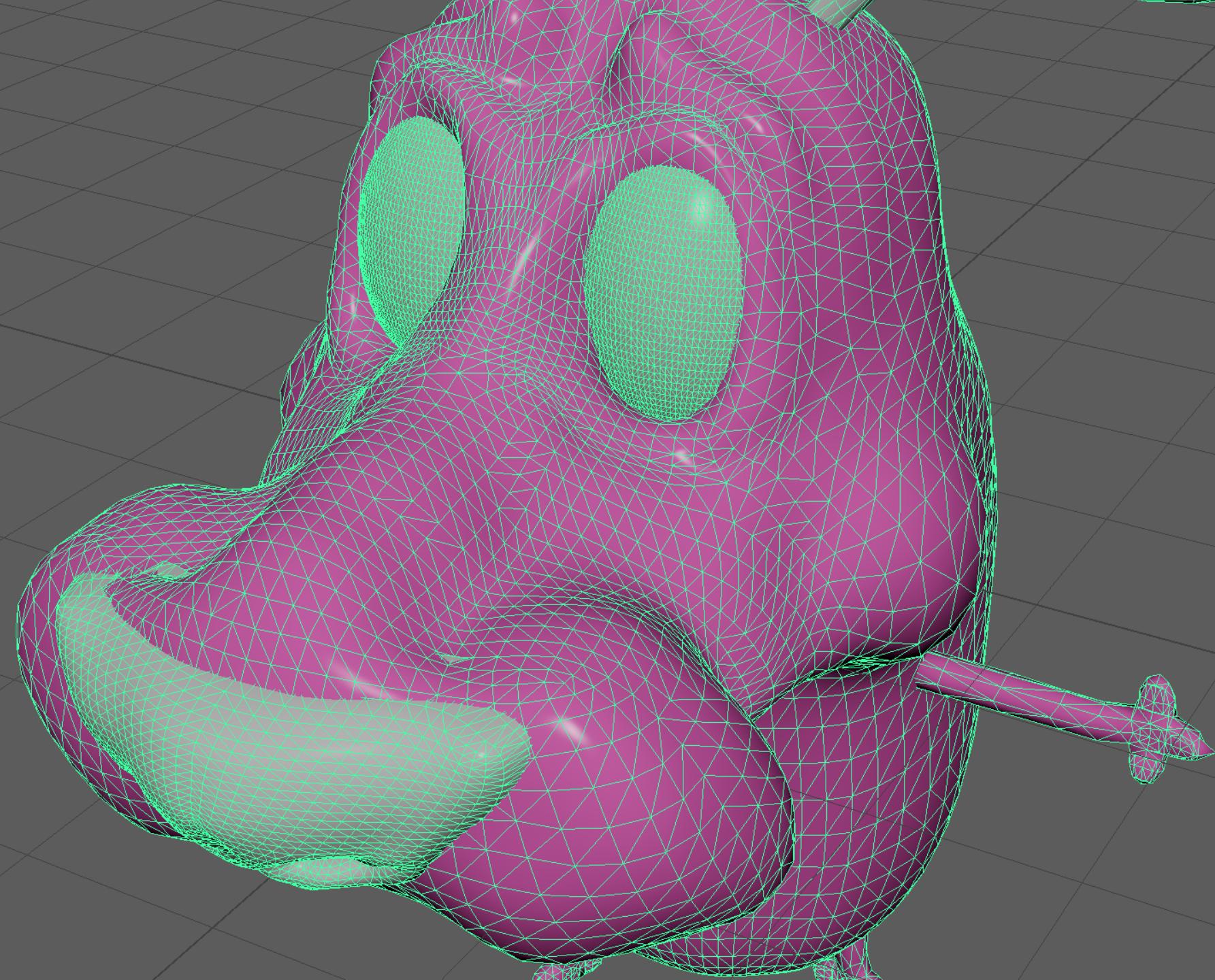
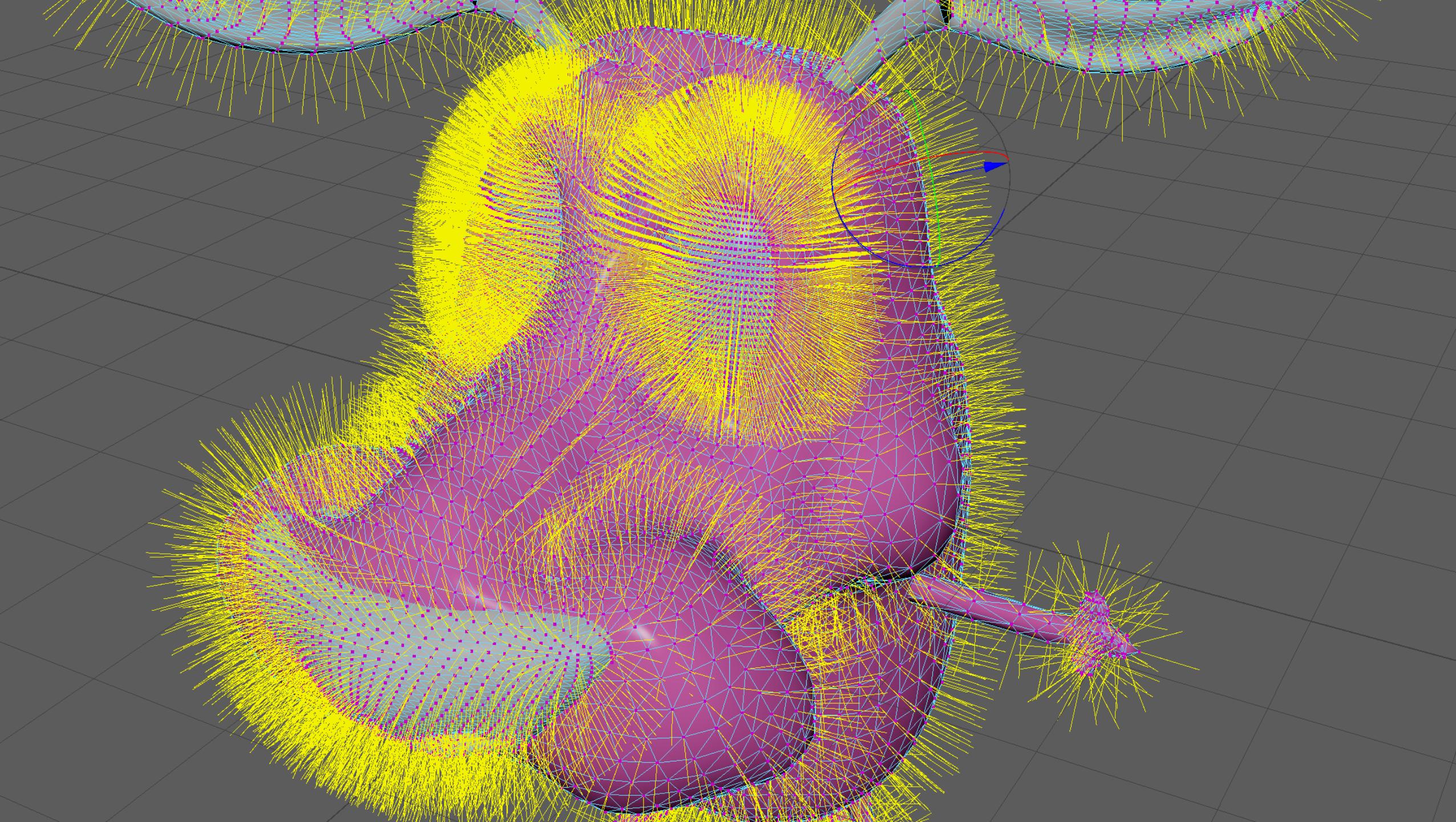


