    end.

```

Slika 2.5: Inkrementalni algoritam za crtanje duži

$x_i$  za 1, vrednost za  $y_i$  treba povećati za  $m$  i time je eliminisano množenje. Ovo ilustruje prirodu **inkrementalnih algoritama** — u svakom koraku izračunavanja pravimo na osnovu prethodnog koraka. Primetimo da se u ovom pristupu ne treba eksplisitno starati o slobodnom članu  $B$ ; za početnu tačku uzima se tačka sa najmanjom  $x$  koordinatom.

Ako je  $|m| > 1$ , onda je prilikom uvećanja  $x$  koordinate za 1, na osnovu veze  $y_{i+1} = y_i + m$  potrebno uvećati  $y$  koordinatu za vrednost veću od 1. Stoga je potrebno obrnuti uloge osama  $x$  i  $y$ , inkrementiranjem  $y$  koordinate za 1, a  $x$  koordinate za  $\Delta x = \Delta y/m = 1/m$



Ovaj algoritam se često zove *DDA* (digital differential algorithm); DDA je

mehanički uređaj koji rešava diferencijalne jednačine numeričkim metodama.

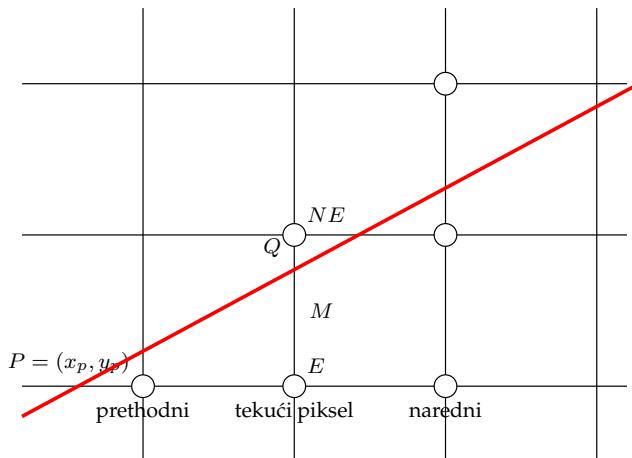
Nedostatak ovog pristupa je to što je vrednost  $m$  realna vrednost i, kao takva, ona može biti izračunata neprecizno; u uobičajenim okolnostima (relativno kratke duži) ta nepreciznost nije bitna. Ali ako je recimo  $m$  sa vrednosti 0.99 zaokruženo na vrednost 0.9, duži dužine veće od 10 biće neprecizno nacrtane.

#### 2.2.4 Midpoint algoritam za crtanje duži (varijanta Bresenhamovog algoritma)

Ključni nedostaci osnovnog inkrementalnog algoritma su potreba zaokruživanja i to što promenljive moraju da budu realnog tipa. Takođe, s obzirom na to da se nagib računa kao razlomak, vertikalne prave se moraju posebno razmatrati.

Bresenham (1965) je razvio algoritam za crtanje duži koji koristi samo celobrojnu aritmetiku. Ista tehnika može da se koristi i za krug. Algoritam je originalno razvijen za digitalni ploter. Dokazano je da se algoritmom minimizuju rastojanja od idealne duži.

Pitteway (1967) i Van Aken (1984) su razvili **midpoint algoritam** koji se za duži i celobrojne krugove ponaša isto kao Bresenhamov algoritam, ali može da se uopšti na proizvoljne krive drugog reda.



Pretpostavimo da je nagib prave između 0 i 1 (ostale vrednosti za nagib se mogu razmatrati pogodnim simetrijama oko glavnih osa). Neka je potrebno spojiti tačke  $(x_0, y_0)$  i  $(x_1, y_1)$ , pri čemu smo sa  $(x_0, y_0)$  označili donju levu krajnju tačku, a sa  $(x_1, y_1)$  gornju desnu.

Pretpostavimo da smo označili piksel  $P(x_p, y_p)$  i da treba da izaberemo između piksela jednu poziciju desno ( $E=$ east) i piksela jednu poziciju desno i jednu poziciju iznad ( $NE=$ northeast). U Bresenhamovom algoritmu računa se vertikalno rastojanje od tačke  $E$  i tačke  $NE$  do presečne tačke prave koju rasterizujemo sa pravom  $x = x_p + 1$ . U midpoint algoritmu posmatramo odnos tačke  $M = midpoint$  (središte duži određene tačkama  $E$  i  $NE$ ) sa pravom koju rasterizujemo. Ako je tačka  $M$  "ispod" prave koja sadrži duž, onda je

pravoj bliža  $NE$ , a inače  $E$ . Greška, računata kao vertikalno rastojanje između odabranog piskela i date prave, je uvek  $\leq 0.5$

Kako proveriti da li se prava nalazi iznad ili ispod tačke  $M$  korišćenjem samo celobrojne aritmetike? Predstavimo pravu kao implicitnu funkciju sa koeficijentima  $a, b$  i  $c$ :

$$F(x, y) = ax + by + c = 0$$

Ako je  $dx = x_1 - x_0$  i  $dy = y_1 - y_0$ , pravu možemo zapisati i u obliku:

$$y = \frac{dy}{dx}x + B$$

i dobijamo narednu implicitnu jednačinu prave:

$$F(x, y) = dy \cdot x - dx \cdot y + B \cdot dx = 0$$

Pritom važi:  $a = dy$ ,  $b = -dx$ ,  $c = B \cdot dx$

Lako se može proveriti da je vrednost  $F(x, y)$  jednaka 0 za tačke na pravoj, pozitivna za tačke "ispod" prave, a negativna za tačke "iznad" prave. Dakle, da bismo odredili položaj tačke  $M$  potrebno je jedino odrediti znak izraza  $F(M) = F(x_p + 1, y_p + \frac{1}{2})$ . S obzirom na to da se odluka bazira na vrednosti ove funkcije, vrednost  $d = F(x_p + 1, y_p + \frac{1}{2})$  zovemo **promenljiva odlučivanja**. Ako je  $d < 0$ , onda se tačka  $M$  nalazi iznad prave, te biramo piksel  $E$ ; ako je  $d > 0$ , onda se tačka  $M$  nalazi ispod prave i onda biramo piksel  $NE$ ; ako je  $d = 0$ , onda je sve jedno i, po dogovoru, biramo piksel  $E$ .

Šta se dešava sa pozicijom tačke  $M$  i vrednošću promenljive  $d$  za narednu liniju mreže? Ako smo odabrali piksel  $E$  ili  $NE$ , kako odabratи sledeći piksel?

U odlučivanju se koristi prethodna vrednost promenljive odlučivanja:

- Ako smo odabrali  $E$ :

$$\begin{aligned} d_{new} &= F(x_p + 2, y_p + \frac{1}{2}) = a(x_p + 2) + b(y_p + \frac{1}{2}) + c = \\ &= a + a(x_p + 1) + b(y_p + \frac{1}{2}) + c = a + d_{old} \end{aligned}$$

Tj.:

$$d_{new} = d_{old} + a = d_{old} + dy$$

$$d_{new} = d_{old} + \Delta_E$$

$\Delta_E = dy$  se može posmatrati kao korektivni faktor i nazivamo ga **razlikom unapred**

- Ako smo odabrali  $NE$ :

$$\begin{aligned} d_{new} &= F(x_p + 2, y_p + \frac{3}{2}) = a(x_p + 2) + b(y_p + \frac{3}{2}) + c = \\ &= a + b + a(x_p + 1) + b(y_p + \frac{1}{2}) + c = a + b + d_{old} \end{aligned}$$

Tj.:

$$d_{new} = d_{old} + a + b = d_{old} + dy - dx = d_{old} + \Delta_{NE}$$

gde je  $\Delta_{NE} = dy - dx$

```

procedure Line_midpoint(x0, y0, x1, y1 : integer);
var
    dx, dy, x, y, f : integer;

begin
    dx := x1 - x0;
    dy := y1 - y0;
    y := y0;
    f := 2*dy - dx;
    for x := x0 to x1 do
        begin
            setpixel(x, y);
            if (f <= 0)
                begin
                    f := f + 2*dy;
                end
            else
                begin
                    y := y+1;
                    f := f + 2*(dy-dx);
                end
        end
    end.
end.
```

Slika 2.6: Midpoint algoritam za crtanje duži

U svakom koraku algoritam bira između dva piksela na osnovu znaka promenljive odlučivanja koja se računa u prethodnoj iteraciji. Nakon toga ažurira se vrednost promenljive odlučivanja dodavanjem vrednosti  $\Delta_E$  ili  $\Delta_{NE}$  u zavisnosti od odabira piksela. Ovim se sve svodi samo na prosta sabiranja.

Za prvu tačku  $(x_0, y_0)$  važi  $F(x_0, y_0) = 0$  (jer je na pravoj). Za prvo središte važi:

$$d_{start} = F(x_0 + 1, y_0 + \frac{1}{2}) = a(x_0 + 1) + b(y_0 + \frac{1}{2}) + c =$$

$$ax_0 + by_0 + c + a + b/2 = F(x_0, y_0) + a + b/2 = a + b/2$$

Da bi se izbeglo deljenje sa 2 u  $d_{start}$ , sve vrednosti se množe sa dva (a svi relevantni znakovi ostaju isti). Dakle:

- $d_{start} = 2a + b = 2dy - dx$
- ako je  $d_{old} \leq 0$ , onda  $d_{new} = d_{old} + 2dy$
- ako je  $d_{old} > 0$ , onda je  $d_{new} = d_{old} + 2dy - 2dx$

### 2.2.5 Crtanje duži – dodatna pitanja

Među problemima koje treba razmotriti jeste i poredak zadatih tačaka: duži  $P_0P_1$  i  $P_1P_0$  treba da se crtaju isto (tj. da se sastoje od istih piksela). Jedino mesto gde je izbor piksela zavisan od smera prave je u slučaju kada prava prolazi tačno kroz središnju tačku i kada je vrednost promenljive odlučivanja 0. Stoga odlučujemo da radimo simetrično, tj. kada se ide “sleva nadesno” bira se  $E$  za  $d = 0$ , a kada se ide “zdesna nalevo” bira se  $SW$  (south-west)).

Može i jednostavno da se poredak tačaka svede na poredak “sleva nadesno” ali to ne daje ispravan rezultat ako se koriste stilovi.

Drugi problem koji treba razmotriti tiče se intenziteta boja. Razmotrimo duži od deset tačaka sa nagibom 0 i nagibom 1. Obe imaju po deset piksela, ali je ova druga duža  $\sqrt{2}$  puta od prve. Toliko je puta i intenzitet prve duži veći od intenziteta druge. Ako su na uređaju raspoložive samo dve boje, to ne može da se popravi. Ako su na raspolaganju nijanse boje, onda one mogu da se upotrebe i biraju u zavisnosti od nagiba duži. U te svrhe koristi se i tehnika antialiasing.

## 2.3 Crtanje kruga

Zadatak: postaviti boje piksela tako da aproksimiraju krug sa centrom u pikselu  $(0,0)$ .

Zahtevi:

- Slično kao za duži: da niz piksela bude što bliže idealnom krugu, da crtanje bude što je moguće brže, da nacrtani krug bude “neprekidan” itd.
- Posebno se razmatra varijanta za crtanje duži u tehnici antialiasing.

### 2.3.1 Osnovni zadatak

- Debljina kruga 1.
- Jednobitna slika.
- Crtanje tačke je primitiva.
- Pretpostavlja se da je središte kruga tačka (tj. piksel)  $(0,0)$ ; algoritam se može trivijalno uopštiti tako da radi i za druge slučajeve
- Dovoljno je odrediti sve tačke kruga u jednom kvadrantu; na osnovu tih tačaka se određuju i ostale tačke (na osnovu simetrije), mada u realnim implementacijama može da se, zbog efikasnosti, sličan kod navede više puta.

### 2.3.2 Naivni algoritam

- Iz jednačine kruga:

$$x^2 + y^2 = R^2$$

sledi:

$$y = \pm \sqrt{R^2 - x^2}$$

- Za  $i = 1, 2, 3, \dots$ , pri čemu se  $x_i$  inkrementira za po 1, treba osvetliti tačku:

$$(x_i, [\sqrt{R^2 - x_i^2} + 0.5])$$

Tačno (koliko je to moguće), ali neefikasno.

- Sličan neefikasan metod, korišćenjem parametarske jednačine kruga: crtati tačke

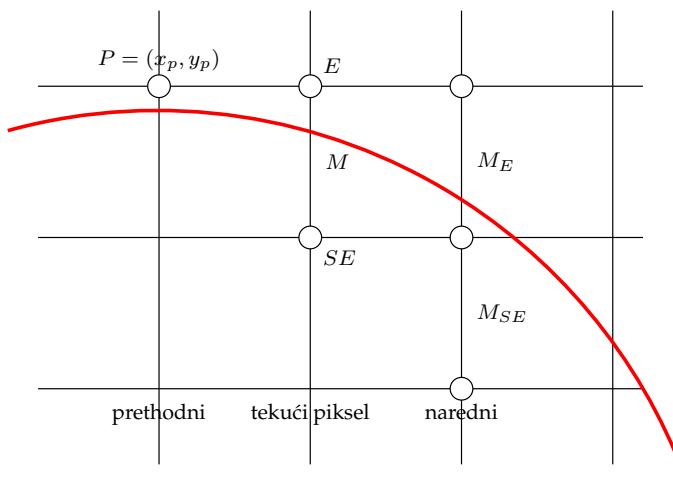
$$([R \cos \varphi_i + 0.5], [R \sin \varphi_i + 0.5])$$

za  $\varphi_i = 1^\circ, 2^\circ, 3^\circ, \dots, 90^\circ$ .

- Kako vrednost  $x$  raste, povećavaju se praznine između tačaka; to je lako rešiti korišćenjem i simetrije kruga po još jednoj osi (pravoj  $y = x$ ).

### 2.3.3 Bresenham (midpoint) inkrementalni algoritam

- Bresenham (1977): originalno razvijen za digitalni ploter.
- Midpoint algoritam: varijanta Bresenhamovog algoritma.
- Razmatra se samo osmina kruga.
- Vrednost  $x$  ide od 0 do  $R/\sqrt{2}$
- Osnovna ideja slična je ideji za crtanje duži: u svakom koraku treba odabrati jednu od dve moguće tačke.



- Vrednost  $F(x, y) = x^2 + y^2 - R^2$  je jednaka 0 u tačkama koje pripadaju krugu, pozitivna je u spoljašnjosti kruga, a negativna u unutrašnjosti kruga.
- Ako je tačka  $M$  u unutrašnjosti kruga, onda je tačka  $E$  bliža krugu nego tačka  $SE$ . Ako je tačka  $M$  u spoljašnjosti kruga, onda je tačka  $SE$  bliža krugu.

- Promenljiva odlučivanja jednaka je:

$$d_{old} = F(x_p + 1, y_p - \frac{1}{2}) = (x_p + 1)^2 + (y_p - \frac{1}{2})^2 - R^2$$

- Ako je  $d_{old} < 0$ , izabrana je tačka  $E$ , ako je  $d_{old} > 0$ , izabrana je tačka  $SE$ . Ako je  $d_{old} = 0$ , onda je izbor tačke stvar dogovora i bira se tačka  $SE$ .

- Ako je  $d_{old} < 0$ , tačka  $E$  je izabrana i važi:

$$\begin{aligned} d_{new} &= F(x_p + 2, y_p - \frac{1}{2}) = (x_p + 2)^2 + (y_p - \frac{1}{2})^2 - R^2 = \\ &= d_{old} + (2x_p + 3) \end{aligned}$$

- Ako je  $d_{old} \geq 0$ , tačka  $SE$  je izabrana i važi:

$$\begin{aligned} d_{new} &= F(x_p + 2, y_p - \frac{3}{2}) = (x_p + 2)^2 + (y_p - \frac{3}{2})^2 - R^2 = \\ &= d_{old} + (2x_p - 2y_p + 5) \end{aligned}$$

- U slučaju  $d_{old} < 0$  koristi se  $\Delta_E = 2x_p + 3$ , a u slučaju  $d_{old} \geq 0$  koristi se  $\Delta_{SE} = 2x_p - 2y_p + 5$ . Primetimo da obe ove razlike zavise od koordinata tačke  $P$ .

- Početni uslov (za celobrojni poluprečnik):

$$\begin{aligned} F(0, R) &= 0 \\ F(1, R - \frac{1}{2}) &= 1 + (R^2 - R + \frac{1}{4}) - R^2 = \frac{5}{4} - R \end{aligned}$$

- Nije dobro to što prva vrednost za  $d$  nije celobrojna
- Koristićemo zamenu  $h = d - \frac{1}{4}$ : tada je inicijalna vrednost  $h = 1 - R$ , a poređenje  $d < 0$  postaje  $h < -\frac{1}{4}$ . Međutim, kako je inicijalna vrednost celobrojna i kako su vrednosti koje se dodaju ( $\Delta_E$  i  $\Delta_{SE}$ ) celobrojne, ovaj uslov može da se zameni sa  $h < 0$ ; tako uvedenu promenljivu  $h$  na kraju ipak označimo sa  $d$ .

#### 2.3.4 Bresenham inkrementalni algoritam — unapređena verzija

- Vrednosti  $\Delta_E$  i  $\Delta_{SE}$  su linearne i za njihovo izračunavanje takođe možemo koristiti tehniku inkrementalnog uvećavanja. Tako će dodatno biti smanjen broj sabiranja/oduzimanja.
- Ako je u jednom koraku izabrana tačka  $E$ , onda se tačka izračunavanja, referentna tačka pomera iz  $(x_p, y_p)$  u  $(x_p + 1, y_p)$ . Vrednost  $\Delta_{Eold}$  u  $(x_p, y_p)$  je jednaka  $2x_p + 3$ . Dakle, vrednost  $\Delta_{Enew}$  (u tački  $(x_p + 1, y_p)$ ) je jednaka

$$\Delta_{Enew} = 2(x_p + 1) + 3 = \Delta_{Eold} + 2$$

Dodatno, vrednost  $\Delta_{SEold}$  u  $(x_p, y_p)$  jednaka je  $2x_p - 2y_p + 5$  a vrednost  $\Delta_{SEnew}$  (u tački  $(x_p + 1, y_p)$ ) je jednaka

$$\Delta_{SEnew} = 2(x_p + 1) - 2y_p + 5 = \Delta_{SEold} + 2$$

(ove vrednosti zovemo **razlike drugog reda**)

```

procedure MidpointCircle (radius : integer);
var
  x, y, d : integer;
begin
  x := 0;
  y := radius;
  d := 1-radius;
  setpixel(x, y);

  while (y > x) do
    begin
      if d < 0 then { select E }
        begin
          d := d + 2*x + 3;
          x := x + 1;
        end
      else { select SE }
        begin
          d := d + 2*(x-y) + 5;
          x := x + 1;
          y := y - 1;
        end
      setpixel(x, y);
    end
  end.

```

Slika 2.7: Algoritam za crtanje kruga

- Analogno, ako je izabrana tačka *SE*, onda se tačka izračunavanja, referentna tačka pomera iz  $(x_p, y_p)$  u  $(x_p+1, y_p-1)$ . Vrednost  $\Delta_{Eold}$  u  $(x_p, y_p)$  je jednaka  $2x_p + 3$ . Dakle, vrednost  $\Delta_{Enew}$  (u tački  $(x_p + 1, y_p - 1)$ ) je jednaka

$$\Delta_{Enew} = 2(x_p + 1) + 3 = \Delta_{Eold} + 2$$

Dodatno, vrednost  $\Delta_{SEold}$  u  $(x_p, y_p)$  jednaka je  $2x_p - 2y_p + 5$  a vrednost  $\Delta_{SEnew}$  (u tački  $(x_p + 1, y_p - 1)$ ) je jednaka

$$\Delta_{SEnew} = 2(x_p + 1) - 2(y_p - 1) + 5 = \Delta_{SEold} + 4$$

### 2.3.5 Crtanje kruga čije središte nije u koordinatnom početku

- Ako središte kruga nije tačka  $(0, 0)$  nego  $(a, b)$ , koristi se algoritam sličan navedenom.
- Nije isplativo za svaki piksel pojedinačno dodavati  $a$  i  $b$  (u odnosu na piksele kruga sa središtem  $(0, 0)$ ).

```

procedure MidpointCircle2 (radius : integer);
var
    x, y, d, deltaE, deltaSE : integer;
begin
    x := 0;
    y := radius;
    d := 1 - radius;
    deltaE := 3;
    deltaSE := -2*radius + 5;
    setpixel(x, y);

    while (y>x) do
        begin
            if d < 0 then      { select E }
                begin
                    d := d + deltaE;
                    deltaE := deltaE + 2;
                    deltaSE := deltaSE + 2;
                    x := x + 1;
                end
            else      { select SE }
                begin
                    d := d + deltaSE;
                    deltaE := deltaE + 2;
                    deltaSE := deltaSE + 4;
                    x := x + 1;
                    y := y - 1;
                end
            setpixel(x, y);
        end
    end.

```

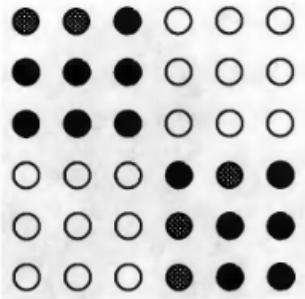
Slika 2.8: Algoritam za crtanje kruga — unapredjena verzija

- Sve razlike prvog i drugog reda su iste (ako je poluprečnik isti) bez obzira na središte kruga.
- Razlikuje se samo početni piksel – umesto  $(0, R)$ , prvi uključeni piksel treba da bude  $(a, R + b)$ , a umesto uslova  $y > x$  treba koristiti uslov  $y-b > x-a$  ili efikasnije  $y > x+(b-a)$ .

## 2.4 Popunjavanje poligona

### 2.4.1 Popunjavanje proizvoljnog regiona

Ponekad, nakon crtanja niza primitiva, želimo i da ih obojimo. Kažemo da je region **4-povezan** ako se svaka dva piksela tog regiona mogu povezati nizom piksela korišćenjem samo poteza nagore, nadole, uлево и udesno. Nasuprot ovome, region je **8-povezan** ako se svaka dva piksela tog regiona mogu povezati nizom piksela korišćenjem poteza nagore, nadole, uлево, udesno, gore-levo, gore-desno, dole-levo, dole-desno. Primetimo da je svaki 4-povezan region i 8-povezan.

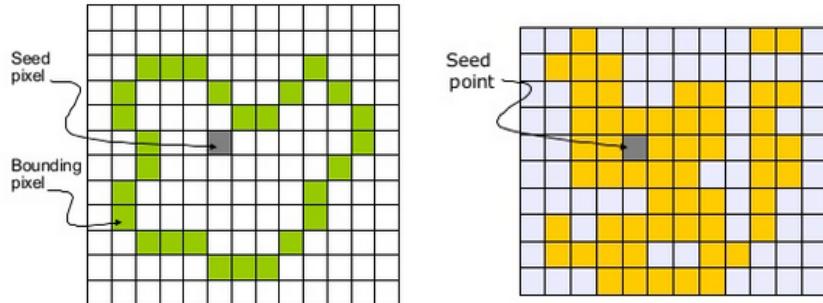


Slika 2.9: Crni pikseli na slici definišu 8-povezan region koji nije 4-povezan

Svaki algoritam za popunjavanje regiona se može podeliti na četiri komponente:

- *metod propagacije* koji utvrđuje koju narednu tačku treba razmatrati,
- *proceduru startovanja* koja inicijalizuje algoritam,
- *proceduru ispitivanja pripadnosti* kojom se utvrđuje da li je piksel unutar regiona i treba da se oboji i
- *proceduru postavljanja vrednosti* kojom se menja boja piksela.

Region se može definisati na dva različita načina: za svaki od njih koristimo početni piksel  $P$ . **Region definisan unutrašnjošću** jeste najveći povezani region piksela čija je boja ista kao i boja piksela  $P$ . **Region definisan granicom** je najveći povezani region piksela čija boja nije jednaka nekoj graničnoj vrednosti. S obzirom na to da većina algoritama za popunjavanje regiona radi rekurzivno i rekurzija se zaustavlja kada naredni piksel već ima novu boju, problemi se mogu pojaviti ako se nova boja pojavi u regionu koji je definisan granicom, jer se neke grane rekurzije mogu prerano odseći.



Slika 2.10: Na slici levo prikazan je region definisan granicom: zeleni pikseli na slici levo ograničavaju unutrašnjost regiona. Na slici desno prikazan je region definisan unutrašnjošću: žuti pikseli na slici desno označavaju unutrašnjost regiona.

Algoritmi kojima se popunjava region definisan unutrašnjošću nazivaju se **flood-fill** algoritmi, dok se algoritmi koji popunjavaju region definisan granicom nazivaju **boundary-fill** algoritmi.

Najjednostavniji metod propagacije jeste da se pomerimo iz početnog piksela u sva četiri ili osam smerova i da rekurzivno primenimo algoritam. U ovim algoritmima se pretpostavlja da je već određena unutrašnjost regiona.

**Flood-fill** algoritam za popunjavanje proizvoljne 4-povezane konture – ne nužno poligona ima sledeću formu:

```
procedure FloodFill4(x, y, old_color, new_color : integer);
begin
  if (getPixel(x,y) == old_color) then
    begin
      setPixel(x,y,new_color);
      FloodFill4(x-1,y,old_color,new_color);
      FloodFill4(x+1,y,old_color,new_color);
      FloodFill4(x,y-1,old_color,new_color);
      FloodFill4(x,y+1,old_color,new_color);
    end
  end.
```

U flood-fill algoritmu kreće se od početne vrednosti piksela  $(x, y)$  u unutrašnjosti oblasti – **seed-a** počev od kog se popunjava oblast. Za njega se proverava da li je vrednost boje stara; ako jeste, on se boji novom bojom; ukoliko je vrednost boje razmatranog piksela izmenjena ne treba se vraćati u taj piksel. Kod za 8-povezanu verziju imao bi 8 rekurzivnih poziva umesto 4.

**Boundary-fill** algoritam za popunjavanje proizvoljne 4-povezane konture ima narednu formu:

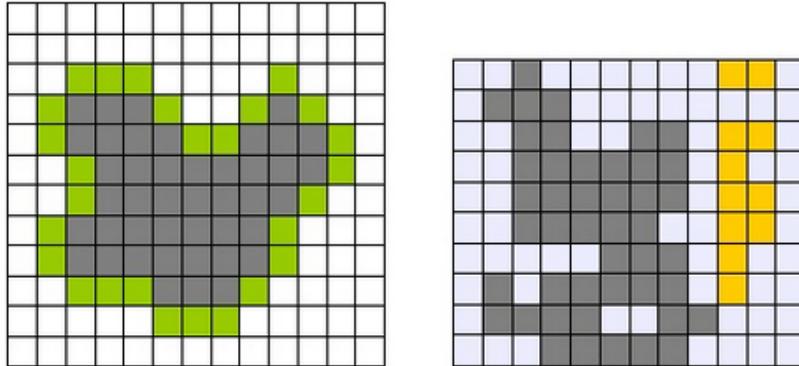
```
procedure BoundaryFill4(x, y, boundary_color, new_color : integer);
```

```

begin
    color c = getPixel(x,y);
    if (c != boundary_color && c != new_color) then
        begin
            setPixel(x,y,new_color);
            BoundaryFill4(x-1,y,boundary_color,new_color);
            BoundaryFill4(x+1,y,boundary_color,new_color);
            BoundaryFill4(x,y-1,boundary_color,new_color);
            BoundaryFill4(x,y+1,boundary_color,new_color);
        end
    end.

```

Ova vrsta algoritama, iako jednostavna, ima visok stepen rekurzije i može da uzrokuje prekoračenje steka rekurzije. Postoje mnoga unapređenja ovih algoritama koja mi ovde nećemo razmatrati.



Slika 2.11: Na slici levo prikazan je region definisan granicom nakon popunjavanja sivim bojom algoritmom boundary-fill. Na slici desno prikazan je region definisan unutrašnjošću nakon popunjavanja sivom bojom algoritmom flood-fill (prepostavka je da je region 4-povezan).

#### 2.4.2 Naivni algoritam za popunjavanje poligona

Zadatak koji razmatramo jeste kako obojiti sve piksele koji se nalaze u unutrašnjosti poligona istom bojom (ili istim uzorkom).

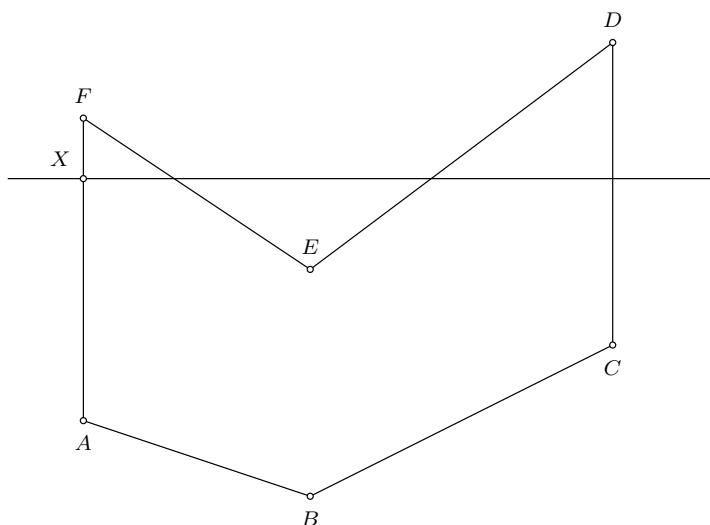
Najjednostavniji način popunjavanja poligona bio bi da se za svaku tačku na slici (pojedinačno) proveri da li pripada unutrašnjosti poligona ili ne. Postoje efikasni algoritmi za proveravanje da li tačka pripada unutrašnjosti poligona. Najpre se određuje najmanji pravougaonik (sa stranicama paralelnim koordinatnim osama) koji sadrži dati poligon, a zatim se za svaku tačku tog pravougaonika proverava da li pripada unutrašnjosti poligona ili ne.

#### 2.4.3 Scan popunjavanje poligona

Vremenom su razvijeni efikasniji algoritmi za popunjavanje poligona. Oni uglavnom rade sa **scan linijama** – to su horizontalne linije na rasterskim sistemima.

Algoritam za scan popunjavanje poligona koji ćemo opisati u nastavku radi i nad konveksnim i nad konkavnim poligonima, čak i sa onima koji imaju samopresecanje ili rupe u unutrašnjosti. Na slici 2.12 prikazan je poligon i jedna scan linija koja prolazi kroz njega. Potrebno je na svakoj scan liniji odrediti piksele koji se nalaze u unutrašnjosti poligona.

Ideja algoritma je sledeća: na svakoj scan liniji treba odrediti tačke koje pripadaju stranicama poligona. Za svaki presek scan linije sa stranicom poligona inkrementiramo odgovarajući brojač i razmatramo njegovu parnost — inicijalno je brojač paran (tj. ima vrednost 0) i njegova parnost se menja svaki put prilikom nailaska na tačku neke stranice. Kod tačaka sa neparnim stanjem treba započeti bojenje, a kod tačaka sa parnim treba ga zaustaviti.



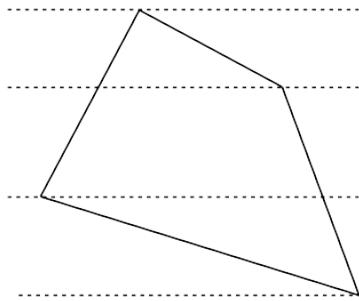
Slika 2.12: Poligon i jedna scan linija

Pritom se u računanju parnosti, za jednu stranicu poligona računa tačka preseka sa najmanjom  $y$  koordinatom ( $y_{min}$ ), ali ne i tačka sa najvećom  $y$  koordinatom ( $y_{max}$ ). Ovim se u stvari obezbeđuje da ako je u pitanju lokalni minimum ili maksimum, parnost se neće promeniti; inače se parnost menja.

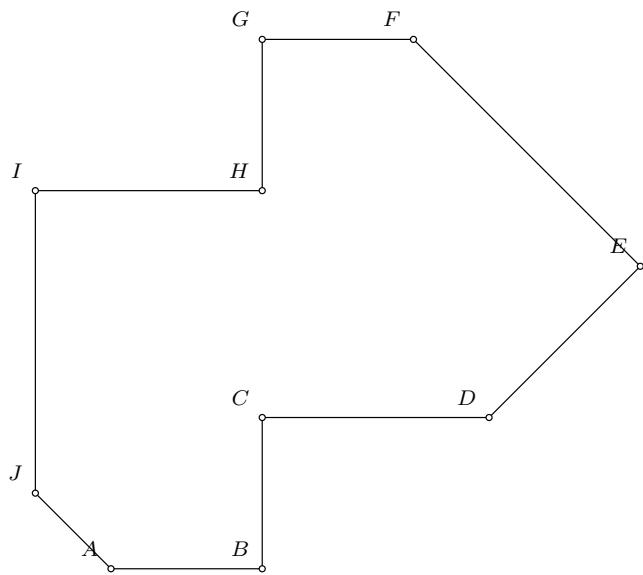
Potrebno je na odgovarajući način obraditi horizontalne ivice, tako što kod njih ne treba brojati nijednu tačku preseka. Razmotrimo poligon prikazan na slici 2.14 i neke od slučajeva. U slučaju ivice  $AB$ , teme  $A$  je teme sa minimalnom  $y$  koordinatom stranice  $JA$ , a stranica  $AB$  ne utiče na parnost brojača. Dakle, vrednost brojača je neparna i crta se ivica  $AB$ . Vertikalna ivica  $BC$  ima svoju minimalnu koordinatu u tački  $B$ , a stranica  $AB$  opet ne utiče na vrednost brojača, stoga vrednost brojača u temenu  $B$  postaje parna i prestajemo sa iscrtavanjem piksela na ovoj scan liniji.

U temenu  $J$  stranica  $IJ$  ima minimalnu  $y$  koordinatu, a stranica  $JA$  nema, stoga brojač postaje neparan i iscrtavaju se pikseli sve do stranice  $BC$ .

S obzirom na to da su horizontalne stranice poligona susedne nekim drugim



Slika 2.13: Računanje parnosti za krajnje tačke duži

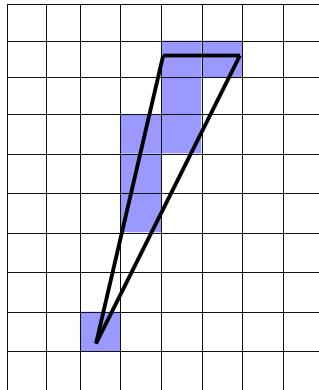


Slika 2.14: Primer poligona čije su neke ivice horizontalne

stranicama, efekat pravila da se kod horizontalnih ivica ne broje tačke preseka je da se donje horizontalne ivice poligona boje, a gornje ne.

Ovakav algoritam ne boji „gornje“ i „desne“ „krajnje“ tačke; on ne boji neke potrebne tačke (tačke na stranicama ionako je lako obojiti), a sigurno ne boji nijednu pogrešnu tačku.

Poligoni čije su neke susedne stranice dovoljno blizu kreiraju **sliver**: to je uska poligonalna oblast koja sadrži po jedan ili nijedan piksel za neke scan linije (zbog pravila da se boje samo pikseli koji se nalaze u unutrašnjosti ili na levim i donjim ivicama). Ovaj problem može se rešavati tehnikom antialiasing.



Slika 2.15: Sliver problem

Kako što efikasnije odrediti presek scan linije sa stranicama poligona? Određivanje preseka scan linije sa svim stranicama poligona nije racionalno (jer najčešće samo nekoliko stranica seče scan liniju). Takođe, najčešće stranice koje seče  $n$ -ta scan linija seče i  $(n + 1)$ -a scan linija. Stoga se kao i kod originalnog midpoint algoritma, koristi inkrementalni pristup da bi se izračunali preseci jedne scan linije na osnovu prethodne scan linije, bez toga da analitički računamo presek scan linije sa svakom stranicom poligona. Pritom midpoint algoritam treba modifikovati tako da se uvek biraju tačke koje pripadaju unutrašnjosti poligona.

Na osnovu ideje midpoint algoritma, može se odrediti novi presek sa stranicom na osnovu starog preseka na sledeći način: prilikom prelaska na narednu scan liniju,  $y$  koordinata se uvećava za 1:

$$y_{i+1} = y_i + 1$$

S druge strane, ako je  $m$  koeficijent prave koja sadrži stranicu, važi:

$$y_{i+1} = mx_{i+1} + B$$

odnosno:

$$x_{i+1} = 1/m \cdot (y_{i+1} - B) = 1/m(y_i + 1 - B) = 1/m \cdot (y_i - B) + 1/m = x_i + 1/m$$

Navedenu vezu je potrebno modifikovati tako da koristi celobrojnu aritmetiku. U midpoint algoritmu za sken konverziju duži, izbegavali smo rad sa realnom aritmetikom računajući vrednost celobrojne promenljive odlučivanja i proverom njenog znaka da bismo izabrali piksel najbliži matematičkoj pravoj; ovde bismo voleli da koristimo celobrojnu aritmetiku da bismo uradili potrebno zaokruživanje za računanje najbližeg piksela u unutrašnjosti.

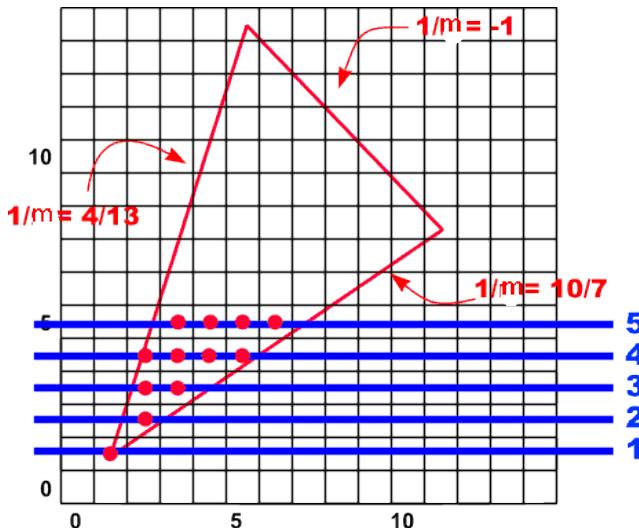
Razmotrimo stranicu sa koeficijentom nagiba većim od 1 koja je „leva“ stranica (parnost pre ovog preseka je 0); desne stranice i ostale vrednosti koeficijenata nagiba se računaju na sličan način, a vertikalne stranice predstavljaju specijalan slučaj.

$$x_{i+1} = x_i + 1/m$$

$$m = (y_{max} - y_{min}) / (x_{max} - x_{min})$$

$$x_{i+1} = x_i + (x_{max} - x_{min}) / (y_{max} - y_{min})$$

Razmotrimo posebno celi i razlomljeni deo vrednosti  $(x_{max} - x_{min}) / (y_{max} - y_{min})$ : kada razlomljeni deo pređe 1, onda inkrementiramo celi deo; kada je razlomljeni deo jednak 0, onda možemo da obojimo piksel  $(x, y)$ , a kada je razlomljeni deo različit od 0, onda je potrebno izvršiti zaokruživanje takvo da  $x$  pripada unutrašnjosti poligona. Kada razlomljeni deo pređe 1, onda  $x$  povećavamo za 1 (i pomeramo se jedan piksel udesno) i razlomljeni deo smanjujemo za 1. Opisani postupak može se svesti na celobrojni račun.



Za levu ivicu poligona:  $x_{min} = 1, x_{max} = 5, y_{min} = 1, y_{max} = 14$ , odnosno:  $(x_{max} - x_{min}) / (y_{max} - y_{min}) = 4/13$

$x_1 = 1$  (razlomljeni deo je jednak 0)

$x_2 = x_3 = x_4 = 2$  (razlomljeni deo je  $> 0$ , zaokružujemo tako da pripada unutrašnjosti poligona)

$x_5 = 3$  (razlomljeni deo je prešao 1, pa smo uvećali  $x$  za 1)

...

Praktična implementacija bi se sastojala iz narednih koraka. Tačke preseka se čuvaju u specijalnoj strukturi podataka koja čuva podatke o **aktivnim stranicama** –  $A$ . Pod aktivnim stranicama podrazumevamo one koje se sekutkuju sa tekućom scan linijom. Polazimo od inicijalne scan linije, koja ima  $y$  vrednost kao najmanja  $y$  vrednost svih temena poligona. Inicijalno je skup  $A$  prazan. Inicijalno skup  $S$  sadrži sve stranice poligona (sortirane po najmanjoj  $y$  vrednosti temena).

Sve dok skup aktivnih stranica nije prazan ili skup stranica  $S$  nije prazan, treba raditi sledeće:

1. Prebaci iz skupa  $S$  u skup  $A$  one stranice čija je minimalna  $y$  vrednost jednakata  $y$  vrednosti scan linije.
2. Sortiraj stranice u skupu  $A$  rastuće prema  $x$  koordinati tačke preseka sa scan linijom.

```

procedure LeftEdgeScan(xmin, ymin, xmax, ymax : integer);
begin
  x := xmin;
  y := ymin;
  brojilac := xmax-xmin;
  imenilac := ymax-ymin;
  inkrement := imenilac;
  for y := ymin to ymax do
    begin
      SetPixel(x,y);
      inkrement := inkrement+brojilac;
      if inkrement > imenilac then
        begin { prekoracenje, zaokruzi na sledeci }
          { piksel i smanji inkrement }
          x := x+1;
          inkrement := inkrement - imenilac;
        end;
    end;
end.

```

Slika 2.16: Algoritam LeftEdgeScan (za koeficijent prave veći od 1)

3. Oboji sve piksele na scan liniji između neparnih i parnih preseka.
4. Izbaci iz skupa  $A$  one stranice čija je maksimalna  $y$  vrednost jednaka  $y$  vrednosti scan linije.
5. Uvećaj  $y$  vrednost scan linije za 1.
6. Ažuriraj tačke preseka kao u algoritmu LeftEdgeScan

Ovaj algoritam koristi i povezanost ivica da bi izračunao  $x$  koordinate preseka, kao i povezanost scan linija (uz sortiranje) da bi odredio opsege piksela koje treba obojiti na jednoj scan liniji.

## 2.5 Kliping/seckanje linija (eng. clipping)

Zadatak:

- Za dati pravougaonik, nacrtati samo duži i delove duži koji mu pripadaju
- Ako su koordinate donjeg levog ugla pravougaonika  $(x_{min}, y_{min})$  a gornjeg desnog  $(x_{max}, y_{max})$ , onda treba da budu nacrtane samo tačke  $(x, y)$  za koje važi

$$x_{min} \leq x \leq x_{max}, \quad y_{min} \leq y \leq y_{max}$$

- Rezultat je uvek jedna duž

- Analiza krajnjih tački:
  - ako su obe krajnje tačke unutar pravougaonika nacrtaj celu duž
  - ako je jedna unutar a druga van onda mora da se vrši odsecanje
  - ako su obe tačke van pravougaonika onda se ništa ne zna
- Kliping krugova i elipsi može da dâ više lukova
- Kliping za krugove i elipse može da se svede na kliping (dovoljno malih) duži.

### 2.5.1 Naivno rešenje

- Za svaku duž ispitati da li njena temena pripadaju pravougaoniku, ako da, onda prihvati celu tu duž; ako ne, onda odrediti preseke sa svim pravama koje određuju stranice pravougaonika, ispitati raspored itd.
- Parametarski oblik za duž je pogodan za određivanje preseka duži:

$$x = x_0 + t(x_1 - x_0), \quad y = y_0 + t(y_1 - y_0)$$

gde je  $t \in [0, 1]$

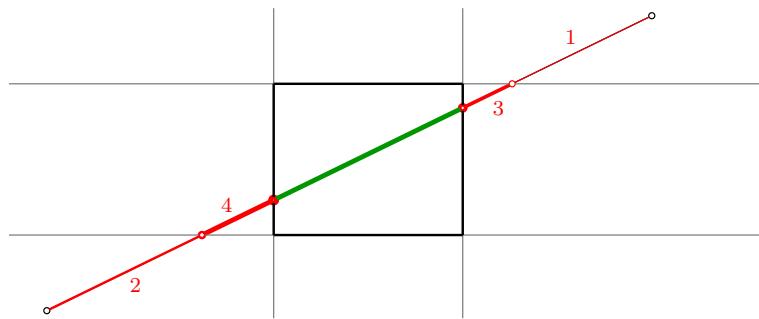
- Isti takav oblik po parametru  $s$  napravimo za svaku od stranica i duži se sekutako je presečna tačka za vrednosti  $t \in [0, 1], s \in [0, 1]$ .
- Ovaj pristup ipak zahteva mnogo izračunavanja i neefikasan je.

### 2.5.2 Cohen-Sutherland-ov algoritam

- Cohen-Sutherland-ov algoritam vrši inicijalne testove na datoj duži da bi utvrdio da li se mogu izbeći izračunavanja preseka
- Ako obe temena duži pripadaju pravougaoniku, onda trivijalno treba prihvati celu tu duž.
- Ako se duž ne može trivijalno prihvati, onda se vrše provere regionalne. Npr. ako su obe  $x$  koordinate temena duži manje od  $x_{min}$ , onda treba odbaciti celu tu duž (analogno za  $x_{max}, y_{min}, y_{max}$ ). Dakle, na osnovu dva jednostavna poređenja moguće je utvrditi da se duž može trivijalno odbaciti.
- Ako je jedno teme duži van pravougaonika, onda treba odbaciti deo duži do pravougaonika i nastaviti analogno.
- Ravan se deli na 9 regionala, svakom regionalu se dodeljuje 4-bitni kôd.
- Svaki bit u ovom kodu se postavlja na 1 (tačno) ili 0 (netačno), a 4 bita u kodu odgovaraju sledećim uslovima:
  - Prvi bit (najmanje težine): "iznad gornje stranice",  $y > y_{max}$
  - Drugi bit (najmanje težine): "ispod donje stranice",  $y < y_{min}$
  - Treći bit (najmanje težine): "desno od desne stranice",  $x > x_{max}$
  - Cetvrti bit (najmanje težine): "levo od leve stranice",  $x < x_{min}$

- Tačka se nalazi unutra samo ako sva četiri bita imaju vrednost 0

1001	1000	1010
0001	0000	0010
0101	0100	0110



- Za svaku duž računaju se kodovi obe krajnje tačke  $K_1$  i  $K_2$ : duži kod kojih je  $K_1 = 0$  i  $K_2 = 0$  se trivijalno prihvataju, duži koje se u celosti nalaze u poluravni van stranice se trivijalno odbacuju – to možemo utvrditi bitovskom konjunkcijom njihovih kodova ( $K_1 \& K_2 \neq 0$ ).
- Ako duž ne možemo trivijalno da prihvatimo ili odbacimo onda je de-limo na dve duži i onda trivijalno odbacujemo ili prihvatamo barem jednu od dve dobijene duži:
  - koristimo stranice pravougaonika za podelu duži
  - biramo redosled za proveru preseka sa stranicama:  
TOP - BOTTOM - RIGHT - LEFT
  - računamo koordinate presečne tačke (stranica ima ili x ili y koordinatu konstantnu pa je možemo zameniti u jednačini prave)
  - nastavljamo sa skraćenom duži

```
#define TOP 1
#define BOTTOM 2
#define RIGHT 4
#define LEFT 8

char CompOutCode(double x, double y,
    double xmin, double ymin, double xmax, double ymax) {
    char code = 0;
    if (y > ymax)
        code |= TOP;
    else if (y < ymin)
        code |= BOTTOM;
    if (x > xmax)
        code |= RIGHT;
    else if (x < xmin)
        code |= LEFT;
    return code;
}
```

```
void CohenSutherlandLineClipAndDraw(
    double x0, double y0, double x1, double y1,
    double xmin, double ymin, double xmax, double ymax) {
    bool accept = false, done = false;
    char pointCode0, pointCode1, pointOut;
    double x, y;

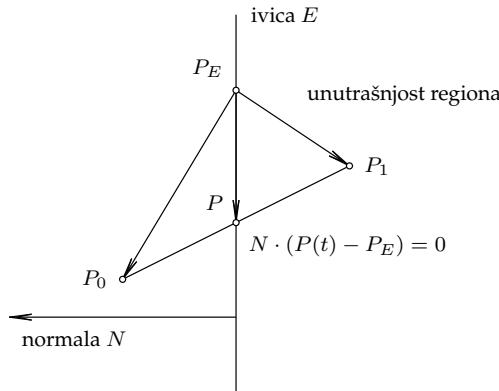
    pointCode0 = CompOutCode(x0, y0, xmin, ymin, xmax, ymax);
    pointCode1 = CompOutCode(x1, y1, xmin, ymin, xmax, ymax);
    do {
        if (pointCode0 == 0 && pointCode1 == 0) { /* prihvati duz i izadji */
            accept = true;
            done = true;
        }
        else if ((pointCode0 & pointCode1) != 0)
            done = true; /* odbaci duz i izadji */
        else { /* izabereti tacku koja je van pravougaonika */
            if (pointCode0 != 0) then
                pointOut = pointCode0;
            else
                pointOut = pointCode1;
            if (pointOut & TOP) {
                x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0); /* prava y = ymax */
                y = ymax;
            }
            else if (pointOut & BOTTOM) {
                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0); /* prava y = ymin */
            }
        }
    } while (!done);
}
```

```
        y = ymin;
    }
    else if (pointOut & RIGHT) {
        y = y0+(y1-y0)*(xmax-x0)/(x1-x0); /* prava x = xmax */
        x = xmax;
    }
    else (pointOut & LEFT) { /* prava x = xmin */
        y = y0+(y1-y0)*(xmin-x0)/(x1-x0);
        x = xmin;
    }
    if (pointOut==pointCode0) {
        x0 = x; y0 = y; pointCode0 = CompOutCode(x0,y0);
    }
    else {
        x1 = x; y1 = y; pointCode1 = CompOutCode(x1,y1);
    }
}
while(!done);
if (accept)
    MidpointLine(x0,y0,x1,y1)
}
```

### 2.5.3 Cyrus-Beckov algoritam

- Cohen-Sutherland-ov algoritam je verovatno još uvek najkorišćeniji klipping algoritam. Cyrus i Beck su 1978. objavili suštinski drugačiji klipping algoritam koji je u opštem slučaju efikasniji.
- Cyrus-Beckov algoritam primenljiv je na bilo kakav konveksni klipping region. Liang i Barsky su kasnije nezavisno razvili efikasniji parametarski klipping algoritam. Liang-Barsky varijanta optimizovana je za specijalni slučaj — pravougaonik sa horizontalnim/vertikalnim stranicama.
- Parametarski oblik duži  $P_0P_1$ :

$$P(t) = P_0 + (P_1 - P_0)t$$



- Odrediti gde duž seče prave kojima pripadaju stranice kliping regiona: rešavamo po jednom po  $t$  za svaku od stranica, utvrđujemo koji od ovih preseka pripada pravougaoniku
- Za tačku  $P_E$  možemo izabrati proizvoljnu tačku sa ivice  $E$
- $N \cdot (P_1 - P_E) < 0$  za tačku  $P_1$  koja je unutar poligona;
- $N \cdot (P_0 - P_E) > 0$  za tačku  $P_0$  koja je izvan poligona;
- $N \cdot (P(t) - P_E) = 0$  za tačku  $P(t)$  koja je na stranici poligona.  
gde je  $N$  normala stranice poligona usmerena ka spoljašnjosti poligona.
- Neka je  $D = \overrightarrow{P_0P_1}$
- uslov iz kojeg se može izračunati tačka preseka:

$$N \cdot (P(t) - P_E) = 0$$

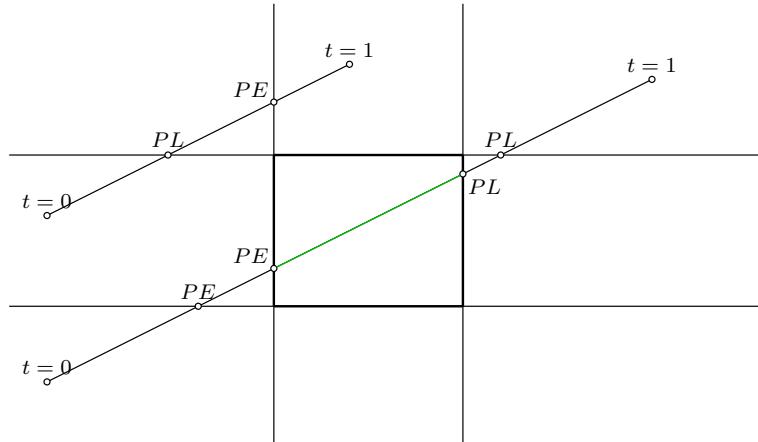
$$N \cdot (P_0 + (P_1 - P_0)t - P_E) = 0$$

$$N \cdot (P_0 - P_E + (P_1 - P_0)t) = 0$$

$$t = \frac{N \cdot (P_0 - P_E)}{-N \cdot (P_1 - P_0)}$$

$$t = \frac{N \cdot (P_0 - P_E)}{-N \cdot D}$$

- Proverava se da li je imenilac različit od nule, tj da li važi  $D \neq 0$  (tj. tačke  $P_0$  i  $P_1$  su različite),  $N \neq 0$  (što ne važi samo u slučaju greške),  $-N \cdot D \neq 0$  (stranica i duž koja se secka nisu paralelne).



- Eliminišu se vrednosti za  $t$  koje su van intervala  $[0,1]$ . Koje od preostalih vrednosti proizvode unutrašnje preseke? Ne možemo samo uzeti vrednosti koje se nalaze najdublje unutra
- Potrebno je izvršiti klasifikaciju na "potencijalno ulazne" i "potencijalno izlazne" tačke.
- $N \cdot D < 0 \Rightarrow PE$  (ugao između vektora  $N$  i  $D$  je  $> 90^\circ$ , pa je  $\cos$  manji od 0) (*PE* = potentially entering)
- $N \cdot D > 0 \Rightarrow PL$  (ugao između vektora  $N$  i  $D$  je ugao  $< 90^\circ$ , pa je  $\cos$  veći od 0) (*PL* = potentially leaving)
- Sve preseke je potrebno sortirati, izabrati maksimum od ulaznih i  $t = 0$  – nju proglašavamo vrednošću  $t_E$  i minimum od izlaznih i  $t = 1$  - nju proglašavamo vrednošću  $t_L$ . Ako je  $t_L < t_E$  ne postoji presek
- PE* u izračunavanjima može biti bilo koja tačka stranice kliping regionala, npr. njeno teme.
- Jednostavnosti radi, prepostavlja se da duž  $P_0P_1$  nije paralelna stranicama poligona. Ako je paralelna, vrši se posebna analiza.

