Gledanje i perspektiva

Dr. sci. Emir Skejić, vanr. prof.
Fakultet elektrotehnike Tuzla

Homogene transformacije

$$\mathbf{v}' = \mathbf{M} \cdot \mathbf{v}$$

$$\begin{bmatrix} v'_x \\ v'_y \\ v'_z \\ 1 \end{bmatrix} = \begin{bmatrix} a_x & b_x & c_x & d_x \\ a_y & b_y & c_y & d_y \\ a_z & b_z & c_z & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix}$$

$$v'_x = a_x v_x + b_x v_y + c_x v_z + d_x$$

$$v'_y = a_y v_x + b_y v_y + c_y v_z + d_y$$

$$v'_z = a_z v_x + b_z v_y + c_z v_z + d_z$$

$$1 = 0v_x + 0v_y + 0v_z + 1$$

3D transformacije

- Do sada smo proučili mnoštvo korisnih 3D transformacija:
  - Rotacija
  - Translacija
  - Skaliranje
  - Smicanje
  - Refleksija
- Ovo su primjeri *afinih* transformacija
- Sve transformacije mogu biti kombinirane u jednu 4x4 matricu, sa [0 0 0 1] u zadnjem redu
- To podrazumijeva 12 konstanti ili 12 stepeni slobode (engl. *Degree od Freedom – DOF*)

ABCD vektori

- Rekli smo da se informacije o translaciji lako ekstrahuju direktno iz matrice, dok se informacije o rotaciji/skaliranju/smicanju/refleksiji kodiraju u gornjem 3x3 segmentu matrice
- 9 konstanti u gornjoj 3x3 matrici gradi 3 vektora pod imenom **a**, **b** i **c**
- Ako o matrici razmišljamo kao o transformaciji iz prostora objekta u svjetski prostor, tada je vektor **a** ustvari x osa objekta transformirana u svjetski prostor, **b** je y osa objekta u svjetskom prostoru, a **c** je z osa objekta u svjetskom prostoru
- **d** je, naravno, pozicija u svjetskom prostoru.

Primjer: Yaw

- Kosmički brod lebdi u vasioni, s matricom **W**. Pilot želi zakrenuti brod za 10 stepeni ulijevo (engl. *yaw*). Pokazati kako modificirati matricu **W** da bi se to postiglo.
- Napomena:
  - Rotacija oko x-ose naziva se *penjanje* ili *poniranje* (engl. *pitch*)
  - Rotacija oko y-ose naziva se *okretanje* (engl. *yaw*)
  - Rotacija oko z-ose naziva se *kotrljanje* (engl. *roll*)

Primjer: Yaw

- Jednostavno, zarotirajmo matricu **W** oko njenog vlastitog vektora **b**, koristeći matricu "rotacije oko proizvoljne ose":

$$\mathbf{M} = \mathbf{T}(\mathbf{W.d}) \cdot \mathbf{R}_b(\mathbf{W.b}, 10°) \cdot \mathbf{T}(-\mathbf{W.d})$$
$$\mathbf{W}' = \mathbf{M} \cdot \mathbf{W}$$

gdje je $R_a(\mathbf{a},\theta)$=

$$\begin{bmatrix} a_x^2 + c_\theta(1-a_x^2) & a_x a_y(1-c_\theta) - a_z s_\theta & a_x a_z(1-c_\theta) + a_y s_\theta & 0 \\ a_x a_y(1-c_\theta) + a_z s_\theta & a_y^2 + c_\theta(1-a_y^2) & a_y a_z(1-c_\theta) - a_x s_\theta & 0 \\ a_x a_z(1-c_\theta) - a_y s_\theta & a_y a_z(1-c_\theta) + a_x s_\theta & a_z^2 + c_\theta(1-a_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Prostori

- Do sada smo razmatrali sljedeće prostore
  - Prostor objekta (lokalni prostor)
  - Svjetski prostor (globalni prostor)
  - Prostor kamere

Prostor kamere

- Recimo da želimo renderirati sliku stolice iz određene tačke pogleda kamere
- Stolica je postavljena u svjetskom prostoru s matricom **W**
- Kamera je postavljena u svjetskom prostoru s matricom **C**
- Sljedeća transformacija transformira vrhove iz prostora objekta stolice u svjetski prostor, a zatim iz svjetskog prostora u prostor kamere:

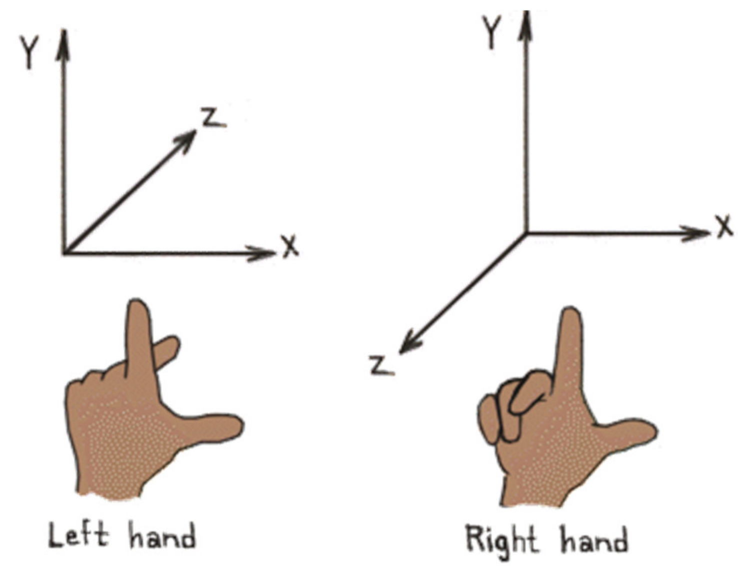
$$\mathbf{v}' = \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

- Sada kada je objekt transformiran u prostor relativno u odnosu na kameru, možemo se usredsrediti na sljedeći korak, a to je projekcija ovog 3D prostora u 2D prostor slike

Pozicija

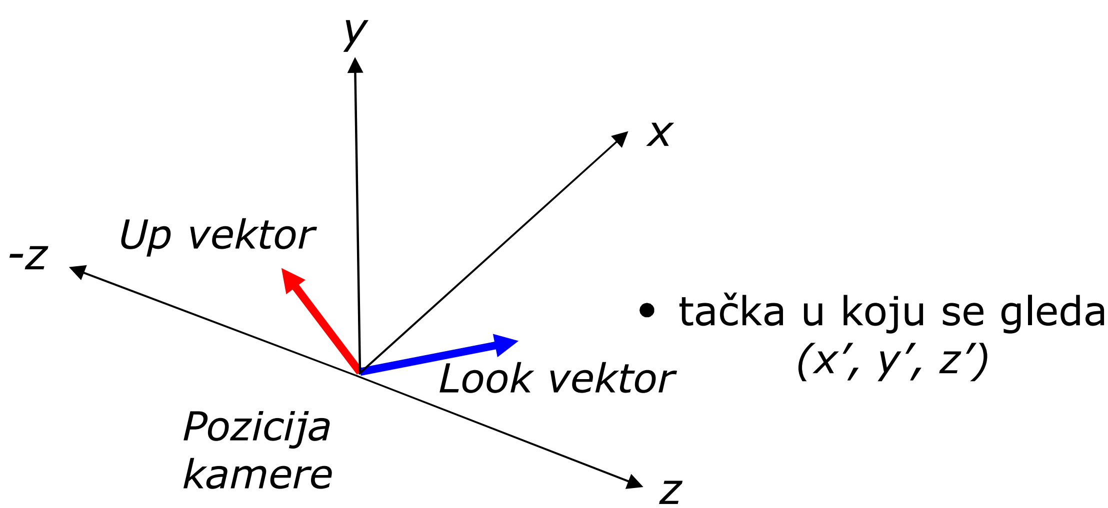
- Gdje je kamera locirana u odnosu na ishodište?
- Za našu kameru u 3D prostoru koristimo desni koordinatni sistem
  - Otvorite desnu šaku, poravnajte dlan i palac sa $+x$ osom, pokažite kažiprstom

    duž $+y$ ose, te pokažite srednjim prstom prema $+z$ osi
  - Ako gledate u ekran, z osa će biti pozitivna ako je usmjerena prema vama



