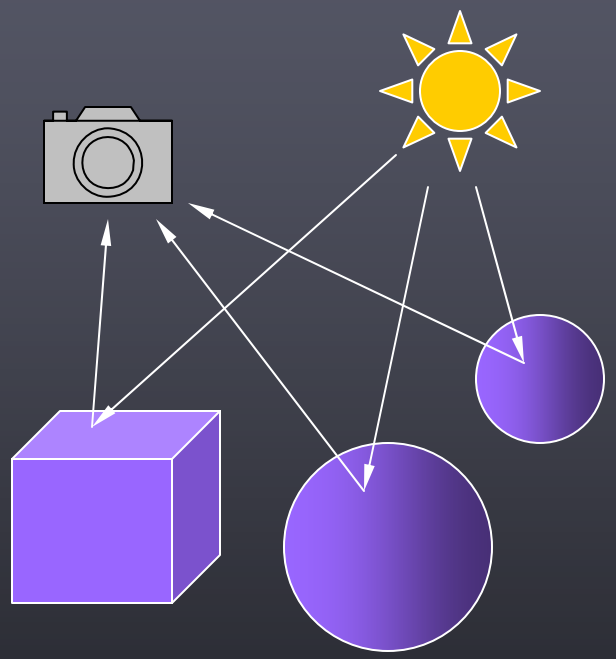
Transformationer i 3D

Gustav Taxén
gustavt@csc.kth.se

2D1640 Grafik och Interaktionsprogrammering VT 2007

Bakgrund

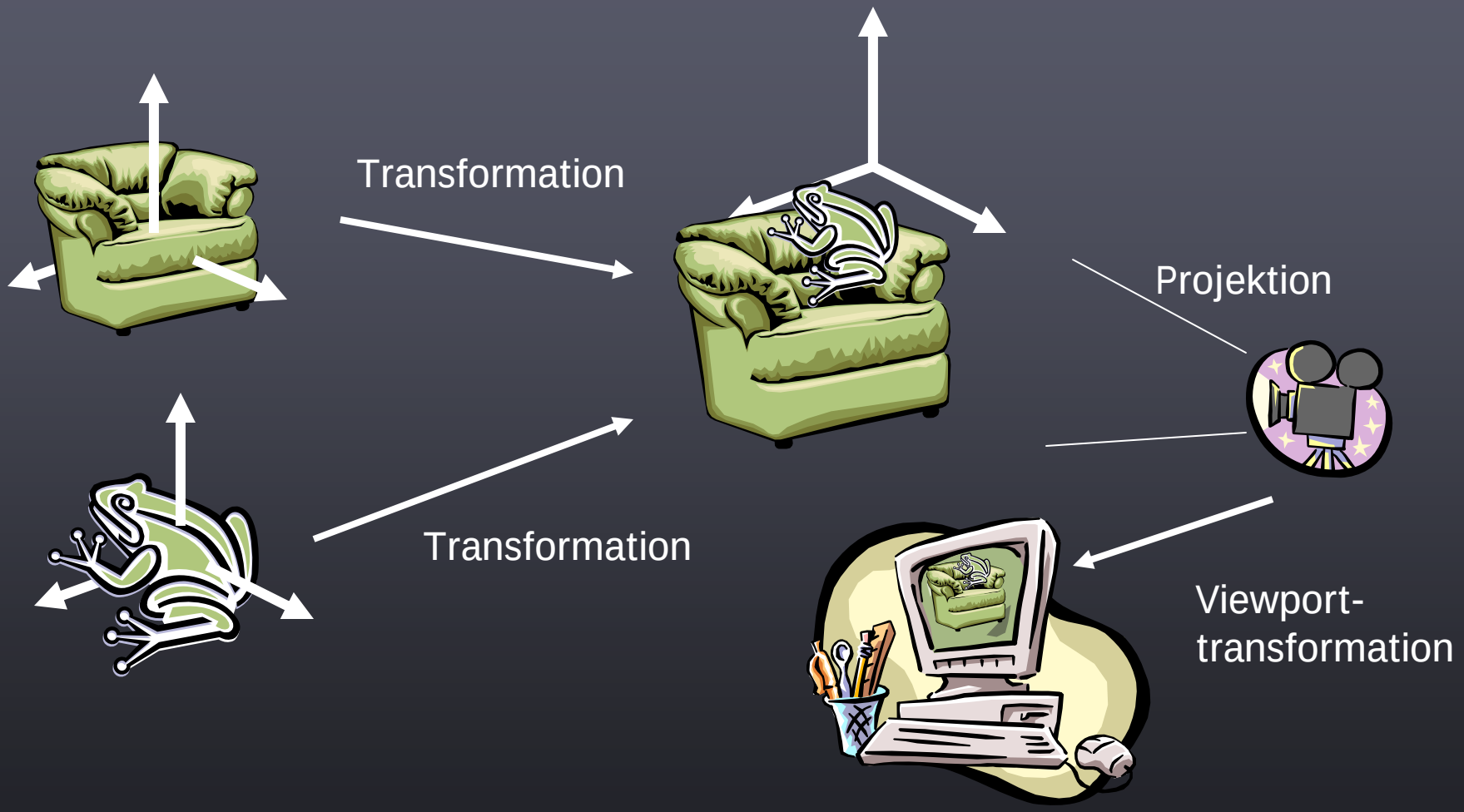
- Ett smidigt sätt att arbeta med 3D-grafik är att tänka sig att man har en virtuell kamera som ”betraktar” de föremål man vill rita.
- Om vi simulerar hur ljus från ljuskällor reflekteras av ytor och når kameran kan vi få en bild som i bästa fall ser ut som ett fotografi.



Bakgrund

- Vi börjar med att ta reda på hur vi kan transformera 3D-föremål och gå från tre dimensioner till två.
- Vi tar hand om ljusreflektion och skuggning senare i kursen!

Transformationer i 3D



Matematiska byggstenar

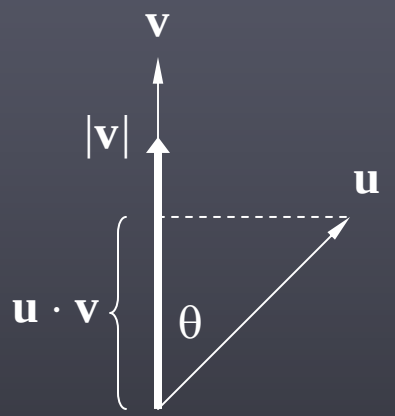
- Skalär (storhet) u
- Punkt (position) P
- Vektor (längd och riktning) $\mathbf{v}$

- Kvaternion (orientering) q

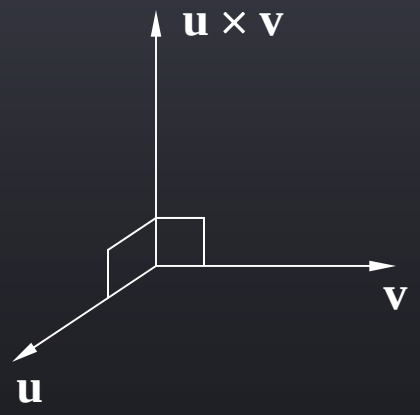
Skalär- och kryssprodukt

Två av de vanligaste operationerna i datorgrafik.

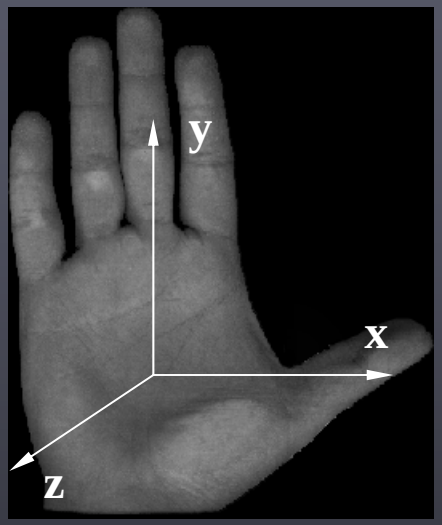
$$\mathbf{u} \cdot \mathbf{v} = (u_x v_x, u_y v_y, u_z v_z) = |\mathbf{u}||\mathbf{v}| \cos \theta$$



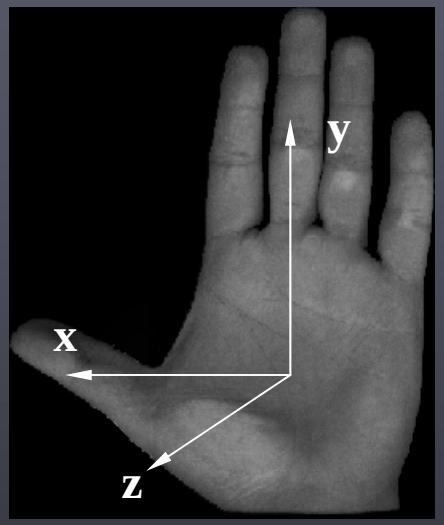
$$\mathbf{u} \times \mathbf{v} = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x)$$



Höger- och vänsterhänt koordinat-system



OpenGL använder ett **högerhänt** koordinatsystem med
x-axeln till höger, **y**-axeln uppåt.
z-axeln pekar utåt ur skärmen.



