Računalna grafika – laboratorijske vježbe 4

Damir Horvat

Fakultet organizacije i informatike, Varaždin

Sadržaj

Teorija

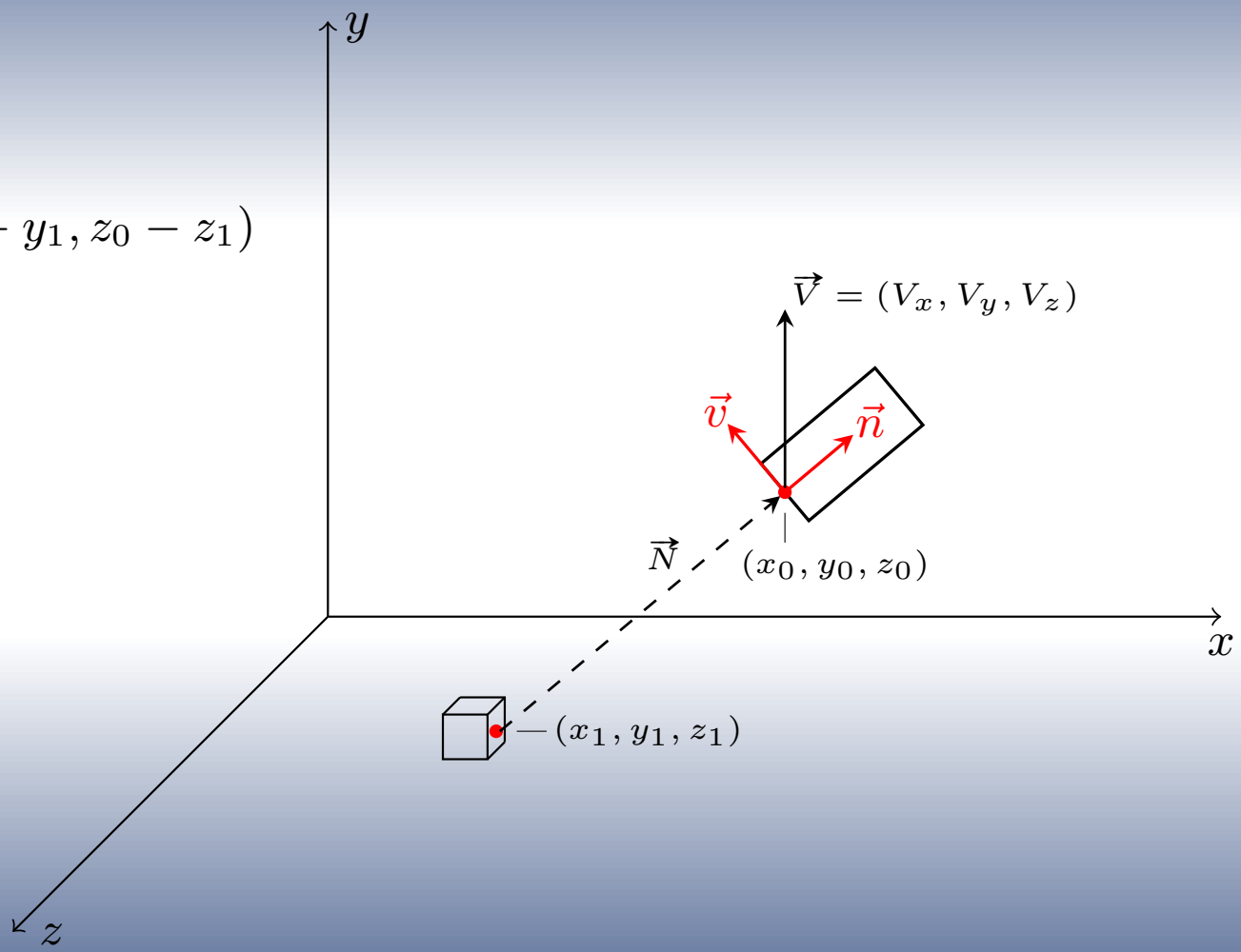
$$\vec{N} = (x_0 - x_1, y_0 - y_1, z_0 - z_1)$$

$$\vec{n} = \frac{\vec{N}}{|\vec{N}|}$$

$$\vec{U} = \vec{V} \times \vec{n}$$

$$\vec{u} = \frac{\vec{U}}{|\vec{U}|}$$

$$\vec{v} = \vec{n} \times \vec{u}$$



Teorija

- Želimo da se koordinatni sustav kamere

$$\left(T_0; (\vec{u}, \vec{v}, \vec{n})\right)$$

poklopi s globalnim koordinatnim sustavom

$$\left(O; (\vec{i}, \vec{j}, \vec{k})\right)$$

- Drugim riječima, želimo kameru pozicionirati u ishodište globalnog koordinatnog sustava tako da gleda u negativnom smjeru z-osi.

Teorija

- Najprije translacijom za vektor $(-x_0, -y_0, -z_0)$ pomaknemo kameru u ishodište globalnog koordinatnog sustava. Ta transformacija ima u homogenim koordinatama matrični zapis

$$P = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Teorija

- Na kraju treba vektore iz ortonormirane baze $(\vec{u}, \vec{v}, \vec{n})$ redom preslikati u vektore ortonormirane baze $(\vec{i}, \vec{j}, \vec{k})$.
- Iz teorije linearnih operatora znamo da postoji jedinstveni ortogonalni operator koji jednu ortonormiranu bazu preslikava u neku drugu ortonormiranu bazu.
- Ortogonalni operatori su jako zgodni jer se računanje njihovog inverza svodi na transponiranje. Drugim riječima, ako je R ortogonalni operator, tada je $R^{-1} = R^T$.