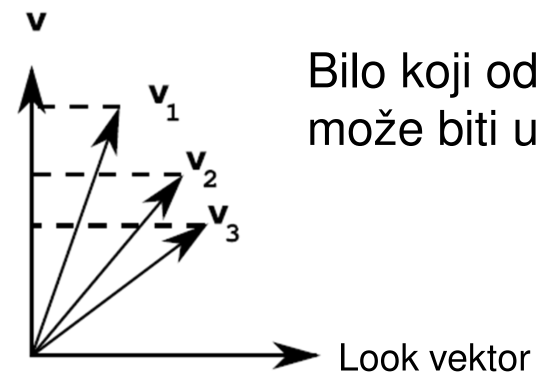
Orijentacija: Look i Up vektori 1/2

- Orijentacija se specificira pomoću tačke u 3D prostoru u koju se gleda (ili smjera u kojem se gleda) i ugla rotacije oko ovog smjera
- Ovo odgovara *look* vektoru i *up* vektoru

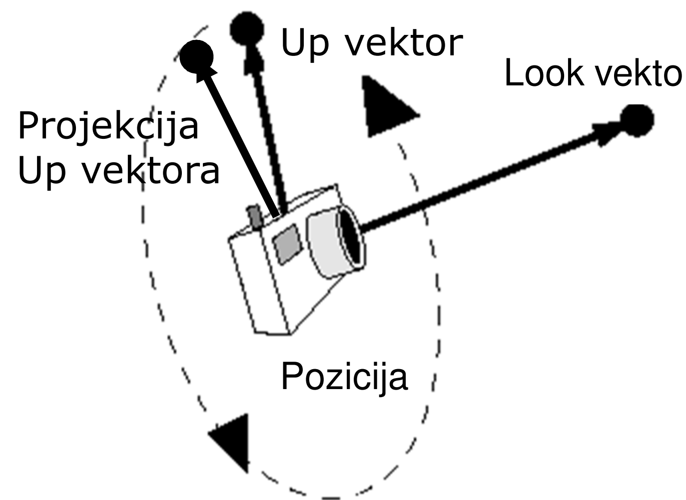


Orijentacija: Look i Up vektori 2/2

- *Look vektor*
  - Smjer u kojem kamera pokazuje
  - Tri stepena slobode; može biti bilo koji vektor u 3D prostoru
- *Up vektor*
  - određuje kako se kamera rotira oko *Look vektora*
  - naprimjer, držanje kamere horizontalno ili vertikalno
  - *Up vektor* ne smije biti paralelan sa *Look vektorom* ali ne mora biti ni ortogonalan – stvarna orijentacija će biti definirana komponentom *Up* vektora ortogonalnom na *Look* vektor, koja leži u ravni s *Look* vektorom kao normalom

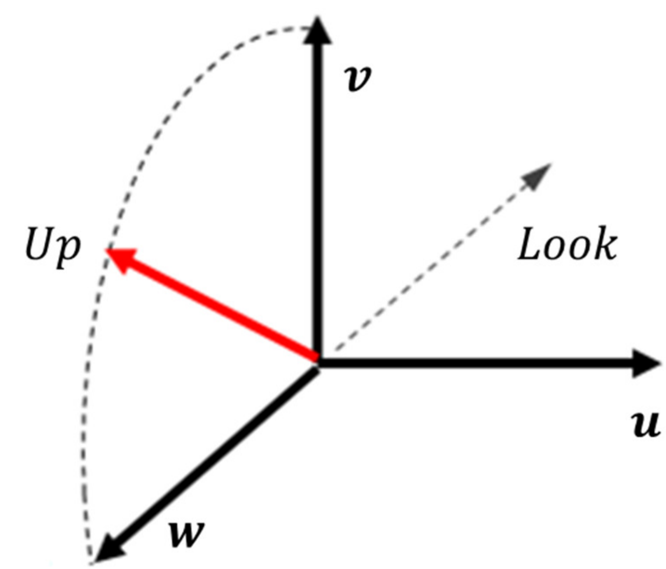


Bilo koji od $v, v1, v2, v3$ može biti up vektor



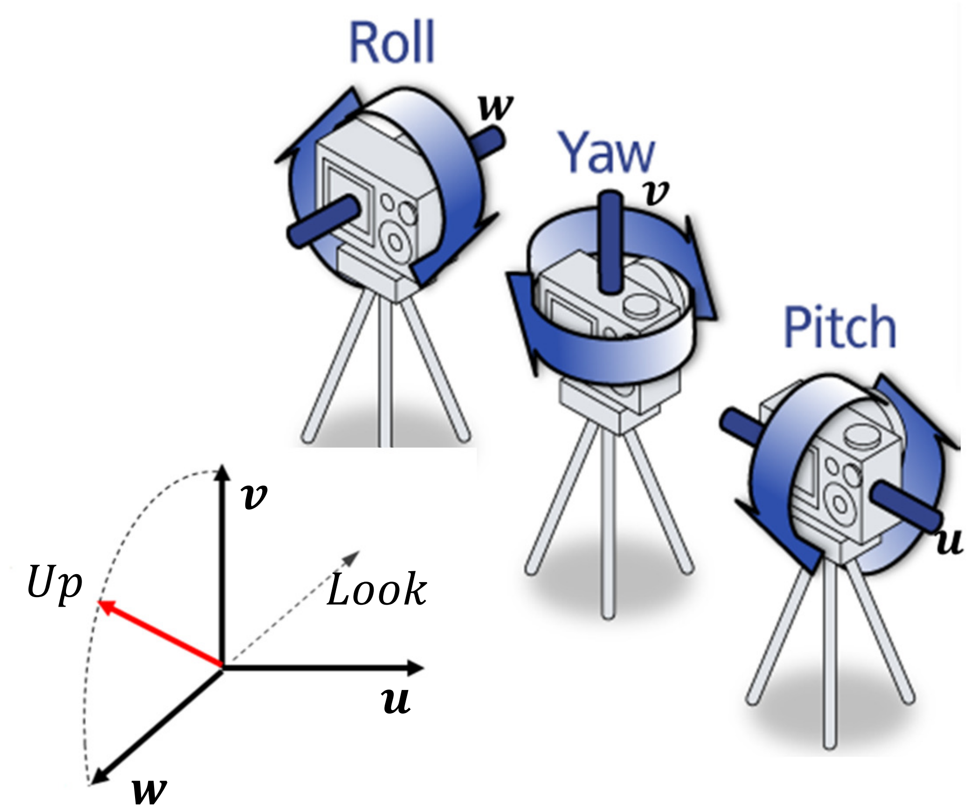
Koordinatni prostor kamere 1/2

- Ekvivalenti osa ***x***, ***y*** i ***z*** u prostoru kamere su jedinični vektori ***u, v*** i ***w*** (ovo ne treba pomiješati s homogenom koordinatom, **w**)
  - **Također je riječ o desnom koordinatnom sistemu**
  - ***w*** je jedinični vektor u suprotnom smjeru od *look* vektora
  - ***v*** je komponenta *up* vektora ortogonalna na *look* vektor, normalizirana na jediničnu dužinu
  - ***u*** je jedinični vektor ortogonalan i na **v** i na **w**