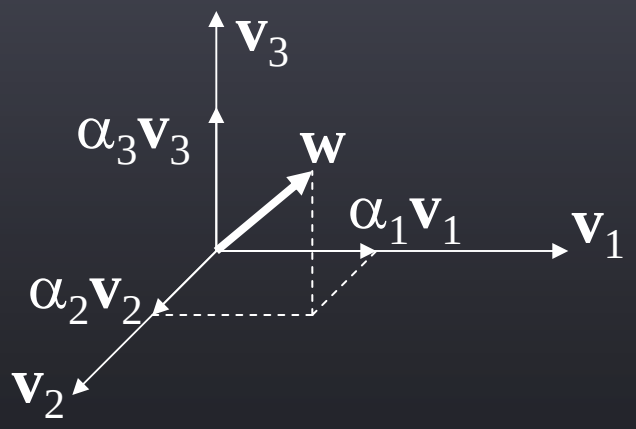
Direct 3D använder ett **vänsterhänt** koordinatsystem med
x-axeln till höger, **y**-axeln uppåt.
z-axeln pekar in i skärmen.

I fortsättningen arbetar vi i **högerhänta** koordinatsystem.

Basvektorer

Givet n st n-dimensionella **basvektorer** $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n$ som alla är vinkelräta mot varandra kan vi skriva vektorn $\mathbf{w}$ som

$$\mathbf{w} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \alpha_3\mathbf{v}_3 + ... + \alpha_n\mathbf{v}_n$$



Basvektorer

Vi inför ”vektorer i vektorer” för att kunna skriva

$$\mathbf{w} = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \ldots + \alpha_n\mathbf{v}_n$$

lite kortare som:

$$\mathbf{w} = \mathbf{a}^{\mathrm{T}} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \ldots \\ \mathbf{v}_n \end{pmatrix}$$

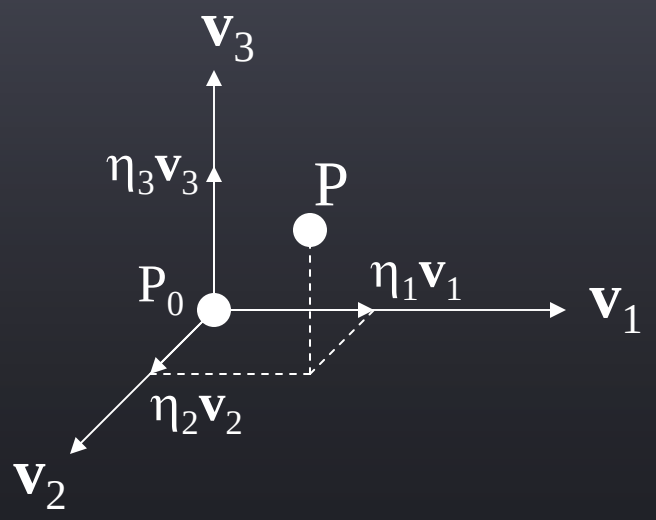
där

$$\mathbf{a} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \ldots \\ \alpha_n \end{pmatrix} \qquad \mathbf{a}^{\mathrm{T}} = \begin{bmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_n \end{bmatrix}$$

Ramar (frames)

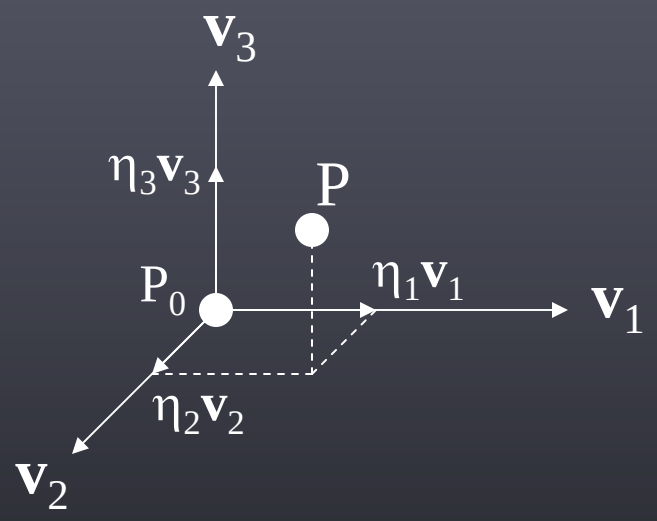
Vi definierar en **ram** som $(\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_n, P_0)$
där P_0 kallas ramens **origo**.
För punkter i denna ram gäller:

$$P = P_0 + \eta_1\mathbf{v}_1 + \eta_2\mathbf{v}_2 + \eta_3\mathbf{v}_3 + ... + \eta_n\mathbf{v}_n$$



Ramar (frames)

eller $P = \mathbf{p}^T \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \cdots \\ \mathbf{v}_n \\ P_0 \end{pmatrix}$



där

$\mathbf{p}^T = \begin{bmatrix} \eta_1 & \eta_2 & \ldots & \eta_n & 1 \end{bmatrix}$

Vi definierar $1 \cdot P = P$ och $0 \cdot P = 0$

Ramar (frames)

En **riktningsvektor** $\mathbf{w}$ i denna ram skrivs

$$\mathbf{w} = \mathbf{a}^{\mathrm{T}} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \cdots \\ \mathbf{v}_n \\ \mathrm{P}_0 \end{pmatrix}$$

där

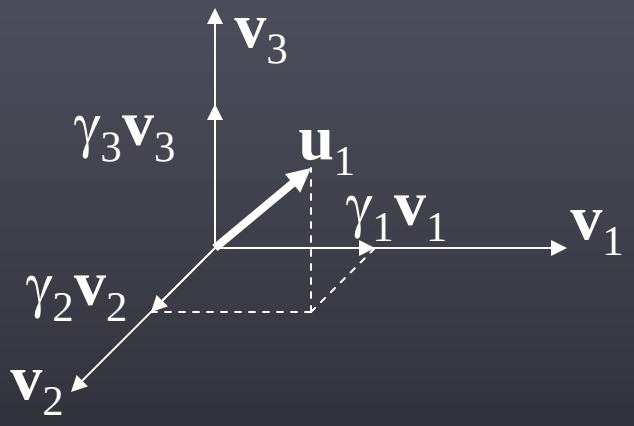
$$\mathbf{a}^{\mathrm{T}} = \begin{bmatrix} \alpha_1 & \alpha_2 & \ldots & \alpha_n & 0 \end{bmatrix}$$

Riktningsvektorer definieras alltså oberoende av origo.

Byte av ram

Givet två ramar ($\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3$, P_0) och ($\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$, Q_0), kan vi skriva

$$\mathbf{u}_1 = \gamma_{11}\mathbf{v}_1 + \gamma_{12}\mathbf{v}_2 + \gamma_{13}\mathbf{v}_3$$
$$\mathbf{u}_2 = \gamma_{21}\mathbf{v}_1 + \gamma_{22}\mathbf{v}_2 + \gamma_{23}\mathbf{v}_3$$
$$\mathbf{u}_3 = \gamma_{31}\mathbf{v}_1 + \gamma_{32}\mathbf{v}_2 + \gamma_{33}\mathbf{v}_3$$
$$Q_0 = \gamma_{41}\mathbf{v}_1 + \gamma_{42}\mathbf{v}_2 + \gamma_{43}\mathbf{v}_3 + P_0$$



Byte av ram

$$\mathbf{u}_1 = \gamma_{11}\mathbf{v}_1 + \gamma_{12}\mathbf{v}_2 + \gamma_{13}\mathbf{v}_3$$
$$\mathbf{u}_2 = \gamma_{21}\mathbf{v}_1 + \gamma_{22}\mathbf{v}_2 + \gamma_{23}\mathbf{v}_3$$
$$\mathbf{u}_3 = \gamma_{31}\mathbf{v}_1 + \gamma_{32}\mathbf{v}_2 + \gamma_{33}\mathbf{v}_3$$
$$Q_0 = \gamma_{41}\mathbf{v}_1 + \gamma_{42}\mathbf{v}_2 + \gamma_{43}\mathbf{v}_3 + P_0$$