Teorija

- Dakle, postoji jedinstveni ortogonalni operator R takav da je

$$R(\vec{u}) = \vec{i}, \quad R(\vec{v}) = \vec{j}, \quad R(\vec{n}) = \vec{k}.$$

- Tada je

$$R^T(\vec{i}\,) = \vec{u}, \quad R^T(\vec{j}\,) = \vec{v}, \quad R^T(\vec{k}) = \vec{n}$$

pa u kanonskoj bazi operator R^T ima matrični zapis

$$R^T = \begin{bmatrix} u_x & v_x & n_x \\ u_y & v_y & n_y \\ u_z & v_z & n_z \end{bmatrix}.$$

Teorija

- Stoga operator R u kanonskoj bazi ima matrični zapis

$$R = \begin{bmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{bmatrix},$$

a u homogenim koordinatama

$$R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Transformacija pogleda

- Transformacija iz globalnog koordinatnog sustava (GKS) u koordinatni sustav kamere (KSK)

$$\begin{bmatrix} x_{\text{KSK}} \\ y_{\text{KSK}} \\ z_{\text{KSK}} \\ 1 \end{bmatrix} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{GKS}} \\ y_{\text{GKS}} \\ z_{\text{GKS}} \\ 1 \end{bmatrix}$$

- Prilikom transformacije pogleda, transformacija se vrši na svim objektima u 3D sceni. Na taj način nakon transformacije pogleda, svi objekti ostaju i dalje u istom položaju prema kameri. Jedino što dobivamo transformacijom pogleda jest da se kamera nalazi u ishodištu i da gleda u negativnom smjeru z-osi u GKS sustavu.

Zašto radimo transformaciju pogleda?

- U KSK je lako ispitati kada je točka iza kamere. Točka je iza kamere ako je $z_{\text{KSK}} > 0$.

- Prilikom projiciranja na dvodimenzionalni ekran treba odrediti bliže i udaljenije objekte tako da znamo koji objekt prikazati na ekranu, a koji sakriti ukoliko se oni projiciraju u iste piksele na ekranu. U KSK bliži objekt je onaj čije točke imaju veće negativne z-koordinate. Drugim riječima, ako za točke $T_1(x_1, y_1, z_1)$ i $T_2(x_2, y_2, z_2)$ u KSK vrijedi

$$z_1 < 0, \quad z_2 < 0, \quad z_1 > z_2$$

tada su obje točke ispred kamere i točka T_1 se nalazi ispred točke T_2. Preciznije, točka T_1 se nalazi u ravnini $z = z_1$ koja je ispred ravnine $z = z_2$ u kojoj se nalazi točka T_2.

Zašto radimo transformaciju pogleda?

- Projekciju 3D scene uvijek radimo na neku ravninu koja je okomita na smjer gledanja kamere, tj. koja je okomita na z-os u koordinatnom sustavu kamere.
- U KSK ortogonalna projekcija se svodi samo na ispuštanje z-koordinate.
- Isto tako, kasnije ćemo vidjeti da se i perspektivna projekcija u KSK obavlja prema vrlo jednostavnim formulama.
- U GKS je kompliciranije raditi projekcije s obzirom da ravnina projekcije koja je okomita na smjer gledanja kamere ne mora biti paralalena s niti jednom od koordinatnih ravnina iz globalnog koordinatnog sustava.

Dodavanje kamere u scenu

Primjer 1. – implementacija metode `postaviKameru`

- Klasi `MT3D` dodajte metodu

  `postaviKameru(x0, y0, z0, x1, y1, z1, Vx, Vy, Vz)`

  koja omogućuje transformaciju u koordinatni sustav kamere postavljene u točki (x_0, y_0, z_0) globalnog koordinatnog sustava. Kamera je usmjerena prema točki (x_1, y_1, z_1), a vektor (V_x, V_y, V_z) određuje smjer prema gore (*view-up vector*), tj. položaj y-osi koordinatnog sustava kamere.

- Matrica transformacije koja se generira kod poziva metode `postaviKameru` ostaje zapamćena i primijenjuje se kod svakog sljedećeg crtanja (mijenja se tek novim pozivom metode `postaviKameru`).

Modifikacija `MT3D` klase

- Treba dodati (privatni) atribut `kamera` u kojemu se čuva transformacija pogleda. U konstruktoru se taj atribut postavi na jediničnu matricu reda 4. To znači da je po defaultu kamera već u ishodištu i da gleda u negativnom smjeru z-osi.
- Treba implementirati metodu `VP` za računanje vektorskog produkta.
- Treba implementirati metodu `mnoziMatrice` za računanje produkta dvije matrice. Kôd je sličan kao za `mult` metodu, samo na ulazu trebaju biti dva parametra (matrice koje zelimo pomnožiti u danom poretku).
- Treba implementirati metodu `postaviKameru` koja će generirati transformaciju pogleda.

Modifikacija `Ortho` klase

- Treba modificirati metodu `trans` tako da uz matricu transformacije bude uključena i matrica pogleda, tj. da se uvijek nakon geometrijskih transformacija primijeni i transformacija pogleda.