Koordinatni prostor kamere 2/2

- Postoje tri uobičajene transformacije koje koriste ose koordinatnog sistema kamere (engl. *camera space axes*)
- Roll:
  - Rotiranje kamere oko $\boldsymbol{w}$
- Yaw:
  - Rotiranje kamere oko $\boldsymbol{v}$
- Pitch:
  - Rotiranje kamere oko $\boldsymbol{u}$



- Translirati kameru u ishodište i poravnati ose sa standardnim osama, zatim upotrijebiti matrice rotacije za obavljanje ovih transformacija a potom *un-align* i *un-translate*.

Prostor slike

- Ono što nam je doista potrebno je preslikavanje iz 3D prostora u specijalni "2.5D" *prostor slike*
- Za *praktične* geometrijske svrhe, o prostoru slike se treba razmišljati kao o pravom 2D prostoru, mada svaki vrh u ovom prostoru ima i dubinu (z koordinata), i tako je ustvari, matematski uzevši, 3D prostor…
- Reći ćemo da je vidljivi dio slike u intervalu od -1 do 1 po x i y, sa 0,0 u centru slike
- z koordinata također će biti u intervalu od -1 do 1 i reprezentirat će dubinu (1 je najbliže, a -1 najdalje)
- Prostor slike ponekad se naziva i *normalizirani prostor pogleda* ili drugačije…

Projekcije pogleda

- Do sada smo naučili kako transformirati objekte iz prostora objekta u svjetski prostor te u prostor kamere
- Ono što nam je sada potrebno je nekakva vrsta transformacije koja uzima tačke u 3D prostoru kamere i transformira ih u 2.5D prostor slike
- Ovu kategoriju transformacija označavat ćemo kao *projekcije pogleda* (ili samo projekcije)
- Jednostavne *ortografske* projekcije pogleda mogu biti tretirane kao afina transformacija primijenjena nakon transformacije u prostor kamere
- Složenije *perspektivne* projekcije zahtijevaju neafinu transformaciju nakon koje slijedi dodatna operacija dijeljenja za konverziju iz 4D homogenog prostora u prostor slike
- Također, moguće je uraditi i detaljnije *nelinearne projekcije* da bi se postigli efekti sočiva u vidu ribljeg oka, itd., ali to zahtijeva značajne izmjene u procesu renderinga a ne samo u fazi "transformacija vrhova", pa se time nećemo baviti…

Ortografska projekcija

- *Ortografska* projekcija je afina transformacija i tako čuva paralelnost pravaca
- Primjer ortografske projekcije može biti pogled na kuću odozgo nadolje ili profil automobila
- Ortografska projekcija je jako korisna za određene aplikacije, ali se ne koristi za generiranje realističnih slika

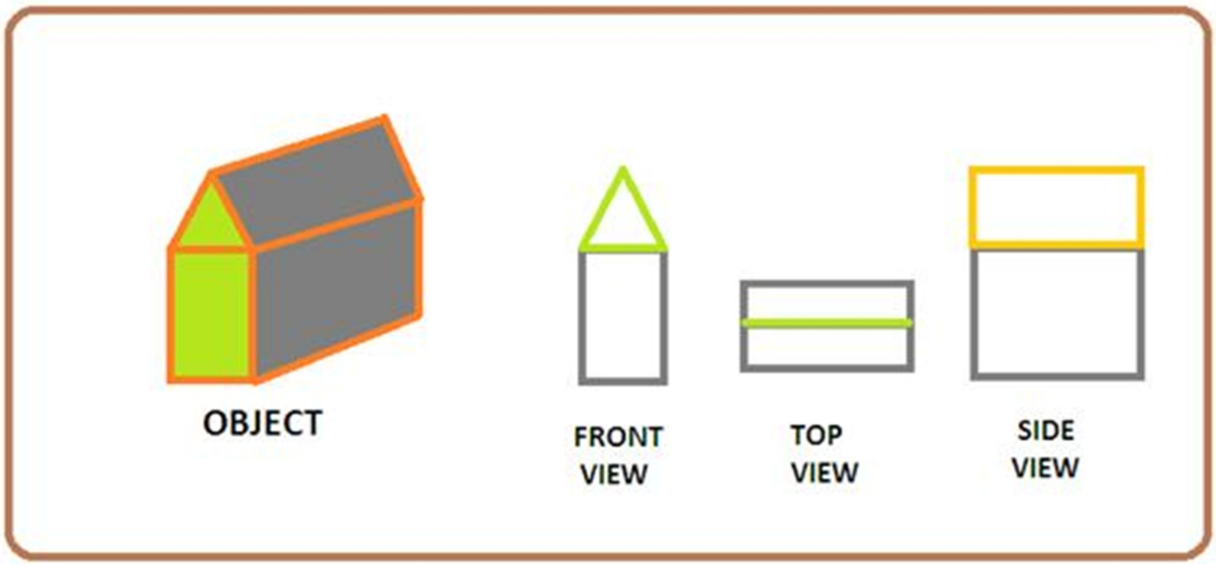
Ortografska projekcija 1/2

$$\mathbf{v}' = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{P}_{ortho}(left, right, bottom, top, near, far) =$$