Det är smidigare att skriva rambytet på matrisform:

$$\begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ Q_0 \end{pmatrix} = \underbrace{\begin{pmatrix} \gamma_{11} & \gamma_{12} & \gamma_{13} & 0 \\ \gamma_{21} & \gamma_{22} & \gamma_{23} & 0 \\ \gamma_{31} & \gamma_{32} & \gamma_{33} & 0 \\ \gamma_{41} & \gamma_{42} & \gamma_{43} & 1 \end{pmatrix}}_{\mathbf{M}} \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ P_0 \end{pmatrix}$$

Punkter och rambyte

Ta nu en punkt P och definiera den i två olika ramar:

$P = \alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \alpha_3\mathbf{v}_3 + P_0$ i den första ramen

$P = \beta_1\mathbf{u}_1 + \beta_2\mathbf{u}_2 + \beta_3\mathbf{u}_3 + Q_0$ i den andra

Hur är α- och β-värdena relaterade?

$$P = \begin{bmatrix}\beta_1 & \beta_2 & \beta_3 & 1\end{bmatrix}\begin{pmatrix}\mathbf{u}_1\\ \mathbf{u}_2\\ \mathbf{u}_3\\ Q_0\end{pmatrix} = \mathbf{b}^T\mathbf{M}\begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\\ P_0\end{pmatrix} = \begin{bmatrix}\alpha_1 & \alpha_2 & \alpha_3 & 1\end{bmatrix}\begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\\ P_0\end{pmatrix} = \mathbf{a}^T\begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\\ P_0\end{pmatrix}$$

(enl. ovan)

(föregående bild)

Punkter och rambyte

$$\mathbf{b}^T\mathbf{M}\begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\\ \mathrm{P}_0\end{pmatrix} = \mathbf{a}^T\begin{pmatrix}\mathbf{v}_1\\ \mathbf{v}_2\\ \mathbf{v}_3\\ \mathrm{P}_0\end{pmatrix}$$

$$\Leftrightarrow$$

$$\mathbf{b}^T\mathbf{M} = \mathbf{a}^T$$

$$\Leftrightarrow$$

$$\mathbf{a} = \mathbf{M}^T\mathbf{b}$$

och omvänt:

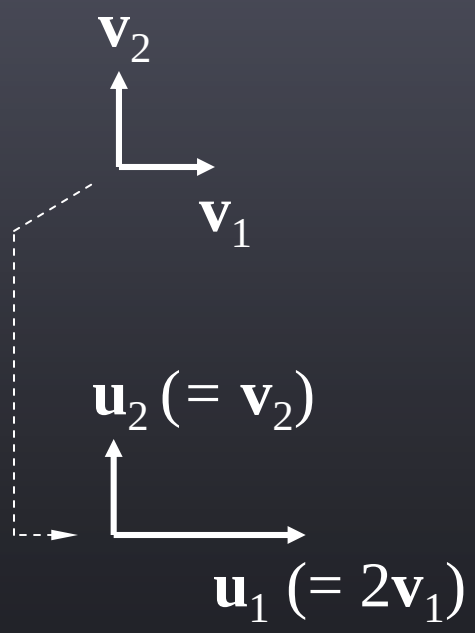
$$\mathbf{b} = (\mathbf{M}^T)^{-1}\mathbf{a}$$

D.v.s. P's koordinater i ram 2 fås om man multiplicerar koordinaterna för ram 1 med $(\mathbf{M}^T)^{-1}$.

Exempel

Antag att vi arbetar i 2D och har två ramar
$(\mathbf{v}_1, \mathbf{v}_2, P_0)$ och $(\mathbf{u}_1, \mathbf{u}_2, Q_0)$
och att ramarna är relaterade till varandra enligt följande:

$$\mathbf{u}_1 = 2\mathbf{v}_1$$
$$\mathbf{u}_2 = \mathbf{v}_2$$
$$Q_0 = P_0$$



$$\mathbf{M} = \mathbf{M}^T = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{M}^{-1} = (\mathbf{M}^T)^{-1} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

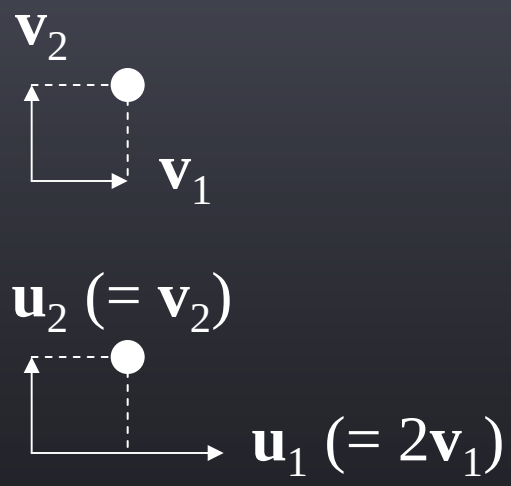
Exempel

Tag nu punkten $\mathbf{a} = (1 \;\; 1 \;\; 1)^T$ i den första ramen.
Vilken är dess motsvarighet i den andra ramen?

$$\mathbf{b} = (\mathbf{M}^T)^{-1}\mathbf{a}$$

ger oss

$$\mathbf{b} = \begin{pmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{a} = \begin{bmatrix} 0.5 & 1 & 1 \end{bmatrix}^T$$

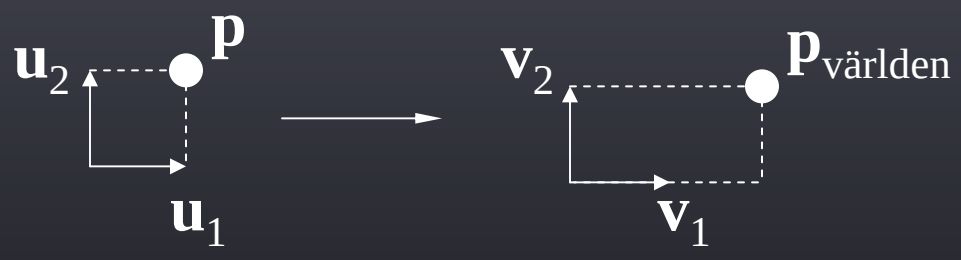


Vi kan också se rambytet som en **transformation**.
Exempel:

Antag att "världen" är ram 1
och att vi gör ett temporärt rambyte till ram 2.

Vi specificerar nu punkten $\mathbf{p} = (1 \;\; 1 \;\; 1)^T$ i ram 2.

I ram 1 blir den $\mathbf{p}_{\text{världen}} = \mathbf{M}^T\mathbf{p} = (2 \;\; 1 \;\; 1)^T$



Med andra ord är byte av ram och transformation ekvivalenta begrepp!

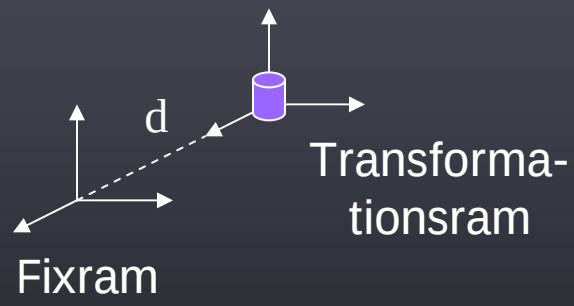
Fixram och transformationsram

Ofta jobbar man med en "fixram" och en "transformationsram". Fixramen är fast och transformationsramen tillåts variera. Exempel:

Våra koordinater

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 - d \\ 1 \end{pmatrix}$$

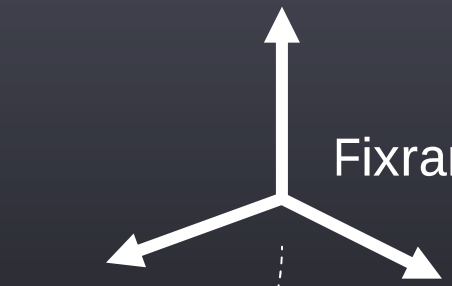
$\mathbf{M}^T$ — Punkt i transf.-ramen — Motsv. punkt i fixramen



Fixram och transformationsram

Specificera en modell i "sitt eget" koordinatsystem, **objektkoordinater**.
"Placera ut" modellen i "världen" genom att definiera en transformationsram.
Multiplicera objektkoordinaterna med $\mathbf{M}^{\mathrm{T}}$ för att få modellens koordinater i fixramen.

