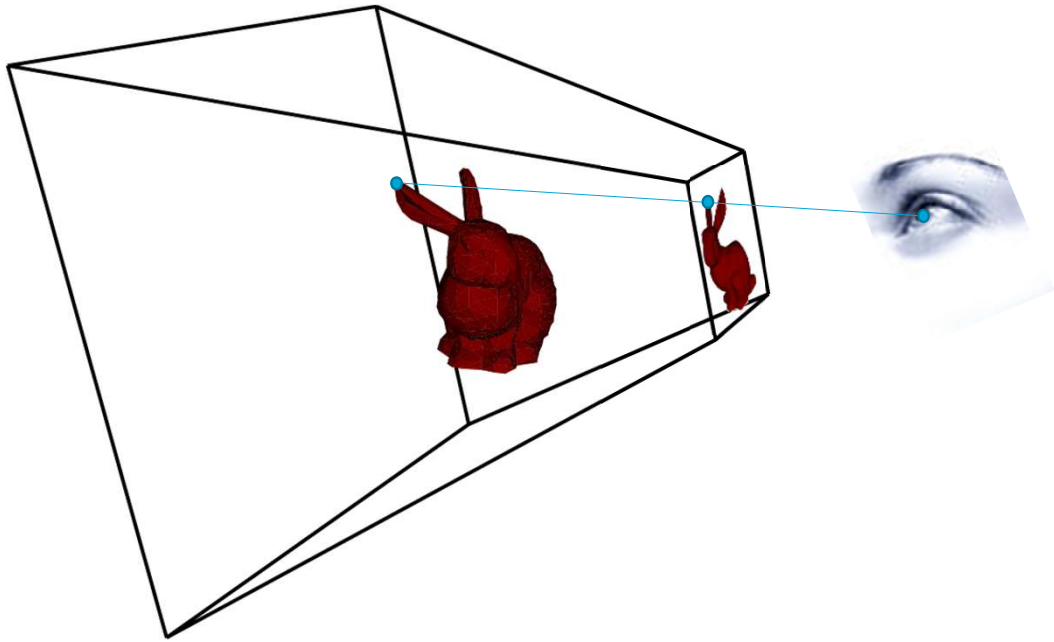
3D Computer Graphics and Animation

Geometry Pipeline
Enter the Matrix

Elmar Eisemann

Delft University of Technology

Perspective

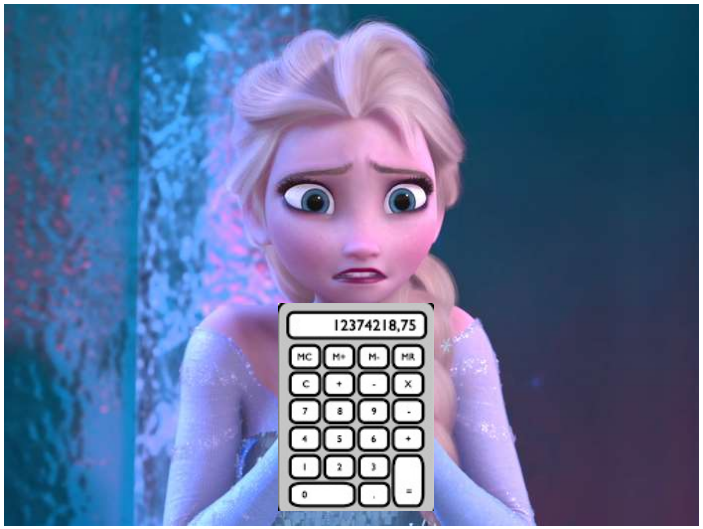


Today

- How to build a virtual camera?
- How can projective geometry help us?
  What are homogeneous coordinates?
- How to transform objects using projective geometry?
- Next time: Full projective camera model
  Complex transformations

Do you wanna build a camera...?

...mathematically? ☺

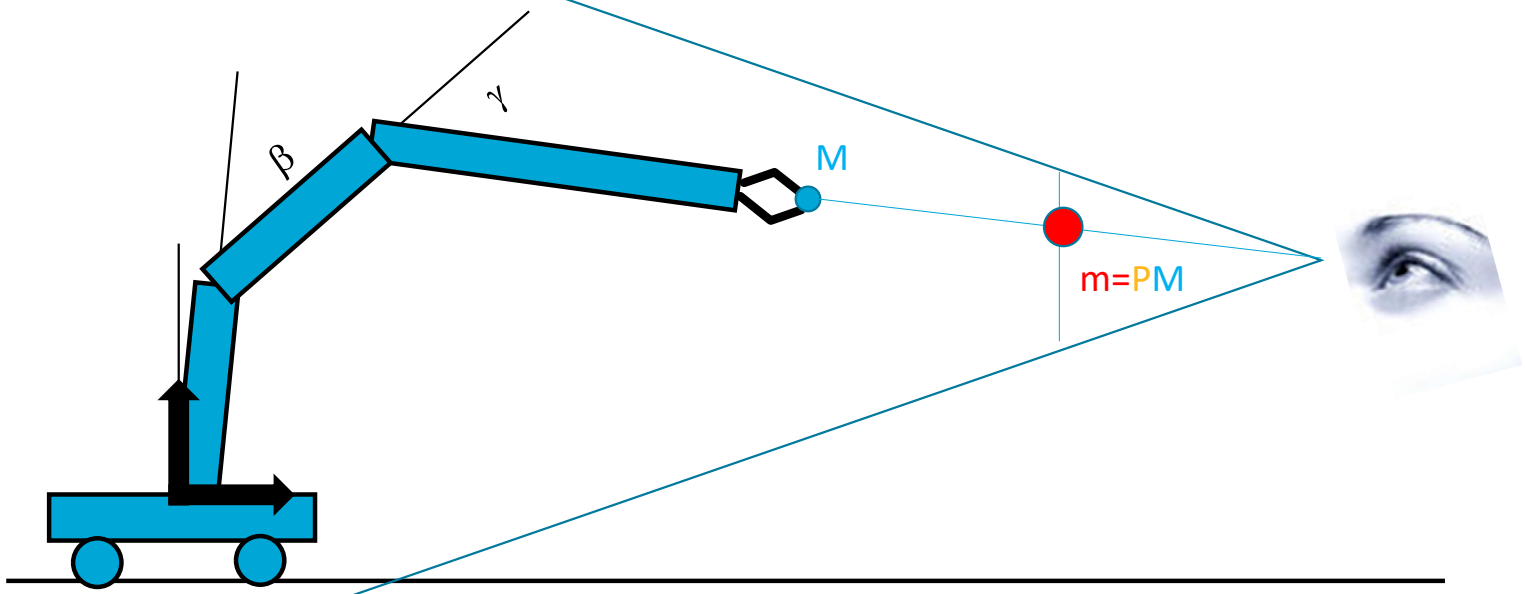


What does this mean?

- Given a 3D point, we should find a function that results in the point's projection in the photo.

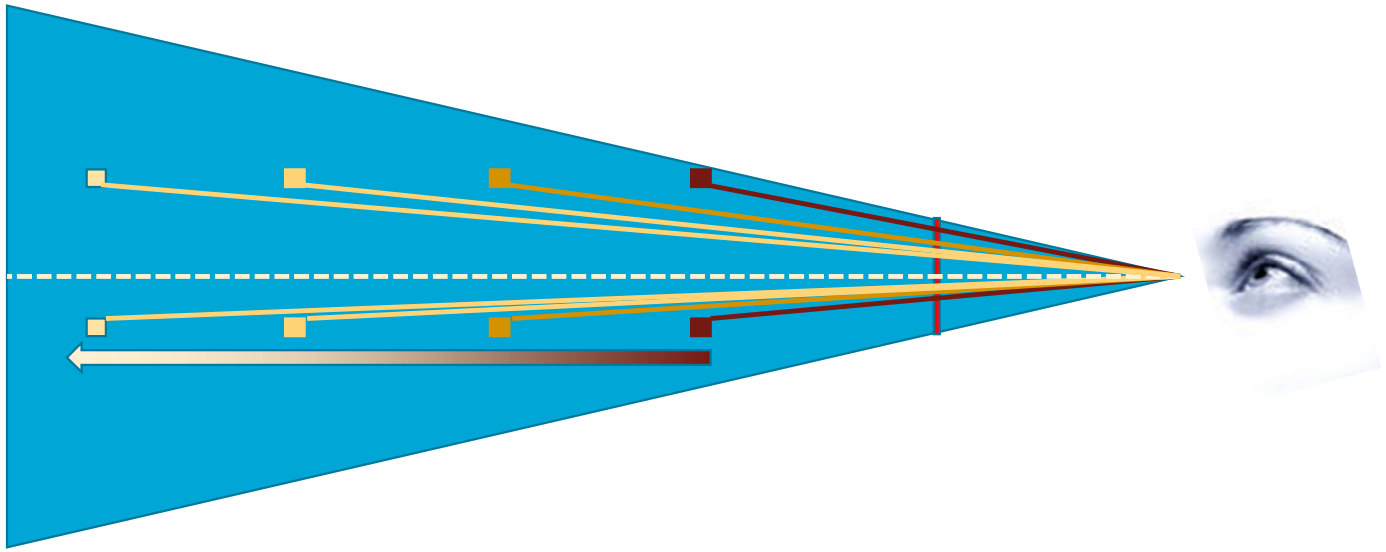
We will see today:

We can find matrix P such that the projected pixel position m of point M is PM.



How to draw with accurate perspective?

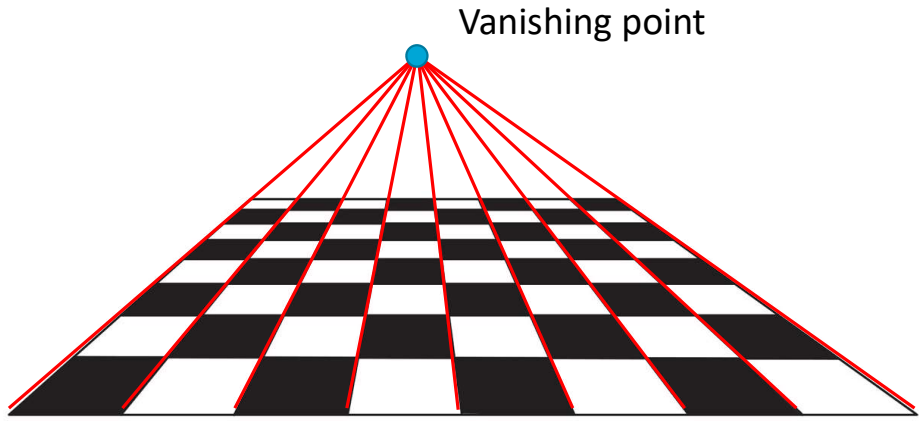
Linear Perspective



Linear Perspective

Linear Perspective

- Central Perspective



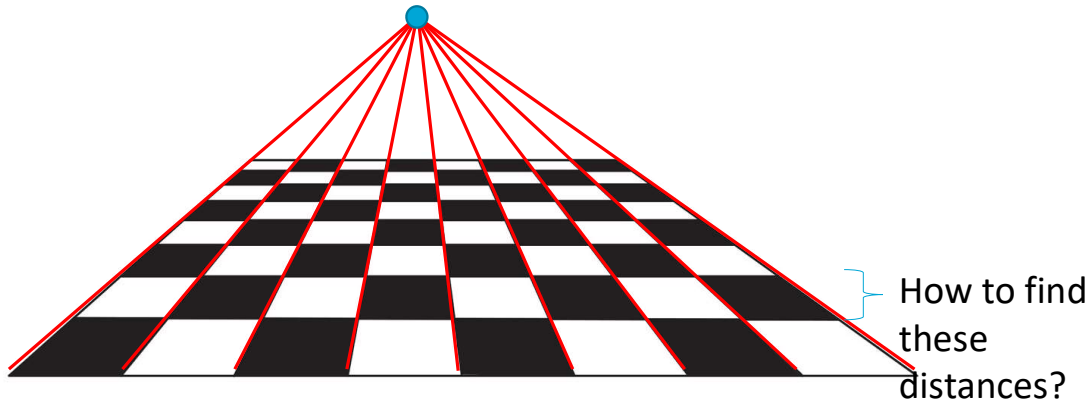
Johannes Vermeer, *The Milkmaid,* Rijksmuseum

Detail of an x-ray of *The Milkmaid*

(courtesy Rijksmuseum)

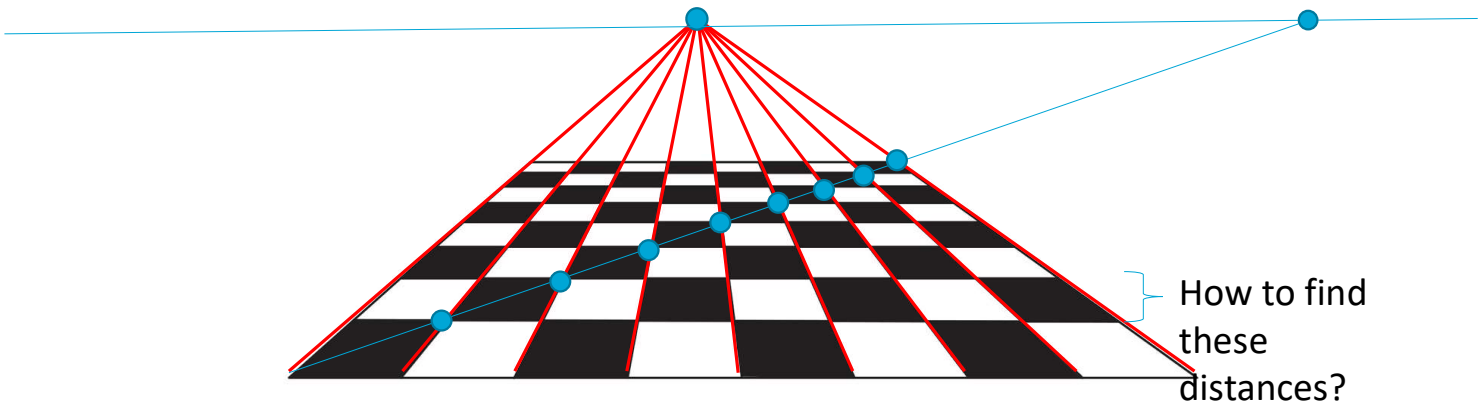
Linear Perspective

- Central Perspective



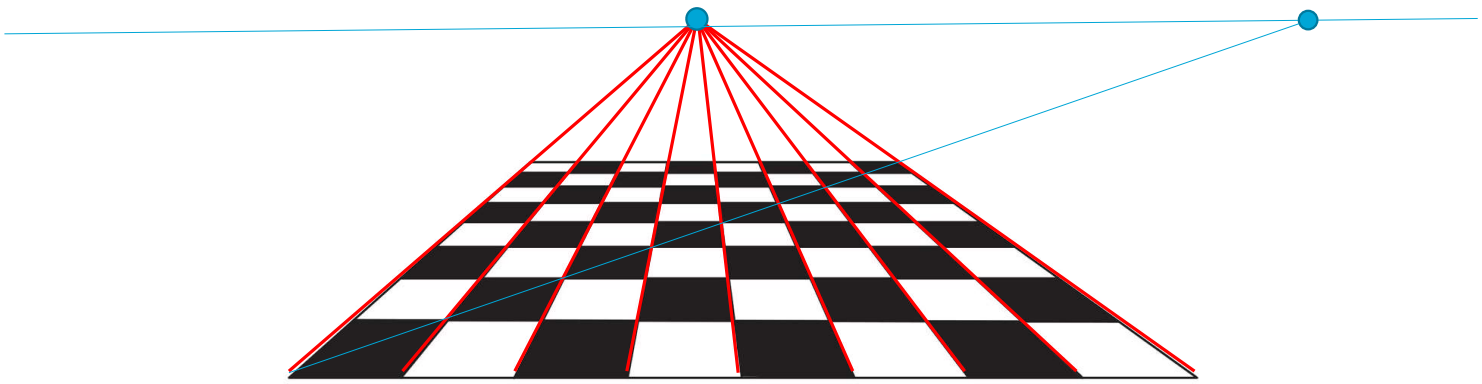
Linear Perspective

- Central Perspective



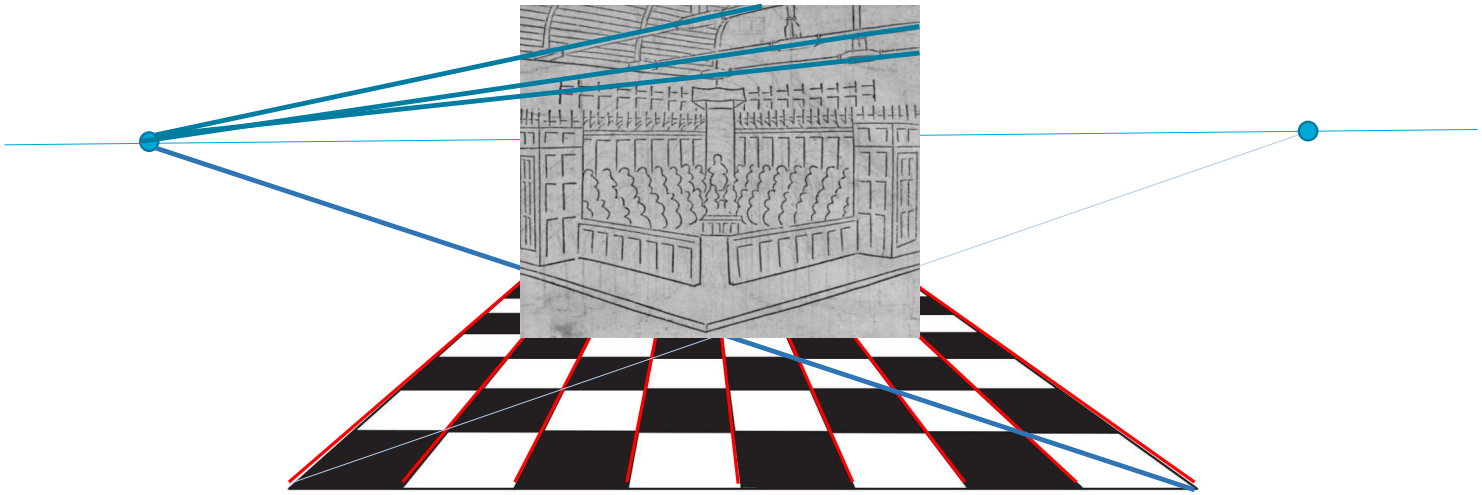
Linear Perspective

- Central Perspective

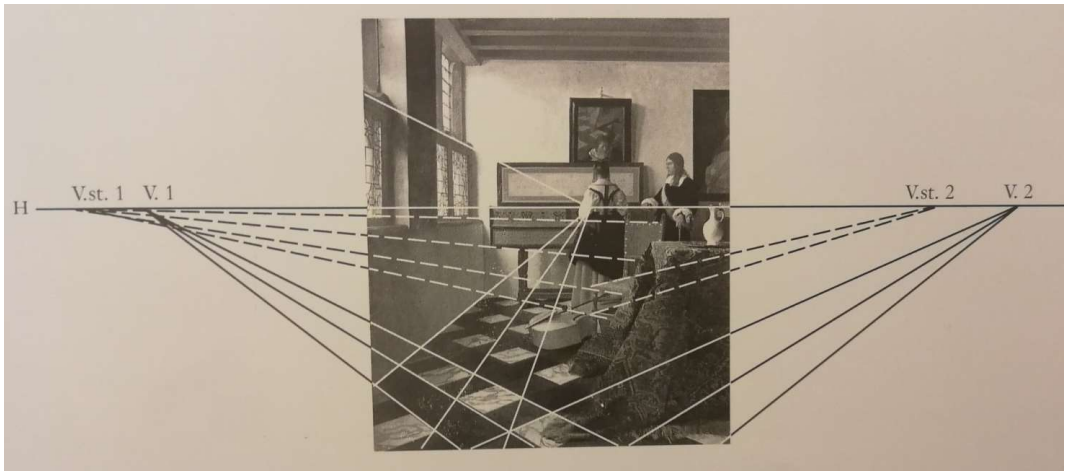


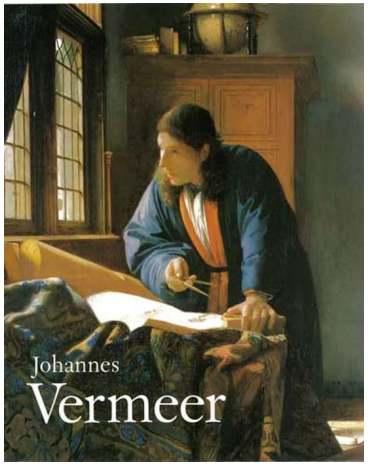
Linear Perspective

- Two Point Perspective



Johannes Vermeer, *The Music Lesson*, ca. 1662-1665, Royal Collection Trust



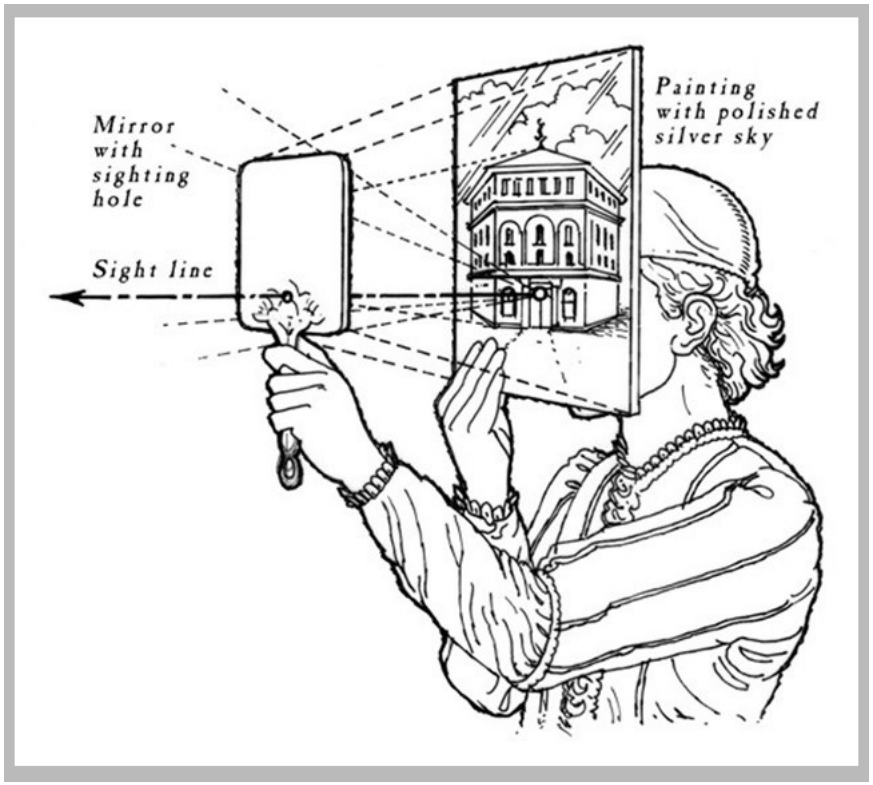


How to find the correct location
for these vanishing points?