```

izracunaj Ni i odredi P_Ei za svaku stranicu;
za svaku duž koju treba iseckati uradi sledeće:
if (P1==P0)
    seckaj kao tacku
else {
    t_E=0; t_L=1;
    za svakog kandidata za presek uradi sledeće:
    if N_i * D <> 0 then {
        /* ignorisi paralelne linije */
    }
}

```

```

    izracunaj t;
    upotrebi znak N_i * D da kategorizujes kao PE ili PL;
    if (PE) t_E = max(t_E,t);
    if (PL) t_L = min(t_L,t);
}
if (t_E>t_L)
    return nil;
else
    return P(t_E) i P(t_L);
}

```

- Liang-Barsky varijanta:

Kliping ivica	normala $N$	$P_E$	$P_0 - P_E$	$t = \frac{N \cdot (P_0 - P_E)}{-N \cdot D}$
left: $x = x_{min}$	(-1, 0)	$(x_{min}, y_{min})$	$(x_0 - x_{min}, y_0 - y_{min})$	$\frac{-(x_0 - x_{min})}{x_1 - x_0}$
right: $x = x_{max}$	(1, 0)	$(x_{max}, y_{max})$	$(x_0 - x_{max}, y_0 - y_{max})$	$\frac{-(x_0 - x_{max})}{x_1 - x_0}$
bottom: $y = y_{min}$	(0, -1)	$(x_{min}, y_{min})$	$(x_0 - x_{min}, y_0 - y_{min})$	$\frac{-(y_0 - y_{min})}{y_1 - y_0}$
top: $y = y_{max}$	(0, 1)	$(x_{max}, y_{max})$	$(x_0 - x_{max}, y_0 - y_{max})$	$\frac{-(y_0 - y_{max})}{y_1 - y_0}$

- Posmatramo izraz za  $t$ : brojilac je usmereno rastojanje do stranice, a imenilac horizontalna ili vertikalna projekcija duži

## 2.6 Pitanga

- 2.1 Šta je rasterizacija?
- 2.2 Koji su zahtevi kod crtanja linije?
- 2.3 Ako je koeficijent pravca prave (a)  $m = 1/2$ , (b)  $m = 3$  da li crtamo po jedan piksel u svakom redu ili koloni?
- 2.4 Kako ocenjujemo kvalitet dobijene rasterske slike?
- 2.5 Kako radi algoritam grube sile za crtanje duži?
- 2.6 Koja je ideja inkrementalnih algoritama?
- 2.7 Kako funkcioniše midpoint algoritam za crtanje duži?
- 2.8 Ukoliko je  $1 < m < \infty$  između kojih tacaka vršimo odabir?
- 2.9 Čemu nam u midpoint algoritmu služi tačka M između tačaka N i NE?
- 2.10 Šta je promenljiva odlučivanja?
- 2.11 Izračunaj prvu vrednost promenljive odlučivanja za duž koja povezuje tačke sa koordinatama (1,2) i (7,4).
- 2.12 Koja je suštinska razlika u vrednosti promenljive odlučivanja kod crtanja kruga u odnosu na crtanje duži?

- 2.13 Šta se menja ukoliko krug nije u koordinatnom početku?
- 2.14 Koja je vremenska složenost midpoint algoritma za crtanje: (a) duži, (b) kruga?
- 2.15 Kada za region kažemo da je 4-povezan, a kada da je 8-povezan?
- 2.16 Koji algoritam zovemo FloodFill, a koji BoundaryFill?
- 2.17 Kod scan popunjavanja poligona koje se presečne tačke scan linije sa stranicama poligona broje, a koje ne?
- 2.18 Koje se tačke boje kod horizontalnih stranica poligona?
- 2.19 Šta je sliver?
- 2.20 Kako se računa presek  $(i + 1)$ -ve scan linije na osnovu na  $i$ -te scan linije?
- 2.21 Koja je vremenska složenost algoritma LeftEdgeScan?
- 2.22 Koje se tačke preseka dobijaju algoritmom LeftEdgeScan za duž sa krajnjim tačkama  $(2, 3)$  i  $(5, 8)$ ?
- 2.23 Kod Cohen-Sutherlandovog algoritma šta važi za duž koja se secka ako je  $K_1 = 0$  i  $K_2 = 0$ ? Šta važi ako je  $K_1 \& K_2 <> 0$ ?
- 2.24 Koja je vremenska složenost Cohen-Sutherland algoritma?
- 2.25 Koji uslov važi za presečnu tačku duži  $P_0P_1$  i normalu stranice?
- 2.26 Šta treba da važi da bi tačka bila okarakterisana kao potencijalno ulazna / izlazna?
- 2.27 Šta znači kada je  $t_L < t_E$ ?
- 2.28 Koja je vremenska složenost Cyrus-Beckovog algoritma?
- 2.29 Na kakav se region može primeniti Liang-Barsky varijanta algoritma za kliping?

## Glava 3

# Geometrijske osnove

### 3.1 Jednačine prave i ravni

Jednačina prave u ravni je:

$$ax + by + c = 0$$

Jednačina ravni je:

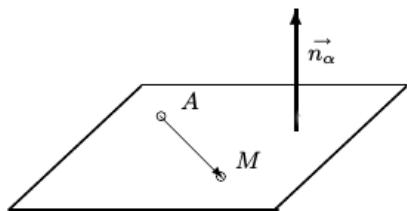
$$Ax + By + Cz + D = 0$$

Ovo dobijamo na osnovu narednog razmatranja: ravan je određena tačkom koja joj pripada i njenim vektorom normale. Neka ravn  $\alpha$  pripada tačka  $A : (x_0, y_0, z_0)$ , neka je njen vektor normale  $\vec{n}_\alpha : [A, B, C]$  i neka je  $M : (x, y, z)$  proizvoljna tačka ravn  $\alpha$ . Tada se jednačina ravn  $\alpha$  dobija iz činjenice da su vektori  $\overrightarrow{AM}$  i  $\vec{n}_\alpha$  upravni, tj. da je njihov skalarni proizvod jednak nuli:

$$0 = \vec{n}_\alpha \cdot \overrightarrow{AM} = [A, B, C] \cdot [x - x_0, y - y_0, z - z_0] = A(x - x_0) + B(y - y_0) + C(z - z_0) = Ax + By + Cz + D$$

Dakle, normala ravn koja je data jednačinom  $Ax + By + Cz + D = 0$  je  $[A, B, C]$ .

Ako tri nekolinearne tačke  $P_1$ ,  $P_2$  i  $P_3$  pripadaju ravn, onda je vektor  $\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$  kolinearan vektoru normale ravn  $[A, B, C]$ . Ako su tačke  $P_1$ ,  $P_2$ ,  $P_3$  kolinearne, onda ovim tačkama nije određena ravan i vektorski proizvod  $\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_3}$  jednak je 0.



Rastojanje tačke  $(x_0, y_0, z_0)$  od ravni koja je data jednačinom  $Ax + By + Cz + D = 0$  određeno je jednakošću:

$$d = \frac{Ax_0 + By_0 + Cz_0 + D}{\sqrt{A^2 + B^2 + C^2}}$$

Za sve tačke sa jedne strane ravni, vrednosti  $Ax + By + Cz + D$  imaju isti znak. Dakle, da bi se odredilo sa koje strane ravni je tačka sa koordinatama  $(x_0, y_0, z_0)$  dovoljno je izračunati vrednost  $Ax_0 + By_0 + Cz_0 + D$ .

### 3.2 Vektori

- Vektor je  $n$ -torka vrednosti. U daljem tekstu smatraćemo da su te vrednosti iz skupa realnih brojeva.
- Vektori se sabiraju tako što se sabiraju vrednosti pojedinačnih elemenata:

$$\vec{v}_1 + \vec{v}_2 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 + x_2 \\ y_1 + y_2 \end{bmatrix}$$

- Vektori se množe skalarom tako što se pojedinačni elementi množe skalarom:

$$a\vec{v} = a \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax \\ ay \end{bmatrix}$$

- Skalarni proizvod dva vektora se računa na sledeći način:

$$\vec{v}_1 \cdot \vec{v}_2 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = x_1x_2 + y_1y_2$$

odnosno ako su vektori dimenzije 3 na sledeći način:

$$\vec{v}_1 \cdot \vec{v}_2 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \cdot \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = x_1x_2 + y_1y_2 + z_1z_2$$

- Vektori dimenzije 2 mogu da opisuju tačke u Dekartovoj ravni, a vektori dimenzije 3 mogu da opisuju tačke u Dekartovom prostoru.
- Dužina vektora  $\vec{v}$  (označava se sa  $|\vec{v}|$ ) jednaka je  $\sqrt{\vec{v} \cdot \vec{v}}$ .
- Jedinični vektor istog smera i pravca kao vektor  $\vec{v}$  određen je sa  $\vec{v}/|\vec{v}|$ .
- Ugao  $\phi$  između dva vektora  $\vec{v}$  i  $\vec{u}$  može se izračunati na osnovu sledeće veze:  $\vec{v} \cdot \vec{u} = |\vec{v}||\vec{u}| \cos \phi$ .

### 3.3 Linearne transformacije

Za funkciju  $f$  kažemo da je **linearna funkcija** ako važi:

1.  $f(v + w) = f(v) + f(w)$  za sve vrednosti  $v$  i  $w$  iz domena funkcije  $f$
2.  $f(cv) = cf(v)$  za sve skalare  $c$  i vrednosti  $v$  iz domena funkcije  $f$

Grafički posmatrano to su transformacije tačaka oko koordinatnog početka (pri njima koordinatni početak ostaje invarijantan). Ove transformacije uključuju skaliranje i rotaciju, ali ne i translaciju jer se njome koordinatni početak pomera.

Pitanje: kako na osnovu definicije pokazati da je koordinatni početak invarijantan kod linearnih transformacija?

Linearne transformacije se mogu predstaviti invertibilnim matricama. Krenimo od 2D transformacija – one se mogu predstaviti 2D matricama

**Baza vektorskog prostora** je skup vektora  $\{V_1, V_2\}$  za koje važi:

1. vektori iz datog skupa su linearne nezavisni
2. svaki vektor iz vektorskog prostora se može izraziti kao linearna kombinacija vektora iz ovog skupa:  $V = c_1V_1 + c_2V_2$

Vektore iz ovog skupa nazivamo baznim vektorima.

Neka je  $e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$  i  $e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$  standardna baza vektorskog prostora i neka je  $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$  matrica transformacije  $T$ . Izračunajmo u šta se slikaju bazni vektori:

$$T\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix}$$

$$T\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$$

Primetimo da su kolone matrice transformacije upravo jednake matrici  $T$  primenjenoj na vektore  $e_1$  i  $e_2$ .

$$T(e_1) = \begin{bmatrix} a \\ c \end{bmatrix}, T(e_2) = \begin{bmatrix} b \\ d \end{bmatrix}$$

Iz ovog dobijamo strategiju za izvođenje matrica transformacija: izvodimo kolonu po kolonu matrice transformacije razmatranjem kako željena transformacija deluje na bazne vektore standardne baze.

## 3.4 2D transformacije

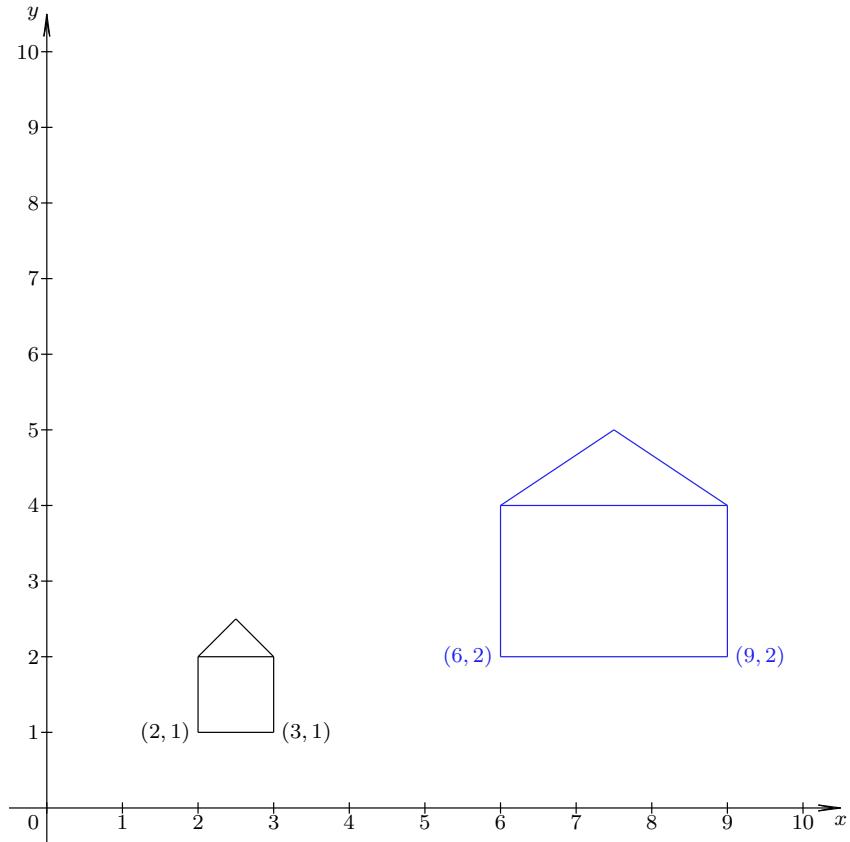
### 3.4.1 Osnovne transformacije

#### Skaliranje

- Tačke se mogu skalirati (istezati) duž  $x$  i duž  $y$  ose. Skaliranje ne mora nužno da bude uniformno. Npr, možemo da skaliramo po  $x$  osi za faktor 3, a po  $y$  osi za faktor 2 ( $s_x = 3, s_y = 2$ )

$$v = \begin{bmatrix} x \\ y \end{bmatrix} - \text{originalno teme} \rightarrow v' = \begin{bmatrix} x' \\ y' \end{bmatrix} - \text{novo teme},$$

$$v' = Sv.$$



- Izvedimo matricu skaliranja  $S$  razmatranjem na koji način vektori  $e_1$  i  $e_2$  treba da budu transformisani:

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow s_x \cdot e_1 = \begin{bmatrix} s_x \\ 0 \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow s_y \cdot e_2 = \begin{bmatrix} 0 \\ s_y \end{bmatrix}$$

Na ovaj način izveli smo kolone matrice transformacije, te je matrica skaliranja  $S$  jednaka:

$$S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

- Primetimo da je matrica  $S$  dijagonalna matrica: pri ovoj transformaciji svaku od koordinata množimo odgovarajućim faktorom:

$$x' = s_x \cdot x$$

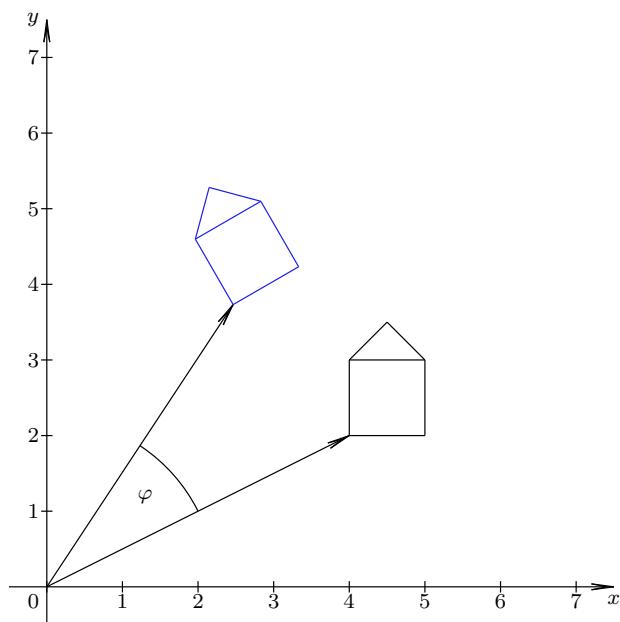
$$y' = s_y \cdot y$$

- Neka svojstva skaliranja:

- ne čuvaju se uglovi između pravih u ravni (osim kada je skaliranje uniformno, tj  $s_x = s_y$ )
- ako objekat ne počinje u koordinatnom početku skaliranje će ga približiti ili udaljiti od koordinatnog početka (što često nije ono što želimo)

### Rotacija

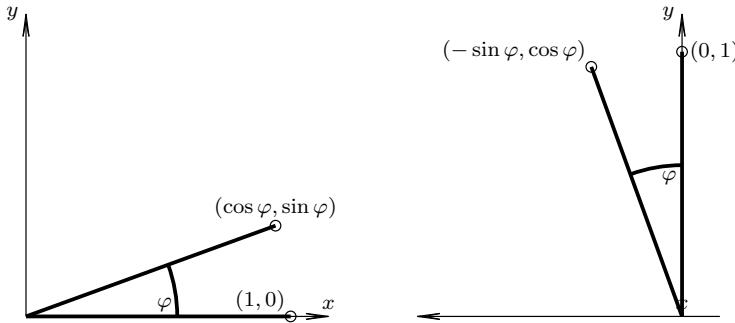
- Razmatramo rotaciju oko koordinatnog početka za ugao  $\varphi$ , pri čemu su uglovi pozitivno orijentisani.



- Izvedimo matricu rotacije  $R_\varphi$  razmatranjem na koji način ona treba da utiče na vektore  $e_1$  i  $e_2$

$$e_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \cos \varphi \\ \sin \varphi \end{bmatrix}$$

$$e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -\sin \varphi \\ \cos \varphi \end{bmatrix}$$



- Stoga je  $R_\varphi = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix}$

tj:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \varphi & -\sin \varphi \\ \sin \varphi & \cos \varphi \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

tj.

$$v' = R_\varphi \cdot v$$

$$x' = x \cdot \cos \varphi - y \cdot \sin \varphi$$

$$y' = x \cdot \sin \varphi + y \cdot \cos \varphi$$

- Neka svojstva rotacije:

- čuva dužine objekata
- čuva uglove među parovima objekata

### Translacija

- Translacija nije linearne transformacija (koordinatni početak se ne čuva), te se ne može predstaviti  $2 \times 2$  invertibilnom matricom
- Translaciju je moguće predstaviti korišćenjem sabiranja:

$$x' = x + t_x \quad y' = y + t_y$$

odnosno u matričnoj formi:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

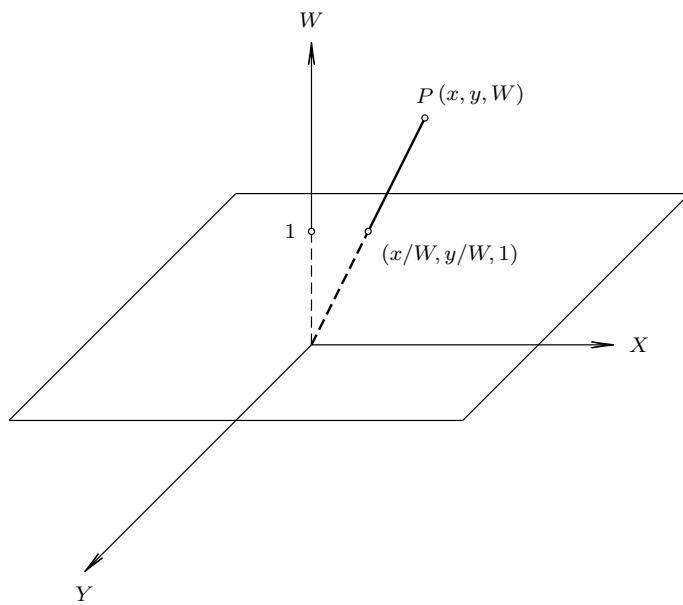
tj.

$$v' = v + T$$

- Međutim, korišćenje sabiranja vektora nije konzistentno sa razmatranim metodom matrica transformacija. Poželjno je sve transformacije predstaviti na isti način da bismo kombinaciju transformacija mogli da izrazimo korišćenjem kompozicije njihovih matrica. Iz ovog razloga prelazimo na homogene koordinate.

### 3.4.2 Homogene koordinate

- Bilo bi dobro da translacija ( $v' = v + T$ ), skaliranje ( $v' = S \cdot v$ ) i rotacija ( $v' = R \cdot v$ ) imaju istu formu.
- Koristeći homogene koordinate sve ove tri transformacije imaju formu množenja matrica.
- Homogene koordinate se koriste u geometriji od pedesetih godina, a u računarskoj grafici od šezdesetih godina dvadesetog veka.
- Uz pomoć homogenih koordinata 2D tačka se predstavlja trojkom vrednosti  $(x, y, W)$ .
- Dve trojke  $(x, y, W)$  i  $(x', y', W')$  predstavljaju istu tačku ako postoji broj  $t$  takav da je  $x = tx'$ ,  $y = ty'$  i  $W = tW'$ .
- Bar jedna od vrednosti  $x, y, W$  nije 0.
- Ako je  $W \neq 0$ , onda tačku možemo predstaviti u obliku  $(x/W, y/W, 1)$  i  $(x/W, y/W)$  zovemo Dekartovim koordinatama homogene tačke.
- Tačka  $(x, y, 0)$  je beskonačno daleka tačka u pravcu  $(x, y)$ .
- Homogenim 2D tačkama odgovaraju trojke, a te trojke odgovaraju tačkama u Dekartovom prostoru.
- Skupu svih trojki kojima odgovara jedna homogena tačka odgovara prava Dekartovog prostora. Zaista, sve tačke oblika  $(tx, ty, tW)$  ( $t \neq 0$ ) pripadaju jednoj pravoj Dekartovog prostora.
- Ako **homogenizujemo** (podelimo sa  $W$ ) jednu tačku, onda dobijamo njenu reprezentaciju  $(x/W, y/W, 1)$  koja u dekartovskom smislu pripada ravni  $W = 1$ .



### 3.4.3 2D transformacije u homogenim koordinatama

- Izrazimo sada prethodno pomenute transformacije u terminima homogenih koordinata. Kada budemo dobili homogene koordinate slike, podelićemo  $x$  i  $y$  koordinatu sa vrednošću  $W$  koordinate i dobiti dekartovske koordinate.

- Tačka  $v = \begin{bmatrix} x \\ y \end{bmatrix}$  se sada prikazuje kao:

$$v = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

i želimo da je preslikamo u tačku:

$$v' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

- Kako izmeniti postojeće matrice transformacija? Za linearne transformacije (skaliranje, rotaciju, ...) potrebno je "ugraditi" postojeću matricu u gornji levi ugao  $3 \times 3$  jedinične matrice:

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Što se translacije tiče, matrica translacije se može predstaviti "ugrađivanjem" vektora translacije u treći kolonu jedinične matrice:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Možemo izvršiti proveru:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$$

- Kao što možemo da vidimo, koordinate tačke su pravilno translirane i pritom smo, korišćenjem homogenih koordinata, translaciju izrazili u vidu množenja matrica.
- Primeri matrica transformacija u terminima homogenih koordinata:

- skalirati za faktor 5 u smeru  $x$  ose i za faktor 7 u smeru  $y$  ose

$$S = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- rotirati za 78 stepeni

$$R = \begin{bmatrix} \cos(78) & -\sin(78) & 0 \\ \sin(78) & \cos(78) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- translirati za -5 u smeru  $x$  ose i 7 u smeru  $y$  ose

$$T = \begin{bmatrix} 1 & 0 & -5 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{bmatrix}$$

#### 3.4.4 Inverzne transformacije

- Kada hoćemo da poništimo transformaciju, potreban nam je inverz matrice transformacije. Zahvaljujući homogenim koordinatama, sve matrice su invertibilne.
- Lako se dobija:

- Inverzna transformacija *skaliranju*  $S$

$$S = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

je skaliranje kojem odgovara matrica:

$$S^{-1} = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Inverzna transformacija *rotaciji* za ugao  $\varphi$  je rotacija za ugao  $-\varphi$ . Ovo je tačno zato što je inverz matrice rotacije, njen transponat (lako se može proveriti njihovim množenjem da se dobija jedinična matrica)
- Inverzna transformacija *translaciji*  $T$

$$T = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

je translacija kojoj odgovara matrica:

$$T^{-1} = \begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

#### 3.4.5 Kompozicija transformacija

- Kada imamo veliki broj transformacija na raspolaganju interesuje nas kako ih iskombinovati. Na objekat na sceni primenjuje se niz različitih transformacija i interesuje nas kako ih predstaviti? Transformacija je funkcija i važi da je:  $(f \cdot g)(i) = f(g(i))$ . Ako funkcije  $f$  i  $g$  posmatramo kao matrice  $M_1$  i  $M_2$ , a ulaz kao vektor  $v$ , kompozicija transformacija biće jednaka:  $M_1 M_2 v$ .

- Da bismo konstruisali složenije transformacije, formiramo kompoziciju matrica transformacija. Na primer, ukoliko želimo da napravimo transformaciju kojom se tačka skalira, nakon toga rotira, a zatim translira do bili bismo narednu matricu transformacije:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Napomena 1: obratiti pažnju da se matrice primjenjuju *zdesna ulevo*.
- Napomena 2: bitan je poredak matrica jer množenje matrica u opštem slučaju nije komutativno, te u opštem slučaju transformacije ne komutiraju. U nekim specijalnim slučajevima one komutiraju; na primer, translacije međusobno komutiraju; rotacije oko iste tačke međusobno komutiraju i sl.
- Rotacija oko tačke  $P(x_1, y_1)$  (koja nije koordinatni početak) za ugao  $\varphi$  se može realizovati na sledeći način:
  - transliraj tačku  $P$  u koordinatni početak
  - rotiraj za ugao  $\varphi$
  - transliraj koordinatni početak u tačku  $P$

$$\begin{aligned} T(x_1, y_1) \cdot R(\varphi) \cdot T(-x_1, -y_1) &= \\ = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 \\ \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} &= \\ = \begin{bmatrix} \cos \varphi & -\sin \varphi & x_1(1 - \cos \varphi) + y_1 \sin \varphi \\ \sin \varphi & \cos \varphi & y_1(1 - \cos \varphi) - x_1 \sin \varphi \\ 0 & 0 & 1 \end{bmatrix} & \end{aligned}$$

- Analogno bi se razmatralo skaliranje u odnosu na neku tačku (koja nije koordinatni početak).
- Primer:* rotirati objekat za 90 stepeni, a zatim ga skalirati 4x – sve to oko centra objekta, a ne oko koordinatnog početka.  
Bitan koncept je *pojednostaviti problem*: translirati objekat u koordinatni početak, skalirati, rotirati i translirati nazad na sledeći način:  $T^{-1}SRT$ . Ovu transformaciju treba primeniti na sva temena objekta koji treba transformisati.
- Dodatak:* transformacija **smicanja** ili **iskošenja** (engl. shear): treba iskosititi objekat u smeru neke od koordinatnih osa. Na taj način se  $x$  koordinata iskošuje nadesno, a  $y$  koordinata ostaje ista ili obrnuta. Na ovaj način, recimo, kvadrat postaje paralelogram.

- Jednačine transformacija bile bi:

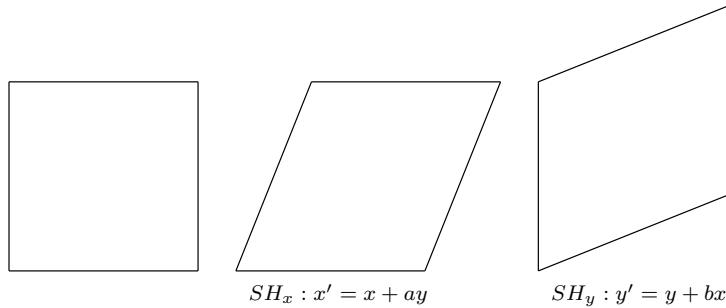
$$x' = x + a \cdot y$$

$$y' = y$$

odnosno:

$$x' = x$$

$$y' = y + b \cdot x$$



- Odgovarajuće matrice transformacija su:

$$SH_x = \begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix}$$

$$SH_y = \begin{bmatrix} 1 & 0 \\ b & 1 \end{bmatrix}$$

tj. u homogenim koordinatama:

$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Vratimo se inverznim transformacijama: šta je inverz kompozicije transformacija?
- Inverz kompozicije transformacija je kompozicija inverza pojedinačnih transformacija u obrnutom redosledu:

$$(M_1 M_2 \dots M_n)^{-1} = M_n^{-1} \dots M_2^{-1} M_1^{-1}$$

Zašto?

**Translacija:** Kompozicija translacija je translacija:

$$\begin{bmatrix} 1 & 0 & t'_x \\ 0 & 1 & t'_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t''_x \\ 0 & 1 & t''_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t'_x + t''_x \\ 0 & 1 & t'_y + t''_y \\ 0 & 0 & 1 \end{bmatrix}$$

**Skaliranje:** Kompozicija skaliranja je skaliranje:

$$\begin{bmatrix} s'_x & 0 & 0 \\ 0 & s'_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s''_x & 0 & 0 \\ 0 & s''_y & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s'_x \cdot s''_x & 0 & 0 \\ 0 & s'_y \cdot s''_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

**Rotacija:** Kompozicija rotacija je rotacija:

$$\begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \beta & -\sin \beta & 0 \\ \sin \beta & \cos \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos \alpha \cos \beta - \sin \alpha \sin \beta & -(\cos \alpha \sin \beta + \sin \alpha \cos \beta) & 0 \\ \sin \alpha \cos \beta + \cos \alpha \sin \beta & \cos \alpha \cos \beta - \sin \alpha \sin \beta & 0 \\ 0 & 0 & 1 \end{bmatrix} =$$

$$\begin{bmatrix} \cos(\alpha + \beta) & -\sin(\alpha + \beta) & 0 \\ \sin(\alpha + \beta) & \cos(\alpha + \beta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

#### 3.4.6 Efikasnost izračunavanja

- Umesto da za svaku tačku primenjujemo množenje jednom matricom ili nizom matrica, obično najpre izračunamo matricu kompozicije i pokušavamo da iskoristimo njenu specifičnu formu.
- Često su transformacione matrice oblika:

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

pa je umesto množenja matrica dovoljno koristiti jednakosti:

$$x' = x \cdot a + y \cdot b + t_x$$

$$y' = x \cdot c + y \cdot d + t_y$$

- Za transformacije složenih objekata u realnom vremenu potrebno je voditi računa o svakom množenju. Na primer, u jednakostima za rotaciju:

$$x' = x \cdot \cos \varphi - y \cdot \sin \varphi$$

$$y' = x \cdot \sin \varphi + y \cdot \cos \varphi$$

$\cos \varphi$  se može aproksimirati vrednošću 1 ako je ugao  $\varphi$  mali i na taj način date jednakosti postaju:

$$x' = x - y \cdot \sin \varphi$$

$$y' = x \sin \varphi + y$$

(vrednost  $\sin \varphi$  se, naravno, računa samo jednom a zatim koristi kao konstanta).

- Da bi se smanjilo nagomilavanje greške, bolje je u drugoj jednakosti koristiti  $x'$  umesto  $x$ :

$$x' = x - y \cdot \sin \varphi$$

$$y' = x' \sin \varphi + y = (x - y \cdot \sin \varphi) \sin \varphi + y = x \sin \varphi + y(1 - \sin^2 \varphi)$$

jer  $1 - \sin^2 \varphi$  bolje aproksimira  $\cos \varphi$  nego 1.

#### 3.4.7 Izometrijske i affine transformacije

- Matrica je **ortogonalna** ako vektori koji čine njene *kolone* predstavljaju ortonormiranu bazu tj. ako je intenzitet svake kolone jednak 1 i ako je skalarni proizvod svake dve kolone jednak 0. Analogno važi i za *vrste* ortogonalne matrice.
- Svakoj **izometrijskoj transformaciji** odgovara transformaciona matrica oblika:

$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

gde je njena gornja leva podmatrica  $2 \times 2$  ortogonalna, tj. važi:

- $a^2 + c^2 = 1$
- $b^2 + d^2 = 1$
- $ab + cd = 0$

- Važi i obratno, ovakvim matricama odgovaraju izometrijske transformacije.
- Izometrijske transformacije čuvaju uglove i dužine (ponekad se zovu i *rigid-body* transformacije).
- Rotacijama i translacijama odgovaraju ovakve matrice. Bilo kom nizu rotacija/translacija odgovara ovakva matrica.
- **Afine transformacije** čuvaju kolinearnost, odnose rastojanja između kolinearnih tačaka i paralelnost, ali ne čuvaju nužno uglove i dužine.
- Translacija, rotacija, skaliranje i smicanje su affine transformacije i bilo koji niz rotacija/translacija/skaliranja/smicanja je affina transformacija.
- Rotacija, skaliranje i smicanje su linearne transformacije (i one se mogu opisati  $2 \times 2$  matricama).
- S obzirom na to da se matrica affinih transformacija može zapisati kao:

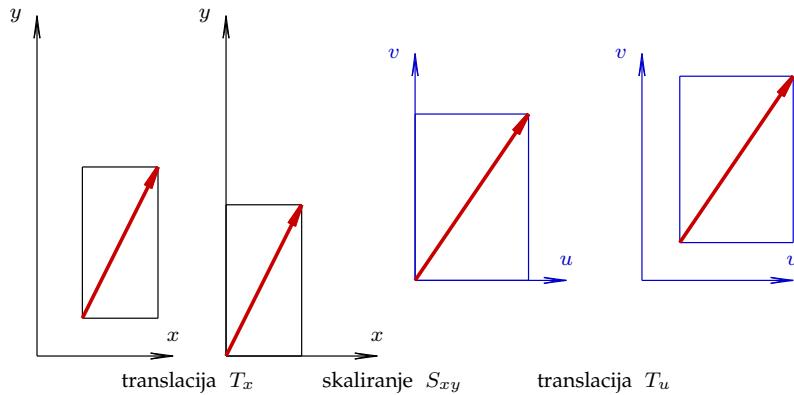
$$\begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} T^* & t^* \\ \mathbf{0} & 1 \end{bmatrix}$$

važi da se svaka afina transformacija može predstaviti kao kompozicija jedne linearne transformacije i jedne translacije:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = T^* \begin{bmatrix} x \\ y \end{bmatrix} + t^*$$

### 3.4.8 Preslikavanje slike u prozor

- Preslikavanje figure iz jednog koordinatnog sistema u drugi je standardni problem u računarskoj grafici. Javlja se recimo kod preslikavanja slike iz nekog koordinatnog sistema u prozor ekrana.
- Prepostavimo da treba preslikati sadržaj pravougaonika sa levim donjim temenom  $(x_{min}, y_{min})$  i gornjim desnim temenom  $(x_{max}, y_{max})$  u koordinatnom sistemu  $(x, y)$  (pri čemu su stranice pravougaonika paralelne osama) u pravougaonik sa levim donjim temenom  $(u_{min}, v_{min})$  i gornjim desnim temenom  $(u_{max}, v_{max})$  u koordinatnom sistemu  $(u, v)$  (pri čemu su stranice pravougaonika paralelne osama).



- Svaku tačku pravougaonika treba:
  - preslikati translacijom koja tačku  $(x_{min}, y_{min})$  preslikava u koordinatni početak:

$$T_x = \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

- preslikati skaliranjem koje tačku  $(x_{max}, y_{max})$  preslikava u tačku  $(u_{max}, v_{max})$ ; primetimo pritom da su dužine stranica pravougaonika  $(x_{max} - x_{min}, y_{max} - y_{min})$  i  $(u_{max} - u_{min}, v_{max} - v_{min})$ ;

$$S_{xu} = \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & 0 \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- preslikati translacijom koja koordinatni početak preslikava u tačku  $(u_{min}, v_{min})$ :

$$T_u = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

- Kompozicija navedenih transformacija opisana je matricom koja je dobijena množenjem odgovarajućih matrica:

$$M_{xu} = T_u S_{xu} T_x = \begin{bmatrix} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} & 0 & -x_{min} \frac{u_{max}-u_{min}}{x_{max}-x_{min}} + u_{min} \\ 0 & \frac{v_{max}-v_{min}}{y_{max}-y_{min}} & -y_{min} \frac{v_{max}-v_{min}}{y_{max}-y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

### 3.5 3D transformacije

- Homogene koordinate za 3D prostor definišu se analogno homogenim 2D koordinatama:
  - tačka  $(x, y, z)$  3D prostora predstavlja se četvorkom  $(x, y, z, W)$ .
  - dve četvorke predstavljaju istu tačku, ako i samo ako je jedna četvorka umnožak druge
  - četvorka  $(0, 0, 0, 0)$  nije dozvoljena
  - standardna reprezentacija za tačku  $(x, y, z, W)$ , pri čemu je  $W \neq 0$ , je  $(x/W, y/W, z/W, 1)$ , a postupak svođenja na standardnu reprezentaciju zovemo *homogenizacija*
  - skupu svih trojki kojima odgovara jedna homogena tačka odgovara prava u 4-dimenzionom prostoru jer sve tačke oblika  $(tx, ty, tz, tW)$  ( $t \neq 0$ ) pripadaju jednoj pravoj 4D prostora.
- Transformacije su predstavljene matricama 4x4.
- Orientacija uglova je po dogovoru pozitivna.

#### 3.5.1 Osnovne transformacije

**Translacija:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Kompozicija dve translacije je translacija.

**Rotacija:**

- U 2D prostoru postojala je samo jedna osa rotacije, ovde ih može biti beskonačno mnogo, tj može se rotirati oko proizvoljnog vektora  $\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$

- Matrica ove transformacije je prilično komplikovana, pa pokušajmo korišćenjem drugačijeg pristupa.
- Svaka rotacija se može predstaviti kao kompozicija 3 različite rotacije oko 3 ose: oko  $x$  ose u  $yz$  ravni za ugao  $\varphi_1$ , oko  $y$  ose u  $xz$  ravni za ugao  $\varphi_2$  i oko  $z$  ose u  $xy$  ravni za ugao  $\varphi_3$ .

**Rotacija oko  $z$  ose:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \varphi & -\sin \varphi & 0 & 0 \\ \sin \varphi & \cos \varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Kompozicija dve rotacije oko  $z$  ose je rotacija oko ose  $z$ .

**Rotacija oko  $x$  ose:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \varphi & -\sin \varphi & 0 \\ 0 & \sin \varphi & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

**Rotacija oko  $y$  ose:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \varphi & 0 & \sin \varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \varphi & 0 & \cos \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Da bismo dobili kompozitnu matricu rotacije, možemo iskombinovati ove matrice transformacije.
- Za datu osu  $u$  i zadati ugao  $\Psi$ , računanje ova tri ugla za koja treba vršiti rotaciju može biti teško. Rešenje? Učiniti problem jednostavnijim preslikavanjem ose  $u$  na jednu od koordinatnih osa. To možemo uraditi na sledeći način:
  1. odrediti ugao  $\Theta$  za koji treba rotirati osu  $u$  oko  $y$  ose da bi došla u ravan  $xy$ ,
  2. odrediti ugao  $\Phi$  za koji treba rotirati oko  $z$  ose da bi se ona poravnala sa  $x$  osom,
  3. sad kada je vektor  $u$  porvnat sa  $x$  osom izvršiti rotaciju objekta za ugao  $\Psi$  oko  $x$  ose,
  4. izvršiti inverzne transformacije transformacijama poravnanja (rotacijama).
- Sve navedene matrice imaju inverzne matrice. Translacije i rotacije su izometrijske transformacije i njima odgovaraju matrice čije su gornjeleve  $3 \times 3$  matrice ortogonalne.

**Skaliranje:**

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Kompozicija dva skaliranja je skaliranje.

**Smicanje ili iskošenje:** U 3D prostoru smicanje podrazumeva iskošenje objekta u smeru paralelnom nekoj koordinatnoj ravni.

$$\begin{bmatrix} 1 & sh_{xy} & sh_{xz} & 0 \\ sh_{yx} & 1 & sh_{yz} & 0 \\ sh_{zx} & sh_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Na primer, matrica smicanja u smeru koordinatne ravni  $Oxy$  imala bi formu:

$$\begin{bmatrix} 1 & 0 & a & 0 \\ 0 & 1 & b & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Korisne su za generisanje perspektivnih projekcija (na primer, kako nacrtati (3D) kocku na (2D) ekranu).



### 3.5.2 Kompozicije transformacija

- Svakoj kompoziciji rotacija, skaliranja, translacija i smicanja odgovara neka matrica oblika:

$$\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R^* & t^* \\ \mathbf{0} & 1 \end{bmatrix} =$$

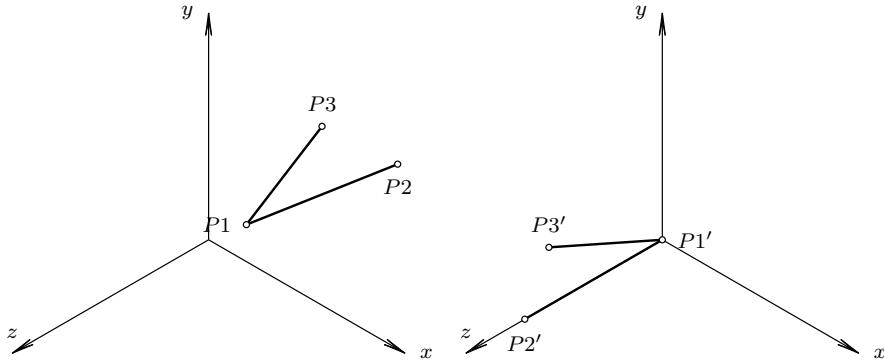
gde matrica  $R^*$  opisuje kombinovani efekat rotacija, skaliranja i smicanja, a  $t^*$  vektor koji opisuje kombinovani efekat svih translacija.

- U skladu sa ovim, umesto korišćenja množenja matricom  $4 \times 4$ , može se koristiti efikasnije izračunavanje:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = R^* \begin{bmatrix} x \\ y \\ z \end{bmatrix} + t^*$$

### 3.5.3 Primer kompozicije 3D transformacija

Zadatak je odrediti transformaciju koja preslikava trojku tačaka  $(P_1, P_2, P_3)$  u podudarnu trojku tačaka  $(P'_1, P'_2, P'_3)$ , tako da je  $P'_1$  koordinatni početak, duž  $P'_1P'_2$  pripada pozitivnom delu  $z$  ose, a duž  $P'_1P'_3$  leži u delu  $Oyz$  ravni u kome je pozitivna vrednost  $y$  koordinate.



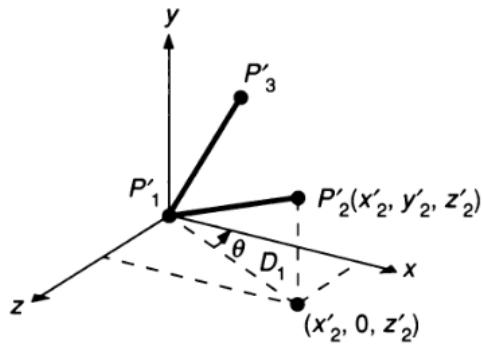
Postoje dva pristupa rešavanju ovog problema: prvi se sastoji u tome da iskombinujemo primitivne rotacije i translacije, a drugi koristi svojstva ortogonalnih matrica.

U prvom pristupu, ideja je da težak problem izdelimo na jednostavnije potprobleme. U ovom slučaju željena transformacija se može izvesti u četiri koraka:

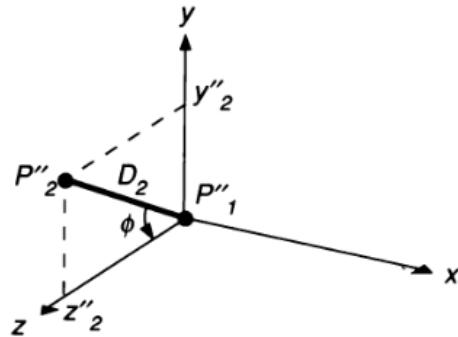
1. translirati  $P_1$  u koordinatni početak;
2. rotirati oko  $y$  ose tako da  $P_1P_2$  pripada  $(y, z)$  ravni (videti sliku 3.1);
3. rotirati oko  $x$  ose tako da  $P_1P_2$  pripada  $z$  osi (videti sliku 3.2);
4. rotirati oko  $z$  ose tako da  $P_1P_3$  pripada  $(y, z)$  ravni (videti sliku 3.3).

Rezultujuća matrica je oblika:  $T = R_z(\alpha) \cdot R_x(\Phi) \cdot R_y(\Theta - 90) \cdot T(-x_1, -y_1, -z_1)$ .

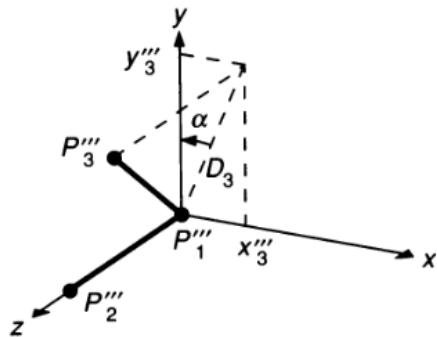
Drugi metod se zasniva na neposrednom određivanju koeficijenata matrice transformacije (i uz korišćenje svojstava ortogonalnih matrica).



Slika 3.1: Rotacija oko  $y$  ose: projekcija duži  $P'_1P'_2$  koja je dužine  $D_1$  se obara u ravan  $yz$ . Ugao za koji se rotira je  $-(90 - \Theta) = \Theta - 90$



Slika 3.2: Rotacija oko  $x$  ose:  $P''_1P''_2$  se rotira u  $z$  osu za pozitivan ugao  $\Phi$ ;  $D_2$  je dužina duži



Slika 3.3: Rotacija oko  $z$  ose: projekcija duži  $P'_1P'_3$  čija je dužina  $D_3$  se rotira za pozitivni ugao  $\alpha$  u  $y$  osu, dovodeći time samu duž u ravan  $Oyz$ .

### 3.6 Preslikavanje tačaka, pravih i ravnih

- Matrice transformacija određuju slike pojedinačnih tačaka.
- Slika prave se može odrediti kao prava određena slikama dve tačke orig-

inalne prave.

- Slika ravni se može odrediti kao ravan određena slikama tri nekolinearne tačke originalne ravni.
- Slika ravni se može odrediti i koristeći njen vektor normale.

### 3.7 Transformacije kao promena koordinatnog sistema

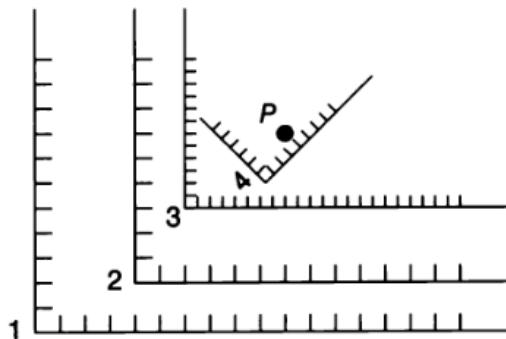
- Do sada su transformacije korišćene za preslikavanje skupa tačaka jednog koordinatnog sistema u isti taj koordinatni sistem. U tom kontekstu, možemo reći da je objekat transformisan, a da je koordinatni sistem ostao isti. S druge strane, istu transformaciju možemo opisati kao promenu koordinatnog sistema (pri čemu smatramo da se objekat ne transformiše, već se samo računaju njegove koordinate u novom koordinatnom sistemu).
- Prvi stil razmišljanja može biti pogodniji kada se objekat kreće, a drugi, na primer, kada više objekata u svojim pojedinačnim koordinatnim sistemima treba objediti u jedinstven globalni koordinatni sistem.
- Za promenu koordinatnog sistema mogu se koristiti iste tehnike kao i za transformacije.
- Označimo sa  $M_{i \leftarrow j}$  matricu transformacije koja prevodi koordinatni sistem  $j$  u koordinatni sistem  $i$ . Neka je  $P^{(i)}$  reprezentacija tačke  $P$  u koordinatnom sistemu  $i$ ,  $P^{(j)}$  u sistemu  $j$  i  $P^{(k)}$  u sistemu  $k$ . Tada važi:

$$P^{(i)} = M_{i \leftarrow j} P^{(j)} = M_{i \leftarrow j} M_{j \leftarrow k} P^{(k)} = M_{i \leftarrow k} P^{(k)}$$

i

$$M_{i \leftarrow j} M_{j \leftarrow k} = M_{i \leftarrow k}$$

- Navedena jednakost pokazuje i da je, za proizvoljnu promenu koordinatnog sistema, dovoljno imati opis svodenja iz različitih sistema na jedan sistem.



Slika 3.4: Koordinatni sistemi 1, 2, 3 i 4

- Na slici 3.4 prikazana su četiri različita koordinatna sistema. Možemo jednostavno zaključiti da je matrica transformacije iz koordinatnog sistema 2 u sistem 1 jednaka  $M_{1 \leftarrow 2} = T(4, 2)$  i slično  $M_{2 \leftarrow 3} = T(2, 3) \cdot S(0.5, 0.5)$  i  $M_{3 \leftarrow 4} = T(6.7, 1.8) \cdot R(45^\circ)$ , te je stoga npr:  $M_{1 \leftarrow 3} = T(4, 2) \cdot T(2, 3) \cdot S(0.5, 0.5)$ .
- Matrice  $M_{i \leftarrow j}$  imaju isti oblik kao i matrice transformacija.
- Primer: naredna matrica:

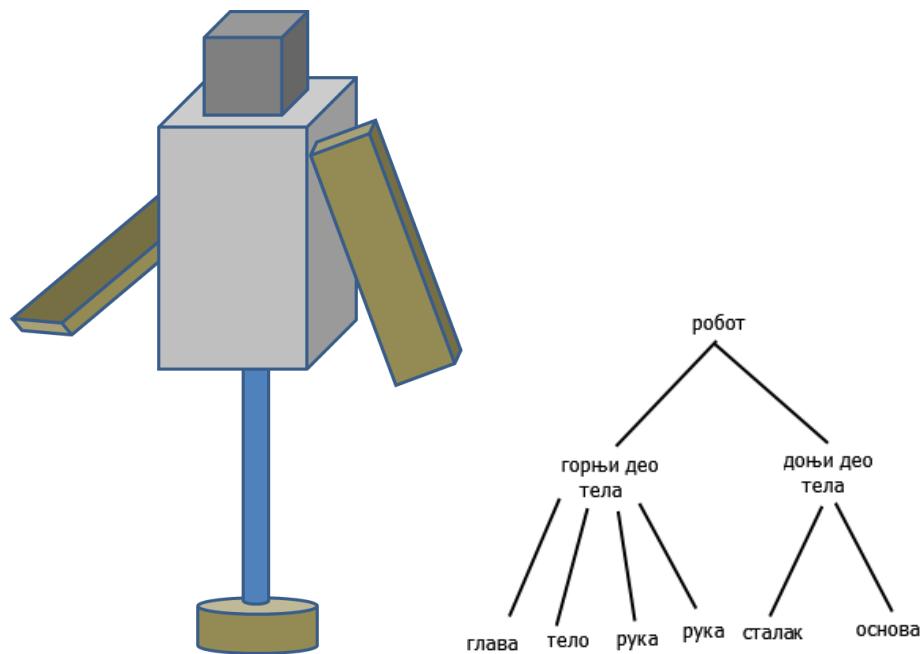
$$M_{R \rightarrow L} = M_{L \rightarrow R} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

transformiše desno orijentisan koordinatni sistem u levo orijentisan koordinatni sistem (i obratno).

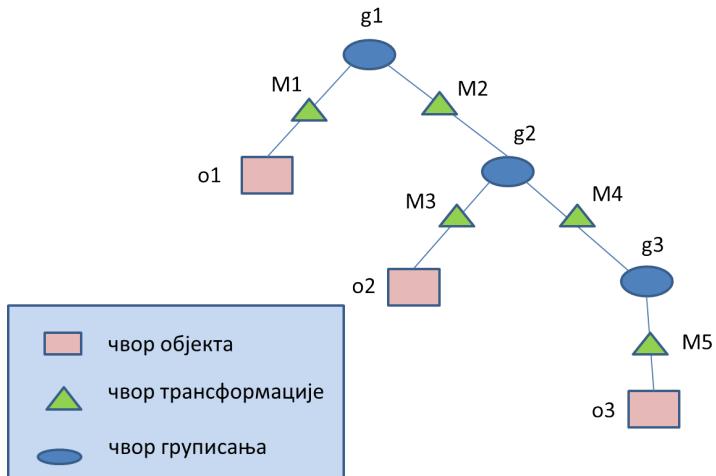
- Primer promene koordinatnog sistema: ako je automobil opisan u jednom koordinatnom sistemu i želimo da odredimo koordinate jedne tačke na točku dok se okreće, možemo najpre da za čitav točak promenimo koordinatni sistem (i izaberemo neki pogodan, npr. onaj u kojem je središte točka koordinatni početak i u kojem je osa točka jednaka  $y$  osi) i da u njemu računamo nove koordinate tačke.

### 3.8 Transformacije i graf scene

- Objekti su najčešće sačinjeni od komponenti. Na primer, ako posmatramo robota, on se sastoji od gornjeg i donjeg dela, a svaki od njih od pojedinačnih komponenti.

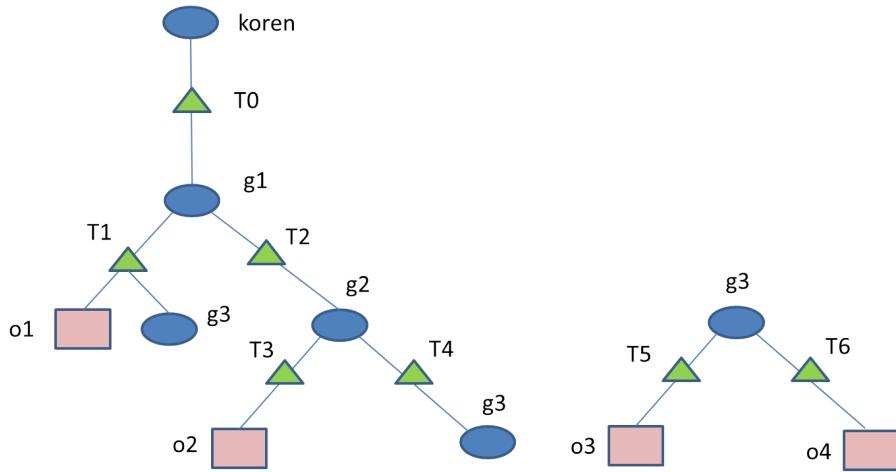


- Graf scene se najčešće sastoji iz sledećih elemenata:
  - objekata (to su kocke, valjci, lopte, ...) koji se čuvaju kao čvorovi (podrazumevano su jedinične veličine i pozicionirani u koordinatnom početku),
  - atributa (boja, tekstura, ...) koji se takođe čuvaju kao čvorovi,
  - transformacija koje su predstavljene takođe čvorovima.
- Da bismo pojednostavili format grafa prepostavimo:
  - da se atributi čuvaju kao komponente čvorova,
  - čvor transformacije deluje na podstablu,
  - samo listovi su grafički elementi,
  - svi čvorovi koji nisu listovi niti čvorovi transformacija su čvorovi grupisanja.
- Primena transformacija ide na sledeći način:
  1. različite transformacije se primenjuju na svaki od listova (glava, osnova robota, ...),
  2. transformacije se primenjuju na grupe objekata (gornji i donji deo tela, ...).
- Ovaj format omogućava da umesto da pravimo novu primitivu za svaki novi objekat koji nam je potreban, možemo samo primeniti transformaciju na manji skup primitiva da bi se kreirali složeni 3D oblici.
- Kumulativna matrica transformacije se gradi kako se penjemo uz stablo. Transformacije na višem nivou se dodaju na početak niza.
- Primer:



- Takođe, moguće je jednostavno ponovo iskoristiti već definisane grupe objekata. Ovo se može desiti ako imamo više sličnih komponenti u sceni: na primer, 2 robotove ruke.

- Transformacije definisane u okviru grupe 3 se ne menjaju: postoje različite kumulativne matrice transformacija za svako korišćenje grupe 3 kao celine, na primer:



- Postoje dve vrste transformacija pri modelovanju:
  - transformacije instance* – koriste se za pozicioniranje, izmenu veličine, orijentacije da bi se ta instanca dobro uklopila u scenu odnosno u objekat višeg nivoa
  - zglobne transformacije* – koriste se za simulaciju pokreta u zglobu tokom animacije.
- Ponovno korišćenje komponenti je jedan od osnova hijerarhijskog modelovanja. Postoje dva načina ponovnog korišćenja:
  - ponovno korišćenje dizajna – ovde nema ponovnog korišćenja komponenti jer želimo da postignemo nezavisnost komponenti – npr da imamo mogućnost individualne kontrole količine rotacije u svakom od zglobova noge robota
  - ponovno korišćenje komponenti – ukoliko imamo čvrstu komponentu koja nema unutrašnjih zglobova ona se može ponovo iskoristiti kao komponenta.

### 3.9 Pitanja

- Ravan je data svojom jednačinom  $Ax+By+Cz+D = 0$ . Kako za dve tačke u prostoru jednostavno odrediti da li su sa različitih strana ove ravni?
- Koliki je ugao između vektora  $\vec{u}(-8, 6)$  i  $\vec{v}(3, 4)$ ?
- Ako je 2D transformacija data matricom transformacije  $T$ , a  $e_1$  i  $e_2$  predstavljaju standardnu bazu vektorskog prostora, čemu su jednake vrednosti  $T(e_1)$  i  $T(e_2)$ ? Kako nam ovo može poslužiti za izvođenje matrice transformacije  $T$ ?

- 3.4 Čemu je jednaka matrica skaliranja? Da li se skaliranjem čuvaju uglovi između pravih u ravni?
- 3.5 Kako izgleda matrica rotacije oko koordinatnog početka za ugao  $\phi$ ? Kako je možemo dobiti?
- 3.6 Kako se homogenim koordinatama zadaje tačka u 2D?
- 3.7 Koja je to tačka sa homogenim koordinatama  $(x, y, 0)$ ?
- 3.8 Čemu u Dekartovom prostoru pripadaju sve homogenizovane koordinate tačaka?
- 3.9 Kako u homogenim koordinatama predstaviti naredne transformacije: skaliranje, rotaciju, transliranje?
- 3.10 Šta važi za afine transformacije?
- 3.11 Koja je transformacija inverzna skaliranju/translaciji/rotaciji?
- 3.12 Kako se primenjuje kompozicija transformacija?
- 3.13 Šta je kompozicija više skaliranja/translacija/rotacija?
- 3.14 Sta je smicanje?
- 3.15 Kog je oblika matrica izometrijskih transformacija?
- 3.16 Koje transformacije od pomenutih jesu izometrijske? (a) rotacija (b) skaliranje (c) translacija
- 3.17 Kako se može predstaviti svaka afina transformacija?
- 3.18 Dokazati da 2D rotacija i skaliranje komutiraju ako je  $s_x = s_y$  ili je  $\Theta = n\pi$  za celobrojno  $n$ , a inače ne komutiraju.
- 3.19 Data je jedinična kocka sa donjim levim temenom u tački  $(0,0,0)$  i gornjim desnim u tački  $(1,1,1)$ . Izvesti matricu transformacije potrebne za rotiranje kocke za  $\Theta$  stepeni oko glavne dijagonale u smeru suprotnom od smera kazaljke na časovniku kada se gleda duž dijagonale ka koordinatnom početku.
- 3.20 Kako odrediti sliku prave pri nekoj transformaciji? Kako odrediti sliku ravni pri nekoj transformaciji?
- 3.21 U kojim situacijama je potrebno izvršiti promenu koordinatnog sistema?
- 3.22 Od čega se sastoji graf scene?
- 3.23 Da li uz svaki objekat u grafu scene čuvamo i njegovu dimenziju (npr stranicu kocke)?
- 3.24 Čemu služe čvorovi grupisanja u grafu scene?
- 3.25 Nacrtati graf scene za model kućice koja se sastoji od kocke čija je stranica dužine 2 i na kojoj je postavljena piramida i čiji se donji levi ugao nalazi na poziciji  $(2, 2)$ .

## *Glava 4*

---

# Projektovanje

---

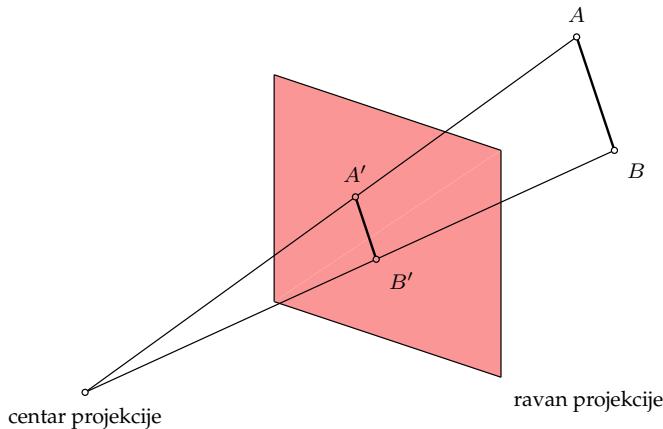
Prikazivanje 3D objekata je znatno komplikovanije od prikazivanja 2D objekata iz razloga što su uređaji za prikaz dvodimenzionalni. Rešenje za ovo neslaganje između trodimenzionalnih objekata i dvodimenzionalnih uređaja za prikaz leži u uvođenju projekcija: 3D objekti se obično prikazuju tako što se prikazuje njihova projekcija na ravan. Konceptualno, objekti iz 3D sveta se odsecaju u odnosu na **3D zapreminu pogleda** (eng. 3D view volume), a zatim projektuju na ravan. Sadržaj projekcije zapremine pogleda na ravan projekcije, koji se naziva **prozor** (eng. window) se dalje transformiše u oblik za prikaz.

U opštem slučaju **projekcija** je preslikavanje iz koordinatnog sistema dimenzije  $n$  u koordinatni sistem dimenzije manje od  $n$ . Od posebnog interesa za računarsku grafiku su projekcije iz 3D u 2D. Projekcije koje preslikavaju u ravan nazivamo **planarne projekcije**. Mnoge kartografske projekcije nisu planarne.