Groda-modell i dess objektkoordinatsystem



Transformations-ram

Matriser

Translation

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{M}^T \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Skalning

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

Matriser

Rotation kring z-axeln

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{R}_z(\theta) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation kring x-axeln

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{R}_x(\theta) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation kring y-axeln

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \mathbf{R}_y(\theta) \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Multipla transformationer

Vi kan sätta samman transformationer genom att multiplicera matriserna:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{S}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{TS}}$$

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{TS}} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ s\alpha_2 \\ \alpha_3 - d \\ 1 \end{pmatrix}$$

Multipla transformationer

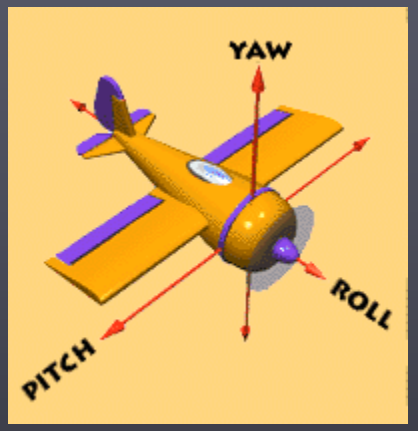
Observera att
ordningen på multiplikationen kan spela roll!

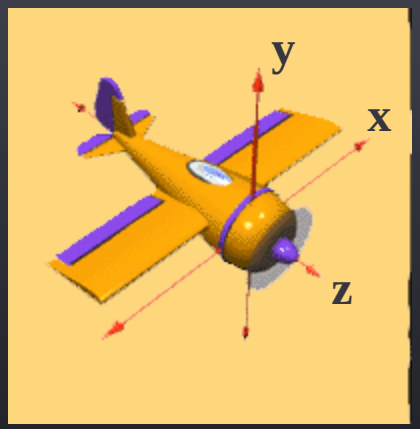
$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{S}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{TS}}$$

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{S}} \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & s & -sd \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{ST}}$$

Att specificera orientering

- Vi börjar med att pröva yaw-pitch-roll.
- Antag att flygplanet är modellerat så det initialt "pekar" längs **z**-axeln.
- Vi roterar först kring **z**-axeln (roll), sedan kring **y**-axeln (yaw), sist kring **x**-axeln (pitch).
- Kallas för att arbeta med **Euler angles** eller **Eulermatriser**.

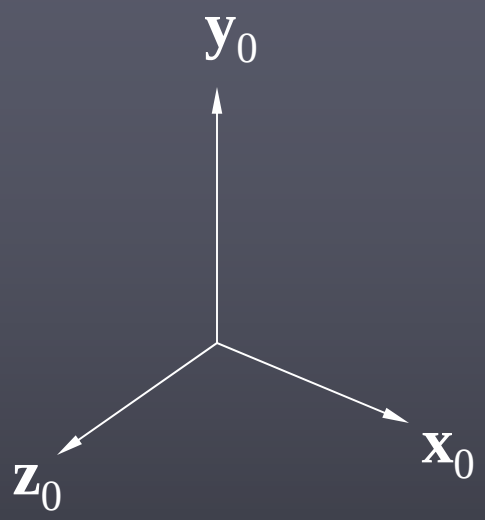


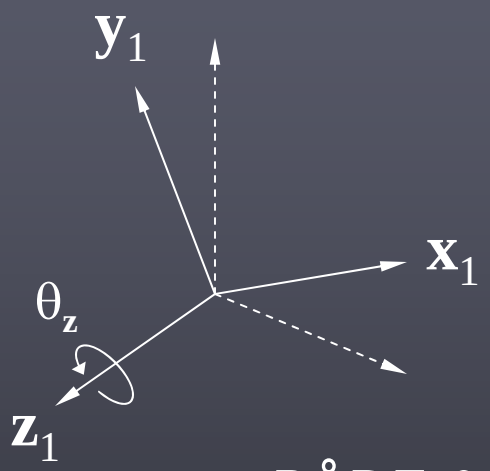


Gimbal lock

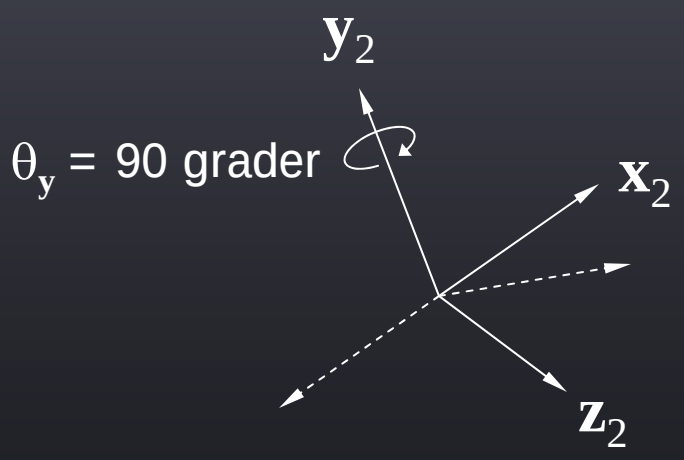
- Om vi roterar en av axlarna så den sammanfaller med en av de andra kan vi hamna i ett läge där vi tappar information!
- Situationen kallas för **Gimbal lock**.
- Beror på att rotationerna alltid görs i samma ordning.

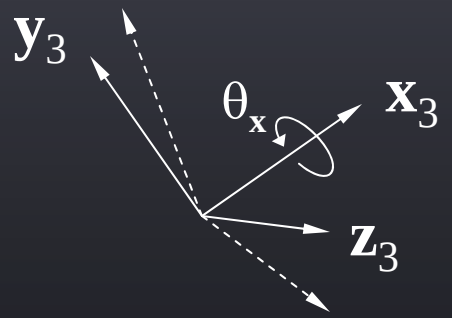
Gimbal lock: exempel





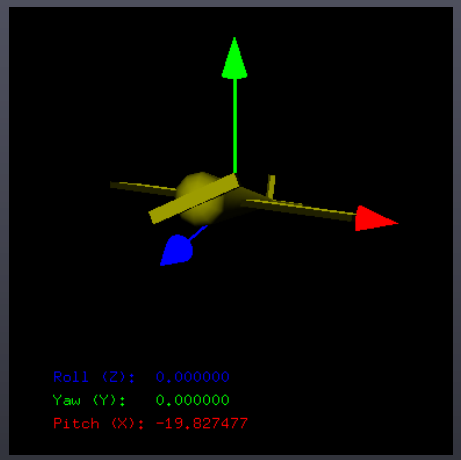
BÅDE θ_x och θ_z beskriver "pitch"!

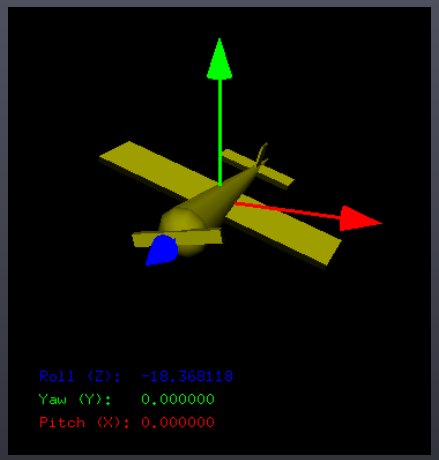




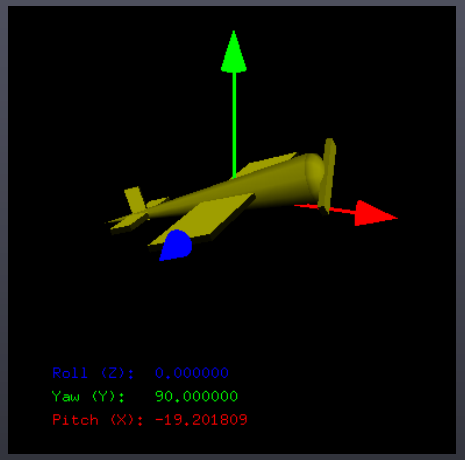
Gimbal lock: exempel

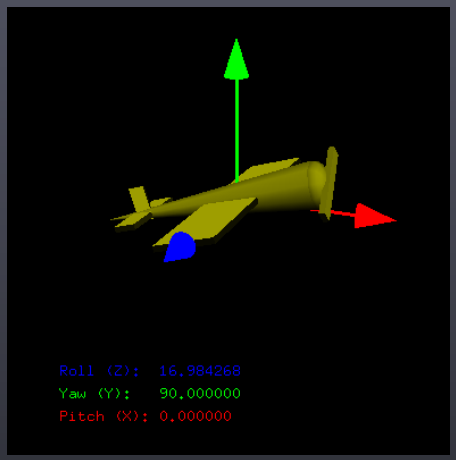
Exempel: Rotationsordningen är Z, Y, X.





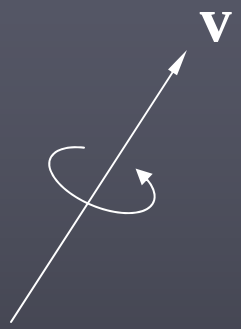
Vinkeln för y är 0:
rotationer kring x och z
funkar som förväntat.





Om vinkeln för y är 90 grader:
rotationen kring x och z
går inte att särskilja!