Linear Perspective

$$\left\{ \emptyset \colon \bar{F} - -1 < \int_{-1}^{\infty} \bigcup_{A \in A} \cos^{-1}\left(\|A\| \right) \, d\Xi'' \right\}$$
$$= \prod_{Z_{\mathscr{J}}=1}^{0} \mathfrak{t}\left(- - \infty, \ldots, \frac{1}{1} \right) - \cdots \pm 2X$$
$$\neq \bigcap_{\mathscr{F}=1}^{2} \mathbf{x}\left(d, \ldots, 1 \times e \right) \vee S\left(r_{i,\mathscr{X}}\left(\Phi' \right), \ldots, L(\mathscr{W}_{G,\Psi}) \cap -1 \right)$$
$$\geq \frac{\overline{\varphi^{(\epsilon)}}}{-O} - \log\left(-c_Z(\mathfrak{s}') \right).$$

Linear Perspective



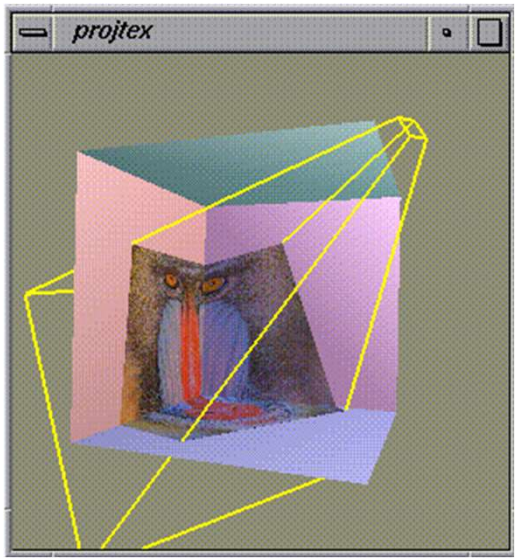
Filippo Brunelleschi – 1377–1446

How to convince in a modern age?

Spotlight Technology

- Create virtual image for camera

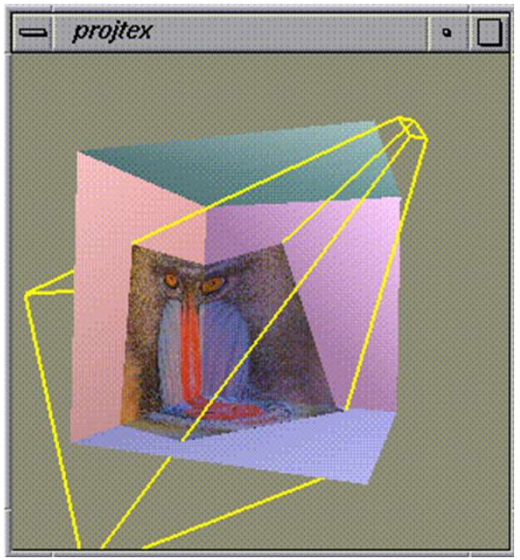
- Project on screen and film with real camera



Spotlight Technology

- **Create virtual image for virtual camera**

- Project on screen and film with real camera



Linear Perspective

$$\begin{aligned}
&\left\{ \emptyset \colon \bar{F} - -1 < \int_{-1}^{\infty} \bigcup_{A \in A} \cos^{-1}\left(\|A\| \right) \, d\Xi'' \right\} \\
&= \prod_{Z_{\mathscr{J}}=1}^{0} \mathfrak{t}\left(- - \infty, \ldots, \frac{1}{1} \right) - \cdots \pm 2X \\
&\neq \bigcap_{\mathscr{F}=1}^{2} \mathbf{x}\left(d, \ldots, 1 \times e \right) \vee S\left(r_{i,\mathscr{X}}(\Phi'), \ldots, L(\mathscr{W}_{G,\Psi}) \cap -1 \right) \\
&\geq \frac{\overline{\varphi^{(\epsilon)}}}{-O} - \log\left(-c_Z(\mathfrak{s}') \right).
\end{aligned}$$

Linear Perspective

Mathematical description:

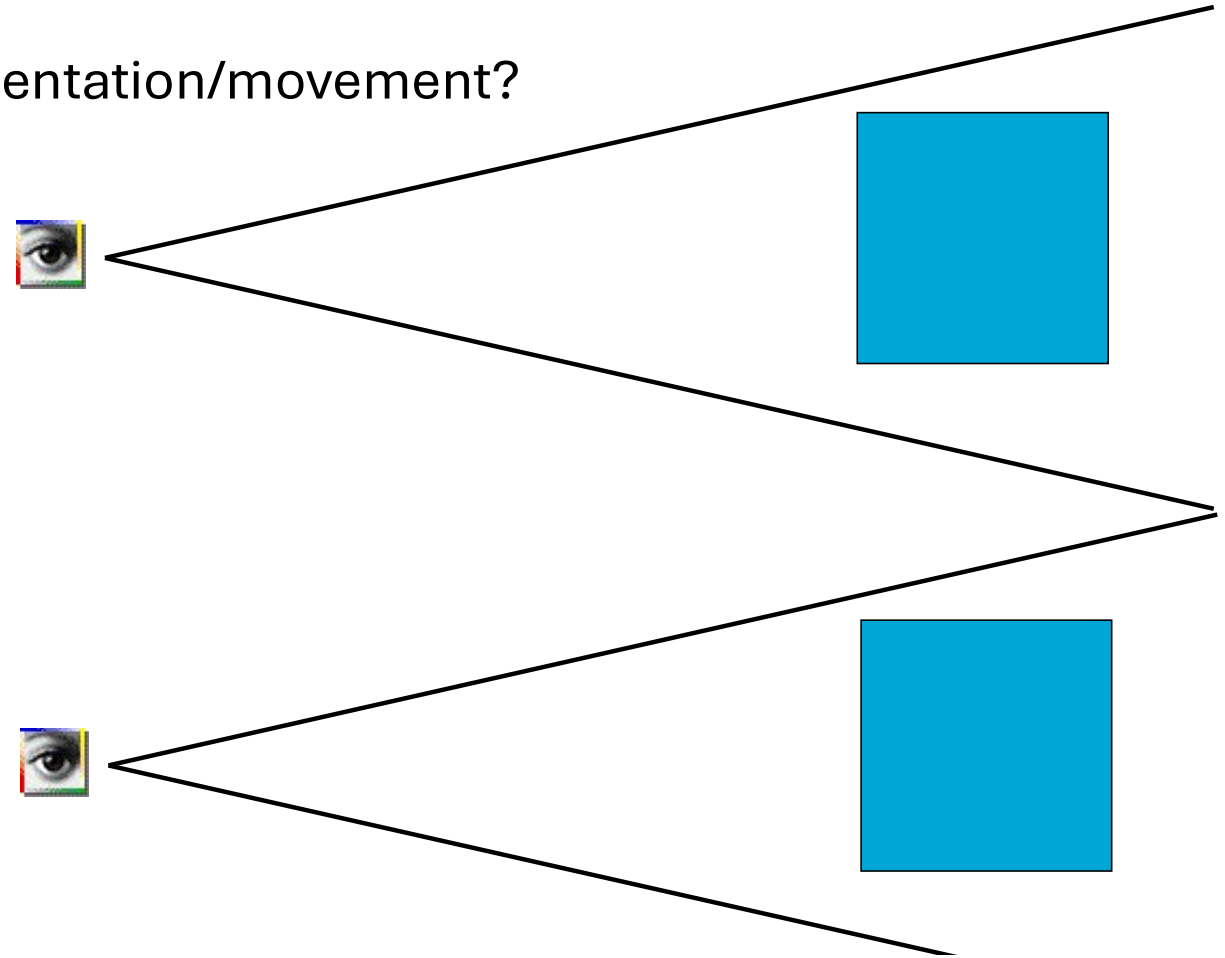
- **Easy** because with **projective geometry** everything is **linear** using **homogeneous coordinates**

$$P = \begin{pmatrix} a_x & 0 & t_0 \\ 0 & t_1 & t_2 \end{pmatrix}$$

Justified reaction when your teacher says something like this…

Virtual Camera Model

- Camera orientation/movement?

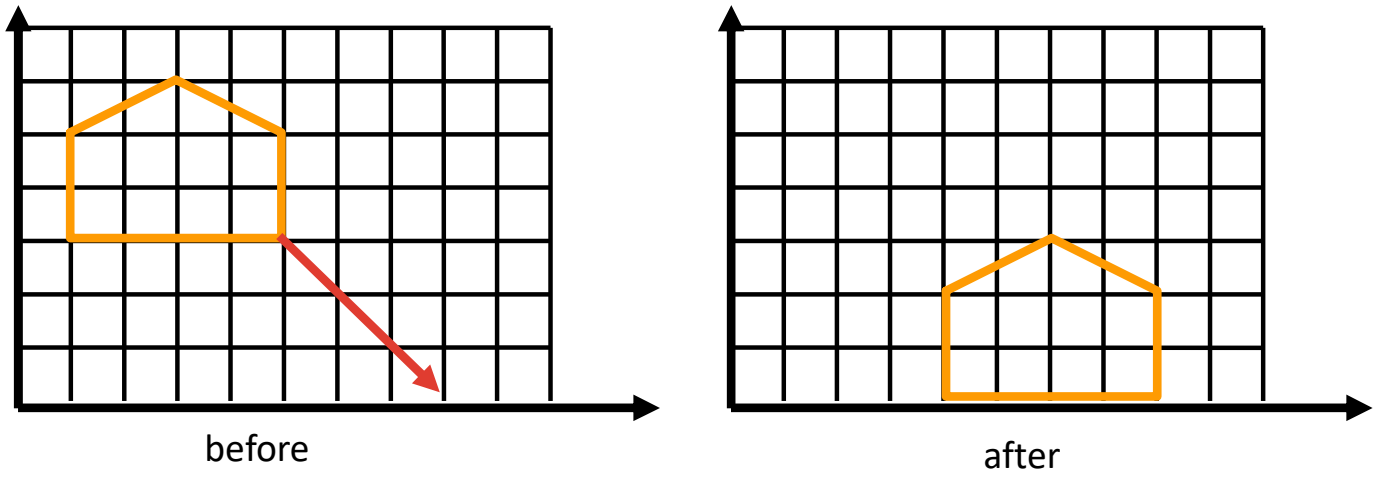


Movement and Orientation in 2D

- Starting in 2D
  - Simpler to represent

Translations

- Simple Modification :
  - $x' = x+t_x$
  - $y' = y+t_y$

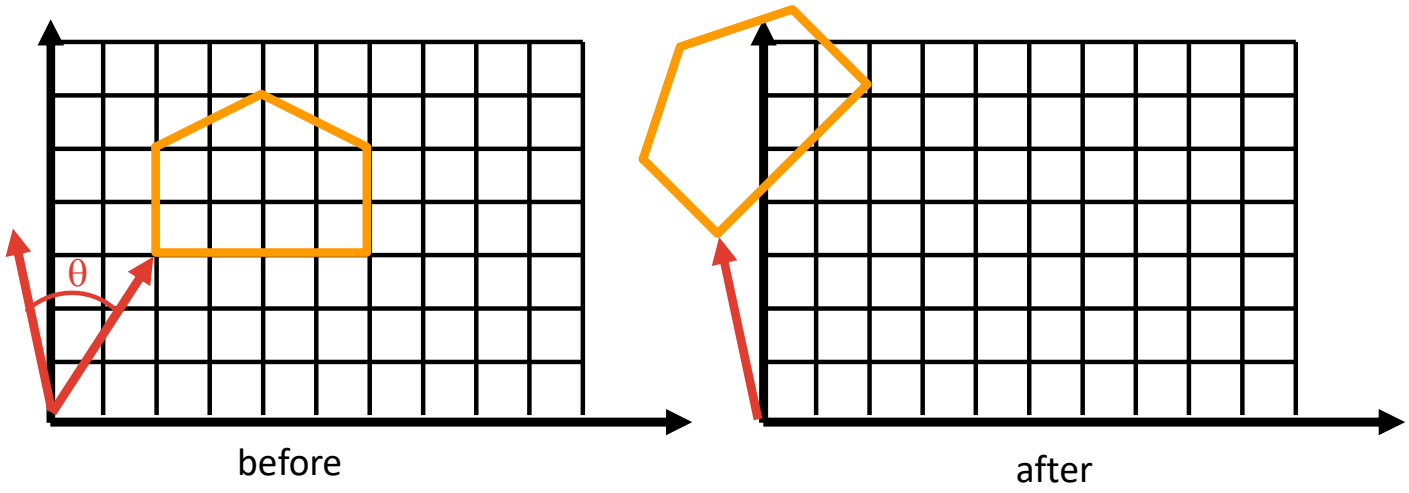


Translation

- Is a sum of vectors: P’ = P + T

Rotation

- Rotation in 2D :
  - $x' = \cos\theta\, x - \sin\theta\, y$
  - $y' = \sin\theta\, x + \cos\theta\, y$



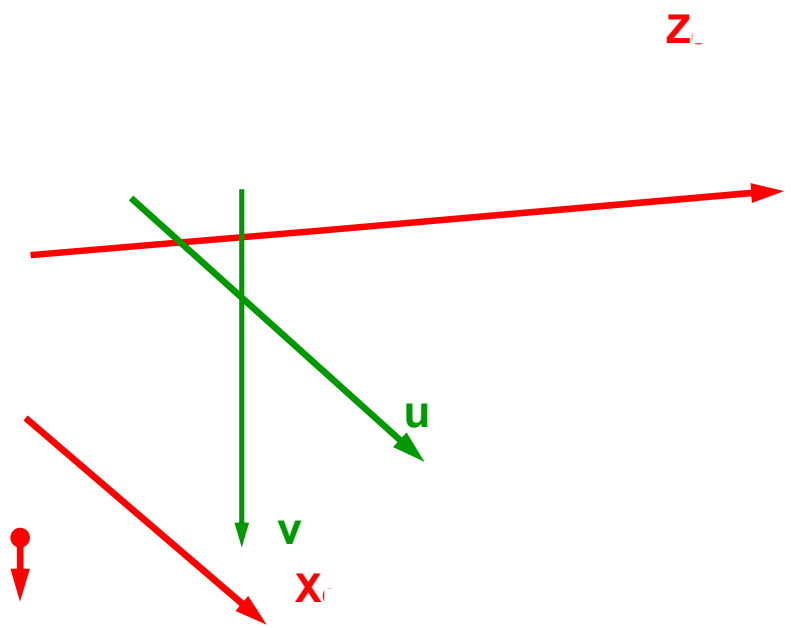
Rotation

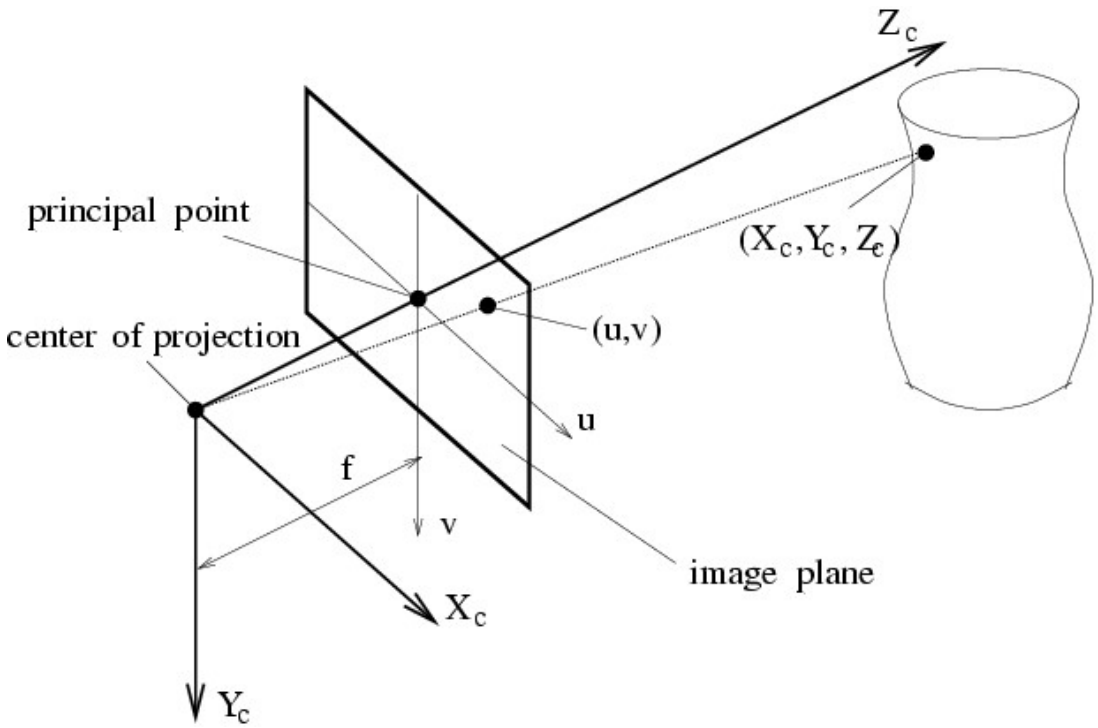
- Is a matrix multiplication:

$$P' = \mathbf{R}P$$

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

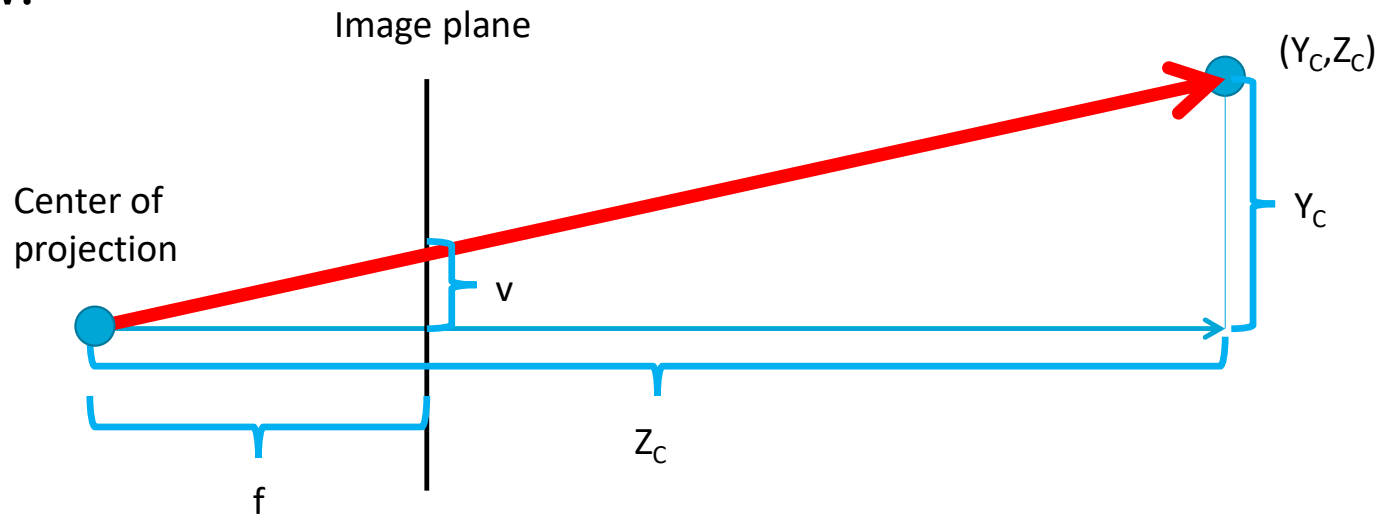
Perspective Projection




Z_c
principal point
(X_c, Y_c, Z_c)
center of projection
(u,v)
u
f
v
image plane
X_c
Y_c