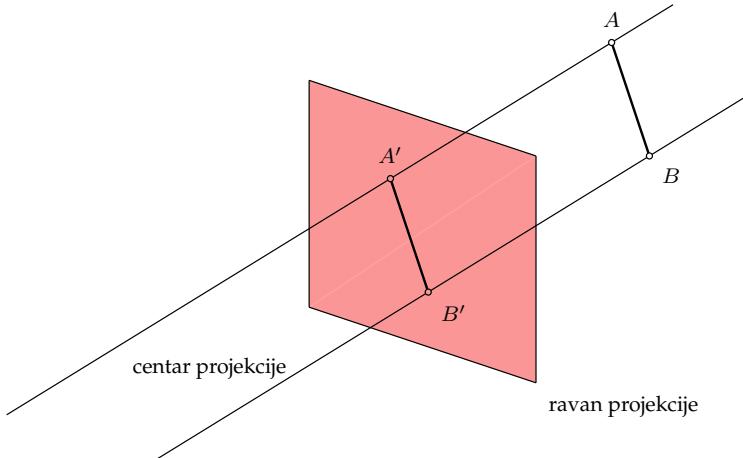
Projekcija iz 3D u 2D je određena **centrom projekcije** i **ravnim projekcijama**.

### **4.1 Tipovi projektovanja**

- 3D planarne projekcije mogu biti:
  - **perspektivne** (ako je centar projekcije na konačnom rastojanju od ravni projekcije),



- **paralelne** (ako je centar projekcije na beskonačnom rastojanju od ravni projekcije).



- Za perspektivnu projekciju eksplisitno se zadaje **centar projekcije** (simulira se šta naše oko odnosno kamera vidi), a za paralelnu projekciju zadaje se **pravac projekcije** (prave ne konvergiraju ka "oku", tj. centru projekcije).
- Paralelna projekcija se često koristi u inženjerstvu i arhitekturi, za potrebe merenja.
- U opštem slučaju je projekcija određena položajem ravni projekcije u odnosu na glavne ose objekta (odgovarajući ugao i pozicija) i uglom koji pravac projektovanja zahvata sa ravni projekcije
- Centar projekcije može biti predstavljen homogenim koordinatama:  $(x, y, z, 1)$ .
- Pravac projekcije može biti zadat kao vektor, tj. kao razlika dve tačke (predstavljene homogenim koordinatama):  $(x, y, z, 1) - (x', y', z', 1) = (a, b, c, 0)$ . Dakle, pravcu projekcije odgovara beskonačno daleka tačka. Perspektivna projekcija čiji je centar beskonačno daleka je upravo paralelna projekcija.

- Vizualni efekat perspektivne projekcije odgovara ljudskom vizualnom sistemu i daje tzv. perspektivno skraćivanje. Mada slika u ovoj projekciji izgleda realistično, njen praktičan značaj može biti mali za određivanje tačnih oblika i dimenzija predstavljenih objekata; paralelne prave se ne preslikavaju uvek u paralelne prave.
- Paralelna projekcija daje manje realističnu sliku; određivanje oblika i dimenzija je jednostavnije nego kod perspektivne projekcije; paralelne prave se preslikavaju u paralelne prave; kao i u perspektivnoj projekciji uglovi se čuvaju samo na ravnima koje su paralelne sa ravni projekcije.

## 4.2 Perspektivna projekcija

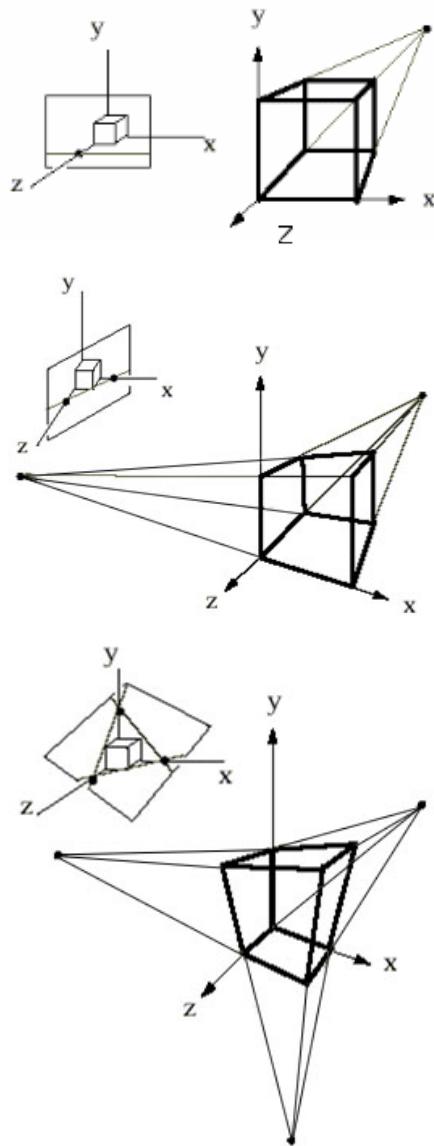
- Koristi je ljudski vizualni sistem; često se koristi u umetnosti.
  - Prednost: daje realističan prikaz i osećaj trodimenzionalnosti objekta
  - Mane: ne čuva se oblik objekta, niti odnosi (osim kada objekat seče ravan projekcije)
- Razlikuje se od paralelne projekcije u tome što se paralelne prave koje nisu paralelne ravn projekcije sekut. Veličina objekta se smanjuje sa povećanjem rastojanja i skraćenje nije uniformno



Slika 4.1: Ukoliko bismo ovu sliku prikazali korišćenjem paralelne projekcije, šine se ne bi sekle (preuzeto iz kursa U Brown)

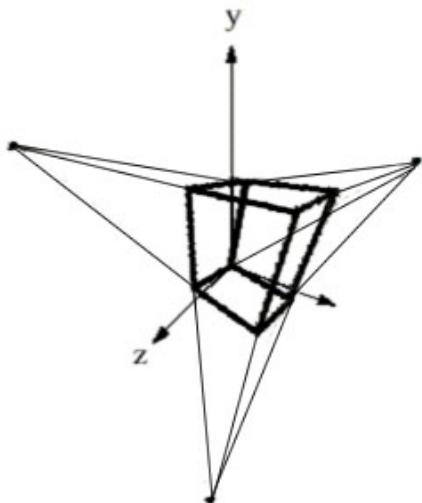
- Perspektivne projekcije bilo kog skupa paralelnih pravih (koje nisu paralelne ravn projekcije) sekut se u tzv. **tački nedogleda** (engl. vanishing point). U 3D paralelne prave se sekut u beskonačno dalekoj tački te se na tačku nedogleda može gledati kao na projekciju beskonačno daleke tačke.
- Ako je skup pravih paralelan sa koordinatnom osom, onda njenu tačku nedogleda zovemo **osna tačka nedogleda**. Postoje najviše tri takve tačke i prema njihovom broju mogu se razvrstavati perspektivne projekcije.

- Za pravougaone figure čije su normale strana upravne na koordinatne ose  $x$ ,  $y$  i  $z$ , broj osnih tačaka nedogleda je jednak broju koordinatnih osa koje presecaju ravan projekcije



Slika 4.2: Projekcija sa jednom, dve i tri osne tačke nedogleda (preuzeto iz kursa U Brown)

- Šta se dešava ukoliko se ista figura okrene tako da normale stranica nisu upravne na koordinatne ose  $x$ ,  $y$  i  $z$ ? Čak i ako ravan projekcije seče samo jednu osu ( $z$  osu), kreiraju se tri tačke nedogleda, te se nadalje mogu dobiti rezultati identični situaciji u kojoj ravan projekcije preseca sve tri koordinatne ose.



Slika 4.3: Perspektivna slika rotirane kocke (preuzeto iz kursa U Brown)

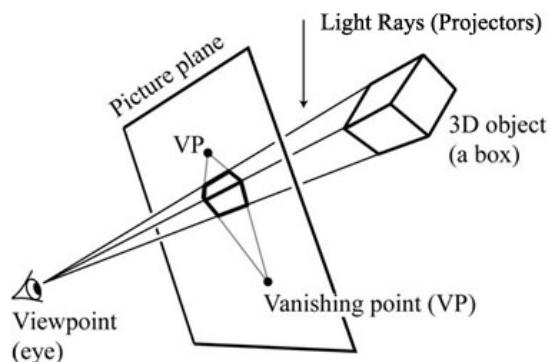
- Koncept tačke nedogleda se koristi u umetnosti, dok se u računarskoj grafici koristi koncept **tačke pogleda**, pri čemu je tačka pogleda lokacija virtuelne kamere (oka)



Leonardo da Vinči: studija za Poklonjenje mudraca (oko 1481)

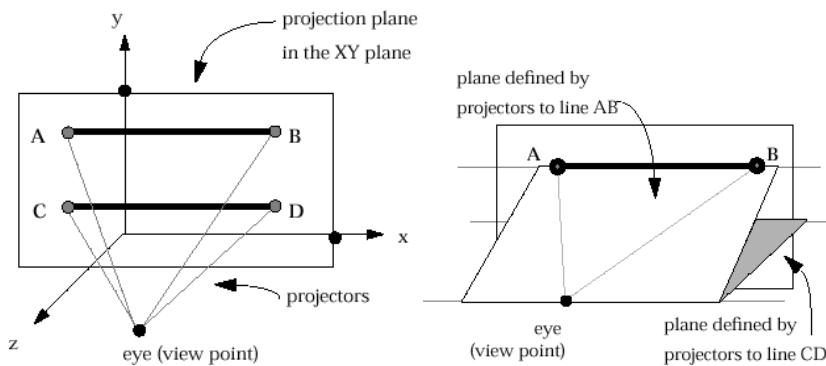


- Videli smo dva pristupa razumevanju perspektivne projekcije:
  1. perspektivna slika je rezultat skraćenja kao posledica konvergencije nekih paralelnih linija ka tački nedogleda
  2. perspektivna slika je presek ravni projekcije sa zracima svetla od objekta ka oku

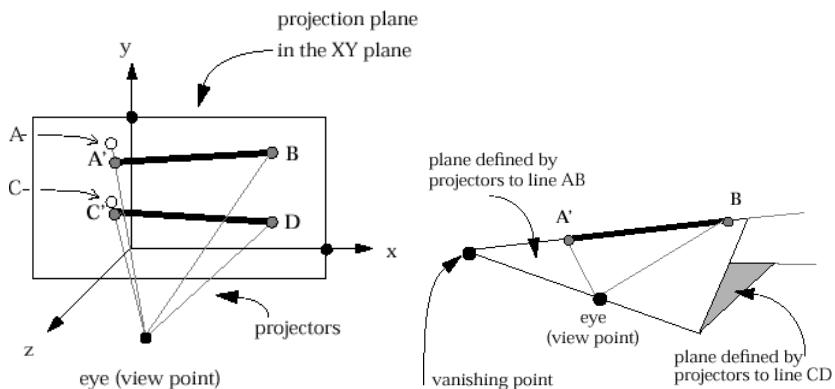


- Moguće je iskombinovati ova dva pristupa

- Na primer, neka je zadatak isprojektovati dve paralelne duži  $AB$  i  $CD$  na ravan  $xy$ . Projektivne prave iz oka ka dužima  $AB$  i  $CD$  određuju dve ravni koje se sekaju u pravoj koja sadrži tačku pogleda ili oka. Ova prava *ne seče* ravan projekcije  $xy$  jer joj je paralelna. Stoga ne postoji tačka nedogleda.



- Ukoliko su projekcije duži  $AB$  i  $CD$  (pri čemu su tačke  $A$  i  $C$  iza ravni projekcije) na ravan  $xy$  duži  $A'B$  i  $C'D$ , pri čemu duži  $A'B$  i  $C'D$  nisu paralelne, onda projektivne prave iz oka ka dužima  $A'B$  i  $C'D$  određuju dve ravni koje se sekaju po pravoj koja sadrži tačku pogleda. Ova prava *seče* ravan projekcije i tačka preseka je tačka nedogleda.

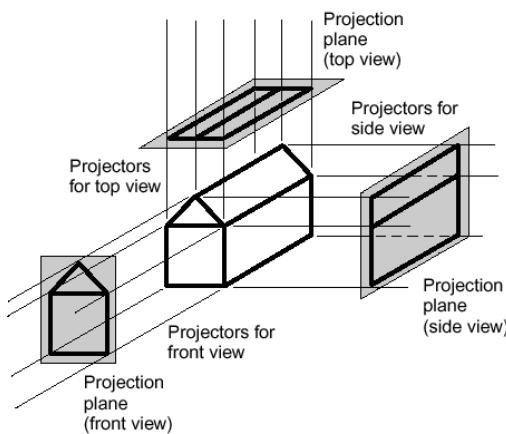


### 4.3 Paralelna projekcija

- Paralelne projekcije se dele prema odnosu pravca projekcije i normale ravni projekcije. Ako su ovi pravci jednaki (ili suprotni), onda projekciju zovemo **ortogonalna**. Ako to nije slučaj, onda je u pitanju **kosa projekcija**.
- U zavisnosti od odnosa projekcione ravni i koordinatnih osa, razlikujemo dve vrste ortogonalnih projekcija:
  - **ortografske** – kod njih je projekciona ravan upravna na neku od koordinatnih osa,

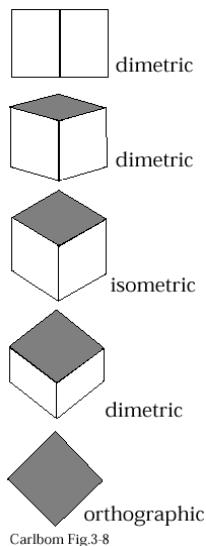
- **aksonometrijske** – kod njih projekciona ravan zahvata proizvoljni ugao sa koordinatnim osama.
- Postoji tri tipa ortografskih projekcija:
  - **pogled spreda** (eng. front-elevation),
  - **pogled odozgo** (eng. top-elevation) i
  - **pogled sa strane** (eng. side-elevation).

U svakoj od ovih projekcija, ravan projekcije je normalna na jednu od koordinatnih osa.



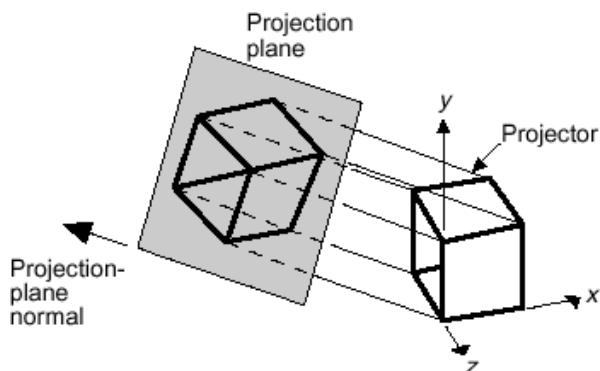
Slika 4.4: Ortogonalna projekcija: pogled spreda, odozgo i sa strane (preuzeto iz kursa U Brown)

- Najčešće se koristi za inženjerske crteže i arhitektonske crteže;
  - prednosti: dobijaju se precizne mere, u svim pogledima su objekti istih dimenzija,
  - manja: ne dobija se realističan pogled ili osećaj za 3D prirodu objekta, najčešće je potrebno koristiti više različitih pogleda da bi se dobio trodimenzionalni osećaj objekta.
- **Aksonometrijska projekcija** je paralelna projekcija koja koristi ravan projekcije koja nije paralelna nijednoj od koordinatnih osa (za razliku od perspektivnog projektovanja, skraćivanje je *uniformno*, i na njega ne utiče rastojanje od centra projekcije). Čuva se paralelnost pravih, ali ne i uglovi. Razlikujemo:
  - **izometrijsku projekciju**: normala ravni projekcije zahvata jednak ugao sa svim trima koordinatnim osama.
  - **dimetrijsku projekciju**: ugao između normale ravni projekcije i neke dve koordinatne ose je isti; potrebna su dva odnosa skaliranja,
  - **trimetrijsku projekciju**: uglovi između normale ravni projekcije i koordinatnih osa su međusobno različiti; potrebna su tri odnosa skaliranja.



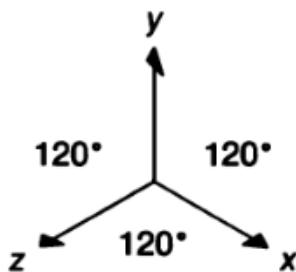
Slika 4.5: Tipovi aksonometrijskih projekcija

- Izometrijsko projektovanje je aksonometrijsko projektovanje u kojem normala ravni projektovanja zahvata podudarne uglove sa sve tri koordinatne ose (na ovaj način svim jednakim dužinama odgovaraju jednake dužine projekcija). Ovo znači da ako je normala ravni projekcije  $(n_1, n_2, n_3)$  onda važi:  $|n_1| = |n_2| = |n_3|$ . Postoji 8 takvih vektora normalne (po jedan u svakom oktantu). Na slici 4.7 prikazana je konstrukcija izometrijske projekcije duž pravca  $(1, -1, -1)$ . Izometrijsko projektovanje se koristi za ilustracije po katalozima, 3D modelovanje u realnom vremenu (Maya, AutoCad, ...)



Slika 4.6: Konstrukcija izometrijske projekcije jedinične kocke

- prednosti: nisu potrebni višestruki pogledi, ilustruje 3D prirodu objekta
- mane: nedostatak skraćenja daje "iskriviljen" prikaz, korisniji je za pravougaone nego za zakrivljene oblike

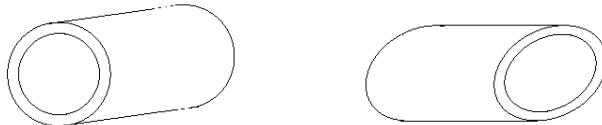


Slika 4.7: Izometrijska projekcija jediničnih vektora, sa pravcem projekcije  $(1, 1, 1)$



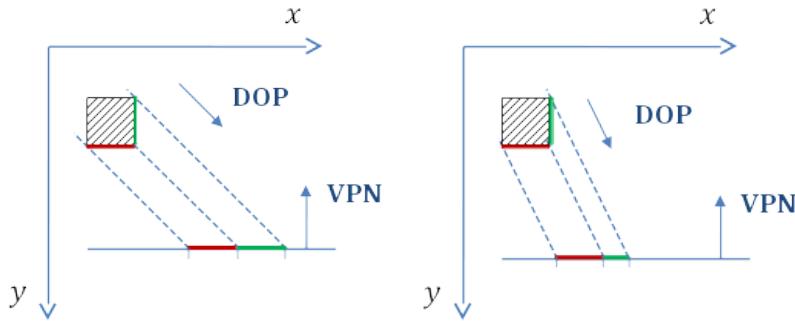
Slika 4.8: Primer korišćenja aksonometrijskih projekcija u računarskim igrama (preuzeto sa kursa U Brown)

- Aksonometrijske projekcije su godinama korišćene u video igrama. I danas se i dalje koriste kada je cilj da se objekti u daljini vide jednako dobro kao i bliži objekti (npr. u strategijama)
- **Kosa projekcija** koristi ravan projekcije koja je *upravna na neku od koordinatnih osa*, a pravac projekcije nije jednak (niti suprotan) normali ravni projekcije.
  - Prednosti: može da predstavi tačan oblik jedne strane objekta (mogu se izvesti precizna merenja); nedostatak perspektivnog skraćenja olakšava poređenje veličina; donekle daje utisak 3D izgleda objekta
  - Mane: objekti mogu da izgledaju iskrivljeno ukoliko se pozicija ravn projekcije ne izabere pažljivo (npr. krugovi postaju elipse, slika 4.9 b)); nedostaje skraćenje, pa je izgled nerealističan
- Neka od pravila za postavljanje ravn projekcije kod kose projekcije:



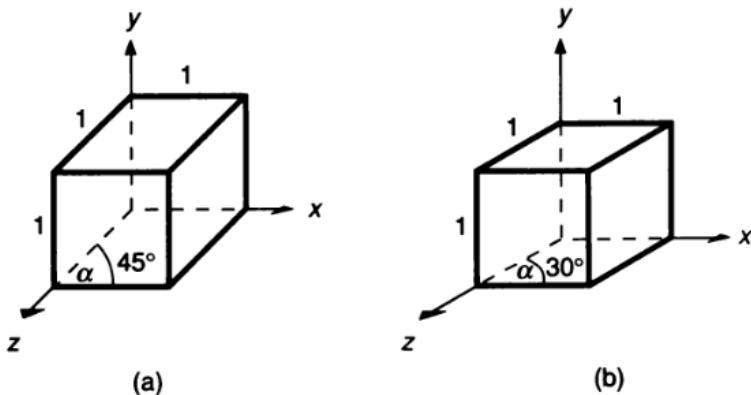
Slika 4.9: a) Ravan projekcije paralelna kružnoj strani objekta, b) Ravan projekcije nije paralelna kružnoj strani objekta (preuzeto sa kursa U Brown)

- ravan projekcije treba da bude paralelna strani objekta koja je najmanje pravilna, odnosno onoj strani koja sadrži najveći broj zakrivljenih površina
- ravan projekcije treba da bude paralelna najdužoj glavnoj strani objekta
- ravan projekcije treba da bude paralelna strani od interesa
- Najčešće korišćeni tipovi kosih projekcija su **cavalier** i **cabinet**.

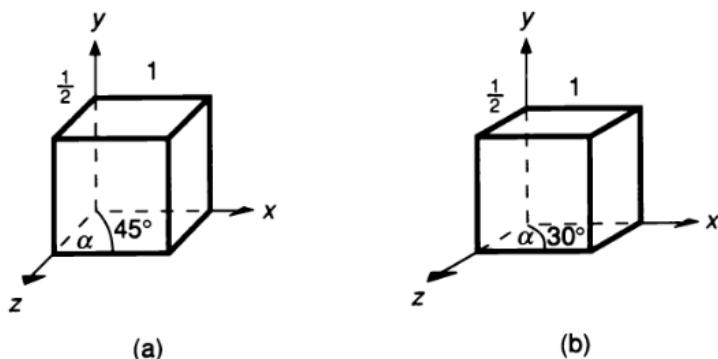


Slika 4.10: a) cavalier projektovanje b) cabinet projektovanje: DOP (direction of projection) - pravac projektovanja, VPN (view plane normal) - normala ravnini projektovanja (preuzeto sa U Brown)

- cavalier projektovanje koristi pravac projektovanja koji zahvata ugao od  $45^\circ$  sa ravni projektovanja. Kao rezultat, projekcije duži normalnih na ravan projekcije imaju iste dužine kao i same duži (tj. za njih nema skraćivanja)
- Ove projekcije se međusobno mogu razlikovati po uglu koji zahvata pravac projekcije sa koordinatnim osama. Tim uglom je određen i ugao između projekcija pravih koje su paralelne sa ravni projekcije i koje su normalne na nju. Na gornjoj slici taj ugao je  $45^\circ$ . Taj ugao je obično  $45^\circ$  ili  $30^\circ$
- cabinet projektovanje koristi pravac projektovanja koji zahvata ugao  $\arctg(2) \approx 63.4^\circ$  sa ravni projektovanja. Kao rezultat, projekcije duži normalnih na ravan projekcije imaju dva puta manje dužine u odnosu na same duži (što daje nešto realističniji prikaz)
- Slično kao i u cavalier projektovanju, ove projekcije se međusobno mogu razlikovati po uglu koji zahvata pravac projekcije sa koordinatnim osama.



Slika 4.11: Cavalier projekcije projekcija jedinične kocke na ravan  $z = 0$ . Sve projekcije stranica su jedinične dužine.



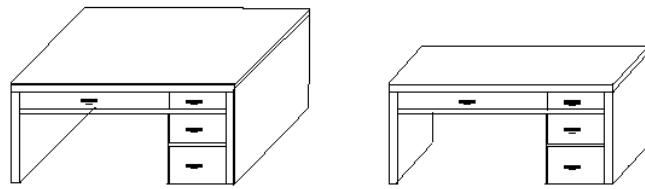
Slika 4.12: Cabinet projekcija jedinične kocke na ravan  $z = 0$ . Projekcije stranica paralelnih osama  $x$  i  $y$  su jedinične dužine

Tim uglom je određen i ugao između projekcija pravih koje su paralelne sa ravni projekcije i koje su normalne na nju. Na gornjoj slici taj ugao je  $45^\circ$ . Taj ugao je obično  $45^\circ$  ili  $30^\circ$ .

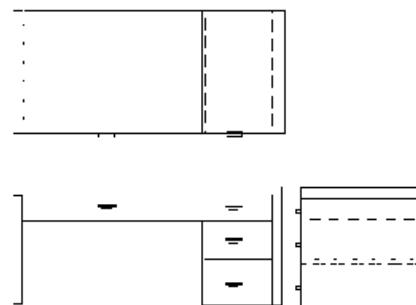
#### 4.4 Tipovi planarnih projekcija

#### 4.5 Primer izračunavanja projekcija tačaka

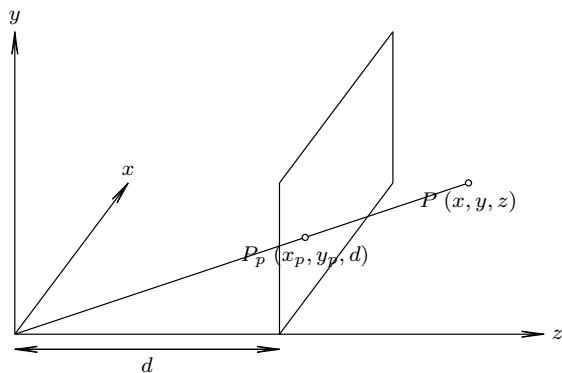
- Razmotrimo sledeći jednostavan primer perspektivnog projektovanja takvog da je:
  - centar projektovanja koordinatni početak;
  - ravan projekcije ravan  $z = d$ .



Carlbo Fig. 3-2



Slika 4.13: a) cavalier projektovanje b) cabinet projektovanje c) ortogonalno projektovanje iz više pogleda (preuzeto sa U Brown)



- Koordinate projekcije  $P_p(x_p, y_p, z_p)$  tačke  $P(x, y, z)$  računamo na osnovu sličnosti trouglova:

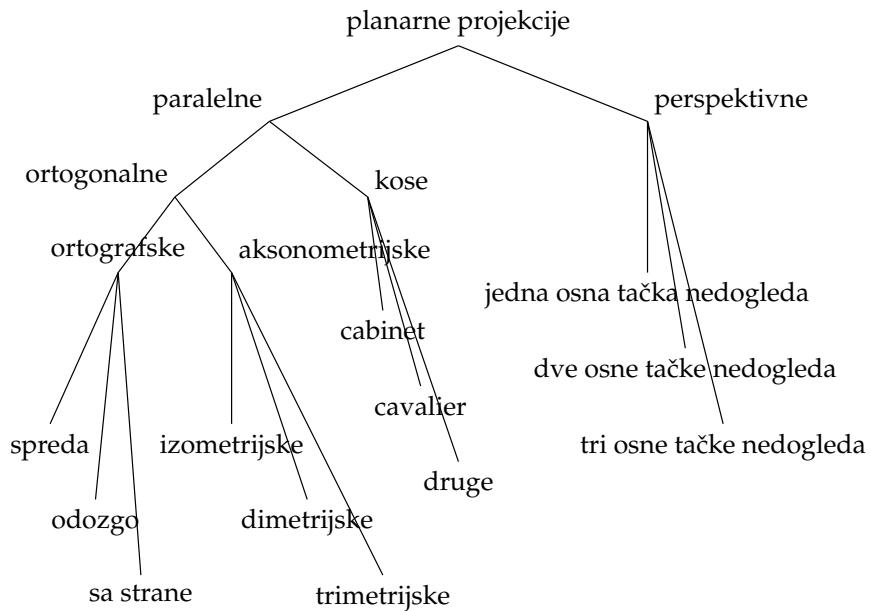
$$\frac{x_p}{d} = \frac{x}{z} \quad \frac{y_p}{d} = \frac{y}{z}$$

odakle je

$$x_p = \frac{x}{z/d} \quad y_p = \frac{y}{z/d}$$

(vrednosti nisu definisane za  $z = 0$ )

- Rastojanje  $d$



- Ovoj transformaciji odgovara matrica:

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

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{persp} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \frac{z}{d} \end{bmatrix}$$

$$\left( \frac{X}{W}, \frac{Y}{W}, \frac{Z}{W} \right) = (x_p, y_p, z_p) = \left( \frac{x}{z/d}, \frac{y}{z/d}, d \right)$$

- U 3D dekartovskom sistemu  $\left( \frac{x}{z/d}, \frac{y}{z/d}, d \right)$  je projekcija tačke  $(x, y, z)$ .

**4.6 Pitana**

- 4.1 Šta je projekcija?
- 4.2 Čime je određena projekcija?
- 4.3 Kako se mogu klasifikovati 3D planarne projekcije? U odnosu na šta se vrši klasifikacija?
- 4.4 Koje su karakteristike perspektivne projekcije?
- 4.5 Koje paralelne prave se sekut u perspektivne projekcije?
- 4.6 Kako zovemo tačku u kojoj se sekut paralelne prave? Koliko može biti takvih tačaka?
- 4.7 Kako zovemo tačku u kojoj se sekut prave paralelne koordinatnoj osi? Koliko najviše takvih tačaka može postojati?
- 4.8 Za pravougaone figure čije su normale strana upravne na koordinatne ose  $x, y, z$  kako se može odrediti broj osnih tačaka nedogleda?
- 4.9 Prema čemu se dele paralelne projekcije i koje dve vrste paralelnih projekcija razlikujemo?
- 4.10 Kako se dele ortogonalne projekcije u odnosu na ugao koji zahvata ravan projekcije sa koordinatnim osama?
- 4.11 Koliko tipova ortografskih projekcija razlikujemo?
- 4.12 Kako se dele aksonometrijske projekcije i šta važi za svaki od tipova?
- 4.13 Šta važi za vektor normale ravni projekcije kod izometrijskog projektovanja?
- 4.14 Šta važi za ravan projekcije, a šta za pravac projektovanja kod kosog projektovanja?
- 4.15 Nabrojati prednosti i mane kosog projektovanja.
- 4.16 Kako se bira ravan projekcije kod kosog projektovanja?
- 4.17 Koja su dva najčešća tipa kosih projekcija? Koliko je ugao između pravca projektovanja i normale ravni projektovanja kod svakog od ovih tipova?
- 4.18 Koliku dužinu imaju projekcije duži normalnih na ravan projekcije kod cavalier projektovanja, a koliku kod cabinet projektovanja?
- 4.19 Po čemu se međusobno mogu razlikovati cabinet projekcije?
- 4.20 Kod koje projekcije je ravan projekcije upravna na koordinatnu osu: (a) ortografska projekcija (b) izometrijska projekcija (c) cavalier projekcija?
- 4.21 Kod koje projekcije je projekcionala ravan upravna na pravac projekcije: (a) aksonometrijska projekcija (b) ortografska projekcija (c) cabinet projekcija?

- 
- 4.22 Kod koje projekcije normala ravni projekcije zahvata jednak ugao sa sve tri koordinatne ose? (a) kod aksonometrijske (b) kod izometrijske (c) kod trimetrijske projekcije
  - 4.23 Napisati matricu transformacije u homogenim koordinatama koja odgovara perspektivnom preslikavanju kod koga je ravan projektovanja ravan  $z = 0$ , a centar projekcije sa koordinatama  $(0, 0, -d)$ .
  - 4.24 Napisati matricu transformacije u homogenim koordinatama koja odgovara ortografskoj projekciji na ravan  $z = 0$ .

## *Glava 5*

---

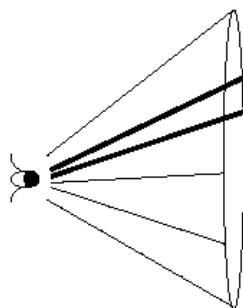
# Zadavanje sintetičkog modela kamere

---

Zašto ne renderovati sve što možemo da vidimo? Naše vidno polje je oko  $180^\circ$ . Čak i ako bismo izbacili deo koji možemo videti perifernim vidom i ograničili se na polje od  $120^\circ$ , slike bi delovalo iskrivljeno. To je delom tako zato što težimo tome da slike zauzimaju relativno mali deo vidnog polja. Na primer, računarski monitor na ugodnoj vidnoj distanci obuhvata samo  $25^\circ$ . Stoga je potrebno napraviti kompromis i voditi se pristupom koji koriste fotografija, a to je da se renderuje samo skroman deo vidnog polja.

**Oblast prikaza** (engl. *viewport*) je pravougaona oblast ekrana u kojoj se scena renderuje.

**Zapremina pogleda** (engl. *view volume*) sadrži sve ono što kamera vidi. U slučaju oka ta oblast je kupa. Matematika potrebna za kliping objekata u odnosu na konusnu površ je skupa, pa je *aproksimiramo* piramidom.

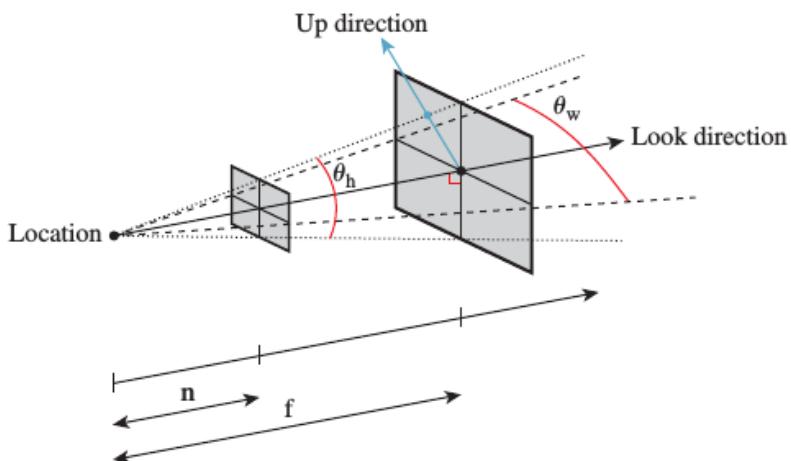


Slika 5.1: Konusna perspektivna zapremina pogleda (preuzeto sa U Brown)

## 5.1 Zadavanje kamere kod perspektivnog projektovanja

Kako konstruisati zapreminu pogleda kod perspektivnog projektovanja? Potrebno je poznavati 6 parametara sintetičkog modela kamere:

- poziciju kamere (tačka),
- smer pogleda i smer nagore (dva vektora),
- rastojanje prednje i zadnje ravni odsecanja (dva skalara) i
- vidno polje (ugao u stepenima).



Slika 5.2: Deo specifikacije sintetičke kamere (preuzeto iz knjige CG: Principles and Practice)

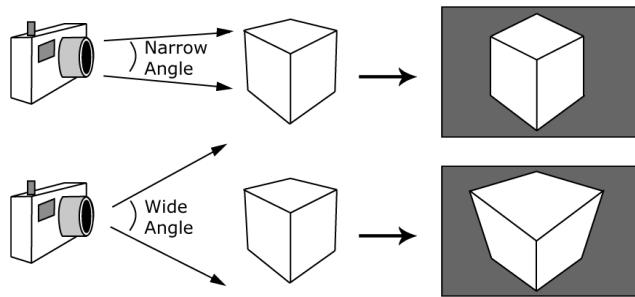
Na slici 5.2 su prikazane osnovne komponente specifikacije sintetičke kamere. Sintetičku kameru razmatramo kao "tačkastu": to znači da svi zraci koji ulaze u kameru ulaze kroz jednu tačku koja se naziva **pozicija kamere**. U postavci *realne* kamere, potrebno je voditi računa o raznim aspektima: poziciji kamere, njenoj orijentaciji, vidnom polju (koje se podešava kontrolom zuma na kameri). Za bolje kamere, postavlja se i žižna daljina (rastojanje tačaka koje su u najvećem fokusu na slici) i dubina polja (koliko daleko ispred i iza žižne daljine će objekti biti u fokusu). U modelu sintetičke kamere najčešće se ne razmatraju poslednje dve stavke jer je idealna kamera u fokusu na svim rastojanjima.

Pozicija kamere predstavlja centar projekcije i ona se nalazi iza ravni projektovanja u odnosu na objekat koji se projektuje.

Pored pozicije kamere važnu ulogu igraju i dva vektora: vektor pogleda (engl. look vector) i vektor nagore (engl. up vector). **Vektor pogleda** zadaje smer u kome je kamera usmerena. Ukoliko pustimo zrak iz pozicije kamere koji je usmeren u pravcu pogleda "udarićemo" u objekat koji će se prikazati u centru slike dobijene kamerom.

**Uglovi vidnog polja** opisuju koliko daleko od vektora pogleda, izraženo u stepenima, kamera može da vidi. Osnovna sintetička kamera proizvodi

kvadratnu sliku pa je vidno polje isto i u horizontalnom i u vertikalnom smeru. Ipak, u nekim sistemima je potrebno zadati posebno horizontalno, a posebno vertikalno vidno polje; u drugim pak sistemima treba zadati horizontalno vidno polje i **odnos visine i širine** (engl. aspect ratio) prozora prikaza, a vertikalno vidno polje se onda računa na osnovu tih podataka. Odnos visine i širine definiše dimenziju slike koja se projektuje, nakon čega se ona preslikava u oblast prikaza; dobra ideja je da ovaj odnos bude isti za prozor pogleda i oblast prikaza da ne bi došlo do deformisanja (istezanja ili suženja) slike.



Slika 5.3: Slika dobijena korišćenjem sočiva sa uskim i širokim vidnim uglom (preuzeto sa U Brown)

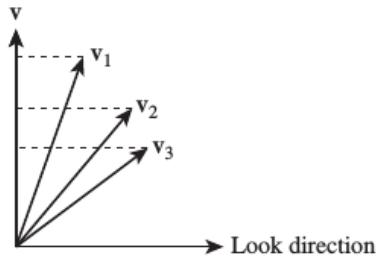
Vidno polje određuje koji deo scene će stati u zapreminu pogleda; što je veći ugao vidnog polja to će veći deo scene stati u zapreminu pogleda. Ovim se u stvari zadaje količina perspektivnog iskrivljenja na slici, od nula (kod paralelne projekcije) do velike vrednosti (kod širokougaonog objektiva); odabir ugla vidnog polja je analogan procesu kada fotograf biraju tip sočiva.

Jednu suptilnu tačku prilikom specifikacije sintetičke kamere predstavlja zadavanje **vektora nagore**, koji orijentise kameru oko vektora pogleda. Ako bismo zamislili vertikalni jedinični vektor nacrtan na pozadini kamere, onda ovaj vektor zajedno sa vektorom pogleda određuje ravan koja se naziva vertikalna ravan kamere. Prva ideja bi bila da korisnik treba da zada ovaj vektor (označimo ga sa  $v$ ), ali to u praksi nije tako jednostavno, te se od korisnika zahteva da zada proizvoljan nenula vektor u ovoj ravni (koji je različit od vektora pogleda) i na osnovu njega se računa vektor  $v$ . Na slici 5.4 prikazan je ovaj koncept: bilo koji od vektora  $v_1, v_2, v_3$  može od strane korisnika biti zadat kao vektor nagore, a onda se vektor  $v$  računa kao vektor u vertikalnoj ravni kamere upravan na vektor pogleda.

Dakle, **orientacija kamere** je definisana jediničnim vektorom upravnim na vektor pogleda u ravni definisanoj vektorom pogleda i vektorom nagore, tj. u vertikalnoj ravni kamere.

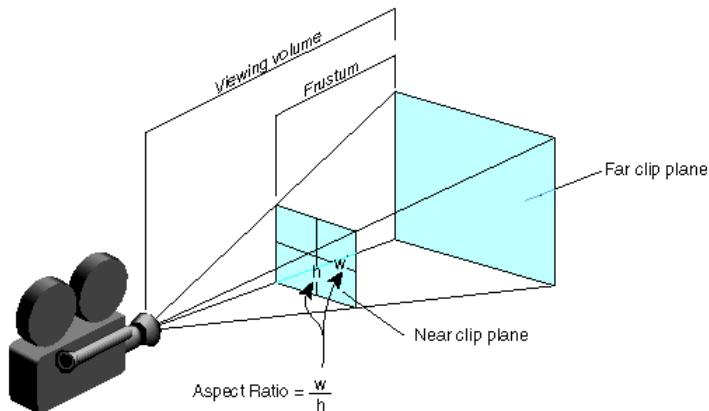
U ovom trenutku zadato je gde je kamera pozicionirana, u kom pravcu je usmerena i kako se telo kamere rotira oko smera pogleda. Vidno polje određuje koliko široku oblast kamera "vidi". Ovim smo indirektno opisali četverostranu zapreminu pogleda sa pravougaonim poprečnim presekom. Preostalo je da se zadaju još dve komponente: rastojanje prednje i zadnje ravni odsecanja.

**Prednja i zadnja ravan odsecanja** su paralelne ravni projektovanja i označavaju se vrednostima  $n$  (od engl. near) i  $f$  (od engl. far) koje predstavljaju njihova rastojanja od kamere. Prednja i zadnja ravan odsecanja isecaju



Slika 5.4: Bilo koji od vektora  $v_1, v_2, v_3$  može biti zadat kao vektor nagore, a onda se vektor  $v$  računa kao vektor u ravni određenoj vektorom pogleda i vektorom nagore, koji je upravan na vektor pogleda (preuzeto iz knjige: CG: Principles and Practice)

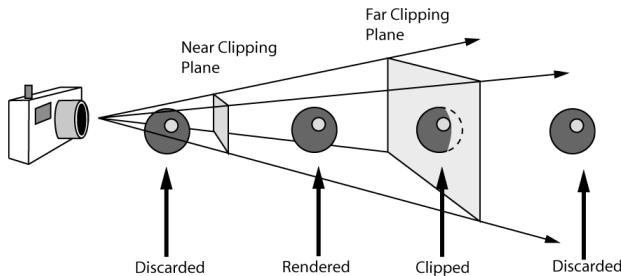
**zarubljenu četvorostranu piramidu pogleda** (engl. frustum). Objekti koji se nalaze u okviru ove oblasti biće prikazani na slici, a objekti van ove oblasti neće.



Slika 5.5: Rastojanja do prednje i zadnje ravni odsecanja se mere u smeru vektora pogleda (preuzeto sa U Brown)

Ovo je veoma koristan mehanizam: na ovaj način ograničava se domet pogleda kamere renderovanjem (delova) objekata koji se nalaze između prednje i zadnje ravni odsecanja i odsecanjem svega van njih. Razlog za to može biti što ne želimo da crtamo objekte koji su preblizu kameri (blokirali bi ostatak scene) ili predaleko od kamere (jer su isuviše mali da bi bili vizualno značajni, a uzimaju znatno vreme za renderovanje). Na ovaj način eliminise se i razmatranje objekata koji su iza kamere. Sve ovo donosi velike uštede u vremenu renderovanja.

U industriji igara se često koristio koncept prednje i zadnje ravni odsecanja da bi se eliminisali udaljeni objekti. Ponekad i danas možemo da vidimo sledeći efekat: dok se tokom igre pomeramo unapred udaljeni objekat se odjednom pojavi na ekranu. Ovaj efekat nije poželjan. Opšte rešenje koje se dugo koristilo jeste da se objekti u daljinu renderuju korišćenjem pozadinske ma-

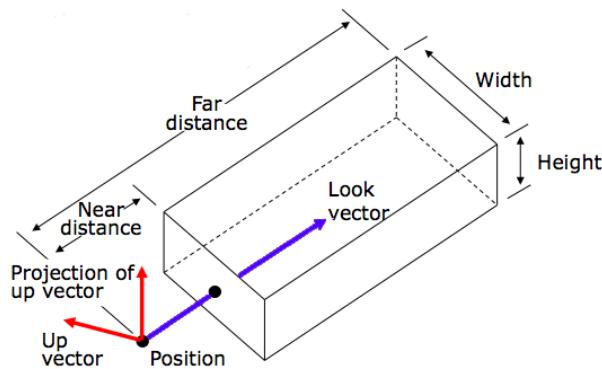


Slika 5.6: Primer odsecanja objekata u odnosu na prednju i zadnju ravan odsecanja (preuzeto sa U Brown)

gle te da se oni postepeno pojavljaju prilikom prilaska objektu. U novim igrama postoje bolji sistemi renderovanja i često se objekti prikazuju sa različitim nivoom detaljnosti. Stoga, kada su objekti udaljeni oni se renderuju korišćenjem manjeg broja poligona.

## 5.2 Zadavanje kamere kod paralelnog projektovanja

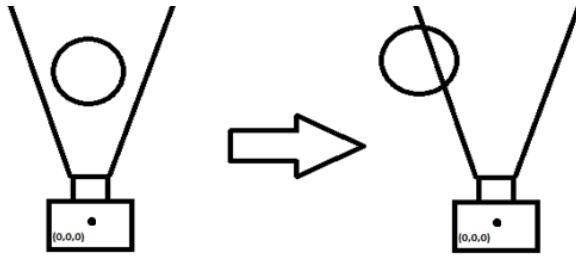
Da bismo konstruisali zapreminu pogleda kod paralelnog projektovanja, potrebno je zadati iste parametre kao i kod perspektivnog projektovanja, osim uglova vidnog polja, ali je zato potrebno zadati visinu i širinu zapremine pogleda. Paralelnna zapremina pogleda je u stvari paralelopiped (slika 5.7). Kod paralelnog projektovanja, objekti se prikazuju u istoj veličini koliko god bili udaljeni od kamere jer su svi projektivni zraci međusobno paralelni. Skraćenje je uniformno i zavisi od ugla koji zraci projekcije zahvataju sa ravni projekcije; dakle, ne postoji perspektiva zavisna od dubine. Kod paralelne zapremine pogleda odsecanje se izvodi jednostavnije jer su jednostavnije jednačine ravni, a takođe jednostavnije je projektovati 3D scenu u 2D projektivnu ravan jer nema perspektivnog skraćenja.



Slika 5.7: Paralelna zapremina pogleda (preuzeto sa U Brown)

### 5.3 Načini modelovanja kamere

Postoji više načina za modelovanje kamere. U opštem slučaju i kamera i objekti na sceni mogu se nezavisno transformisati, tj. pomerati. U restriktivnijem modelu kamera ostaje fiksirana na jednoj poziciji, te da bismo transformisali kameru potrebno je primeniti inverzne transformacije na objekte na sceni.

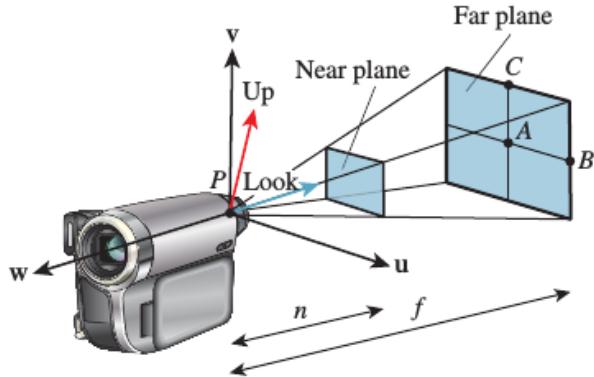


Slika 5.8: Transformacija kamere udesno izvedena kao transformacija objekata ulevo (preuzeto sa U Brown)

### 5.4 Izgradnja matrica transformacija na osnovu zadatih parametara kamere

Sada ćemo iz zadate specifikacije sintetičke kamere izgraditi:

1. ortonormirani koordinatni sistem sa koordinatnim početkom u središtu kamere,
2. nekoliko tačaka na kupi pogleda (kao na slici 5.9).



Slika 5.9:  $uvw$  okvir kamere, vektor pogleda, vektor nagore i vektor  $v$ , tačke  $A, B, C$  i  $P$  (preuzeto iz knjige CG: Principles and Practice)

Prilikom pisanja jednačina vektor nagore označićemo sa  $\vec{vup}$ , a vektor pogleda sa  $\vec{look}$ . Tačka  $P$  označava poziciju kamere.

Najpre ćemo izgraditi ortonormirani bazu  $\vec{u}$ ,  $\vec{v}$ ,  $\vec{w}$  u obrnutom redosledu. Vektor  $\vec{w}$  je jedinični vektor čiji je smer suprotan od smera vektora pogleda, te važi:

$$\vec{w} = \frac{-\overrightarrow{\text{look}}}{\|\overrightarrow{\text{look}}\|} \quad (5.1)$$

Da bismo konstruisali vektor  $\vec{v}$ , prvo je potrebno isprojektovati vektor  $\overrightarrow{vup}$  na ravan upravnu na vektor  $\vec{w}$ , tj. obzirom da su vektori  $\vec{w}$  i vektori  $\overrightarrow{\text{look}}$  istog pravca na ravan upravnu na smer vektora  $\overrightarrow{\text{look}}$ , a zatim postaviti da on bude jediničnog intenziteta:

$$\vec{v} = \overrightarrow{vup} - \text{proj}_{\vec{w}} \overrightarrow{vup} \quad (5.2)$$

S obzirom na to da su vektori  $\overrightarrow{vup}$  i  $\vec{w}$  jediničnog intenziteta, važi:

$$\text{proj}_{\vec{w}} \overrightarrow{vup} = (\overrightarrow{vup} \cdot \vec{w}) \vec{w} \quad (5.3)$$

odnosno važi:

$$\vec{v} = \overrightarrow{vup} - (\overrightarrow{vup} \cdot \vec{w}) \vec{w} \quad (5.4)$$

$$\vec{v} = \frac{\vec{v}}{\|\vec{v}\|} \quad (5.5)$$

Konačno, da bismo dobili ortonormirani sistem, vektor  $u$  se računa na sledeći način:

$$\vec{u} = \vec{v} \times \vec{w} \quad (5.6)$$

Sada ćemo izračunati tačke  $A$ ,  $B$ ,  $C$  i  $P$ . Ivica  $AB$  je naspram polovine horizontalnog vidnog polja iz  $P$  i na rastojanju je  $f$  od  $P$ , stoga važi:

$$\tan \frac{\Theta_h}{2} = \frac{AB}{f} \quad (5.7)$$

te je stoga:

$$AB = f \tan \frac{\Theta_h}{2} \quad (5.8)$$

gde  $\Theta_h$  označava horizontalni vidni ugao, konvertovan u radijane, odnosno:

$$\Theta_h = \text{VidnoPolje} \frac{\pi}{180} \quad (5.9)$$

i na sličan način se računa i  $\Theta_v$  - vertikalno vidno polje i dužina  $AC$ :

$$P = \text{Position} \quad (5.10)$$

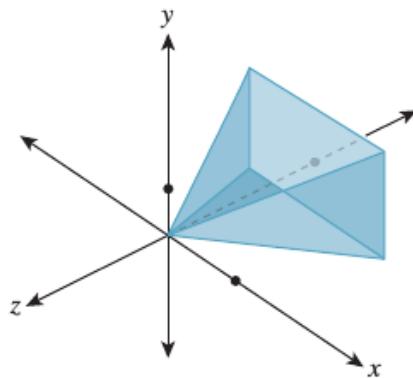
$$A = P - f \cdot \vec{w} \quad (5.11)$$

$$B = A + f \tan \frac{\Theta_h}{2} \vec{u} = P + f \tan \frac{\Theta_h}{2} \vec{u} - f \cdot \vec{w} \quad (5.12)$$

$$C = A + f \tan \frac{\Theta_v}{2} \vec{v} = P + f \tan \frac{\Theta_v}{2} \vec{v} - f \cdot \vec{w} \quad (5.13)$$

Može se primetiti da za sada nigde u računu nije korišćeno rastojanje od prednje ravni odsecanja  $n$ .

Sada ćemo iskoristiti tačke  $A$ ,  $B$ ,  $C$  i  $P$  za transformisanje zarubljene piramide pogleda u kanonsku zarubljenu piramidu pogleda prikazan na slici 5.10 koji se naziva i **standardna perspektivna zapremina pogleda**.



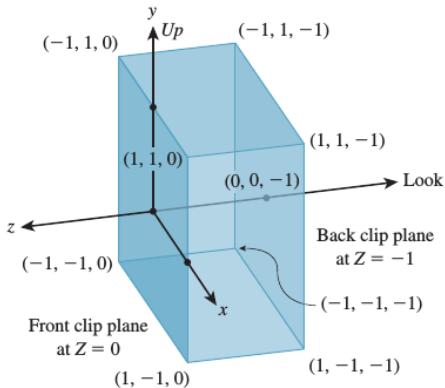
Slika 5.10: Standardna perspektivna zapremina pogleda je piramida koja ima granice od -1 do 1 po  $x$  i  $y$  i od 0 do -1 po  $z$  (skala po  $z$  je preuvečana na slici)

Sve što treba uraditi jeste zadati gde ove četiri tačke treba da se preslikaju. Potrebno je preslikati tačku  $P$  u koordinatni početak, tačku  $A$  u središte zadnje strane što je  $(0, 0, -1)$ , tačku  $B$  u središte desne ivice zadnje strane što je  $(1, 0, -1)$ , a tačku  $C$  u središte gornje ivice što je  $(0, 1, -1)$ . Matricu kojom se izvodi ova transformacija označićemo sa  $M_{persp}$ , a odgovarajuću transformaciju sa  $T_{persp}$ . Ovom transformacijom se tačke zadnje ravni odsecanja transformišu u ravan  $z = -1$ . Obzirom da se rastojanja duž zraka iz  $P$  do  $A$  transformišu linearno, tačke sa prednje ravni odsecanja se transformišu u ravan  $z = -n/f$  (ovo se dobija iz odnosa  $n : f = z : -1$  jer se zadnja ravan odsecanja slika u ravan  $z = -1$ ). Ovim smo transformisali proizvoljnu perspektivnu zapreminu pogleda u standardnu perspektivnu zapreminu pogleda i od ovog momenta sve što se radi je nezavisno od parametara kamere (osim što će faktor  $-n/f$  ući u neka računanja).

Sada ćemo primeniti još dve transformacije: prvu, da "otvorimo" piramidalnu zapreminu pogleda u pravougaoni paralelopiped, i drugu, kojom ćemo izvršiti projektovanje duž  $z$  ose. Postoje dva razloga za ovo:

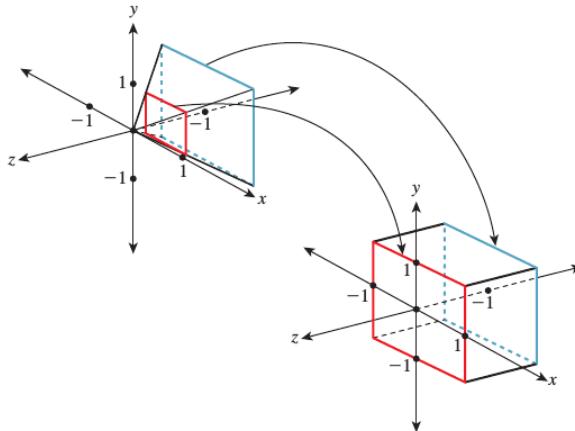
- zapremina pogleda u obliku paralelopipeda je jednostavnija za rad
- kada se vrši projekcija duž  $z$  ose, posebno je jednostavno odrediti koji objekti zaklanjavaju neke druge objekte, odnosno testiranje vidljivosti postaje trivijalno (ovo će biti od koristi u dizajniranju  $z$ -bafer algoritma)

**Standardna paralelna zapremina pogleda** je paralelopiped čije granice idu od -1 do 1 po  $x$  i  $y$  osi i od 0 do -1 po  $z$  osi. Njena prednja ravan odsecanja je  $z = 0$ , a zadnja  $z = -1$ . Sada ćemo transformisati deo standardne perspektivne zapremine pogleda između transformisane prednje i zadnje ravni



Slika 5.11: Standardna paralelni zapremina pogleda

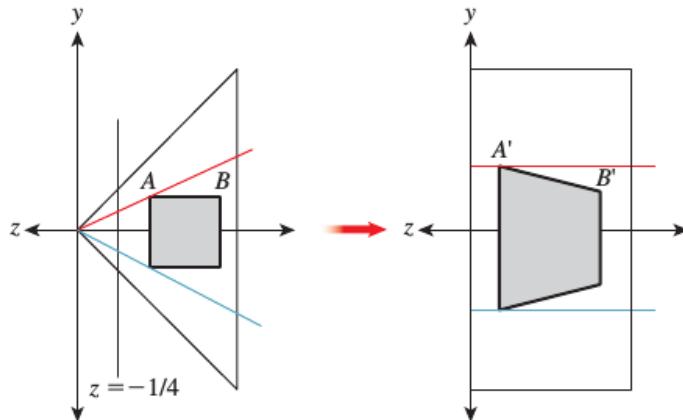
(deo između  $z = -n/f$  i  $z = -1$ ) u standardnu paralelnu zapreminu pogleda. Transformacija koja se koristi je projektivna transformacija u kojoj se svi zraci koji prolaze kroz zapreminu pogleda ka koordinatnom početku transformišu u zrake koji prolaze kroz zapreminu pogleda ka  $xy$  ravni u smeru  $z$  ose i to njenog pozitivnog dela (slika 5.12).



Slika 5.12: Transformacija perspektivne u paralelnu zapreminu pogleda

Primena ove transformacije ne utiče na finalni rezultat procesa renderovanja jer perspektivno projektovanje koje je oblika  $(x, y, z) \rightarrow (x/z, y/z, 1)$  u zapremini pogleda pre transformacije je isto kao paralelni projekcija  $(x, y, z) \rightarrow (x, y, 1)$  transformisanog oblika u zapremini pogleda nakon transformacije. Ovo je jednostavno uočiti ako bacimo pogled na dvodimenzionalni isečak, tj.  $yz$  ravan. Pogledajmo npr. kvadrat prikazan na slici 5.13 koji zauzima srednju polovinu perspektivnog pogleda na scenu. Informacija o tome koje tačke su zaklonjene nekim drugima se utvrđuje poretkom tačaka duž zraka od tačke pogleda do scene, te je tačka  $B$  zaklonjena bližom ivicom kvadrata. Nakon transformacije ovaj zrak postaje zrak u smeru  $-z$ . I dalje je tačka  $B'$  zaklon-

jena prednjom ivicom kvadrata. Takođe, i dalje transformisani kvadrat zauzima središnji deo paralelnog pogleda scene.



Slika 5.13: Standardna perspektivna zapremina pogleda levo (sa prednjom ravni odsecanja  $z = -1/4$ ) sadrži mali kvadrat koji se transformiše u paralelogram u paralelnoj zapremini pogleda desno

Podsetimo se da je projektivnu transformaciju u  $R^3$  moguće napisati kao linearnu transformaciju u  $R^4$  korišćenjem homogenih koordinata tačaka prćenim homogenizacijom koordinata. Ako sa  $c = -n/f$  označimo  $z$  koordinatu prednje ravni odsecanja nakon transformacije u standardnu perspektivnu zapreminu pogleda, matricu linearne transformacije iz perspektivne zapremine pogleda u paralelnu zapreminu pogleda (u oznaci  $M_{pp}$ ) možemo zapisati kao:

$$M_{pp} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/(1+c) & -c/(1+c) \\ 0 & 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & f/(f-n) & n/(f-n) \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Pošto će uskoro biti potrebno homogenizovati koordinate, matrica se može pomnožiti sa  $f - n$  i nakon toga može se koristiti jednostavnija matrica:

$$M_{pp} = \begin{bmatrix} f - n & 0 & 0 & 0 \\ 0 & f - n & 0 & 0 \\ 0 & 0 & f & n \\ 0 & 0 & -(f - n) & 0 \end{bmatrix}$$

Nećemo izvoditi ovu matricu, već nam je cilj da se uverimo da ona zaista transformiše zarubljenu piramidu između prednje i zadnje ravni odsecanja ( $z = c$  i  $z = -1$ ) standardne perspektivne zapremine pogleda u standardnu paralelnu zapreminu pogleda. To ćemo uraditi proverom uslova za uglove perspektivne zapremine pogleda.

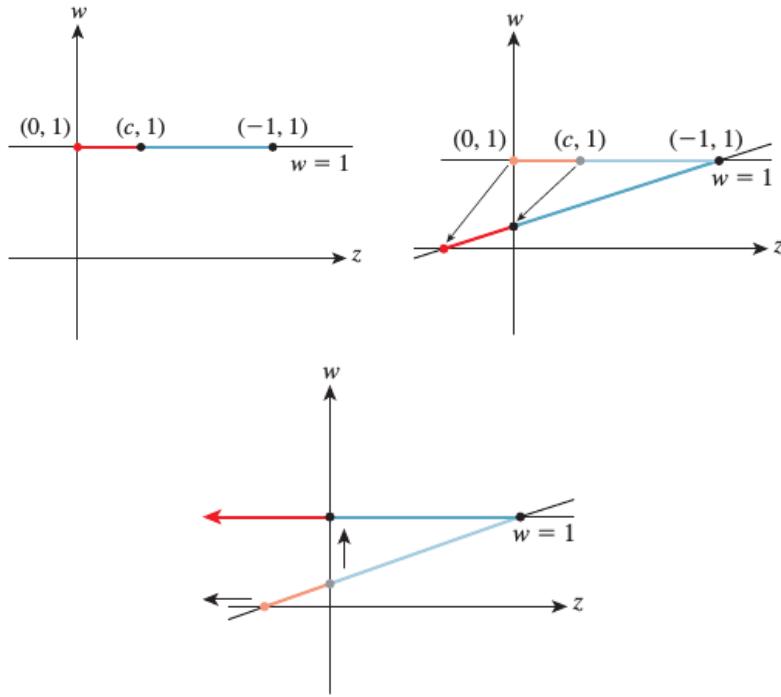
Razmotrimo npr. gornji desni prednji ugao zarubljene piramide. On ima koordinate  $(-c, -c, c)$  (podsetimo se da je  $c = -n/f$  negativno, pa je  $-c$  pozitivno). Nakon primene transformacije  $M_{pp}$  postaje:

$$M_{pp} = \begin{bmatrix} f-n & 0 & 0 & 0 \\ 0 & f-n & 0 & 0 \\ 0 & 0 & f & n \\ 0 & 0 & -(f-n) & 0 \end{bmatrix} \cdot \begin{bmatrix} -c \\ -c \\ c \\ 1 \end{bmatrix} = \begin{bmatrix} -c(f-n) \\ -c(f-n) \\ cf+n \\ -(f-n)c \end{bmatrix}$$

Nakon homogenizacije dobija se:

$$\begin{bmatrix} 1 \\ 1 \\ \frac{cf+n}{-(f-n)c} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}$$

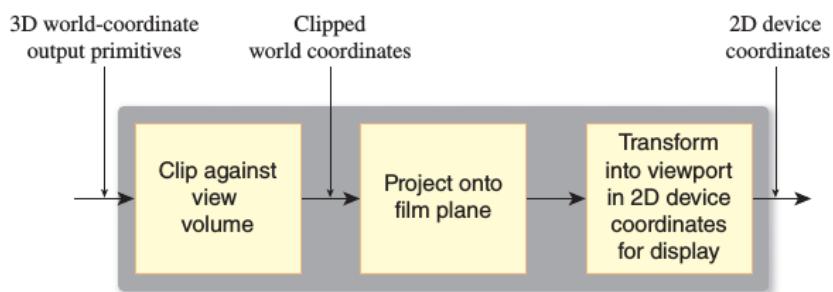
a to je gornji desni prednji ugao standardne paralelne zapremine pogleda.



Slika 5.14: a) Bočni pogled zarubljene piramide i zapremine pogleda u  $zw$  ravnji pre transformacije (plavi segment označava  $zw$  isečak zarubljene piramide između prednje i zadnje ravni odsecanja; crveni segment označava deo vidnog polja između oka i prednje ravni odsecanja), b) nakon primene transformacije  $M_{pp}$  (tačke na zadnjoj ravni odsecanja ( $z = -1$ ) ostaju neizmenjene, prednja ravan odsecanja se transformiše da pripada pravoj  $z = 0$ , a oko pripada  $w = 0$  pravoj) c) nakon homogenizacije prednja ravan odsecanja ostaje u  $z = 0$ , dok se pozicija oka šalje u beskonačnost u smeru  $z$  ose, čime prave koje se sekaju u oku postaju paralelne  $z$  osi. Ova tri ograničenja su dovoljna da bi se jedinstveno odredila matrica  $M_{pp}$

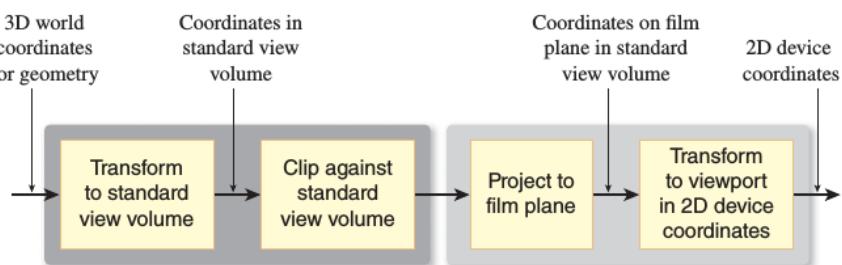
## 5.5 Proces renderovanja

Grafička obrada se uobičajeno izvodi na sledeći način: prvo se geometrijski modeli postavljaju na 3D scenu različitim geometrijskim transformacijama (kao u primeru robota koji se kreira na osnovu osnovnih geometrijskih tela jedinične veličine: lopte, kupe, valjka, ...). Nakon toga se ovi modeli posmatraju kroz kameru što rezultuje transformacijom njihovih svetskih koordinata u koordinate u standardnoj perspektivnoj zapremini pogleda, a nakon toga se transformišu u koordinate u standardnoj paralelnoj zapremini pogleda. Konačno, ovi modeli se projektuju u 2D sliku i ova slika se transformiše u prozor za prikaz.



Slika 5.15: Obrada geometrijskih primitiva prilikom kreiranja slika

Pritom se geometrijske reprezentacije svih modela na sceni obrađuju na način prikazan na slici 5.15: 3D svetske koordinate primitiva (obično trouglovi) se odsecaju u odnosu na zapreminu pogleda; drugim rečima, one primitive koje su izvan zapremine pogleda se ne razmatraju. Trougao koji je delimično unutar a delimično van zapremine pogleda se odseca i deli na nove trouglove.



Slika 5.16: Transformacija u standardnu zapreminu pogleda pojednostavljuje odsecanje

Prikazane transformacije kamere omogućavaju da umesto da se vrši odsecanje svetskih koordinata primitiva u odnosu na zapreminu pogleda kamere, svetske koordinate se transformišu u standardnu zapreminu pogleda u kojoj se odsecanja izvode znatno jednostavnije. U standardnoj zapremini pogleda

odsecanje se svodi na odsecenja u odnosu na ravan  $z = -1$  ili jednostavne ravni kao što su recimo ravan  $x = z$  ili ravan  $y = -z$ . Projekcija na ravan filma u drugom koraku niza sa slike 5.15 nije više generička projekcija na ravan u 3D prostoru, već projekcija na standardnu ravan u standardnoj paralelnoj zapremini pogleda, što vodi ka tome da je potrebno samo zaboraviti  $z$  koordinatu (slika 5.16).

Transformacija standardne perspektivne zapremine pogleda u standardnu paralelnu zapreminu pogleda izvodi se množenjem matricom  $M_{pp}$ , ali je pre homogenizacije potrebno odseći objekte kod kojih je  $z < 0$ , zato što se objekat kod koga je  $z < 0$  i (ako istovremeno važi)  $w < 0$  nakon homogenizacije transformiše u objekat za koji važi  $z > 0$  i  $w = 1$ . U praksi ovo znači da bi se objekti iza kamere pojavili ispred nje i renderovali, što svakako ne želimo.

Nakon ove prve faze odsecanja može se izvršiti homogenizacija i izvršiti odsecanje u odnosu na  $x$  koordinatu i  $y$  koordinatu, kao i u odnosu na zadnju ravan odsecanja po  $z$ . Sve ove operacije su jednostavne jer podrazumevaju odsecanje u odnosu na ravan paralelnu koordinatim ravнима.

Dakle, kompletan niz operacija za dati trougao čije su svetske koordinate poznate bio imao sledeću formu:

1. pomnožiti koordinate temena trougla matricom  $M_{pp}M_{persp}$  čime se tačke transformišu prvo u standardnu perspektivnu zapreminu pogleda, a zatim u standardnu paralelnu zapreminu pogleda
2. izvršiti odsecanje da bi se eliminisale tačke kod kojih važi:  $z < 0$
3. izvršiti homogenizaciju koordinata tačaka  $(x, y, z, w) \rightarrow (x/w, y/w, z/w, 1)$  pri čemu se na  $w$  koordinatu nadalje može zaboraviti
4. izvršiti odsecanje u odnosu na ravni  $x = +/-1, y = +/-1, z = +/-1$
5. pomnožiti koordinate tačaka matricom  $M_{wind}$  čime se koordinate tačaka transformišu u koordinate piksela

## 5.6 Pitanja

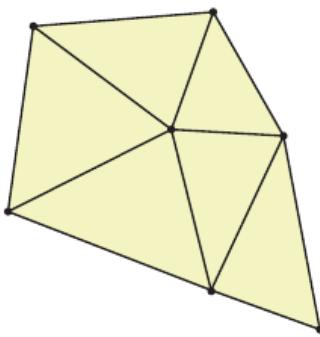
- 5.1 Nabrojati 6 parametara sintetičkog modela kamere.
- 5.2 Šta se zadaje vektorom pogleda?
- 5.3 Šta se zadaje uglovima vidnog polja?
- 5.4 Kakav je odnos uglova vidnog polja i stepena perspektivnog iskrivljenja na slici?
- 5.5 Da li vektor nagore mora biti upravan na vektor pogleda?
- 5.6 Čemu služe prednja i zadnja ravan odsecanja i zašto je bitno da ih definišemo?  
Kako se one zadaju?
- 5.7 Kojim se telom definiše zapremina pogleda kod paralelnog projektovanja?  
Koji se parametri menjaju u odnosu na perspektivno projektovanje?
- 5.8 Pokazati da i ovaj pristup daje isti  $uvw$  koordinatni sistem za zadati vektor pogleda i vektor nagore:  $w = \frac{-\text{look}}{\|\text{look}\|}, t = w \times vup, u = \frac{t}{\|t\|}, v = u \times w$ .

- 5.9 Kako izgleda standardna perspektivna zapremina pogleda?
- 5.10 Koja je jednačina prednje ravni odsecanje kod standardne perspektivne zapremine pogleda?
- 5.11 Kako izgleda standardna paralelna zapremina pogleda?
- 5.12 Zašto se vrši transformacija iz perspektivne u paralelnu zapreminu pogleda?
- 5.13 Iz kojih se koraka sastoji prevođenje 3D svetskih koordinata primitiva u 2D koordinate na uređaju za prikaz?

## Glava 6

# Načini opisivanja figura u 2D i 3D

Bavićemo se opisivanjem mreže trouglova kao najčešće korišćenom reprezentacijom figura u računarskoj grafici. **Mreža trouglova** se sastoји od velikog broja trouglova međusobno povezanih ivicama koji na taj način formiraju površ (slika 6.1).



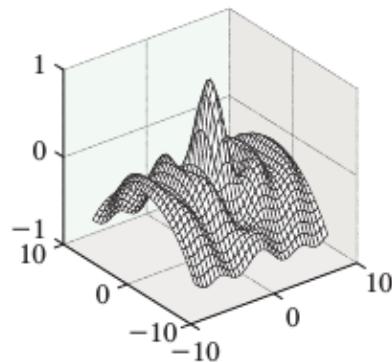
Slika 6.1: Mreža trouglova sačinjena od temena, ivica i trougaonih strana

Ponekad se koriste i mreže četvorouglova ili drugih poligona (slika 6.2), ali postoje problemi pri njihovoј upotrebi. Na primer, moguće je napraviti četvorougao čija sva četiri temena ne leže u jednoj ravni, dok se ovo ne može desiti u slučaju trougla jer trougao uvek leži u jednoj ravni.

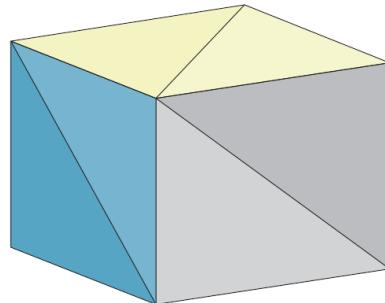
Nije teško uvideti kako napraviti određene figure korišćenjem mreža trouglova. Recimo, počev od nekog poliedra delimo njegove strane na trouglove. Na slici 6.3 prikazan je ovaj proces za kocku.