Stari kod

```
trans(m) {
  this._matrica = m._matrica;
}
```

Novi kod

```
trans(m) {
  this._matrica = m.mnoziMatrice(m._kamera, m._matrica);
}
```

Metoda za računanje vektorskog produkta

- Sjetimo se formule

$$\vec{u} \times \vec{v} = \begin{vmatrix} \vec{i} & \vec{j} & \vec{k} \\ u_0 & u_1 & u_2 \\ v_0 & v_1 & v_2 \end{vmatrix} = \left(u_1 v_2 - u_2 v_1,\ u_2 v_0 - u_0 v_2,\ u_0 v_1 - u_1 v_0\right)$$

- Implementacija vektorskog produkta

```
VP(u,v) {
    let vek = [0,0,0];
    vek[0] = //code
    //code
    return vek;
}
```

Predložak za metodu `mnoziMatrice`

- $A = [a_{ij}]$ tipa (m, n), $B = [b_{ij}]$ tipa (n, p), $AB = [c_{ij}]$ tipa (m, p)

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, p$$

```
mnoziMatrice(m1,m2) {
  let rez = [[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
      }
    }
  }
  return rez;
}
```

Matrica za transformaciju pogleda

$$\vec{N} = (x_0 - x_1, y_0 - y_1, z_0 - z_1), \quad \vec{U} = \vec{V} \times \vec{n}, \quad \vec{v} = \vec{n} \times \vec{u}$$

$$\begin{bmatrix} x_{\text{KSK}} \\ y_{\text{KSK}} \\ z_{\text{KSK}} \\ 1 \end{bmatrix} = \begin{bmatrix} u_0 & u_1 & u_2 & 0 \\ v_0 & v_1 & v_2 & 0 \\ n_0 & n_1 & n_2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{GKS}} \\ y_{\text{GKS}} \\ z_{\text{GKS}} \\ 1 \end{bmatrix}$$

$$\vec{n} = \frac{\vec{N}}{|\vec{N}|}$$

$$\vec{u} = \frac{\vec{U}}{|\vec{U}|}$$

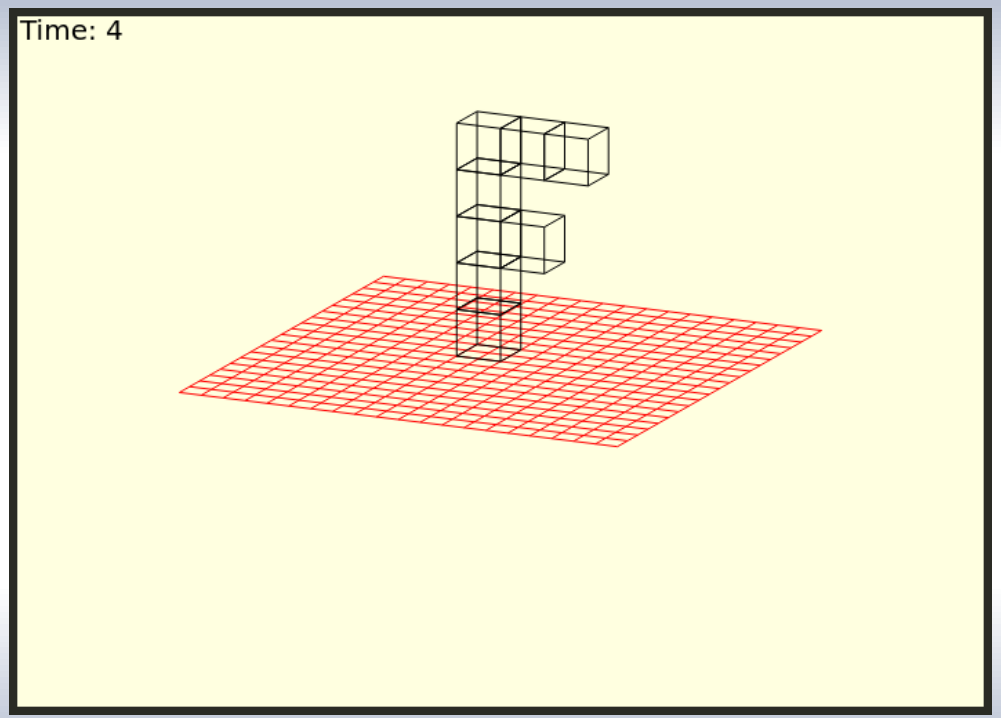
$$\begin{bmatrix} x_{\text{KSK}} \\ y_{\text{KSK}} \\ z_{\text{KSK}} \\ 1 \end{bmatrix} = \begin{bmatrix} u_0 & u_1 & u_2 & -u_0x_0 - u_1y_0 - u_2z_0 \\ v_0 & v_1 & v_2 & -v_0x_0 - v_1y_0 - v_2z_0 \\ n_0 & n_1 & n_2 & -n_0x_0 - n_1y_0 - n_2z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\text{GKS}} \\ y_{\text{GKS}} \\ z_{\text{GKS}} \\ 1 \end{bmatrix}$$