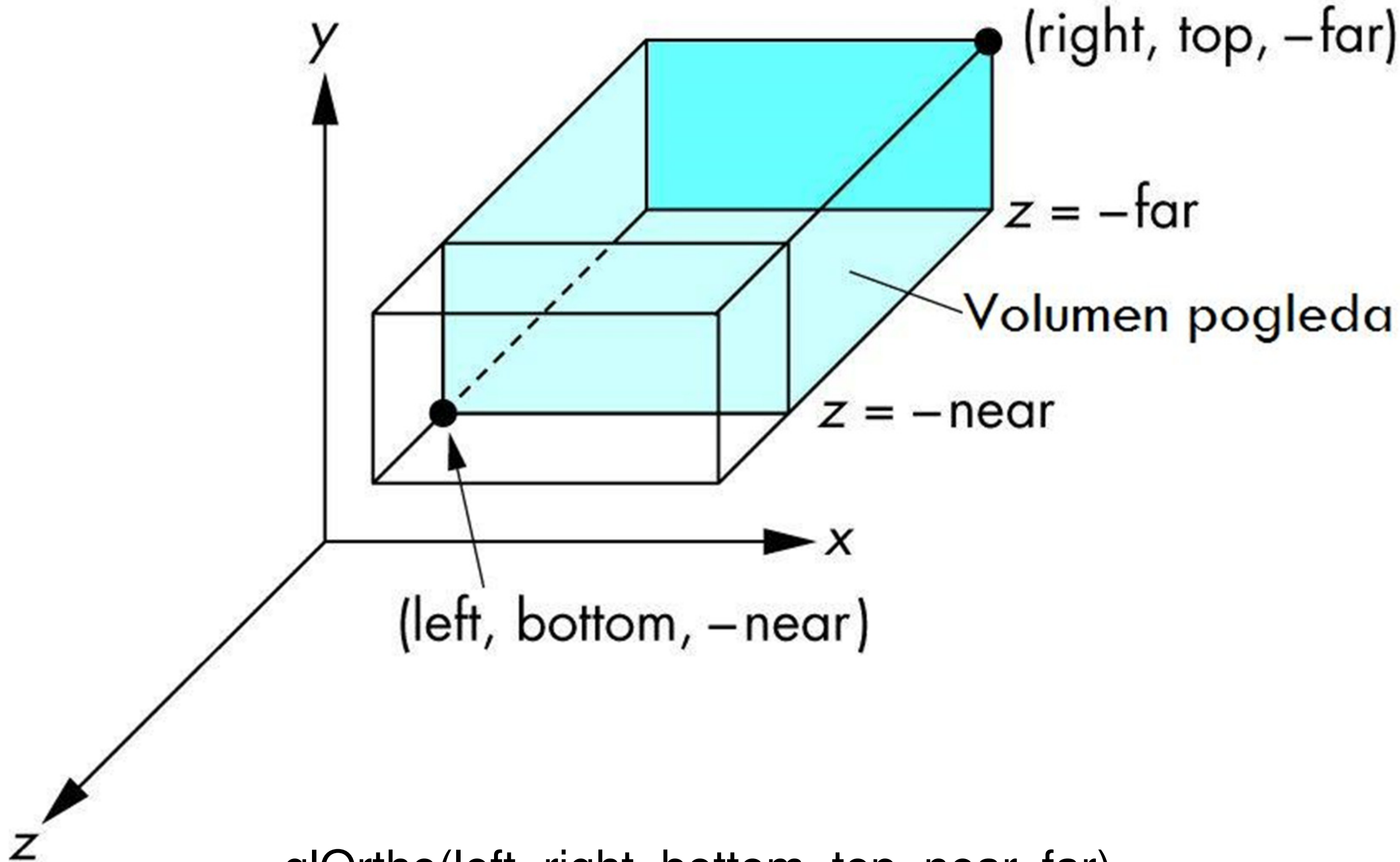
$$\begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right + left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{far - near} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ortografska projekcija 2/2



Primjer ortografske projekcije: nacrt (engl. *front view*), tlocrt (engl. *top view*), bokocrt (engl. *side view*)

Ortografska projekcija u OpenGL-u



glOrtho(left, right, bottom, top, near, far)

Perspektivna projekcija

- Kod perspektivne projekcije objekti postaju manji kako se udaljavaju od kamere, kao što je slučaj s fotografijom ili video slikom
- Cilj većine proizvođača sočiva za kamere "iz stvarnog života" je postići savršenu perspektivnu projekciju
- Realističnije računarski generirane slike koriste perspektivnu projekciju
- Fundamentalno svojstvo perspektivne projekcije je da paralelni pravci *neće* nužno ostati paralelni nakon transformacije (pravi ih neafinim)

Volumen pogleda

- Kamera definira neku vrstu 3D oblika u svjetskom prostoru koji predstavlja volumen vidljiv tom kamerom
- Naprimjer, normalna perspektivna kamera s pravougaonom slikom opisuje jednu vrstu beskonačne piramide u prostoru
- Vrh piramide je u ishodištu kamere i piramida projektuje prema van ispred kamere u prostor
- U računarskoj grafici, tipično je onemogućiti da ova piramida bude beskonačna njenim odsijecanjem na nekoj udaljenosti od kamere. Ovo se naziva *dalja ravan odsijecanja* (engl. *far clipping plane*)
- Također, odsijecanjem vrha piramide postavljamo limit na njen najbliži doseg. Ovo se naziva *bliža ravan odsijecanja* (engl. *near clipping plane*)
- Piramida s odsječenim vrhom poznata je pod imenom *frustum*. Standardni perspektivni volumen pogleda se naziva *frustum pogleda*

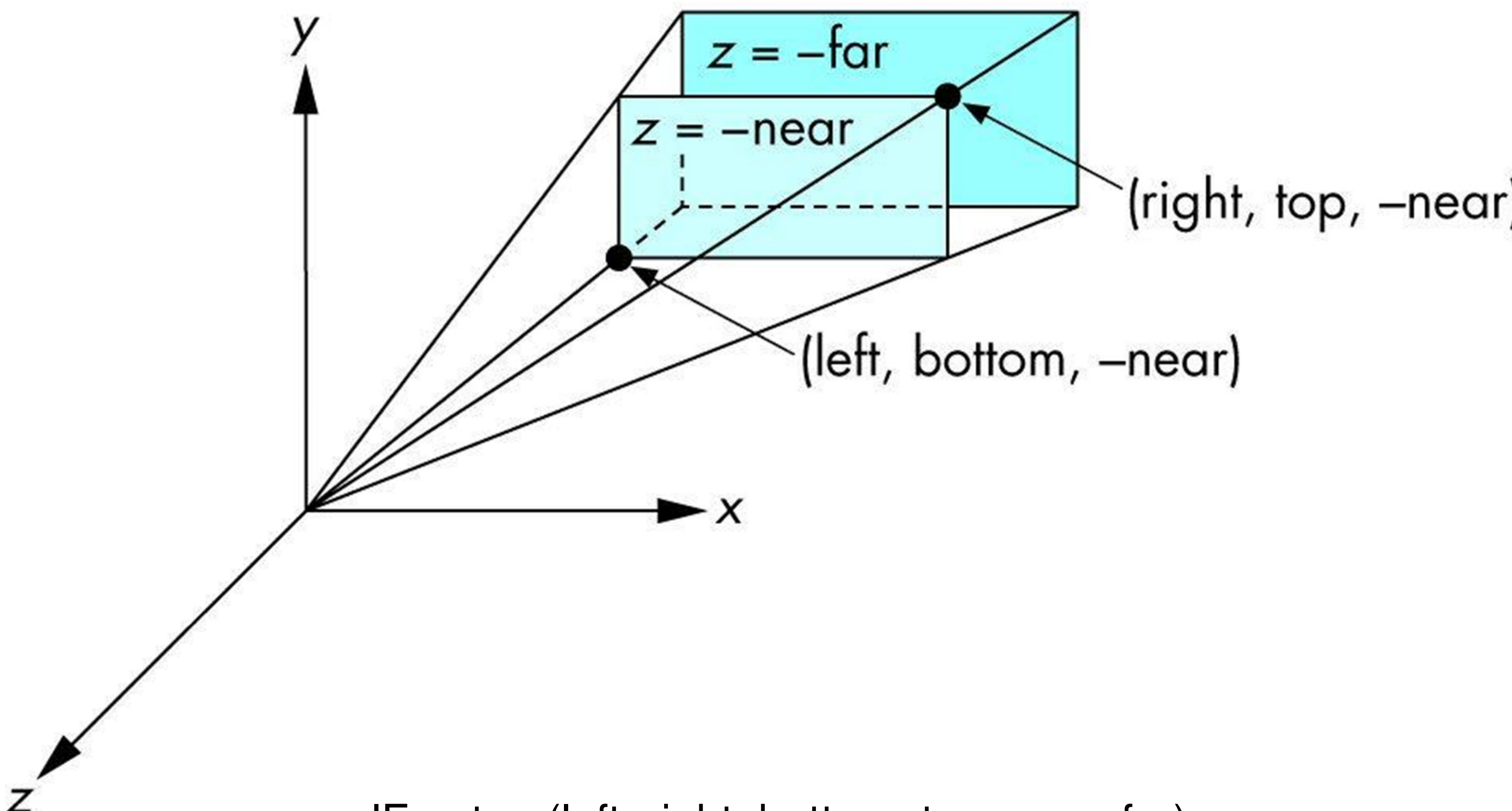
Frustum pogleda

- U određenom smislu, o ovom frustumu se može razmišljati kao o deformisanoj kocki pošto ima šest stranica, a svaka stranica ima četiri brida
- Ako o ovoj kocki razmišljamo kao da je u intervalu od -1 do 1 po svakoj dimenziji *xyz*, o perspektivnoj projekciji možemo razmišljati kao o preslikavanju iz ovog frustuma pogleda u *normalizirani prostor pogleda* ili *prostor slike*
- Potreban nam je način da ovu transformaciju matematski predstavimo