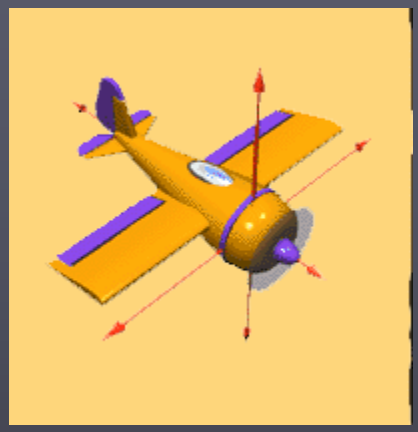
Quaternions



Antag att vi kan beskriva
rotation kring en godtycklig
vektor **v**.

Man kan visa att vilken
orientering som helst kan beskrivas som
en (1 st) rotation kring en vektor.

Quaternions

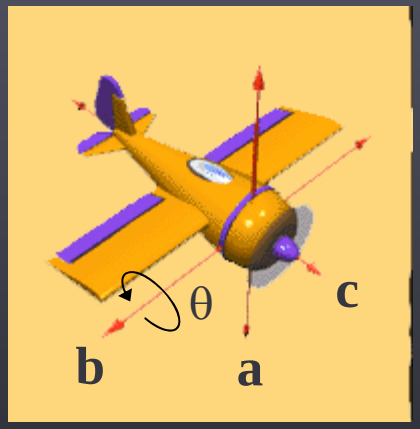


Antag att vi kan skriva flygplanets orientering i förhållande till "världens" koordinatsystem som q.

Idén är nu att utgå från q och beskriva förändringen i orientering över en bildruta som rotationer kring flygplanets tre **nuvarande** koordinataxlar.

Quaternions

Antag att vi drar flygplanets
styrspak mot oss,
planets nos ska rotera uppåt θ radianer.

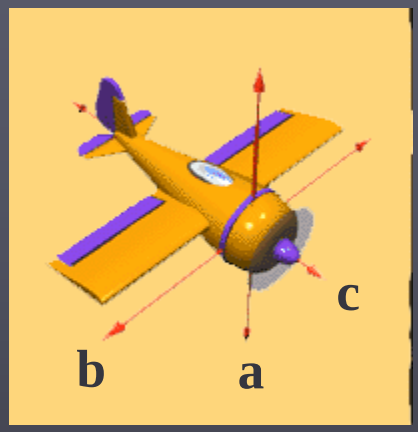


Vi har q som tar oss från
"världskoordinatsystemet" till
planets nuvarande orientering.

Vi tillverkar nu en $q_{\mathbf{b}}$ som
roterar θ radianer kring axeln $\mathbf{b}$.

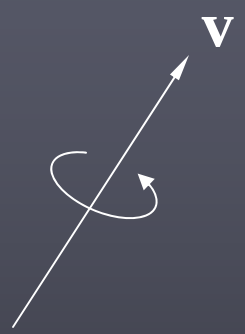
Vi gör sedan
$q = q$ "+" $q_{\mathbf{b}}$

Quaternions



- Rotera från "världskoordinatsystemet" till nuvarande koordinatsystem **abc**
- Uppdatera "heading": rotera kring **a** och få **ab'c'**
- Uppdatera "pitch": rotera kring **b'** och få **a'b'c"**
- Uppdatera "roll": rotera kring **c"** och få **a"b"c"**
- Sätt q till resultatet

Quaternions



En matematisk struktur som kan beskriva rotation kring en godtycklig vektor är **kvaternioner** (**quaternions**).

En kvaternion q definieras så här:

$$q = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q})$$
$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$
$$\boldsymbol{q} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$$

Observera att $\mathbf{q}$ INTE är rotationsaxeln $\mathbf{v}$!

Quaternions

Lite kvaternionalgebra: tag två kvaternioner a och b:

$$a = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q})$$
$$b = (p_0, p_1, p_2, p_3) = (p_0, \mathbf{p})$$

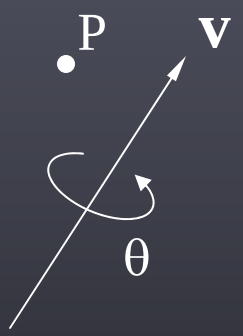
Addition och multiplikation:

$$a + b = (p_0 + q_0, \mathbf{p} + \mathbf{q})$$
$$ab = (p_0 q_0 - \mathbf{p} \cdot \mathbf{q}, q_0 \mathbf{p} + p_0 \mathbf{q} + \mathbf{p} \times \mathbf{q})$$

Norm och invers:

$$|a|^2 = q_0^2 + \mathbf{q} \cdot \mathbf{q}$$
$$a^{-1} = (1 / |a|^2)(q_0, -\mathbf{q})$$

Quaternions



Tag en punkt P i 3D. Skapa kvaternionen

$$p = (0, \mathrm{P})$$

Välj rotationsaxeln $\mathbf{v}$ och normera den.
Låt rotationsvinkeln vara θ.

Skapa nu kvaternionen r enligt

$$r = (\cos(\theta/2), \sin(\theta/2)\mathbf{v})$$
$$r^{-1} = (\cos(\theta/2), -\sin(\theta/2)\mathbf{v})$$

Man kan visa att

$$p' = rpr^{-1}$$

är resultatet av att rotera punkten P
θ radianer kring vektorn $\mathbf{v}$.

Om a och b är kvaternioner som representerar rotation enl. ovan gäller också att

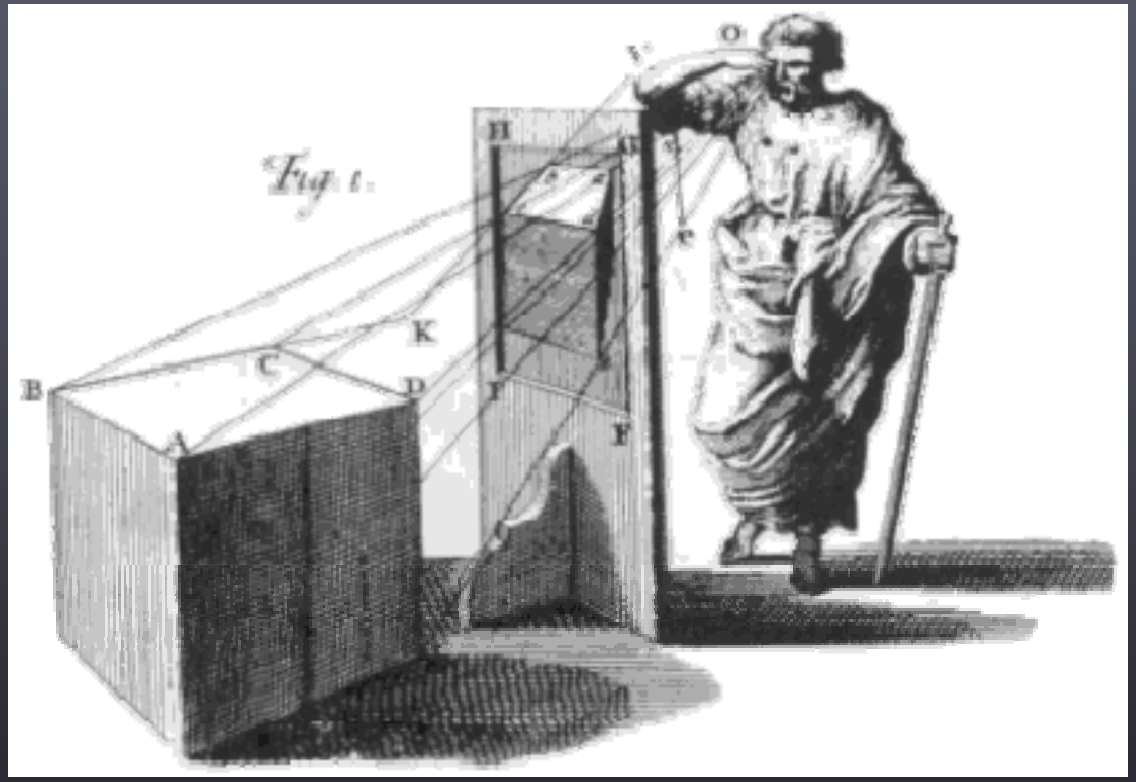
$$q = ab$$

är resultatet av att kombinera rotationerna (först a, sedan b).