Animacija slova F

Primjer 2. – Slovo F u `Ortho` klasi

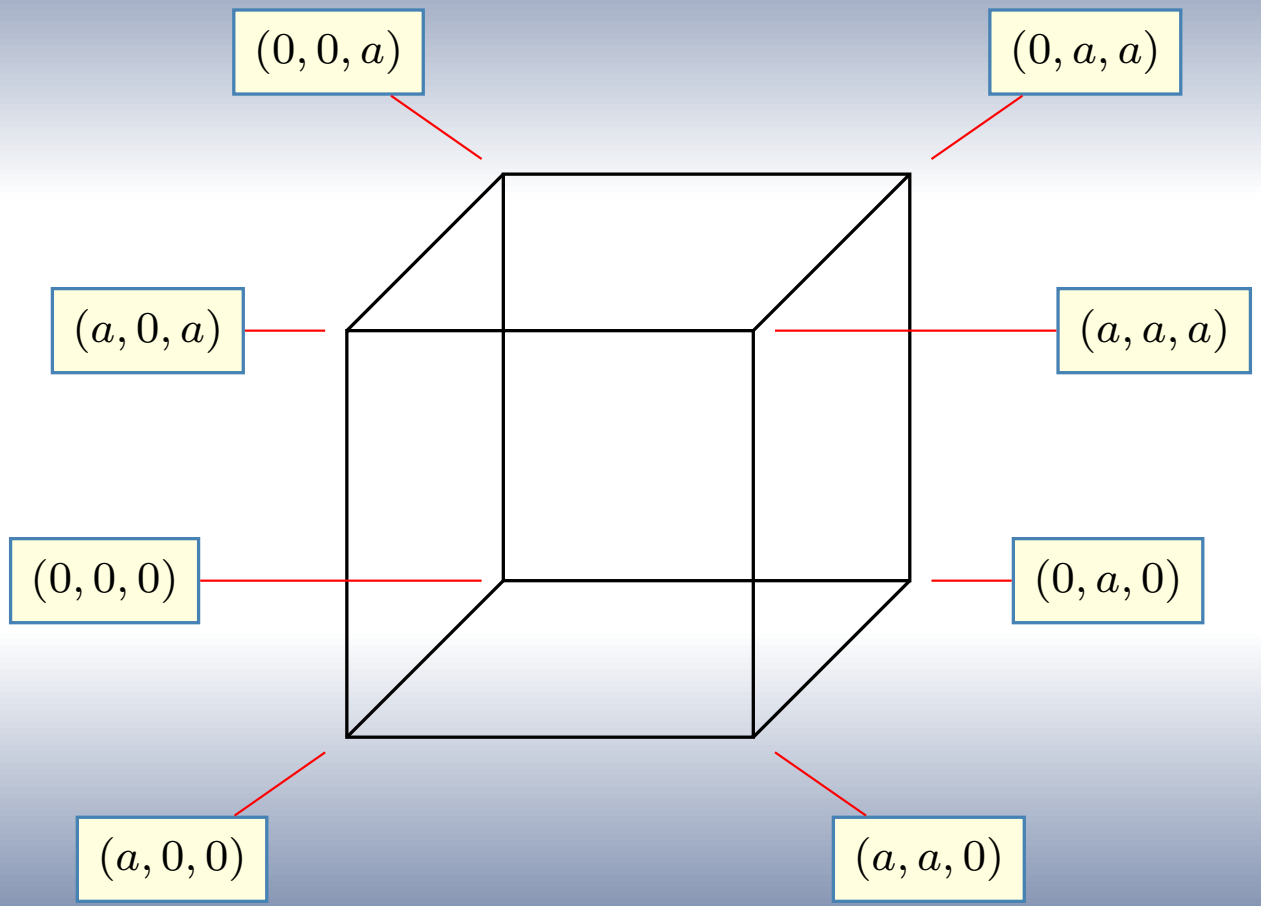
- Mrežom ravnih linija vizualizirajte xz-ravninu, a na nju postavite stilizirano slovo F sačinjeno od osam kocaka kako je prikazano na sljedećem slajdu.
- Kamerom kružite oko slova F mijenjajući više puta visinu na kojoj se nalazi kamera unutar određenih granica.
- Napravite animaciju u kojoj se visina kamere kontinuirano mijenja.