Neke figure nije lako predstaviti trouglovima, ali ih je moguće **aproksimirati** korišćenjem mreže trouglova. Jedan način da se ovo uradi jeste da se utvrde lokacije velikog broja tačaka na figuri i da se onda susedne lokacije povežu mrežnom strukturuom. Ovakva aproksimacija, ukoliko su tačke dovoljno blizu jedna drugoj, može dati privid glatke površi.

Jedna od najznačajnijih osobina mreže trouglova jeste njena **uniformnost**. Ona nam omogućava primenu različitih operacija sa garancijama koje je lako dokazati; takođe, pojednostavljuje isprobavanje jednostavnih ideja. Jedna od operacija koje možemo izvesti nad mrežom je **podela** (engl. subdivision) pri-



Slika 6.2: Korišćenje poligonalnih mreža za predstavljanje čajnika i glatke talasaste površi



Slika 6.3: Mreža trouglova koja ima geometriju kocke

kojoj se jedan trougao zamenjuje sa nekoliko manjih trouglova na prilično jednostavan način. Podela se obično koristi da bi se mreža koja ima oštре tačke ili ivice učinila glatkijom. Naravno, ponovljenim podelama značajno se povećava broj trouglova što može imati veliki uticaj na vreme renderovanja.

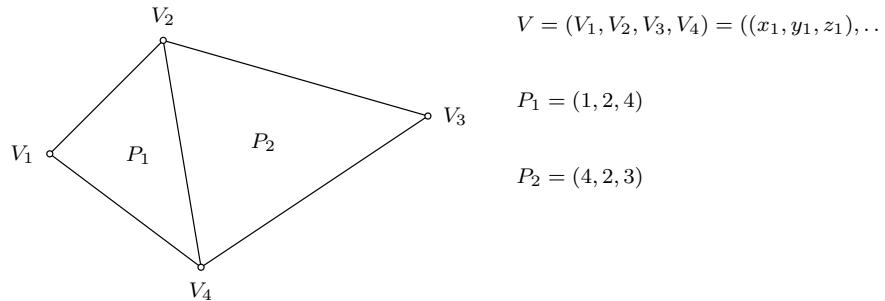
Još jedna važna operacija sa mrežom je **pojednostavljivanje** i njome se mreža zamenjuje drugom mrežom koja joj je slična (topološki ili geometrijski) ali koja ima kompaktniju strukturu. Ako se ova operacija iznova ponavlja možemo doći do sve jednostavnije reprezentacije iste površi.

Mreže trouglova su u velikoj upotrebi i zbog toga što je geometrija trougla dobro izučena. Međutim nisu svi objekti na svetu pogodni za predstavljanje mrežama trouglova. Na primer, za neke oblike je karakteristično da imaju visok nivo detaljnosti u svakoj razmeri (recimo šljunak).

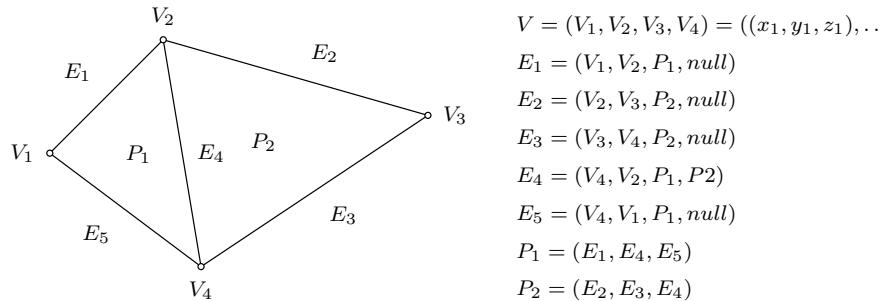
Postoji više različitih reprezentacija mreža poligona:

**eksplicitna reprezentacija:** svaki poligon je opisan nizom svojih temena; za jedan poligon, ovo rešenje je prostorno-efikasno; međutim, za više poligona ovo rešenje može da troši mnogo prostora jer veliki broj temena može da pripada raznim poligonima iz skupa. Dodatno, veliki problem u ovoj reprezentaciji je i to što ne postoji eksplicitna informacija o zajedničkim temenima i ivicama. Da bi se proverilo da li je jedno teme istovremeno teme nekog drugog poligona treba proveriti sva njegova temena. Štaviše, ta provera može da bude nepouzdana jer zbog grešaka u računu dve reprezentacije (u dva poligona) jedne iste tačke mogu da se razlikuju. Prilikom iscrtavanja mreže poligona svaka ivica će biti (nepotrebno) iscrtavana dva puta.

**reprezentacija sa pokazivačima na liste indeksa temena:** svaka tačka (svako teme) je zapisana tačno jednom u listi temena. Poligon je reprezentovan kao lista indeksa temena. Ova reprezentacija štedi prostor. Dodatno, ona omogućava lako ispitivanje da li dva poligona imaju zajedničko teme. S druge strane, i u ovoj reprezentaciji se svaka ivica crta dva puta.



**reprezentacija sa pokazivačima na liste indeksa ivica:** Alternativno, poligoni mogu biti reprezentovani kao liste indeksa ivica, dok je svaka ivica opisana parom tačaka. Dodatno, u opisu ivice mogu da budu i poligoni kojima pripada (radi efikasnijih obrada).



Svaka od navedenih reprezentacija ima prednosti i mane i izbor treba napraviti u skladu sa konkretnim zadatkom

## 6.1 "Mreže" u 2D

Analogon mreža trouglova u prostoru, u slučaju za jedan manje dimenzije, je kolekcija duži u ravni. Ovakve mreže zvaćemo **jednodimenzionim (1D) mrežama**.

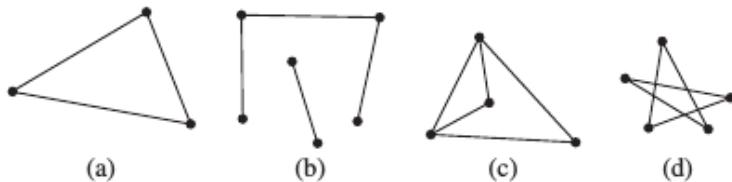
1D mreža se sastoji od *temena* i *ivica* koje povezuju temena. Ovakvu strukturu možemo opisati korišćenjem dve liste:

- liste temena i njihovih lokacija (tačke u ravni)
- liste ivica od kojih je svaka zadata *uređenim* parom indeksa temena

Na narednoj slici date su tabele kojima je opisana jedna jednostavna 1D mreža:

Vertices		Edges	
1	(0, 0)	1	(1, 2)
2	(0.5, 0)	2	(2, 3)
3	(1.5, 1)	3	(3, 4)
4	(0, 2.0)	4	(4, 1)
5	(3, 0)	5	(5, 6)
6	(4, 0)		

Ovakva struktura podataka ima zanimljivo svojstvo: u tabeli ivica mreže data je *topologija* mreže (koje ivice su međusobno susedne), dok je u tabeli sa temenima data njena njena *geometrija*.



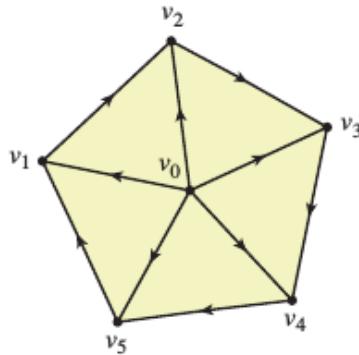
Slika 6.4: 1D mreža koja se sastoji od skupa temena i pravolinijskih ivica između njih. Najčešće nas interesuju one kod kojih svako teme pripada jednoj ili dvema ivicama (slika (a) i (b)). Postoje, takođe, i drugačije mreže kod kojih jedno teme može da pripada većem broju ivica (slika (c)) i one kod kojih se ivice presecaju u tačkama koje nisu u skupu temena (slika (d))

**Granica** 1D mreže se definiše kao težinska suma temena mreže pri čemu se koeficijent uz svako teme određuje na sledeći način: ivica od temena  $i$  do temena  $j$  ima koeficijent +1 uz teme  $j$ , a -1 uz teme  $i$ . Često kažemo da je granica ivice  $ij$  jednaka  $j - i$ . Na primer, za mrežu koja je data prethodnom tabelom, granica ima vrednost:

$$(v_2 - v_1) + (v_3 - v_2) + (v_4 - v_3) + (v_1 - v_4) + (v_6 - v_5) = v_6 - v_5$$

Neformalno, kažemo da se ivica sastoji od temena 5 i 6. Motivaciju za uvođenje pojma granice prestavlja situacija kada razmatramo neke interesantnije mreže,

kao na primer onu sa slike 6.5. Njena granica ima vrednost  $v_1 + v_2 + v_3 + v_4 + v_5 - 5v_0$ .



Slika 6.5: Strelice prikazane na dotoj mreži, usmerene od temena  $i$  do temena  $j$ , ukazuju na to da je  $(i, j)$  ivica mreže, a ne  $(j, i)$ .

Za 1D mrežu čija je vrednost granice nula važi da je jednostavno definisati njenu unutrašnjost i spoljašnjost. Ovakvu mrežu nazivamo **zatvorenom**.

1D mrežu kod koje je stepen svakog temena 2 nazivamo **mrežom mnogostruktosti** (engl. manifold mesh). Motivacija za uvođenje ovakvog termina potiče iz topologije gde važi da kod 1D mnogostruktosti svaka tačka ima okolinu koja izgleda kao segment prave. Sa ovakvim mrežama je jednostavnije raditi. Nije nužno da ovakva mreža ima samo jednu komponentu povezanosti.

Mreže u kojima je svaka ivica predstavljena *uređenim parom* nazivamo **orientisanim mrežama**. Ako bismo ivice definisali kao neuređene parove, dobili bismo **neorientisanu mrežu** i za nju pojam granice ne bi imao smisla.

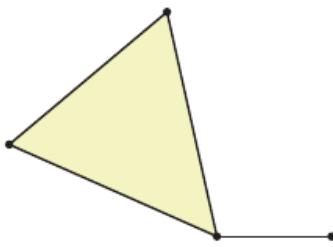
Struktura podataka koja se koristi za 1D mreže (a nalik je onoj koja se koristi za 2D mreže) sastoji se od:

- tabele sa temenima (svaki element tabele sadrži indeks temena i njegove koordinate)
- tabele sa ivicama (svaki element tabele sadrži uređen par temena)
- tabele suseda (ova tabela za svako teme sadrži uređene ciklične liste ivica koje se sutiču u tom temenu)

U ovakvoj reprezentaciji dodavanje novog temena je složenosti  $O(1)$  – jedino je potrebno dodati ga u tabelu sa temenima. Dodavanje nove ivice  $(i, j)$  je u najgorem slučaju  $O(e)$ , gde je  $e$  broj ivica u tabeli, jer je pored dodavanja ivice u tabelu ivica (koje je složenosti  $O(1)$ ) potrebno dodati je tabeli suseda i za teme  $i$  i za teme  $j$  (potrebno je dodati je na pravo mesto u cikličnom poretku ivica oko temena  $i$ , odnosno  $j$ ). Korišćenjem ove reprezentacije jednostavno je dobiti ivice koje su susedne nekom temenu, dobiti drugi kraj ivice koja počinje iz datog temena, kao i naći oba temena date ivice. Što se brisanja ivice tiče, u najgorem slučaju je složenosti  $O(e)$ .

## 6.2 Mreže u 3D

Analogno situaciji u 2D, da bismo zadali mrežu u 3D potrebno je nabrojati temena i trouglove mreže. A šta je sa ivicama? Ako temena  $i$ ,  $j$  i  $k$  formiraju trougao, tada se ivice  $(i, j)$ ,  $(j, k)$  i  $(k, i)$  smatraju delom strukture mreže. Ovo nam daje za posledicu da ne možemo imati ivicu koja "visi" (kao na slici 6.6), dok su izolovana temena i dalje dozvoljena.



Slika 6.6: Trougao sa dodatnom ivicom koja "visi" ne može biti deo strukture mreže

Kao i u slučaju 1D mreža, dodavanje novih temena i trouglova se efikasno izvršava, ali je brisanje temena sporo (jer je potrebno naći sve pridružene trouglove i obrisati ih). Ako čuvamo i listu susednih trouglova za svako od temena (koja je neuređena) onda je cena dodavanja mala, ali je cena brisanja visoka.

Korišćenjem ovog pristupa ne možemo dodati novu ivicu u 2D mrežu, ali se možemo pitati da li je neka ivica – ivica mreže. Odgovor na ovo pitanje bio bi složenosti  $O(T)$ , gde je  $T$  broj trouglova, jer je potrebno uraditi iscrpnu pretragu svih trouglova u mreži. Ipak, u nekim specijalnim slučajevima ove operacije se mogu učiniti efikasnijim.

Konačna 2D mreža je **mreža mnogostrukosti** ako se ivice i trouglovi čije je jedno teme  $v$  mogu poređati u cikličnom poretku:  $t_1, e_1, t_2, e_2, \dots, t_n, e_n$  bez ponavljanja tako da je ivica  $e_i$  stranica trouglova  $t_i$  i  $t_{i+1}$ . Ovo nam daje da za svaku ivicu postoje tačno dve strane kojima pripada. U topologiji važi da kod 2D mnogostrukosti svaka tačka ima okolinu koja izgleda kao disk.

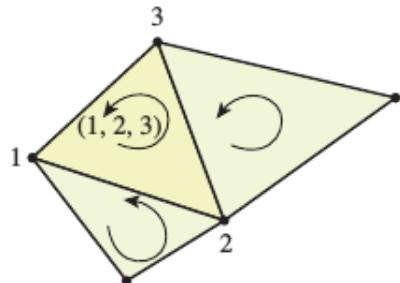
Ovakvu mrežu možemo čuvati u strukturi nalik onoj opisanoj za 1D mreže (sastoji se od tabele sa temenima, tabele sa trouglovima i tabele susedstva). Lista suseda temena sa indeksom  $i$  sastoji se od trouglova koji sadrže teme sa indeksom  $i$ .

Mreže mnogostrukosti nažalost ne dozvoljavaju dodavanja i brisanja trouglova: svako dodavanje ili brisanje trougla bi pokvarilo svojstva ove mreže.

Često će nam biti od značaja da vodimo računa o **orientaciji** trouglova u mreži (na taj način trouglove (1,2,3) i (2,1,3) posmatramo kao različite). Jedan način da zadamo orientaciju jeste određivanjem vektora normale: ako su lokacije temena trougla  $P_i$ ,  $P_j$  i  $P_k$  onda računamo vektor  $(P_j - P_i) \times (P_k - P_i)$  koji je upravan na ravan kojoj pripada trougao.

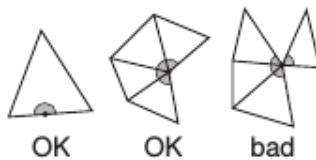
Ako dva susedna trougla imaju konzistentno orijentisane vektore normala, onda će se ivica koju dele javiti kao  $(i, j)$  u jednom trouglu, a kao  $(j, i)$  u drugom. Ako mreža mnogostrukosti može da orijentiše svoje trouglove tako da

se svaka ivica javlja po jednom u svakom od smerova, onda se mogu zadati i konzistentno orijentisani vektori normala.



Slika 6.7: Dva susedna trougla mreže sa konzistentnim vektorima normale (tj. strelice vektora normale su sa iste strane mreže). Primetimo da se ivica  $(i, j)$  javlja kao ivica jednog trougla, a ivica  $(j, i)$  kao ivica drugog. U opštem slučaju, u konzistentno orijentisanoj mreži, svaka ivica se javlja dva puta u različitim smerovima

Mreže mnogostrukosti su pogodne za rad, ali nekada je potrebno dozvoliti da mreže imaju granicu. Ovakve mreže nisu mreže mnogostrukosti jer tačka na granici ima okolinu koja je sa jedne strane "odsečena". Dakle, to su mreže kod kojih umesto da imamo susedne trouglove koji obrazuju ciklus, oni obrazuju lanac čiji prvi i poslednji element dele samo jednu svoju stranicu sa ostalim trouglovima u lancu. Druga stranica prvog trougla koji se sustiče u temenu se sadrži samo u prvom trouglu i ni u jednom drugom trouglu mreže (slično važi i za poslednji trougao). Ova nedeljena ivica se naziva **graničnom ivicom** a teme **graničnim temenom**. Temena koja nisu granična nazivamo **unutrašnjim temenima**. Za ovakve mreže u stvari važi da:

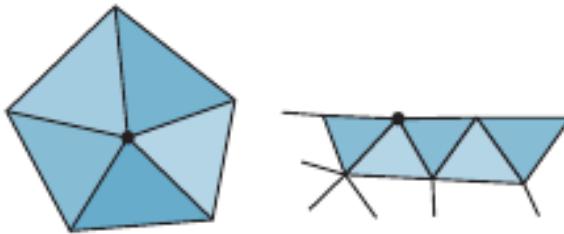


Slika 6.8: Uslovi za ivicu mreže mnogostrukosti sa granicom.

- svaka ivica je sadržana u jednom ili dva trougla
- svako teme je sadržano u jednom skupu trouglova povezanih stranicama

Slično definisanju granice neke ivice  $ij$  kao sume  $v_j - v_i$  možemo definisati i granicu trougla u mreži sa temenima  $i, j$  i  $k$  kao sumu ivica

$$(i, j) + (j, k) + (k, i)$$



Slika 6.9: Razlikujemo *teme mnogostrukosti*, prikazano na levoj strani slike, koje sadrži ciklus trouglova oko sebe i *granično teme* koje ima lanac trouglova oko sebe – prvi i poslednji trougao dele samo jednu svoju ivicu sa drugim trouglovima u mreži.

Ako uvedemo da je  $(i, j) = -1(j, i)$ , granica se može zapisati kao:

$$(i, j) + (j, k) - (i, k)$$

Granicu kolekcije orijentisanih trouglova možemo definisati kao sumu njihovih granica. Za orijentisane mreže mnogostrukosti ova granica imaće vrednost 0 jer ako je  $(i, j)$  deo granice jedne strane,  $(j, i) = -(i, j)$  je deo granice druge strane.

Za orijentisane mreže mnogostrukosti sa granicom, granica će se sastojati od tačno onih ivica koje smo identifikovali kao granične ivice. U opštem slučaju, orijentisanu mrežu mnogostrukosti bez graničnih ivica nazivamo **zatvorenom**.

### 6.2.1 Winged-edge struktura podataka

U slučajevima mreža kod kojih strane ne moraju biti trouglovi, može se koristiti drugačija struktura podataka, tzv. **winged-edge** struktura podataka. Ovaj pristup ivice posmatra kao “građane prvog reda”. Naime, za svaku ivicu čuvaju se pokazivači na dva temena koje ta ivica povezuje, na dve strane čiji je ona deo i, ono što je najznačajnije, na prethodnu i sledeću ivicu prilikom obilaska u smeru obrnutom od smera kazaljke na časovniku njegove leve i desne strane (slika 6.10). Svako teme i strana, takođe, sadrže pokazivač na jednu, proizvoljnu ivicu, koja im pripada.

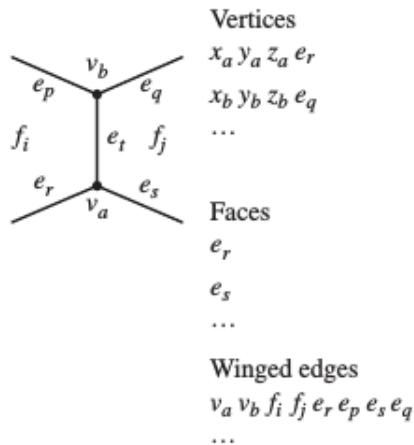
U ovoj reprezentaciji računanje susedstva je jednostavno, ali su operacije umetanja i brisanja zahtevne.

### 6.2.2 Ojlerova formula

Za proveru da li je zatvorena površ dobro definisana koristi se Ojlerova formula. Ojlerova formula za površ koja nema “rupu” glasi:

$$V - E + F = 2$$

gde je  $V$  broj temena,  $E$  broj ivica, a  $F$  broj strana površi.



Slika 6.10: Winged-edge struktura podataka čuva pokazivače na prethodnika i sledbenika za strane na strane svake ivice

Primer površi koja nema "rupe" bila bi kocka, dok bi torus bio primer površi sa "rupom". Ukoliko površ ima "rupe" formula glasi:

$$V - E + F = 2 - 2G$$

gde je sa  $G$  označen broj rupa.

U slučaju mreže trouglova, ako prepostavimo da je svako teme deo nekog trougla i da je mreža zatvorena, ovo se može pojednostaviti: svaki trougao ima 3 ivice i svaku ivicu dele dva trougla. Stoga je broj ivica  $E = 3/2T$ . Stoga važi:

$$V - 3/2T + T = 2 - 2G$$

a ovo se svodi na:

$$V - 1/2T = 2 - 2G$$

### 6.2.3 Memorijski zahtevi za predstavljanje mreže

Prepostavimo da cele i realne brojeve predstavljamo sa po 4 bajta. Ako mrežu predstavimo korišćenjem tabele temena i tabele trouglova onda je potrebno  $12V$  bajtova za  $V$  temena (teme je predstavljeno sa po 3 koordinate) i  $12T$  bajtova za  $T$  trouglova (svaki trougao se predstavlja sa tri celobrojna indeksa). U kom su odnosu broj temena i broj trouglova? Kao što smo već pomenuli, za mreže koje predstavljaju zatvorene površi, na osnovu Ojlerove formule važi:

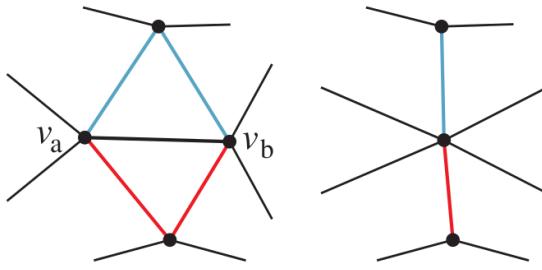
$$V - 1/2T = 2 - 2G$$

gde je  $V$  broj temena,  $T$  - broj trouglova a  $G$  broj "rupa".

Za površi sa malim brojem rupa koje su predstavljene velikim mrežama desna strana jednakosti je bliska nuli, te važi da je broj trouglova približno jednak dvostrukom broju temena. Ovim dobijamo da je ukupna memorija potrebna za ovu reprezentaciju približno  $18T$  bajtova.

### 6.2.4 Operacije nad mrežom

Jedna od prednosti mreža trouglova je što njihova homogenost čini određene operacije jednostavnim za izvođenje. Prilikom pojednostavljivanja mreže jedna od standardnih operacija je **sažimanje ivice** (engl. edge collapse), kojom se jedna ivica sažima sve dok njena dužina ne postane 0 čime dva susedna trougla nestaju.



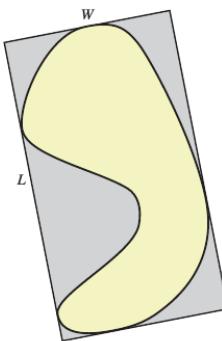
Prilikom stapanja dva temena treba izabrati lokaciju za novo teme. Lokacija zavisi od željenog cilja: ako želimo da minimizujemo račun, onda možemo izabrati jedno od dva stara temena; ukoliko želimo da očuvamo neki oblik, možemo da izaberemo središte duži određene ovim temenima; ako ovakvo usrednjavanje pomera puno tačaka a to nije vizuelno poželjno, možemo da izaberemo novu tačku tako da minimizujemo maksimalno rastojanje tačaka mreže od najbliže nove tačke mreže.



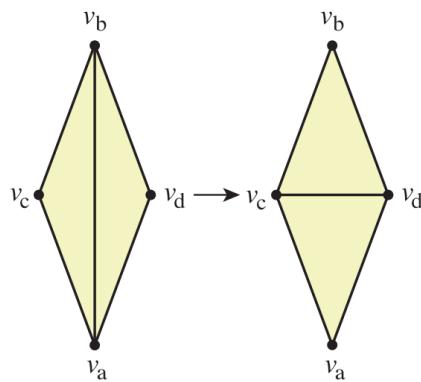
Slika 6.11: Različiti geometrijski izbori za sažimanje ivice u 2D

Nekada se mreža može deformisati u smislu da pojedinačni trouglovi postanu jako dugi i uski. Kažemo da ovakvi trouglovi imaju loš **aspect ratio**. U opštem slučaju, aspect ratio se može definisati pronalaženjem onog pravougaonika među svim pravougaonicima u kojima je objekat sadržan i koji dodiruju sa sve 4 stranice objekat čiji je odnos visine i širine najveći.

U ulepšavanju mreže (kojim pokušavamo da postignemo da se mreža sastoji od trouglova bliskih jednakostaničnim trouglovima i da se postignu još neka lepa svojstva) operacija **zamene ivice** (engl. edge swap) pomaže da se dva dugačka i uska trougla transformišu u dva skoro jednakostanična trougla.



Slika 6.12: Računanje aspect ratio-a



Obe ove operacije zahtevaju minimalne izmene same strukture podataka.

### 6.3 Pitanja

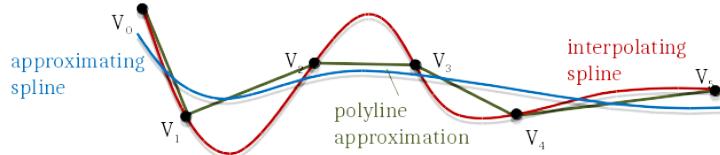
- 6.1 Zašto se prilikom modelovanja figura pomoću mreža za primitive najčešće biraju trouglovi, a ne poligoni sa većim brojem stranica?
- 6.2 Kako predstavljamo mrežu u 2D?
- 6.3 Kako se definiše granica 2D mreže?
- 6.4 Kada kažemo da je 1D mreža zatvorena?
- 6.5 Kada za neku 1D mrežu kažemo da je mreža mnogostruktosti?
- 6.6 Koje je složenosti operacija dodavanja novog temena u 1D mrežu, a koje složenosti operacija dodavanja nove ivice?
- 6.7 Da li je kod 1D mreža mnogostruktosti dozvoljeno dodati novu ivicu u mrežu?
- 6.8 Koja se reprezentacija koristi za predstavljanje mreže u 3D?

- 6.9 Kada za 2D mrežu kažemo da je mreža mnogostrukosti? Da li je kod njih dozvoljeno dodati novi trougao u mrežu? Da li je dozvoljeno obrisati neki trougao iz mreže?
- 6.10 Za koju mrežu kažemo da je mreža mnogostrukosti sa granicom?
- 6.11 Čemu je jednaka vrednost granice kod (a) orijentisanih mreža mnogostrukosti bez granice (b) orijentisanih mreža mnogostrukosti sa granicom?
- 6.12 Opisati winged edge strukturu podataka. Za šta se ona koristi?
- 6.13 Kako glasi Ojlerova formula (a) za površi bez rupa (b) za površi sa rupama?
- 6.14 Opisati operacije sažimanja ivice i zamene ivice.

## Glava 7

# Opisivanje krivih i površi u 3D

Svaku 1D mrežu možemo predstaviti skupom temena i ivica. Međutim, glatke krive i površi su neophodne u mnogim primenama računarske grafike. Kako njih predstaviti? Ne želimo da ih čuvamo u rasterskom obliku, već nam je potrebna efikasnija matematička reprezentacija. Ideja je da pamtimo listu kontrolnih tačaka i da nađemo neki način da glatko interpoliramo između njih. Deo po deo linearna aproksimacija nije glatka i ukoliko nemamo baš puno kontrolnih tačaka izgleda ružno. Korišćenje trigonometrijskih funkcija je skupo, a njima se teško manipuliše. Polinomi višeg reda se relativno jeftino računaju i njima se jednostavno manipuliše.



Najčešće se koristi polinomijalna interpolacija, pri čemu su polinomi trećeg stepena. Za parametarske krive, 3 je najmanji stepen takav da kriva ne mora nužno da pripada ravni. Zaista, kriva stepena 2 je potpuno određena trima tačkama, a te tri tačke određuju ravan (kojoj pripada ta kriva).

Splajnovi su parametarske krive vođene kontrolnim tačkama ili kontrolnim vektorima, reda tri ili više.

Krive se mogu opisati na jedan od sledećih načina:

**eksplicitna reprezentacija:**  $y = f(x), z = g(x)$

potencijalni problem: za jednu vrednost  $x$  postoji samo jedna tačka na krivoj, te je u nekim slučajevima potrebno kombinovati više krivih

**implicitna reprezentacija:**  $f(x, y, z) = 0$

potencijalni problem: ovakva reprezentacija može da dâ više rešenja nego što nam treba; na primer, za modelovanje kruga možemo da koristimo jednačinu  $x^2 + y^2 = 1$ , ali kako modelovati polukrug? Potrebno je koristiti dodatne uslove, što može stvoriti druge probleme.

**parametarska reprezentacija:**  $x = x(t), y = y(t), z = z(t), 0 \leq t \leq 1$

ova reprezentacija nema većinu problema koje imaju prethodne dve. Za funkcije  $x$ ,  $y$  i  $z$  se često koriste kubni polinomi. Time se zadata kriva aproksimira deo po deo polinomijalnim krivim.

## 7.1 Osnovne polinomske krive

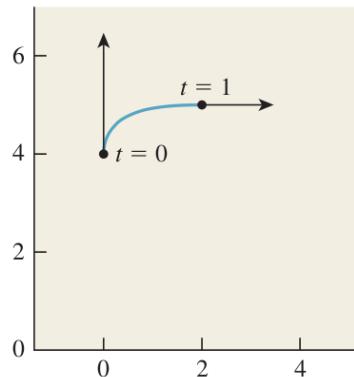
Postoje dva osnovna načina za zadavanje krive koja su u širokoj upotrebi. Ona odgovaraju tome kako se može zadati duž:

- možemo zadati krajnje tačke  $P$  i  $Q$ ;
- možemo zadati jednu krajnju tačku  $P$  i vektor  $v$  ka drugoj krajnjoj tački (pa je stoga  $Q = P + v$ ).

Mi ćemo se u nastavku teksta ograničiti na kubne krive.

### 7.1.1 Hermitova kriva

**Primer 1:** Neka se automobil kreće duž  $y$  ose sa vektorom brzine  $[0, 3]^T$  i stiže u tačku  $(0, 4)$  u trenutku  $t = 0$ . Potrebno je modelovati kretanje automobila: kako skreće i usporava tako da je u trenutku  $t = 1$  u poziciji  $(2, 5)$  sa vektorom brzine  $(2, 0)$  (videti sliku dole).



Potreban nam je način da "zlepimo" dva dela putanja automobila da bismo dobili glatko kretanje. Tražimo fini način da povežemo deo putovanja duž  $y$ -ose sa delom putanje duž prave  $y = 5$ . Nakon toga možemo u svakom momentu rotirati automobil da ga poravnamo sa tangentom interpolacionog puta.

Najpre ćemo uopštiti problem. Za date pozicije  $P$  i  $Q$  i vektore brzina  $v$  i  $w$  odrediti funkciju  $\gamma : [0, 1] \rightarrow \mathbb{R}^2$  tako da je  $\gamma(0) = P$ ,  $\gamma(1) = Q$ ,  $\gamma'(0) = v$  i  $\gamma'(1) = w$ . Rešenje ima sledeći oblik:

$$\begin{aligned}\gamma(t) &= (2t^3 - 3t^2 + 1)P + (-2t^3 + 3t^2)Q + (t^3 - 2t^2 + t)v + (t^3 - t^2)w \\ &= (1-t)^2(2t+1)P + t^2(-2t+3)Q + t(t-1)^2v + t^2(t-1)w.\end{aligned}$$

Rešenje se dobija prostom zamenom gornja četiri uslova u opšti oblik kubnog polinoma  $\gamma(t) = a_3t^3 + a_2t^2 + a_1t + a_0$ .

Rezultujuća kriva naziva se **Hermitova kriva** za ulaz  $P, Q, v$  i  $w$ . Četiri polinoma u gornjoj jednačini nazivaju se **Hermitove bazne funkcije**.

U Hermitovoj bazi, ako hoćemo da promenimo početnu tačku, potrebno je samo izmeniti koeficijent prvog polinoma; na taj način neće biti izmenjena početna brzina, krajnja tačka, niti krajnja brzina. Ako bismo, umesto na ovaj način, krivu izrazili kao linearu kombinaciju funkcija  $\{t^3, t^2, t, 1\}$  izmena rešenja tako da se promeni samo početna tačka rezultovala bi izmenom svih koeficijenata. Stoga, za ovaj problem, baza koja se sastoji od stepena  $t$  bila bi loš izbor, dok Hermitova baza predstavlja dobar izbor.

Sve funkcije Hermitove baze su kubni polinomi. Polinom  $a_0 + a_1t + a_2t^2 + a_3t^3$  možemo zapisati korišćenjem množenja matrica kao:

$$a_0 + a_1t + a_2t^2 + a_3t^3 = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} 1 \\ t \\ t^2 \\ t^3 \end{bmatrix}$$

Ako sa  $\mathbf{T}(t)$  označimo vektor koji sadrži stepene po  $t$ , tj.  $\mathbf{T}(t) = [1 \ t \ t^2 \ t^3]^T$  možemo zapisati:

$$\gamma(t) = [P; Q; v; w] \cdot \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \cdot \mathbf{T}(t)$$

Prvi činilac jeste matrica koju nazivamo **geometrijskom matricom** krive i označavamo sa  $G$ . Njene kolone su koordinate od  $P, Q, v$  i  $w$ . Drugi činilac je tzv. **bazna matrica** koju označavamo sa  $M$  i koja sadrži koeficijente polinoma za Hermitovu krivu, od najnižeg do najvišeg stepena. Suštinski ona predstavlja *promenu baze* za kubne polinome koja se sastoji od četiri Hermitova polinoma u bazu  $\{1, t, t^2, t^3\}$ .

Stoga se skaćeno Hermitova kriva može zapisati kao:

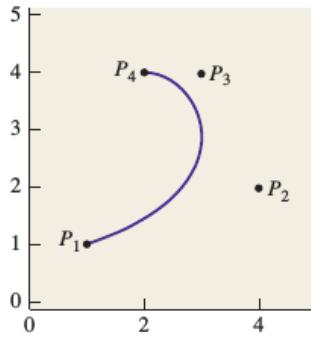
$$\gamma(t) = GMT(t)$$

### 7.1.2 Bezijerove krive

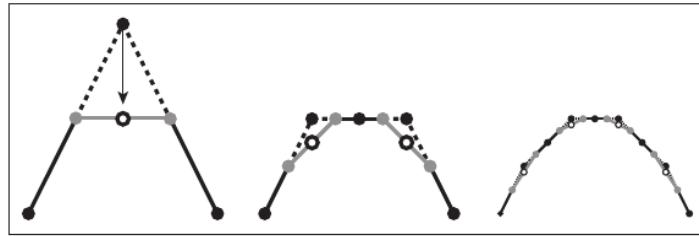
Drugi tip krive koji razmatramo jeste **Bezijerova kriva**. Ona se gradi od četiri tačke  $P_1, P_2, P_3, P_4$ . Kriva počinje u tački  $P_1$ , završava se u tački  $P_4$  i ima inicijalnu brzinu  $3(P_2 - P_1)$  i završnu brzinu  $3(P_4 - P_3)$ .

Intuicija iza ovoga je sledeća: imamo skup kontrolnih tačaka nad kojima želimo da kreiramo glatku krivu. Jednostavno povezivanje ovih tačaka dužima vodi neglatkoj funkciji (imaće oštре uglove). Možemo zamisliti "izglađivanje" ovog poligona isecanjem oštredih uglova, čime se dobija novi poligon koji je glatkiji ali i dalje ne dovoljno glatka funkcija. Ovaj postupak možemo ponavljati iznova i svaki put bismo dobili sve glatkiji poligon. Ako bismo ovo radili beskonačno mnogo puta dobili bismo krivu koja je iz klase  $C^1$ .

Bezijerova kriva data je jednačinom:



Slika 7.1: Bezijerova kriva počinje u tački  $P_1$ , ide ka tački  $P_2$ , a završava se u tački  $P_4$  dolazeći iz pravca tačke  $P_3$



Slika 7.2: Postupak podele za kvadratnu Bezijerovu krivu

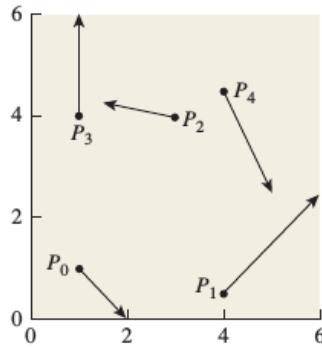
$$\gamma(t) = [P_1; P_2; P_3; P_4] \cdot \begin{bmatrix} 1 & -3 & 3 & -1 \\ 0 & 3 & -6 & 3 \\ 0 & 1 & 3 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \mathbf{T}(t)$$

te kao što vidimo ovoga puta geometrijska matrica sadrži 4 tačke, a bazna matrica sadrži drugačije koeficijente. Može se učiniti da je specifikacija Bezijerove krive manje prirodna od Hermitove (obzirom da je uloga tačaka  $P_2$  i  $P_3$  malo nejasna u poređenju sa pojmom tangente u početnoj i krajnjoj tački). Prednost Bezijerove krive je u tome što su svi zadati podaci tačke, pa ako želimo da transformišemo Bezijerovu krivu treba samo transformisati tačke (drugačije je transformisati vektor).

## 7.2 Nadovezivanje krivih i Catmull-Rom splajn

- Za nadovezivanje krivih jednu na drugu možemo se rukovoditi sledećim kriterijumima:
  - ograničenja nad krajnjim tačkama;
  - ograničenja nad tangentnim vektorima (tj. izvodima);
  - ograničenja vezana za neprekidnost u krajnjim tačkama.
- Kažemo da kriva pripada klasi  $C^0$  ako je neprekidna. Kažemo da kriva pripada klasi  $C^1$  ako je diferencijabilna i njen izvod je neprekidan (i

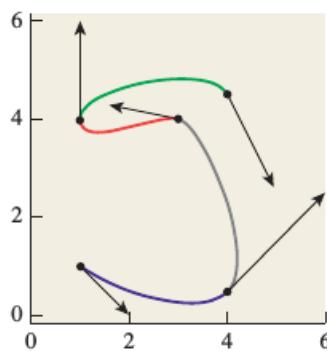
kažemo da je neprekidno diferencijabilna). Kažemo da kriva pripada klasi  $C^2$  ako je dvaput diferencijabilna i njeni prvi i drugi izvod su neprekidni. Analogno se definiše klasa  $C^n$ .



Slika 7.3: Dat je niz tačaka i vektora; želimo da nađemo krivu koja prolazi kroz date tačke sa datim vektorima kao vektorima brzine

Prepostavimo da imamo niz tačaka  $P_0, P_1, \dots, P_n$  i pridruženih vektora  $v_0, v_1, \dots, v_n$  i želimo da nadjemo krivu  $\gamma : [1, n] \rightarrow \mathbb{R}^2$  koja prolazi kroz date tačke sa datim vektorima brzina. Možemo svakako iskoristiti Hermitovu formulaciju da nadjemo krivu  $\gamma_0 : [0, 1] \rightarrow \mathbb{R}^2$  koja počinje u  $P_0$ , završava se u  $P_1$  i ima početnu i završnu tangentu  $v_0$  i  $v_1$ . Možemo, takođe, odrediti krivu  $\gamma_1 : [0, 1] \rightarrow \mathbb{R}^2$  koja počinje u tački  $P_1$ , završava se u  $P_2$  i ima brzine  $v_1$  i  $v_2$  i slično pronaći krive  $\gamma_2, \dots, \gamma_{n-1}$ .

Rezultat sastavljen od krivih  $\gamma$  je neprekidna diferencijabilna kriva koja prolazi kroz svaku tačku sa odgovarajućom tangentom. Pojedinačni delovi  $t_i \rightarrow \gamma_i(t - i)$  se nazivaju **segmentima krive**, kompletne kolekcije **splajn**, a date tačke i vektori **kontrolnim podacima**.

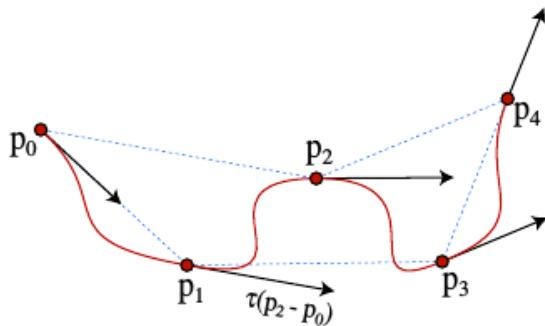


Slika 7.4: Kolekcija segmenata krive koja formira krivu kao rešenje problema

**Catmull-Rom splajn** je primer krive definisane nizom kontrolnih tačaka. On predstavlja rešenje narednog problema: za dati niz tačaka  $P_0, \dots, P_n$  pronaći glatku krivu koja prolazi kroz tačku  $P_i$  u trenutku  $t = i$ , sa svojstvom da ako

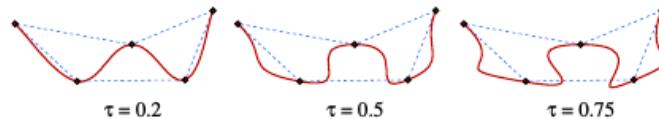
su tačke na istom rastojanju rezultujuća kriva je samo prava linija interpolacije od prve do poslednje tačke.

Ideja je jednostavna: ako bismo mogli da izaberemo tangentu u svakom  $P_i$  mogli bismo da koristimo Hermitove krive kao i do sad. Ideja ovog pristupa je da se koriste prethodna i naredna kontrolna tačka za navođenje: treba izabrati tangentni vektor u  $P_i$  da bude u smeru  $P_{i+1} - P_{i-1}$  od prethodne do naredne kontrolne tačke.



Slika 7.5: Primer Catmull-Rom splajna

Ovaj vektor se najčešće skalira nekom vrednošću  $\tau \in [0, 1]$  koju nazivamo **tenzijom splajna**. Ovaj parametar splajna određuje koliko oštro se kriva savija u kontrolnim tačkama (slika 7.6). Nadalje ćemo koristiti vrednost  $\tau = 1/3$ , odnosno tangentni vektor:  $v_i = \frac{1}{3}(P_{i+1} - P_{i-1})$ .



Slika 7.6: Efekat odabira različitih vrednosti za tenziju splajna.

Dakle, možemo definisati krivu na segmentu  $P_i P_{i+1}$  tako što zamenimo gornje vrednosti vektora tangenti u jednačinu Hermitove krive i dobijamo:

$$\gamma_i(t) = \left[ P_i; \quad P_{i+1}; \quad \frac{P_{i+1}-P_{i-1}}{3}; \quad \frac{P_{i+2}-P_i}{3} \right] \cdot \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \cdot \mathbf{T}(t)$$

odakle bismo dobili vrednost  $\gamma_i(t)$  kao funkciju tačaka  $P_{i-1}, P_i, P_{i+1}, P_{i+2}$ .

Za  $P_0$  koristimo  $v_o = \frac{2}{3}(P_1 - P_0)$  (što bismo dobili ako bi postojala i tačka  $P_{-1}$  simetrično tački  $P_1$  u odnosu na  $P_0$ ). Slično za  $P_n$ .

Pojedinačne funkcije su beskonačno diferencijabilne u većini tačaka (jer su definisane polinomima), ali u zajedničkim tačkama su samo jednom diferencijabilne. Ovo je u redu za neke primene, ali recimo ne bi bile pogodne za definisanje putanje kamere jer bi rezultujuće kretanje posmatraču delovalo "skokovito".

### 7.3 Kubni B-splajn

**Kubni B-splajnovi**<sup>1</sup> su slični Catmull-Rom splajnovima. Ključne razlike su u tome što su kubni B-splajnovi:

- $C^2$  glatki, tj. i prvi i drugi izvod su neprekidne funkcije,
- neinterpolišući, tj. oni prolaze blizu kontrolnih tačaka ali ne kroz njih u opštem slučaju.

Iako B-splajnovi ne prolaze kroz kontrolne tačke, njihov dodatni stepen neprekidnosti ih čini atraktivnim za mnoge primene.

### 7.4 Crtanje krivih

**Prvi pristup:** vrednost  $t$  se povećava za konstantnu (malu) vrednost i tačka  $(x(t), y(t), z(t))$  se pravom linijom spaja sa prethodno određenom tačkom (ovaj pristup se može efikasno implementirati imajući u vidu da je kriva opisana polinomom trećeg stepena i da je njene izvode lako opisati)

**Drugi pristup:** rekurzivna podela krive na segmente dok se ne dođe do pravih linija (tj. približno pravih linija); implementira se različito za različite tipove krivih, a razlikuju se i testovi da li se segment može aproksimisati pravom linijom.

### 7.5 Pitana

- 7.1 Da li je dobro čuvati krive u rasterskom obliku? Čemu služe kontrolne tačke prilikom zadavanja krive?
- 7.2 Zašto deo po deo linearna aproksimacija ne predstavlja najbolji izbor? Zašto trigonometrijske funkcije nisu odgovarajući izbor? Zasto aproksimacija polinomima stepena 3 predstavlja pogodan izbor?
- 7.3 Navesti tri različita načina na koje je moguće zadati krivu.
- 7.4 Definisati Hermitovu krivu.
- 7.5 Šta je karakteristično za Hermitovu bazu?
- 7.6 Šta je geometrijska matrica Hermitove krive i čime je određena?
- 7.7 Definisati Bezierovu krivu.
- 7.8 Šta je splajn?
- 7.9 Da li je kod Catmull-Rom splajna dat vektor tangente u svakoj od kontrolnih tačaka?
- 7.10 Navesti razlike između Catmull-Rom splajna i B-splajna.
- 7.11 Koja su dva načina za crtanje krivih?

---

<sup>1</sup>postoje i linearni, kvadratni, ... B-splajnovi ali kubni se najviše koriste



## Glava 8

# Vidljivost

### 8.1 Uvod (“To render or not to render, that is the question...”)

Određivanje vidljivih delova površi je fundamentalni problem u računarskoj grafici. Renderovanje objekata koji nisu vidljivi je neefikasno, a ujedno i netačno. Problem utvrđivanja koje su površi vidljive nazivamo **određivanjem vidljivih površina** (engl. visible surface determination – VSD), odnosno **odbacivanjem skrivenih površina** (engl. hidden surface removal – HSR), zavisno od smera kojim se pristupa problemu.

Prilikom određivanja vidljivih površina postoje dva različita cilja: jedan je obezbeđivanje *tačnosti* algoritma, a drugi rad na njegovoj *efikasnosti*. Algoritam određivanja vidljivih površina fokusiran na tačnost renderovanja mora da sa tačnošću utvrdi da li između dve tačke postoji linija vidljivosti bez prepreka. Osnovna primena ovakvog tipa algoritma je određivanje **primarne vidljivosti**, odnosno utvrđivanje koje su površine vidljive sa pozicije kamere. Na ovaj način se samo delovi scene koji su zaista vidljivi iscrtavaju na slici i na taj način se garantuje tačan rezultat renderovanja. Danas najpopularniji metodi koji obezbeđuju tačno određivanje vidljivosti su:

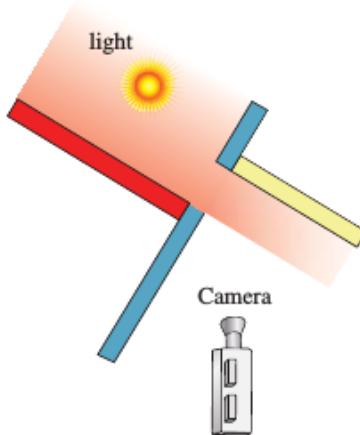
- rej kasting algoritam (engl. ray casting algorithm) i
- algoritam zasnovan na baferu dubine (engl. depth-buffer algorithm).

Algoritam **konzervativnog određivanja vidljivosti** je razvijen za potrebe efikasnosti. U ovom algoritmu razdvajaju se delovi scene koji su verovatno vidljivi od delova scene koji sigurno nisu vidljivi. Eliminacija delova koji nisu vidljivi smanjuje broj potrebnih testova vidljivosti, ali sama po sebi ne garantuje korektnost. Kada se rezultat ovakvim metodama može dobiti mnogo brže nego algoritmima koji garantuju tačnost, renderovanje se može ubrzati na sledeći način: najpre se primeni algoritam konzervativnog određivanja vidljivosti i na taj način se vrši odsecanje početnog skupa, a zatim se na njega primeni algoritam koji garantuje tačno određivanje vidljivosti. Na primer, efikasnije je utvrditi da se sfera koja ograničava mrežu trouglova nalazi iza kamere i stoga je nevidljiva za kameru nego zasebno testirati svaki trougao te mrežu na vidljivost.

**Odbacivanje zadnje strane (naličja) objekta ili dela van zarubljene piramide pogleda** (engl. backface culling i frustum culling) su dva jednostavna i efikasna metoda za konzervativno testiranje vidljivosti. **Blokirajuće odbacivanje** (engl. occlusion culling) je složenija varijanta odbacivanja dela van zarubljene piramide pogleda, koja u obzir uzima i blokiranje između objekata na sceni. U cilju smanjenja cene konzervativnog testiranja vidljivosti razvijene su neke sofisticirane prostorne strukture podataka, kao što je recimo **stablo binarnog particionisanja prostora** (engl. Binary Space Partition tree), koje ćemo u nastavku zvati BSP stablo.

Dakle, često je efikasno pristupiti problemu sa jednim ili dva brza konzervativna metoda koja sužavaju prostor koji se razmatra odbacivanjem vrednosti koje su očigledno netačne, a zatim primeniti neki sporiji ali tačni metod za razmatranje manjeg broja preostalih mogućnosti.

Primarna vidljivost nam govori koje površi emituju ili razbacuju svetlost ka kameri i one su jedine površi koje direktno utiču na sliku. Međutim, treba imati na umu da globalno osvetljenje ne sme u potpunosti da eliminiše tačke koje nisu vidljive iz oka kamere, jer iako one možda ne razbacuju svetlost direktno ka kameri, mogu da utiču na sliku. Na slici 8.1 prikazan je primer kada odbacivanje površi koja je nevidljiva iz oka kamere menja sliku, jer ta površ rasipa svetlost na druge površi koje jesu vidljive iz oka kamere. Značaj indirektnog uticaja tačaka koje nisu vidljive iz oka kamere na sliku predstavlja razlog zašto se vidljivost definiše za proizvoljan par tačaka, a ne samo za oko kamere i tačku sa scene.



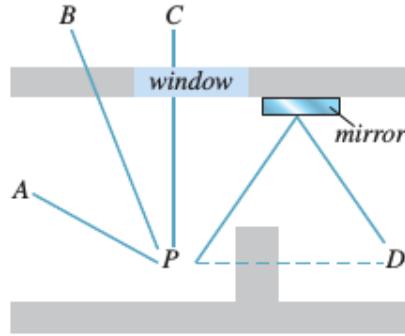
Slika 8.1: Žuti zid je osvetljen samo svetlošću reflektovanom od strane sakrivenog crvenog poligona. Ako bismo ga uklonili, ovaj zid bi bio osvetljen samo svetlošću plave površi

### 8.1.1 Funkcija vidljivosti

Algoritmi određivanja vidljivih površina zasnovani su na preciznoj definiciji pojma vidljivosti. Ipak, s obzirom na to da se često na sceni nalaze velike kolekcije površi, iscrpno testiranje vidljivosti bilo bi neefikasno. Stoga tražimo

način da amortizujemo cenu za veliki broj površi i veliki broj parova tačaka.

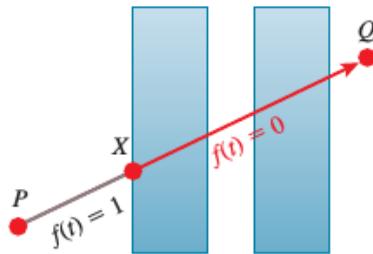
Za date tačke  $P$  i  $Q$  na sceni, definišemo da je **funkcija vidljivosti**  $V(P, Q) = 1$  ako ne postoji presek između scene i otvorene duži  $(P, Q)$ , a inače je  $V(P, Q) = 0$ . Nekada je zgodno raditi sa **funkcijom blokiranja**  $H(P, Q) = 1 - V(P, Q)$ . Za funkciju vidljivosti važi svojstvo simetričnosti:  $V(P, Q) = V(Q, P)$ .



Slika 8.2:  $V(P, A) = 1$  jer nema prepreka između ove dve tačke;  $V(P, B) = 0$  jer postoji zid između ove dve tačke;  $V(P, C) = 0$  jer iako se iz tačke  $P$  može videti tačka  $C$  kroz prozor, prozor se smatra preprekom; slično,  $V(P, D) = 0$  iako  $P$  vidi odraz tačke  $D$  u ogledalu

Primetimo da se vidljivost odnosi striktno na geometrijsku liniju pogleda. Ako su  $P$  i  $Q$  razdvojeni prozorskim staklom,  $V(P, Q) = 0$  jer neprazni deo scene (prozorsko staklo) preseca duž  $PQ$ . Slično, ako posmatrač u  $Q$  nema direktnog pogleda na  $P$ , ali može da vidi odraz od  $P$  ili njegovu senku, i dalje kažemo da je  $V(P, Q) = 0$ .

Neka je  $X$  prva tačka scene koju pogađa zrak sa početkom u tački  $P$  u smeru  $\omega$ . Tačka  $X$  razdvaja zrak na dva dela. Možemo definisati funkciju vidljivosti  $f$  kao funkciju rastojanja od tačke  $P$ , odnosno neka je  $f(t) = V(P, P + wt)$ : za  $0 \leq t \leq |X - P|$  važi  $f(t) = 1$ , a za  $t > |X - P|$  važi  $f(t) = 0$ . Dakle, izračunavanje funkcije vidljivosti se svodi na određivanje prve tačke preseka  $X$  na osnovu početne tačke zraka  $P$  i pravca zraka  $\omega$ . Prva tačka preseka duž zraka predstavlja rezultat **upita preseka**.



Slika 8.3: Vidljivost parametrizovana rastojanjem duž zraka

Primetimo da smo funkciju vidljivosti definisali na otvorenoj duži koja ne

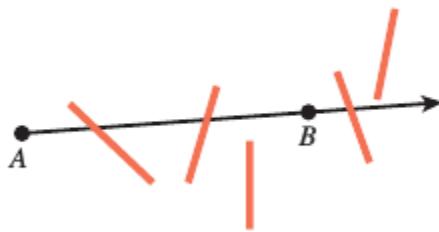
uključuje krajnje tačke. Ovo znači da ako zrak iz tačke  $P$  ka tački  $Q$  prvo seče geometriju scene u tački  $X$  različitoj od  $P$ , onda je  $V(P, X) = 1$ . Da smo razmatrali zatvorene duži, tada nikada ne bi bilo nikakve vidljivosti između površi na sceni – one bi zaklanjale same sebe.

### 8.1.2 Primarna vidljivost

**Primarna vidljivost** je vidljivost između tačke sa otvora kamere i tačke na sceni. Da bi se slika renderovala, potrebno je izvršiti po jedan test vidljivosti za svaki zrak svetla uperen u ravan slike. U najjednostavnijem slučaju postoji po jedan zrak u centru svakog piksela. Tačkasta kamera ima otvor koji je dimenzije nula, te za svaku tačku iz ravni slike postoji tačno jedan zrak kojim putuje svetlost. Nazivamo ga **primarni zrak** za tu tačku sa ravni slike. S obzirom na to da prilikom izračunavanja funkcije vidljivosti ili upita preseka svi razmatrani zraci dele jednu krajnju tačku (sa otvora kamere), postoji mogućnost da se ove operacije ubrzaju.

### 8.1.3 Pokrivenost

**Pokrivenost** je specijalan slučaj vidljivosti koji se odnosi na tačke iz *ravni slike*. Za scenu koja se sastoji od jednog primitivnog objekta, pokrivenost primarnim zrakom je binarna vrednost 1 ako zrak preseca primitivu između ravni slike i beskonačnosti, a 0 inače. Za scenu koja se sastoji od više primitiva, primitive mogu da zaklanjaju jedna drugu u smeru pogleda kamere. **Dubinska složenost** zraka je broj puta koji on preseca scenu. Za proizvoljnu tačku  $P$  definisemo **kvantitativnu nevidljivost** kao broj preseka sa drugim primitivama koje se nalaze između početka zraka i tačke  $P$  (slika 8.4).



Slika 8.4: Kvantitativna nevidljivost tačke  $B$  u odnosu na tačku  $A$  je 2; dubinska složenost zraka je ukupan broj površina koje preseca zrak. Zrak iz tačke  $A$  kroz tačku  $B$  ima dubinsku složenost 3.

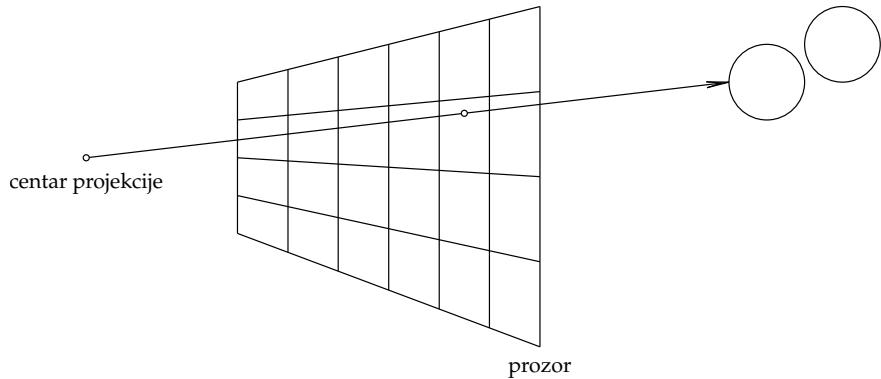
Slučaj kada je presek zraka i površi duž, a ne konačan broj tačaka je problematičan (slika 8.5). Ljudi su “veštii” u kreiranju ovakvih situacija, postavljanjem ivica u čvorove celobrojne mreže, poravnanjem površi sa koordinatnim osama, što se veoma retko dešava u realnoj situaciji. Za zatvorene poligonalne modele uobičajeno je da se ovakvi preseci ne računaju.



Slika 8.5: Zrak tangentan na površinu koju preseca

## 8.2 Rej kasting

**Rej kasting** ili **rej trejsing** algoritam utvrđuje da li je neka površ vidljiva praćenjem imaginarnog zraka svetlosti od oka kamere do objekta na sceni. Dakle, za zadato oko kamere (centar projekcije) i prozor proizvoljne ravni pogleda (koji je podeljen pravilnom mrežom tako da elementi odgovaraju pikselima potrebne rezolucije), za svaki piksel se prati po jedan zrak od centra projekcije kroz sam piksel do najbližeg objekta na sceni. Boja piksela se postavlja na boju najbližeg objekta.



