Računalna grafika – laboratorijske vježbe 3

Damir Horvat

Fakultet organizacije i informatike, Varaždin

Sadržaj

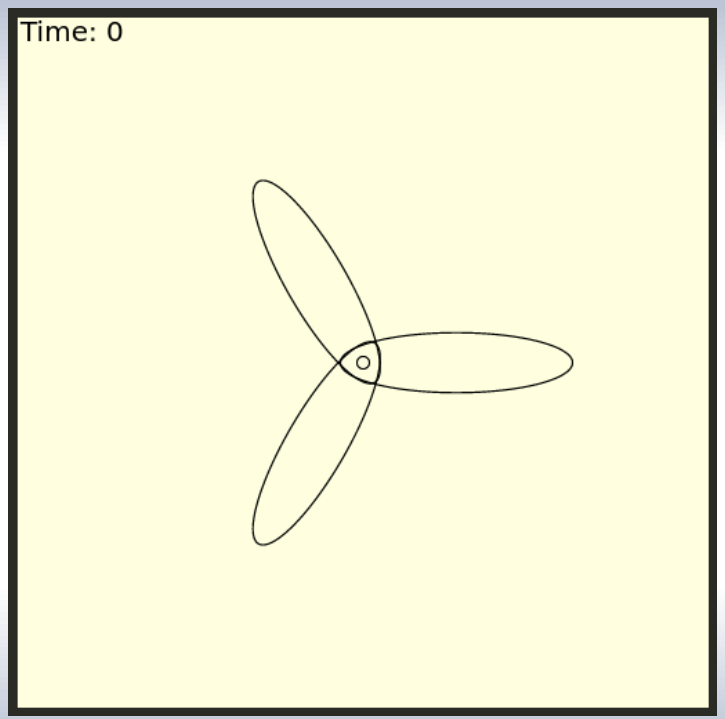
Animirani ventilator

Primjer 1. – Animirani ventilator

- Pomoću `GKS` i `MT2D` klase te metode za crtanje elipse napravite animirani ventilator kako je prikazano na sljedećem screenshotu.
- Ventilator neka rotira određenom brzinom oko svojeg središta.

- Ukoliko želite, možete omogućiti pauziranje i pokretanje animacije, npr. pomoću neke tipke na tastaturi.

Screenshot ventilatora



▸ Link na animaciju

Ideja za implementaciju

- Napraviti funkciju `ventilator()` koja će crtati ventilator. Ventilator se sastoji od jedne male kružnice i tri elipse koje treba pomoću odgovarajućih transformacija dovesti u položaj kako je prikazano na screenshotu. To je zapravo statični ventilator, tj. ventilator u početnom položaju prije nego što počne animacija.
- Nakon što uspijete dobiti statični ventilator, tada u svoju funkciju dodajte transformaciju koja će rotirati taj ventilator za kut φ oko njegovog ishodišta (male kružnice). O varijabli φ brinut će se animacija preko `requestAnimationFrame` metode koja će *kontinuirano* mijenjati tu varijablu od $0°$ do $360°$. Kada vrijednost dođe do $360°$ osigurajte da ponovo krene od $0°$ kako ne biste dobili prevelike brojeve ukoliko bi animacija trajala dulje vrijeme.
- Za animaciju ventilatora treba istovremeno rotirati sve tri elipse, svaku za određene kutove koji ovise o varijabli φ.

Kostur koda za crtanje ventilatora

```
function ventilator(){
  //primijeni transformacije za crtanje male kruznice
  elipsa(r, r); //nacrtaj malu kruznicu (odaberi polumjer r)

  //PRVA ELIPSA: rotiraj elipsu za postizanje animacije
  //transformiraj elipsu za staticni ventilator
  elipsa(a, b); //nacrtaj elipsu (odaberi poluosi a i b)

  //DRUGA ELIPSA: rotiraj elipsu za postizanje animacije
  //transformiraj elipsu za staticni ventilator
  elipsa(a, b); //nacrtaj elipsu (odaberi poluosi a i b)

  //TRECA ELIPSA: rotiraj elipsu za postizanje animacije
  //transformiraj elipsu za staticni ventilator
  elipsa(a, b); //nacrtaj elipsu (odaberi poluosi a i b)
}
```

Implementacija MT3D klase

Primjer 2. – Implementacija MT3D klase

- Implementirajte klasu MT3D matričnih reprezentacija geometrijskih transformacija u 3D.
- Pripadne metode moraju generirati matrice transformacija u homogenim koordinatama, dakle 4×4 matrice.
- Implementirajte metode za translaciju, skaliranje, rotaciju oko koordinatnih osi, zrcaljenja s obzirom na koordinatne osi i koordinatne ravnine, identitetu.
- Preko `mult(m)` metode omogućite kompoziciju geometrijskih transformacija u MT3D klasi.