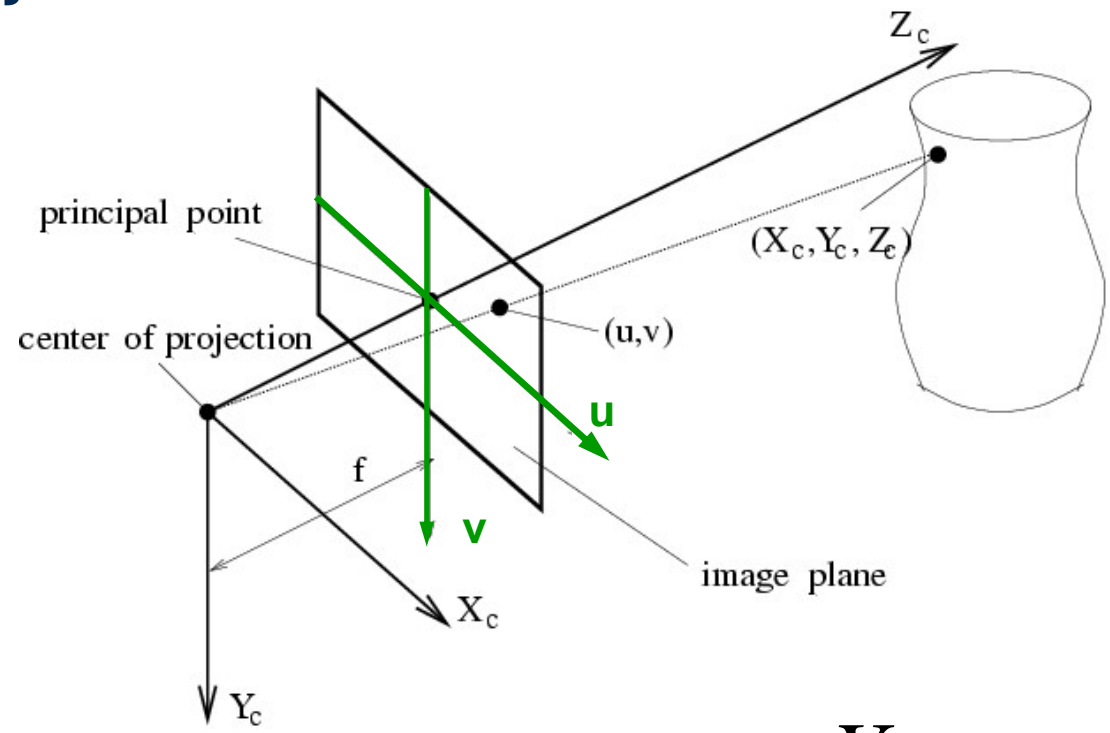
Perspective Projection

- sideview:



Similar triangles: $v / f = Y_C / Z_C$

Perspective Projection



$$u = f\frac{X_c}{Z_c} \qquad v = f\frac{Y_c}{Z_c}$$

Virtual Camera Model

Projecting a scene point with the camera:

- Apply camera position (adding an offset)
- Apply rotation (matrix multiplication)
- Apply projection (non-linear scaling)

Our camera starts to become complicated

and not well adapted to a hardware solution…

There has to be a better way…

What we want:

- Simple, concise notation
- Unification
  - Translation, rotation, projection

And if I am allowed to dream:

Do everything with a matrix

} Dreams can come true!

Is that really possible?

- Imagine a point X and a matrix T that describes a translation…

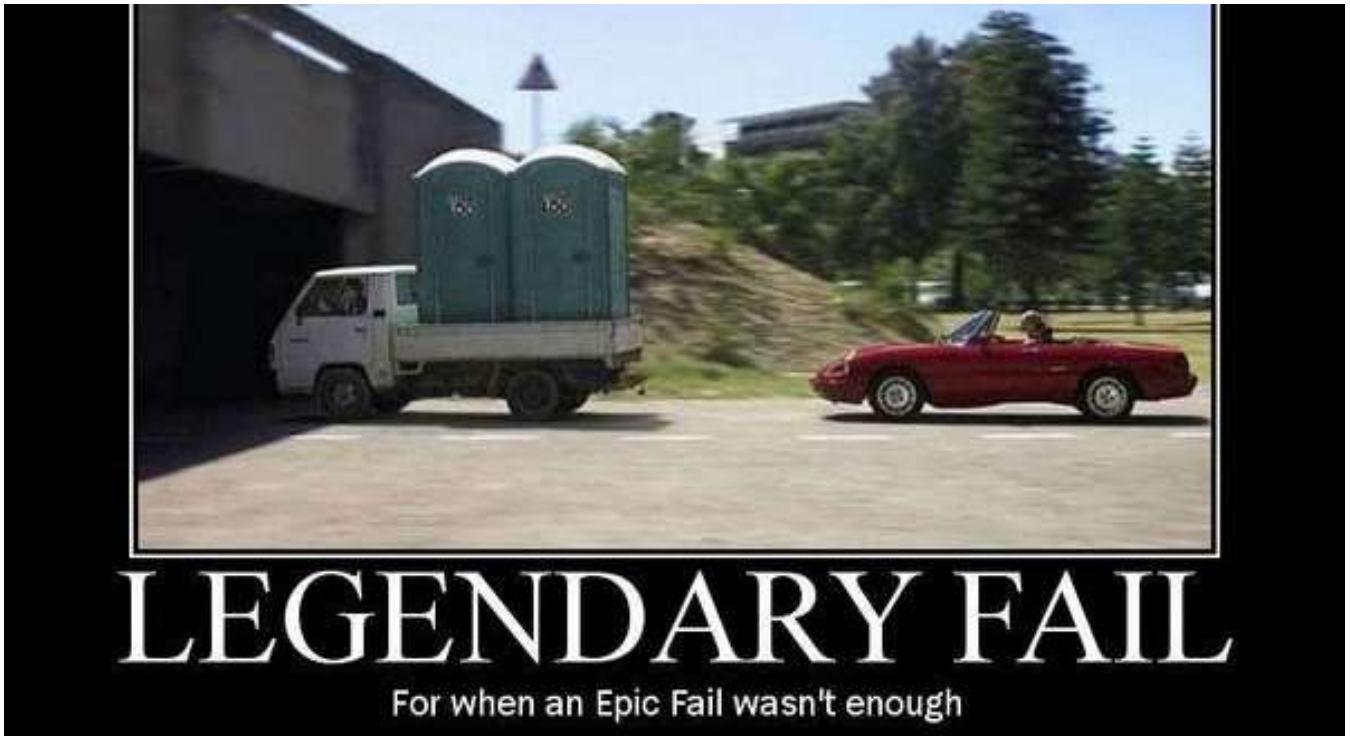
- T (X) = X+t
- T(X+0)

  =T(X)+T(0)

  =X+t+0+t = X+2t

Fail...

- Translations are not linear nor are projections, thus, cannot be represented by a matrix ...



- **Teacher: “Easy** because with **projective geometry** everything is **linear** using **homogeneous coordinates”**

Me when I was young

Today

- How to build a virtual camera?
- **How can projective geometry help us?**
  What are homogeneous coordinates?
- How to transform objects using projective geometry?
- Next time: Full projective camera model
  Complex transformations

Projective Geometry

With homogeneous coordinates…

- Translations and rotations are matrices

We will see:

- …a camera projection is a matrix

- … combining matrices allows us to define hierarchical-object dependencies
  (earth rotating around the sun, a hand moving with the arm…)


THE SWEET SMELL
OF EPIC SUCCESS

Homogenous Coordinates - Definition

- N-D *projective space* P^n is represented by N+1 coordinates, has no null vector, but a special equivalence relation:

Two points p, q are **equal**

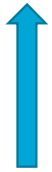
iff (if and only if)

exists a!=0 such that p*a = q

Examples in a 2D projective space P^2:
(2,2,2) = (3,3,3) = (4,4,4)=(π, π, π)
(2,2,2) $\neq$ (3,1,3)
(0,1,0) = (0,2,0)
(0,0,0) not part of the space

Homogeneous Coordinates

To embed a standard vector space R^n in an n-D projective space P^n, we can map:
$(x_0,x_1,\ldots x_{n-1})$ in R^n to $(x_0,x_1,\ldots x_{n-1}, 1)$ in P^n

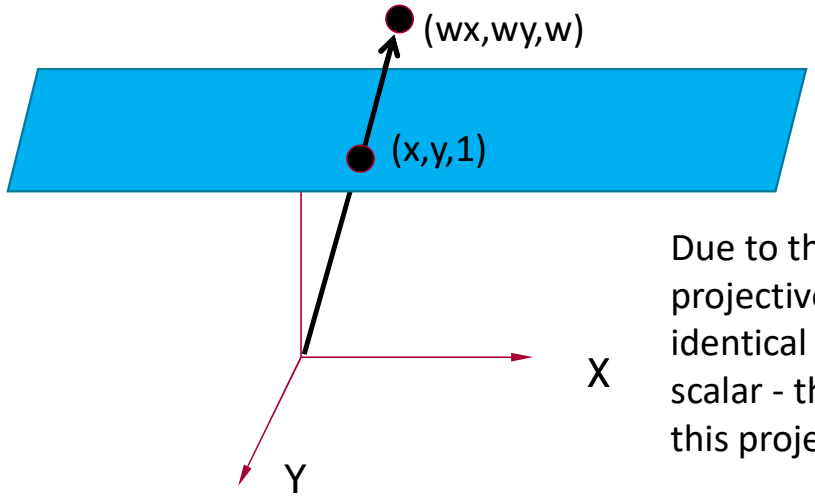


Typically, the last coordinate in a projective space is denoted with w.

Homogeneous Coordinates

A point (x,y) in R^2 embedded in a projective space corresponds to (x,y,1). All points (x,y,1) form a plane (referred to as *affine plane*)

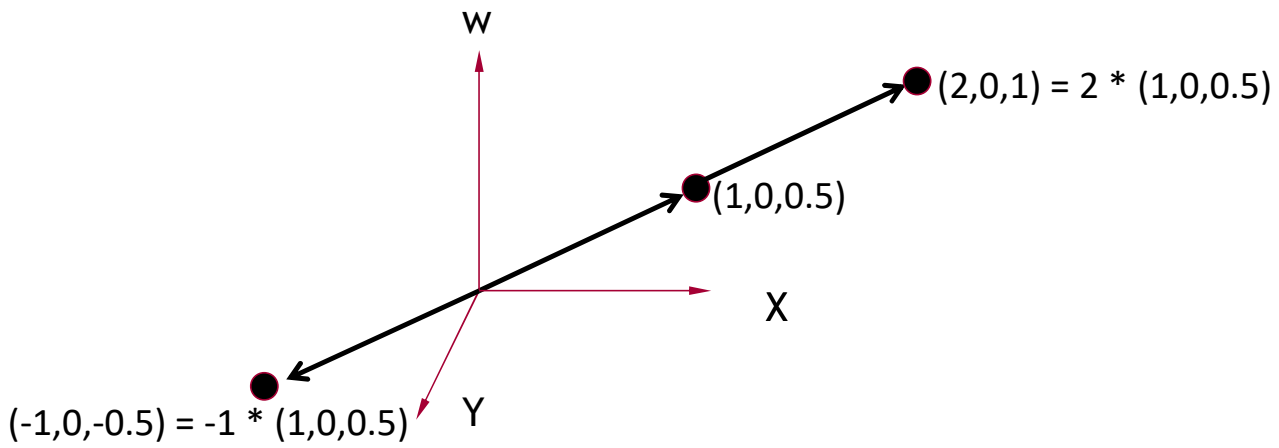
The points in this plane correspond to points in our standard vector space R^2



Due to the definition of a projective space, points are identical when multiplied by any scalar - they are, thus, lines in this projective space!

Homogeneous Coordinates

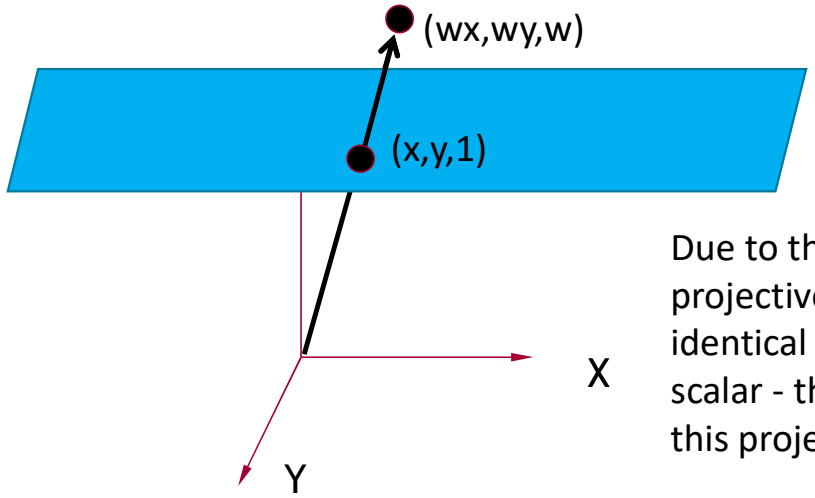
- Example:



Homogeneous Coordinates

A point (x,y) in R^2 embedded in a projective space corresponds to (x,y,1). All points (x,y,1) form a plane (referred to as *affine plane*)

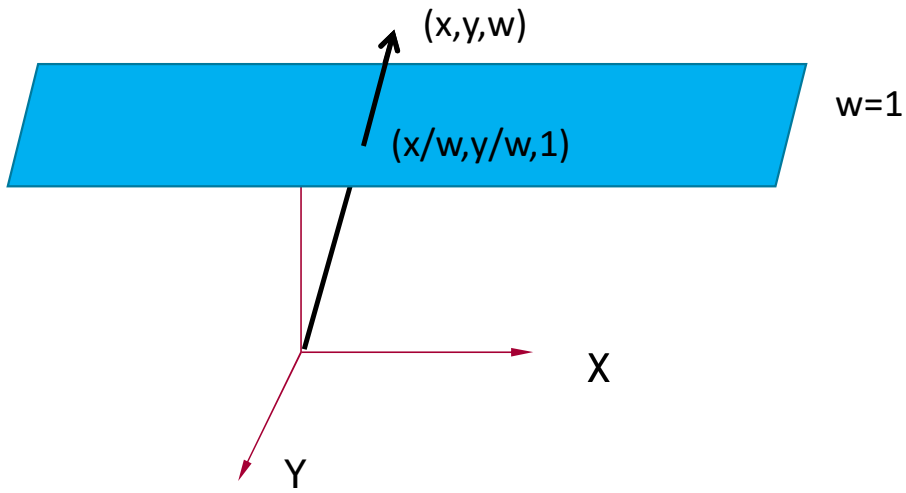
The points in this plane correspond to points in our standard vector space R^2



Due to the definition of a projective space, points are identical when multiplied by any scalar - they are, thus, lines in this projective space!

Homogeneous Coordinates

- We also can map points from P^2 to R^2 by dividing the coordinates by the last: In P^2 (x,y,w) is equal to (x/w,y/w,1) and corresponds to (x/w,y/w) in R^2.



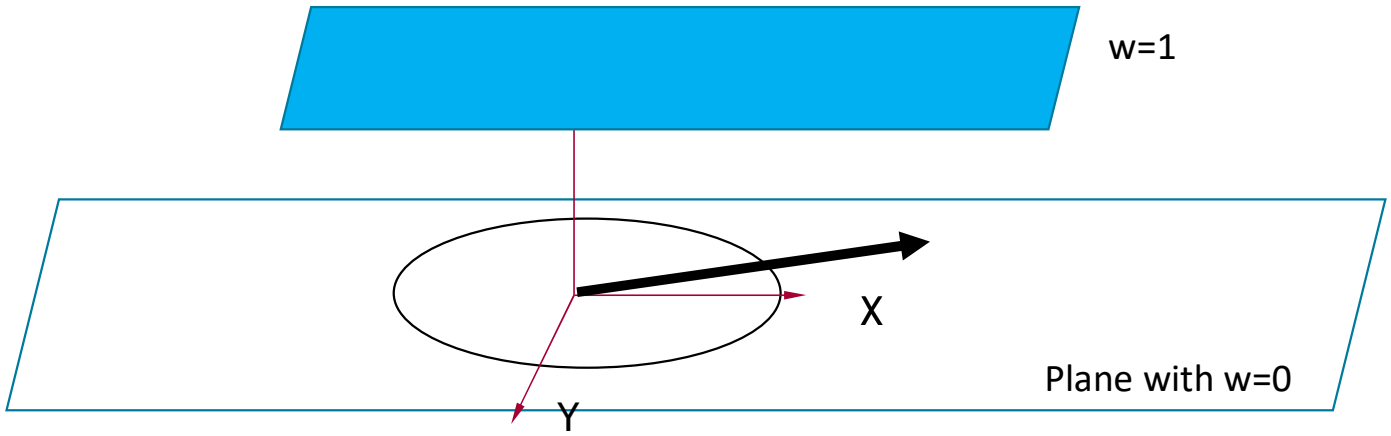
Homogeneous Coordinates

In the general case, from projective to standard vector space:

- For $(x_0,x_1,\ldots x_{n-1},w)$, the corresponding point in R^n is $(x_0/w,x_1/w,\ldots x_{n-1}/w)$ if w!=0
- $(x_0,x_1,\ldots x_{n-1},0)$ **has no correspondence** in R^n!

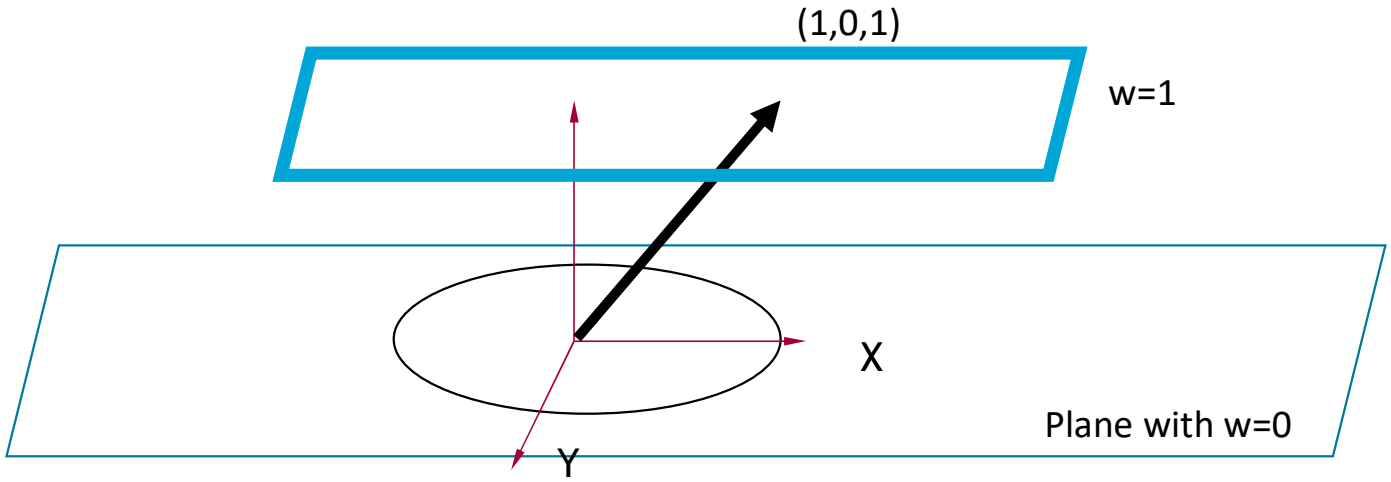
Homogeneous Coordinates

- What about the points with w=0?



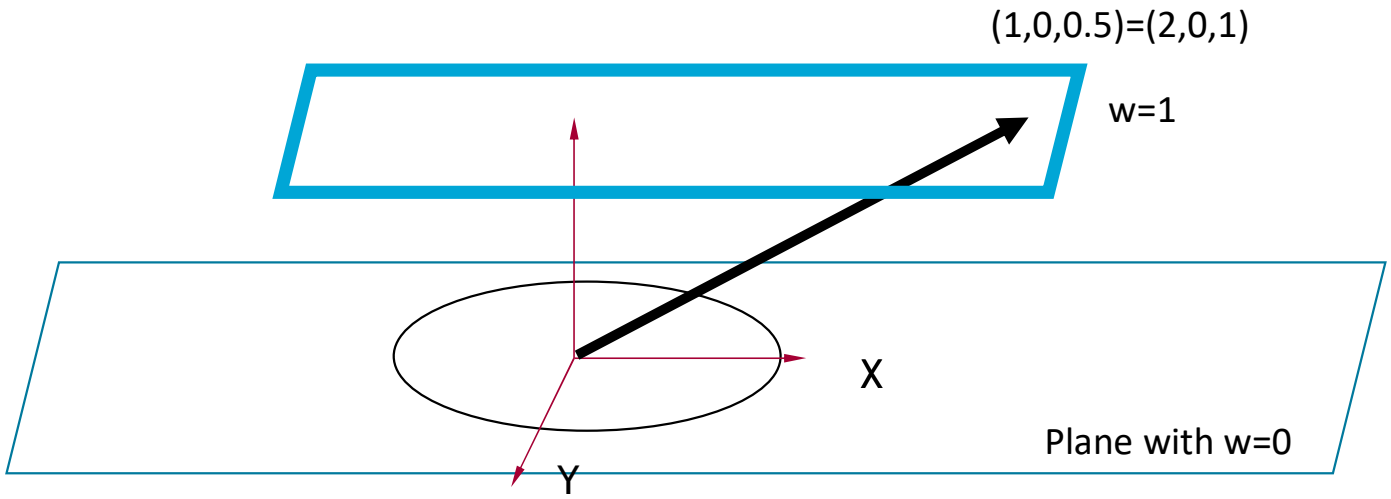
Homogeneous Coordinates

- What about the points with w=0?
- Let's see what happens for a point (1,0,w), when we decrease w...