Quaternions

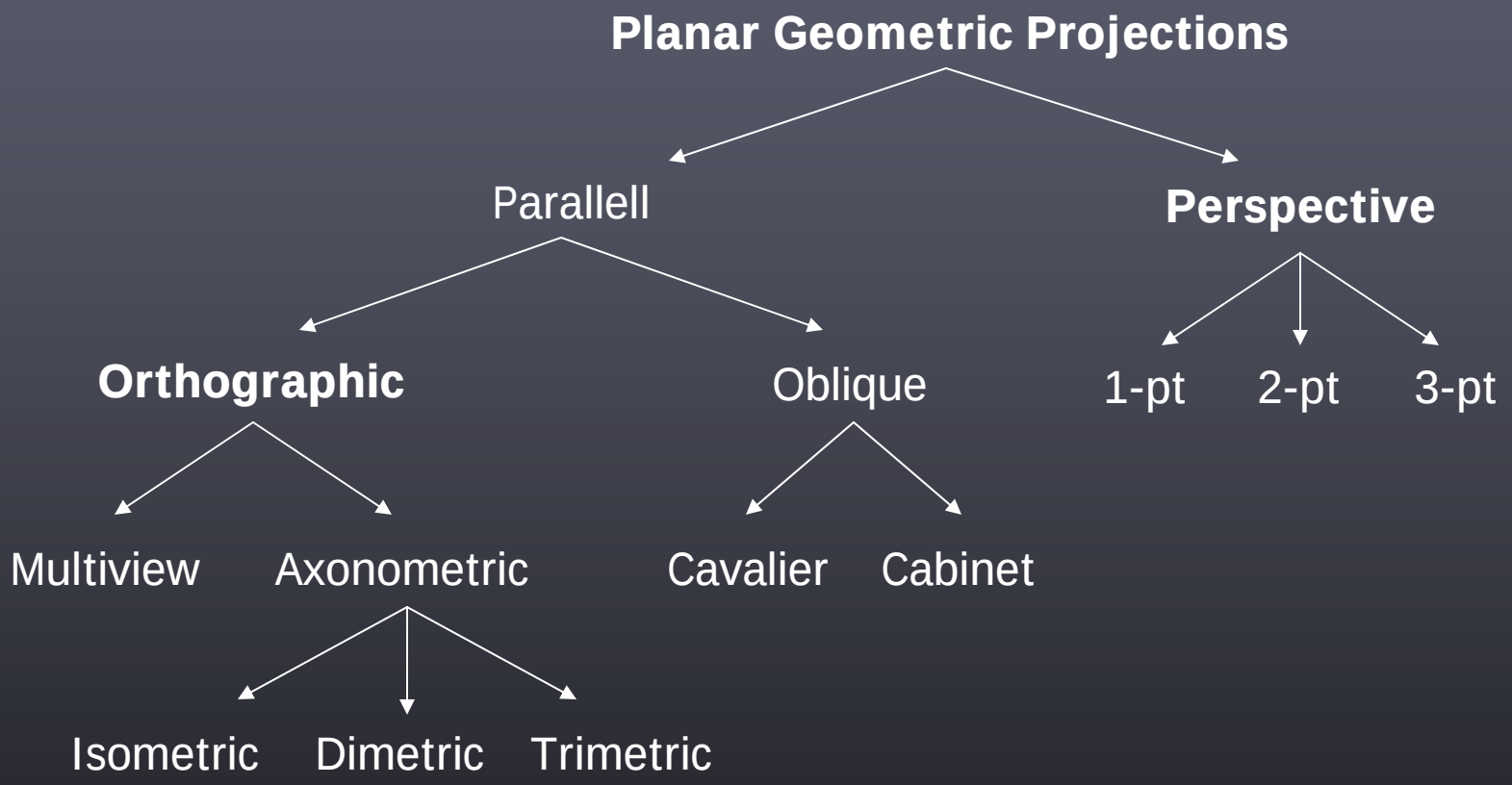
- Det går enkelt att konvertera en kvaternion till en rotationsmatris (och tvärt om), se kursbunten eller kursboken!
- Det kan löna sig att lagra orienteringar som kvaternioner: 4 flyttal ist. för 3x3 = 9 (för en rotationsmatris)
- Kvaternioner gör det enkelt att interpolera mellan orienteringar, se kursbunten!

Projektioner



Brook Taylor, **New Principles of Linear Perspective**, 1719.

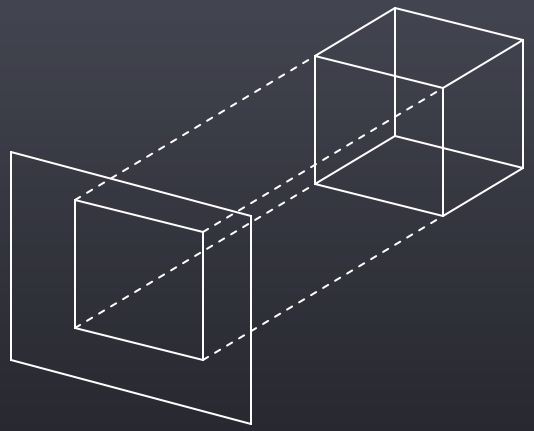
Projektioner

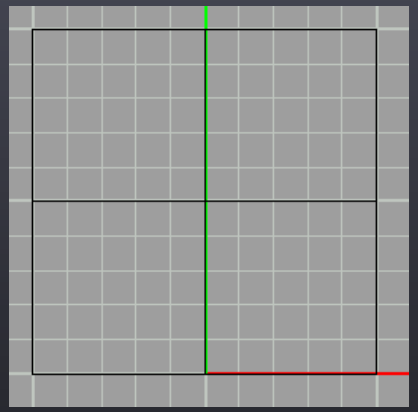


Alla projektionerna finns beskrivna i kursbunten.

Ortografisk projektion

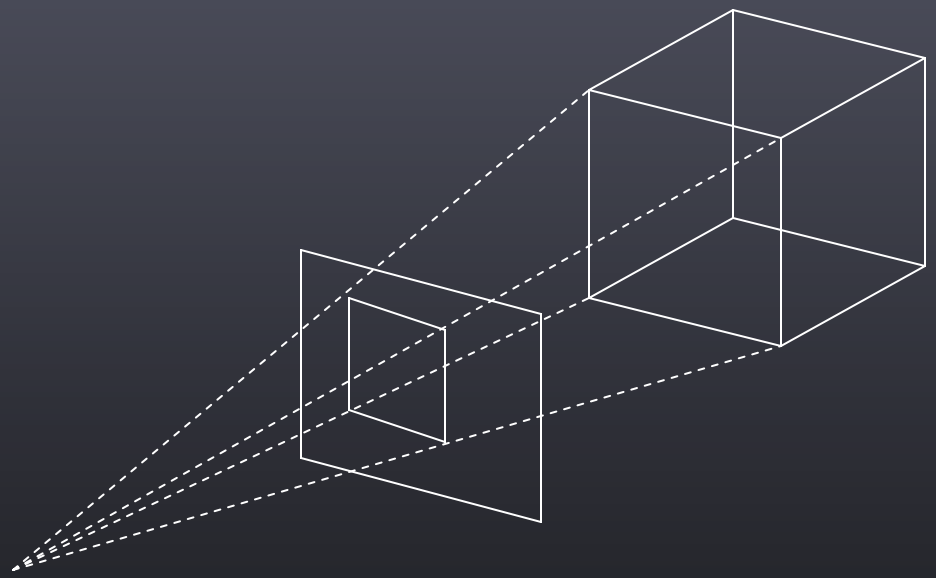
Projektorerna är vinkelräta mot projektionsytan.
Avstånd och vinklar bevaras i projektionen.

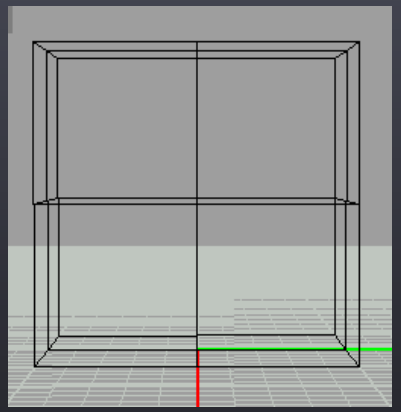




Perspektivprojektion (enpunkts)

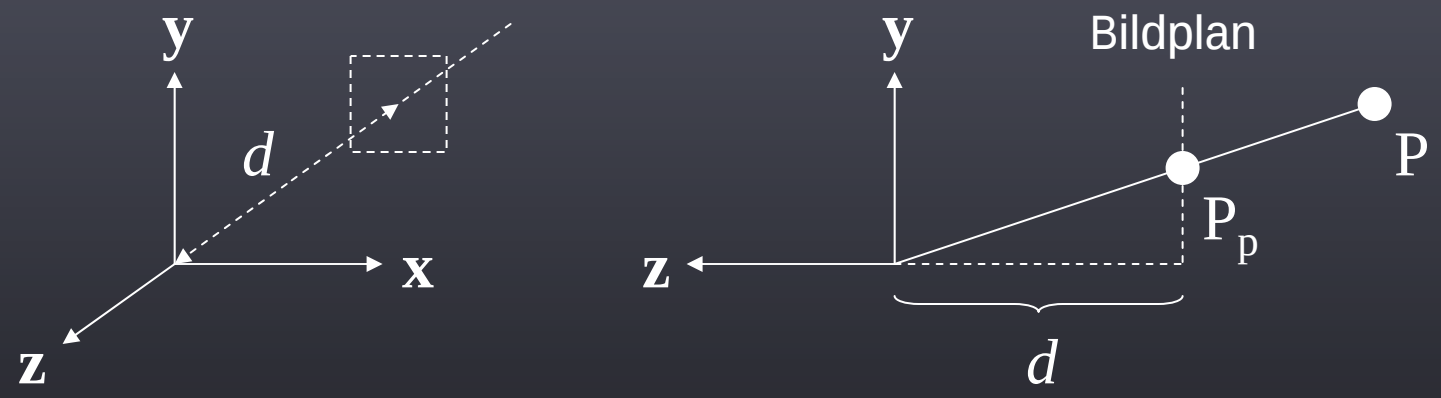
Storlek minskar med avstånd från projektionsytan.
Avstånd och vinklar bevaras inte i projektionen.