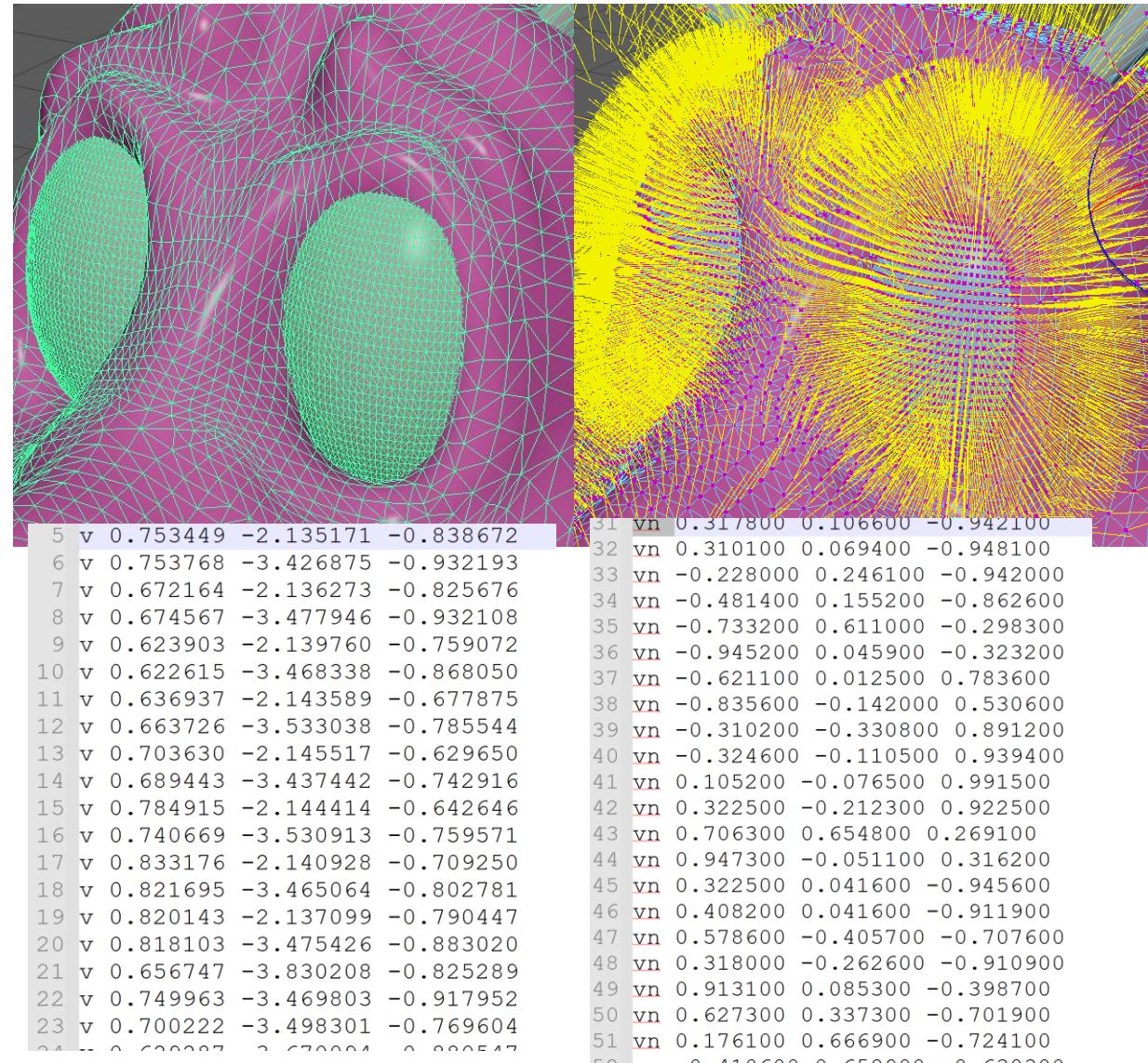


TRANSFORMACIJE IN KOORDINATNI SISTEMI



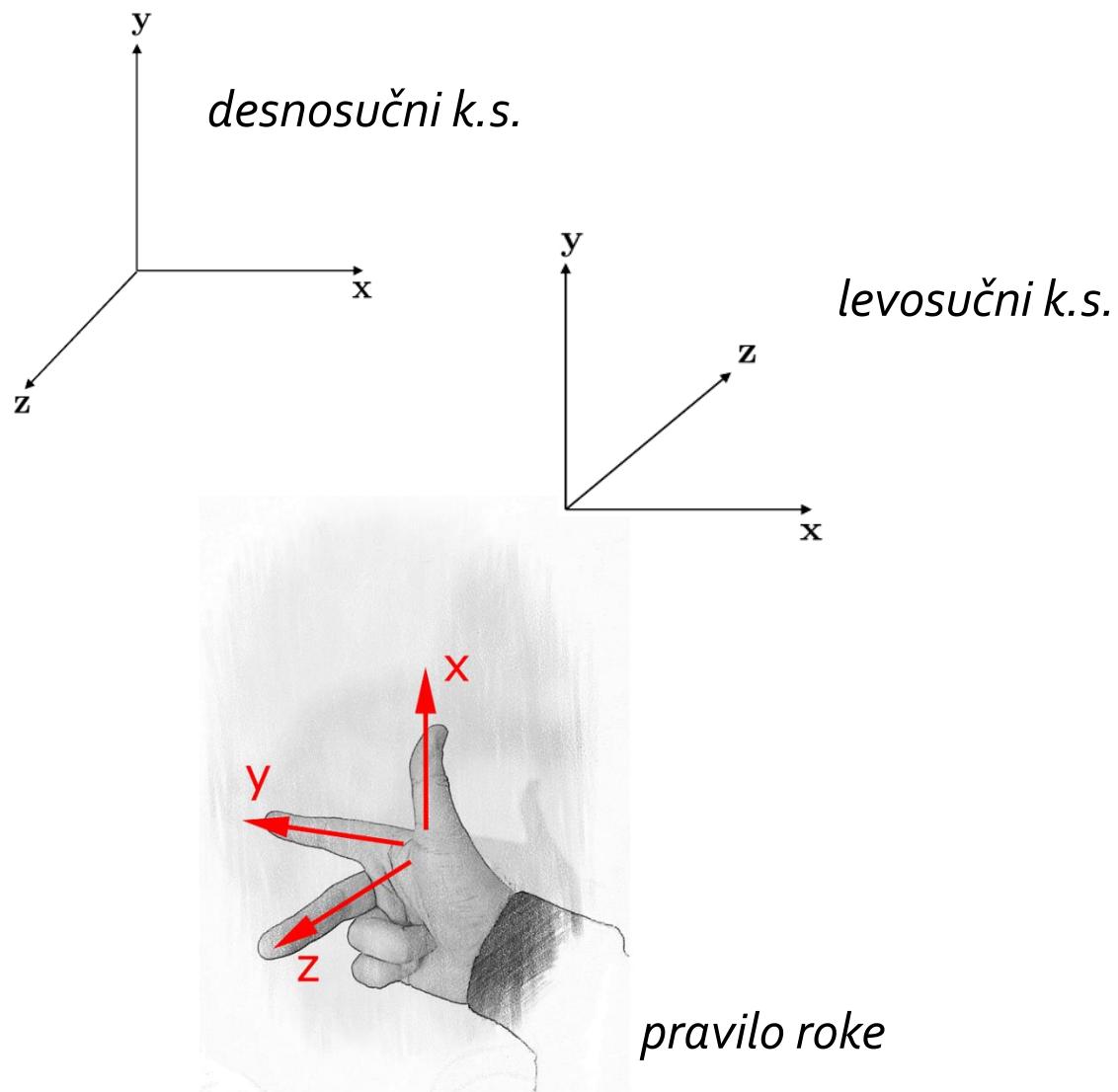


- V interaktivni grafiki sceno predstavimo s trikotniki
 - torej z naborom **točk** in povezav med njimi
 - uporabljamo tudi **normale – vektorji**, ki so pravokotni na trikotnike
- **Točka**
 - določa položaj v prostoru
 - točke obstajajo le v koordinatnih sistemih
- **Vektor**
 - ima velikost in smer v nekem vektorskem prostoru
 - nima položaja v prostoru



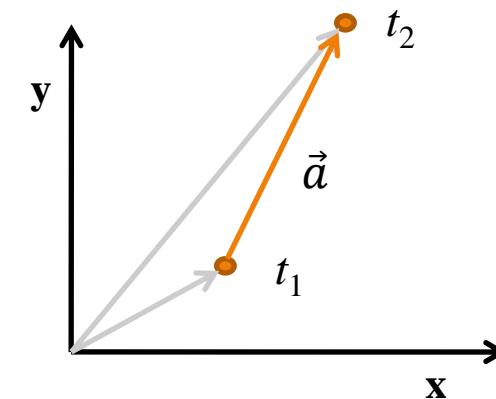
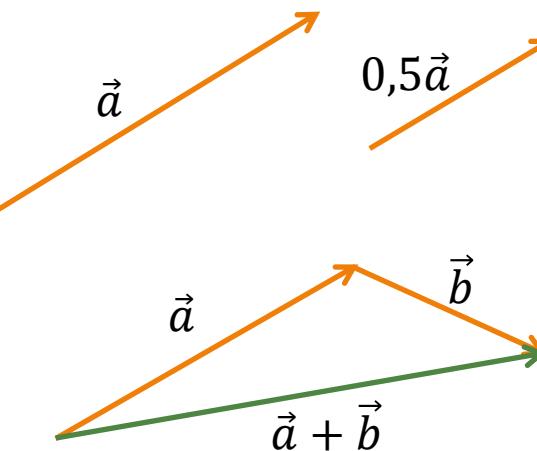
Koordinatni sistem v 3D

- Evklidski koordinatni sistem
 - bazni vektorji imajo **dolžino 1** (enotski vektorji)
 - bazni vektorji so **pravokotni** drug na drugega – ortonormirana baza
- V R^3 imamo dve možnosti za postavitev baznih vektorjev – **levosučni in desnosučni k.s.**
 - različni sistemi uporabljajo različne variante – OpenGL je privzeto desnosučni, DirectX levosučni



Računanje s točkami in vektorji

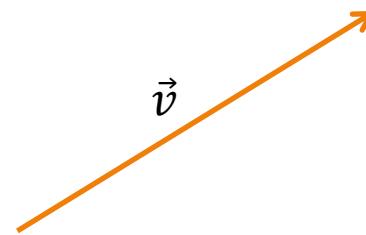
- Množenje vektorja s skalarjem ga podaljša (skrajša)
 - $c\vec{a} = \vec{b}$
- Vektor + vektor je vektor
 - $\vec{a} + \vec{b} = \vec{c}$
- Točka plus vektor je premaknjena točka
 - $t_1 + \vec{a} = t_2$
- Razlika dveh točk je vektor
 - $t_2 - t_1 = \vec{v}$
- Točka + točka – nima smisla



Velikost vektorjev

- **Velikost** (druga norma) vektorja v 3D:

- $|\vec{v}| = \sqrt{x^2 + y^2 + z^2}$



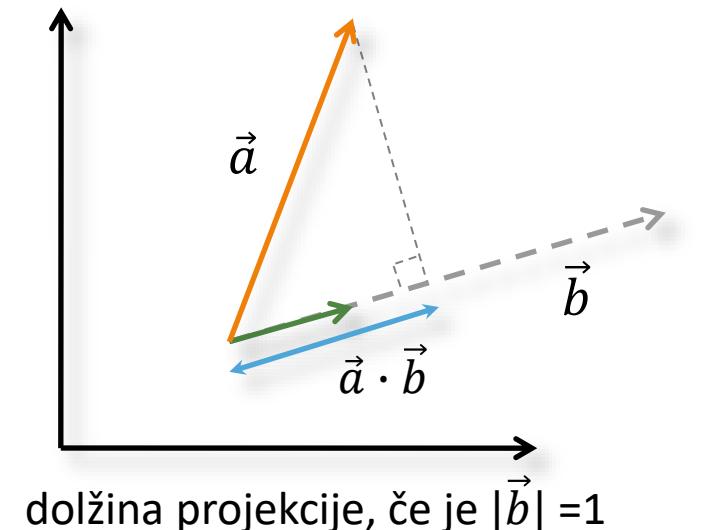
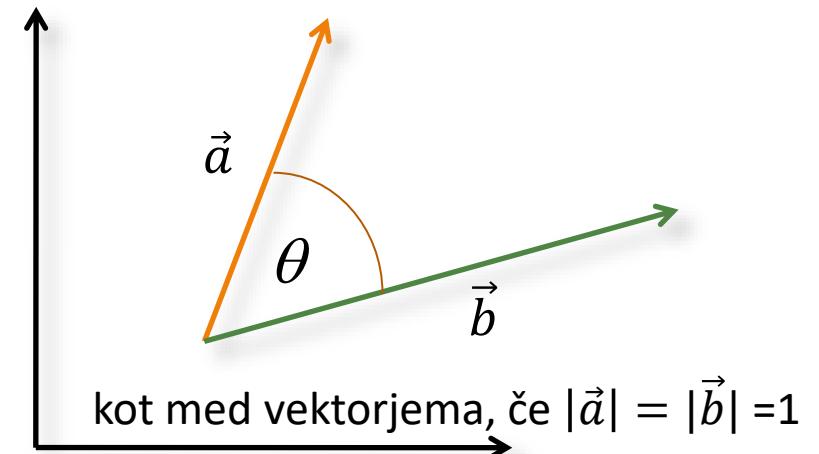
- **Enotski vektor** je vektor velikosti 1

- Če želimo dobiti enotski vektor,
ga **normiramo**

- $\vec{v}_n = \frac{\vec{v}}{|\vec{v}|}$

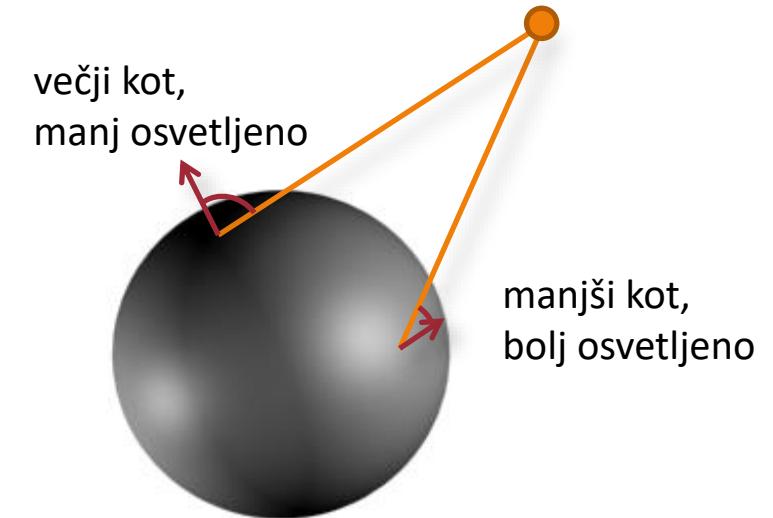
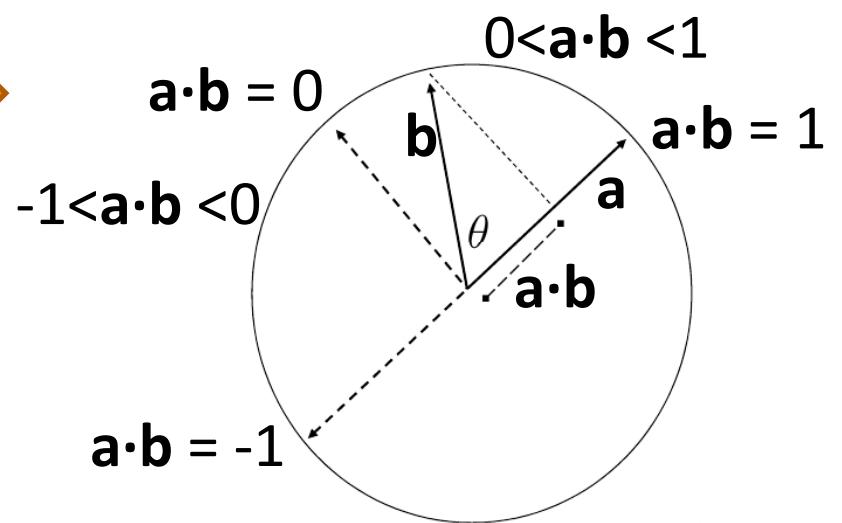
Skalarni produkt

- Vrednost (skalar) med dvema vektorjema, ki pove:
 - kot med vektorjema
 - dolžino projekcije na enotski vektor
- Je neodvisen od koord. sistema
- Izračun:
 - $\vec{a} \cdot \vec{b} = \sum_i a_i b_i$
 - v 3D: $\vec{a} \cdot \vec{b} = a_x b_x + a_y b_y + a_z b_z$
- Pomen:
 - $\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos\theta$