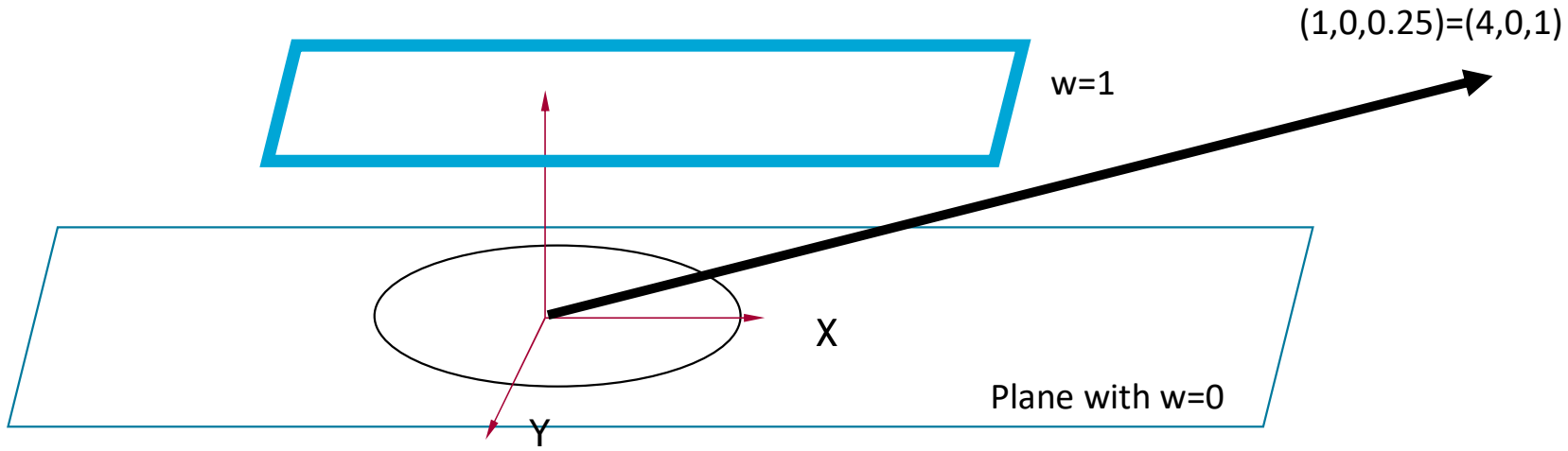
Homogeneous Coordinates

- What about the points with w=0?
- Let's see what happens for a point (1,0,w), when we decrease w…



Homogeneous Coordinates

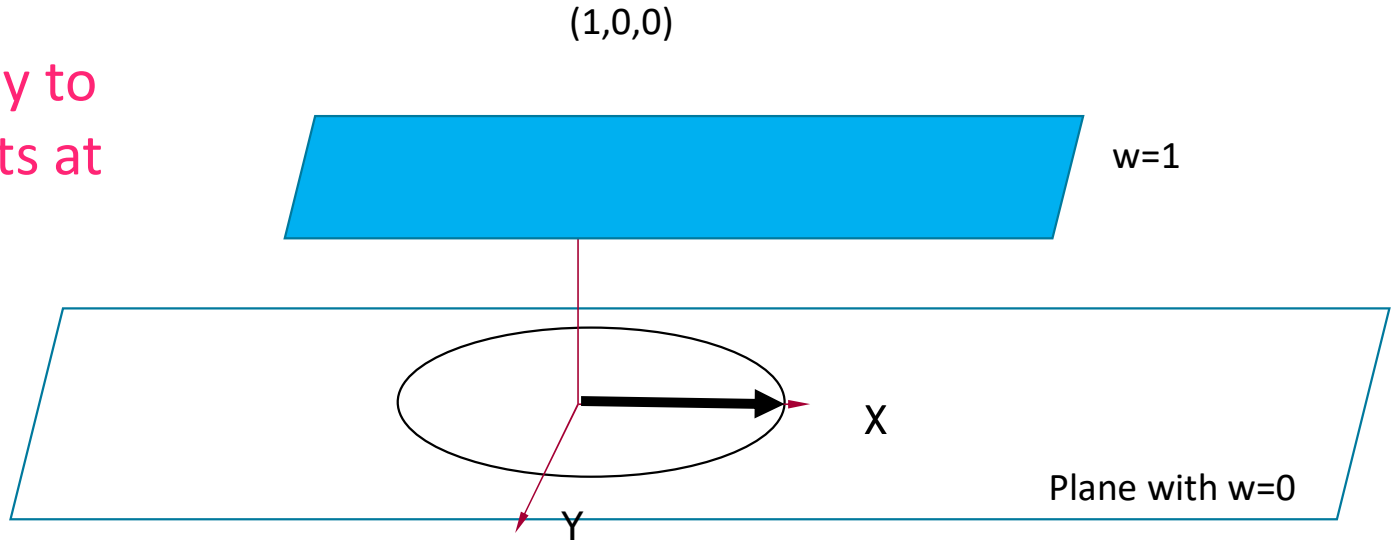
- What about the points with w=0?
- Let's see what happens for a point (1,0,w), when we decrease w…



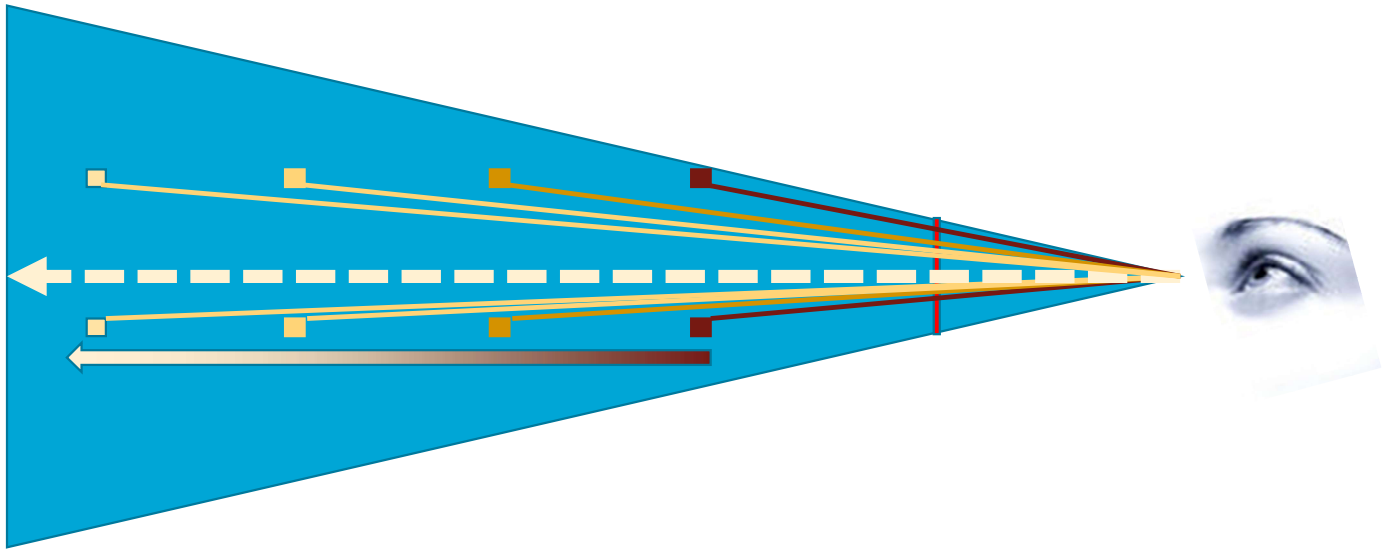
Homogeneous Coordinates

- What about the points with w=0?
- Let's see what happens for a point (1,0,w), when we decrease w…

We have a way to describe points at INFINITY!



Linear Perspective



Linear Perspective

Homogeneous Coordinates

We wanted:

Easy transformations = matrices

Translations, rotations,
scaling, projection should all be matrices…

Concatenating transformations
means simply matrix multiplications!

Translations in R^2

- To translate vector (x,y),
  we add (t_x,t_y) to obtain: $(x+t_x,y+t_y)$

In a projective space,
we would like to have a matrix M, such that

$M(x,y,1) = (x+t_x,y+t_y,1)$

Translations in homog. coordinates

- R^2

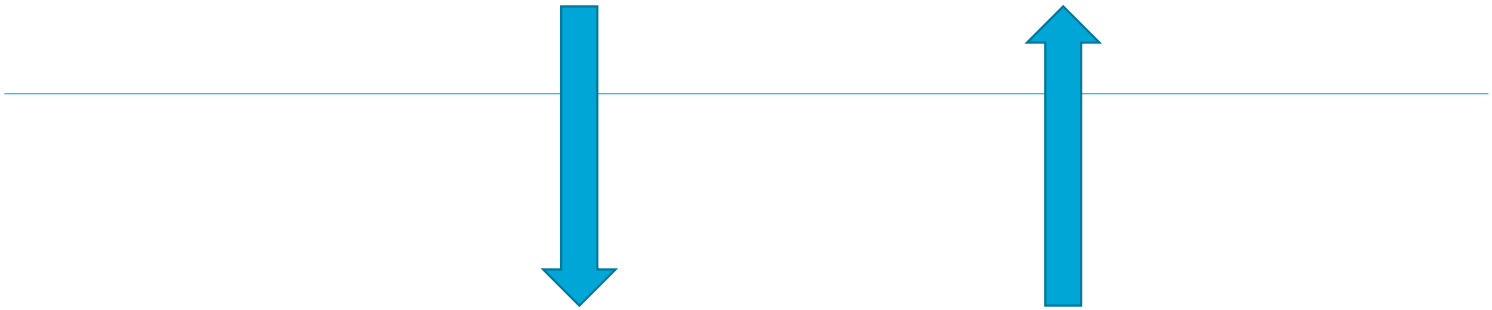
$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

- P^2

Translations in homog. coordinates

- R^2

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$



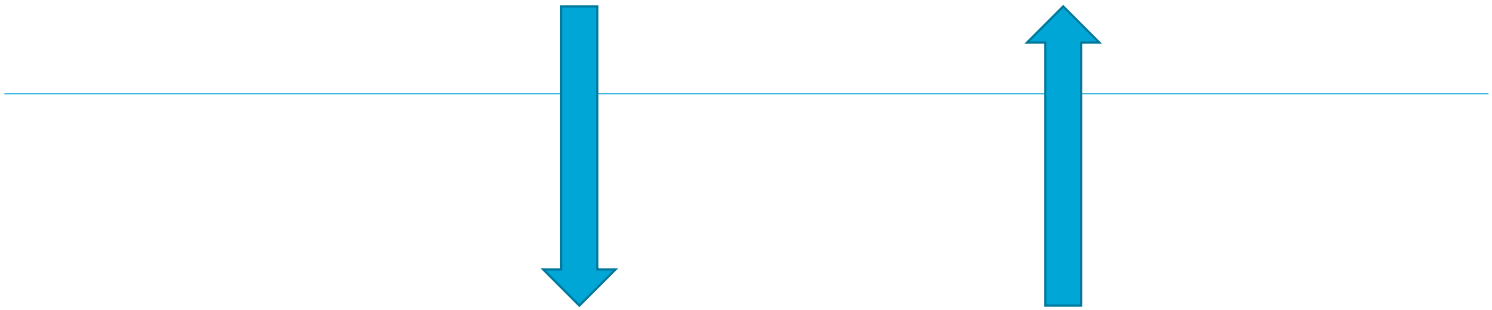
- P^2

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

Translations in homog. coordinates

- R^2

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

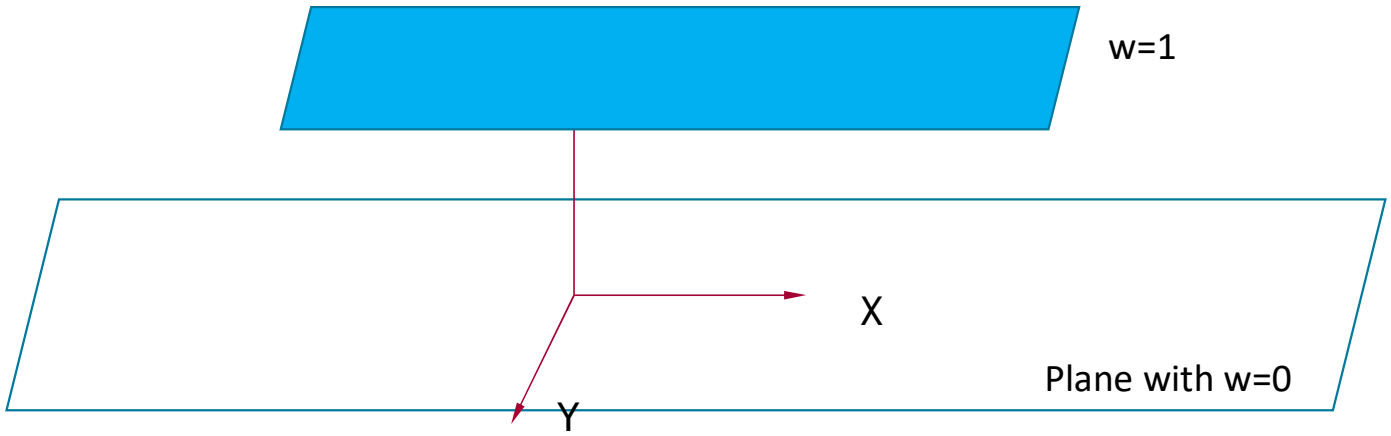


- P^2

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

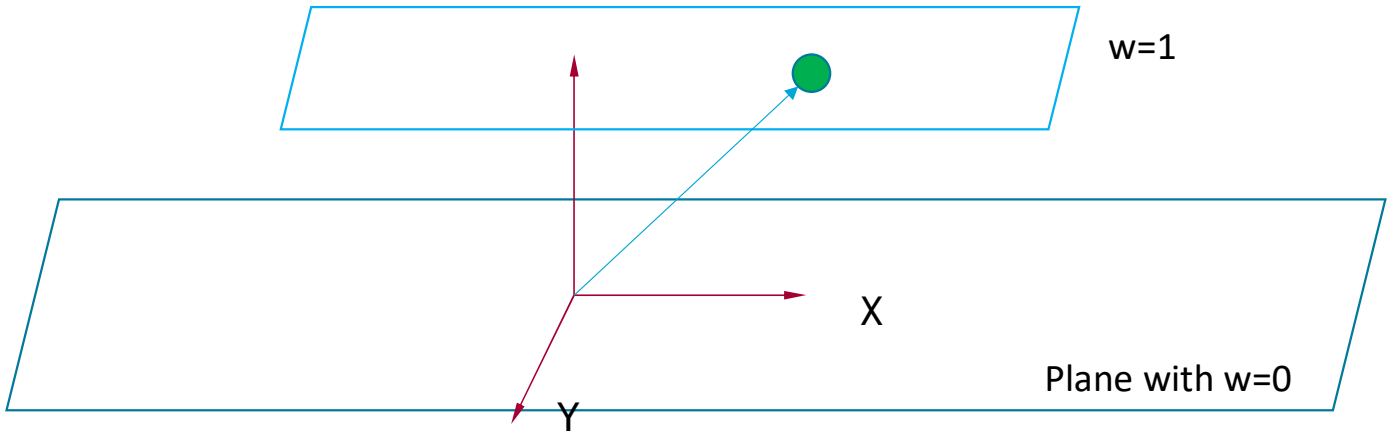
How does this work?

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$



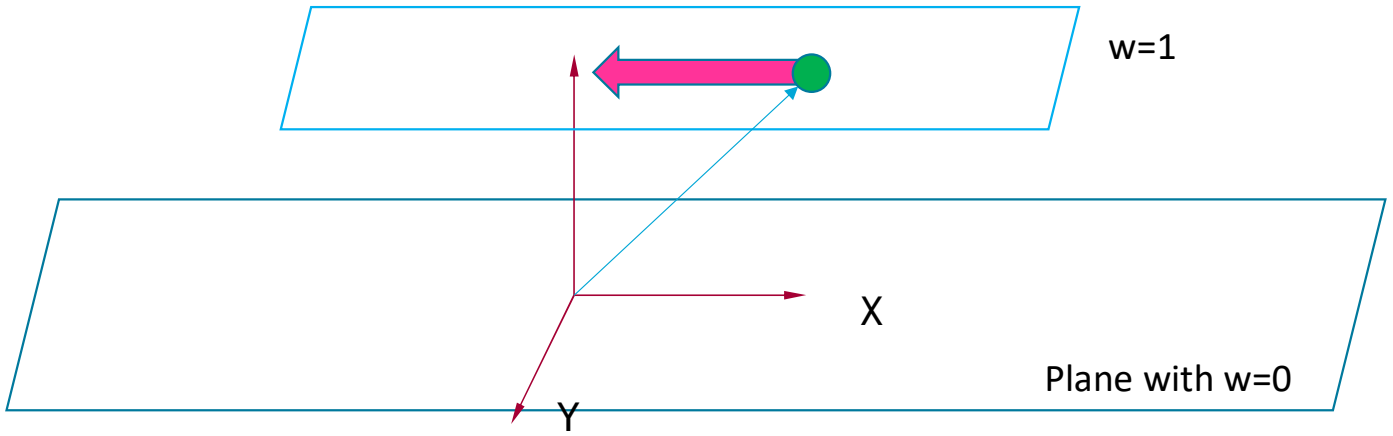
How does this work?

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



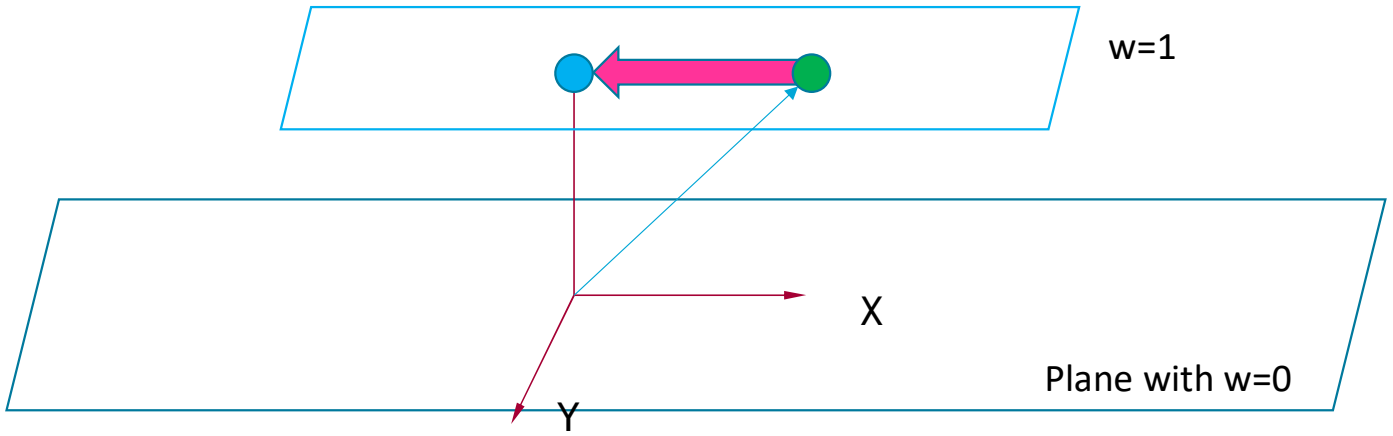
How does this work?

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



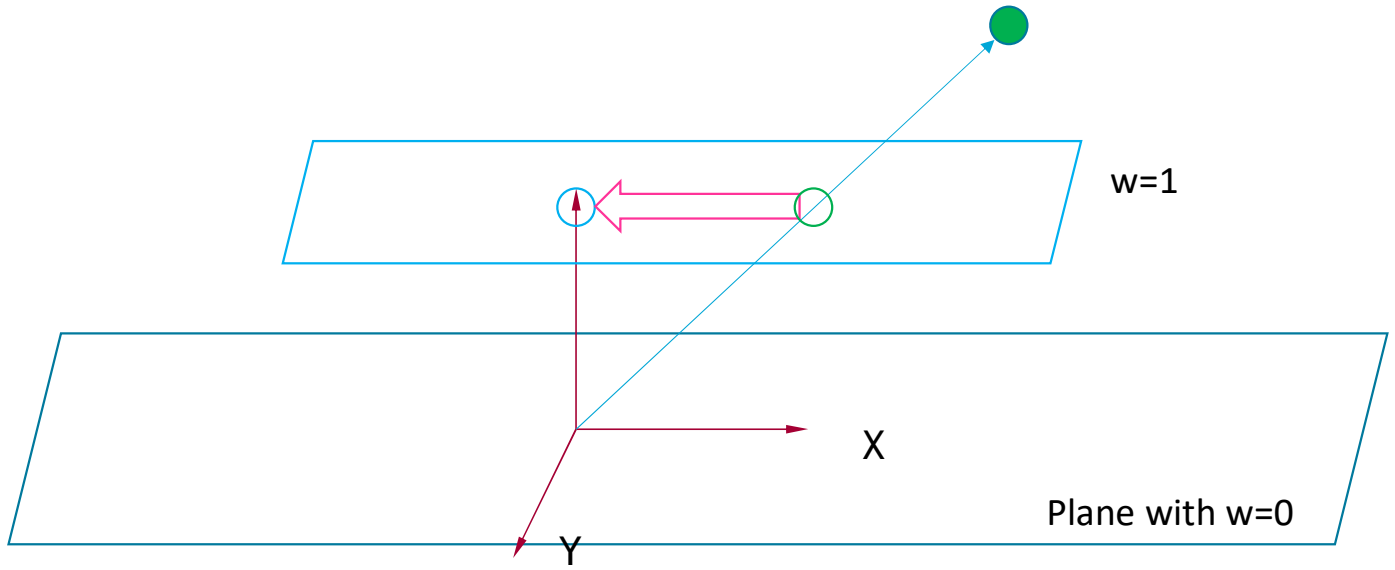
How does this work?