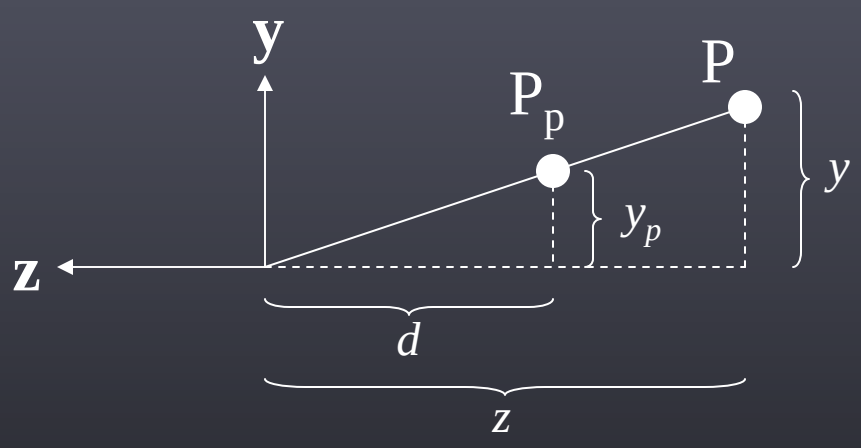
Perspektivprojektion

Antag att vi projicerar punkten P på ett bildplan som ligger d enheter från origo och är parallellt med x-y-planet:



Perspektivprojektion

Likformiga trianglar:



$$P = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$$

$$P_p = \begin{bmatrix} x_p & y_p & d & 1 \end{bmatrix}^T$$

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

$$\Leftrightarrow$$

$$y_p = \frac{yd}{z} = \frac{y}{(z/d)}$$

Analogt för x_p.
Så den projicerade punkten blir:

$$P_p = \begin{bmatrix} x/(z/d) & y/(z/d) & d & 1 \end{bmatrix}^T$$

Perspektivprojektion

Vi vill kunna skriva perspektivprojektionen som en matris $\mathbf{P}$.
För att kunna göra det måste vi införa en ny operation.

Om vi transformerar en punkt

$$\mathrm{P} = \begin{bmatrix} x & y & z & w \end{bmatrix}^{\mathrm{T}}$$

så att w blir skilt från 1 ser vi till att alltid
dividera med w innan vi fortsätter, d.v.s.

$$\mathrm{P} = \begin{bmatrix} x/w & y/w & z/w & 1 \end{bmatrix}^{\mathrm{T}}$$

Perspektivprojektion

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix}$$

Om $z/d \neq 1$ dividerar vi alla komponenterna med z/d innan vi använder punkten så att vi får

$$\begin{pmatrix} x/(z/d) \\ y/(z/d) \\ d \\ 1 \end{pmatrix}$$

Detta kallas **perspektivdivision** i OpenGL.

Perspektivprojektion

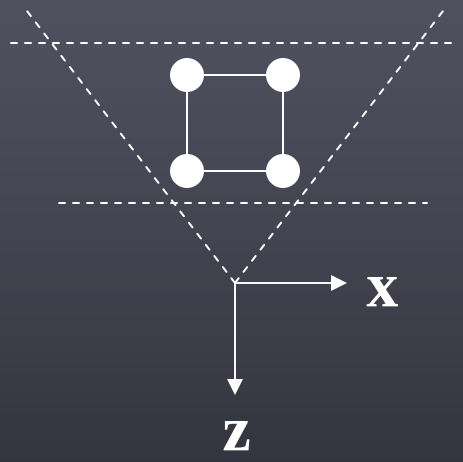
I OpenGL är projektionerna lite mer komplicerade än vad som sagts här. Anledningen är att man vill kunna klippa grafiska primitiver mot den volym som ”syns” på projektionsplanet på ett och samma sätt oavsett vilken sorts projektion man har.
I OpenGL klipps alla polygoner mot volymen

$$-1 \leq x \leq 1$$
$$-1 \leq y \leq 1$$
$$-1 \leq z \leq 1$$

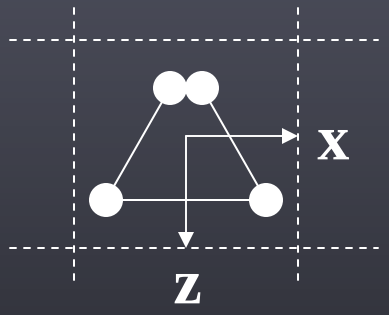
Man måste m.a.o. konstruera en projektionsmatris som svarar mot detta.

Perspektivprojektion

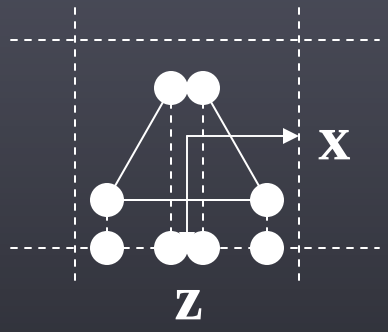
Matrisen är en kombination av två operatorer:



1) Distortion



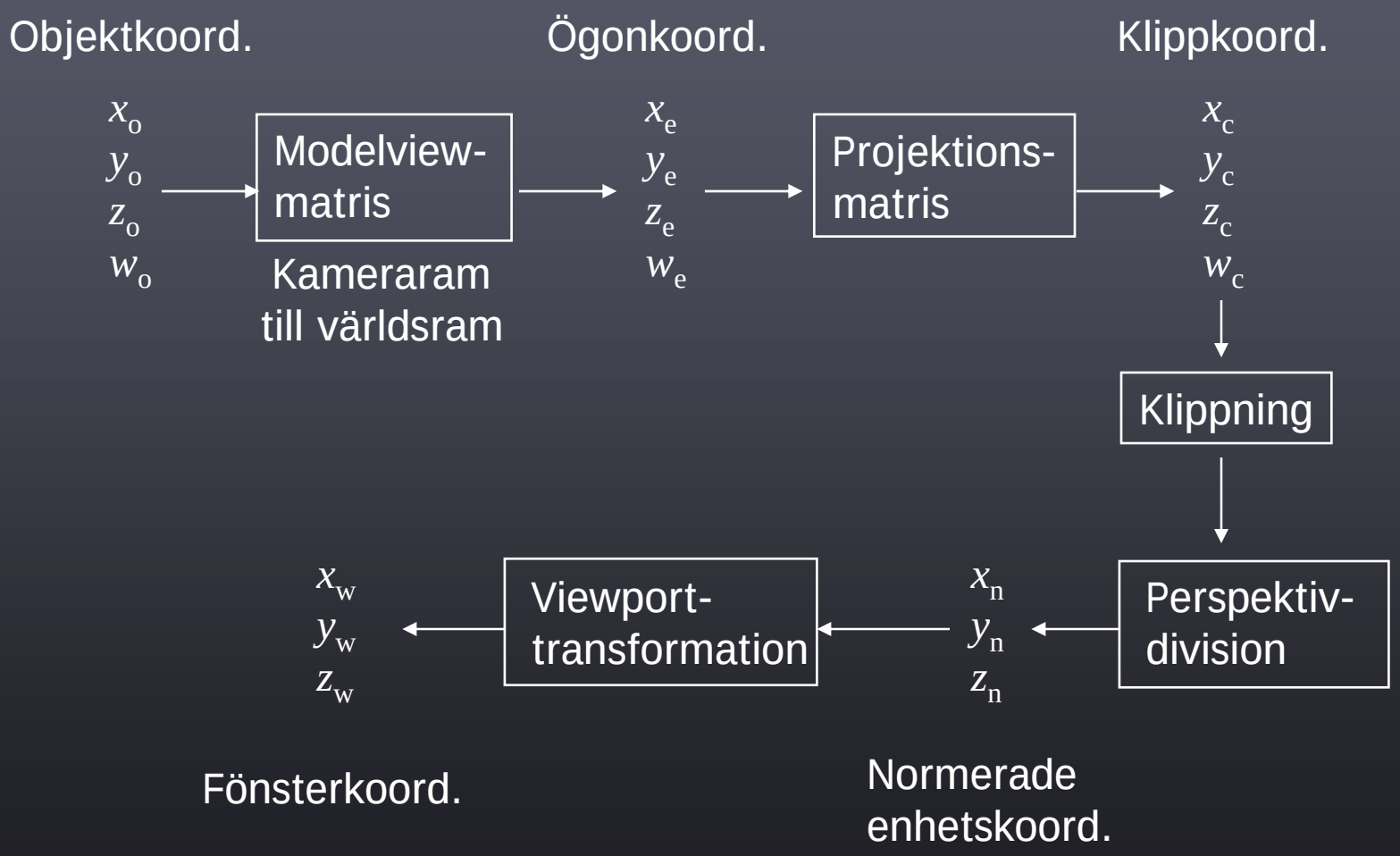
2) Ortografisk projektion



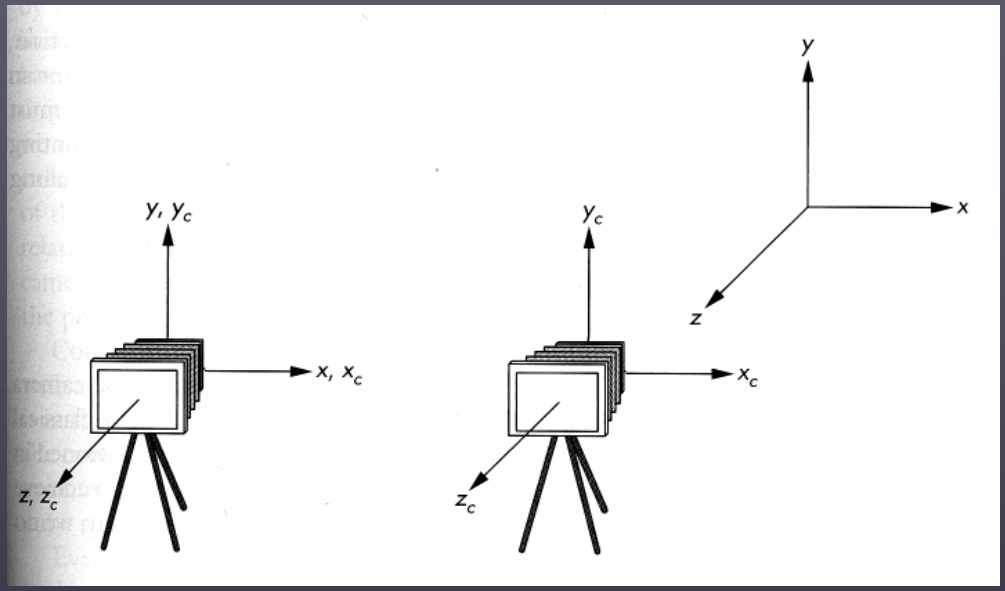
Klippning görs i detta läge.

Detaljer finns i kursboken.

Transformationer i OpenGL



Att positionera kameran

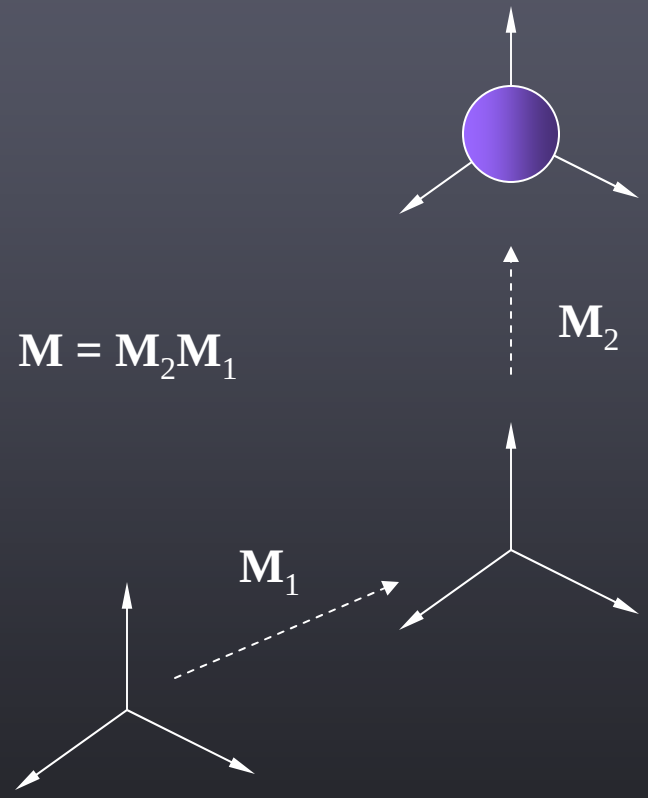


Antingen tänker man att kameran flyttas i förhållande världens ram, **eller** att världsramen flyttas så att den hamnar framför kameran. Operationerna är inverser av varandra.

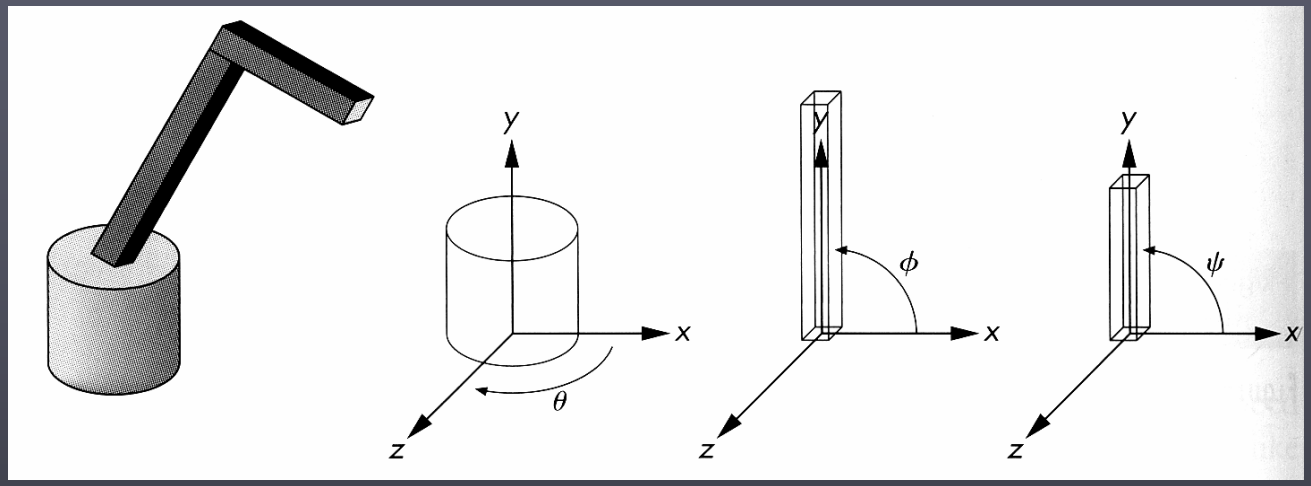
I OpenGL är kameraramen fix så rent matematiskt gäller det senare alternativet. Men det finns en funktion som hjälper en att ”flytta världen” så att det motsvarar att flytta kameran.

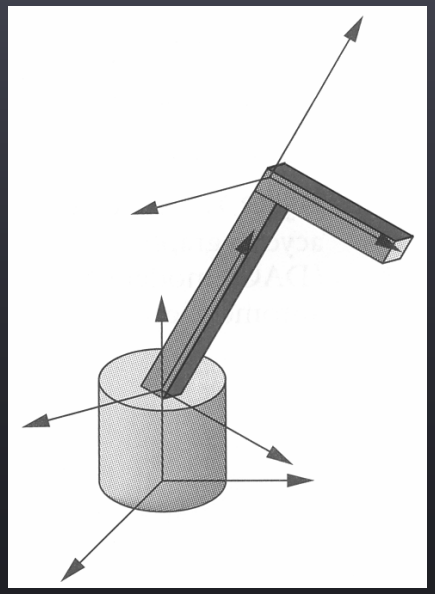
Kombinationer av transformationer

- På motsvarande sätt som i 2D.
- Varje transformation motsvarar ett byte av ram.
- Så successiva transformationer är också successiva byten av koordinatsystem!



Hierarkiska modeller





Hierarkiska modeller i OpenGL