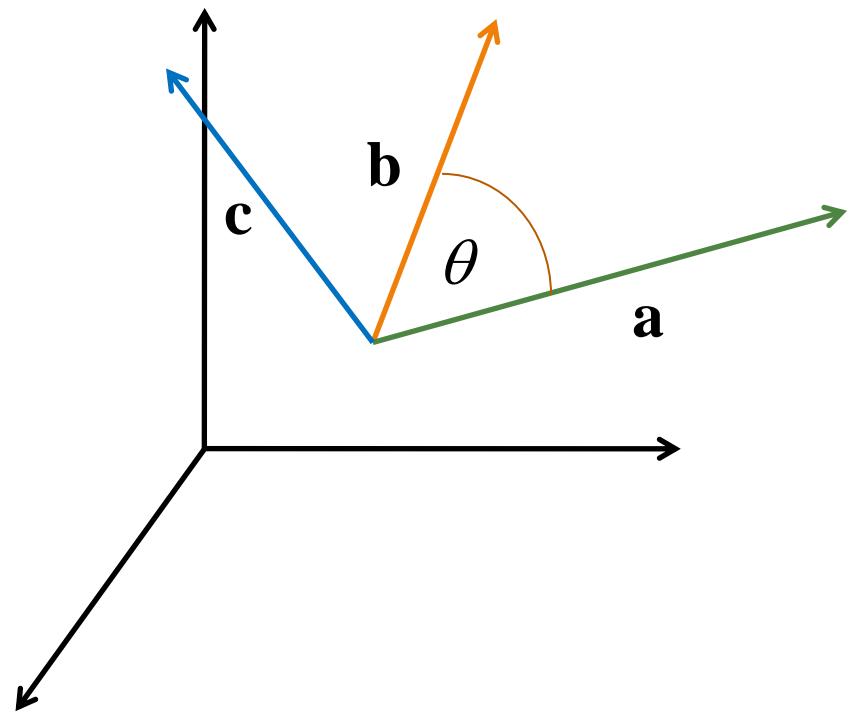
Skalarni produkt

- Vrednost in predznak skalarnega produkta dveh vektorjev dolžine 1
- Skalarni produkt ima veliko uporab pri RG
 - npr. pri računanju osvetlitve nas zanima kot vpadne svetlobe
 - pri detekciji trkov nas zanima v katero smer se predmet premika relativno na drug predmet, pod kakšnim kotom se zaletita
 - razdalja od točke do ravnine
 - katera stran poligona gleda proti kamери
 - pretvorbe med koordinatnimi sistemi
 - ...
- Demo



Vektorski (križni) produkt

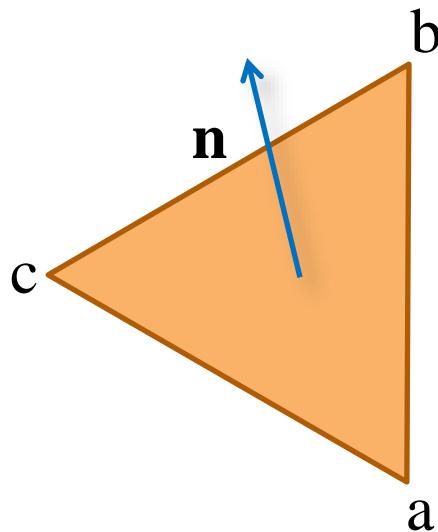
- Definiran le v **3D** prostoru
 - $\vec{c} = \vec{a} \times \vec{b}$
 - $|\vec{a} \times \vec{b}| = |\vec{a}| |\vec{b}| \sin\theta$
 - dolžina je ploščina paralelograma s stranicama a in b
- Rezultat je vektor, ki je **pravokoten** na \vec{a} in \vec{b} v **desnosučnem** k.s.
- **Ni** komutativen
- $\vec{a} \times \vec{b} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{bmatrix}$
 - izračun – xyzzy pravilo



[Demo](#)

Vektorski (križni) produkt

- V RG je pomemben pri iskanju **normal** – vektorjev pravokotnih na ravnino, 3D predmet ipd.
- **Primer** – imamo poligon z robnimi točkami a, b in c.
 - Normalo izračunamo s vektorskim produkтом
 - $\vec{n} = (b - a) \times (c - a)$
- Smer postavitve točk določa smer normale!





Transformacije

Transformacije

- Predmete v grafiki tipično predstavimo z **množico točk**
 - npr. oglišči poligonov
- Transformacije **preslikajo** eno konfiguracijo točk v drugo
- Spremenijo položaj, usmeritev, velikost in obliko predmetov
- Določajo tudi projekcijo iz 3D v 2D prostor



original



skaliranje



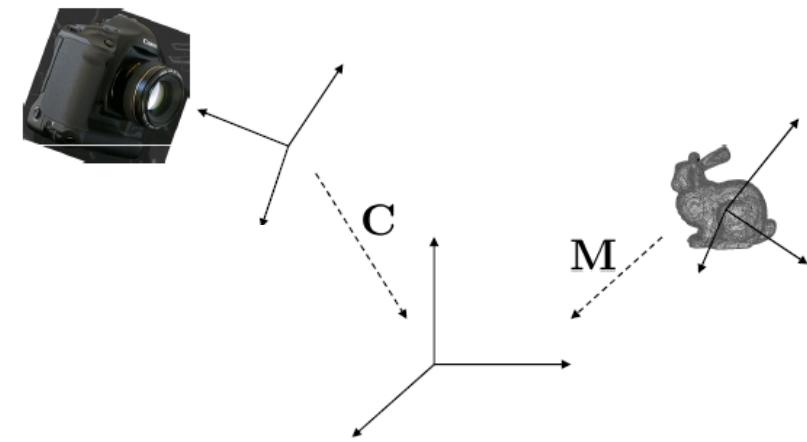
rotacija



striženje

Zakaj/kje transformacije

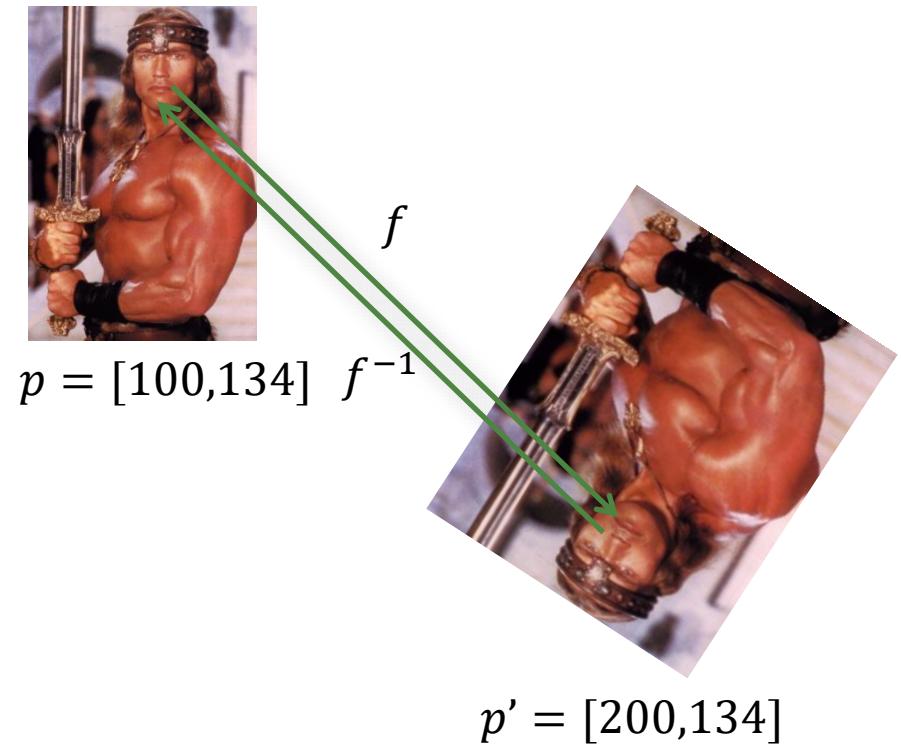
- Predmeti imajo položaj, usmeritev in velikost v prostoru.
 - translacija, rotacija, skaliranje
- Na predmete "gleda" kamera, ki ima položaj, usmeritev
 - translacija, rotacija
- 3D točke na koncu projiciramo na 2D projekcijsko ravnino
 - planarna projekcija



Preslikava

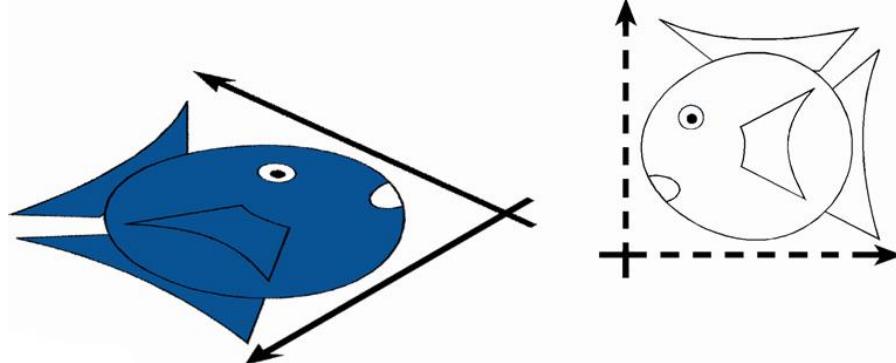
- **Transformacija** (preslikava) je funkcija, ki vzame točko ali vektor in jo preslika v neko drugo točko ali vektor:
 - npr.: preslika točko $p = [x, y]$ v originalni sliki (prostoru) v točko $p' = [x', y']$ v transformiranem prostoru