Screenshot slova F u ortogonalnoj projekciji



▸ Link na animaciju

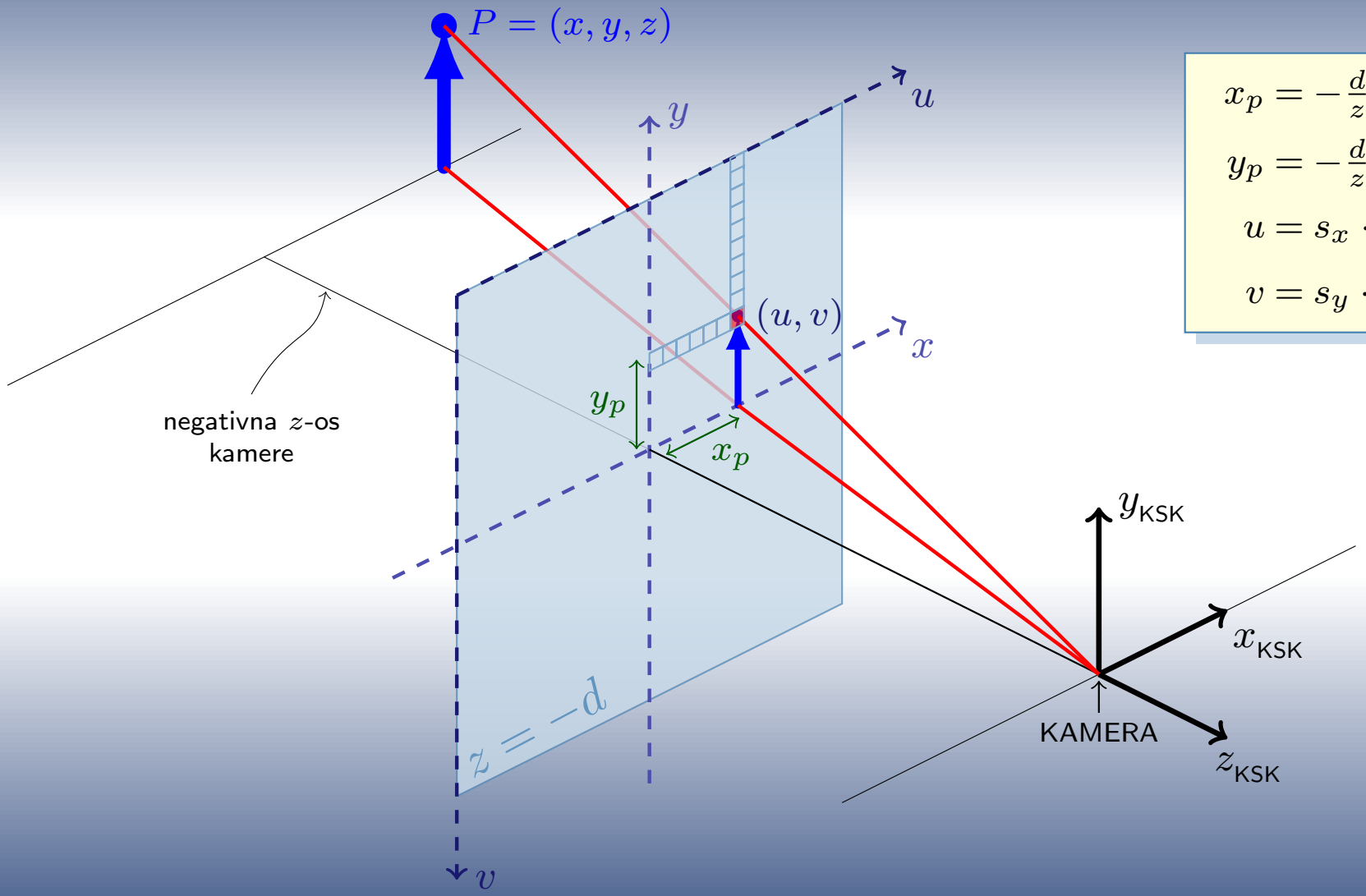
Koordinate vrhova kocke



Slovo F u `Ortho` klasi – parametri kamere

- Po želji možete napraviti drugu animaciju u kojoj se visina kamere mijenja diskretno, npr. svake dvije sekunde.
- Poigrajte se s ostalim parametrima kamere, npr. da se prilikom promjene visine kamere mijenja i njezin pogled tako da je y-koordinata položaja kamere jednaka y-koordinati točke prema kojoj kamera gleda (kamera stalno gleda horizontalno prema objektu).
- Promijenite *view-up vector* i uočite što se događa. Također približite ili udaljite kameru od objekta i uočite što se događa.

Teorija



$$x_p = -\frac{d}{z}x$$

$$y_p = -\frac{d}{z}y$$

$$u = s_x \cdot x_p + p_x$$

$$v = s_y \cdot y_p + p_y$$

Teorija

- Matrica promatrane perspektivne projekcije u homogenim koordinatama

$$\lambda \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{bmatrix}, \quad \lambda \in \mathbb{R} \setminus \{0\}$$

Naime, vrijedi

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ -\frac{z}{d} \end{bmatrix} = -\frac{z}{d} \begin{bmatrix} -\frac{d}{z}x \\ -\frac{d}{z}y \\ -d \\ 1 \end{bmatrix}$$

`Persp` klasa

Primjer 3. – `Persp` klasa

- Implementirajte klasu `Persp` koja omogućuje perspektivno projiciranje i ima sljedeće metode.
- `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 `Persp` klase

- Treba malo modificirati već implementiranu `Ortho` klasu tako da dodamo još jedan (privatni) atribut `distance` u kojemu se čuva udaljenost kamere od ravnine na koju se projicira.
- Postojećim konstruktorima iz `Ortho` klase treba dodati još jednu varijablu za udaljenost kamere od ravnine na koju se projicira.
- `trans` metoda ostaje ista kao i u `Ortho` klasi.
- Metode `postaviNa` i `linijaDo` treba također malo modificirati. Kao u `Ortho` klasi, prije pretvaranja prirodnih koordinata u piksel koordinate, treba primijeniti matricu transformacije da se dobiju transformirane točke. Nakon toga se transformirane točke perspektivno projiciraju na dvodimenzionalnu ravninu (u ovom slučaju se z-koordinata ne zaboravlja, već se koristi za određivanje koordinata projiciranih točaka). Nakon toga projicirane točke prevodimo u piksel koordinate.