Ideja implementacije MT3D klase

- Ideja je potpuno ista kao i kod klase MT2D.
- Klasa neka ima jedan (privatni) atribut `matrica` u koji se sprema geometrijska transformacija.
- Konstruktor MT3D klase postavlja atribut `matrica` na jediničnu matricu.
- Svaki put kada se pozove metoda koja generira matricu neke geometrijske transformacije, generirana matrica se preko `mult` metode množi s desne strane s matricom koja je trenutno u atributu `matrica`, a atributu `matrica` se pridružuje vrijednost tog produkta. Na taj način je omogućena kompozicija geometrijskih transformacija u MT3D klasi.

Imena metoda i njihove matrice

- Identiteta – postavlja matricu transformacije na jediničnu matricu

```
identitet()
```

- Translacija za vektor (p_x, p_y, p_z)

```
pomakni(px, py, pz)
```

- Skaliranje s faktorima s_x, s_y, s_z

```
skaliraj(sx, sy, sz)
```

- Matrica identitete

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

- Matrica translacije

$$\begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica skaliranja

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

Imena metoda i njihove matrice

- Zrcaljenje na osi x

  `zrcaliNaX()`

- Zrcaljenje na osi y

  `zrcaliNaY()`

- Zrcaljenje na osi z

  `zrcaliNaZ()`

- Matrica zrcaljenja na x-osi

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica zrcaljenja na y-osi

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica zrcaljenja na z-osi

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Imena metoda i njihove matrice

- Zrcaljenje na xy-ravnini

  `zrcaliNaXY()`

- Zrcaljenje na xz-ravnini

  `zrcaliNaXZ()`

- Zrcaljenje na yz-ravnini

  `zrcaliNaYZ()`

- Matrica zrcaljenja na xy-ravnini

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica zrcaljenja na xz-ravnini

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica zrcaljenja na yz-ravnini

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Imena metoda i njihove matrice

- Rotacija oko osi x

  `rotirajX(kut)`

- Rotacija oko osi y

  `rotirajY(kut)`

- Rotacija oko osi z

  `rotirajZ(kut)`

- Matrica rotacije oko x-osi

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & -\sin\varphi & 0 \\ 0 & \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica rotacije oko y-osi

$$\begin{bmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrica rotacije oko z-osi

$$\begin{bmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Primjeri implementacija nekih metoda

```
class MT3D {
  constructor() {
   //code
  }

  //kompozicija transformacija
  mult(m) {
   //code
  }

  //transformacije
  pomakni(px, py,pz) {
    let m = [[1,0,0,px],[0,1,0,py],[0,0,1,pz],[0,0,0,1]];
    this.mult(m);
  }
  ...
}
```

Predložak za metodu `mult`

```
mult(m) {
  let m1 = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]];
  for (let i = 0; i < 4; i++) {
    for (let j = 0; j < 4; j++) {
      for (let k = 0; k < 4; k++) {
        //code
      }
    }
  }
  this._matrica = m1;
}
```

`Ortho` klasa

Primjer 3. – `Ortho` klasa

- Implementirajte klasu `Ortho` koja omogućava ortogonalnu projekciju linija definiranih u 3D globalnom koordinatnom sustavu na xy-ravninu sa sljedećim metodama.
- `postaviNa(x, y, z)` – postavlja početak linije na poziciju (x, y, z) u 3D globalnim koordinatama
- `linijaDo(x, y, z)` – povlači liniju od posljednje zapamćene pozicije do zadane pozicije (x, y, z) u 3D globalnim koordinatama
- `trans(m)` – zadaje se matrica transformacije iz klase `MT3D` koja se primjenjuje prije crtanja u globalnim koordinatama
- `koristiBoju(c)` – linija se crta bojom `c`
- `povuciLiniju()` – povlači liniju pozivom HTML5-rutine `stroke()`

Ideja implementacije `Ortho` klase

- Treba malo modificirati već implementiranu `GKS` klasu pri čemu postojeći konstruktori ostaju uglavnom nepromijenjeni (atribut u kojemu se čuva matrica transformacije treba staviti na 4×4 jediničnu matricu).
- `trans` metoda ostaje ista kao i u `GKS` klasi.
- Metode `postaviNa` i `linijaDo` treba prilagoditi za 4×4 matrice i treba uvesti treću koordinatu z. Dakle, prije pretvaranja prirodnih koordinata u piksel koordinate, treba primijeniti matricu transformacije da se dobiju transformirane točke. Nakon toga se transformirane točke pretvore u piksel koordinate.
- S obzirom da se radi o ortogonalnoj projekciji na xy-ravninu, ta projekcija zapravo zaboravlja z-koordinatu pa je dovoljno u piksel koordinate pretvoriti samo x i y koordinate od transformirane točke koje se koriste kod crtanja na canvasu.