$$p' = f(p)$$

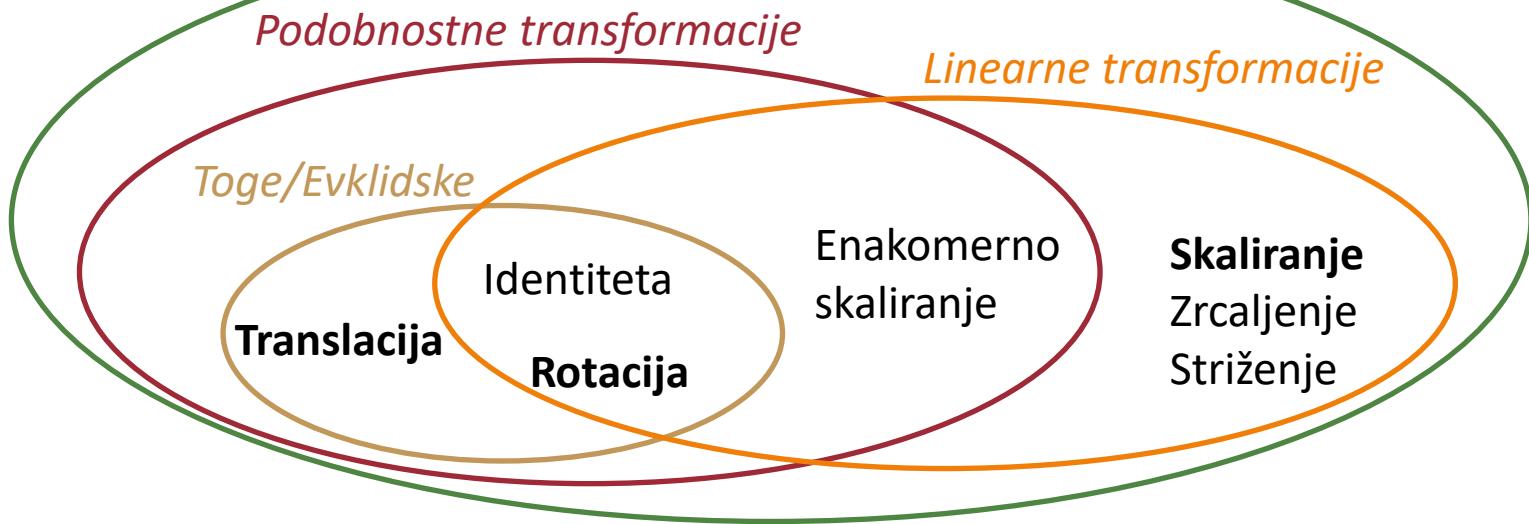


Afine transformacije

- Ohranjajo vzporedne črte

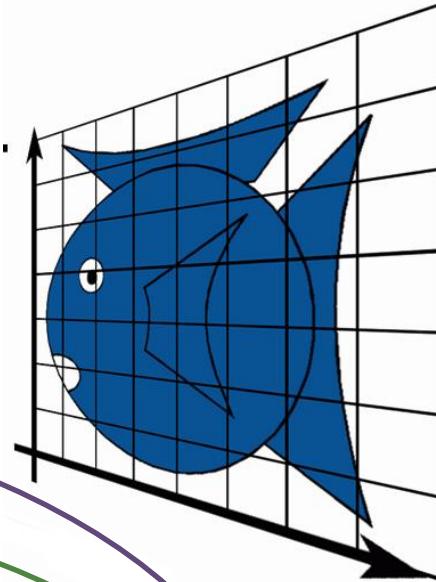
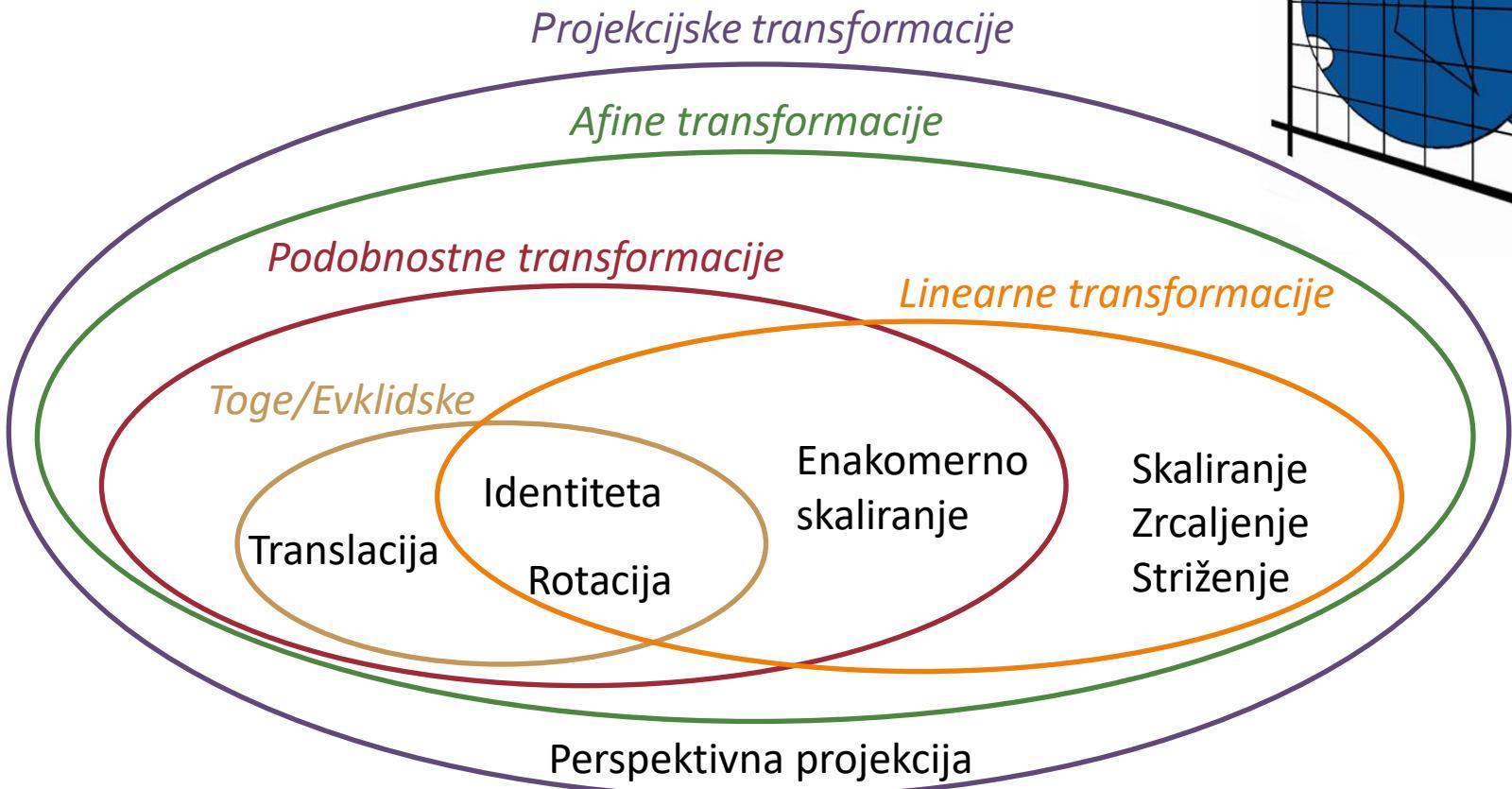


Afine transformacije



Projekcijske transformacije

- Ohranjajo črte



Afine 2D transformacije

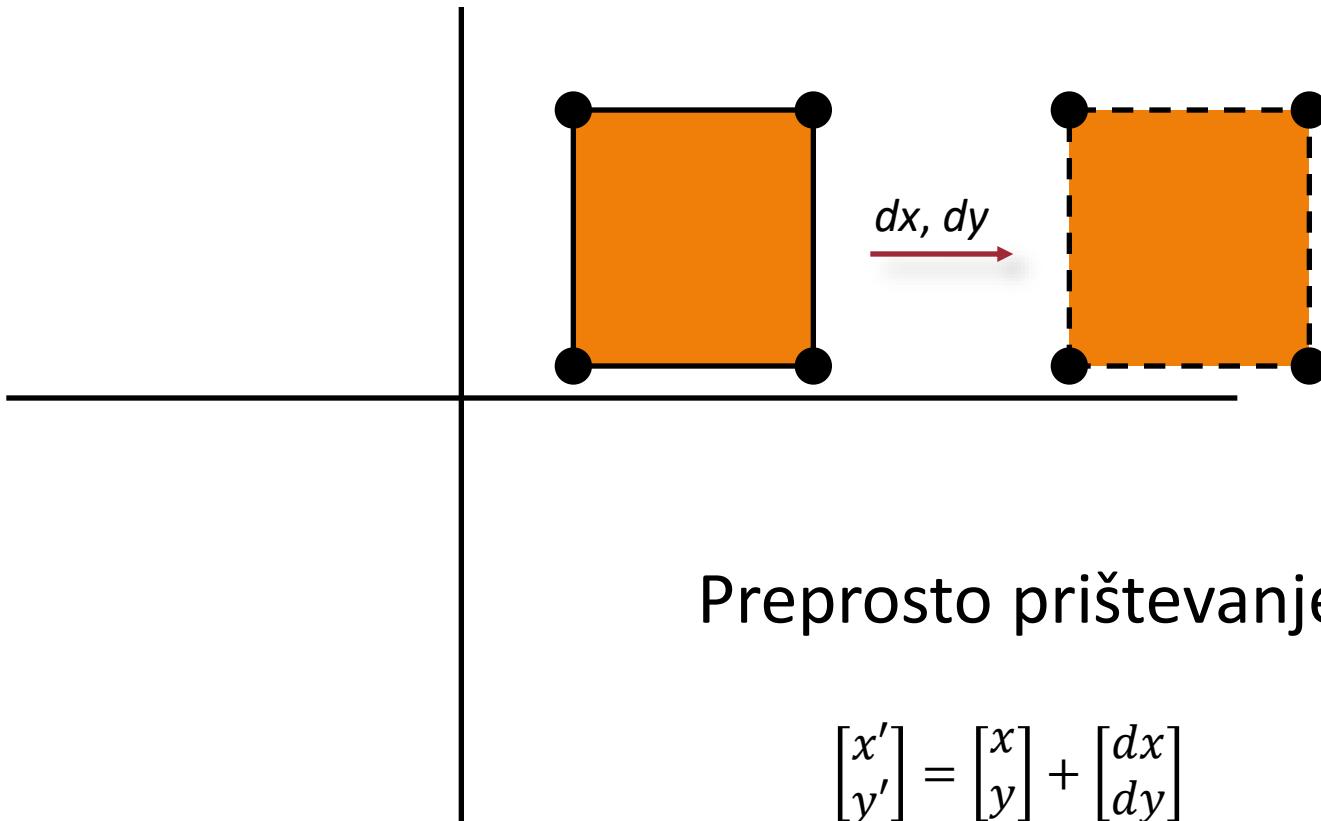
Predstavitev afinih transformacij v 2D

$$\begin{aligned}x' &= ax + by + c \\y' &= dx + ey + f\end{aligned}$$

$$\begin{bmatrix}x' \\ y'\end{bmatrix} = \begin{bmatrix}a & b \\ d & e\end{bmatrix} \begin{bmatrix}x \\ y\end{bmatrix} + \begin{bmatrix}c \\ f\end{bmatrix}$$


$$p' = Mp + t$$

Translacija

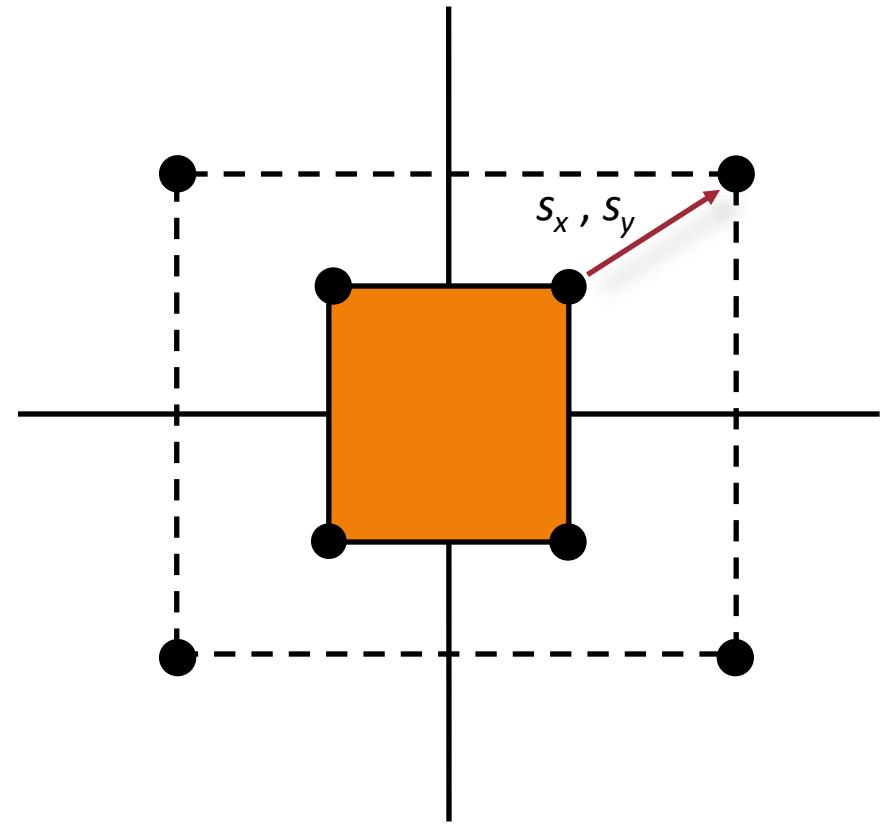


Skaliranje

- Množenje koordinat s faktorji skaliranja

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

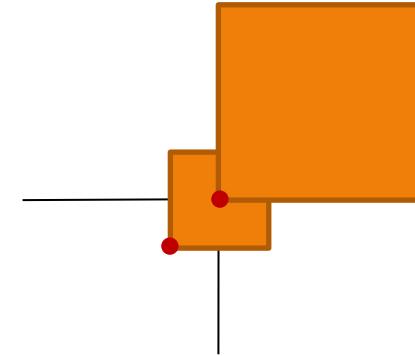
- Skaliranje poteka okoli **središča k.s.!**
- Če je s_x različen od s_y , dobimo **neenakomerno** skaliranje



Skaliranje okoli poljubne točke

■ Transformacije sestavimo

- premik točke skaliranja v središče k.s.
- skaliranje
- potem premik nazaj



Skaliranje glede na točko $[dx, dy]$:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} dx \\ dy \end{bmatrix} \right) + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

Rotacija

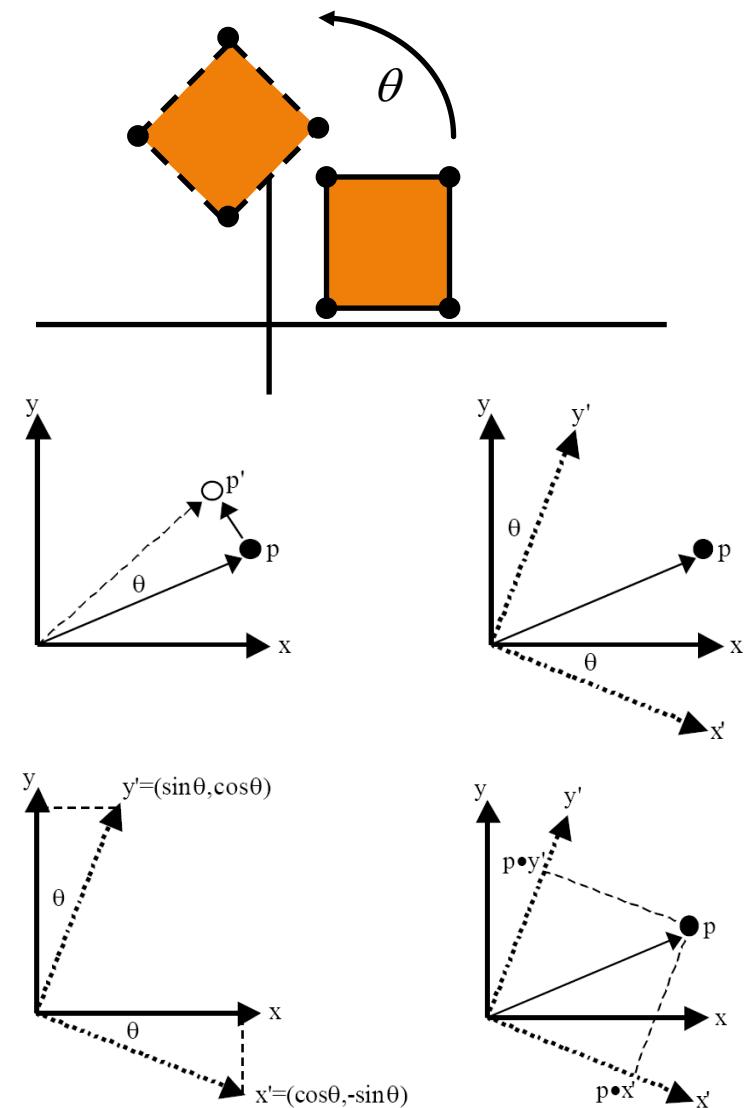
- Rotacija okoli središča k.s. je ekvivalentna rotacijsi k.s. v obratni smeri
 - enotski vektorji novega k.s. so:

$$\mathbf{x}' = \begin{bmatrix} \cos \theta \\ -\sin \theta \end{bmatrix} \quad \mathbf{y}' = \begin{bmatrix} \sin \theta \\ \cos \theta \end{bmatrix}$$