Novosti u konstruktoru

```
class Persp {
  constructor(...,dist,...) {
     //code
     this._distance = dist;
     //code
  }
  //kodovi ostalih metoda
}
```

Računanje slike vektora odnosno radijvektora

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

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

Ortogonalna projekcija

$$x_{\text{pr}} = x'$$
$$y_{\text{pr}} = y'$$

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

$$x_{\text{pr}} = -\frac{d}{z'}x' \qquad y_{\text{pr}} = -\frac{d}{z'}y'$$

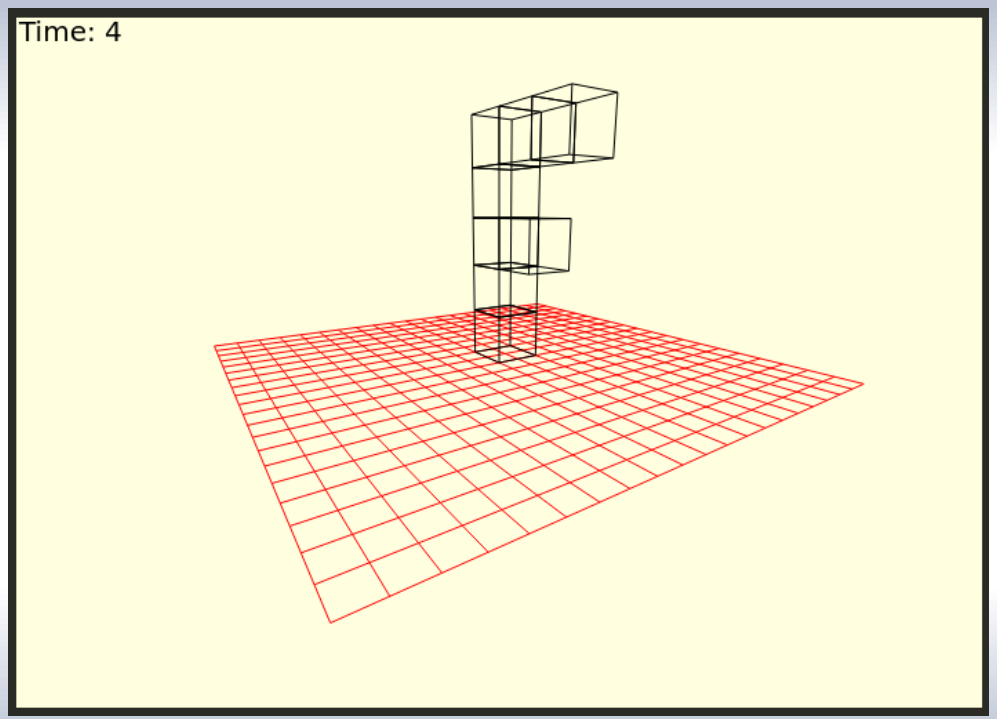
$$x_{\text{pix}} = s_x \cdot x_{\text{pr}} + p_x \qquad y_{\text{pix}} = s_y \cdot y_{\text{pr}} + p_y$$

Animacija slova F

Primjer 4. – Slovo F u `Persp` klasi

- Mrežom ravnih linija vizualizirajte xz-ravninu, a na nju postavite stilizirano slovo F sačinjeno od osam kocaka kako je prikazano na sljedećem slajdu.
- Kamerom kružite oko slova F mijenjajući visinu na kojoj se nalazi kamera unutar određenih granica.
- Napravite animaciju gdje se visina kamere kontinuirano mijenja.
- Za crtanje koristite `Persp` klasu.

Screenshot slova F u perspektivnoj projekciji

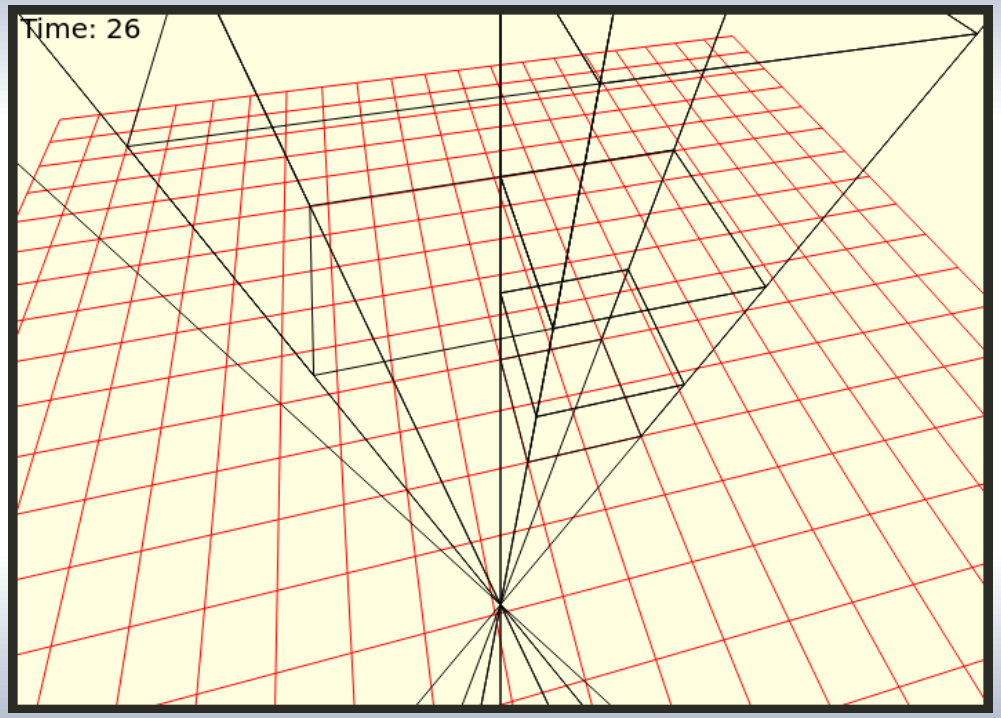


▸ Link na animaciju

Slovo F u `Persp` klasi – igranje s parametrima

- Mijenjajte udaljenost d ravnine projiciranja od kamere u konstruktoru `Persp`. Što uočavate?
- Približavajte i udaljavajte kameru od slova F, tako da se kamera šeće po kružnicama većeg ili manjeg polumjera oko ishodišta u xz-ravnini.
- Ukoliko je kamera na kružnici dovoljno velikog polumjera, tada je s animacijom i slikom sve u redu.
- Međutim, ukoliko se kamera "previše" približi slovu F, događaju se određene anomalije kako je prikazano na screenshotu na sljedećem slajdu.
- Zbog čega se događaju te anomalije? Kako to možemo popraviti?
- Uočite da nije problem ukoliko približavate projicirajuću ravninu kameri, samo je problem kada se kamera "previše" približi objektima u 3D sceni.