```

izaberi centar projekcije i prozor na projekcionoj ravni;
za svaku scan liniju na slici uradi sledeće:
    za svaki piksel na scan liniji uradi sledeće:
        begin
            odredi polupravu iz centra projekcije kroz piksel;
            za svaki objekat na sceni uradi sledeće:
                ako poluprava sece objekat i ako je presecna tacka najbliža do sada onda
                    zapamti presek i ime objekta;
                oboji piksel bojom najbližeg preseka
        end
    
```

Rej trejsing algoritam su razvili Appel (1968) i Goldstein i Nagel (1968, 1971). Appel je koristio algoritam da odredi da li je tačka u senci, a Goldstein i Nagel su ga originalno razvili za simuliranje putanja balističkih projektila. Tek kasnije se uvidelo da ima koristi u računarskoj grafici. Naziv *rej kasting* se obično koristi samo za varijantu za određivanje vidljivosti, a naziv *ray tracing* za puni rekurzivni algoritam (kojim se obrađuju i senke, refleksija i refrakcija).

U osnovnoj verziji algoritma za svaki piksel  $(x_1, y_1, z_1)$  potrebno je odrediti presečne tačke sa svim objektima tj. potrebno je odrediti vrednost parametra  $t$  tako da tačka  $x = x_0 + t(x_1 - x_0)$ ,  $y = y_0 + t(y_1 - y_0)$ ,  $z = z_0 + t(z_1 - z_0)$ , pripada objektu; sva ta izračunavanja mora da budu brza. Za efikasnu primenu ray tracing algoritma koriste se mnoge optimizacije, kao što su:

- optimizacija izračunavanja preseka
- izbegavanje (nepotrebnih) izračunavanja preseka (na bazi koherencije, neprekidnosti)
- hijerarhije (objekat koji pripada nekom telu  $V$  ne može da se seče sa zrakom svetla ako taj zrak ne seče telo  $V$ )

Postoje i varijante algoritma koje se bave problemom antialiasing (četiri zraka za svaki piksel, detektovanje i prikazivanje veoma malih objekata, itd)

Rej kasting algoritam predstavlja direktni proces odgovaranja na upit preseka.

Složenost rej kasting algoritma u odnosu na  $n$  objekata (najčešće trouglova) u nizu je  $O(n)$  operacija. Linearna složenost je nepraktična za velike i složene scene, posebno imajući u vidu da su te scene uglavnom baš one kod kojih skoro sve površi nisu vidljive iz date tačke. Stoga se za skoro sve primene koriste neke druge strukture podataka da bi se postigla složenost manja od linearne.

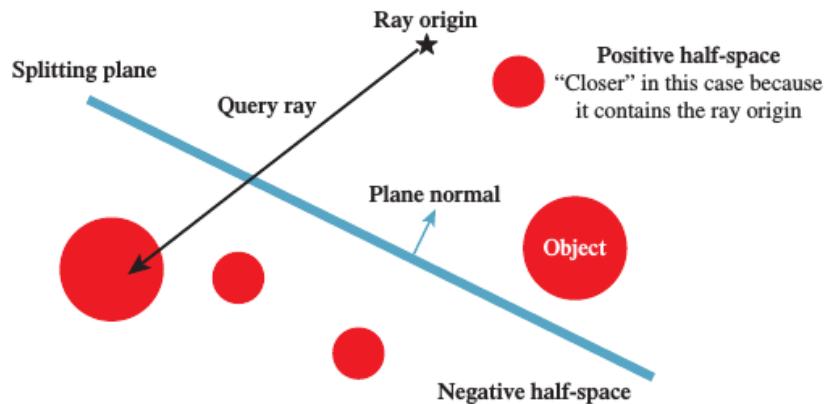
### 8.2.1 Binarno stablo prostornog particionisanja (BSP stablo)

**BSP stablo** je struktura podataka koja se koristi za uređivanje geometrijskih primitiva na osnovu njihovog položaja i dometa. BSP stablo podržava nalaženje prvog preseka između zraka i primitive. Ovaj algoritam često ima logaritamsku složenost u funkciji broja primitiva. Na ovaj način se rej kasting algoritam može učiniti upotebljivim za velike scene.

BSP stablo se može koristiti i za računanje funkcije vidljivosti. Algoritam kojim se računa vrednost funkcije vidljivosti je skoro identičan upitu za prvi presek: on se zaustavlja sa povratnom vrednošću `false` kada se detektuje proizvoljni presek, a inače vraća vrednost `true`.

Postoji nekoliko varijanti BSP strukture, a ovde će biti izložena njena jednostavna varijanta. U jednostavnom BSP stablu svaki unutrašnji čvor predstavlja **ravan razdvajanja** (koja nije deo geometrije scene) a svaki list geometrijsku primitivu na sceni. Ravan deli prostor na dva poluprostora. *Pozitivni poluprostor* sadrži sve tačke u ravni razdvajanja, kao i tačke sa one strane ravni kojoj pripada vektor normale ravni, a *negativni poluprostor* sadrži sve tačke u ravni i sa strane ravni suprotne od vektora normale ravni. Prilikom kreiranja punog stabla i pozitivni i negativni poluprostor biće podeljeni dodatnim ravn-

ima, sve dok svaka primitiva ne bude razdvojena od ostalih barem jednom ravni.



Slika 8.6: Ravan razdvajanja za jedan unutrašnji čvor BSP stabla deli scenu koja se sastoji od 5 sfera na dva poluprostora

Unutrašnji čvor BSP stabla ima najviše dva sina, koje označavamo **pozitivnim** i **negativnim**. Algoritam konstrukcije stabla garantuje da pozitivno podstablo sadrži samo primitive koje su u pozitivnom poluprostoru ravni, a negativno samo one koje su u negativnom poluprostoru ravni. Ako primitiva sa scene preseca ravan razdvajanja, onda je algoritam deli na dve primitive u odnosu na tu ravan.

U algoritmu za izračunavanje funkcije vidljivosti, sav posao obavlja rekurzivna funkcija *sece*. Tačka  $Q$  je vidljiva iz tačke  $P$  ako ne postoji nikakav presek između duži  $PQ$  i geometrije u podstablu sa korenom cvor. Kada je cvor list on sadrži jednu geometrijsku primitivu, stoga funkcija *sece* testira da li je prazan presek prave i primitive. Na slici 8.8 dat je prikaz prolaska kroz iteracije algoritma za scenu koja se sastoji iz diskova.

Ako je cvor unutrašnji čvor, onda on sadrži ravan razdvajanja koji kreira dva poluprostora. Klasifikujemo sinove čvora cvor na bližeg i daljeg u odnosu na tačku  $P$ . Na slici 8.6 prikazan je primer klasifikacije u unutrašnjem čvoru. S namerom da ponovo iskoristimo strukturu algoritma za blizak problem pro-nalaženja prvog preseka, biramo da prvo idemo u bližeg sina. To je zato što ako postoji neki presek između duži  $PQ$  i scene u poluprostoru blizi, on mora biti bliži tački  $P$  od bilo kog preseka u poluprostoru dalji.

Ako duž  $PQ$  leži u potpunosti u jednom poluprostoru, rezultat testa da li postoji presek za tekući čvor svodi se na rezultat testa u tom poluprostoru. Inače, ako postoji presek, onda on može da postoji u oba poluprostora, pa algoritam rekurzivno posećuje obe.

U najgorem slučaju algoritam mora da poseti svaki čvor stabla. U praksi se ovo retko događa. Obično je  $PQ$  malo u odnosu na veličinu scene i ravni isecaju prostor u konveksne regije koji ne leže duž iste linije. Stoga očekivano je dobiti relativno usku pretragu u dubinu sa vremenom izvršavanja proporcionalnom visini stabla.