- koordinate točke v novem k.s. dobimo s projekcijo na nove vektorje – skalarni produkt

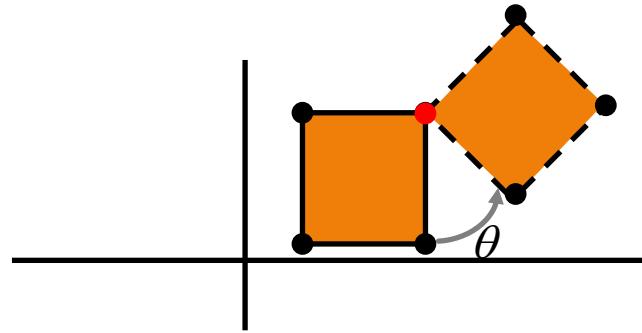
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Pozitivni koti so v **nasprotni smeri urinega kazalca** (desnosučno)



Rotacija okoli poljubne točke

- Kot pri skaliranju transformacije **sestavimo**
 - najprej premik točke rotacije v središče k.s., potem rotacija, potem premik nazaj



Rotacija glede na točko $[dx, dy]$:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \left(\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} dx \\ dy \end{bmatrix} \right) + \begin{bmatrix} dx \\ dy \end{bmatrix}$$

2D transformacije - povzetek

- Translacija: $P' = P + T$
- Skaliranje: $P' = S * P$
- Rotacija: $P' = R * P$

- Težava pri sestavljanju transformacij:
 - Imamo heterogene operacije (seštevanje in množenje)
 - Želimo homogene operacije (npr. vse samo množenje), tako da lahko transformacije združujemo

Homogene koordinate

- Matematični konstrukt, ki omogoča, da vse affine transformacije pretvorimo v **množenje matrik** in jih tako enostavno sestavljamo
- Uvedemo **dodatno koordinato za zapis točk**
 - v RG za točke postavimo $w = 1$, za vektorje $w = 0$
 - affine transformacije ne spremnjajo vrednosti w
 - projekcijske spremnjajo w
- Pretvorbo iz točke v homogenih koordinatah v običajne koordinate dobimo z deljenjem

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ w \end{bmatrix}, w <> 0 \quad \text{oz.} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}, w < 0 \quad \text{v 3D}$$

$$\mathbf{p} = \begin{bmatrix} x \\ w \\ y \\ w \end{bmatrix} \quad \text{oz.} \quad \mathbf{p} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{bmatrix} \quad \text{v 3D}$$

2D transformacije v homogenih koordinatah

- Bistvo vsega: **translacija** postane **množenje**
- Skaliranju in rotaciji dodamo eno vrstico/stolpec

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

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

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

Sestavljanje transformacij

- To omogoča **enostavno sestavljanje transformacij**
 - vse transformacije so predstavljene z množenjem
 - transf. matrike lahko med seboj zmnožimo v eno samo matriko
- Pri sestavljanju je **važen vrstni red!** Množenje matrik ni komutativno.
 - $T^*S \neq S*T$
- Vrstni red operacij je od **desne proti levi**
 - prva operacija je najbolj desna – S v zadnjem primeru na desni

rotacija okoli poljubne točke v **navadnih** koordinatah:

$$x' = R*(x+t)-t$$

rotacija okoli poljubne točke v **homogenih** koordinatah:

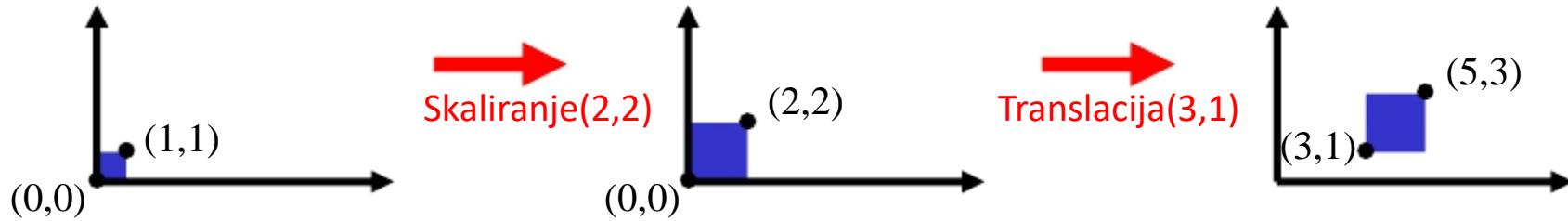
$$x' = T^{-1} * R * T^* x = M * x$$

translacija, rotacija in skaliranje okoli središča:

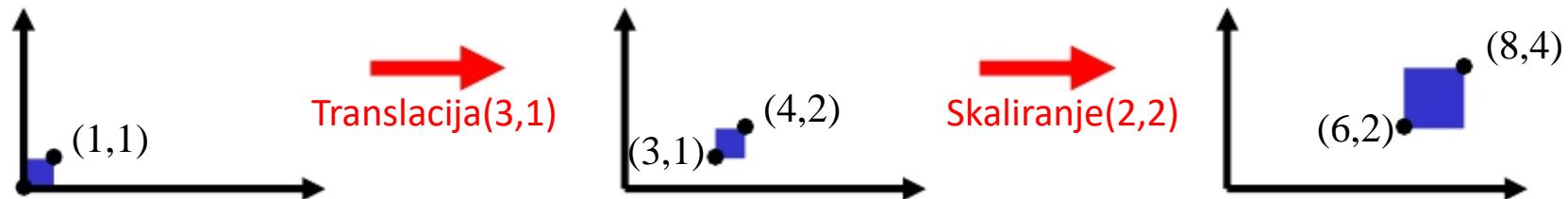
$$T^* R^* S * x = M * x$$

Sestavljanje ni komutativno

Skaliranje pred translacijo: $p' = T(S p) = TS p$



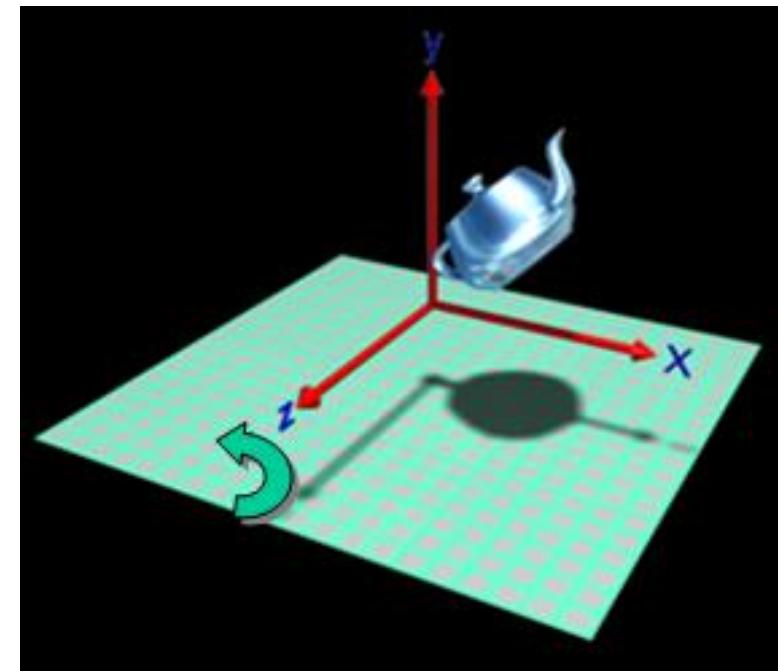
Translacija pred skaliranjem: $p' = S(T p) = ST p$



Afine 3D transformacije

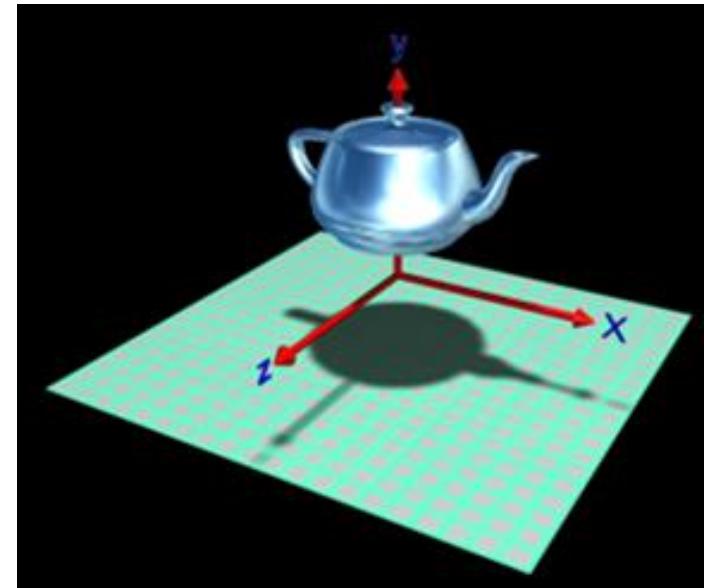
3D transformacije

- Uvedemo 3. dimenzijo in še vedno delamo s homogenimi koordinatami
- Posplošitev je za večino transformacij enostavna, nekoliko se zaplete pri predstavitvi rotacij



3D translacija

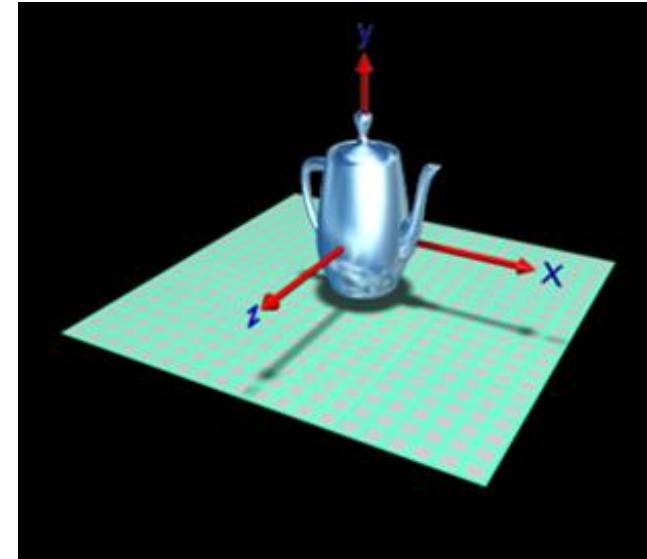
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



3D skaliranje

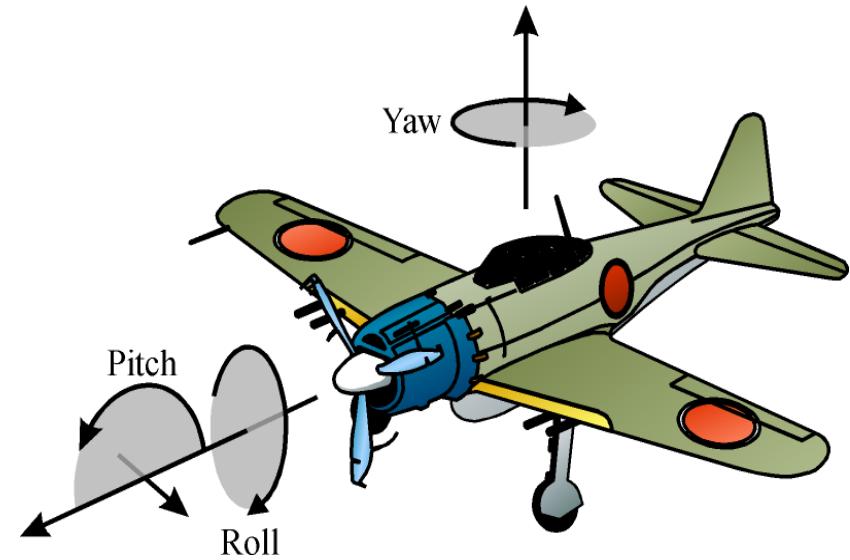
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Enako kot pri 2D: če želimo skalirati okoli poljubne točke v prostoru, uporabimo sestavljanje s translacijo v izhodišče k.s.