Anomalije u `Persp` klasi



Dodatak

Opis problema

- Trebamo malo inteligentnije implementirati `postaviNa` i `linijaDo` metode ukoliko želimo izbjeći anomalije prilikom približavanja kamere objektima u 3D sceni.
- Naime, perspektivna projekcija neke beskonačno daleke točke preslikava u konačne točke i neke konačne točke preslikava u beskonačno daleke točke. Kratko rečeno, to je zapravo uzrok anomalijama koje smo vidjeli na prethodnom slajdu.
- Ograničimo li se na našu promatranu perspektivnu projekciju, ta projekcija svaki pravac p koji siječe ravninu $z = 0$ (i ne prolazi kroz ishodište) "lomi" na dva dijela u sljedećem smislu.

Opis problema

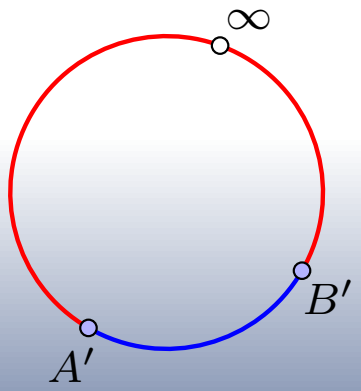
- Točku u kojoj pravac p siječe ravninu $z = 0$, perspektivna projekcija preslikava u beskonačno daleku točku koja se nalazi u toj ravnini. Isto tako, beskonačno daleku točku pravca p preslikava u točku koja je presjek ravnine projekcije i paralele s pravcem p kroz centar projekcije.
- Ukoliko spajamo linijom dvije točke, od kojih je jedna ispred ravnine $z = 0$ (ima negativnu z-koordinatu), a jedna iza ravnine $z = 0$ (ima pozitivnu z-koordinatu), tada u projekciji spajamo te točke pogrešnim segmentom. Trebali bismo te dvije projicirane točke spajati preko beskonačno daleke točke, ali s njom u praksi ne raspolažemo. Naime, kako dužina siječe ravninu $z = 0$, taj presjek se projicirao u beskonačno daleku točku pa se dužina "razlomila" na dva dijela.

Opis problema

- Perspektivna projekcija svaki pravac koji ne prolazi kroz centar projekcije ponovo preslikava u pravac (jer se može prikazati matrično u homogenim koordinatama), samo je problem što jednu njegovu konačnu točku preslikava u beskonačno daleku pa u praksi zapravo imamo "razlomljeni" pravac jer ne možemo uhvatiti tu beskonačno daleku točku na ekranu.
- Sa stajališta projektivne geometrije, projektivni pravac je zapravo kružnica, a euklidski pravac je kružnica iz koje je izbačena jedna točka i onda tu izbačenu točku zovemo beskonačno dalekom točkom euklidskog pravca. Na projektivnom modelu možemo zornije objasniti ponašanje perspektivne projekcije.

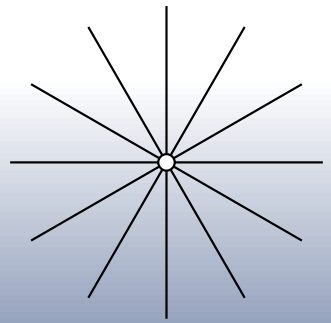
Opis problema

- Ukoliko dužina $\overline{AB}$ siječe ravninu projekcije, tada se ona u projekciji preslika u onaj dio pravca koji sadrži beskonačno daleku točku (crveni dio na slici). U trenutnoj implementaciji `linijaDo` metode u projekciji zapravo krajeve projicirane dužine $\overline{A'B'}$ spajamo preko dijela koji ne sadrži beskonačno daleku točku (plavi dio na slici) jer drukčije niti ne možemo s obzirom da nam beskonačno daleka točka nije dostupna. Dakle, umjesto da crtamo crvenu dužinu, mi zapravo crtamo plavu dužinu i zbog toga se događaju uočene anomalije.



Opis problema

- Na screenshotu smo mogli uočiti nacrtani pramen pravaca kroz jednu točku kako je prikazano na donjoj slici. To nije ništa čudno jer kod slova F ima dosta paralelnih dužina i ako te dužine sijeku ravninu $z = 0$, tada se zajednička beskonačno daleka točka pripadnih paralelnih pravaca preslikava u konačnu točku u projicirajućoj ravnini. S obzirom da crtamo pogrešne dužine (one dužine u koje su se projicirali beskonačni dijelovi pripadnih paralenih pravaca), sve takve nacrtane dužine u projekciji prolaze kroz istu točku, tj. onu točku u koju se projicirala beskonačno daleka točka pripadnih paralelnih pravaca.