Frustum pogleda

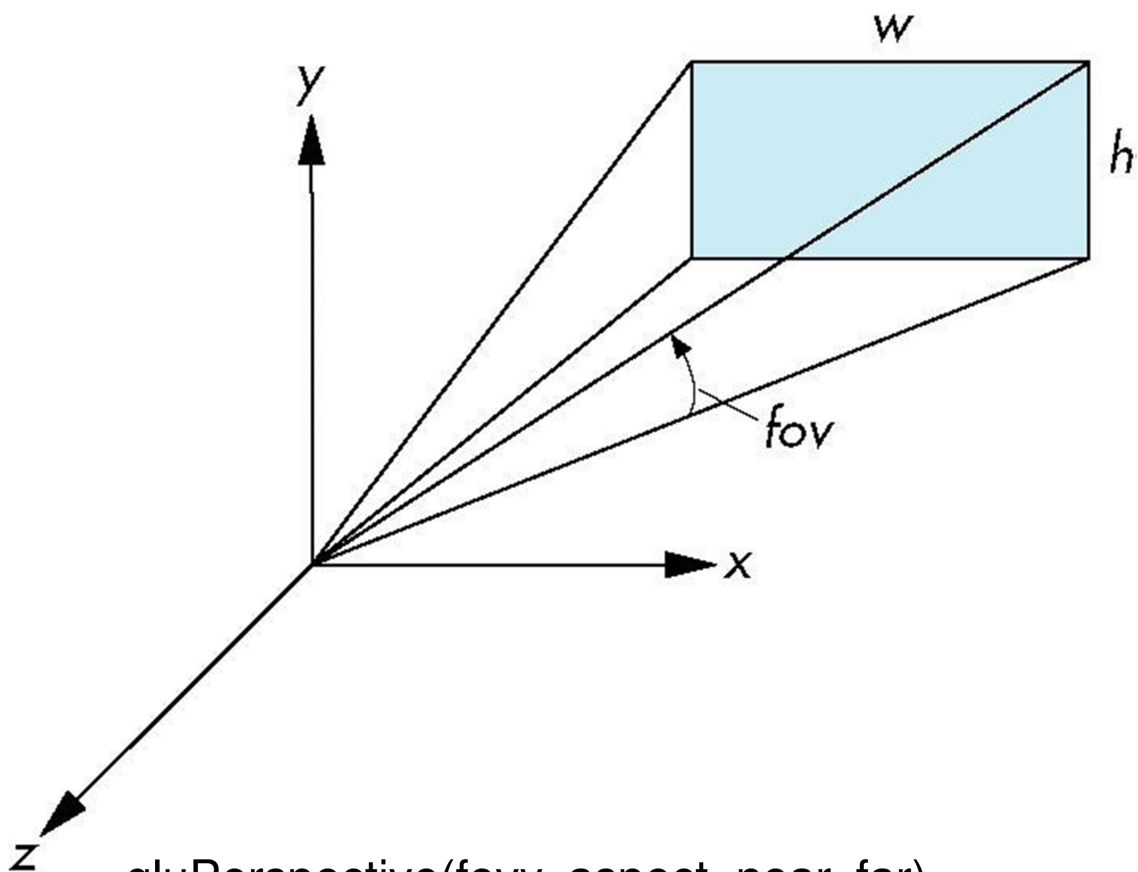
- Frustum pogleda se obično definira pomoću *vidnog polja* (engl. *Field of View – FOV*) i omjera slike (engl. *aspect ratio*), uz udaljenost bliže i dalje ravni odsijecanja
- Zavisno od konvencije, FOV može predstavljati ili horizontalno ili vertikalno vidno polje. Najčešće se uzima da je FOV čitav vertikalni ugao gledanja
- Omjer slike je x/y omjer finalne prikazane slike
- Naprimjer, kada gledamo TV koji je 24" x 18", omjer slike je 24/18 ili 4/3
- Stariji stil TV prijemnika ima omjer slike 4/3, dok moderni TV prijemnici sa širokim ekranom koriste omjer 16/9. Ove veličine su uobičajene i za računarske monitore

Frustum pogleda u OpenGL-u



glFrustum(left, right, bottom, top, near, far)

Vidno polje u OpenGL-u



gluPerspective(fovy, aspect, near, far)

Perspektivna matrica

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{P}_{persp}\left(fov, aspect, near, far\right) =$$

$$\begin{bmatrix} \dfrac{\tan\left(fov/2\right)}{aspect} & 0 & 0 & 0 \\ 0 & \dfrac{1}{\tan\left(fov/2\right)} & 0 & 0 \\ 0 & 0 & \dfrac{near+far}{near-far} & \dfrac{2 \cdot near \cdot far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Perspektivna matrica

- Ako pogledamo perspektivnu matricu, vidjet ćemo da ona nema [0 0 0 1] u zadnjem redu
- To znači da kada transformiramo 3D vektor pozicije [v_x v_y v_z 1], nećemo nužno dobiti 1 za vrijednost četvrte komponente rezultujućeg vektora
- Umjesto toga, dobićemo pravi 4D vektor [v_x' v_y' v_z' v_w']
- Zadnji korak u perspektivnoj projekciji je preslikati ovaj 4D vektor natrag u 3D w=1 subprostor:

$$\begin{bmatrix} v_x & v_y & v_z & v_w \end{bmatrix} \Rightarrow \begin{bmatrix} \dfrac{v_x}{v_w} & \dfrac{v_y}{v_w} & \dfrac{v_z}{v_w} \end{bmatrix}$$

Perspektivne transformacije

- Važno je osigurati da se pravci u 3D prostoru preslikaju na pravce u prostoru slike
- Ovo je jednostavno za x i y, ali su zbog dijeljenja moguće nelinearnosti po z
- Perspektivna transformacija predstavljena na prethodnom slajdu osigurava preslikavanje pravaca, ali su moguće i druge perspektivne transformacije za koje to ne vrijedi