Računanje slike vektora odnosno radijvektora

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a_{03} \\ a_{13} \\ a_{23} \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \left[\begin{array}{ccc|c} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ \hline 0 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$x' = a_{00}x + a_{01}y + a_{02}z + a_{03}$$
$$y' = a_{10}x + a_{11}y + a_{12}z + a_{13}$$
$$z' = a_{20}x + a_{21}y + a_{22}z + a_{23}$$

nije potrebno računati

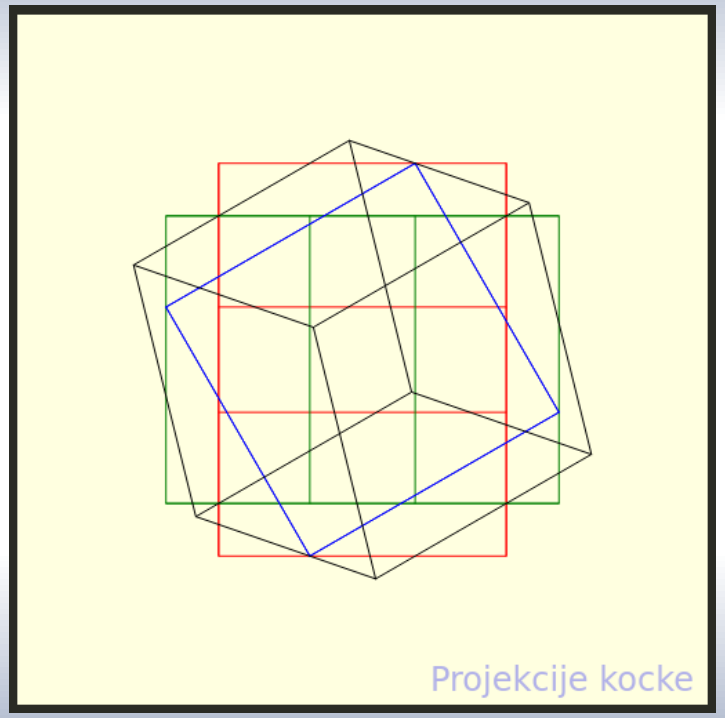
$$x_{\text{pix}} = s_x \cdot x' + p_x$$
$$y_{\text{pix}} = s_y \cdot y' + p_y$$

Ortogonalna projekcija kocke

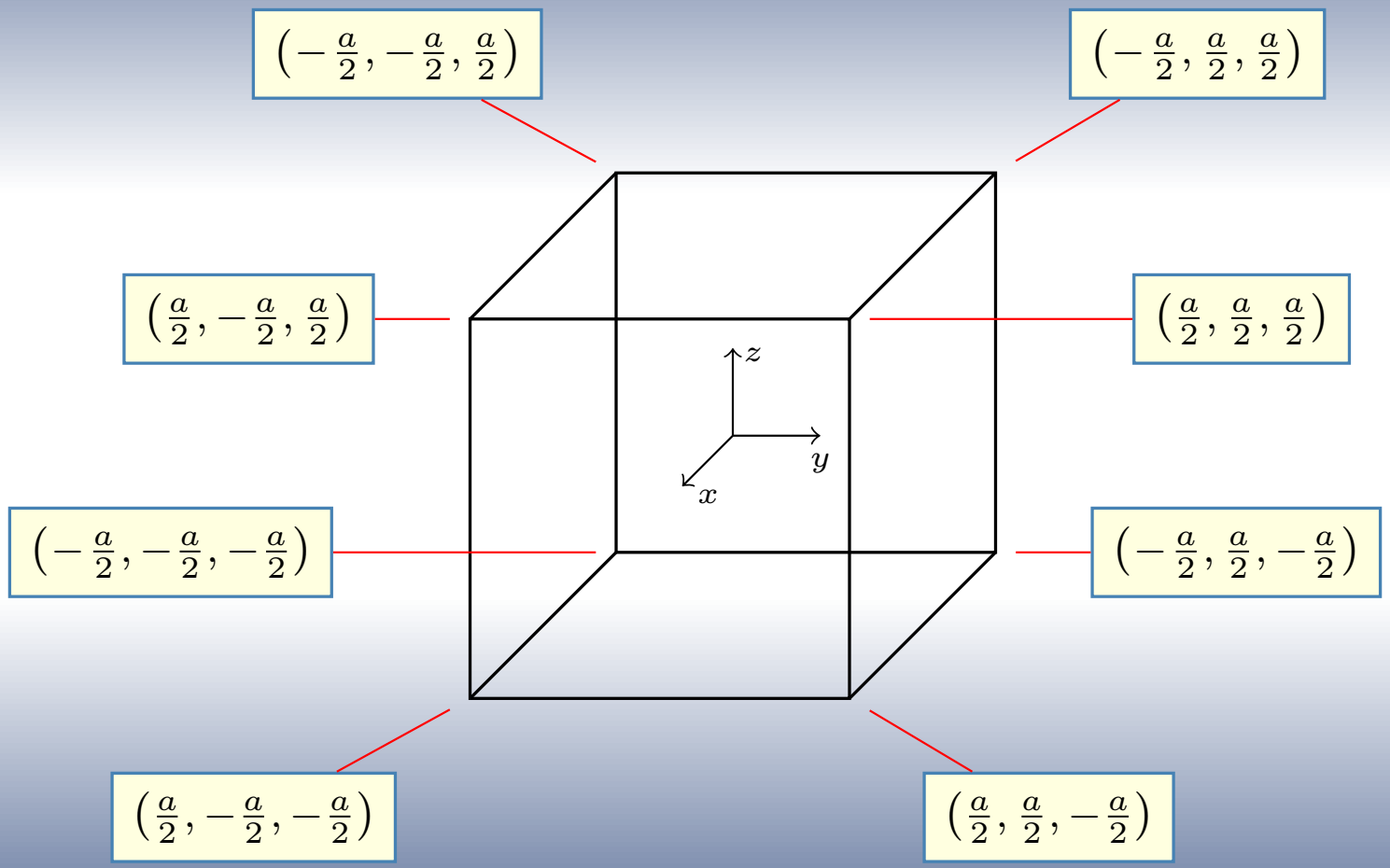
Primjer 4. – Ortogonalna projekcija kocke