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



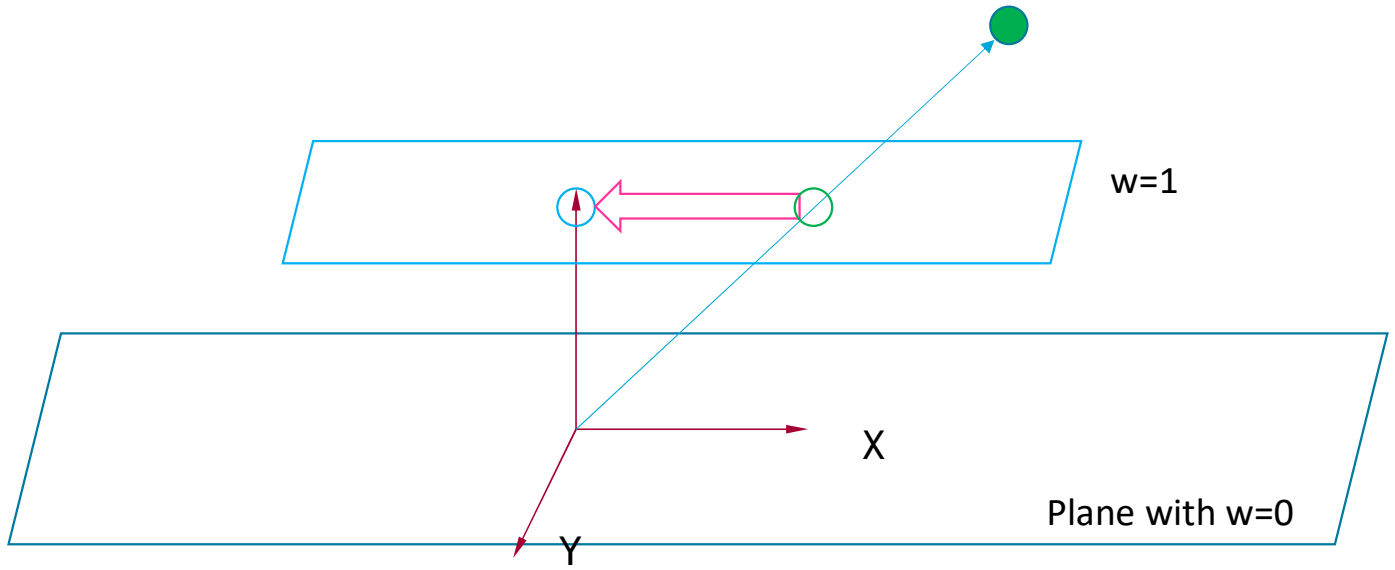
How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix}$$



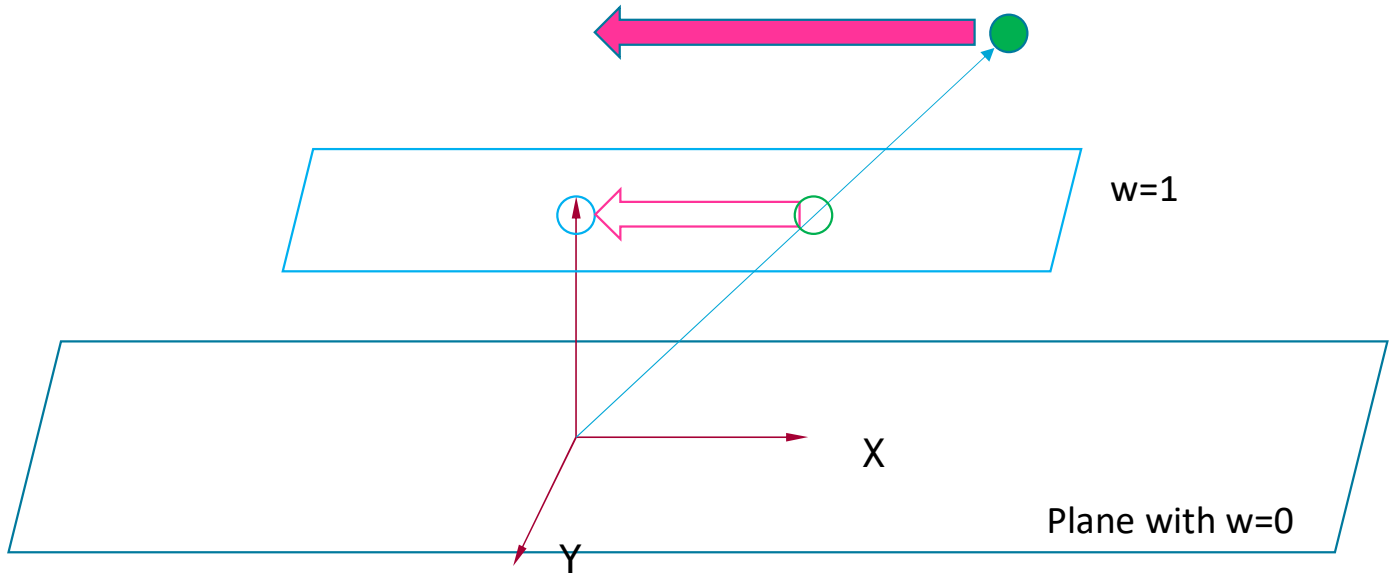
How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ \mathbf{2} \end{bmatrix}$$



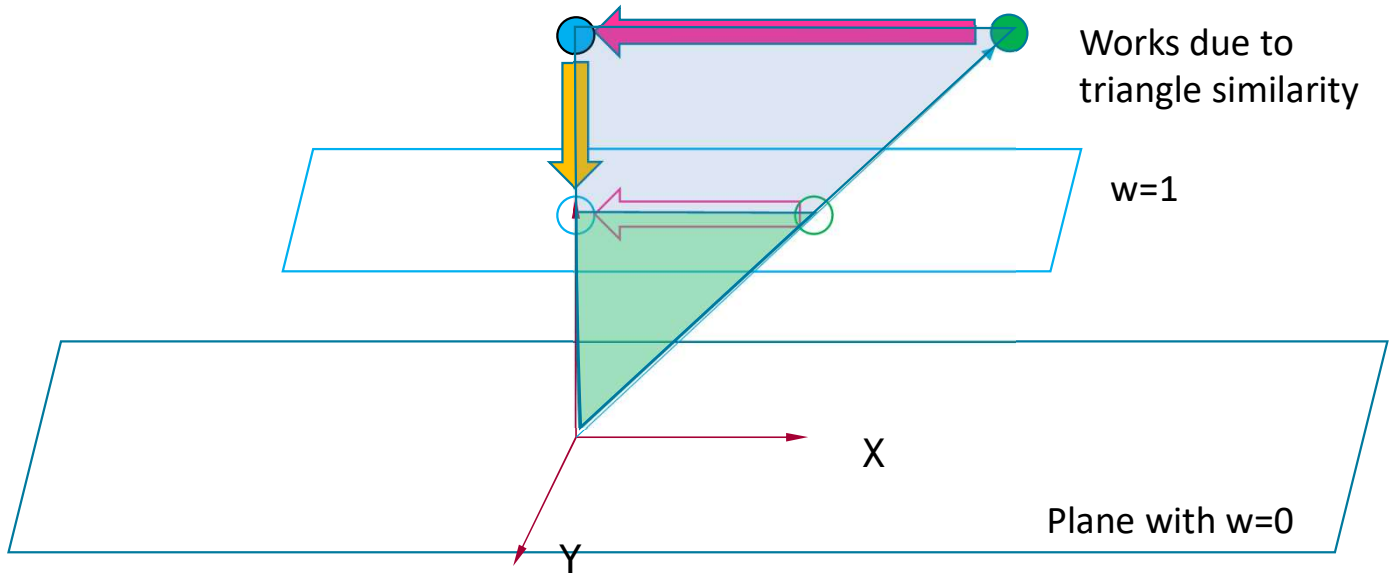
How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 - 2 \\ 0 + 0 \\ 2 \end{bmatrix}$$



How does this work?

$$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 2-2 \\ 0+0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$



Translations in homog. coordinates

- R^2

$$\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \end{bmatrix}$$

- P^2

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + tx \\ y + ty \\ 1 \end{bmatrix}$$

Translations in homog. coordinates

What about points at infinity?

- P^2

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} x+0 \\ y+0 \\ 0 \end{bmatrix}$$

Rotation in homog. coordinates

- R^2

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\theta)\,x - \sin(\theta)\,y \\ \sin(\theta)\,x + \cos(\theta)\,y \end{bmatrix}$$

- P^2

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta)\,x - \sin(\theta)\,y \\ \sin(\theta)\,x + \cos(\theta)y \\ 1 \end{bmatrix}$$

Rotation in homog. coordinates

What about points at infinity?

- P^2

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta)\, x - \sin(\theta)\, y \\ \sin(\theta)\, x + \cos(\theta) y \\ 0 \end{bmatrix}$$

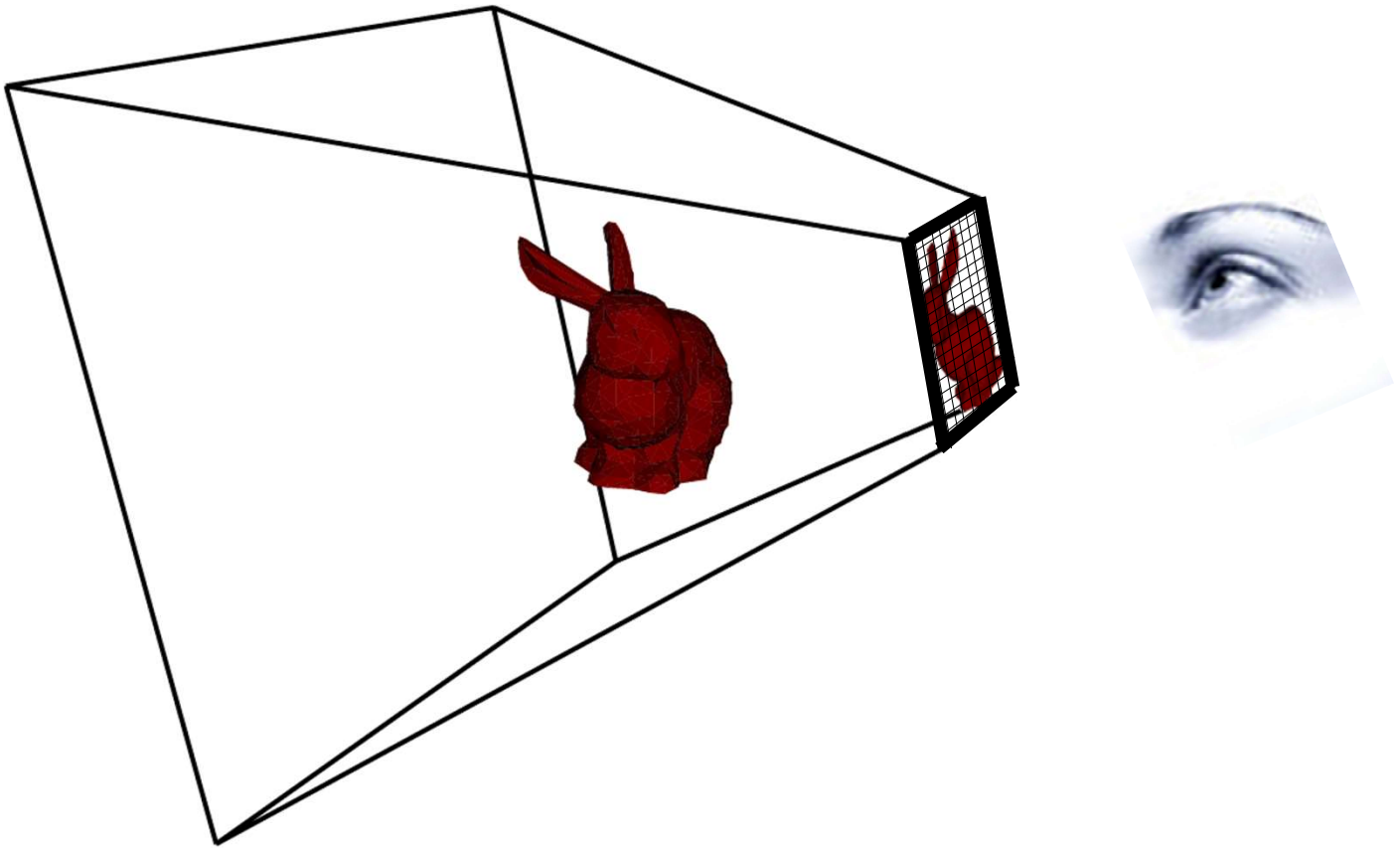
Rotation around point Q

- Rotation around point Q:
  - Translate Q to origin($\mathbf{T}_Q$),
  - Rotate around origin ($\mathbf{R}_\Theta$)
  - Translate back to Q ($\mathbf{T}_{-Q}$).

$$P' = (T_{-Q})\mathbf{R}_\Theta T_Q \, P$$

Exercise: Construct an example yourself with a 45 degree rotation and test the resulting matrix to see if it worked.

What about 3D?



And in 3 dimensions ?

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

- The same!
- Add a fourth coordinate, *w*

- All transformations are 4x4 matrices

Translations in 3D

Translations in 3D

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

$$\begin{cases} x' = x + wt_x \\ y' = y + wt_y \\ z' = z + wt_z \\ w' = w \end{cases}$$

Rotations in 3D

- Rotation = axis and angle

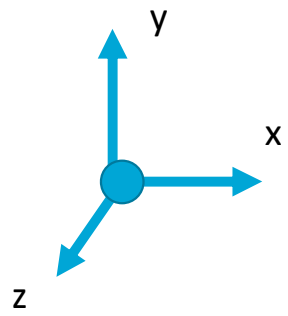
- Rotations around x,y,z axis are simple matrices
  - All axes can be achieved by combining these
    (it is a bit cumbersome though…)

Rotation about *z*-axis

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axis z not changing

Quick verification : rotation of π/2
Should change *x* in *y*, and *y* in -*x*



$$R_z(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about *x*-axis

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Axis x not changing

Quick verification : rotation of π/2
Should change *y* in *z*, and *z* in *-y*

$$R_x(\tfrac{\pi}{2}) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation about *y*-axis

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

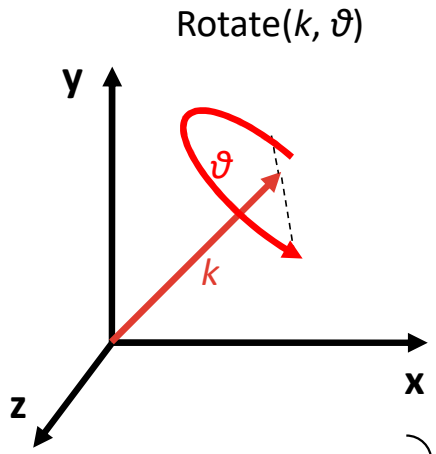
Axis y not changing

Quick verification : rotation of π/2
Should change *z* in *x*, and *x* in -*z*

$$R_y(\tfrac{\pi}{2}) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation around (k_x, k_y, k_z)

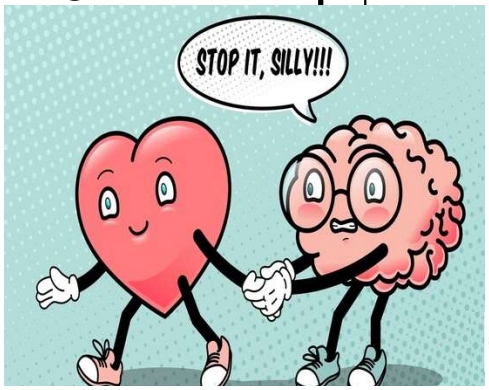
- Rodrigues Formula



$$\begin{pmatrix} k_xk_x(1-c)+c & k_zk_x(1-c)-k_zs & k_xk_z(1-c)+k_ys & 0 \\ k_yk_x(1-c)+k_zs & k_zk_x(1-c)+c & k_yk_z(1-c)-k_xs & 0 \\ k_zk_x(1-c)-k_ys & k_zk_x(1-c)-k_xs & k_zk_z(1-c)+c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

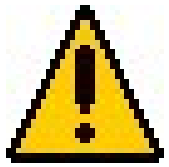
DO NOT LEARN THIS BY HEART!

where $c = \cos\vartheta$ & $s = \sin\vartheta$

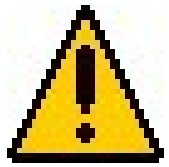


Attention !

- Matrix Multiplication is not commutative
- The order of transformations is important
  - Rotation then translation != transl. then rotation

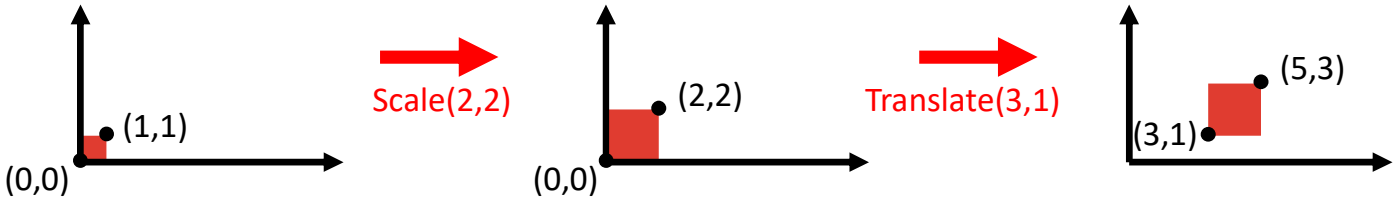


Source of bugs…



Example

Scaling + Translation

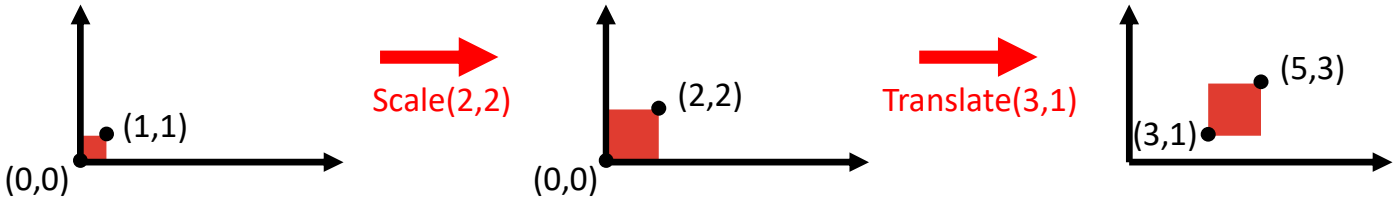


Matrix combination: $p' = T(Sp) = TSp$

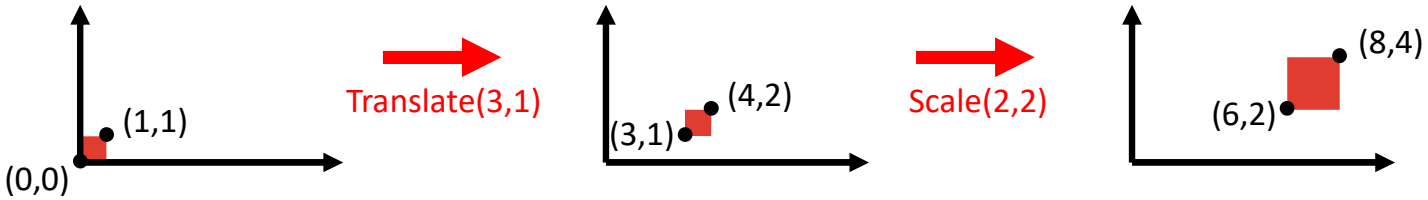
$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

NOT commutative

Scaling + translation: $p' = T(S\,p) = TS\,p$



Translation + scaling: $p' = S(T\,p) = ST\,p$



NOT commutative

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & \\ 0 & 2 & \\ 0 & 0 & 1 \end{pmatrix}$$