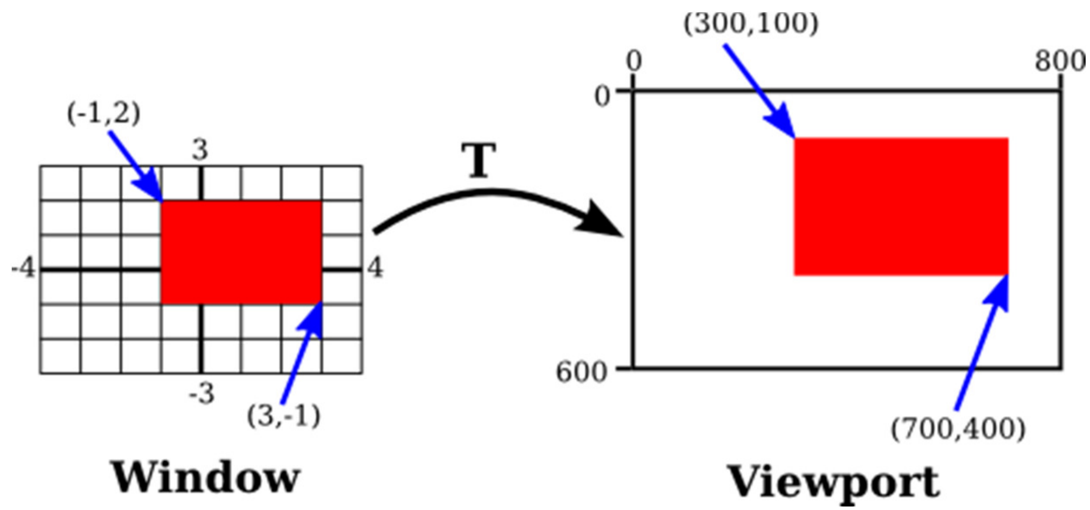
Viewportovi 1/2

- Finalna transformacija uzima tačke iz -1…1 prostora slike i preslikava ih u stvarni pravougaonik od piksela (koordinate uređaja)
- Finalno preslikavanje "na uređaj" (engl. *device mapping*) iz normaliziranog prostora pogleda u stvarni pravougaoni viewport može se definirati kao:

$$\mathbf{D}(x_0, x_1, y_0, y_1) = \begin{bmatrix} (x_1 - x_0)/2 & 0 & 0 & x_0 + (x_1 - x_0)/2 \\ 0 & (y_1 - y_0)/2 & 0 & y_0 + (y_1 - y_0)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Vrijednost dubine se obično preslikava na 32-bitnu vrijednost s nepokretnim zarezom u intervalu od 0 (blizu) do 0xffffffff (daleko)

Viewportovi 2/2



- Pravougaono područje se naziva prozor (engl. *window*) ili prozor pogleda. Transformacija koordinata se koristi za preslikavanje prozora u viewport.
- T predstavlja transformaciju koordinata. T je funkcija koja uzima svjetske koordinate (x,y) u nekom prozoru i preslikava ih u koordinate piksela T(x,y) u viewportu.

Prostori

- Prostor objekta (3D)
- Svjetski prostor (3D)
- Prostor kamere (3D)
- Nenormalizirani prostor pogleda (4D)
- Prostor slike (2.5D)
- Prostor uređaja (2.5D)

Rendering trougla

- Glavne faze u *tradicionalnom grafičkom cjevovodu* su:
  - Transformacija
  - Osvjetljavanje
  - Odsijecanje / Culling
  - Scan konverzija
  - Rendering piksela

Transformacija

- U transformacijskoj fazi, vrhovi se transformiraju iz svog prvobitnog definiranog prostora objekta preko niza koraka u finalni "2.5D" prostor uređaja od stvarnih piksela

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{v}'' = \begin{bmatrix} \frac{v'_x}{v'_w} & \frac{v'_y}{v'_w} & \frac{v'_z}{v'_w} \end{bmatrix}$$

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$

Transformacija: Korak 1

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

- **v**: Prvobitni vrh u prostoru objekta
- **W**: Matrica koja transformira objekt u svjetski prostor
- **C**: Matrica koja transformira kameru u svjetski prostor (C^{-1} će transformirati iz svjetskog prostora u prostor kamere)
- **P**: Neafina matrica perspektivne projekcije
- **v'**: Transformirani vrh u 4D nenormaliziranom prostoru pogleda

- Napomena: Ponekad se ovaj korak dijeli u dva (ili više) koraka. To se često radi da bi se proračuni osvjetljenja i odsijecanja obavili u prostoru kamere (prije primjene neafine transformacije)

Transformacija: Korak 2

$$\mathbf{v}'' = \begin{bmatrix} \dfrac{v'_x}{v'_w} & \dfrac{v'_y}{v'_w} & \dfrac{v'_z}{v'_w} \end{bmatrix}$$

- U sljedećem koraku se tačke iz 4D prostora preslikavaju u normalizirani prostor pogleda, pod imenom *prostor slike*, koji se kreće u intervalu od -1 do 1 po x, y i z
- Od sada pa nadalje, o tački ćemo uglavnom razmišljati kao da je 2D (x i y), s dodatnim informacijama o dubini (z). Ovo se ponekad naziva 2.5D

Transformacija: Korak 3

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$

- U finalnom koraku transformacijske faze, vrhovi se transformiraju iz normaliziranog -1…1 prostora slike i preslikavaju u stvarni pravougaoni *viewport* od piksela

Transformacija

$$\mathbf{v}'_{(4D)} = \mathbf{P} \cdot \mathbf{C}^{-1} \cdot \mathbf{W} \cdot \mathbf{v}$$

$$\mathbf{v}'' = \begin{bmatrix} \dfrac{v'_x}{v'_w} & \dfrac{v'_y}{v'_w} & \dfrac{v'_z}{v'_w} \end{bmatrix}$$

$$\mathbf{v}''' = \mathbf{D} \cdot \mathbf{v}''$$

Matrice u GL-u

- GL ima nekoliko ugrađenih rutina koje pokrivaju gotovo sve matrične operacije koje smo do sada obradili
- GL omogućuje korisniku da specificira transformaciju i projekciju koje će se primijeniti na sve vrhove koji mu se proslijede
- Inače, GL primjenjuje 4-koračni proces transformacije:
  - Prostor objekta u prostor kamere (model-pogled)
  - Projekcija u 4D prostor
  - Dijeljenje s "w" (projekcija u prostor slike)
  - Preslikavanje u koordinate uređaja
- Iako se prva dva koraka mogu iskombinirati u jedan korak, oni često ostaju razdvojeni kako bi se omogućilo da se proračuni odsijecanja i osvjetljenja obave u 3D prostoru kamere

glMatrixMode()

- Naredba glMatrixMode() omogućava korisniku da specificira koju matricu želi postaviti
- U ovom trenutku postoje dvije različite opcije koje nas zanimaju:
  - glMatrixMode (GL_MODELVIEW)
  - glMatrixMode (GL_PROJECTION)
- Matrica *model-pogled* (engl. *model-view matrix*) u GL-u predstavlja $\mathbf{C}^{-1}\cdot\mathbf{W}$ transformaciju, a *matrica projekcije* je matrica **P**

$$\mathbf{v}' = \mathbf{P}\cdot\mathbf{C}^{-1}\cdot\mathbf{W}\cdot\mathbf{v}$$

glLoadMatrix()

- Najdirektniji način da se postavi trenutna matrična transformacija je upotrijebiti glLoadMatrix() i proslijediti niz od 16 brojeva koji grade matricu
- Alternativno, postoji naredba glLoadIdentity() za brzo resetovanje trenutne matrice u jediničnu matricu

glRotate(), glTranslate(), glScale()

- Postoji nekoliko temeljnih matričnih operacija koje vrše rotacije, translacije i skaliranja
- Ove operacije uzimaju trenutnu matricu i modificiraju je primjenjujući novu transformaciju *prije* trenutne, efektivno pridodajući novu transformaciju s desne strane trenutne matrice
- Naprimjer, recimo da se trenutna matrica model-pogled zove **M**. Ukoliko pozovemo glRotate(), nova vrijednost za **M** će biti **M·R**

glPushMatrix(), glPopMatrix()