```

function V(P,Q):
    return !sece(P,Q,cvor)

function sece(P,Q,cvor):
    ako je cvor list:
        return (da li PQ sece primitivu u cvoru cvor)

    blizi = cvor.pozitivniSin
    dalji = cvor.negativniSin

    if P je u negativnom poluprostoru cvora:
        // negativna strana ravni je bliza tacki P
        zameni blizi, dalji

    if sece(P,Q,blizi):
        // zaustavlja se ranije jer je pronadjen presek
        return true

    if P i Q su u istom poluprostoru u cvoru:
        // segment PQ se ne pruza u dalji poluprostor
        return false

    // nakon pretrage blize strane,
    // rekurzivno trazimo presek u daljoj strani
    return sece(P,Q,dalji)

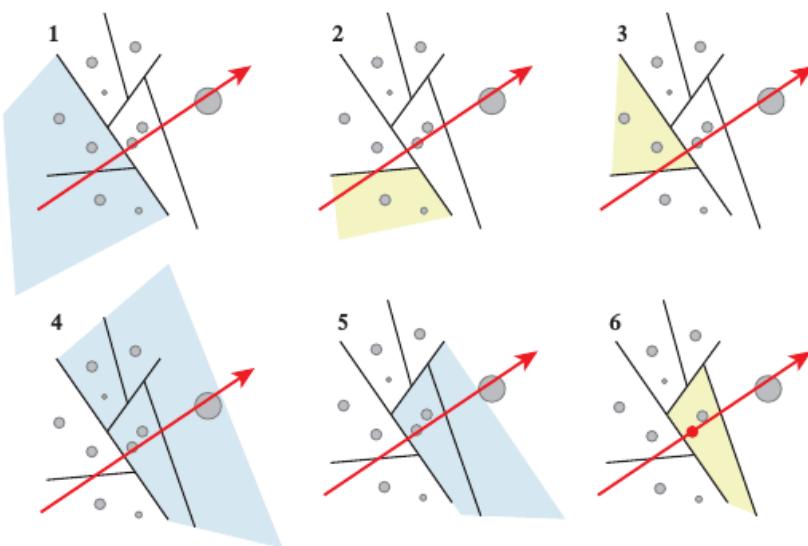
```

Slika 8.7: Algoritam za izračunavanje funkcije vidljivosti za jednostavno BSP stablo

### 8.3 Bafer dubine

**Bafer dubine** (engl. depth buffer) je dvodimenzionalni niz paralelan slići koja se renderuje. Poznat je i pod nazivom **z-bafer**. U najjednostavnijoj varijanti, postoji po jedan uzorak boje za svaki piksel, i po jedan skalar za svaki piksel koji daje neku meru rastojanja centra projekcije od površi koja boji piksel. Da bi se redukovao aliasing koji se dobija korišćenjem jednog uzorka po pikselu, često se koristi više uzoraka boja i dubine za svaki piksel.

Bafer dubine se najčešće koristi da bi se obezbedila korektna vidljivost prilikom rasterizacije. Postoje tri uobičajene primene bafera dubine prilikom određivanja vidljivosti. Prva, dok se scena renderuje, bafer dubine sadrži implicitnu informaciju o utvrđivanju vidljivih površi. Nova površ može da pokrije uzorak samo ako je njena dubina u odnosu na kameru manja od one sadržane u baferu dubine. Ako je tako, onda se nova boja upisuje na mesto stare u baferu dubine, kao i nova vrednost dubine. Ovo je tzv. *implicitna vidljivost* jer sve dok se cela scena ne renderuje nije poznato koja je najbliža vidljiva površina. Kada se renderovanje završi, obezbeđena je korektna informacija o vidljivosti.



Slika 8.8: Praćenje zraka kroz scenu koja sadrži diskove koja se čuva u 2D BSP stablu. Osenčevni delovi prostora odgovaraju čvoru nad kojim algoritam radi u svakom od koraka. Iteracija napreduje u dubinu, sa tim da se prvo silazi u sinu koji je bliži.

Drugo, nakon što se scena renderuje, bafer dubine opisuje prvi presek sa scenom za svaki zrak iz centra projekcije kroz uzorak.

Treće, nakon što se scena renderuje, bafer dubine može da direktno izračuna funkciju vidljivosti u odnosu na centar projekcije. Za tačku  $Q$  važi  $V((0, 0, 0), Q) = 1$  akko je vrednost dubine u tački projekcije  $Q$  manja od dubine  $Q$ . Zašto je od značaja rešiti upite vidljivosti *nakon* što je renderovanje završeno? Mnogi procesi renderovanja imaju višestruke prolaze kroz scenu i frejm-bafer. Mogućnost da efikasno izračunamo upite preseka zraka i vidljivost nakon inicijalnog prolaza omogućava da naknadni prolazi budu efikasniji.

#### 8.4 z-bafer algoritam

- Autor: Catmull 1974.
- Zahteva postojanje memorijskog prostora ne samo za vrednost boje za svaki piksel (frejm bafer F), već i memorijskog prostora za  $z$ -vrednosti (tj.  $z$ -koordinate) za svaki piksel ( $z$ -bafer).
- Sve vrednosti u F baferu su inicijalizovane na boju pozadine, a sve vrednosti u Z baferu su inicijalizovane na -1 (kao na  $z$ -koordinatu zadnje ravni odsecanja). Maksimalna moguća vrednost u  $z$ -baferu je  $z$ -koordinata prednje ravni odsecanja.
- Na sve poligone (nije bitan poređak) koji reprezentuju scenu primenjuje se scan-conversion algoritam. Prilikom scan-konverzije za tačku  $(x, y)$ , ako ta tačka nije od posmatrača dalja od tačke koja je trenutno u baferima, onda vrednosti te tačke zamenjuju postojeće vrednosti u baferima.

```

Procedure zBuffer
var
    pz: integer;
begin
    for y:=0 to YMAX do
        for x:=0 to XMAX do
            begin
                WritePixel(x,y,background_value);
                WriteZ(x,y,-1)
            end