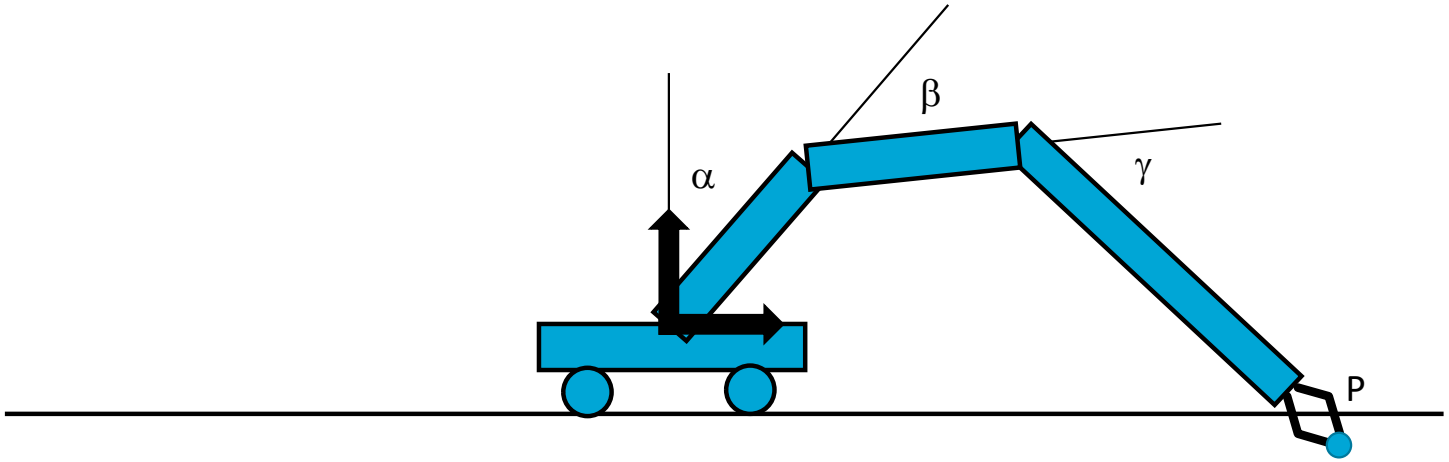
NOT commutative

$$TS = \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 3 \\ 0 & 2 & 1 \\ 0 & 0 & 1 \end{pmatrix}$$

$$ST = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 6 \\ 0 & 2 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

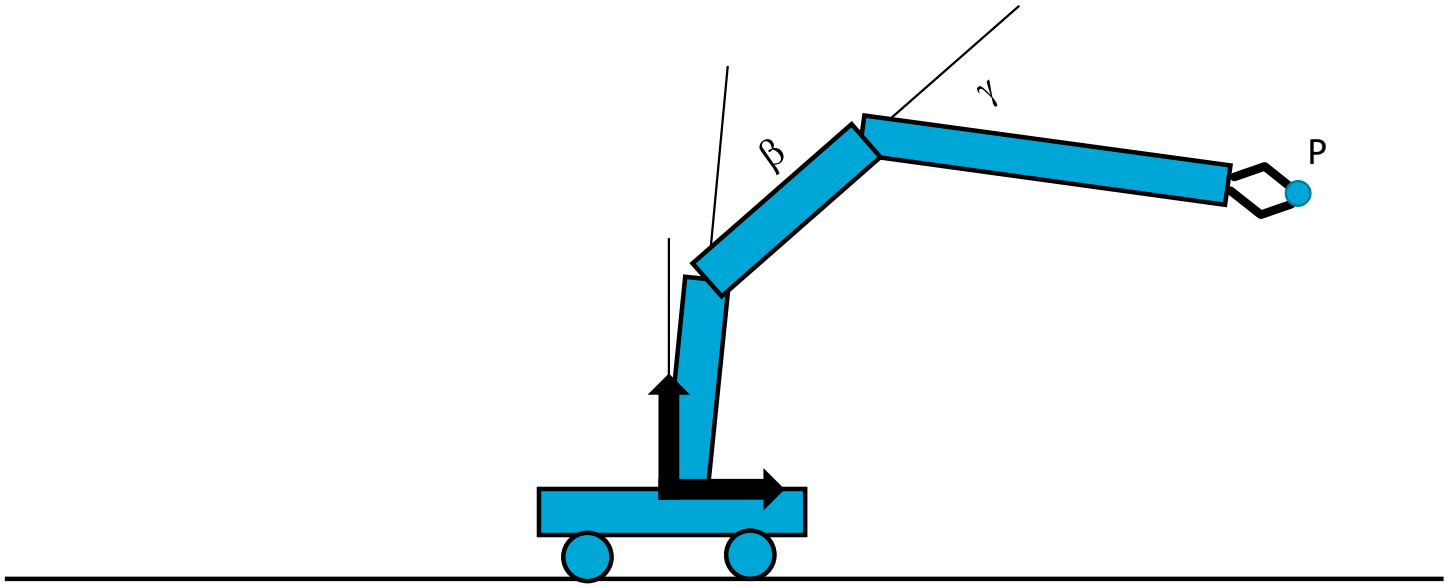
Complex Object Hierarchies

Change angle α and calculate the new position of point P with **one** matrix multiplication!

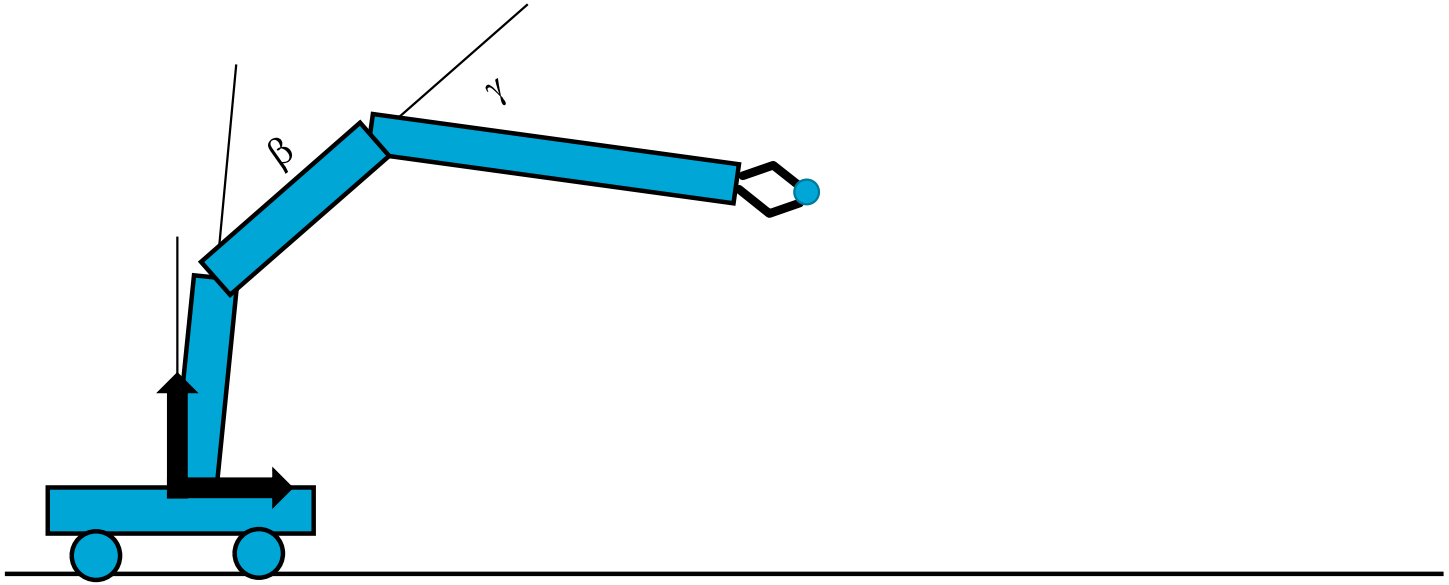


Complex Object Hierarchies

Change angle α and calculate the new position of point P with **one** matrix multiplication!

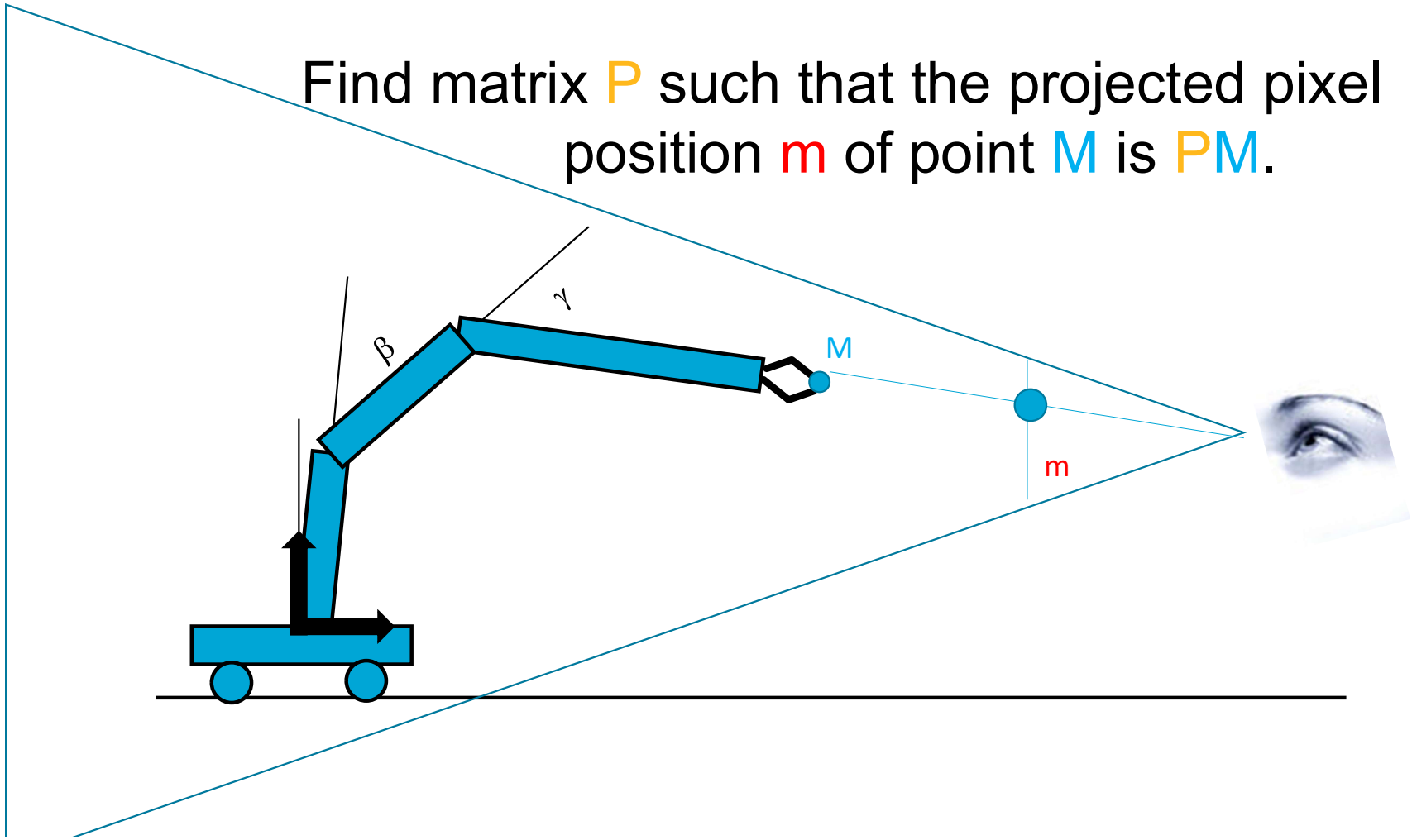


Complex Object Hierarchies



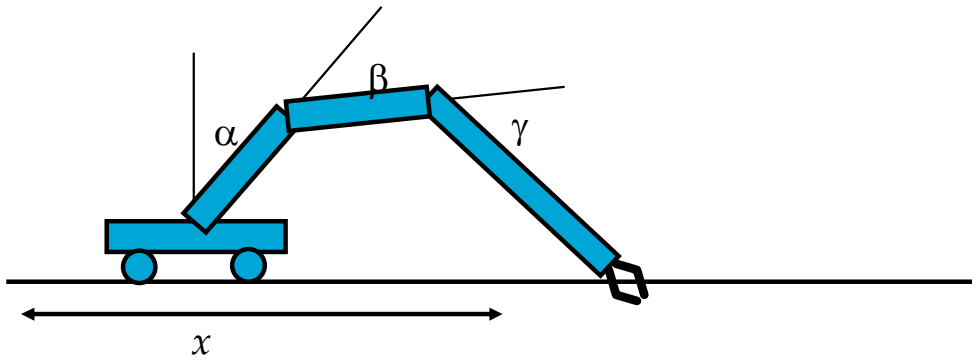
Complex Object Hierarchies

Find matrix P such that the projected pixel position m of point M is PM.



Complex Objects

- Objects are often defined via many components
  - E.g., wheels of cars, fingers on hands on arms …
- Concatenate matrices to place objects relative to each other
  - Changing one matrix affects everything hereafter



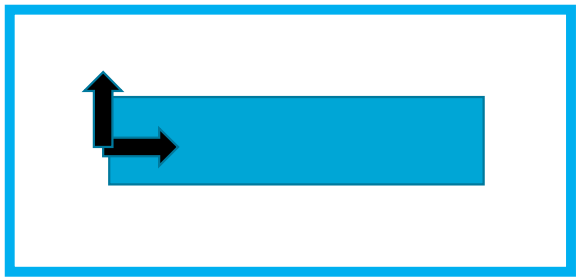
Complex Objects

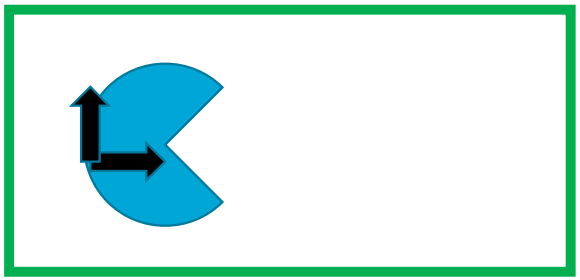
Example:

We make a “complex” robot arm consisting of two parts:

The arm itself and a hand

Both are designed independently
and are at the origin (shown below)





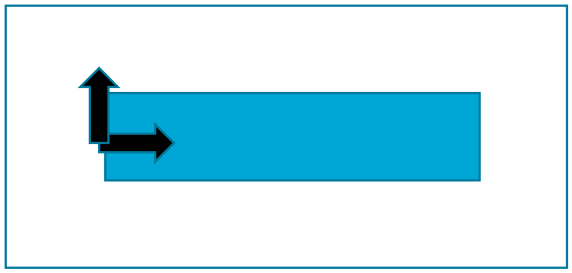
Why concatenate operations?

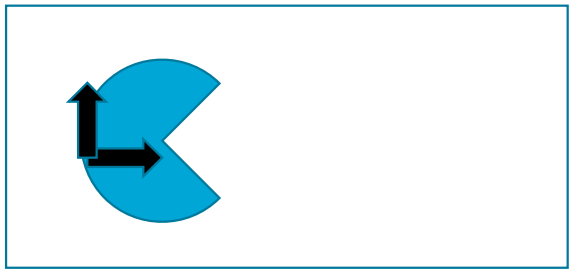
Example:

Robot arm consisting of two parts:

The arm itself and a hand

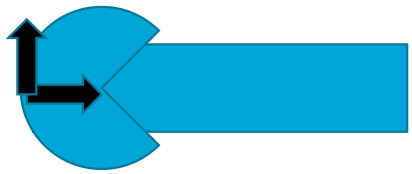
Both are designed independently
and are at the origin (shown below)





Complex Objects

- Using the coordinates as is, you get:



- That does not look right!
- Instead: Let's define a matrix to shift the objects to the wanted relative location

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)

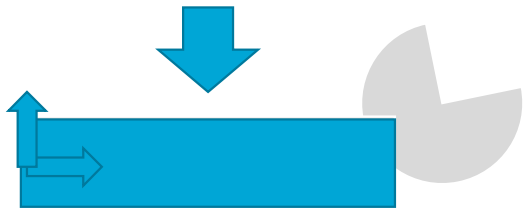
Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm

Resulting positions
After applying the object
vertices to matrix S



Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T

Complex Objects

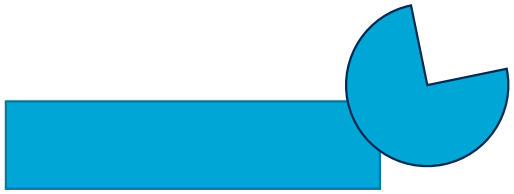
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand



Complex Objects

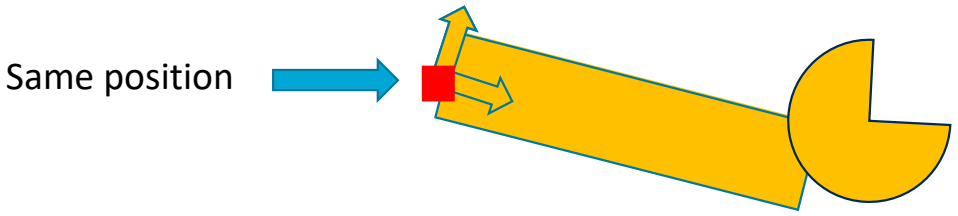
- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

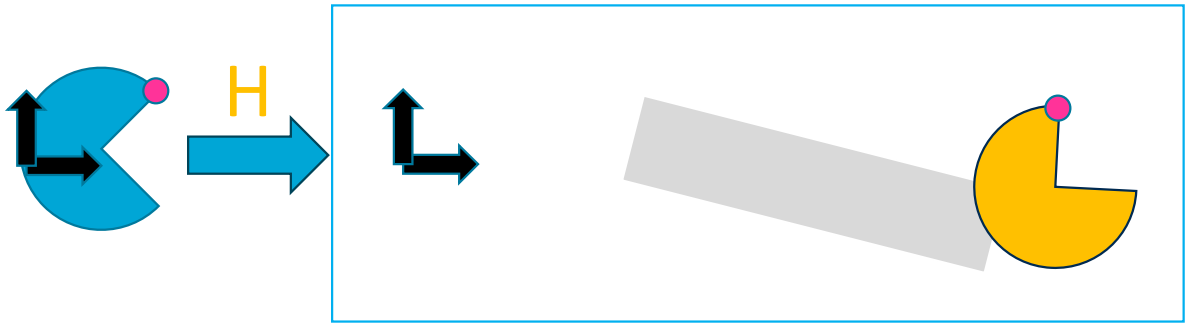
Complex Objects

- Concatenate and apply matrices
  - S:=Translation matrix to position of arm (**S**houlder)
  - **S:=SN, with N a new rotation matrix**
  - Apply S to all vertices of arm
  - T:=translation along arm (to the **J**oint of the hand)
  - J=S T
  - R= Rotation (for **H**and)
  - H:=J R
  - Apply H to all vertices of the hand

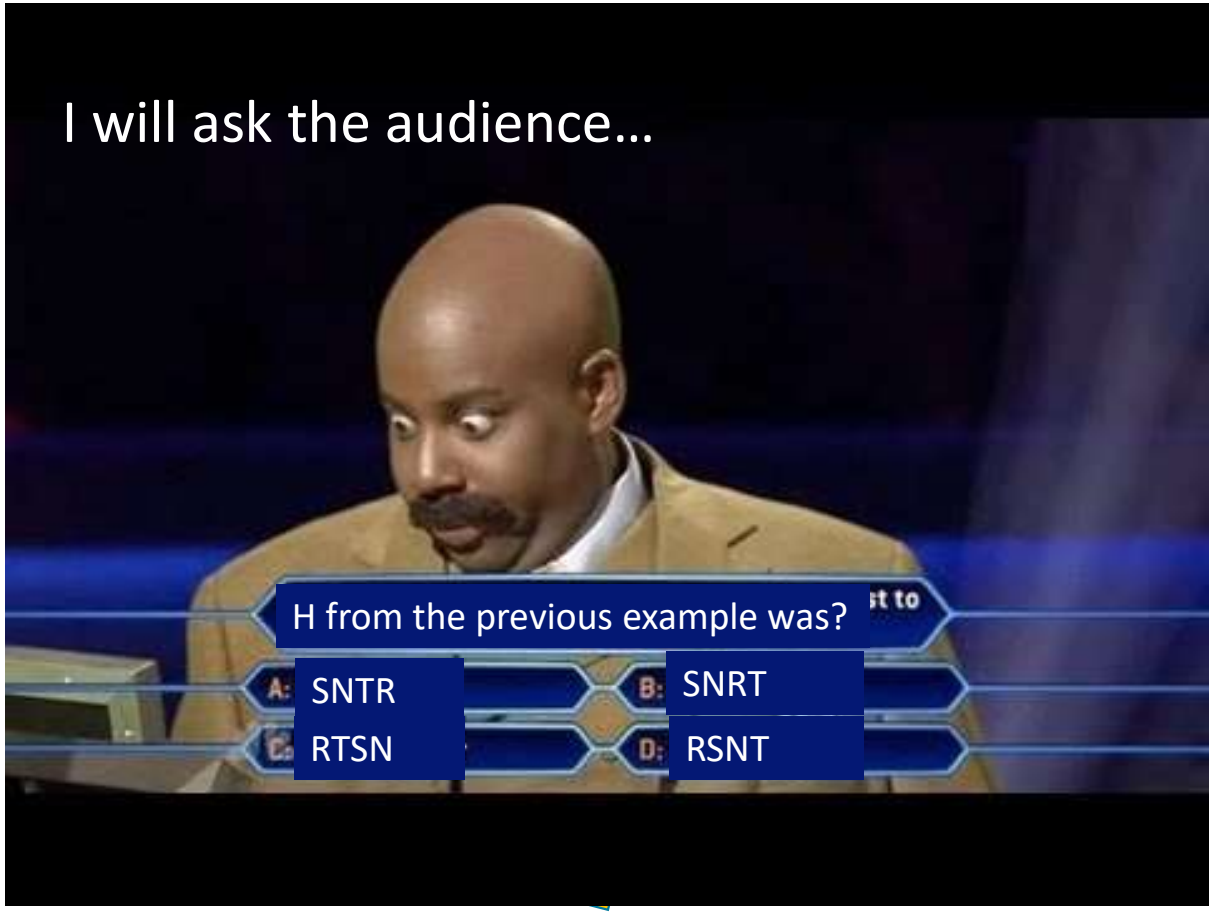


Complex Objects

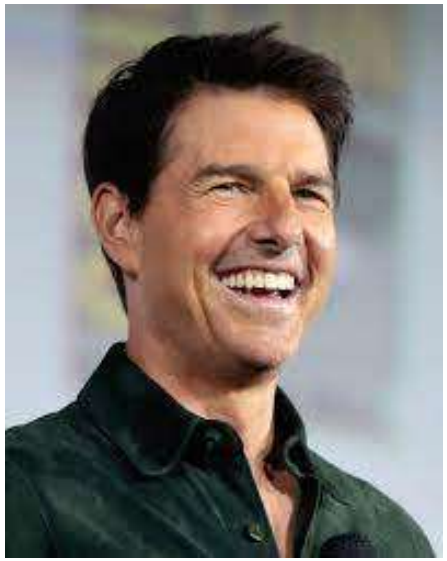
We found a matrix H to transform the vertices from the original hand definition into the wanted position relative to the arm!