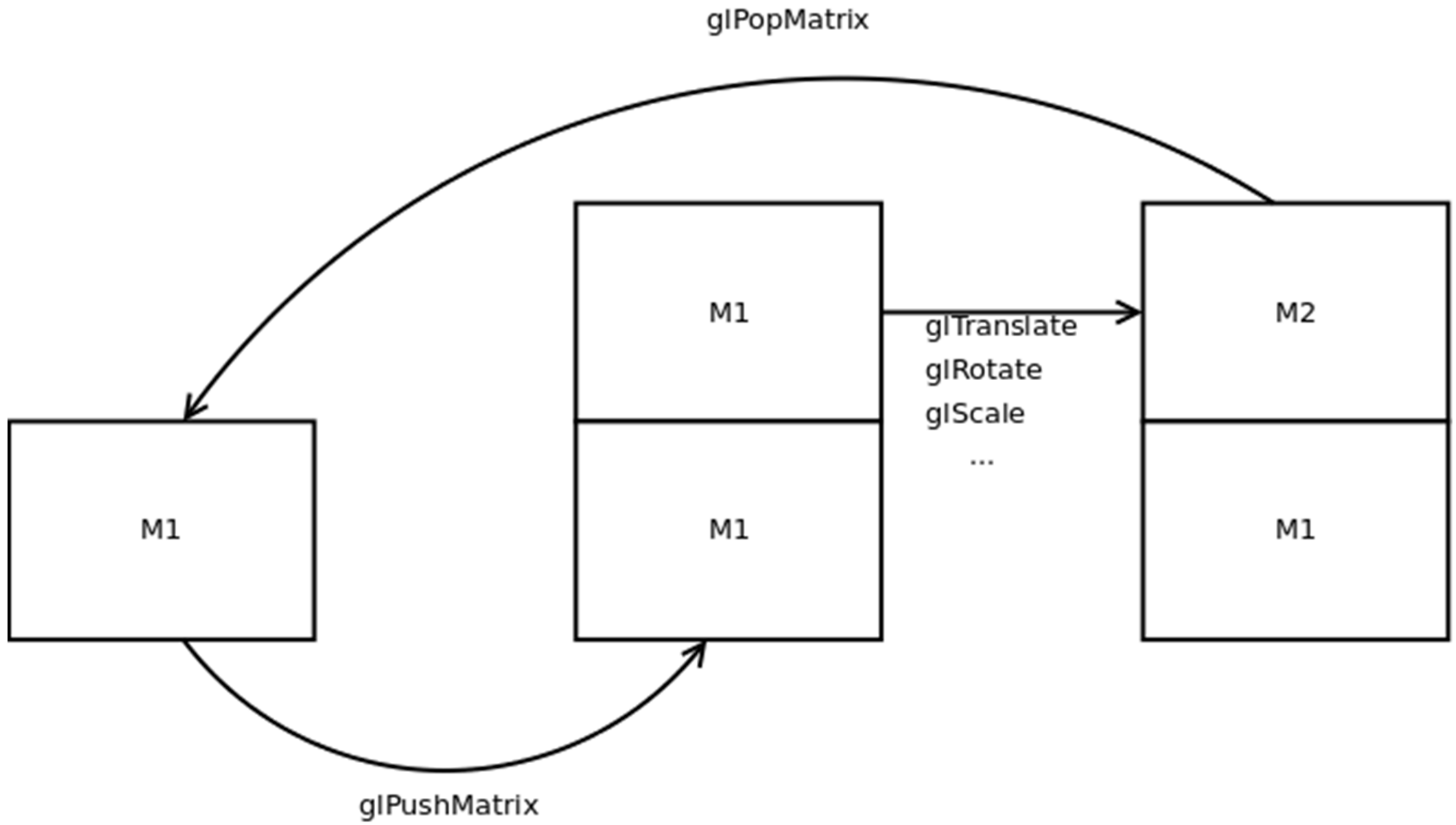
- GL koristi vrlo koristan koncept *matričnog stacka*
- U bilo kojem trenutku postoji "trenutna" matrica
- Ukoliko se pozove glPushMatrix(), stack pointer se povećava, a trenutna matrica se kopira na vrh stacka
- Bilo koja matrična rutina (glRotate…) djelovat će samo na matricu na vrhu stacka
- Zatim se može pozvati glPopMatrix() kako bi se matrica vratila u stanje u kojem je bila prije pozivanja glPushMatrix()

glPushMatrix(), glPopMatrix()
Primjer 1/2

- Naprimjer, pretpostavimo da želite nacrtati automobil...
  - Postavili ste matricu za crtanje tijela automobila, nazovimo je M_1
  - Sada želite nacrtati jedan točak...
    - Možete izračunati M_2 – matricu potrebnu za tačan prikaz točka – ili, budući da je točak relativan u odnosu na tijelo automobila (dakle, postoji matrica M_3 takva da je $M_2 = M_1 * M_3$), vi modificirate M_1.
    - No, automobil ima 4 točka, pa morate sačuvati kopiju M_1
    - Možete to uraditi pozivom glPushMatrix(), te dobiti natrag kopiju pozivom glPopMatrix()

glPushMatrix(), glPopMatrix()