Rotacija 3D predmeta

- Rotacijo predmeta lahko opišemo kot rotacijo preko vseh treh osi
 - prostostne stopnje lahko po letalsku imenujemo "roll, pitch , yaw" (kotaljenje, naklon, odklon)



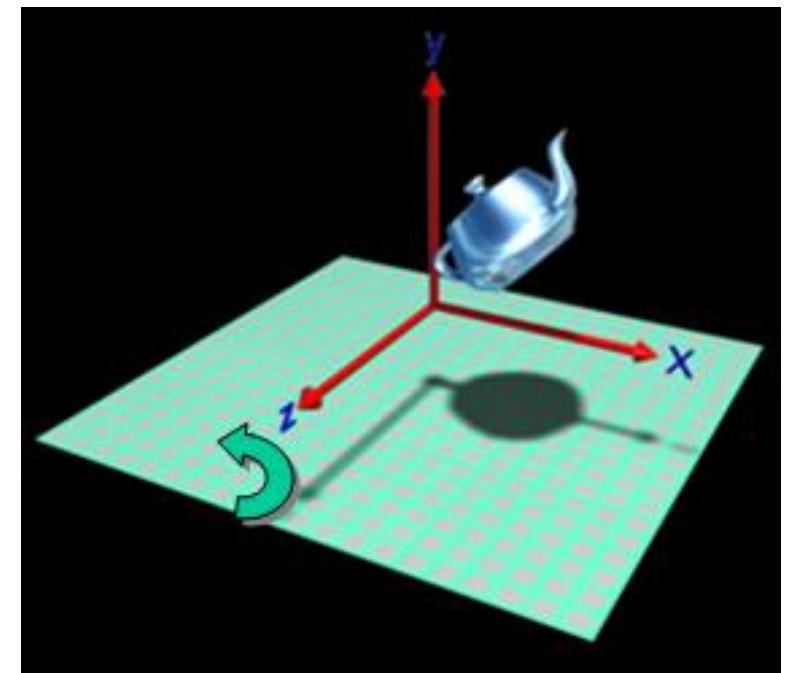
3D rotacija

- Rotacija okoli treh osi, x, y ali z:

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

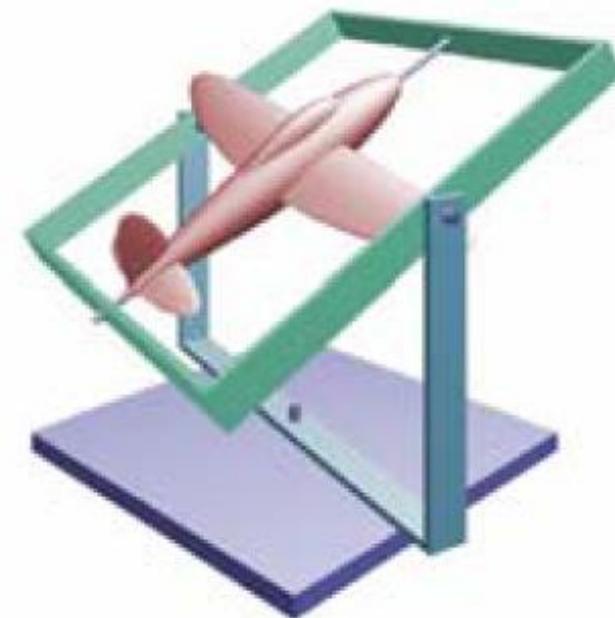
$$Y: \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

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



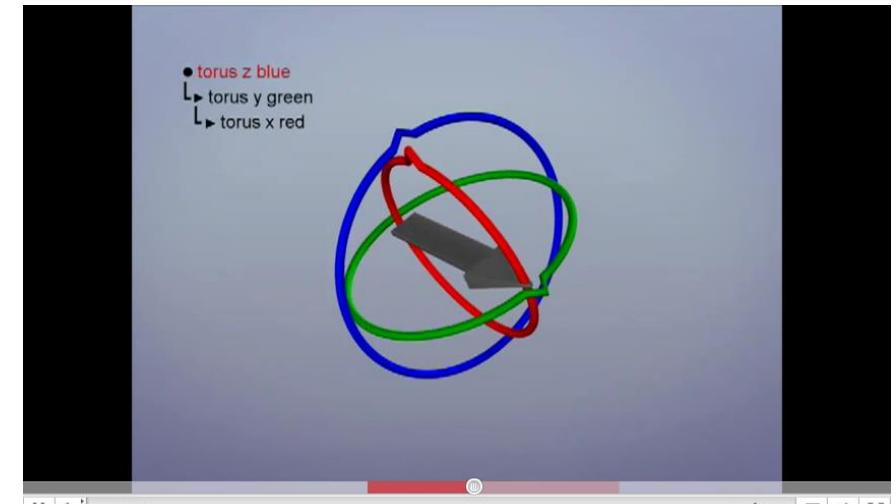
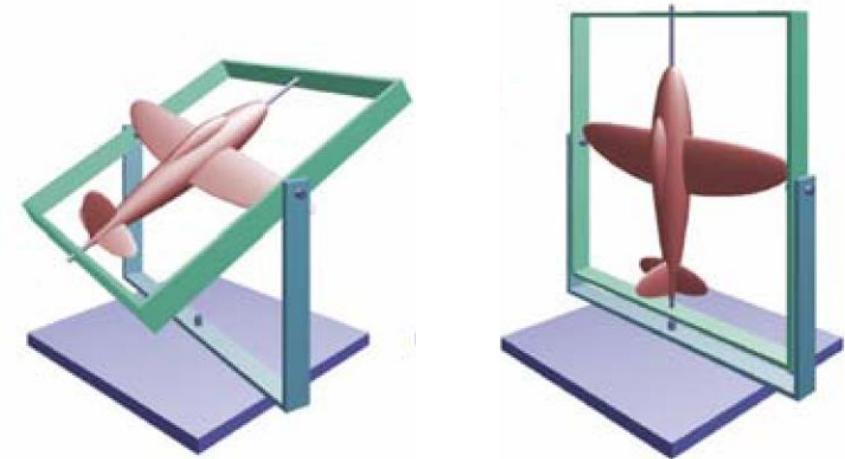
Eulerjevi koti

- **Eulerjevi koti** predstavljajo kote rotacij okrog koordinatnih osi, da dosežemo želeno usmeritev (θ_x , θ_y , θ_z)
 - vsako zahtevano rotacijo lahko opišemo na ta način
- Celotno matriko rotacije zapišemo kot produkt matrik, npr.:
$$M = R(\theta_z)R(\theta_y) R(\theta_x)$$
- **Pozor:** rotacije niso komutativne, **važen je vrstni red**
 - npr. vrtenje za [10, 45, 90] bi lahko zapisali kot
$$R_z(90)R_y(45)R_x(10) \text{ ali } R_y(45)R_x(10)R_z(90) \text{ ali ...}$$



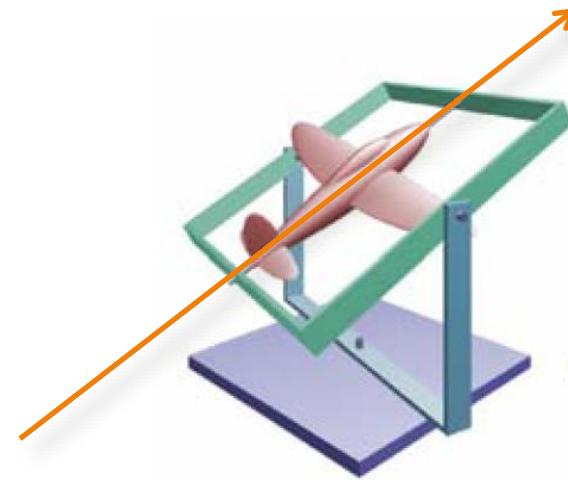
Problemi pri rotacijah z Eulerjevimi koti

- Do problema pride, če se dve osi vrtenja **poravnata** med seboj.
Temu pojavu pravimo **kardanska zapora** (*Gimbal Lock*)
- Poravnanje dveh ali več osi vrtenja predmeta pomeni **izgubo prostostne stopnje**
 - predmet se ne bo vrtel tako, kot smo si zamislili
 - problem pri spremembah in animaciji rotacij

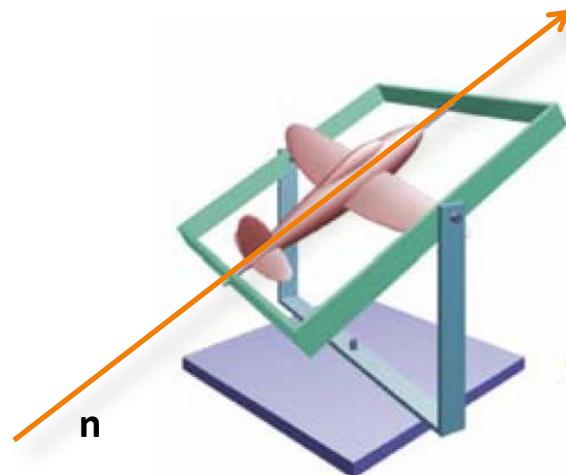


Alternativa za predstavitev rotacij

- Rotacijo predstavimo z **osjo** okoli katere je predmet rotiran **in kotom** rotacije
 - vektor $[x, y, z]$ pove orientacijo osi
 - kot θ pove kot rotacije okoli osi
 - ločeno premikamo os in kot
- Pri animacijah si lahko pri tovrstni predstavitvi pomagamo s **kvaternioni**

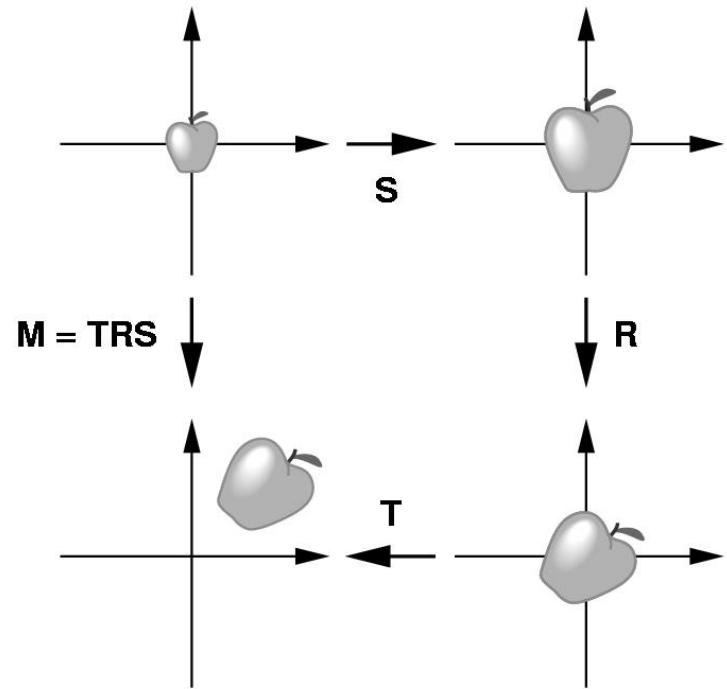


- 4D kompleksno število
 - $q = s + xi + yj + zk$
 - $q = (s, x, y, z) = (s, \mathbf{v})$
 - $i^2 = j^2 = k^2 = ijk = -1$
 - $ij = k, jk = i, ki = j,$
 $ji = -k, kj = -i, ik = -j$
- Rotacijo okoli \mathbf{n} za kot q predstavimo kot:
 - $q = \left(\cos \frac{\theta}{2}, \mathbf{n} \sin \frac{\theta}{2} \right), \|\mathbf{n}\| = 1$
- Točko p kot kvaternion predstavimo z $(0, p)$ in jo zavrtimo kot:
 - $p' = qpq^{-1}$
- Lahko pretvarjamo med kvaternioni in Eulerjevimi koti!