What was the definition of Matrix H?



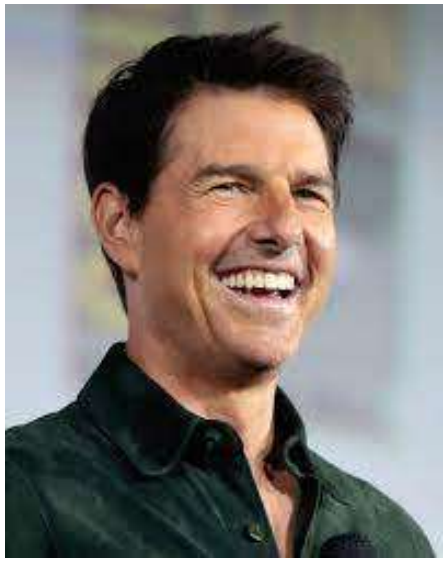
How to find the matrix order?



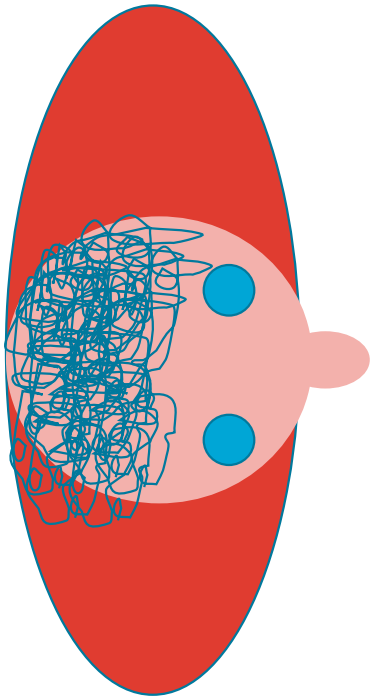
Tom/Penelope “Cross”

wikipedia

Complex Objects - How to find the matrix order?



Tom “Cross”

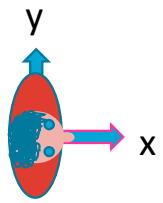


“Cross”

Penelope “Cross”

How to find the matrix order?

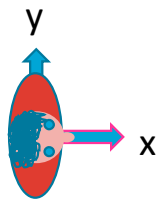
Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.



Tom/Penelope “Cross”

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

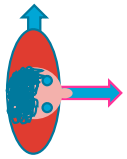


Tom/Penelope “Cross”

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

T=Translate(2,0)



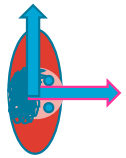
Tom/Penelope “Cross”

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

T=Translate(2,0)

T * “cross”



How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

T=Translate(2,0)

T * “cross”

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

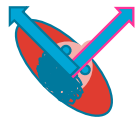
T=Translate(2,0)
R=Rotate(45)

T * “cross”

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

T * R * “cross”



T=Translate(2,0)
R=Rotate(45)

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

T=Translate(2,0)
R=Rotate(45)

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

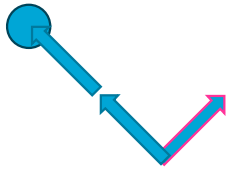
T=Translate(2,0)
R=Rotate(45)

Let’s apply both operations to a point:
$T*R*(0,2,1)^t$

How to find the matrix order?

Matrix applications from **left to right** can be interpreted as moving the origin of our space.
If we depict the origin by “Cross”, matrix operations will move “Cross” (and in turn the origin) around.

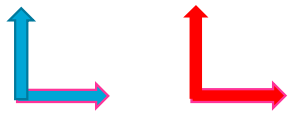
T=Translate(2,0)
R=Rotate(45)



Let’s apply both operations to a point:
$T*R*(0,2,1)^t$

How to find the matrix order?

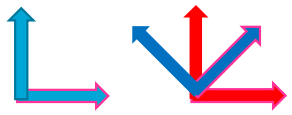
T=translate (2,0)



How to find the matrix order?

T=translate (2,0)

R=rotate(45)

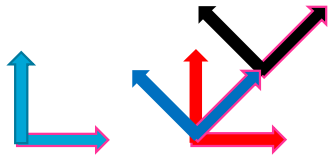


How to find the matrix order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)



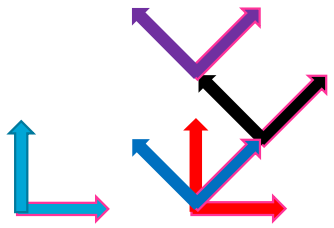
How to find the matrix order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)



How to find the matrix order?

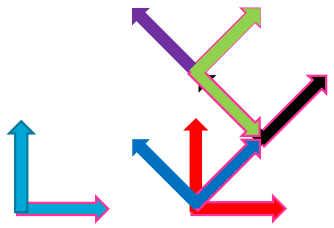
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



How to find the matrix order?

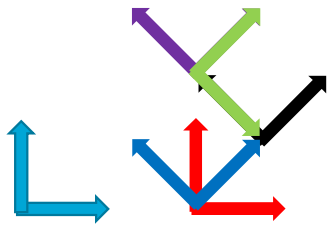
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Multiply left to right in the “playthrough” order

How to find the matrix order?

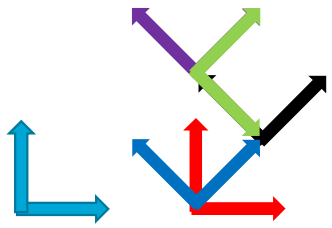
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Multiply left to right in the “playthrough” order

How to find the matrix order?

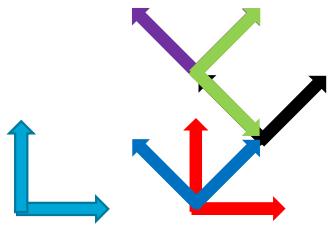
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,0,1)^t$?

How to find the matrix order?

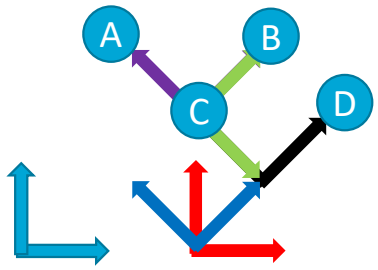
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,0,1)^t$?

How to remember the order?

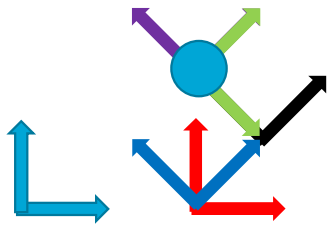
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,0,1)^t$?

How to find the matrix order?

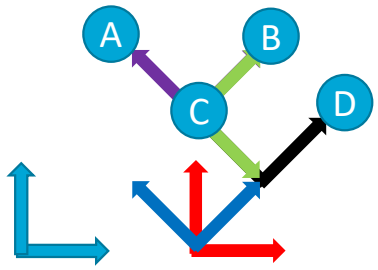
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,1,1)^t$?

How to remember the order?

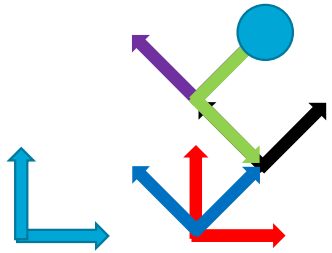
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,1,1)^t$?

How to remember the order?

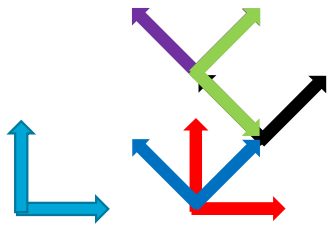
T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)



Matrix for the last position would be: TRTTR

Where is TRTTR $(0,2,2)^t$?

How to find the matrix order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)

(TRTTR) P

Multiply left to right in the “playthrough” order

How to find the matrix order?

T=translate (2,0)

R=rotate(45)

T=translate (1,0)

T=translate (0,1)

R=rotate(-90)

(TRTTR) P

Multiply left to right in the “playthrough” order

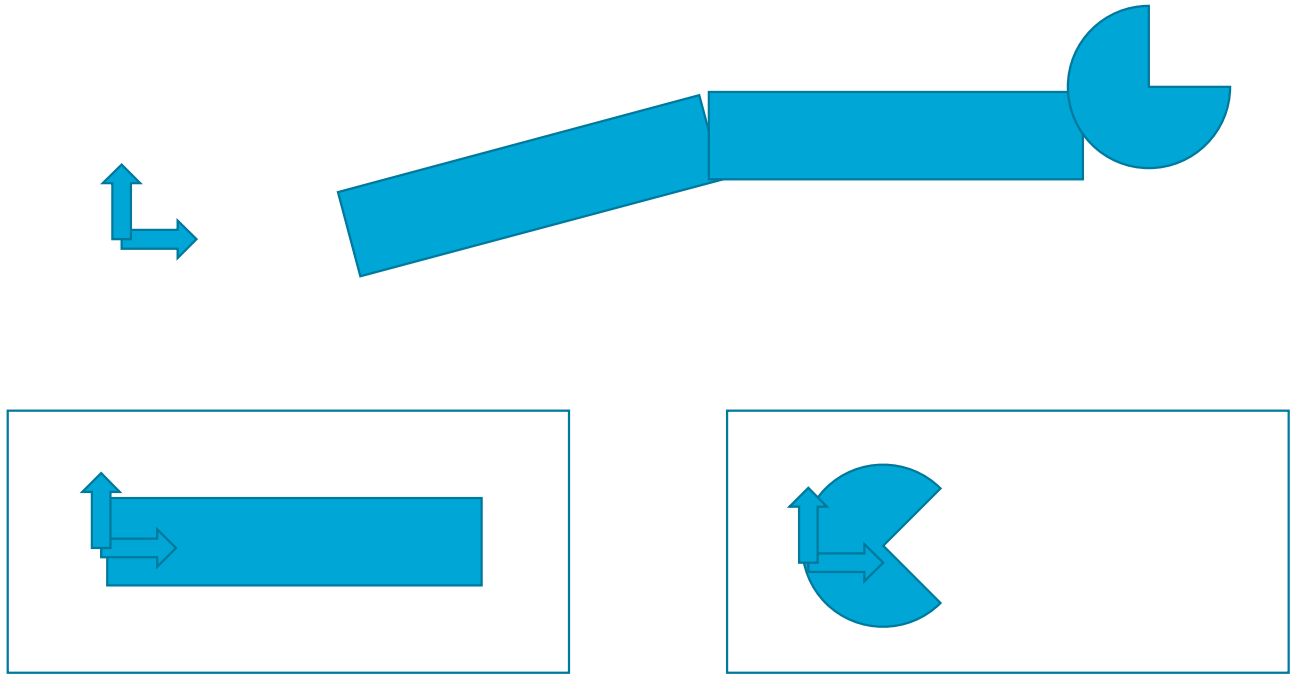
Why does this work? P multiplies mathematically from the right!

Local vs. Global Interpretation

- Two interpretations of **the same mathematics**:
  - Mathematically, points do multiply from the right
    This transforms the points with respect to a GLOBAL coordinate system.
  - Looking at the matrices from the left interprets these transformations as a change of the origin/coordinate system. In other words, a LOCAL coordinate system is established.

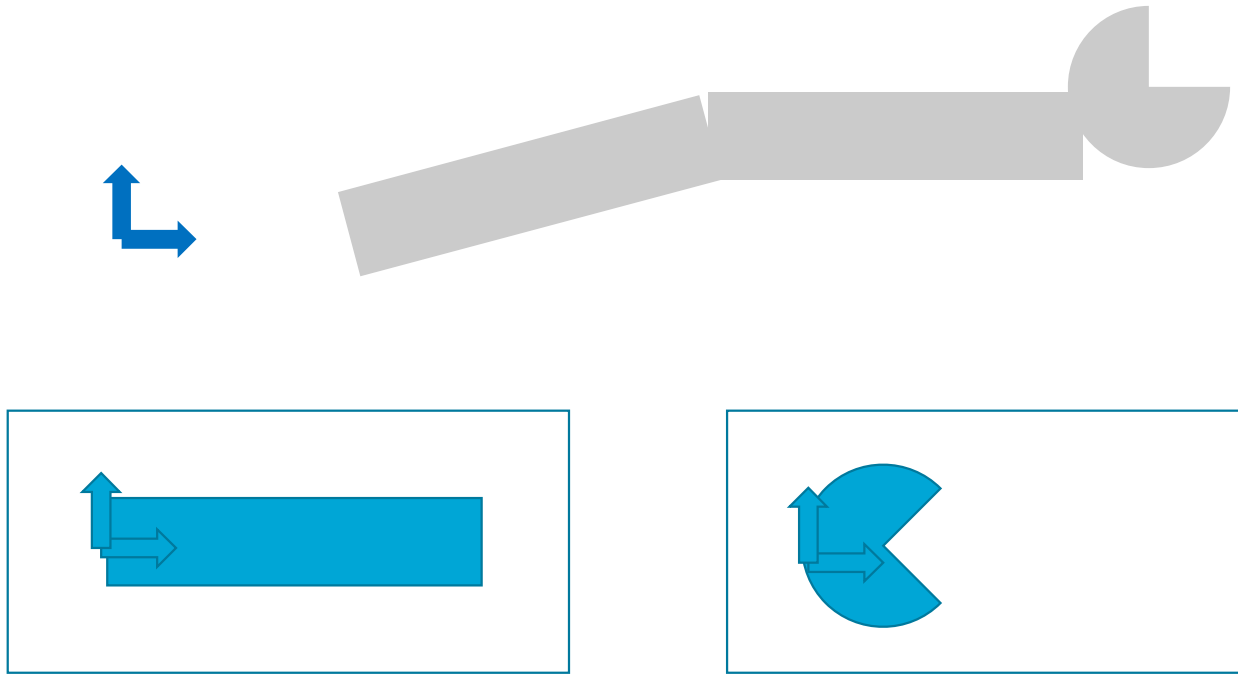
Local Interpretation

- Build complex object dependencies



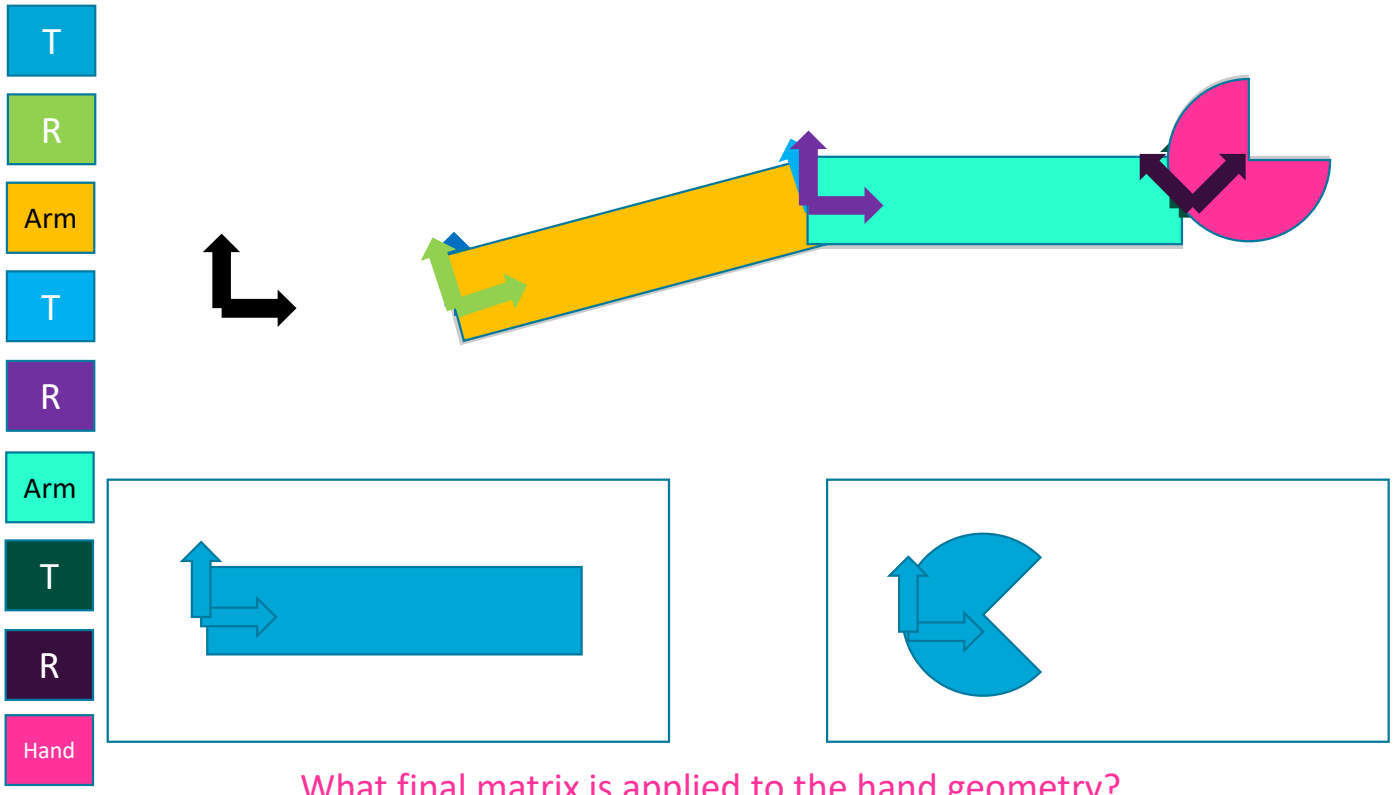
Local Interpretation

- Build complex object dependencies



Local Interpretation

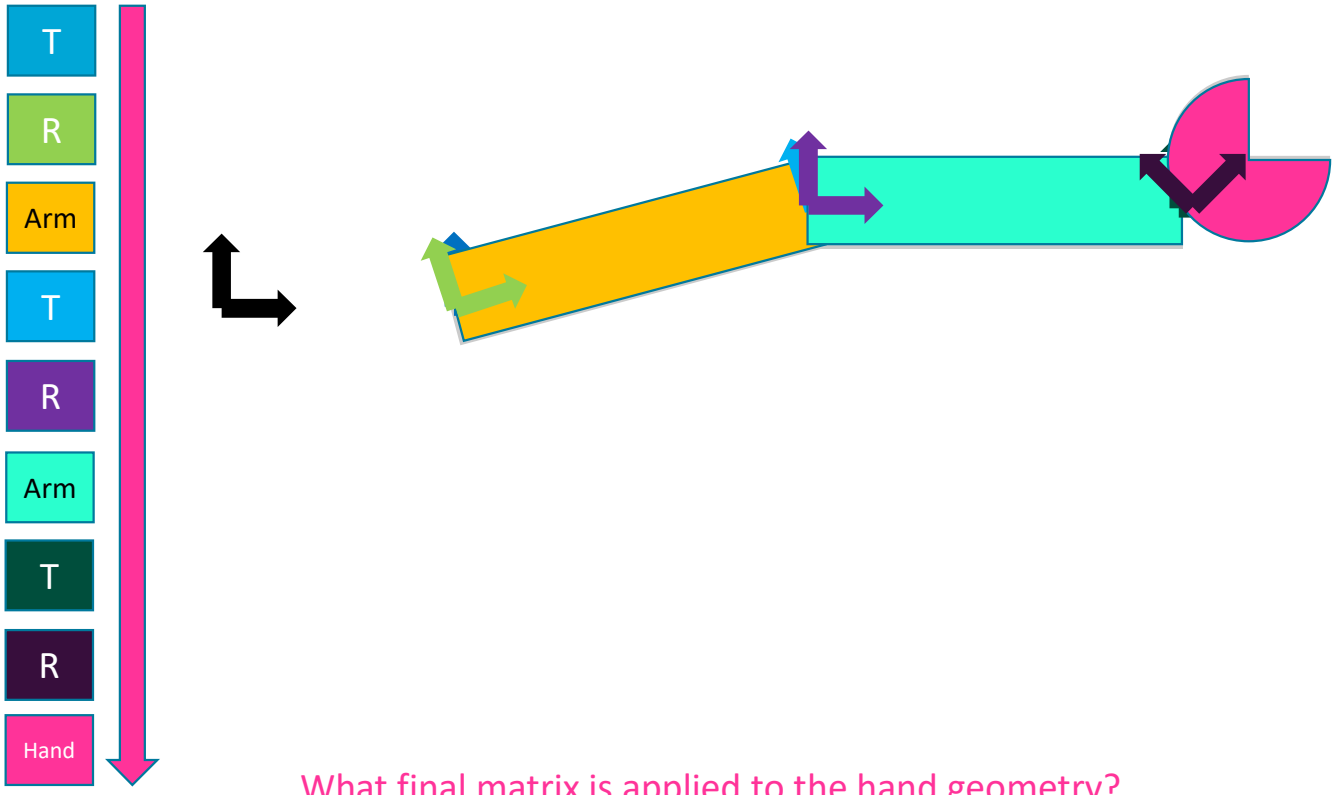
- Build complex object dependencies



What final matrix is applied to the hand geometry?