- Napišite proceduru za crtanje kocke duljine stranice a kojoj se središte nalazi u ishodištu koordinatnog sustava.
- Na istoj slici pomoću napisane procedure, `Orho` i `MT3D` klasa nacrtajte ortogonalne projekcije sljedećih kocaka. Uzmite $a = 1$ za duljinu stranice.
- U crvenoj boji nacrtajte ortogonalnu projekciju početne kocke koja je zarotirana oko x-osi za 30°. U zelenoj boji nacrtajte ortogonalnu projekciju početne kocke koja je zarotirana oko y-osi za 30°. U plavoj boji nacrtajte ortogonalnu projekciju početne kocke koja je zarotirana oko z-osi za 30°.
- U crnoj boji nacrtajte ortogonalnu projekciju početne kocke koja je zarotirana oko x, y i z osi tim redom svaki puta za 30°. Pazite na poredak transformacija u kôdu.

Screenshot ortogonalnih projekcija kocki



Koordinate vrhova kocke



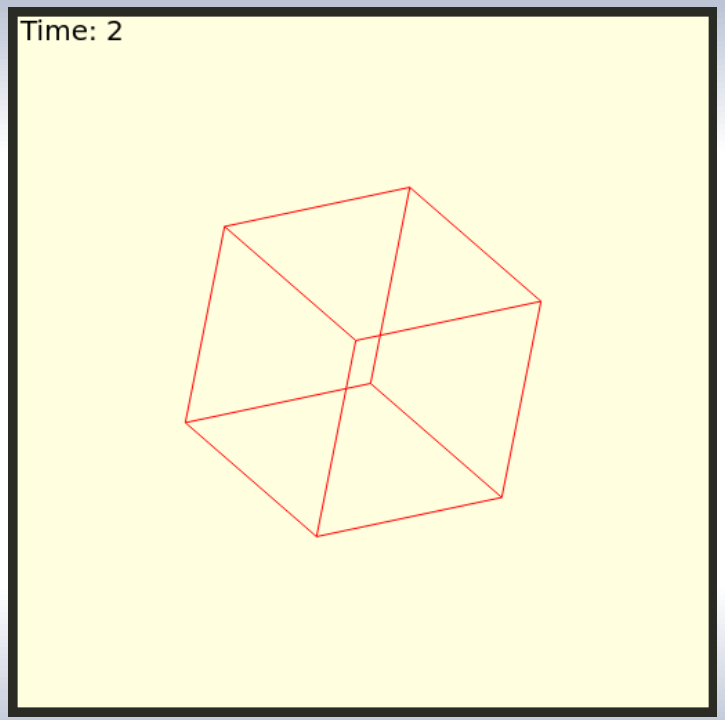
Animacija rotacija kocke

Primjer 5. – Animacija rotacija kocke oko koordinatnih osi

- Napravite animaciju rotacije kocke koja se istovremeno okreće oko sve tri koordinatne osi.
- Koristite `Ortho` i `MT3D` klase.

- Ukoliko želite, možete omogućiti pauziranje i pokretanje animacije, npr. pomoću neke tipke na tastaturi.

Screenshot rotacije kocke



▸ Link na animaciju