    za svaki poligon uradi sledece
    za svaki piksel u projekciji poligona uradi sledece
    begin
        pz:= z-vrednost poligona u tacki sa koordinatama (x,y);
        if pz>=ReadZ(x,y) then
            begin { ova tacka nije dalja }
                WriteZ(x,y,pz);
                WritePixel(x,y,boja poligona u tacki (x,y))
            end
        end
    end
end.

```

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

## 8.5 Algoritmi sa listama prioriteta

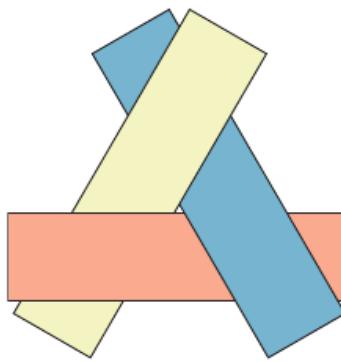
Algoritmi sa listama prioriteta implicitno rešavaju problem vidljivosti tako što renderuju elemente scene u redosledu u kom zaklonjeni objekti imaju viši prioritet te se prvi renderuju i na taj način budu naknadno sakriveni objektima koji se kasnije renderuju. Danas se ovi algoritmi ne koriste često jer su razvijene bolje alternative. Ipak, jednostavnost implicitnog utvrđivanja vidljivosti korišćenjem prioriteta ih čini korisnim za neke specijalne namene.

### 8.5.1 Slikarev algoritam

Posmatramo mogući proces umetnika koji slika pejzaž. Slikar prvo slika nebo, a zatim planine koje zaklanjaju nebo. U prednjem planu, slikar slika drveće preko planina. Ovo se naziva **slikarevim algoritmom** (engl. Painter's

algorithm) u računarskoj grafici. Zaklanjanje i vidljivost se postižu tako što se boje daljih tačaka “pregaze” bojama bližih tačaka. Možemo primeniti ovu ideju na svaku lokaciju uzorka jer za svaki uzorak postoji korektno back-to-front uređenje tačaka koje direktno utiču na njega. U ovoj varijanti algoritam je neefikasan jer zahteva sortiranje svih tačaka, ali daje tačan rezultat.

Iz razloga efikasnosti, slikarev algoritam se često primenjuje na cele primitive, npr. trouglove. U ovom slučaju algoritam ne uspeva. Dok primitive veće od tačaka mogu često da se urede tako da daju tačnu informaciju o vidljivosti, postoje situacije kada ovo nije moguće uraditi. Na slici 8.9 prikazan je slučaj kada to nije moguće uraditi. Međutim, ako dozvolimo podelu primitiva tamo gde se njihove projekcije sekut možemo dobiti uređenje.

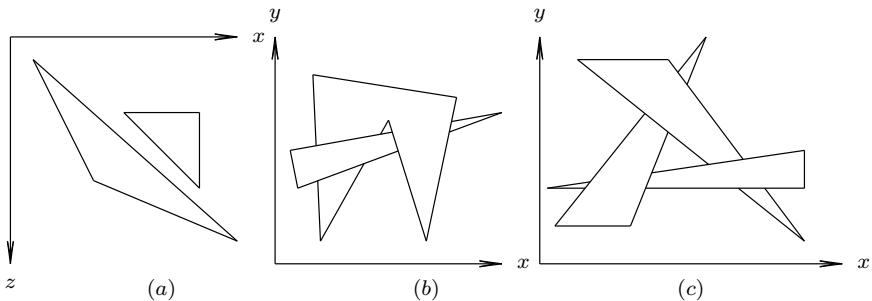


Slika 8.9: Tri konveksna poligona koja se ne mogu ispravno renderovati korišćenjem slikarevog algoritma zbog njihovog uzajamnog preklapanja. Za svaku tačku postoji striktno uređenje po dubini, ali ne postoji korektno uređenje celih pravougaonika

### 8.5.2 Algoritam sortiranja dubine

Newell i drugi su razvili proširenje slikarevog algoritma za poligone tako da se u svim slučajevima dobije tačan izlaz i njega nazivamo **algoritam sortiranja dubine**. On se sastoji iz četiri koraka:

1. dodeliti svakom poligonom ključ sortiranja koji je jednak  $z$ - vrednosti tema koje je najdalje od oblasti za prikaz
2. sortirati sve poligone od najudaljenijeg do najbližeg prema vrednosti ključa
3. otkriti slučajeve kada dva poligona imaju dvosmisleno uređenje. Podeliti ovakve poligone dok delovi nemaju eksplicitno uređenje i postaviti ih na pravo mesto u sortiranoj listi
4. renderovati sve poligone u redosledu prioriteta, od najudaljenijeg do najbližeg



Poligoni se obrađuju jedan po jedan. Za tekući poligon  $P$  (najdalji od posmatrača, nalazi se na kraju uređene liste poligona) mora da se ispita svaki poligon  $Q$  za koji se  $z$  koordinate preklapaju sa  $z$  koordinatama poligona  $P$ , da bismo bili sigurni da poligon  $P$  ne može da zakloni  $Q$  i da se poligon  $P$  stoga može iscrtati pre poligona  $Q$ . Za  $P \cap Q$  se primenjuje sledeća lista testova (koja je uređena u rastućem redosledu složenosti):

1. da li se  $x$  koordinate dva poligona ne preklapaju (tj. ne postoje tačke u  $P \cap Q$  sa istom  $x$  koordinatom)?
2. da li se  $y$  koordinate dva poligona ne preklapaju (tj. ne postoje tačke u  $P \cap Q$  sa istom  $y$  koordinatom)?
3. da li je čitav  $P$  sa suprotne strane ravni poligona  $Q$  u odnosu na posmatrača?
4. da li je čitav  $Q$  sa iste strane ravni poligona  $P$  kao i posmatrač?
5. da li su projekcije poligona  $P$  i  $Q$  na projekcionu ravan disjunktnе?

Ako bar jedan od testova uspe, onda se poligon  $P$  isrtava scan-conversion algoritmom i sledeći poligon dobija ulogu poligona  $P$ . Ako nijedan od ovih testova ne uspe, onda smatramo da se poligoni  $P \cap Q$  potencijalno preklapaju, te proveravamo da li se poligon  $Q$  može iscrtati pre poligona  $P$ . Testove 1, 2 i 5 ne treba ponavljati, a testove 3 i 4 treba ponoviti sa zamenjenim ulogama poligona  $P$  i  $Q$ :

- 3' da li je čitav  $Q$  sa suprotne strane ravni poligona  $P$  u odnosu na posmatrača?
- 4' da li je čitav  $P$  sa iste strane ravni poligona  $Q$  kao i posmatrač?

Ako jedan od ovih testova uspe, onda poligoni  $Q$  i  $P$  zamenjuju uloge.

U primeru (a) uspeva test 3', pa se poligon  $Q$  pomera na kraj liste i uzima ulogu novog poligona  $P$ . U primeru (b) ni ovi testovi ne omogućavaju razrešenje i onda ili  $P$  ili  $Q$  mora biti podeljen na poligone od strane ravni koja sadrži drugi polazni poligon (primenom kliping algoritma)

Primer (c) ilustruje mogućnost beskonačne petlje u algoritmu; da bi ona bila izbegнута, vrši se označavanje svakog poligona koji se pomera na kraj liste. Tako, kad god nijedan od prvih pet testova ne uspe i tekući poligon  $Q$  je već obeležen, ne primenjujemo testove 3' i 4' već delimo  $P$  ili  $Q$  i u listu ih zamenjujemo njihovim delovima.

## 8.6 Odsecanje i odbacivanje u odnosu na zarubljenu piramidu pogleda

Postoje tri uobičajena pristupa povećanju efikasnosti koja su u vezi sa zapreminom pogleda:

- odbacivanje u odnosu na zarubljenu piramidu pogleda – ovim se eliminaju poligoni koji su u potpunosti van zapremine pogleda
- odsecanje u odnosu na prednju ravan odsecanja
- odsecanje u odnosu na kompletну zapreminu pogleda – odsecanje poligona u odnosu na strane i zadnju ravan, iz razloga efikasnosti

Odbacivanje u odnosu na zapreminu pogleda nije teško: testira se svako teme poligona da li se nalazi izvan ravni koja ograničava zapreminu pogleda.

Odsecanje se vrši u skladu sa algoritmom koga smo ranije videli.

Takođe, ako radimo sa neprozirnim, čvrstim telima, pozadina objekta je nužno sakrivena od direktnе linije posmatranja. Sam objekat zaklanja zrake pogleda. Stoga, odbacivanje primitiva koje se nalaze sa zadnje strane objekta može da eliminiše oko polovine geometrije scene.

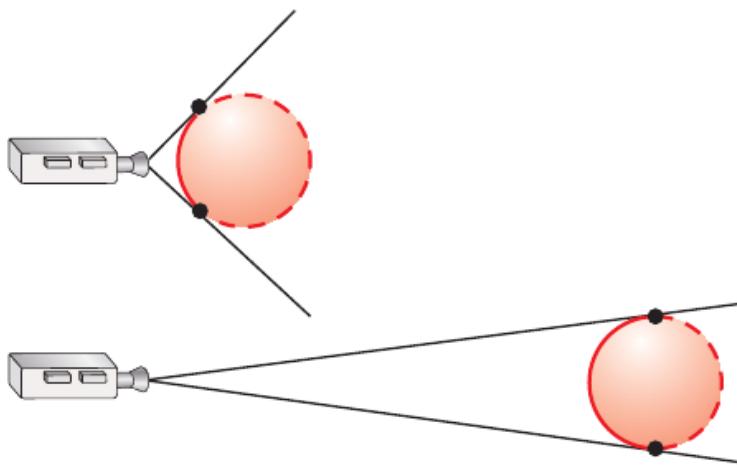
Intuitivno je jasno prepoznati zadnju stranu objekta, ali kako je geometrijski odrediti? Prepostavimo da imamo zatvorenu mrežu poligona bez saopreseka i posmatrača u tački  $Q$  koji se nalazi izvan poliedra definisanog mrežom. Neka je  $P$  teme poligona i  $n$  normala u odnosu na poligon. Poligon definiše ravan koja prolazi kroz  $P$  i ima vektor normale  $n$ . Kažemo da je poligon **sa prednje strane** u odnosu na  $Q$  ako se  $Q$  nalazi u pozitivnoj poluravni poligona, a da je **sa zadnje strane** ako se  $Q$  nalazi u negativnoj poluravni. Ako se  $Q$  nalazi tačno u ravni, tada se poligon nalazi **na konturi** krive koja razdvaja prednju i zadnju stranu objekta.

- $(Q - P) \cdot n > 0$  – poligon je sa prednje strane
- $(Q - P) \cdot n < 0$  – poligon je sa zadnje strane
- $(Q - P) \cdot n = 0$  – poligon je na konturi

Rezultat će biti isti bez obzira na to koje teme poligona razmatramo. Ipak, kao što je prikazano na slici, objekat bliži kameri teži tome da ima veću zadnju stranu od objekta koji se nalazi dalje od kamere.

## 8.7 Pitana

- 8.1 Koja dva cilja razlikujemo pri rešavanju problema određivanja vidljivih površina?
- 8.2 Kako se definiše funkcija vidljivosti između tačaka  $P$  i  $Q$ ?
- 8.3 Kako je moguće definisati funkciju vidljivosti u funkciji rastojanja?
- 8.4 Šta je primarna vidljivost? Zašto je važno definisati pojam vidljivosti za proizvoljan par tačaka, a ne samo za otvor kamere i tačku sa scene?



- 8.5 Šta je dubinska složenost zraka?
- 8.6 Šta je kvantitativna nevidljivost tačke  $P$  u odnosu na tačku  $Q$ ?
- 8.7 Iz kojih se koraka sastoji rej kasting algoritam?
- 8.8 Čemu odgovaraju unutrašnji čvorovi BSP stabla, a čemu listovi? Koji poluprostor nazivamo pozitivnim, a koji negativnim u odnosu na ravan razdvajanja?
- 8.9 Iz kojih se koraka sastoji algoritam za izračunavanje vrednosti funkcije vidljivosti za scenu priказанu BSP stablom?
- 8.10 Koja je očekivana složenost algoritma za nalaženje prvog preseka zraka i primitive ako se koristi BSP stablo?
- 8.11 Šta je bafer dubine ( $z$  bafer)? Koje su njegove moguće primene?
- 8.12 Kako funkcioniše  $z$  bafer algoritam?
- 8.13 Koja je ideja algoritama sa listama prioriteta?
- 8.14 Da li je uvek moguće uređiti dati skup poligona?
- 8.15 Iz kojih koraka se sastoji algoritam sortiranja dubine?
- 8.16 Kod algoritma sortiranja dubine, koji se testovi koriste prilikom razmatranja dva poligona čije se  $z$  koordinate preklapaju?
- 8.17 Na koji način je moguće utvrditi da li je neki poligon sa prednje ili zadnje strane objekta?

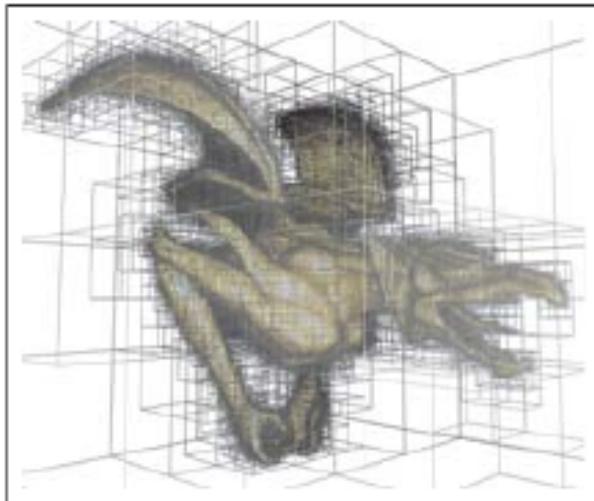
## *Glava 9*

---

# Prostorne strukture podataka

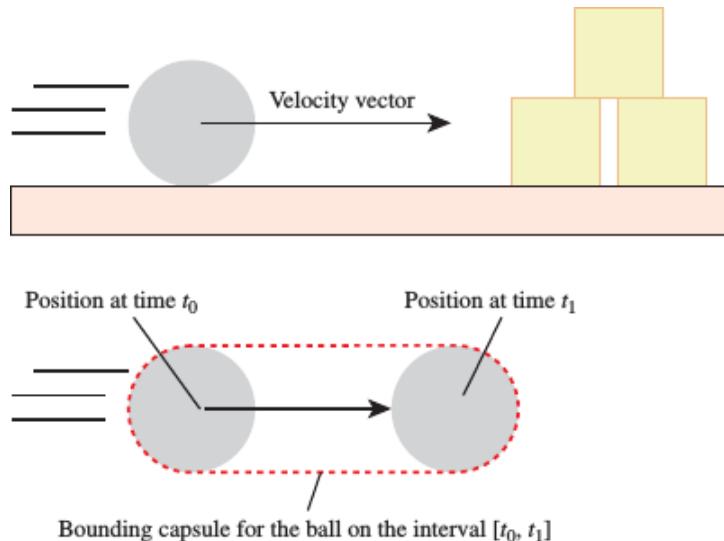
---

**Prostorne strukture podataka** predstavljaju višedimenziono uopštenje klasičnih uređenih struktura podataka, kao što su binarna stabla pretrage. S obzirom na to da prostorne strukture podataka imaju za cilj povećanje prostora za skladištenje a da se pritom smanji vreme izvršavanja upita, poznate su i pod nazivom **ubrzavajuće strukture podataka**. One su jako korisne kada je potrebno naći presek između različitih geometrijskih figura. Na primer, koriste se za određivanje prvog trougla u mreži koji preseca zrak svetla. Ove strukture podataka, iako primarno razvijene za potrebe renderovanja i animacije u računarskoj grafici, danas se koriste i u drugim oblastima – mašinskom učenju, statističkim algoritmima i slično.



Slika 9.1: Model gargojla predstavljen korišćenjem oktrija. Zapremina kocke koja okružuje model se rekurzivno deli na sve manje kocke, formirajući na taj način drvoliku strukturu podataka. Ova reprezentacija omogućava efikasnije upite preseka u poređenju sa iscrpnim prolaskom kroz trouglove mreže.

Mnogi grafički algoritmi se oslanjaju na upite koji se mogu opisati geometrijskim preseцима. Na primer, posmatrajmo animiranu scenu sa slike 9.2 na kojoj je prikazana lopta koja se konstantnom brzinom kreće ka piramidi sastavljenoj od tri poredane kocke. Lopta tokom kretanja po pravolinijskom putu ostavlja trodimenzionalni trag u obliku kapsule (valjak čiji je poluprečnik jednak poluprečniku lopte i koji je sa obe strane zatvoren poluloptom čime se ogradiju sve početne i završne strane sfere). Za svaku diskretnu fizičku simulaciju kretanja lopte tokom nekog vremenskog intervala, može se odrediti kapsula koja ograničava sve lokacije lopte tokom tog vremena. Kocke su statične te je zapremina koju one zauzimaju konstantna. U nekom trenutku dogodiće se sudar između sfere koja se kreće i kocki. Da bismo utvrdili da li se to dešava u tekućem koraku, računamo geometrijski presek kapsule i kocki. Ako je prazan, onda nije sudara, a ako nije prazan, onda se desio sudar i u toj situaciji dinamički sistemi mogu da reaguju na odgovarajući način, na primer, obaranjem kocki i izmenom putanje lopte.



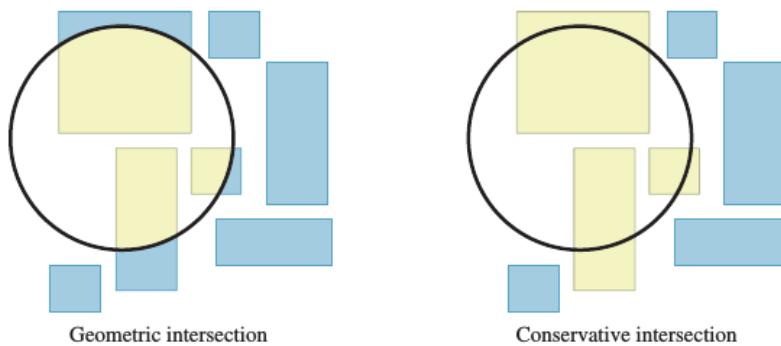
Slika 9.2: Tokom proizvoljnog vremenskog intervala sfera koja se kreće konstantnom brzinom pravi trag u obliku kapsule. Ako presek kapsule i kocki tokom tog intervala nije prazan, onda se u tom vremenskom intervalu desio sudar.

U ovom primeru bilo bi racionalno izračunati kapsulu za svaki korak, a zatim iterirati kroz sve kocke i testirati da li postoji presek sa nekom od njih. Ovo ne bi bilo isplativo u slučaju scene od milion geometrijskih primitiva. Bilo bi dobro da vreme izvršavanja algoritma za određivanje preseka bude manje od linearne u funkciji broja primitiva  $n$ . Očekujemo da korišćenje prostornih struktura podataka u slučaju scene koja sadrži  $n$  primitiva može da smanji cenu ispitivanja preseka na  $O(\log n)$  u prosečnom slučaju.

U opštem slučaju ne postoji najbolja prostorna struktura podataka. Različite strukture su pogodne za različite podatke i upite.

Metode koje implementiraju upite preseka se mogu definisati *precizno* da vraćaju tačan geometrijski presek ili *konzervativno* da vraćaju sve primitive

koje sadrže preseke. Drugi od predloženih načina je značajan jer je nekada teško efikasno predstaviti preseke među jednostavnim oblicima. Npr. ako bismo gledali presek lopte sa skupom trouglova on često sadrži mnoštvo trouglova koji su presečeni površinom lopte. Rezultujući skup nije moguće predstaviti mrežom trouglova.



Slika 9.3: Na slici levo prikazan je geometrijski presek lopte sa skupom kutija u 2D, a desno rezultat konzervativne metode koji je efikasnije izračunati.

## 9.1 Stabla

Kao što predstavljaju značajno ubrzanje u jednodimenzionom slučaju, stabla obezbeđuju značajno ubrzanje i u dve ili više dimenzija. U jednoj dimenziji, podela realne prave je jednostavna: posmatramo sve brojeve koji su veći ili manji od neke vrednosti podele  $v$ . U većim dimenzijama koriste se hiperravnini za podelu prostora i različiti izbori hiperravnini vode različitim strukturama podataka.

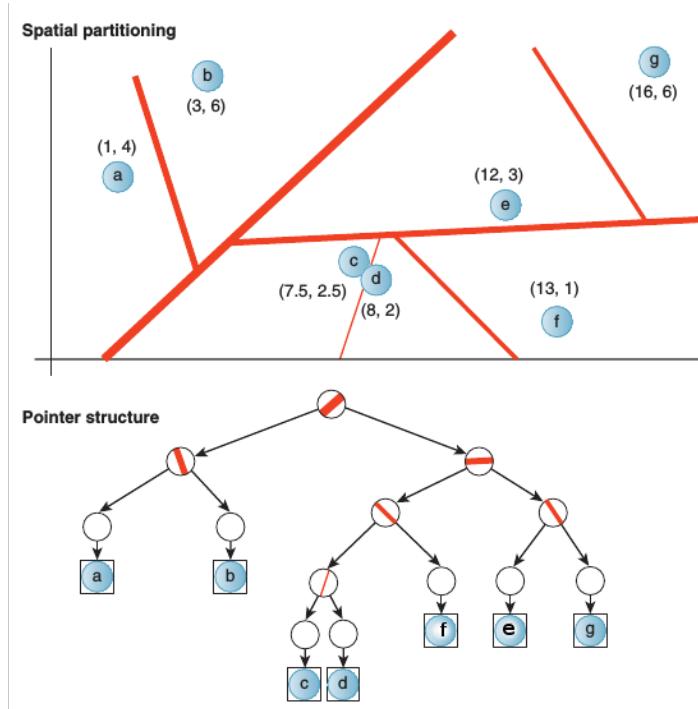
### 9.1.1 BSP stabla

Kod dvodimenzionalnog BSP stabla 2D prostor se deli **pravama razdvajanja** (slika 9.4), dok se kod trodimenzionalnog BSP stabla 3D prostor deli **ravnima razdvajanjima**.

BSP stablo odgovara rekurzivnom binarnom particionisanju prostora, ove particije dele prostor u konveksne potprostore (poligone, poliedre i njihove analogone u višim dimenzijama). Listovi BSP stabla odgovaraju potprostorima (koje ćemo u opštem slučaju zvati poliedrima), a unutrašnji čvorovi ravnima razdvajanja. Unutrašnjim čvorovima odgovaraju i konveksni prostori koji su unija njihove dece.

BSP stabla imaju logaritamsko vreme izvršavanja upita preseka pod određenim pretpostavkama. Pod upitom preseka podrazumevamo određivanje preseka neke geometrijske figure sa konveksnim poliedrom koji odgovara listu BSP stabla ili sa primitivom u listu.

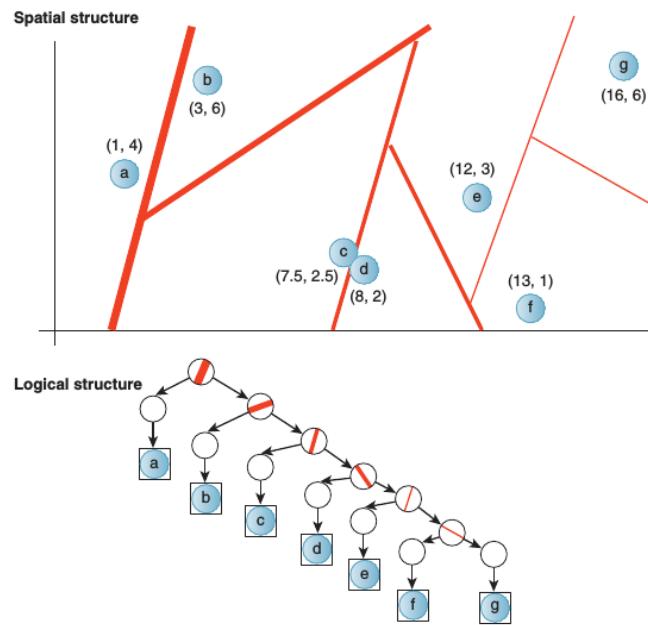
Iako svi listovi predstavljaju konveksne prostore, oni koji odgovaraju prostornim ekstremumima stabla odgovaraju prostorima beskonačne zapremine (npr. potprostor koji sadrži primitivu  $g$  na slici 9.4).



Slika 9.4: Prikaz 2D binarnog stabla prostornog particionisanja (BSP) u vidu particionisanja vrednosti sa pridruženim ključevima. Debljina linije particionisanja predstavlja dubinu tog čvora particionisanja u stablu; koren je predstavljen najdebljom linijom

Stablo koje sadrži  $n$  primitiva mora da čuva reference na sve primitive, te njegova veličina mora biti barem linearna po  $n$ . U nekim situacijama može se desiti da dobijemo jako neuravnoteženo stablo u kome većina čvorova ima samo jednu granu (slika 9.5)

Bitno je napomenuti da ne postoji gornja granica za veličinu stabla. Jedan od razloga za to je taj što se particije mogu dodavati nezavisno od primitiva u slučaju da očekujemo da nove primitive budu naknadno dodata. Ipak u većini algoritama podrazumevaćemo da je stablo izgrađeno pod nekim razumnim pretpostavkama i da ima manje od  $2n$  čvorova. U balansiranom BSP stablu moguće je locirati list korišćenjem  $O(\log n)$  poređenja, dok se ovo vreme može povećati na  $O(n)$  ako stablo nije balansirano. Vreme izvršavanja upita preseka raste sa porastom veličine prostora koji je pokriven geometrijom upita. Srećom ovaj rast je uglavnom u vremenu manjem od linearног. Osnovna pretpostavka je sledeća: ako zrak pređe veliko rastojanje pre nego što nađe na primitivu, on će najverovatnije proći kroz veliki broj particija, a ako pređe kratak put kroz mali broj njih. Dobra intuicija je da vreme izvršavanja može da raste logaritamski sa opsegom geometrije upita, jer očekujemo da za ravnomerno raspodeljene primitive particije formiraju balansirano stablo.



Slika 9.5: Degenerisano BSP stablo sa istim asymptotskim ponašanjem kao i lista. Loše performanse stabla slede iz neefikasnog izbora particija. Ova situacija se može desiti i ako imamo dobru strategiju izgradnje stabla ako se primitive pomeraju nakon postavljanja particija. Postoje i još gore situacije u kojima je većina particija prazna.

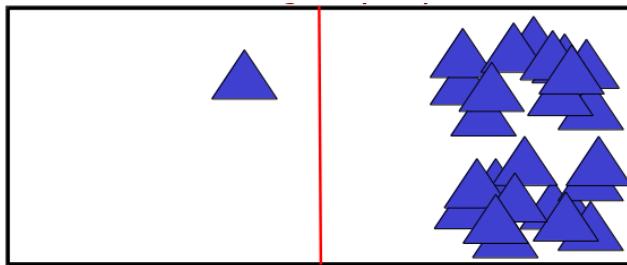
### 9.1.2 Izgradnja BSP stabla

Izbor dobrih ravnih razdvajanja nije jednostavan. Teorijski, postoji beskonačno mnogo ravnih između kojih treba napraviti izbor. Pritom sam izbor zavisi od vrste upita, raspodele podataka i od toga da li želimo da optimizujemo najgori slučaj, najbolji slučaj ili prosečni slučaj. Ako se stablo gradi samo jednom, na početku, onda se može dozvoliti da algoritam konstrukcije stabla bude i veće složenosti. Ukoliko se stablo gradi tokom rada programa ili će biti često menjano tokom rada programa, važno je minimizovati vreme koje zajedno troše izgradnja stabla i izvršavanje upita.

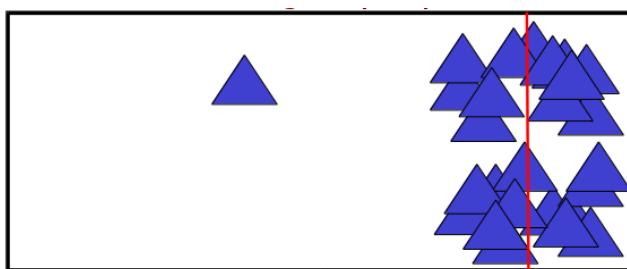
Da bismo pojednostavili problem razmatranja prevelikog broja mogućnosti za ravnih razdvajanja, često je zgodno uvesti neka ograničenja i na taj način razmatrati manji skup opcija. Ovim se, takođe, smanjuje broj bitova koji je potreban za predstavljanje particija, čime se smanjuje cena skladištenja same strukture. Jedno ovakvo ograničenje bilo bi da se razmatraju samo particije poravnate sa koordinatnim osama. Ovakvo BSP stablo naziva se **kd stablo** (engl. *kd tree*). Osa duž koje vršimo podelu može se birati na osnovu podataka.

Bez obzira na postavljena ograničenja, i dalje postoji veliki broj mogućnosti kako izvršiti podelu: neke od često korišćenih mogućnosti su:

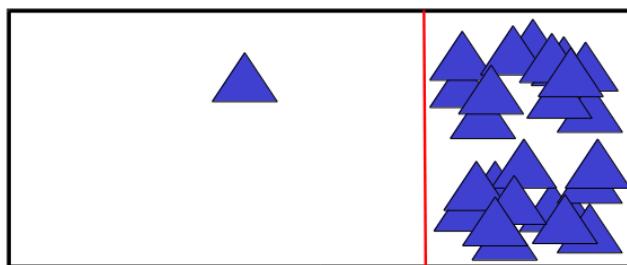
- podela po srednjoj vrednosti u odnosu na opseg (slika 9.6),
- podela po srednjoj vrednosti u odnosu na primitive (slika 9.7),



Slika 9.6: Podela po sredini opsega; zrak sa jednakom verovatnoćom može da uđe u levu i desnu stranu, ali je cena ulaska u desnu mnogo veća

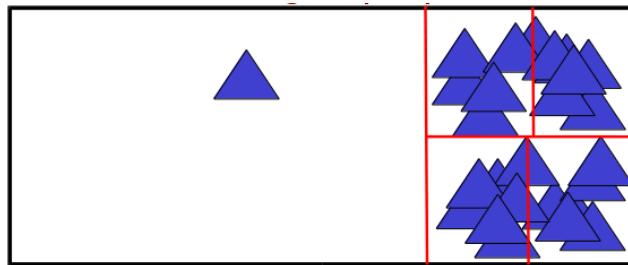


Slika 9.7: Podela po vrednosti medijane pozicija primitiva; cena ulaska u svaku od strana je otprilike jednaka, ali je verovatnoća ulaska zraka u levu stranu mnogo veća – bolja ideja

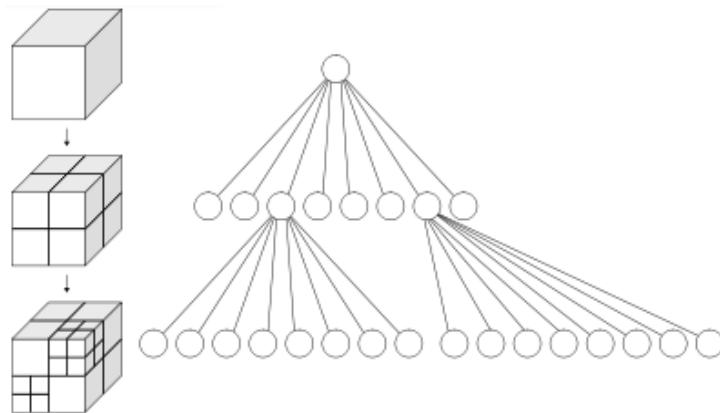


Slika 9.8: Podela u skladu sa optimalnom cenom (korak 1); levi sin ne zahteva dalje podele; desni sin se deli prema medijani i svaki od njegovih sinova dalje

Podela duž sredine opsega svih primitiva u čvoru duž svih koordinatnih osa daje skup ugnježdenih  $k$ -kocki. Tj. svaka podela prolazi kroz centar roditeljskog poliedra. U 3D prostoru ovo se naziva **oktri** (engl. oct tree) i prikazan je na slici 9.10. On se najčešće predstavlja korišćenjem 8 pokazivača ka deci. Ne moraju sve ćelije biti iste veličine, već se gušći delovi scene sastoje od većeg broja kocki. Oktri se konstruišu odozgo naniže: najpre pronađemo kocku koja ograničava celu scenu: ona odgovara korenu stabla koje sadrži sve primitive. U svakoj iteraciji, partitionišemo tekući čvor na 8 oktanata i delimo primitive u oktante: ako se neka primitiva prostire u više oktanata, dodeljujemo je u oba. Rekurzivno nastavljamo sve dok ne dođemo do maksimalne dozvoljene dubine stabla ili ako oktant sadrži dovoljno malo primitiva.



Slika 9.9: Podela u skladu sa optimalnom cenom (korak 2)



Slika 9.10: Oktri. Na levoj strani prikazana je rekurzivna podela kocke na 8 kocki. Desno je prikazan odgovarajući oktri.

Ispitivanje preseka počinje u korenu. Ako je čvor list, ispitati presek sa svim primitivama u tom čvoru, inače iteriramo kroz sinove tog čvora i računamo preseke zraka i u celije koja odgovara sinu: ako postoji presek, nastavljamo rekurzivno na tom sinu.

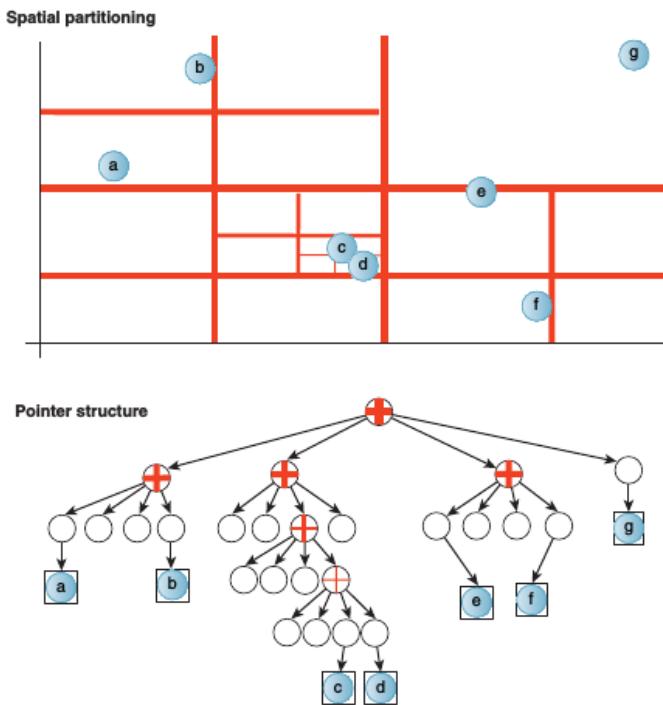
Njegov analogon u 2D je **kuodtri** (engl. quad tree), slika 9.11, a naravno može se proširiti i u 1D ili 4D i još veće dimenzije.

### 9.1.3 Hiperarhija graničnih opsega

Ideja **graničnih opsega** je umetnuti složene objekte u jednostavnije, kao što su sfere, kocke,...(slika 9.12). Ako granični opseg nije vidljiv, neće biti vidljiv nijedan objekat u njegovoј unutrašnjosti. Na koji način ovo može ubrzati rejt casting algoritam? Prvo se može proveriti da li postoji presek zraka sa graničnim opsegom objekta ili presek većeg broja zraka (koji određuju zapanju pogleda) sa graničnim opsegom objekta. Ukoliko presek ne postoji, možemo izuzeti iz daljeg razmatranja potencijalno veoma složeni objekat.

**Hiperarhija graničnih opsega** (engl. Bounding Volume Hierarchy, skraćeno BVH) je prostorno stablo nastalo rekurzivnim ugnježdavanjem graničnih opsega, kao što su recimo pravougaonici čije su stranice poravnate sa koordinatnim osama.

Za razliku od BSP stabala, ovaj tip stabla daje tešnje granice za klasterne



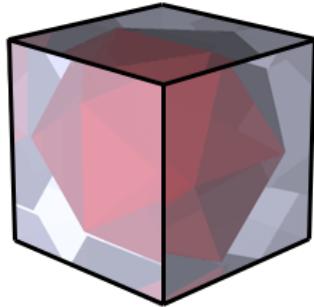
Slika 9.11: Kuodtri. Zbog blizine primitiva  $c$  i  $d$  na ovoj slici, stablo može biti velike dubine. U ovom slučaju bi bolja strategija izgradnje stabla bila da napravimo čvor drugog nivoa koji sadrži ove dve primitive, nego da vršimo podelu sve dok svaki list ne sadrži samo jednu primitivu. Ako bi  $c$  i  $d$  predstavljali klaster od 1000 primitiva, ne bi postojalo efikasno stablo – ili bi puno primitiva pripadalo istom listu, ili bi postojao dugačak lanac unutrašnjih čvorova.

primitive i ima konačnu zapreminu. On se često gradi nakon izgradnje BSP stabla, rekursivnom izgradnjom graničnih opsega, od listova ka korenju. Opsezi bratskih čvorova se često preklapaju u ovoj shemi (slika 9.13). Kao i kod BSP stabala, mogu se kreirati stabla koja razdvajaju primitive da bi se obezbeđio razdvojeni prostor za bratske čvorove.

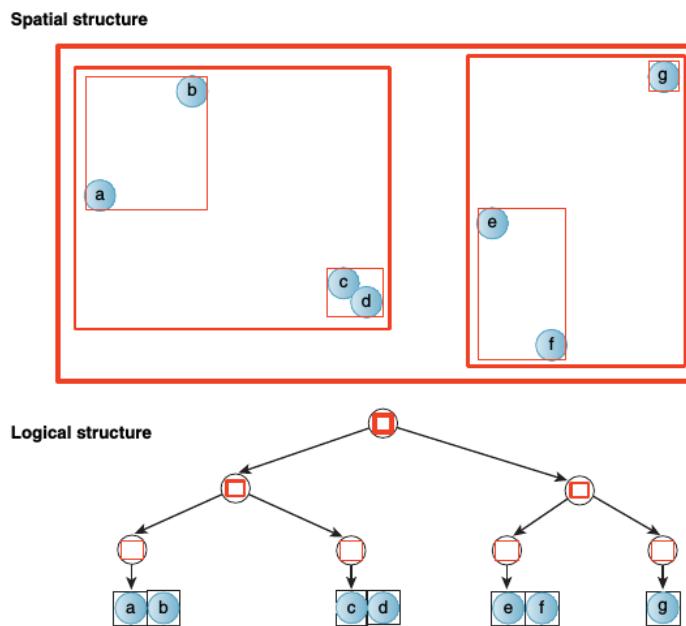
Za granične opsege se najčešće biraju lopte ili kocke poravnate sa koordinatnim osama, jer omogućavaju kompaktno čuvanje i jednostavne upite preseka.

## 9.2 Mreža (rešetka)

Umesto ograničavanja objekata pristupom odozdo naviše, možemo prostor partionisati na ćelije jednakе veličine. **Mreža** (engl. grid) predstavlja uopštenje jednodimenzionog niza. Njom se konačni pravougaoni region deli na ćelije jednakе veličine. Svaka tačka  $P$  u datom opsegu se nalazi u *tačno jednoj* ćeliji mreže. Višedimenzioni indeks te ćelije se dobija oduzimanjem minimalnog temena mreže od tačke  $P$ , deljenjem sa opsegom jedne ćelije i zaokruživanjem naniže (slika 9.14). Za mrežu čije se minimalno teme nalazi u koordinatnom



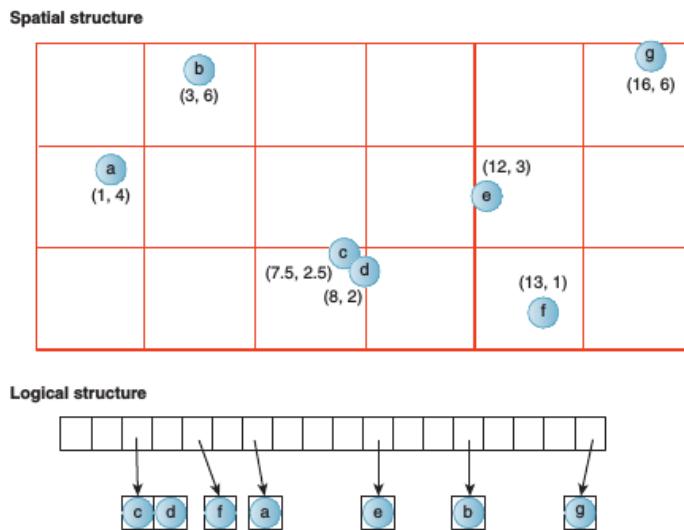
Slika 9.12: Granični opseg složenog objekta.



Slika 9.13: Prikaz dvodimenzione hijerarhije graničnih opsega poravnatih sa koordinatnim osama. Ona predstavlja alternativu BSP stablu koje obezbeđuje tešnje granice ali ne dozvoljava ažuriranje bez modifikacije strukture stabla.

početku i čije su ćelije veličine 1 u svakoj dimenziji, ovo se svodi samo na zaokruživanje naniže koordinata tačke  $P$ . Stoga, ako se veličina ćelije održava konstantnom, ćelija kojoj primitiva pripada se može pronaći u konstantnom vremenu, ali će prostor potreban za skladištenje ove strukture biti proporcionalan sumi broja primitiva i zapremine mreže. Upiti preseka se u opštem slučaju izvršavaju u vremenu proporcionalnom zapremini oblika upita (ili dužini, recimo za zrak).

Mreže predstavljaju jednostavnu strukturu podataka, efikasnije se konstruišu u odnosu na stabla, a pritom operacije dodavanja novih primitiva i brisanja nisu zahtevne ni vremenski ni prostorno. Stoga su pogodne za *di-*



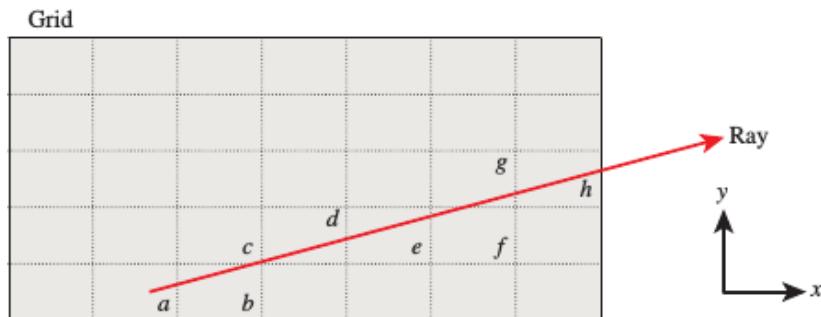
Slika 9.14: Prikaz 2D mreže koja čuva vrednosti sa pokazivačima na ključeve. Logička struktura “odvija” dvodimenzionalni niz uređen prema vrstama, tako da počinje od donjeg reda.

namičke podatke i često se koriste za utvrđivanje da li je došlo do sudara i slične upite.

Podešavanje veličine celije mreže, odnosno broja celija za dati opseg mreže zahteva dobro poznavanje tipova upita koji će biti izvršavani, kao i poznavanje prostorne raspodele podataka.

### 9.2.1 Presek sa zrakom

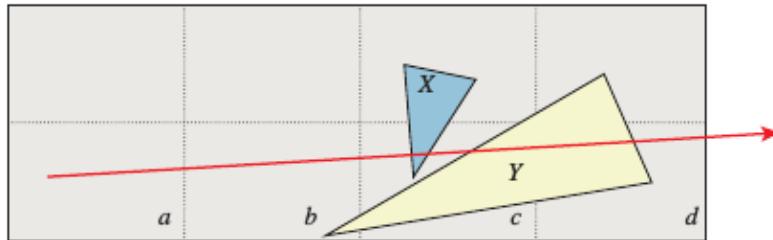
Da bi se pronašao prvi presek zraka sa primitivom u mreži, algoritam za određivanje preseka mora da prođe kroz celije mreže u poretku u kom zrak ulazi u njih (slika 9.15).



Slika 9.15: Praćenje zraka kroz 2D mrežu. Da bi se korektno pratio zrak, potrebno je posetiti celije a, b, c, d, e, f, g i h u tom redosledu.

Obzirom da se primitive mogu protezati kroz više od jedne celije mreže, esencijalno je testirati samo preseke koji se javljaju u toj celiji u svakoj iteraciji.

Primer ove situacije prikazan je na slici 9.16. Posmatrajmo test preseka koji se javlja u iteraciji kada se nalazimo u ćeliji označenoj sa  $b$ . Obzirom da ćelije koje pokriva objekat  $Y$  uključuju ćeliju  $b$ , tokom ove iteracije testiraćemo  $Y$  u odnosu na zrak. Presek postoji, ali se ne nalazi u ovoj ćeliji već u ćeliji  $c$ . Ukoliko bi bio vraćen taj presek, izgubili bismo prvi presek koji se javlja između objekta  $X$  u ćeliji  $c$ .



Slika 9.16: Slučaj kada je esencijalno testirati samo presek koji se nalazi u ćeliji u kojoj se trenutno nalazimo.

Intuicija ukazuje da će se algoritam izvršavati u vremenu proporcionalnom broju ćelija mreže kojima se prolazi. Cena svake iteracije je konstantna, te je algoritam od praktične koristi u slučajevima kada ne očekujemo da zrak putuje predugo pre nego što udari u nešto. Mreža sa  $g$  podela duž  $k$  dimenzija sadrži  $O(g^k)$  ćelija. Najduži put zraka je dijagonalna mreža dužine  $O(g)$ . Za mrežu koja sadrži  $n$  primitive, u najgorem slučaju sve primitive se protežu kroz sve ćelije mreže duž dijagonale, a zrak ne preseca nijednu od njih. Cena upita preseka je stoga  $O(g \cdot n)$ . Naravno, za ovaku situaciju mreža predstavlja loš izbor. Stoga, ponašanje u najgorem slučaju je veoma različito od očekivanog ponašanja. Mreža je pogodnija za scenu koja sadrži približno ravnomerno raspodeljene primitive koje teže tome da stanu u ćeliju mreže.

Na upitima za velike scene sa neravnomernom prostornom raspodelom primitive, stabla pokazuju bolje asymptotsko ponašanje od mreža: kod stabla očekujemo logaritamsko vreme izvršavanja, dok je kod mreža očekivano vreme linearno. Ipak, kroz praznu ćeliju se može brzo proći. Stoga, ako linearan broj puta primenimo operaciju konstantne cene, možemo priuštiti prolazak velikog broja ćelija za svaki zrak. Za scene sa ograničenom veličinom, mreža može da pokaže bolje performanse od stabla za upite preseka, posebno ako je vreme potrebno za izgradnju strukture podataka ugrađeno u vreme izvršavanja.

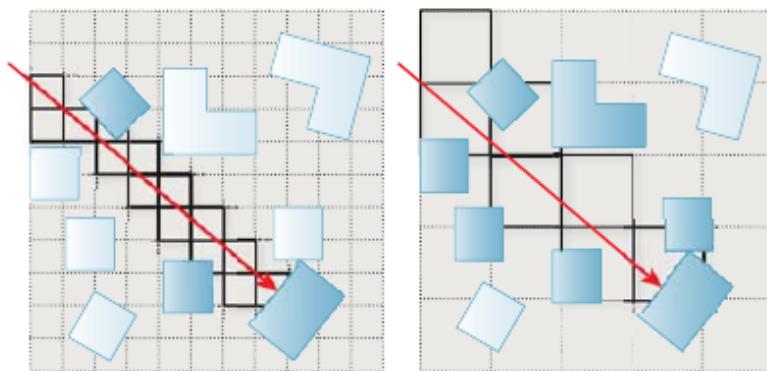
### 9.2.2 Odabir rezolucije mreže

Ako očekujemo mnogo preseka lopti ili kocki, trebalo bi podesiti veličinu mreže na osnovu očekivane veličine presečnog objekta. Cilj je da algoritam za određivanje preseka treba da ispita samo mali broj ćelija (možda 1 do 4).

Presek zraka na mreži traje linearno vreme u funkciji dužine zraka jer je broj ćelija mreže koje treba ispitati asymptotski proporcionalan dužini zraka. Takođe, traje linearno vreme u funkciji broja primitive u mreži koje se nalaze. Ovim se dolazi do dva raličita cilja a to su: minimizovati broj testiranja pre-

seka sa mrežom i minimizovati broj testiranja preseka sa primitivama. Neka razumna pretpostavka je da je utvrđivanje preseka sa mrežom manje skupo od utvrđivanja preseka sa primitivom.

Ako je  $g$  veliko, kvadrati mreže su mali. Ovim se smanjuje broj primitiva koje treba testirati u svakoj nepraznoj ćeliji, ali se povećava broj ćelija koje treba ispitati. Ovaj izbor je dobar za guste scene. Ako je  $g$  malo, kvadrati mreže su veliki. Kod ovakvih mreža zrak se brzo kreće kroz velike prazne prostore, ali se povećava broj primitiva u svakoj nepraznoj ćeliji sa kojom treba vršiti testiranje. Ovo je pogodnije za retke scene.



Slika 9.17: Levo: veliko  $g$  daje ćelije male veličine. Iterator zraka mora da prođe kroz mnogo ćelija ali će mnoge biti prazne. Desno: malo  $g$  daje ćelije velike veličine. Zrak iterira kroz mali broj ćelija, ali one sadrže veliki broj primitiva.

### 9.3 Pitanja

- 9.1 Šta računaju konzervativne metode za rešavanje upita preseka? Zašto su značajne?
- 9.2 Koja je očekivana cena upita preseka na BSP stablima, a koja u najgorem slučaju?
- 9.3 Kako se biraju particije kod  $kd$ -stabla? Kako se najčešće biraju ravni razdvajanja kod  $kd$ -stabala?
- 9.4 Kako izgleda struktura podataka oktri, a kako kuodtri?
- 9.5 Koja je razlika između korišćenja BSP stabala i hijerarhija graničnih opsega?
- 9.6 Koja je struktura podataka pogodnija za dinamičke podatke: BSP stabla ili mreža?
- 9.7 Koja je očekivana cena upita preseka na mreži, a koja u najgorem slučaju?
- 9.8 Uporediti mreže sa malom veličinom ćelije sa mrežama kod kojih je veličina ćelije velika.

## *Glava 10*

---

# **Monohromatska i hromatska svetlost**

---

- Svetlost je oblik elektromagnetskog zračenja: vidljiva svetlost ima talasnu dužinu između 400 i 700 nanometara.
- Koristimo različite reči pri opisivanju boja objekata koji emituju svetlost i onih koji reflektuju svetlost. Na primer, reći ćemo da je neki objekat braon boje, ali gotovo nikada nećemo reći da je svetlost braon boje
- Svetlost je značajna i interesantna tema za fiziku, psihologiju, kognitivne nauke, umetnost, dizajn, računarsku grafiku.
- Boja objekta ne zavisi samo od samog objekta već od izvora svetla koji ga obasjava, od boje okoline i od ljudskog vizuelnog sistema. Neki objekti (tj. materijali) reflektuju svetlost (zid, klupa, papir), dok neki propuštaju svetlost (staklo). Kod nekih objekata se javlja apsorbovanje, kod nekih razbacivanje svetlosti a kod nekih refrakcija (menja se pravac).
- Značaj boja u računarskoj grafici: estetski razlozi, realizam, naglašavanje, poboljšana funkcionalnost i interfejs.

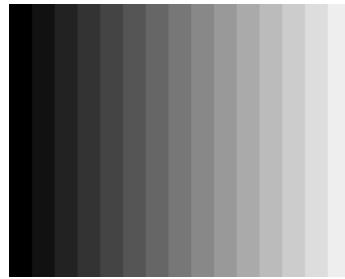
### **10.1 Monohromatska svetlost**

- Monohromatska (ili monospektralna) svetlost je npr. ono što vidimo na crno-belom televizoru/monitoru. Posmatrač nema osećaj za iskustvo koje se vezuje za boje (npr. za crvenu, plavu, zelenu).
- Kvantitet svetlosti je jedino svojstvo monohromatske svetlosti.
- Kvantitet svetlosti može biti razmatran kao fizički pojam energije (kada se govori o **intenzitetu** ili **luminaciji**) ili kao psihološki fenomen (kada se govori o **sjajnosti** (eng. brightness)). Ovi koncepti su bliski i povezani, ali nisu identični.
- Crno-beli monitor može da proizvede mnogo različitih intenziteta svetlosti na poziciji jednog piksela. Mnogi uređaji (npr. štampači) mogu da proizvedu samo dva intenziteta boje (crna i bela). Određene tehnike

omogućavaju ovakvim uređajima da simuliraju dodatne nivoje intenziteta.

## 10.2 Izbor intenziteta

- Prepostavimo da hoćemo da reprezentujemo 256 intenziteta na skali od 0 do 1.



Slika 10.1: Nijanse sive boje

- Raspodela treba da bude ravnomerna u nekom smislu, ali treba voditi računa o sledećem: ljudsko oko je osetljivo na odnose intenziteta svetlosti, pre nego na absolutne intenzitete svetlosti. Ljudskom oku se čini da se intenziteti 0.10 i 0.11 razlikuju isto kao i intenziteti 0.50 i 0.55. Dakle, intenzitet treba da bude opisan logaritamskom (a ne linearном skalom) da bi se dobili jednakim koracima u sjajnosti.
- Da bi se odredilo 256 intenziteta počev od  $I_0$  pa do 1.0 potrebno je da postoji vrednost  $r$  koja odražava odnos dva susedna intenziteta:

$$I_0 = I_0, \quad I_1 = rI_0, \quad I_2 = rI_1 = r^2I_0, \quad \dots, \quad I_{255} = r^{255}I_0 = 1$$

$$r = (1/I_0)^{1/255}, \quad I_j = r^j I_0 = I_0^{(255-j)/255}$$

- Generalno (za  $n$  intenziteta), važi:

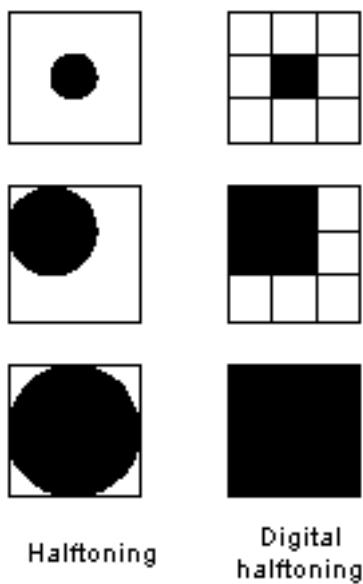
$$r = (1/I_0)^{1/n}, \quad I_j = r^j I_0 = I_0^{(n-j)/n}$$

- Minimalni intenzitet  $I_0$  je za monitore obično negde između 1/200 i 1/700 maksimalnog intenziteta.
- Prikazivanje opisanih intenziteta boja na CRT monitoru (ili na filmu) nije trivijalna stvar i zavisi od konkretnog CRT uređaja. Intenzitet koji CRT emituje je proporcionalan sa  $V^\gamma$ , pri čemu je sa  $V$  označen napon, a vrednost  $\gamma$  uzima vrednost iz skupa [2.35, 2.55]. Ovaj eksponent se naziva **gama** i smatra se merilom nelinearnosti uređaja za prikaz
- Često se koriste unapred izračunate vrednosti (za određivanje napona tj. vrednosti za jedan piksel). Korišćenje ovakvih unapred izračunatih tabela se zove **gama korekcija** (eng. gamma correction)

- Postoji industrijski standard za gama vrednost, ali razlike u procesu proizvodnje, godine, korisnička podešavanja sjajnosti dovode do toga da nijedna dva monitora nemaju identičnu vrednost za  $\gamma$ .
- Ljudsko oko može da razlikuje susedne površine za koje je odnos intenziteta veći od 1.01.
- Na osnovu te vrednosti može se odrediti minimum intenziteta koji mogu da daju prihvatljivu ahromatsku sliku. Za većinu uređaja, ta (idealizovana) vrednost je 64.

### 10.3 Polutoniranje

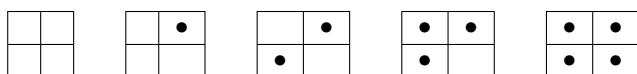
- **Polutoniranje** (engl. halftoning) je tehnika koja na bazi minimalnog broja nivoa intenziteta (npr. crna i bela boja) daje privid više različitih nivoa osvetljenosti.
- Tehnika je veoma stara i nekada se koristila u bojenju tekstila i platna; moderno polutoniranje u štamparstvo je uveo Stephen Hargan 1880. godine.
- Tehnika se uspešno koristi npr. u štampanju novina sa niskom rezolucijom.
- Suštinski, tehnika se u različitim kontekstima (i na različitim medijima) koristi tako što intenzitet boje (a time osvetljenost i veličina obojene površine) zavisi od dužine izlaganja svetlu, mastilu i sl.
- Digitalno polutoniranje je slično polutoniranju u kojem je slika dekomponovana na mrežu polutoniranih celija. Nijanse boja na slici se simuliраju popunjavanjem odgovarajućeg broja celija. Naredna slika pokazuje kako se polutoniranje realizuje digitalno:



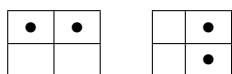
- Najčešći metodi polutoniranja su uzorkovanje, dithering i prostorni dithering (ili difuzija greške (engl. error diffusion)).

### 10.3.1 Uzorkovanje

- Polutoniranje generalno može da koristi ćelije različite veličine. Za razliku od toga, **uzorkovanje** (engl. patterning) koristi ćelije jednake veličine.
- Pogodno korišćenje pojedinog uzorka samo od crnih i belih ćelija stvara privid nijanse sive.
- Uzorkovanje proizvodi sliku čije su dimenzije  $n$  puta veće od polaznih dimenzija, gde je  $n$  dimenzija ćelije.
- Uzorkovanje popravlja vizuelnu rezoluciju, ali loše utiče na prostornu rezoluciju (gube se fini detalji) i zbog toga veličina uzorka/ćelije ne može da se uzima da bude prevelika.
- Za crnu i belu boju postoji 5 osnovnih  $2 \times 2$  uzoraka (za 5 nijansi sive):

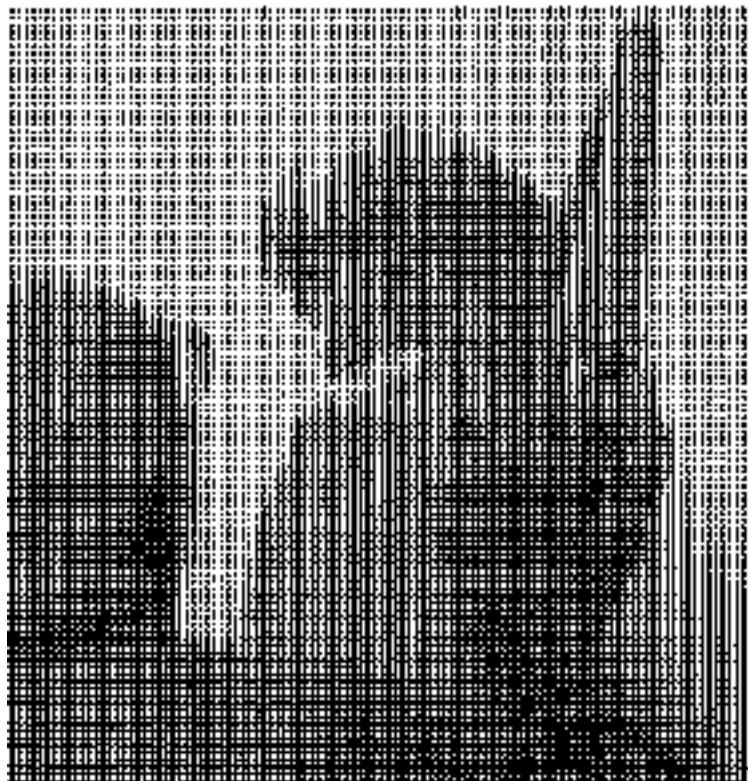


- ne smeju se koristiti sledeći uzorci (zbog vizuelnog utiska linija):



- za ćelije veličine  $n$  treba da postoji  $n + 1$  uzoraka
- Ilustracija korišćenja uzorkovanja za ćelije veličine  $4 \times 4$ :





### 10.3.2 Dithering

- Dithering (podrhtavanje, treperenje) je jedna od tehnika za digitalno polotoniranje.
- Dithering daje sliku istih dimenzija kao i polazna.
- Sistem obično automatski primenjuje ovu tehniku kada su karakteristike uređaja za prikaz podešene na 256 boja ili manje.
- Dithering obično uvećava veličinu slike (u bajtovima), ali je poboljšan izgled vredan te cene.
- Dithering „razbacuje“ piksele različite boje po slici kako bi izgledalo da postoje međuboje na slici sa ograničenim brojem boja.
- Slika se deli na blokove male veličine i definiše se dither matrica koja sadrži različitu vrednost praga za svaki od piksela tog bloka; za svaki piksel sa slike proverava se da li je intenzitet veći od vrednosti praga dither matrice za taj piksel, onda se piksel u izlaznoj slici popunjava.
- Dithering može da koristi jasan uzorak ili slučajan šum.
- U prostornom ditheringu se greška koja nastaje za jedan piksel distribuira okolnim pikselima pre nego što se proverava vrednost praga; postoji uzorak koji u tome daje optimalnu vrednost u smislu minimizacije stvaranja efekta teksture.



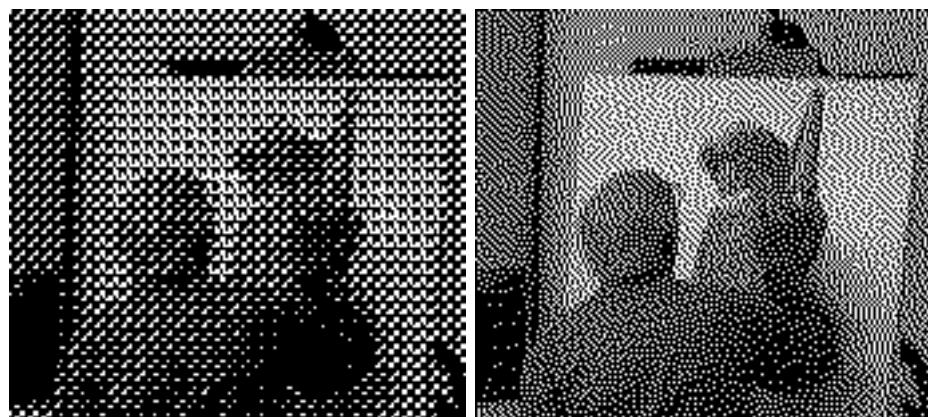
Bez dithering-a (gif slika, 256 boja, 6Kb)



Dithering sa uzorkom (gif slika, 4 boje, 1.31Kb)



Dithering sa šumom (difuzija greške) (gif slika, 4 boje, 2.36Kb)

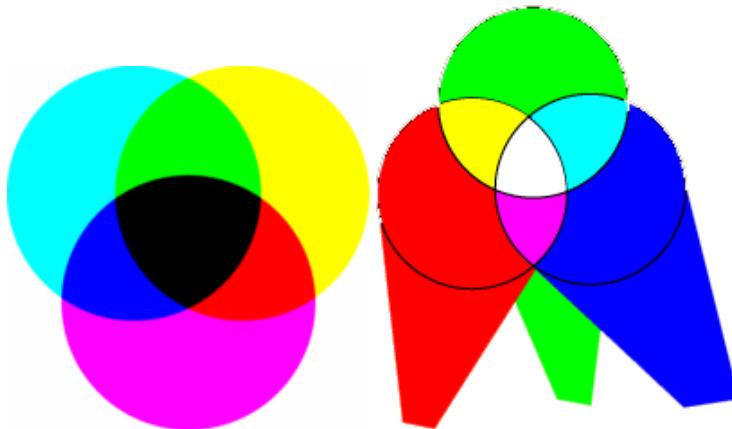


Dithering sa uzorkom i sa šumom

## 10.4 Hromatska (obojena) svetlost

- Većina ljudi je sposobna da doživi boje – to je osećaj koji se javlja dok posmatramo različite spektralne kombinacije svetlosti. Za svetlost talasne dužine od oko 400nm većina ljudi će reći da je plava, dok će za svetlost talasne dužine oko 700nm reći da je crvena.
- Vizualne senzacije uzrokovane obojenom svetlošću su mnogo bogatije nego one uzrokovane monohromatskom.
- Postoji više modela za opisivanje obojene svetlosti.
- Skoro svi modeli za opisivanje svetlosti zasnovani su na (nekim) trima nezavisnim karakteristikama – koje karakteristike su u pitanju, to je stvar izbora.

- Postoji razlika u tome kako se meša svetlost, a kako mešaju boje (pigmenti) – ova razlika je čisto fizička. Ako bismo uperili crveno i zeleno svetlo na ravnomerno reflektujući komad belog papira, reflektovana svetlost bi delovala žuto. Za razliku od toga, ako nanesemo crvenu i zelenu farbu na parče belog papira, apsorbovala bi se sva svetlost osim one koju doživljavamo kao crvenu, odnosno zelenu, i one bi zajedno apsorbovale skoro svu svetlost; tako bismo dobili braon boju. Dakle, farbe se ponašaju kao filter između posmatrača i izvora svetlosti (površi koja reflektuje svetlost). Ova dva fenomena imaju donekle zbirajuća imena, prvi nazivamo *dodavanjem boja*, a drugi *oduzimanjem boja*. U stvari, spektar je taj koji se dodaje odnosno filtrira, a mehanizam percepcije boja ostaje neizmenjen.



Slika 10.2: (a) Propuštanje svetlost kroz nekoliko filtera (oduzimanje boja); (b) "bacanje" različitih svetlosti na jedno isto mesto (dodavanje boja)

- Sa ekspanzijom dijalog prozora za izbor boja sa RGB slajderima, vlada uverenje da je svaka boja mešavina crvene, zelene i plave boje. Ipak, postoje mnoge boje koje se ne mogu dobiti mešanjem crvene, zelene i plave farbe, odnosno crvene, zelene i plave svetlosti. Tačno je da se mnoge boje mogu dobiti na ovaj način, ali ne sve.

#### 10.4.1 CIE model

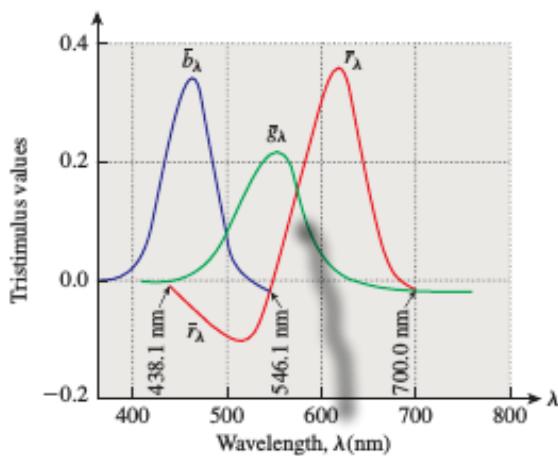
- Boja kao što je recimo narandžasta ne može se izraziti kombinovanjem svetlosti crvene, zelene i plave boje. Ipak, narandžasta boja izgleda kao mešavina jednakih količina crvene i zelene boje, kada se malo "udaljimo" od plave boje, odnosno ako bismo pokušali ovo da iskažemo jednačinom, imali bismo:

$$\text{orange} = 0.45\text{red} + 0.45\text{green} - 0.1\text{blue}$$

Naravno, ne možemo oduzeti plavu boju ukoliko nije prisutna, ali je možemo dodati narandžastoj boji. Dakle važi:

$$1.0orange + 0.1blue = 0.45red + 0.45green$$

u smislu da mešavine boja sa leve i sa desne strane proizvode istu senzaciju. Na ovaj način možemo odrediti koje mešavine crvene, zelene i plave boje su potrebne da bi se dobila svetlost proizvoljne boje.



Slika 10.3: Funkcije odgovarajućih boja kojima se za svaku talasnu dužinu zadaje količina crvene, zelene i plave svetlosti koju treba pomešati da bi se proizvela senzacija jednakog svetlosti talasne dužine  $\lambda$ . Bar jedan koeficijent je negativan za mnoge vrednosti svetlosti što ukazuje na nemogućnost da se ove boje proizvedu mešanjem crvene, zelene i plave boje

- 1931: Commission Internationale de l'Éclairage (CIE)<sup>1</sup> definisao je tri standardne primitive koje je nazvao **X**, **Y** i **Z**, sa svojstvom da trougao sa ova tri temena uključuje sve moguće vizuelne senzacije. Da bi ovo bilo moguće bilo je neophodno kreirati primitive koje imaju *negativne* regione u svom spektru, odnosno koje ne odgovaraju fizički ostvarivim izvorima svetlosti. I pored toga ovakve primitive imaju neke prednosti:

- ako proizvoljni izvor svetla  $T$  zapišemo kao:

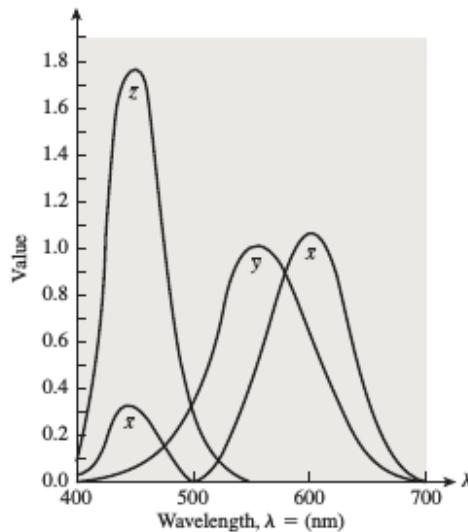
$$T = c_X \mathbf{X} + c_Y \mathbf{Y} + c_Z \mathbf{Z}$$

broj  $c_Y$  doživljavamo kao intenzitet svetla.

- Funkcije koje odgovaraju bojama za **X**, **Y** i **Z** su svuda nenegativne, te se sve boje mogu predstaviti kao nenegativne linearne kombinacije primitiva
- Obzirom da se crvena, zelena i plava primitiva mogu odrediti kao tačke u  $XYZ$  prostoru (tj. kao linearne kombinacije **X**, **Y** i **Z**), svaka njihova kombinacija se takođe može izraziti na ovaj način, te stoga postoji direktna veza između  $XYZ$  koeficijenata i koeficijenata u

<sup>1</sup>Medjunarodna komisija o osvetljenju

sistemu gde se sve boje zadaju kao kombinacije crvene, zelene i plave boje (tzv. *RGB* modela koji će biti kasnije detaljno opisan) i obratno.



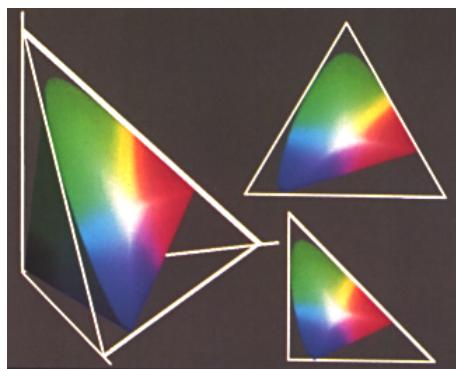
Slika 10.4: Funkcije koje odgovaraju bojama  $\bar{x}$ ,  $\bar{y}$  i  $\bar{z}$  za CIE primitive

- Pretpostavimo da za svetlost  $C$  važi  $C = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z}$ . CIE definije brojeve koji su nezavisni od ukupne sjajnosti deljenjem sa  $X + Y + Z$ , odnosno razmatramo koeficijente:

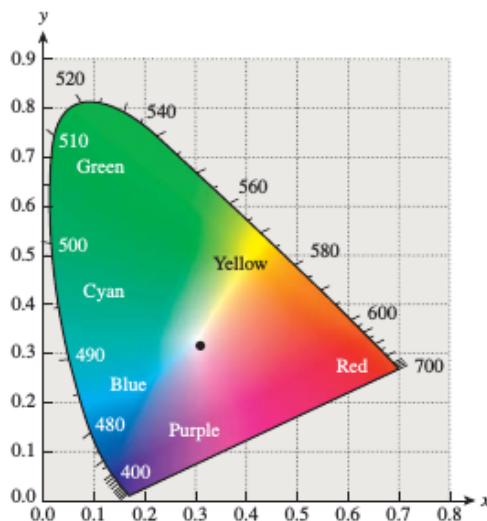
$$\begin{aligned}x &= \frac{X}{X + Y + Z} \\y &= \frac{Y}{X + Y + Z} \\z &= \frac{Z}{X + Y + Z}\end{aligned}$$

Ako se duplira dolazno svetlo, duplira se i svaka od vrednosti  $X$ ,  $Y$  i  $Z$ , ali i suma  $X + Y + Z$ , pa koeficijenti  $x$ ,  $y$ ,  $z$  ostaju nepromenjeni.

Primetimo da je suma  $x+y+z = 1$ , te ako znamo  $x$  i  $y$  možemo izračunati  $z$ . Stoga se skup boja nezavisnih od intenziteta može nacrtati samo u  $xy$  ravni – rezultat je **CIE dijagram obojenosti**. Boje koje su povezane intenzitetom se ne prikazuju (na primer braon boja je narandžasto-crvena za malu vrednost intenziteta). Postoji beskonačan broj ravni koje se projektuju na ravan  $X + Y + Z = 1$  čije se sve boje razlikuju, te ovo nije paleta sa svim bojama. Primetimo da su  $X$  i  $Y$  izabrani tako da dijagram bude tangentan na  $x$  i  $y$  osu. Dijagram ima oblik potkovice. Blizu centra "potkovice" nalazi se **izvor svetlosti**  $C$  (engl. illuminant) koji predstavlja standardnu referencu za "belo" na dnevnoj svetlosti. Nažalost ona ne odgovara vrednostima  $x = y = z = 1/3$ , ali je blizu te tačke.



Slika 10.5: Nekoliko pogleda na ravan  $x+y+z = 1$  u CIE prostoru; dole desno data je projekcija na  $xy$  koordinatnu ravan



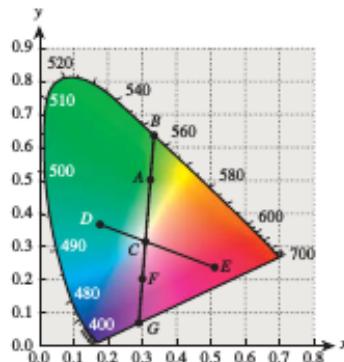
Slika 10.6: CIE dijagram obojenosti. Granica se sastoji od boja koje odgovaraju monospektralnoj svetlosti date talasne dužine, odnosno monohromatskim bojama, prikazanim u nanometrima. Tačke na duži koja povezuje početak i kraj spektralnih tačaka su nespektralne: predstavljaju mešavinu dve monohromatske boje. Tačka u centru je standardna bela svetlost koju nazivamo izvor svetlosti  $C$

- Primetimo da ako znamo  $x$  i  $y$  možemo izračunati  $z$  kao  $1-x-y$ , ali nam ovo ne omogućava da povratimo vrednosti  $X$ ,  $Y$  i  $Z$ . Za to nam treba još neka dodatna informacija. Obično koristimo i vrednost intenziteta  $Y$  da bismo izračunali ove vrednosti na sledeći način:

$$X = \frac{x}{y} Y, \quad Y = Y, \quad Z = \frac{1-x-y}{y} Y$$

- CIE dijagram obojenosti ima različite primene:

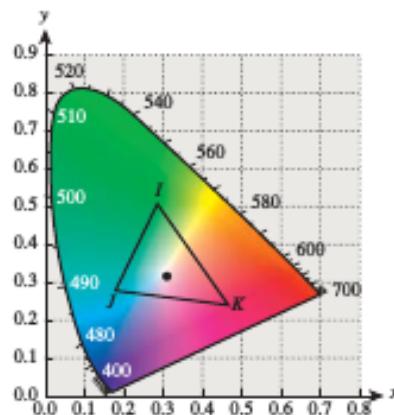
- može se koristiti za definisanje **komplementarnih boja**: boje su komplementarne ako se njihovim kombinovanjem može dobiti izvor svetlosti  $C$ ; ako bismo zahtevali da mešavina bude pola-pola, onda za neke boje ne bi postojale komplementarne.
- dijagram omogućava definisanje **energetske čistoće** (engl. excitation purity): ovaj pojam odgovara zasićenosti. Tačka  $A$  sa slike 10.7 se može predstaviti kombinovanjem izvora svetlosti  $C$  sa bojom  $B$  koja ima čist spektar. Što je tačka  $A$  bliža tački  $B$ , to je ona više spektralno čista (zasićenija). Stoga možemo definisati energetsku čistoću kao količnik dužina duži  $AC$  i  $BC$ . Proširujemo definiciju i na tačku  $C$  čija se energetska čistoća postavlja na 0. Za neke boje, kao što je recimo  $F$ , zrak iz  $C$  kroz  $F$  seče granicu "potkovice" u nespektralnoj tački, ovakve boje nazivamo **nespektralnim**, ali odnos  $CF$  ka  $CG$  i dalje ima smisla i možemo razmatrati energetsку čistoću i za ovakve tačke
- dijagram se može koristiti i za utvrđivanje **skala** (engl. gamut): svaki uređaj koji proizvodi svetlost (kao LCD monitor) može da proizvede opseg boja koje se mogu prikazati na dijagramu. Boje van ove skale se ne mogu proizvesti na uređaju. Uređaj koji može da proizvede dve boje, može takođe da proizvede (podešavanjem količine svake od njih) i hromatografske vrednosti koje su konveksne kombinacije ove dve boje (slično za tri).



Slika 10.7: Boje na dijagramu obojenosti.  $D$  i  $E$  su komplementarne boje

#### 10.4.2 Modeli za rastersku grafiku

- Prikazaćemo nekoliko kolor modela koja se koriste za opis boja koje uređaji mogu da proizvedu. Svi ovi modeli su **ograničeni**, u smislu da mogu da opišu boje do određenog intenziteta – ovo odgovara fizičkim karakteristikama mnogih uređaja: npr. za LCD monitor postoji maksimalna vrednost sjajnosti koju može da proizvede
- Izbor kolor modela može biti motivisan jednostavnosću (kao u RGB modelu), jednostavnim korišćenjem (HSV i HSL model) ili određenim



Slika 10.8: Mešanje boja na dijagramu obojenosti. Boje na pravoj  $IJ$  se mogu kreirati mešanjem boja  $I$  i  $J$ ; sve boje u trouglu  $IJK$  se mogu kreirati mešanjem boja  $I$ ,  $J$  i  $K$ .

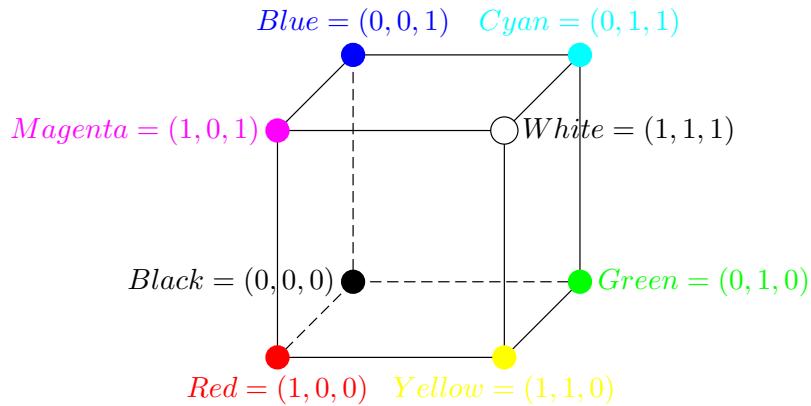
inženjerskim interesom (kao YIQ model koji se koristi za emitovanje TV signala ili CMY model za štampu)

- Kolor model za rastersku grafiku je specifikovan 3D koordinatnim sistemom i podoblašću oblasti svih vidljivih boja (na primer, RGB kolor model je određen kockom).
- Svrha kolor modela je da omogući jednostavno adresiranje boja iz nekog skupa (na primer, spektra CRT uređaja).
- Najčešće korišćeni modeli su
  - RGB i HSV (za CRT monitore)
  - YIQ (za TV kolor sistem)
  - CMY (za štampačke uređaje)
- Treba napomenuti da konvertovanje između modela u opštem slučaju ne mora da ima smisla: CMY je zasnovan na ideji primene mastila na beli papir koji je osvetljen određenim svetлом; reflektovano svetlo ne može biti sjajnije od tog osvetljenja. Konvertovanje boje koja se koristi u ultrasjajnom ekranu u CMY model se stoga ne može izvesti. Prema tome, prilikom stvaranja slike u računarskoj grafici, trebalo bi čuvati sliku bez gubitaka sa svim bitnim informacijama, da bi je bilo moguće kasnije konvertovati u neki drugi format

#### 10.4.3 RGB kolor model

- Jedinična kocka sa temenima:
  - crvena  $(1,0,0)$
  - zelena  $(0,1,0)$ ,

- plava (0,0,1)



- Sive nijanse se nalaze duž glavne dijagonale; pomeranjem sa dijagonale dobijaju se sve zasićenije boje
- RGB vrednosti se moraju interpretirati u zavisnosti od konkretnog uređaja; za prelazak sa RGB komponenti na nove RGB komponente može se odrediti matrica (3x3). Može se desiti da se neka RGB trojka za jedan ekran nakon množenja matricom transformacije transformiše u trojku čije su neke vrednosti veće od 1 ili manje od 0 – ovo ukazuje na to da postoji boja u opsegu prvog monitora koja je van opsega drugog monitora. Ovo se može rešiti ignorisanjem matrice transformacije ili zaokruživanjem ili nekim sofisticiranim metodama: npr kompesovanjem opsega prvog monitora skaliranjem u odnosu na centar opsega tog monitora, tako da se sve boje koje treba prikazati na prvom monitoru preslikavaju u boje koje je moguće prikazati na drugom monitoru.

#### 10.4.4 CMY i CMYK kolor model

- CMY kolor model se koristi za štampače, jer mastilo reflektuje neki deo primljene svetlosti, a apsorbuje drugi deo.
- Na belom papiru nije moguće dobiti belu boju u RGB modelu kombinovanjem najviših intenziteta za crvenu, zelenu i plavu
- Cyan mastilo apsorbuje crveno svetlo, ali reflektuje plavo i zeleno, magenta apsorbuje zeleno, a žuta plavo.
- Jedinična kocka sa temenima za cyan, magenta, žutu boju.
- Boje se opisuju kao mešavine cyan, magenta i žute boje. Kada se pomešaju dva mastila, svetlost koja se reflektuje je ona koju ne apsorbuje nijedna od njih. Stoga, mešavina cyan i magente apsorbuje i crvenu i zelenu boju, dajući pritom refleksiju plave svetlosti.

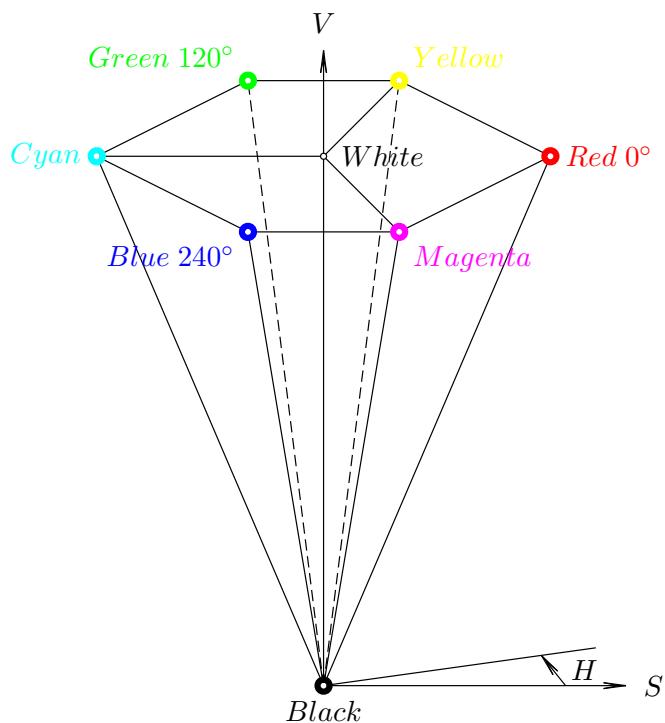
- Jednačina prelaska sa RGB komponenti na CMY komponente:

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

- U koordinatnom početku je bela umesto crne; boja (1,1,1) nije u stvarnosti veoma crna, jer mešavina cyan, magente i žute boje ne uspeva da apsorbuje svu svetlost, stoga štampači često imaju i četvrto mastilo – crno (u oznaci K) koje se koristi da zameni delove tamnije mešavine cyan, magente i žute boje

#### 10.4.5 HSV (HLS) kolor model

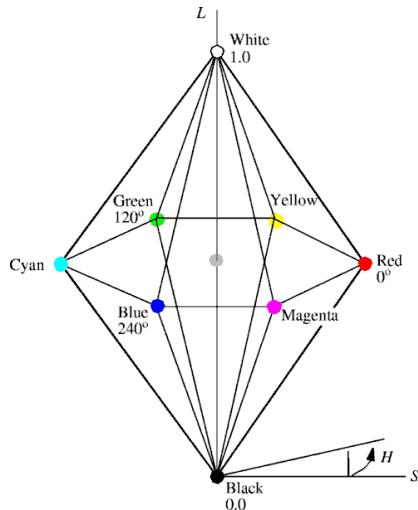
- RGB kolor model nije najzgodniji: individualne komponente ovog modela ne odgovaraju našim perceptivnim karakteristikama kao što su: koliko je svetla boja, koliko je zasićena i slično. Prilikom podešavanja boje od crvene ka narandžastoj malim povećanjem zelene komponente, boja postaje i svetlijia, a ono što bismo mi želeli jeste da samo izmenimo nijansu boje.



Slika 10.9: HSV kolor model

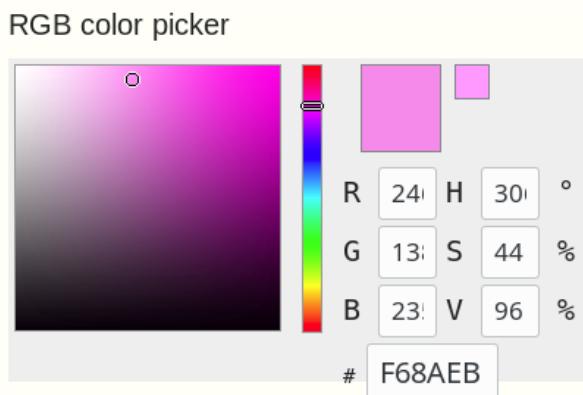
- Dva modela su zastupljena: HSV i HLS
- Komponente HSV kolor modela:

- **nijansa** (engl. hue) – karakteriše boju ( $0^\circ - 360^\circ$ , ili od 0 do 360)
  - **zasićenost** (engl. saturation) – karakteriše koliko se boja razlikuje od sive istog intenziteta (0%-100%, ili od 0 do 240)
  - **sjajnost** (engl. value, brightness) (0%-100%, ili od 0 do 240)
- HSL model ima kao komponentu koliko je boja **svetla** (engl. lightness)

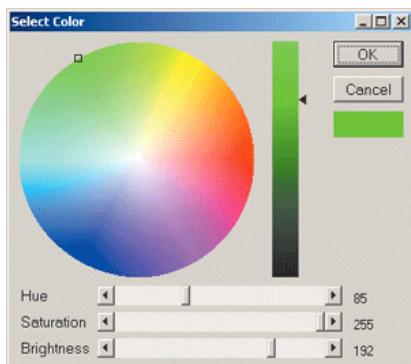


Slika 10.10: HSL kolor model: dvostruka šestostrana piramida; maksimalno zasićene boje su za  $S = 1$ ,  $L = 0.5$

- Postoje jednostavnji algoritmi (formule) za preračunavanje HSV u RGB komponente i obratno.



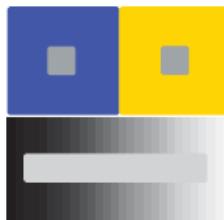
Slika 10.11: Standardni dijalog u kojem je moguće odabratи željenu boju i za nju se prikazuju koeficijenti u *RGB* i *HSV* sistemu.



Slika 10.12: HSV dijalog za odabir boje: prikaz boja za određenu vrednost sjajnosti, boja odgovara veličini ugla (crvena, zelena,...), a zasićenost poluprečniku

## **10.5 Izbor i korišćenje boja**

- Interaktivno biranje boja – ne postoji jedinstveni sistem zadavanja boja koji je najzgodniji svim korisnicima. Mnogi programi mudro omogućavaju korisniku da izabere boju korišćenjem različitih modova (direktnim zadavanjem RGB komponenti, HSV izbor korišćenjem slajdera, klikom na odabir boje iz prikaza spektra boja u vidu diska i sl)
- Treba voditi računa da kombinacija više boja bude što skladnija; mogući su različiti optički efekti



Slika 10.13: Dva siva kvadrata na vrhu deluju da su različite boje; siva traka na dnu je iste boje čitavom dužinom

- Interpoliranje – za dve bliske boje nije teško, ali za udaljene boje postoji mnogo mogućnosti i nijedna od njih nije univerzalno najbolja (voditi računa o tome da linearnom prelazu između dve tačke u jednom modelu ne odgovara nužno linearni prelazak između dve tačke u drugom modelu).
- Redukcija boja (svođenje ukupnog broja boja na unapred zadat broj (po-godno odabranih) boja).
- Reprodukovanje boja.

- Boje u računarskoj grafici i računarstvu (estetski, funkcionalni, psihološki aspekti).

### **10.6 Pitanja**

- 10.1 Koja svojstva ima monohromatska svetlost?
- 10.2 Kojom skalom je opisan intenzitet svetlosti i zašto?
- 10.3 Ako je potrebno zadati  $n$  intenziteta svetlosti počev od  $I_0$  do 1, po kojoj formuli se računa intenzitet  $I_j$  u funkciji početne vrednosti intenziteta  $I_0$ ?
- 10.4 Šta je gama korekcija?
- 10.5 Šta je polutoniranje i koji su najčešći metodi polutoniranja?
- 10.6 Uzorkovanje daje sliku istih dimenzija kao i polazna? (da) (ne)
- 10.7 Dithering daje sliku istih dimenzija kao i polazna? (da) (ne)
- 10.8 Šta je prostorni dithering?



## Glava 11

# Osvetljenje i senčenje

Svaki objekat na 3D sceni reflektuje neki deo svetlosti koji je određen karakteristikama refleksije materijala objekta. Štaviše, svaka tačka sa površine objekta prima svetlost *direktno* od izvora svetlosti (koji nisu blokirani drugim objektima) i *indirektno* od svetlosti reflektovane od drugih objekata na sceni. Složeni algoritmi (zasnovani na fizici) koji direktno modeluju rekurzivnu prirodu međuobjektne refleksije zahtevaju mnogo posla: ako je cilj prikaz u realnom vremenu, oni često zahtevaju više procesorske moći nego što današnji hardver može da pruži. Stoga se računarska grafika u realnom vremenu danas zasniva na tehnikama aproksimacije.

Mi ćemo ovde predstaviti 3D grafičku protočnu obradu fiksnih funkcija koja se sastoji od renderovanja mreža poligona, kojima se aproksimiraju i poliedri i zakrivljene površi, korišćenjem jednostavnih jednačina za modelovanje osvetljenja i senčenja. U novije vreme, sa brzim razvojem grafičkih procesorskih jedinica (GPU), aproksimacije višeg kvaliteta su postale dosežnije u realnom vremenu, kao i mogućnost da simuliramo fiziku interakcija svetlosti.

Da bi se renderovala realistična scena, postoji potreba za modeliranjem refleksija, propuštanja i prelamanja svetla, senki, određivanja boja tačaka objekata, tipova izvora svetla itd.

**Problemi osvetljenja** (eng. *illumination*) i **senčenja** (eng. *shading*) su znatno komplikovаниji problemi od problema vidljivosti. Iz razloga pojednostavljuvanja računanja razmatraćemo pre svega monohromatsku svetlost (tj. biće relevantan samo intenzitet svetla).

Postoji više različitih **modela osvetljenja** i **modela senčenja**. Problem osvetljenja određuje boju pojedinačne tačke sa date površi simuliranjem atributa svetlosti, dok problem senčenja primenjuje model osvetljenja na skup tačaka i boji kompletну površ. Dakle, model senčenja predstavlja širi okvir i on koristi model osvetljenja (neki modeli senčenja pozivaju/koriste model osvetljenja za svaki pojedinačni piksel slike, dok drugi pozivaju model osvetljenja samo za neke piksele, dok se za preostale koristi interpolacija). Neka od rešenja problema osvetljenja zasnovana su na iskustvu i eksperimentima i nisu utemeljena u fizici, ali daju dobre rezultate.

## 11.1 Modeli osvetljenja

### 11.1.1 Samoosvetljenost

Najjednostavniji model osvetljenja je sledeći: za svaki objekat, svakoj tački sa tog objekta pridružen je isti intenzitet svetlosti. U ovom nerealističnom modelu, svako telo kao da je samoosvetljeno.

**Jednačina osvetljenja** za ovaj model je

$$I = k_i$$

gde je  $I$  rezultujući intenzitet svetla, a  $k_i$  je intenzitet svetla pridružen konkretnom objektu.

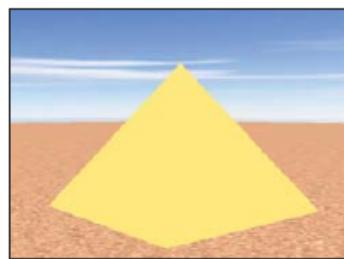
### 11.1.2 Ambijentalno svetlo

**Ambijentalno svetlo** je difuzno, bez usmerenog izvora, koje je proizvod višestrukog odbijanja svetlosti od svih površina prisutnih u okruženju.

Ako prepostavimo da se svetlost rasprostire jednakom u svim smerovima i po svim objektima (nezavisno od lokacije površi i njene orientacije), jednačina osvetljenja je za svaki objekat:

$$I = I_a k_a$$

gde je  $I_a$  (konstantni) intenzitet ambijentalnog svetla, a  $k_a$  je **koeficijent ambijentalne refleksije** objekta i on može da ima vrednosti između 0 i 1.



Slika 11.1: Prikaz piramide ukoliko na sceni postoji samo ambijentalno svetlo

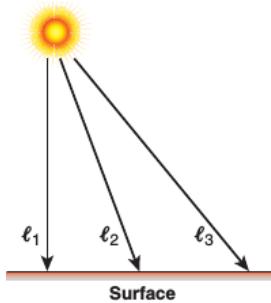
Ambijentalno svetlo obezbeđuje da je svaka površ na sceni osvetljena do nekog stepena, čime se sprečava dobijanje nerealističnih sasvim crnih regiona, odnosno površi koji su daleko od izvora svetlosti.

Količina ambijentalnog svetla se postavlja na minimum, kada je u kombinaciji sa drugim tipovima svetla.

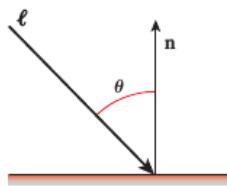
### 11.1.3 Tačkasti izvor svetla i difuzna refleksija

**Tačkasti izvor svetla** ravnomođno širi zrake u svim smerovima iz jedne tačke.

**Difuzna ili lambertovska refleksija** (odbijanje svetlosti) je refleksija od matiranih, hrapavih površina. Ovakve površine izgledaju jednakom osvetljene iz svih uglova posmatranja. Za jednu površinu osvetljenost jedino zavisi od toga koliko direktno svetlost pada na površinu: to se meri uglom  $\theta$  između pravca svetla  $\bar{L}$  i pravca normale površi  $\bar{N}$ .



Slika 11.2: Zraci koji potiču od tačkastog izvora svetla padaju na ravnu površinu pod različitim uglovima



Slika 11.3: Ugao  $\theta$  definisan kao ugao između smera dolazeće svetlosti i normalne površi

Jednačina osvetljenosti u ovom modelu je:

$$I = I_p k_p \cos \theta$$

gde je  $I_p$  jačina tačkastog izvora svetlosti,  $k_p$  koeficijent difuzne refleksije (između 0 i 1) za materijal od kojeg je načinjen objekat i  $\theta$  je ugao između pravca svetla  $\bar{L}$  i pravca normale površi  $\bar{N}$ . Ova jednačina se često naziva i Lambertovo kosinusno pravilo.

Ako su vektori  $\bar{L}$  i  $\bar{N}$  jediničnog intenziteta, onda je:

$$I = I_p k_p (\bar{L} \cdot \bar{N})$$

Ukoliko je izvor svetla dovoljno udaljen od svih objekata, možemo smatrati da je vektor  $\bar{L}$  konstantan za sve objekte i takav izvor svetla zovemo direkcioni izvor svetla.

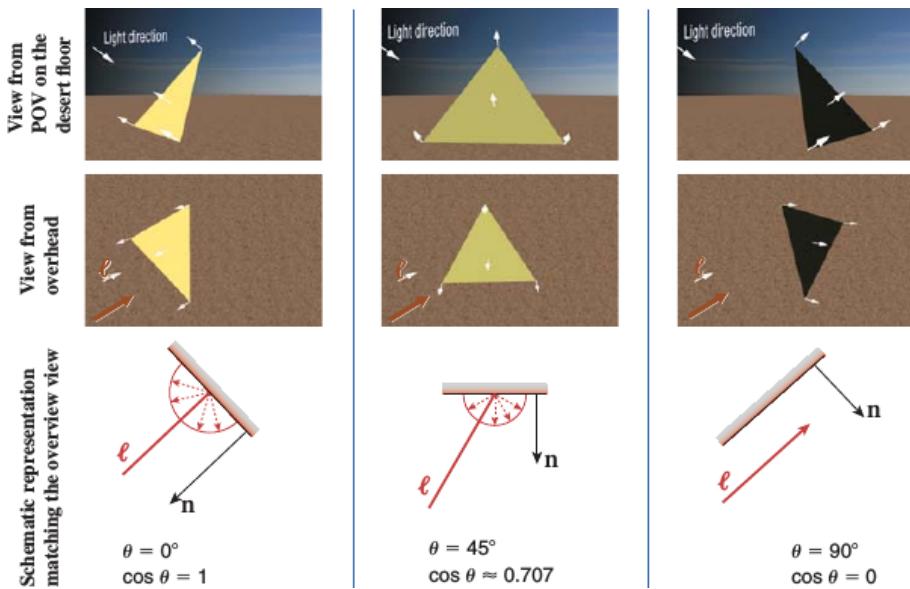
Ukoliko se osvetljenost računa po formuli

$$I = I_p k_p (\bar{L} \cdot \bar{N})$$

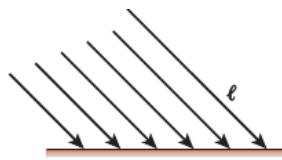
u dobijenom prikazu objekti izgledaju kao da su osvetljeni samo sa jedne strane i da se nalaze u mračnoj prostoriji (tj. zanemaruje se ambijentalna komponenta svetla). Da bi se taj efekat ublažio, često se koristi sledeća jednačina (koja uključuje i ambijentalnu osvetljenost):

$$I = I_a k_a + I_p k_p (\bar{L} \cdot \bar{N})$$

Ukoliko se projekcije dve paralelne površine od istog materijala preklapaju na slici, neće moći da se odredi gde prestaje jedna, a počinje druga (jer su



Slika 11.4: Osjetljivost na svjetlosti na osnovu Lambertovog kosinusnog pravila za tri različite vrednosti ugla  $\theta$



Slika 11.5: Zraci koji potiču od direkcionog izvora svetla, beskonačno udaljenog od ravne površine, padaju na površinu pod istim uglom

projekcije iste boje). Da bi se razrešio ovaj problem uzima se u razmatranje i tzv. **faktor slabljenja izvora svetla** (eng. light-source attenuation factor)  $f_{att}$  uz koji jednačina osjetljivosti postaje:

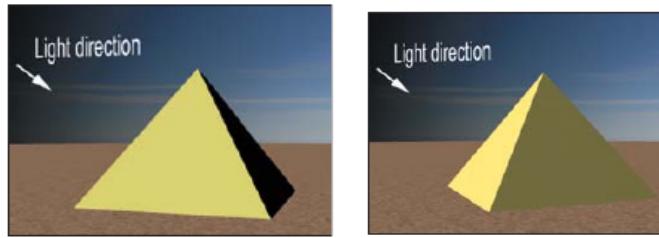
$$I = I_a k_a + f_{att} I_p k_p (\bar{L} \cdot \bar{N})$$

Na ovaj način objekti koji su udaljeniji od izvora svetla prikazuju se kao tamniji, odnosno površi koje se preklapaju, a na različitim rastojanjima su od svetlosti biće međusobno razlučive na slici.

Najjednostavnija forma za  $f_{att}$  je

$$f_{att} = \frac{1}{d_L^2}$$

gde je  $d_L$  rastojanje od objekta do izvora svetla. Međutim, kada je izvor svetla daleko, vrednosti  $\frac{1}{d_L^2}$  se puno ne razlikuju; ako je blizu, onda se za površi sa istim uglom između  $\bar{N}$  i  $\bar{L}$  vrednosti puno razlikuju, te ovaj model ne daje željeni efekat.



Slika 11.6: Renderovanje piramide sa direkcionim izvorom svetla, pri čemu je ugao blizu a)  $90^\circ$  b)  $70^\circ$  za najdesniju stranu koja je vidljivija

Finiji model je opisan jednačinom:

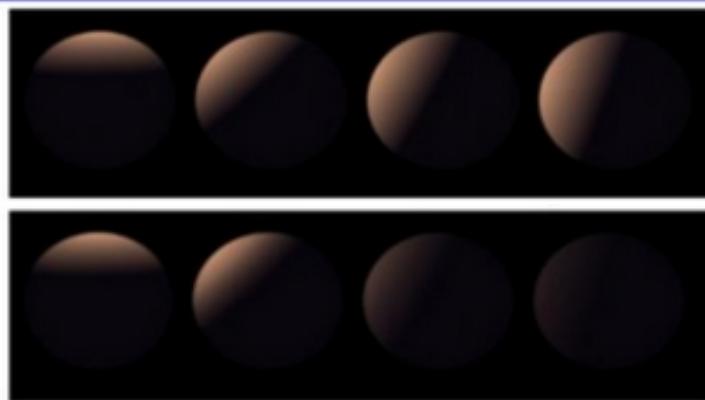
$$f_{att} = \frac{1}{c_1 + c_2 d_L + c_3 d_L^2}$$

gde su  $c_1$ ,  $c_2$  i  $c_3$  korisnički definisane konstante pridružene izvoru svetla.  
Često se ovaj model koristi kada je izvor svetla u tački pogleda.

U razmatranje treba uvesti i rastojanje posmatrača do objekta.

#### 11.1.4 Atmosfersko slabljenje

Kao što osvetljenost treba da slabi sa udaljenošću objekta od izvora svetla, tako treba da slabi i sa udaljenošću objekta od posmatrača.



Slika 11.7: Primer kada se ne uzima u obzir atmosfersko slabljenje svetlosti i primer kada se ono računa

Prednjoj i zadnjoj ravni projektovanja pridružuju se faktori skaliranja  $s_1$  i  $s_2$ , a faktor  $s_o$  se određuje u zavisnosti od rastojanja između objekta i posmatrača (tj. u zavisnosti od  $z$  koordinate). Ukoliko je  $z_o$  (vrednost  $z$  koordinate objekta) jednako  $z_1$ , onda je  $s_o = s_1$  i, slično, ukoliko je  $z_o = z_2$ , onda je  $s_o = s_2$ . U opštem slučaju je:

$$s_o = s_2 + \frac{(z_o - z_2)(s_1 - s_2)}{z_1 - z_2}$$

Cilj je za svako  $s_o$  odrediti prirodno  $I'$  između (izračunate) osvetljenosti  $I$  i osvetljenosti za udaljene objekte  $I_{dc}$  (eng. depth-cue value):

$$I' = s_o I + (1 - s_o) I_{dc}$$

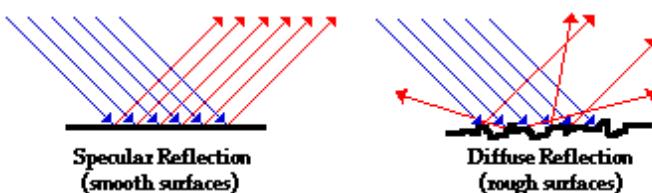
### 11.1.5 Spekularna refleksija

**Spekularna refleksija** se može videti na svakom glatkom, sjajnom objektu (npr. deo jabuke je najsvetlij, dok je ostatak obasjan difuznim svetлом; u tom delu jabuka ima belu, a ne crvenu boju).



Slika 11.8: Prikaz spekularne refleksije – direktne refleksije izvora svetla na sjajnoj površini.

Efekat spekularne refleksije je najjači u pravcu  $\bar{R}$  koji je simetričan pravcu svetlosti  $\bar{L}$  u odnosu na normalu površi  $\bar{N}$ .



Slika 11.9: Razlika između difuzne i spekularne refleksije.

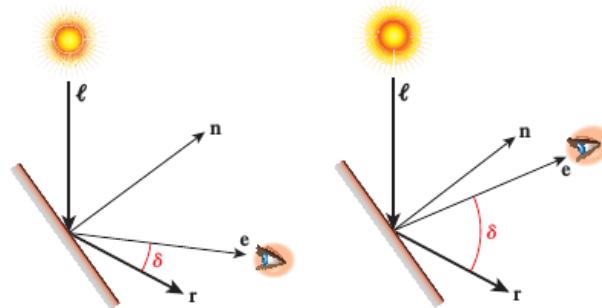
Prema **Phong-ovom modelu**, jednačina ukupne osvetljenosti se zadaje kao kombinacija ambijentalne, lambertovske i spekularne komponente (sumiranje).

jem za sve izvore svetlosti):

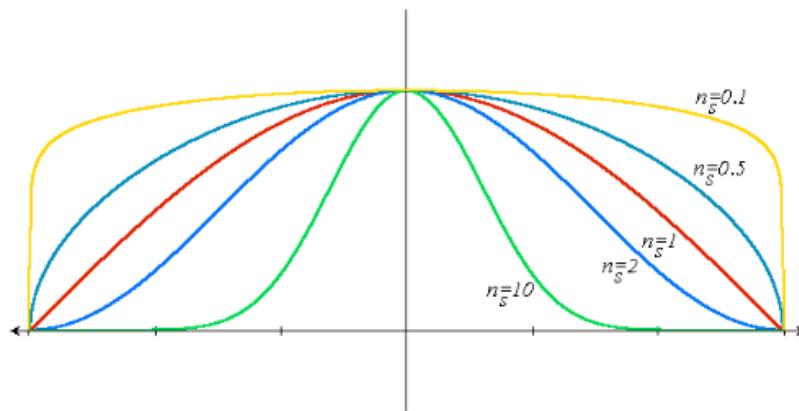
$$I = I_{amb} + I_{diff} + I_{spec}$$

$$I = I_a k_a + f_{att} I_p (k_p \cos \theta + W(\theta) \cos^s \delta)$$

gde je  $\theta$  ugao između  $\bar{L}$  i  $\bar{N}$ ,  $\delta$  ugao između  $\bar{R}$  i  $\bar{V}$  (pravac gledanja),  $s$  je **ekspONENT spekularne refleksije** za materijal (uzima vrednost od 1 do nekoliko stotina za veoma sjajne površi; kod sjajnih površina je veoma oštro smanjenje spekularnog efekta, dok kod površi koje nisu tako sjajne je ovaj efekat ne tako sjajan ali je vidljiv na široj površini), a za  $W(\theta)$  se obično uzima konstantna vrednost  $k_s$  — **koefficijent spekularne refleksije** (između 0 i 1).



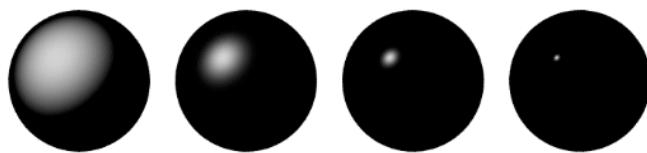
Slika 11.10: Računanje spekularne refleksije u situaciji kada je pravac gledanja a) blizu zraka refleksije b) daleko od zraka refleksije; značajna razlika u vrednosti izraza  $\cos \delta$  postaje još veća kada se digne na veliki stepen te je stoga u drugom slučaju spekularni doprinos blizak nuli.



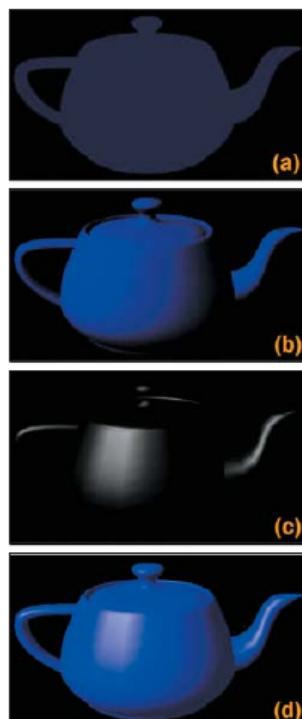
Slika 11.11: Phongova kriva osvetljenosti za različite vrednosti eksponenta spekularne refleksije  $s$ .

Ako su vektori  $\bar{L}$ ,  $\bar{N}$ ,  $\bar{R}$  i  $\bar{V}$  normalizovani, onda važi:

$$I = I_a k_a + f_{att} I_p (k_p (\bar{L} \cdot \bar{N}) + k_s (\bar{R} \cdot \bar{V})^s)$$



Slika 11.12: Prikaz spekularne refleksije za različite vrednosti eksponenta spekularne refleksije  $s$ .



Slika 11.13: Renderovanje čajnika uz prikaz doprinosu svake od tri komponenti koje učestvuju u Phong-ovoj jednačini: a) ambijentalna b) difuzna c) spekularna d) rezultat dobijen sumiranjem svih doprinosova

### 11.1.6 Prošireni izvori svetla

Za razliku od tačkastih izvora svetla, **prošireni** ili **distribuirani** izvori svetla (engl. extended light sources) imaju površinu i, kao posledicu, daju mekše senke. Mi ih nećemo detaljno izučavati.

## 11.2 Hromatska svetlost

Prethodni modeli i jednačine razmatrali su samo monohromatsku svetlost. Hromatska svetlost se obično obrađuje tako što se posebno razmatraju vrednosti njenih triju komponenti.

Neka je sa  $(O_{dR}, O_{dG}, O_{dB})$  zadata difuzna crvena, zelena i plava komponenta nekog objekta u RGB kolor sistemu. U ovom slučaju tri primarne komponente osvetljenja  $I_R, I_G, I_B$  se reflektuju proporcionalno sa  $k_d O_{dR}$ ,  $k_d O_{dG}$  i  $k_d O_{dB}$ . Stoga, u kolor modelu jednačina

$$I = I_a k_a + f_{att} I_p k_d (\bar{L} \cdot \bar{N})$$

daje za crvenu komponentu:

$$I_R = I_a R k_a O_{dR} + f_{att} I_p R k_d O_{dR} (\bar{L} \cdot \bar{N})$$

U idealnom slučaju, hromatsko osvetljenje bi trebalo računati kombinovanjem (neprekidnih) rezultata za sve boje iz spektra. No, i opisani pojednostavljeni model obično daje prihvatljive rezultate.

Na primer, u zavisnosti od talasne dužine Phongova jednačina

$$I = I_a k_a + f_{att} I_p (k_p (\bar{L} \cdot \bar{N}) + k_s (\bar{R} \cdot \bar{V})^n)$$

dobija finiji oblik:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} (k_d O_{d\lambda} (\bar{L} \cdot \bar{N}) + k_s O_{s\lambda} (\bar{R} \cdot \bar{V})^n)$$

gde je  $O_{s\lambda}$  spekularna boja objekta.

## 11.3 Senčenje poligona

Svaka površ se može senčiti izračunavanjem normale na površ u svakoj tački koja je vidljiva i primenom odgovarajućeg modela osvetljenja u toj tački. Međutim, ovaj algoritam grube sile je skup, te razmatramo alternativne algoritme senčenja.

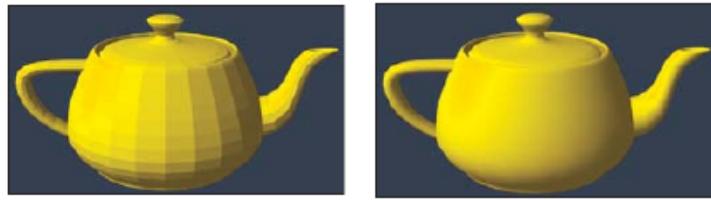
Najjednostavniji model senčenja jeste **ravansko (konstantno) senčenje**, koje koristi model osvetljenja jednom za izračunavanje vrednosti intenziteta koja se koristi za senčenje kompletног poligona. Ovaj model je odgovarajući ukoliko su zadovoljene sledeće pretpostavke:

- izvor svetla je beskonačno daleka tačka, pa je vrednost  $\bar{L} \cdot \bar{N}$  konstantna za sve tačke jednog poligona
- tačka posmatranja je beskonačno daleka tačka, pa je vrednost  $\bar{V} \cdot \bar{N}$  konstantna za sve tačke jednog poligona
- poligoni odgovaraju stvarnom objektu i nisu njegova aproksimacija

Ako su vrednosti  $\bar{L}$  i  $\bar{N}$  konstantne, konstantna je i vrednost  $\bar{R}$  (jer je vektor  $\bar{R}$  simetričan vektoru  $\bar{L}$  u odnosu na  $\bar{N}$ ), pa je konstantna i vrednost  $\bar{R} \cdot \bar{V}$ .

Poslednja pretpostavka, da mreža poligona odgovara stvarnom objektu je najčešće netačna, što negativno utiče na rezultujuću sliku.

Alternativno, može se koristiti **interpolirano (Gouraud) senčenje** u kome se informacije o senkama u svakoj tački poligona izračunavaju linearном interpolacijom vrednosti izračunatim u njegovim temenima.



Slika 11.14: Ravansko senčenje i interpolirano senčenje prikazano na modelu čajnika

## 11.4 Senke

Algoritmi za vidljivost određuju koji se delovi površi mogu videti iz tačke posmatranja, dok algoritmi za senke određuju koji se delovi površi mogu videti iz izvora svetla. Oni delovi površi koji se ne vide iz izvora svetla su u senci.

Vidljivost iz tačkastog izvora svetla je, kao i vidljivost iz tačke posmatranja: sve ili ništa.

Ako je dato više izvora svetla ( $m$ ), Phongova jednačina osvetljenosti postaje:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + \sum_{1 \leq i \leq m} S_i f_{att} I_{p\lambda} (k_d O_{d\lambda} (\bar{L} \cdot \bar{N}) + k_s O_{s\lambda} (\bar{R} \cdot \bar{V})^n)$$

gde  $S_i$  ima vrednost 0 ako je svetlost  $i$  blokirana u ovoj tački, a inače vrednost 1.

## 11.5 Transparentnost

Neki objekti/materijali propuštaju deo svetlosti (npr. staklo); obično se ta svetlost **prelama** (eng. refract), ali se taj efekat često zanemaruje. Stoga, po-drazumevamo da šta god da je vidljivo duž linije pogleda kroz transparentnu površ se geometrijski i nalazi na toj liniji pogleda.

Razmotrićemo dva modela u kojima se transparentnost obrađuje ne vodeći računa o prelamanju (boje dva objekta se kombinuju kada je jedan objekat vidljiv kroz drugi). To su:

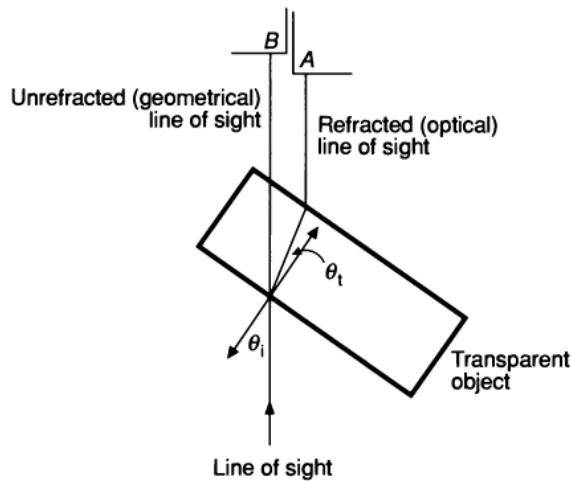
**interpolirana transparentnost:** Ukoliko je poligon  $P_1$  transparentan i nalazi se ispred neprozirnog poligona  $P_2$ , intenzitet svetlosti je jednak:

$$I_\lambda = (1 - k_{t_1}) I'_\lambda + k_{t_1} I''_\lambda$$

gde je  $I'_\lambda$  svetlost za poligon  $P_1$ ,  $I''_\lambda$  za poligon  $P_2$ , a  $k_{t_1}$  je **koeficijent transmisije** za poligon  $P_1$  i on predstavlja meru transparentnosti. Vrednost  $k_{t_1}$  je između 0 i 1; kada je jednaka 0, poligon je neproziran, a kada je jednaka 1 poligon je potpuno transparentan.

**filtrirana transparentnost:** poligon se tretira kao filter koji selektivno propušta različite talasne dužine:

$$I_\lambda = I'_\lambda + k_{t_1} O_{t\lambda} I''_\lambda$$



Slika 11.15: Ako se razmatra i prelamanje objekat A je vidljiv kroz transparentni objekat duž linije pogleda; ako se refrakcija ignoriše, vidljiv je objekat B

gde je  $O_{t\lambda}$  **boja transparencije** poligona  $P_1$  (i može biti posebno zadavana za svaku vrednost  $\lambda$ )

U oba pomenuta modela transparentnosti, ako se još neki transparentni poligoni nalaze ispred ovih poligona, onda se izračunavanje rekurzivno poziva za poligone u redosledu od najudaljenijeg do najbližeg, pri čemu se svaki put koristi prethodno izračunato  $I_\lambda$  kao  $I''_\lambda$ .

Dosta teže je modelovati transparentnost koja uključuje i prelamanje svetlosti. Prelamanje svetlosti opisuje jednakost:

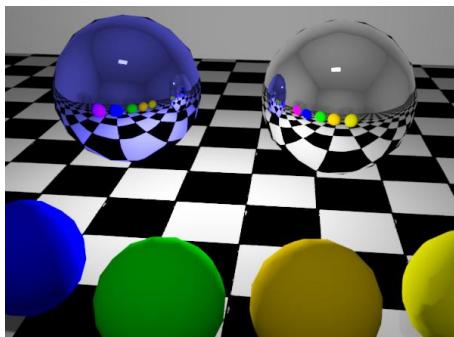
$$\frac{\sin \theta_1}{\sin \theta_2} = \frac{\nu_1}{\nu_2}$$

gde su  $\theta_1$  i  $\theta_2$  upadni ugao i ugao prelamanja, a  $\nu_1$  i  $\nu_2$  indeksi prelamanja materijala kroz koje svetlost prolazi.

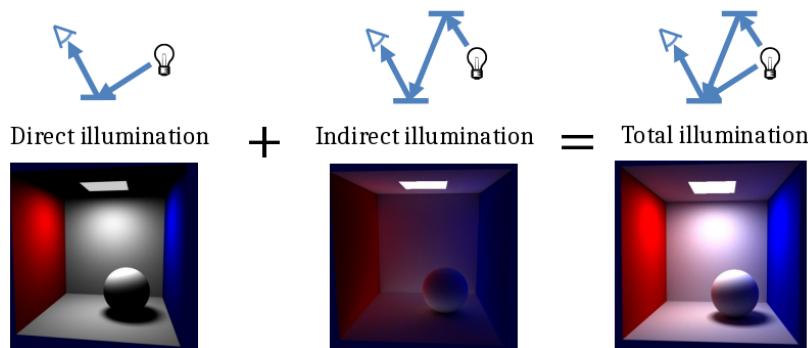
## 11.6 Međuobjektne refleksije i globalno osvetljenje

Međuobjektne refleksije se javljaju kada se na površini objekta reflektuju druge površine u okruženju. Ovaj efekat može da se kreće od spekularne refleksije (koja se menja sa položajem posmatrača), pa do difuzne refleksije (na koju ne utiče pozicija posmatrača).

Modeli osvetljenja računaju boju u tački u terminima svetlosti direktno emitovane od strane izvora svetla i terminima svetlosti koja dolazi do tačke nakon reflektovanja i transmisije kroz razne površi. Ova indirektno reflektovana i transmitovana svetlost se obično zove **indirektno osvetljenje**, dok se pod **direktnim osvetljenjem** podrazumeva svetlost koja dolazi direktno iz izvora svetla do tačke koja se senči. Kombinovanjem ova dva osvetljenja, dobija se **globalno osvetljenje**, koje uzima u obzir interakciju svetlosti sa svim površi na sceni.



Slika 11.16: Međuobjektne refleksije



Slika 11.17: Globalno osvetljenje

### 11.6.1 Rekurzivni ray-tracing algoritam

Osnovni ray-tracing algoritam za vidljivost određuje tačku preseka koja je najbliža posmatraču (i njenu boju). Da bismo odredili senke, upućujemo dodatni zrak svetlosti iz te tačke preseka do svakog izvora svetla. Ako neki od ovih zraka senke preseca neki objekat duž svog puta, onda je polazna tačka u senci i taj izvor svetla ne doprinosi njenoj boji.