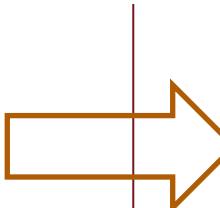
Sestavljanje transformacij

- Sestavljene affine transformacije predstavimo z **množenjem matrik**, torej lahko transformacije poljubno kombiniramo z množenjem matrik
- Najbolj desna matrika bo prva za transformacijo!
- Vrstni red je pomemben – množenje matrik **ni komutativno**
- Tipična sestava osnovnih treh transformacij je TRS
 - translacija
 - *rotacija
 - *skaliranje
 - (* točka)



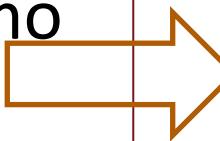
Primer v kodi

- Definiramo transformacijsko **matriko**, ki bo uporabljena pri izrisu



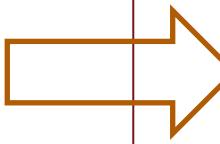
```
mat4.identity(modelMatrix);
mat4.translate(modelMatrix, modelMatrix, [0,0,-10]);
mat4.rotateX(modelMatrix, modelMatrix, -2);
//mat4.rotateY(modelMatrix, modelMatrix, t * 0.6);
mat4.scale(modelMatrix, modelMatrix, [2,1,1]);
```

- Matriko kot uniformo prenesemo v **senčilnik oglišč**



```
queue.writeBuffer(uniformBuffer, 0, modelMatrix);
```

- V **senčilniku oglišč** oglišča pomnožimo z matriko

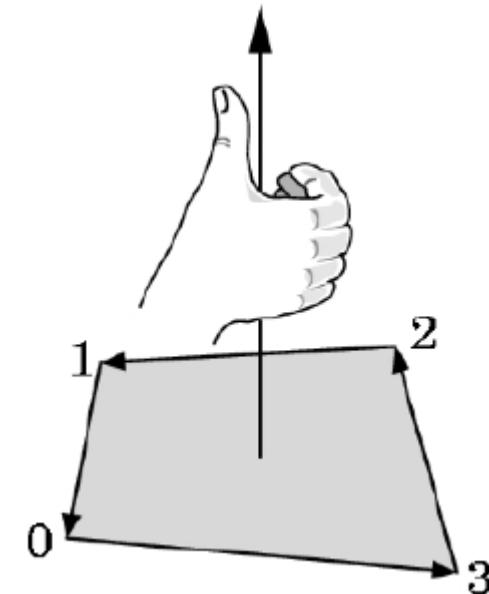


```
struct VertexInput {
    @location(0) position : vec3f,
}
struct VertexOutput {
    @builtin(position) position : vec4f,
}
@group(0) @binding(0) var<uniform> matrix : mat4x4f;
@vertex
fn vertex(input : VertexInput) -> VertexOutput {
    var output : VertexOutput;
    output.position = matrix * vec4(input.position, 1);
    return output;
}
```

Transformacija normal

- Normale so definirane kot vektor pravokoten na drug vektor oz. ploskev
- Se **ne** transformirajo na enak način kot točke:

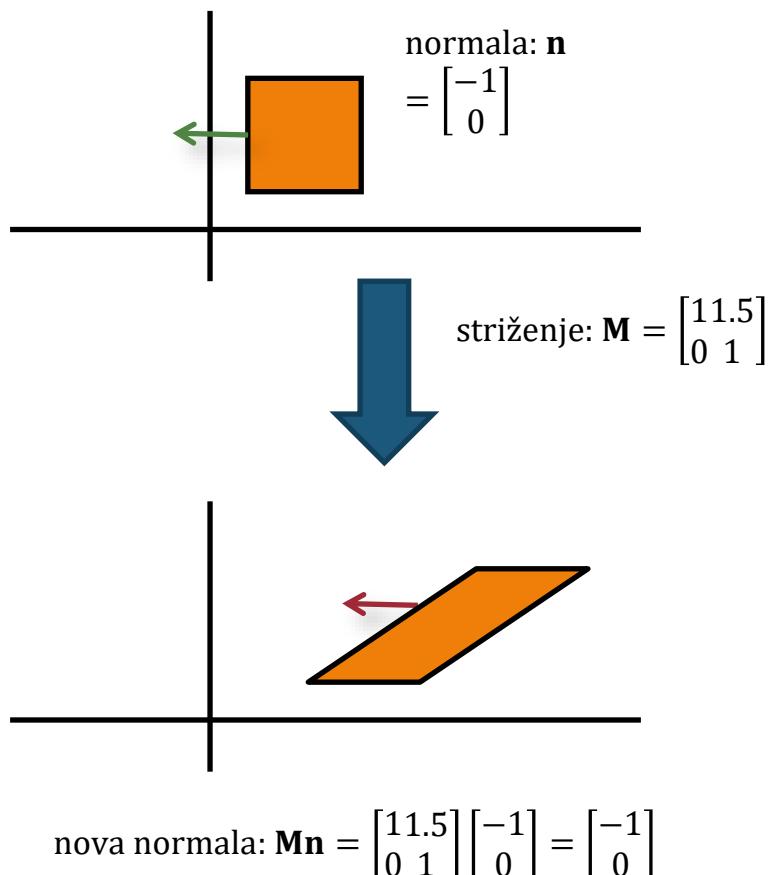
$$\begin{aligned}\mathbf{n} \cdot \mathbf{v} = 0, \text{ oziroma } \mathbf{n}^T \mathbf{v} = 0 \\ \text{če } \mathbf{v} \text{ transformiramo z } \mathbf{M} \text{ in } \mathbf{n} \text{ z } \mathbf{X} \\ (\mathbf{Xn})^T (\mathbf{Mv}) = \mathbf{n}^T \mathbf{X}^T \mathbf{Mv} = 0 \\ \text{če postavimo } \mathbf{X} = (\mathbf{M}^{-1})^T \\ \mathbf{n}^T \mathbf{X}^T \mathbf{Mv} = \mathbf{n}^T \mathbf{M}^{-1} \mathbf{Mv} = \mathbf{n}^T \mathbf{v} = 0\end{aligned}$$



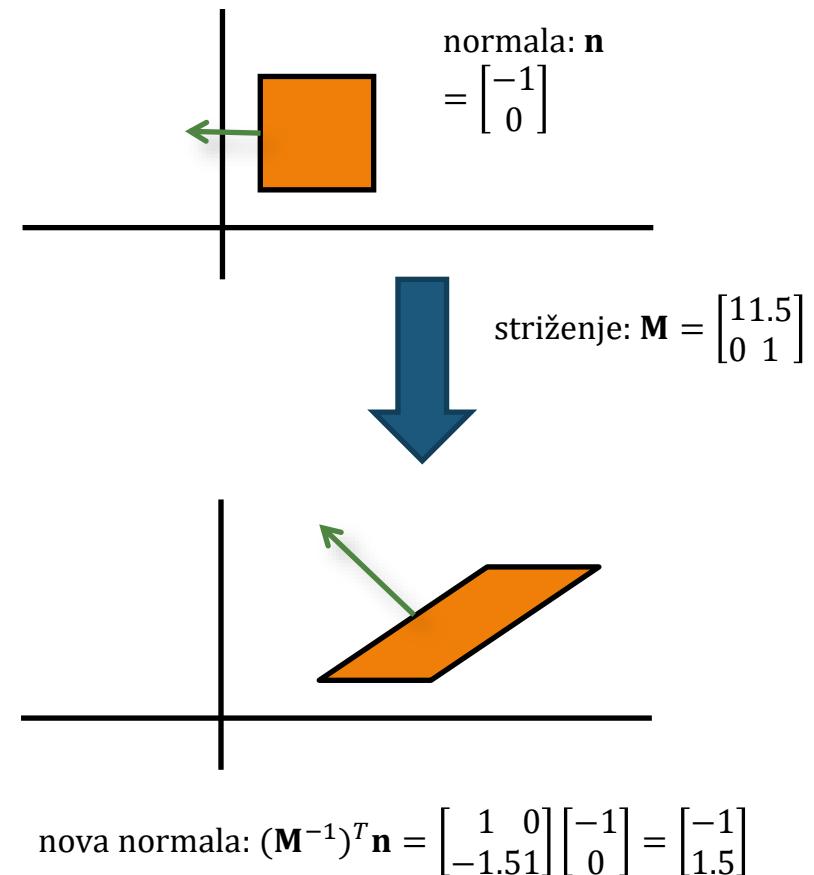
- Če točko transformiramo z \mathbf{M} , normale transformiramo z $(\mathbf{M}^{-1})^T$

Transformacija normal - primer

Napačno:



Pravilno:





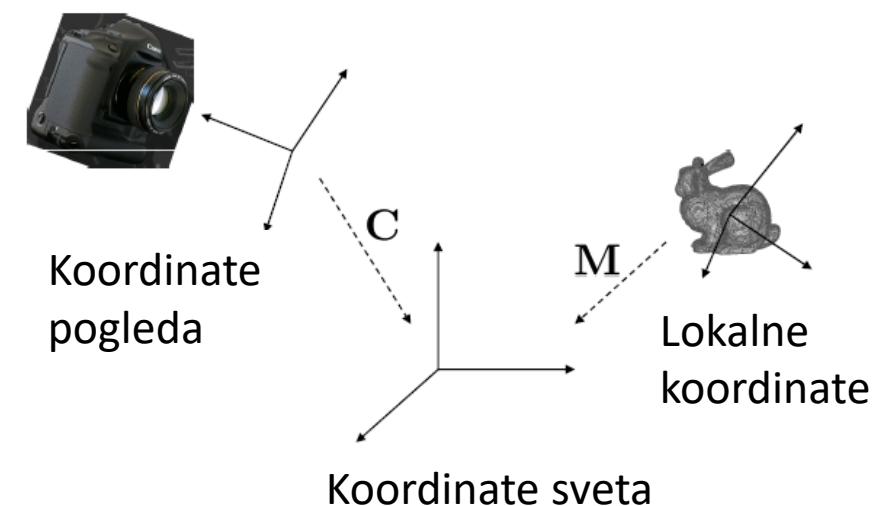
Koordinate pogleda

Koordinatni sistemi v RG

- 3D Scena v RG se sestoji iz predmetov, postavljenih v **prostor – svet**
- Na svet gledamo skozi **kamero**, ki je seveda tudi postavljena nekje v prostoru

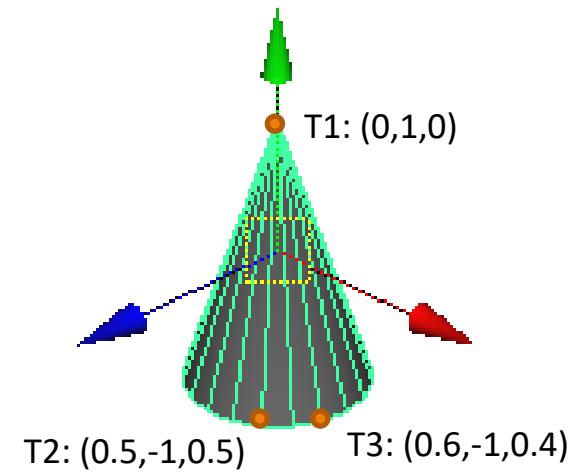


- Predmet
 - lokalne koordinate
- Svet
 - koordinate sveta
- Kamera
 - koordinate pogleda



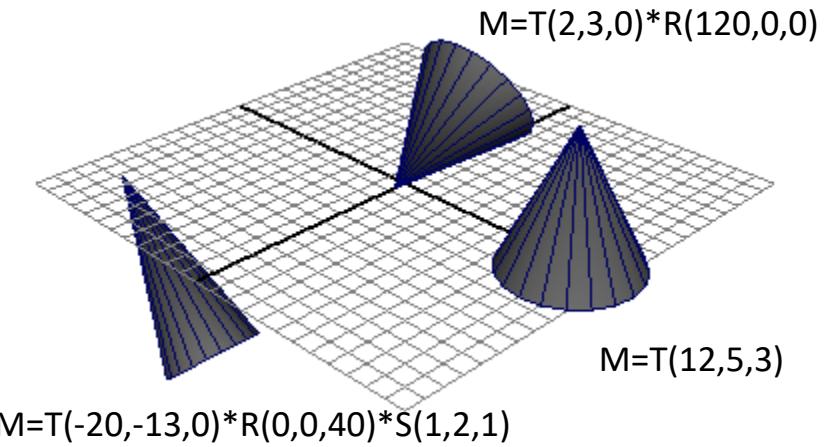
Lokalne koordinate

- Posamezen predmet je narejen relativno glede na svoj **lokalni koordinatni sistem**
 - ta je navadno v središču predmeta, lahko pa tudi na robu – kot določi ustvarjalec predmeta
 - **lokalne koordinate** določajo položaj točk glede na središče lokalnega k.s.