Primjer 2/2

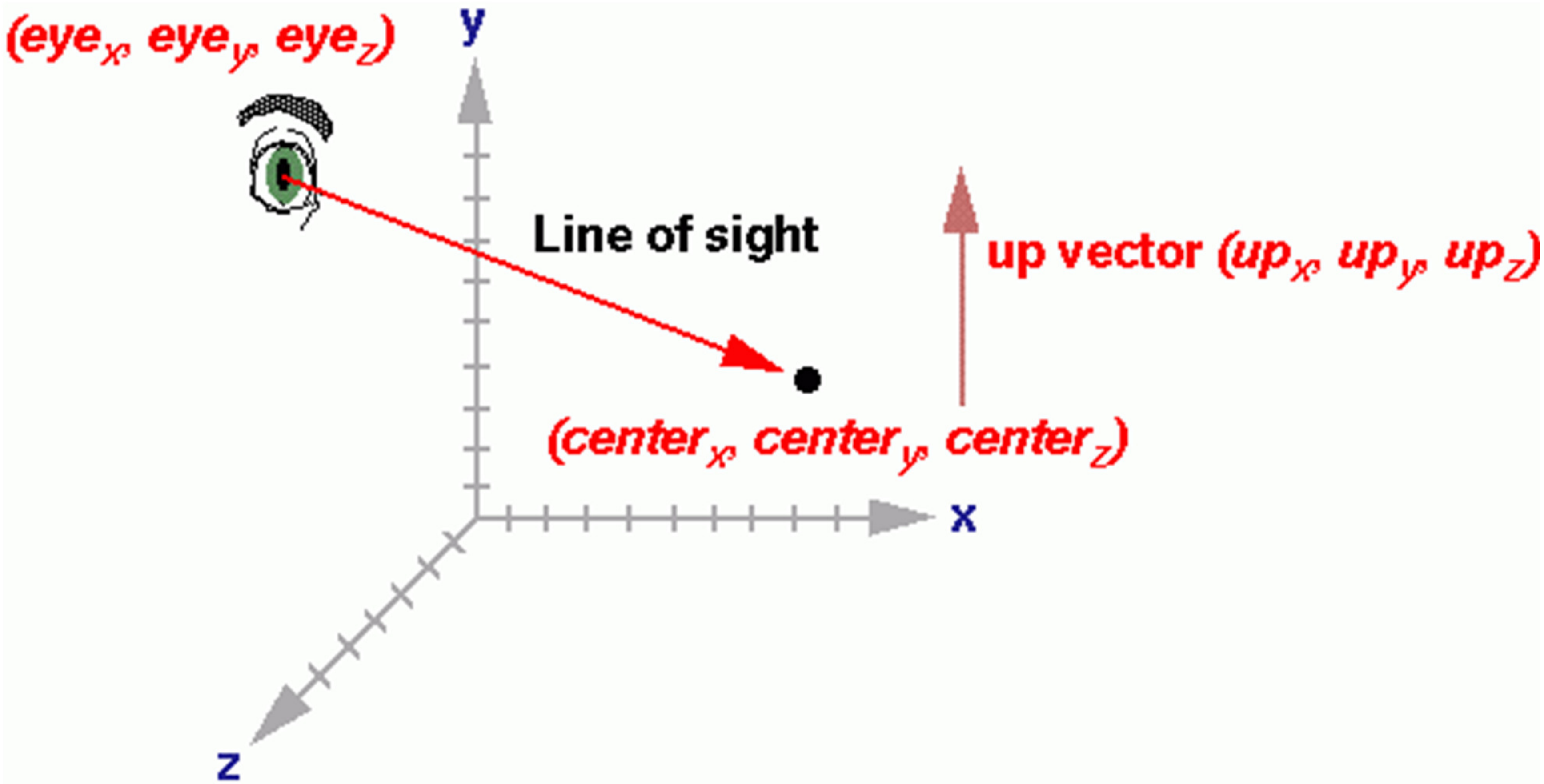


glFrustum(), gluPerspective()

- GL pruža nekoliko funkcija za specificiranje transformacije pogleda
- glFrustum() omogućuje korisniku da definira perspektivni volumen pogleda baziran na stvarnim koordinatama frustuma
- gluPerspective() pruža intuitivniji način za postavljanje perspektivne transformacije specificiranjem FOV-a, omjera slike, udaljenosti bliže i dalje ravni odsijecanja
- glOrtho() omogućuje specificiranje ortografske transformacije pogleda

gluLookAt()

- gluLookAt() implementira funkciju "pogledaj u" o kojoj smo govorili na prethodnom predavanju
- Ova funkcija dozvoljava da se specificira očište (engl. *eye point*) kamere, kao i ciljna tačka u koju se gleda
- Također, dozvoljava i da se definira "up" vektor, za kojeg smo ranije samo pretpostavili da će biti [0 1 0] (y osa)



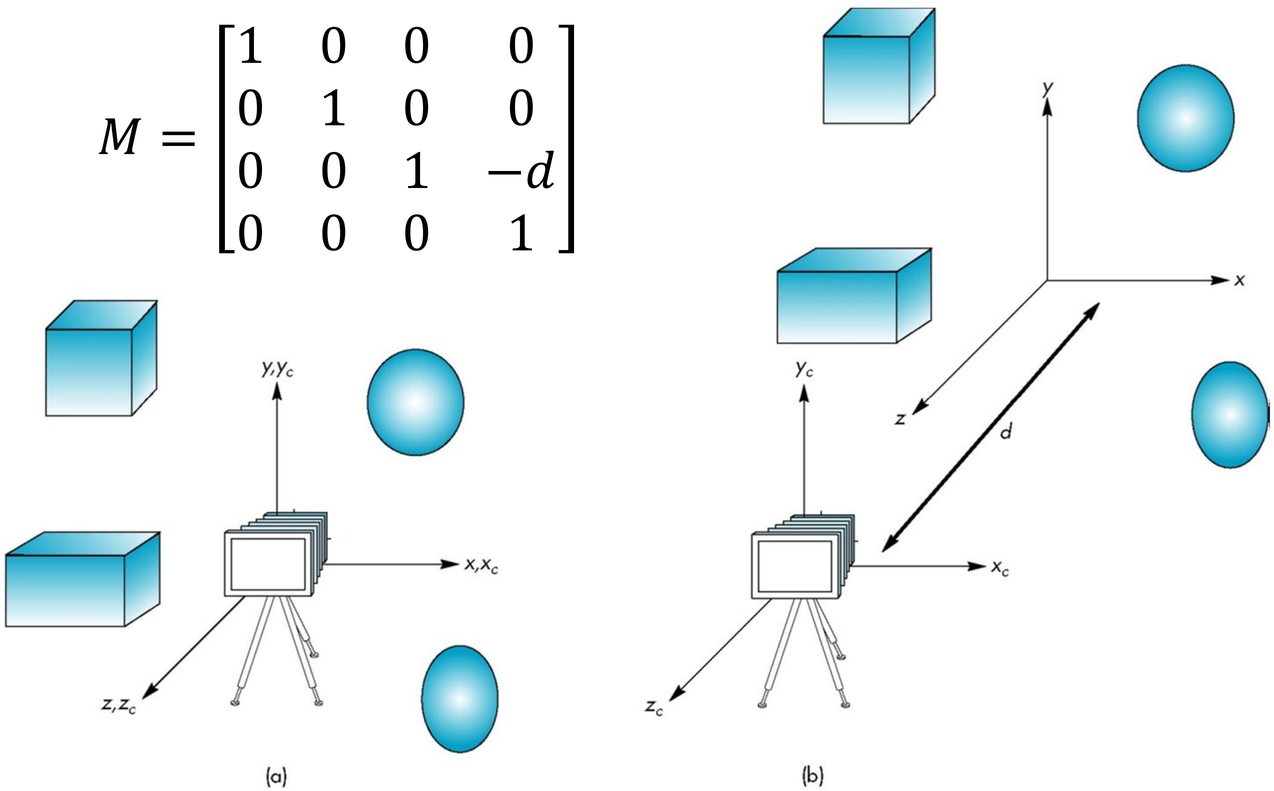
gluLookAt()

void gluLookAt (GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ,
GLdouble centerX, GLdouble centerY, GLdouble centerZ,
GLdouble upX, GLdouble upY, GLdouble upZ);

Pomjeranje kamere

- Ako su objekti na obje strane $z = 0$, moramo pomjeriti koordinatni okvir kamere

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Primjer s GL matricom

```
// Cisti ekran
glClear(GL_COLOR_BUFFER_BIT,GL_DEPTH_BUFFER_BIT);

// Definiranje projekcije
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fov,aspect,nearclip,farclip);

// Definiranje pogleda kamere
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(eye.x,eye.y,eye.z,target.x,target.y,target.z,0,1,0);

// Crta sve objekte
for(each object) {
        glPushMatrix();
        glTranslatef(pos[i].x,pos[i].y,pos[i].z);
        glRotatef(axis[i].x,axis[i].y,axis[i].z,angle[i]);
        Model[i]->Draw();
        glPopMatrix();
}

// Finish
glFlush();
glSwapBuffers();
```

Instance

- Uobičajeno je u računarskoj grafici renderirati nekoliko kopija nekakvog jednostavnijeg oblika kako bi se izgradio složeniji oblik čitave scene
- Naprimjer, mogli bismo renderirati 100 kopija iste stolice postavljene na različitim pozicijama, s različitim matricama
- Ovo označavamo kao *instanciranje* objekta, a jedna stolica u gornjem primjeru označava se kao *instanca* modela stolice