Nova implementacija

- Treba paziti koje točke spajamo dužinom, a koje ne spajamo. Drugim riječima, ne moramo uvijek spajati linijom od zadnje zapamćene pozicije do trenutno pozvane pozicije.
- Stoga u klasu `Persp` trebamo uvesti još tri (privatna) atributa $x_{\texttt{last}}, y_{\texttt{last}}$ i $z_{\texttt{last}}$ u kojima se pamti zadnja pozicija do koje se došlo s crtanjem linije.
- U metodi `postaviNa` moramo zapamtiti poziciju od koje smo počeli crtati liniju tako da tu poziciju možemo testirati u `linijaDo` metodi u kojoj će se donijeti odluka o crtanju linije. Tu poziciju pamtimo u atributima $x_{\texttt{last}}$, $y_{\texttt{last}}$ i $z_{\texttt{last}}$ (to su koordinate u koordinatnom sustavu kamere).
- U `linijaDo` metodi treba paziti koje točke spajamo dužinom. Nakon što pozovemo metodu `linijaDo` s točkom (x, y, z), najprije koordinate te točke prevedemo u koordinatni sustav kamere pa dobivamo točku (x_c, y_c, z_c).

Nova implementacija

Razlikujemo četiri slučaja.

1. $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ i (x, y, z) se nalaze ispred kamere, tj. $z_{\texttt{last}} < 0$ i $z_c < 0$. Tada spojimo točke linijom, tj. trenutna implementacija metode `linijaDo` se dobro ponaša u ovom slučaju. Varijable $x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}}$ stavimo na nove vrijednosti x_c, y_c, z_c.

2. $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ i (x, y, z) se nalaze iza kamere, tj. $z_{\texttt{last}} > 0$ i $z_c > 0$. U tom slučaju ne crtamo nikakvu liniju, nego samo varijablama $x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}}$ redom proslijedimo nove vrijednosti x_c, y_c, z_c.

Nova implementacija

3. Točka $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ je iza kamere, a točka (x, y, z) se nalazi ispred kamere. Drugim riječima, $z_{\texttt{last}} > 0$ i $z_c < 0$. U ovom slučaju prelazimo iz područja iza kamere u područje ispred kamere pa ne smijemo crtati dio dužine koji se nalazi iza kamere. Trebamo pronaći presjek pravca kroz točke $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ i (x_c, y_c, z_c) s ravninom $z_c = 0$. Nakon toga crtamo dužinu od dobivenog presjeka do točke (x_c, y_c, z_c) i varijablama $x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}}$ redom proslijedimo nove vrijednosti x_c, y_c, z_c.

Nova implementacija

4. Točka $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ je ispred kamere, a točka (x, y, z) se nalazi iza kamere. Drugim riječima, $z_{\texttt{last}} < 0$ i $z_c > 0$. U ovom slučaju prelazimo iz područja ispred kamere u područje iza kamere pa ne smijemo crtati dio dužine koji se nalazi iza kamere. Trebamo pronaći presjek pravca kroz točke $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ i (x_c, y_c, z_c) s ravninom $z_c = 0$. Nakon toga crtamo dužinu od točke $(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ do dobivenog presjeka i varijablama $x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}}$ redom proslijedimo nove vrijednosti x_c, y_c, z_c.

Nova implementacija

Napomena

- Kako bismo izbjegli neželjene efekte, u praksi je bolje broj 0 zamijeniti s nekim bliskim negativnim brojem, npr. -0.01.
- Umjesto testiranja $z_c < 0$, $z_{\texttt{last}} > 0$, ..., u implementaciji radimo testiranja $z_c \leqslant -0.01$, $z_{\texttt{last}} \geqslant -0.01$, ...
- Umjesto traženja presjeka pravca s ravninom $z_c = 0$, tražimo presjek s ravninom $z_c = -0.01$.

Presjek pravca i ravnine

- Tražimo presjek pravca (dužine) kroz točke $T_1(x_{\texttt{last}}, y_{\texttt{last}}, z_{\texttt{last}})$ i $T_2(x_c, y_c, z_c)$ s ravninom $z = -0.01$.
- Pravac T_1T_2 možemo gledati kao na pravac koji prolazi točkom T_1 i ima vektor smjera $\overrightarrow{T_1T_2} = (x_c - x_{\texttt{last}},\, y_c - y_{\texttt{last}},\, z_c - z_{\texttt{last}})$. Stoga su njegove parametarske jednadžbe

$$T_1T_2 \ldots \begin{cases} x = x_{\texttt{last}} + (x_c - x_{\texttt{last}})t \\ y = y_{\texttt{last}} + (y_c - y_{\texttt{last}})t, \qquad t \in \mathbb{R}. \\ z = z_{\texttt{last}} + (z_c - z_{\texttt{last}})t \end{cases} \tag{♣}$$

Presjek pravca i ravnine

- Uvrstimo parametarske jednadžbe (♣) pravca T_1T_2 u opći oblik jednadžbe ravnine $z = -0.01$ i dobivamo

$$z_{\texttt{last}} + (z_c - z_{\texttt{last}})t = -0.01$$

iz čega slijedi

$$t = \frac{z_{\texttt{last}} + 0.01}{z_{\texttt{last}} - z_c}. \qquad (\spadesuit)$$

- Točku presjeka dobivamo tako da vrijednost parametra t iz (♠) uvrstimo u parametarske jednadžbe (♣) pravca T_1T_2. Dakle, točka presjeka ima koordinate

$$\big(x_{\texttt{last}} + (x_c - x_{\texttt{last}})t,\ y_{\texttt{last}} + (y_c - y_{\texttt{last}})t,\ -0.01\big).$$