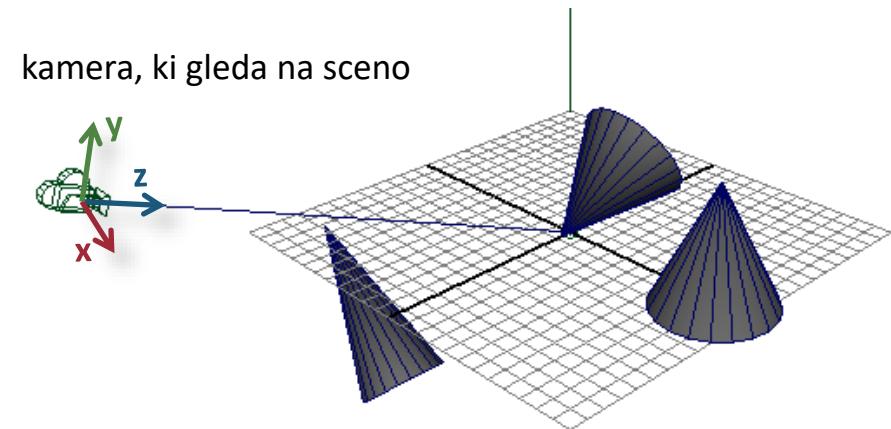
Koordinate sveta

- Predmeti so preko **transformacije modela** (*modeling transform*) postavljeni v **svet**
 - točke se iz lokalnih koordinat transformirajo v **koordinate sveta**
 - transformacija modela je sestavljena iz produkta posameznih afinih transformacij
 - npr. transl. * rot. * skaliranje
 - $M = TRS$



Koordinate pogleda

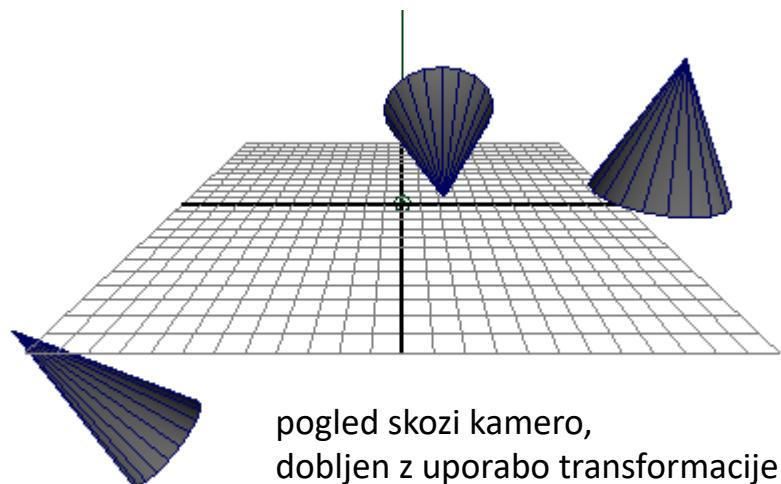
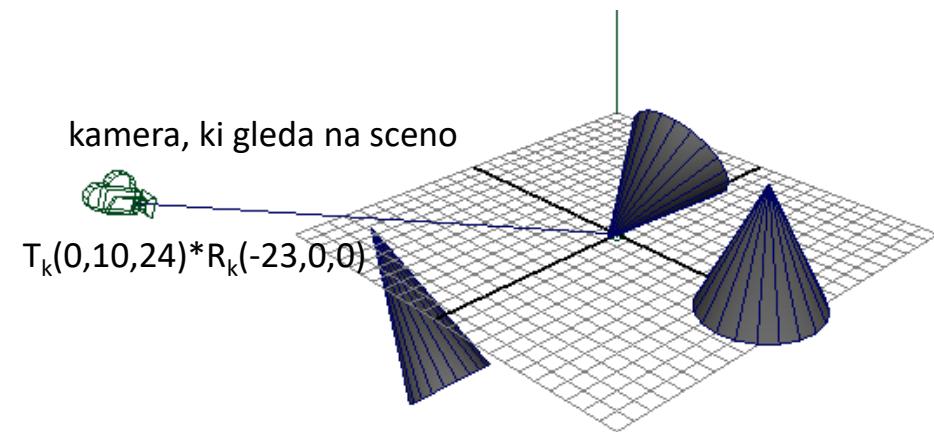
- Na svet gleda **kamera**
 - je na nekem položaju in orientaciji v svetu
- **Projekcijska ravnina:** ravnina, na katero se projicira slika, ki jo vidimo, če pogledamo skozi kamero
- Koordinatni sistem **kamere/pogleda**
 - navadno x-y koordinatni sistem pokriva projekcijsko ravnino
 - z koordinata gleda **v** (levo sučen) ali **iz** (desno sučen) scene
- Predmete iz koordinat sveta v **koordinate pogleda** (kar vidi kamera) preslikamo s **transformacijo pogleda** (*view transform*)



- Položaj kamere je določen s **transformacijo kamere**: $T_k R_k$
- En način za izračun transformacije pogleda je s transformacijo, ki je **obratna** transformaciji kamere:

$$V = (T_k R_k)^{-1} = R_k^{-1} T_k^{-1}$$

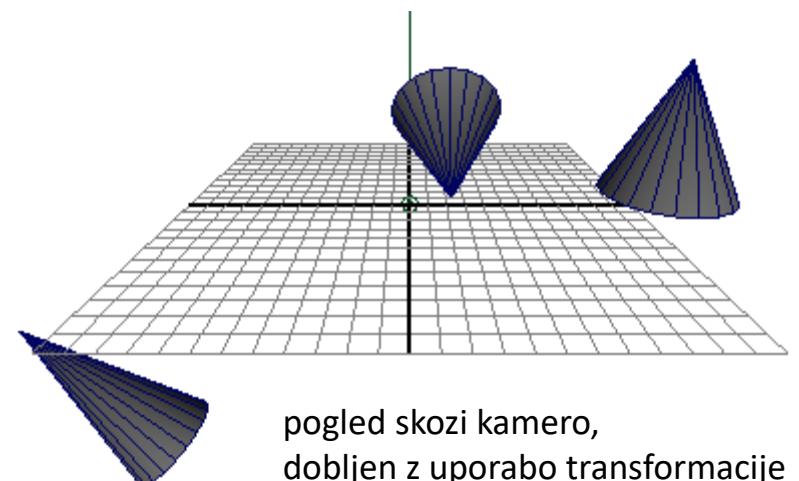
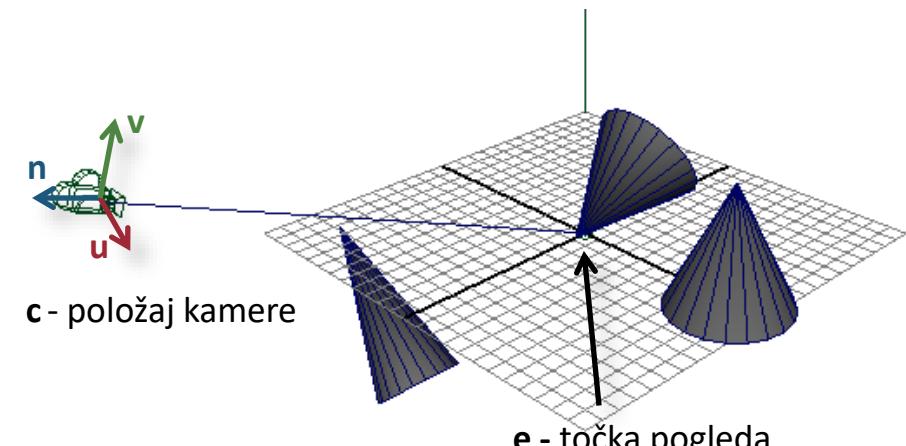
Koordinate pogleda (1)



Koordinate pogleda (2)

- Na svet gleda kamera
 - je na nekem položaju c
 - kamera je obrnjena v točko pogleda g
 - y kaže smer gor (*up axis*)
- K.s. kamere (levo sučen) določimo kot:
 - $n = \frac{c-e}{|c-e|}$ kaže v smeri pogleda
 - $u = \frac{y \times n}{|y \times n|}$ kaže desno
 - $v = \frac{n \times u}{|n \times u|}$ kaže gor
- Transformacija pogleda je matrika pretvorbe med obema k.s.
 - preslikava v nov koordinatni sistem

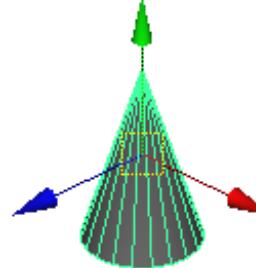
$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} (X - C) = \begin{bmatrix} u_x & u_y & u_z & -u \cdot c \\ v_x & v_y & v_z & -v \cdot c \\ n_x & n_y & n_z & -n \cdot c \\ 0 & 0 & 0 & 1 \end{bmatrix} X$$

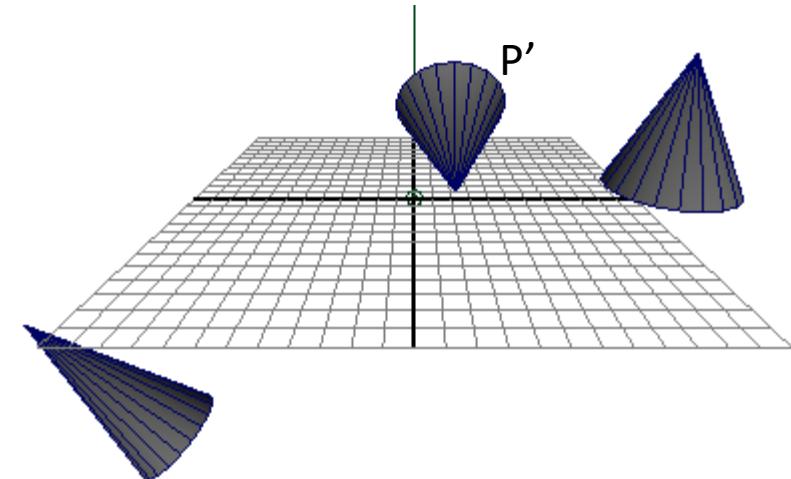


Od lokalnih do koordinat pogleda

- Predmet, ki je predstavljen z matriko točk v lokalnih koordinatah preslikamo v koordinate pogleda
 - matriko točk P preslikamo s transformacijo modela M in transformacijo pogleda V
 - *modelView matrix*
- Na primer:

$$\begin{aligned}P' &= VMP \\&= R_k^{-1} T_k^{-1} T_m R_m S_m P = XP\end{aligned}$$

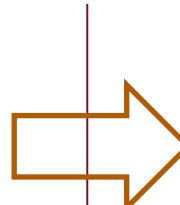

$$P = \begin{bmatrix} 0 & 0.5 & 0.6 & \dots \\ 1 & -1 & -1 & \dots \\ 0 & 0.5 & 0.4 & \dots \\ 1 & 1 & 1 & \dots \end{bmatrix}$$



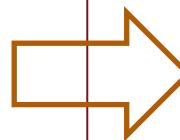
$$P' \cong \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & -0.8 & -0.6 & -7 \\ 0 & 0.6 & 0.8 & -23 \\ 0 & 0 & 0 & 1 \end{bmatrix} P \cong \begin{bmatrix} 2 & 2.5 & 2.6 \\ -8 & -6.7 & -6.7 \\ -22 & -23.8 & -23.7 \\ 1 & 1 & 1 \end{bmatrix}$$

Primer v kodi

- Transformacija modela
 - lokalne koord. -> koordinate sveta
- Transformacija pogleda
 - koordinate sveta -> koordinate pogleda
 - kot inverz premika kamere s translacijo in rotacijo
 - z *lookAt*, kjer definiramo položaj kamere, točko pogleda in nagib kamere
- Obe transformaciji pomnožimo v t.i. *modelView* matriko
 - preslika iz lokalnih koord. -> koordinate pogleda



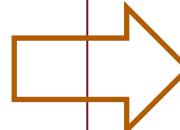
```
// local -> world coordinates (model transform)
mat4.identity(modelMatrix);
mat4.translate(modelMatrix, modelMatrix, [0,0,-10]);
mat4.rotateX(modelMatrix, modelMatrix, -2);
mat4.scale(modelMatrix, modelMatrix, [3,1,1]);
```



```
// world -> view coordinates (view transform)
mat4.identity(viewMatrix);

// first option - inverse camera transform
mat4.translate(viewMatrix, viewMatrix, [-3,-2,4]);
mat4.rotateZ(viewMatrix, viewMatrix, 10*Math.PI/180);
mat4.invert(viewMatrix, viewMatrix);

// second option - lookat (eye, center, up)
mat4.lookAt(viewMatrix, [-3,-2,4], [-1,-1,-4], [0,1,0]);
```



```
// combine into modelView transform
mat4.identity(modelView);
mat4.multiply(modelView, modelView, viewMatrix);
mat4.multiply(modelView, modelView, modelMatrix);
```

REFERENCE

- Matematične osnove: [Vector Math for 3D Computer Graphics](#)
- N. Guid: Računalniška grafika, FERI Maribor