```

select center of projection and window on view plane;
for each scan line in image do
    for each pixel in scan line do
        begin
            determine ray from center of projection through pixel;
            pixel := RT_trace(ray, 1);
        end;

{Intersect ray with objects and compute shade at closest intersection.}
{Depth is current depth in ray tree.}
procedure RT_trace(ray: RT_ray; depth: integer) : RT_color;
begin
    determine closest intersection of ray with an object;

```

```

if object hit then
begin
    compute normal at intersection;
    RT_trace := RT_shade (closest object hit, ray,
                           intersection, normal, depth);
end
else
    RT_trace := BACKGROUND_VALUE;
end;

{Compute shade at point on object, tracing rays for shadows,
reflection, refraction}
procedure RT_shade (
    object : RT_object; {Object intersected}
    ray : RT_ray;       {Incident ray}
    point : RT_point;   {Point of intersection to shade}
    normal : RT_normal; {Normal at point}
    depth : integer      {Depth in ray tree}
) : RT_color;
var
    color : RT_color     {Color of ray}
    rRay, tRay, sRay : RT_ray; {Reflected, refracted, and shadow rays}
    rColor, tColor : RT_Color; {Reflected and refracted ray colors}

begin
    color := ambient term;
    for each light do
        begin
            sRay := ray to light from point;
            if dot product of normal and direction to light is positive then
                compute how much light is blocked by opaque and transparent
                surfaces, and use to scale diffuse and specular terms before
                adding them to color;
        end

    if (depth<maxDepth) then {Return if depth too deep.}
    begin
        if object is reflective then
            begin
                rRay := ray in reflection direction from point;
                rColor := RT_trace(rRay, depth+1);
                scale rColor by specular coefficient and add to color;
            end;

        if object is transparent then
            begin
                tRay := ray in refraction direction from point;
                if total internal reflection does not occur then
                    begin

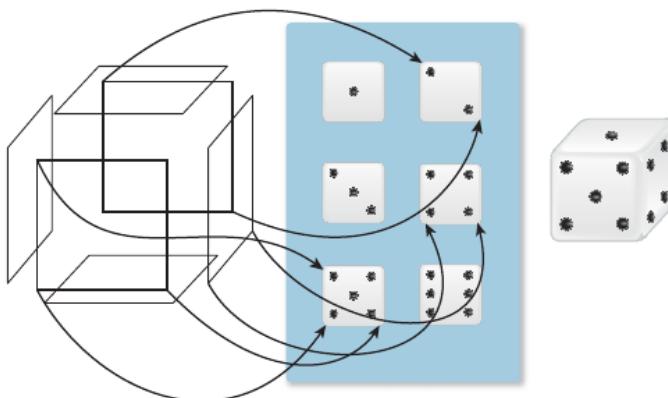
```

```
    tColor := RT_trace(tRay, depth+1);
    scale tColor by transmission coefficient and add to color;
end
end
end
RT_shade := color; {Return color of ray}
end.
```

## Glava 12

# Teksture

Jedna od standardnih komponenti grafičke protočne obrade (engl. pipeline) jeste i **preslikavanje (mapiranje) tekstura** (engl. texture mapping). Prilikom preslikavanja tekstura posmatramo poligon (ili skup poligona) i svakoj tački poligona dodeljujemo boju pretragom kroz sliku tekture; ova tehnika nalikuje lepljenju nalepnica na objekat.

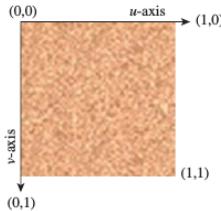


Slika 12.1: Temenima svake od šest strana kocke dodeljene su teksturne koordinate (neke od njih prikazane su strelicama); slika tekture se koristi za određivanje izgleda svake strane kocke. Jedna ista 3D lokacija može imati više pridruženih teksturnih koordinata, jer je deo većeg broja strana.

Teksturne koordinate opisuju kako se tekstura rasteže i deformiše kako bi pokrila neki deo objekta.

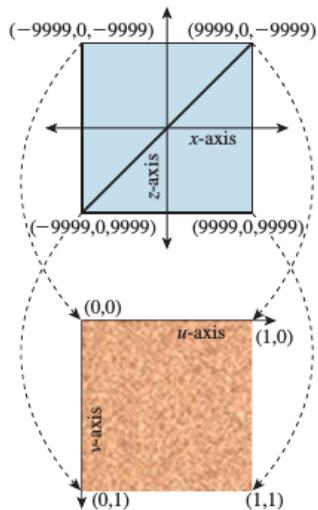
Zašto su nam potrebne teksture? Ako na primer želimo da prikažemo grube materijale ili materijale čija boja varira kao što su cigla, šljunak, mermur ili drvo, ili da napravimo pozadinu poput travnate ravnice ili guste šume, ne preporučuje se pravljenje mreže poligona koja bi sadržala sve detalje fino graduisane strukture materijala. Na primer, razmotrimo složenost mreže kojom bi se modelovale pukotine i rupice grubo klesanog kamena antičkih piramida – od jednostavne mreže piramide koja se sastoji od 4 trougla dobili bismo milione trouglova čime bi eksplodirali i memorijski i vremenski za-

htevi obrade. Ako bismo u ovoj situaciji kao trik koristili preslikavanje tekstura, složene scene (kao što su recimo njive posmatrane iz aviona) ili složeni materijali (npr. asfalt) mogli bi se grubo simulirati bez povećanja složenosti mreže.



Slika 12.2: Koordinatni sistem texture u pokretnom zarezu primjenjen na sliku uzorka peska. Koordinatni početak nalazi se u gornjem levom uglu.

Nanošenje tekstura na 3D površi odgovara činu prekrivanja objekta ras-teglijivim listom ukrasnog papira. Teorijski, za svaku tačku  $P$  sa date površi moramo precizno zadati koju tačku sa papira treba da dodirne. U praksi, ovo preslikavanje zadajemo za svako *teme* površi, dok se za unutrašnje tačke koristi interpolacija. Ovakav mehanizam nanošenja tekstura podrazumeva postojanje koordinatnog sistema u kom možemo da referišemo na pozicije sa slike texture. Prema dogovoru, umesto da koristimo tačne celobrojne koordinate piksela, referišemo na tačke sa slike texture korišćenjem koordinatnog sistema texture koji je u pokretnom zarezu (slika 12.2), čije  $u$  i  $v$  koordinate uzimaju vrednosti na intervalu  $[0,1]$ .



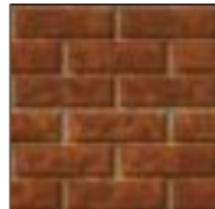
Slika 12.3: Preslikavanje temena svetskih koordinata u model podloge u pustinji sačinjen od dva trougla tako da odgovara koordinatama tekstura.

Ako želimo da preslikamo kvadrat koji čini podlogu slike i koji je predstavljen sa dva komplanarna trougla u kvadrat kojim je predstavljena slika

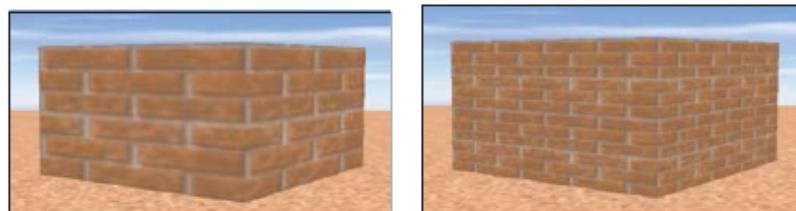
teksture, dovoljno je uglovima teksturne slike dodeliti teksturne koordinate (slika 12.3).

## 12.1 Popločavanje i istezanje

Ako se tekstura koristi da bi simulirala materijal koji je konzistentnog izgleda i bez očiglednih tačaka prekida (npr. pesak, asfalt, cigla), slika tekture se ponavlja da bi se pokrila ciljna površina. U ovom slučaju tekstura je obično mala slika materijala (dobjena ili sintetički ili fotoaparatom), dizajnirana tako da se susedne "pločice" neprimetno uklapaju. Na primer, razmotrimo sliku tekture sa slike 12.4 koja prikazuje šest redova crvenih cigala. Ako je primenimo na svaku stranu pravougaone prizme bez ponavljanja dobićemo realističnu sliku (slika 12.5 a)), ali na ovaj način ne bi mogla da se predstavi visoka zgrada koja se sastoji od velikog broja redova cigala. **Popločavanje** (engl. tiling) omogućava da se broj redova cigala umnoži, čime se dobija slika 12.5 b) koja je više nalik visokoj tvrđavi.



Slika 12.4: Kvadratna slika uzorka cigala.



Slika 12.5: a) Rezultat istezanja jedne kopije tekture cigala na svaki od zidova. b) Rezultat popločavanja višestrukim kopijama tekture cigle na svaki od zidova.

Kada se tekstura koristi kao zamena za jako složeni model (npr. grad koji se gleda sa visine ili oblačno nebo), slika tekture je obično jako velika (da bi se obezbedila visoka rezolucija) i može nastati fotoaparatom ili sintetički (ako npr. predstavlja reljef u nekom svetu fantazije). Ako bismo izvršili popločavanje ovom vrstom tekture, dobio bi se neprirodni izgled scene. Stoga se kod ovakve vrste tekture koristi **istezanje** (engl. stretching), tj. teksturne koordinate se postavljaju tako da slika tekture pokrije celu mrežu. Na primer, za prikaz neba mogla bi se iskoristiti slika tekture data na slici 12.6 koja bi se istezala na unutrašnjoj strani valjka.



Slika 12.6: Slika neba koja se može iskorisiti kao slika tekture.



Slika 12.7: Slika dobijena istezanjem slike neba kao tekture.

## 12.2 Pridruživanje teksturnih koordinata

Pomenuli smo da se modeli često ne opisuju samo svojom geometrijom već i teksturama. Svakoj tački objekta možemo dodeliti neko svojstvo (npr. boju površi) i ovo svojstvo se koristi dalje u renderovanju objekta. Kada određujemo boju piksela u procesu renderovanja, obično to izračunavanje zasnivamo na informaciji o objektu koji se javlja na poziciji tog piksela: npr. u slučaju trougla koristimo njegovu orientaciju da bismo odredili koliko je osvetljena odgovarajuća tačka na sceni. U nekim slučajevima trouglu se ne dodeljuje jedinstvena boja, već se svakom temenu zasebno dodeljuje boja, pa boju tačke od interesa dobijamo interpolacijom. Često su temenima trougla pridružene lokacije mape tekture koja je obično slika pravougonog oblika dimenzije  $n \times k$ . Boja tačke od interesa se određuje pregledanjem lokacija na mapi tekture, kao na slici 12.8.

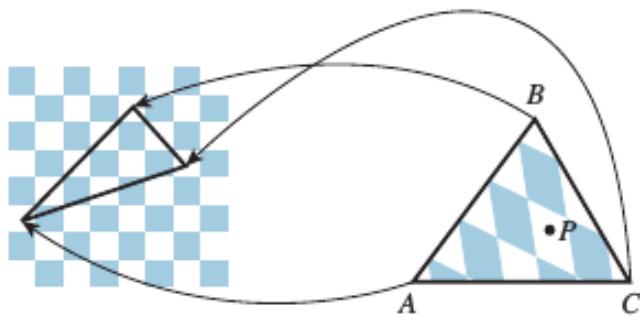
Prirodno je zapitati se kako se temenima trouglova pridružuju lokacije mape tekture. Ovo je zadatak osobe koja pravi model. Postoje neki jednostavnvi modeli kod kojih je ovo pridruživanje jednostavno.

Sfera se često parametrizuje na sledeći način:

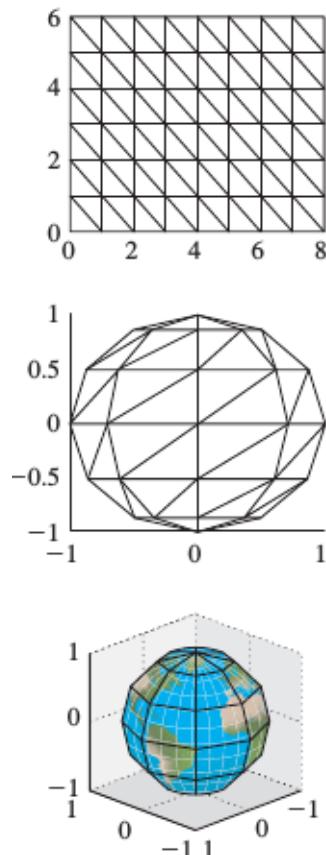
$$S(\theta, \phi) = (\cos \theta \cos \phi, \sin \phi, \sin \theta \cos \phi)$$

gde je  $\theta$  iz intervala  $[0, 2\pi]$ , a  $\phi$  iz intervala  $[-\pi/2, \pi/2]$ . Ako je  $\theta$  konstantno, a menjamo  $\phi$  dobijamo meridijane, a ako  $\phi$  održavamo konstantnim, a menjamo vrednost za  $\theta$  dobijamo paralele.

Ako krenemo sa mrežom trouglova dimenzije  $n \times k$  (kao na slici 12.9) gde je  $n = 8$  i  $k = 6$ , možemo svakom temenu  $(i, j)$ ,  $i \in \{0, 1, \dots, 7\}$ ,  $j \in \{0, 1, \dots, 5\}$  ove mreže pridružiti tačku u 3D prostoru na sledeći način:



Slika 12.8: Boja tačke  $P$  trougla  $ABC$  se određuje na osnovu mape teksture. Tačkama  $A$ ,  $B$  i  $C$  pridružene su tačke na slici sa šahovskom tablom, tačka  $P$  odgovara tački sa belog polja, pa je njena boja teksture bela.



Slika 12.9: Preslikavanje tekstura globusa.

$$\theta = 2\pi j / (n - 1)$$

$$\phi = -\pi/2 + \pi i / (k - 1)$$

$$X = \cos \theta \cos \phi$$

$$Y = \sin \phi$$

$$Z = \sin \theta \cos \phi$$

tj. postavljajući  $\theta$  i  $\phi$  da označavaju geografsku dužinu i geografsku širinu, redom. Oblik nalik sferi koji se dobija prikazan je na srednjoj slici. Takođe, pretpostavimo da nam je na raspolaganju  $100 \times 200$  slika teksture "neprojektovane" mape Zemlje (vertikalna koordinata je proporcionalna geografskoj širini, a horizontalna geografskoj dužini). Temenu na poziciji  $(i, j)$  dodeljujemo teksturne koordinate  $100i/(n - 1), 200j/(k - 1)$  i rezultujući globus renderovan mapom teksture prikazan je na slici u dnu.

U ovom slučaju, način na koji smo kreirali mrežu napravio je ujedno i izbor teksturnih koordinata prirodnim. Ipak postoji jedan problematičan aspekt ovog pristupa a to je što teksturne koordinate zavise od broja piksela na slici koju koristimo kao mapu sveta. Da smo kreirali ovaj oblik i došli do zaključka da rezultat ne izgleda dobro, verovatno bismo želeli da probamo da koristimo sliku veće rezolucije, ali to bi takođe nametnulo menjanje teksturnih koordinata. Zbog ovoga se teksturne koordinate obično zadaju kao brojevi između 0 i 1, predstavljajući na taj način procenat puta pređenog uzduž ili preko slike; stoga teksturne koordinate  $(0.75, 0.5)$  odgovaraju tački koja se nalazi na  $3/4$  puta uzduž (nezavisno od veličine slike) i na pola puta popreko. Uobičajeno je teksturne koordinate označavati sa  $u$  i  $v$ , pa stoga teme slike sada ima 5 atributa:  $x, y, z, u$  i  $v$ .

Ako imamo mrežu u kojoj je svakom temenu dodeljena teksturna koordinata, kako odrediti teksturne koordinate na nekoj lokaciji unutar trougla? Npr. pretpostavimo da imamo  $u$  koordinatu svakog temena mreže; ovakva dodata realnih vrednosti svakom temenu na jedinstven način određuje deo po deo linearne funkcije u svakoj tački mreže: ako je  $P$  tačka trougla  $ABC$ , i  $u$  koordinate tačaka  $A, B$  i  $C$  su redom  $u_A, u_B$  i  $u_C$ , možemo odrediti  $u$  koordinatu tačke  $P$  korišćenjem **baricentričnih koordinata**.

Trouglovi predstavljaju gradivnu jedinicu računarske grafike. Ako trougao ima temena  $A, B$  i  $C$ , tada se tačka  $Q = (1 - t)A + tB$ , gde je  $0 \leq t \leq 1$  nalazi na stranici  $AB$  trougla  $ABC$ . Slično, tačka oblika  $R = (1 - s)Q + sC$  se nalazi na duži  $QC$  ako je  $0 \leq s \leq 1$ . Ako ovo raspišemo dobijamo:

$$R = (1 - s)(1 - t)A + (1 - s)tB + sC$$

Ovim se definiše funkcija

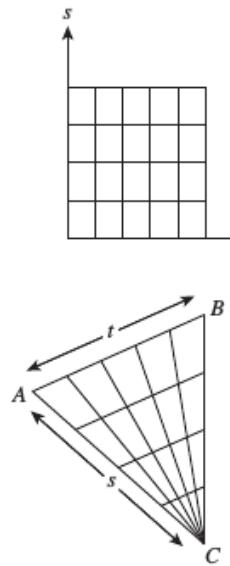
$$F : [0, 1] \times [0, 1] \rightarrow R^2 : (s, t) \rightarrow (1 - s)(1 - t)A + (1 - s)tB + sC$$

čija je slika tačno trougao  $ABC$  (slika 12.10).

Lako se pokazuje da su za  $0 \leq s, t \leq 1$  sva tri koeficijenta  $(1 - s)(1 - t)$ ,  $(1 - s)t$  i  $s$  pozitivna. Njihovim sumiranjem dobijamo:

$$(1 - s)(1 - t) + (1 - s)t + s = (1 - s) + s = 1$$

Dakle sve tačke trougla  $ABC$  su oblika  $\alpha A + \beta B + \gamma C$ , gde je  $\alpha + \beta + \gamma = 1$  i  $\alpha, \beta, \gamma \geq 0$ . Tačke kod kojih je  $\alpha = 0$  leže na stranici  $BC$ , tačke kod kojih



Slika 12.10: Funkcija  $F$  slika jedinični kvadrat u trougao  $ABC$ ; time se kompletan ivica  $s = 1$  slika u tačku  $C$ . Sve ostale duži u kvadratu se slikaju u prikazane linije trougla.

je  $\beta = 0$  leže na stranici  $AC$ , tačke kod kojih je  $\gamma = 0$  leže na stranici  $AB$ . Ako je  $P = \alpha A + \beta B + \gamma C$ , onda brojeve  $\alpha, \beta, \gamma$  nazivamo **baricentričnim koordinatama tačke  $P$**  u odnosu na trougao  $ABC$ .

Pišemo  $P$  u obliku  $P = \alpha A + \beta B + \gamma C$  i definišemo da je  $u_P = \alpha u_A + \beta u_B + \gamma u_C$ . Na isti način možemo izraziti  $v$  koordinatu i na ovaj način se na jedinstven način određuju  $uv$  koordinate tačke  $P$ .



## *Glava 13*

---

# **Aliasing**

---

- **Aliasing efekat** nastaje zbog toga što su objekti kao duži, poligoni, kružnici itd. neprekidni, dok je rasterski uređaj diskretan. Figura dobijena od originalne figure diskretizacijom na rasterskom uređaju zove se **alias**.
- Aliasing se manifestuje na sledeće načine:
  - reckaste/stepenaste ivice
  - pogrešno prikazani fini detalji ili tekstura
  - mali objekti (npr. manji od jednog piksela) mogu da budu ignorisani ili da njihov uticaj bude preveliki

### **13.1 Antialiasing**

- **Antialiasing** obuhvata tehnike koje koriste različite intenzitete osvetljenosti za postizanje veće vizuelne rezolucije



Slika 13.1: Primer duži bez korišćenja antialiasing tehnika (gore) i sa korišćenjem antialiasing tehnika (dole)

- Postoje dva osnovna antialiasing metoda:
  - tretirati piksel ne kao tačku nego kao oblast,
  - povećati rezoluciju za koju se izvršava izračunavanje, a rezultat prikazati na postojećoj rezoluciji (nadsemplovanje).

### 13.1.1 Antialiasing — tretiranje piksela kao oblasti

- Tretirati piksel ne kao tačku (kao u osnovnim verzijama algoritama) nego kao oblast.
- Na primer, u algoritmu za popunjavanje poligona, izračunava se da li piksel (kao matematička tačka) pripada ili ne pripada unutrašnjosti poligona; ako pripada unutrašnjosti, cela oblast piksela se boji istom bojom iako jedan deo te oblasti može da bude u spoljašnjosti idealnog poligona. Rezultat je karakteristična reckasta ivica poligona.
- Ako su raspoložive nijanse (nijanse sive ili neke druge odgovarajuće boje) ovaj efekat je moguće ublažiti: umesto da se svaki piksel oboji datom bojom ili ne, treba izabrati pogodnu nijansu koja odgovara fragmentu oblasti koja se nalazi unutar poligona.
- Na primer, ako je spoljašnjost poligona bela, a unutrašnjost i ivice treba obojiti crnom bojom, onda granične piksele treba obojiti bojom intenziteta proporcionalnog delu piksela (razmatranog kao kvadrat) koji pripada unutrašnjosti poligona.
- Bresenhamov algoritam je moguće jednostavno modifikovati tako da daje aproksimaciju dela (fragmenta) piksela koji se nalazi sa jedne strane prave.

### 13.1.2 Antialiasing — nadsemplovanje

- **Nadsemplovanje** (engl. supersampling) predstavlja antialiasing tehniku koja se zasniva na tome da se poveća rezolucija za koju se izvršava izračunavanje, a da se zatim rezultat prikaže na postojećoj rezoluciji.
- Uređaji za prikaz imaju svoja praktična ograničenja, pa rezultat mora da se prilagodi takvom izlazu.
- Prikazivanje na izlazu sa manjom rezolucijom se realizuje tehnikom **uprosečavanja** (eng. averaging).
- Postoji više vrsta uprosečavanja:
  - *ravnomerne/uniformne* (računa se prosek vrednosti elemenata matrice koja okružuje piksel). Naredne matrice se koriste za smanjivanje rezolucije 2 puta i 4 puta:

1	1
1	1

1	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

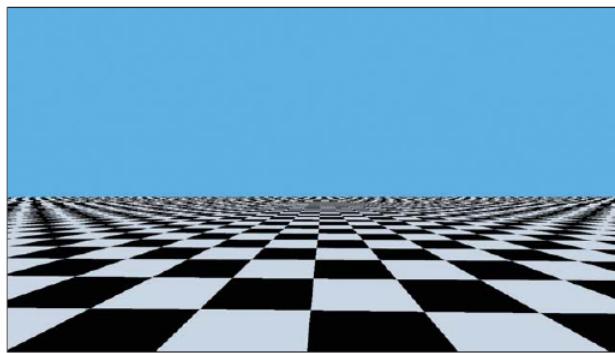
- *sa težinskim faktorima* (računa se zbir vrednosti pomnoženih težinskim faktorima i onda se taj zbir deli zbirom težinskih faktora). Naredna matrica se koristi za smanjivanje rezolucije 3 puta (analogno bi izgledala i matrica za smanjivanje rezolucije 5 puta):

1	2	1
2	4	2
1	2	1

- boja se određuje na osnovu slučajno izabranih tačaka koje pripadaju oblasti piksela.

### 13.2 Teksturni aliasing

Ako trougao ima teksturne koordinate koje prekrivaju veliku površinu slike teksture, ali sam trougao prilikom renderovanja zauzima relativno mali deo finalne slike, onda svaki piksel finalne slike odgovara velikom broju piksela slike teksture. Pristup koji smo razmatrali za svaki piksel finalne slike nalazi jedinstvenu tačku na teksturnoj slici, međutim bilo bi ispravnije više piksela teksturne slike međusobno izmešati, da bi se dobio kombinovani rezultat. Ako se ovo ne radi možemo dobiti efekat koji nazivamo **teksturni aliasing** (slika 13.2).

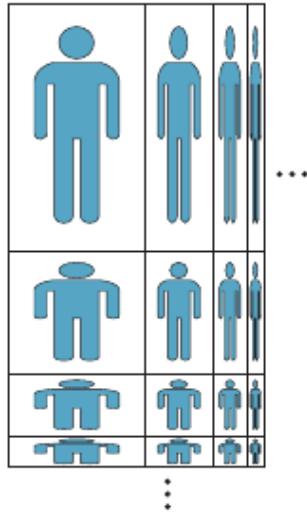


Slika 13.2: Aliasing koji potiče od rej trejsinga sa jedinstvenim uzorkom teksture šahovske table.

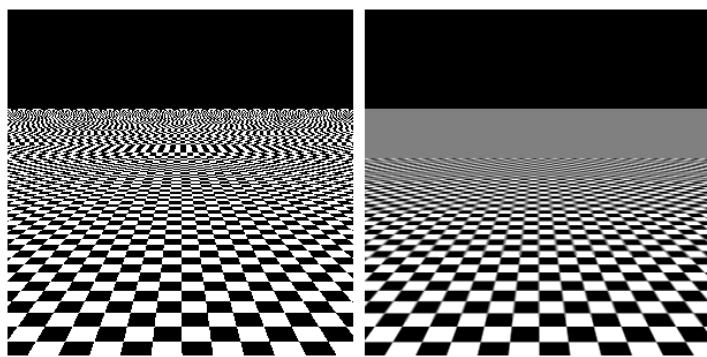
Na slici je prikazan ovaj efekat na jednostavnom primeru u kojem imamo plavo nebo i tlo koje ima teksturu šahovske table. Prilikom rej trejsinga boja svakog piksela je dakle bela, crna ili plava. Dobijena slika izgleda neprirodno, odnosno javlja se tzv. *Moire uzorak* na horizontu.

Ako izaberemo da za svaki piksel koji se renderuje vršimo mešanje piksela teksturne slike, teksturisanje postaje jako sporo. Jedan način da se ovaj efekat ublaži jeste korišćenjem preračunavanja unapred: ono se može uraditi tehnikom **MIP preslikavanja** (engl. MIP mapping).

Kod MIP preslikavanja čuva se ne samo slika, već i njene kopije smanjene različiti broj puta duž obe ose. Proces smanjivanja broja kolona za faktor 2 je jednostavan: parovi susednih kolona se uprosečavaju. Analogno se izvodi i za smanjivanje broja redova.



Slika 13.3: MIP mapa, prikazana shematski. Slika dimenzije  $n \times k$  se čuva u gornjem levom uglu, s desne strane je verzija slike dimenzije  $n \times k/2$ , zatim  $n \times k/4$  itd. Ispod je slika dimenzije  $n/2 \times k$  itd. Preostali kvadrant se popunjava verzijama slike kod kojih su obe dimenzije smanjene. Stajemo sa rekurzivnom podelom kada se slika smanji na samo 1 piksel.



Slika 13.4: Razlika u kvalitetu kada se koristi mip preslikavanje: a) bez mip preslikavanja dobija se Moire uzorak, b) slika nakon primene mip preslikavanja