Local Interpretation

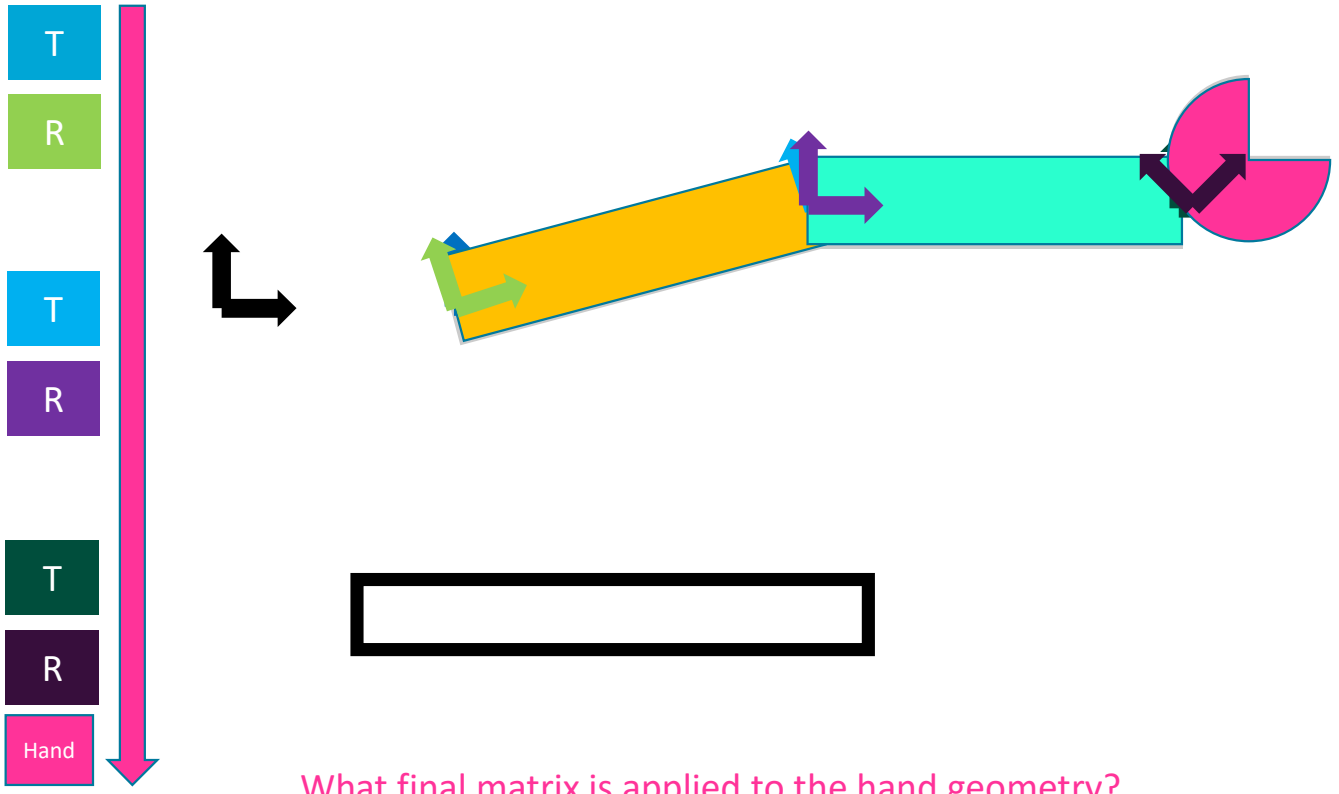
- Build complex object dependencies



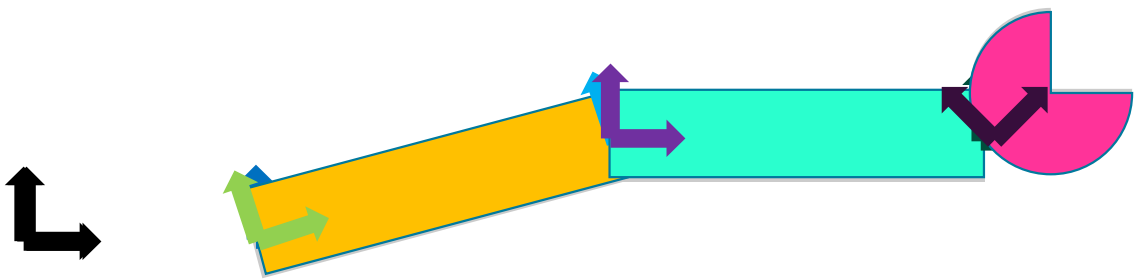
What final matrix is applied to the hand geometry?

Local Interpretation

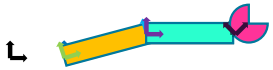
- Build complex object dependencies

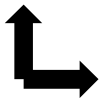


What final matrix is applied to the hand geometry?



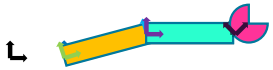
T R T R T R

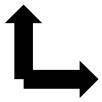




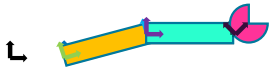
T R T R T R

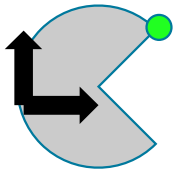
Global Interpretation



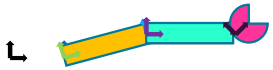


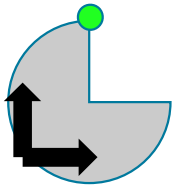
Global Interpretation



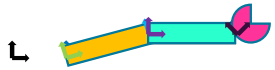


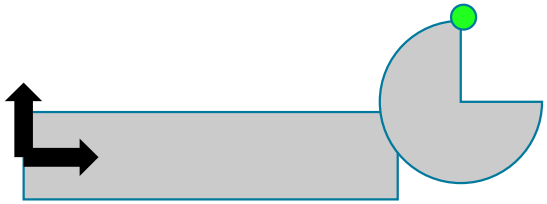
Global Interpretation

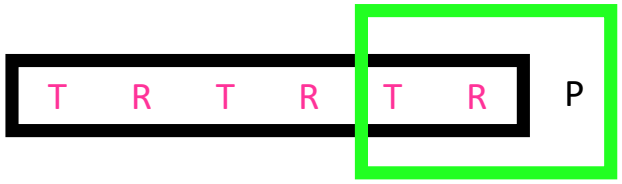




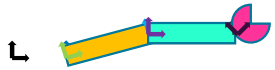
Global Interpretation

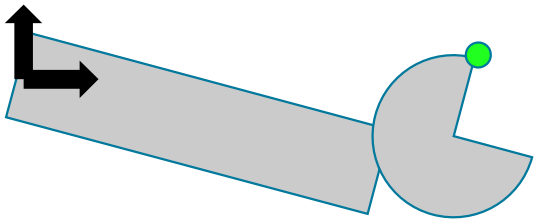


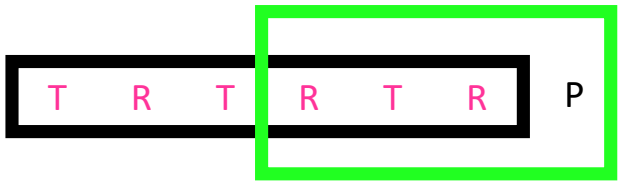




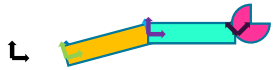
Global Interpretation

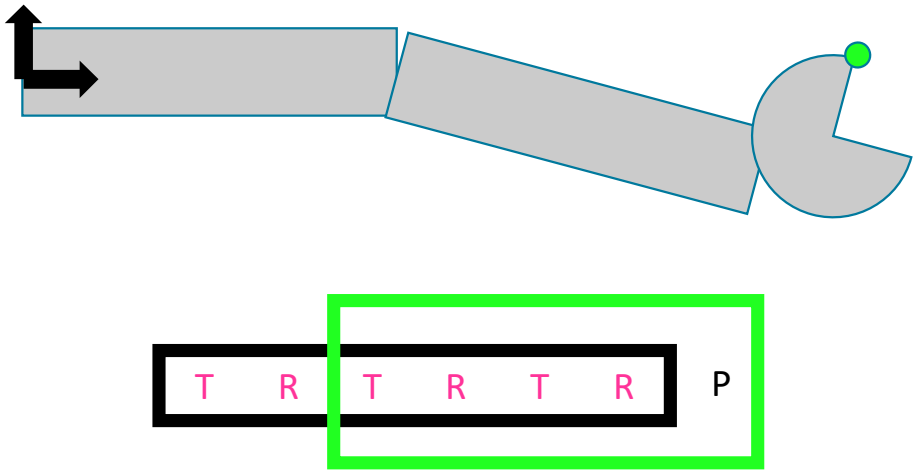




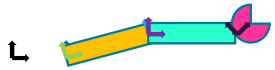


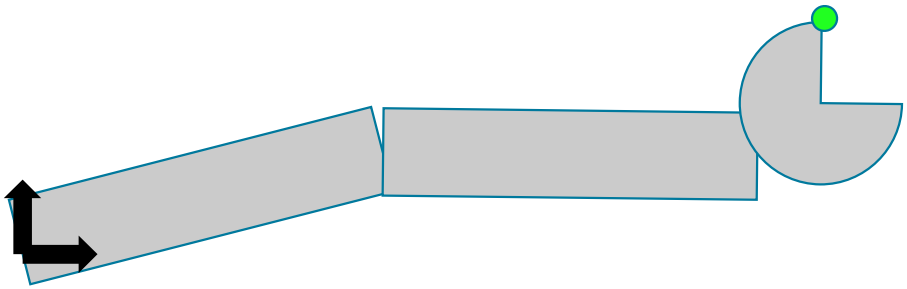
Global Interpretation

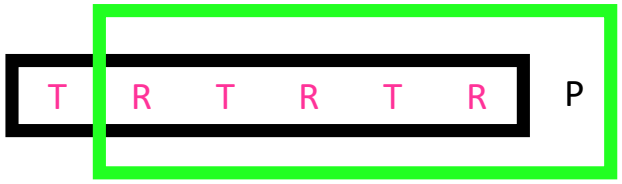




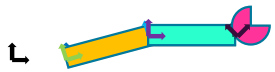
Global Interpretation

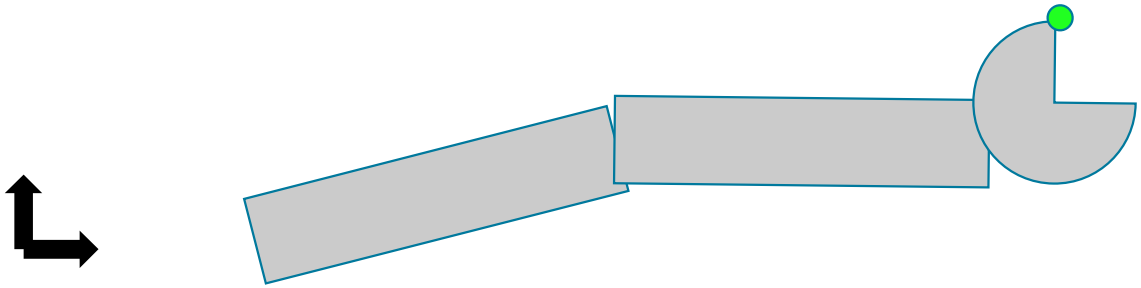




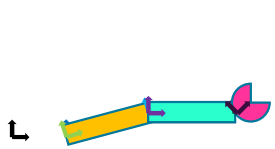


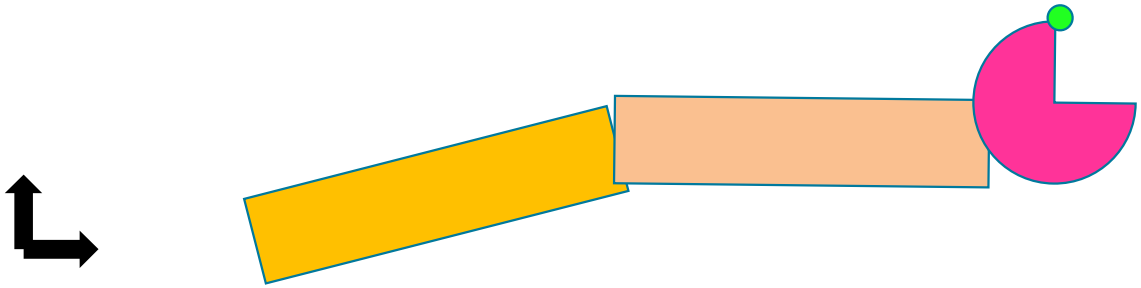
Global Interpretation





Global Interpretation





The point does end up in the expected location

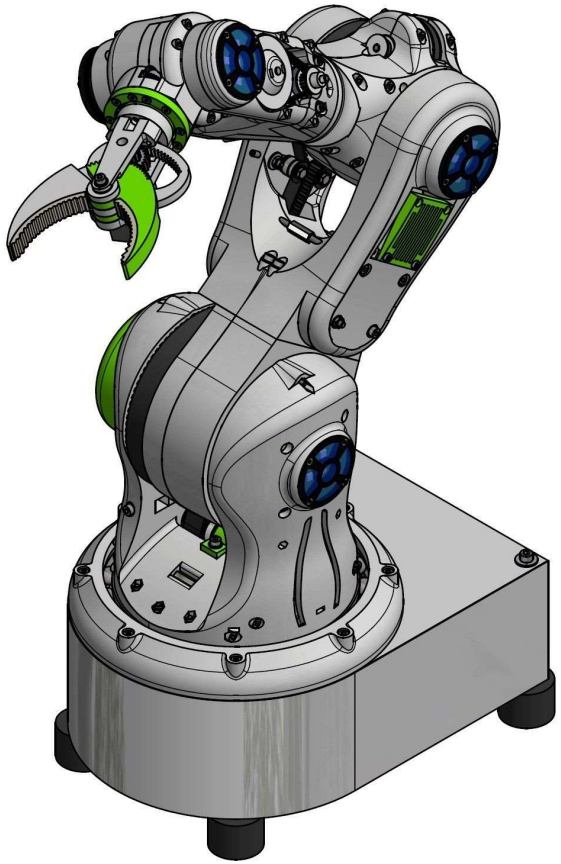
Please note: this is ONE matrix, which makes it very efficient to apply it to all vertices of an object.

Recap Video

Thanks to **Tim Huisman**

Let’s try it out!

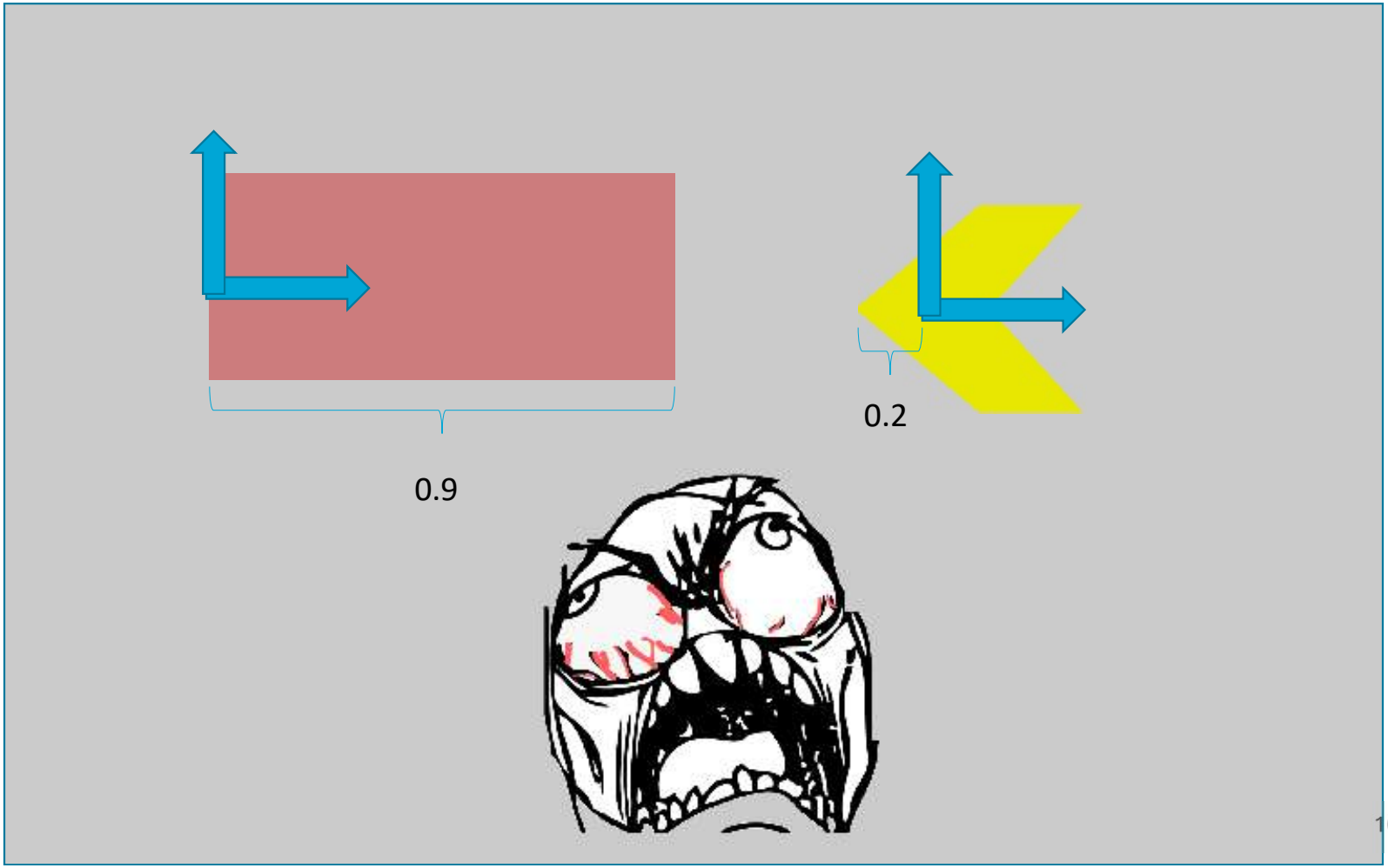
- Professional Robot Arm:



Let’s try it out!

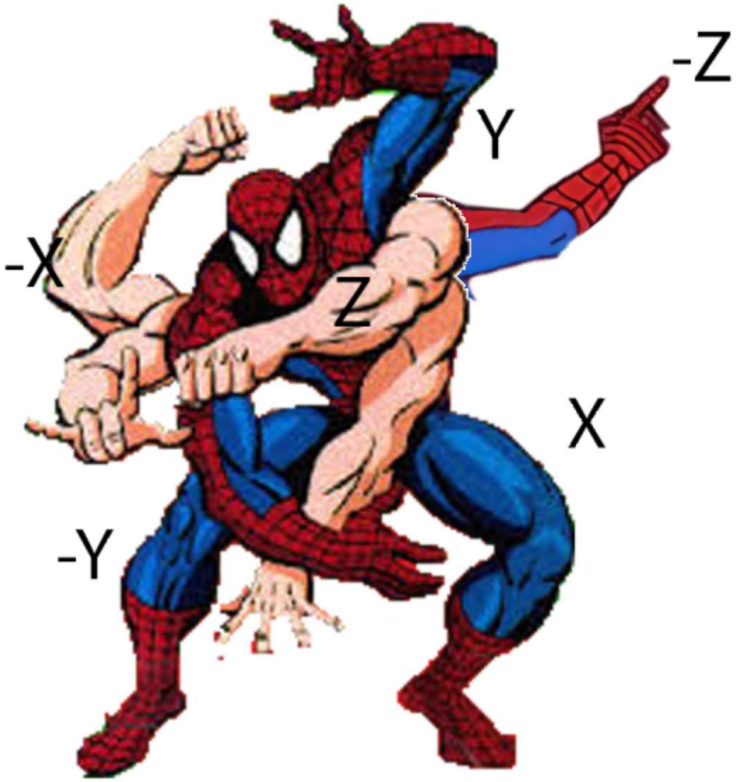
- Professional Robot Arm:

Let's try it out!



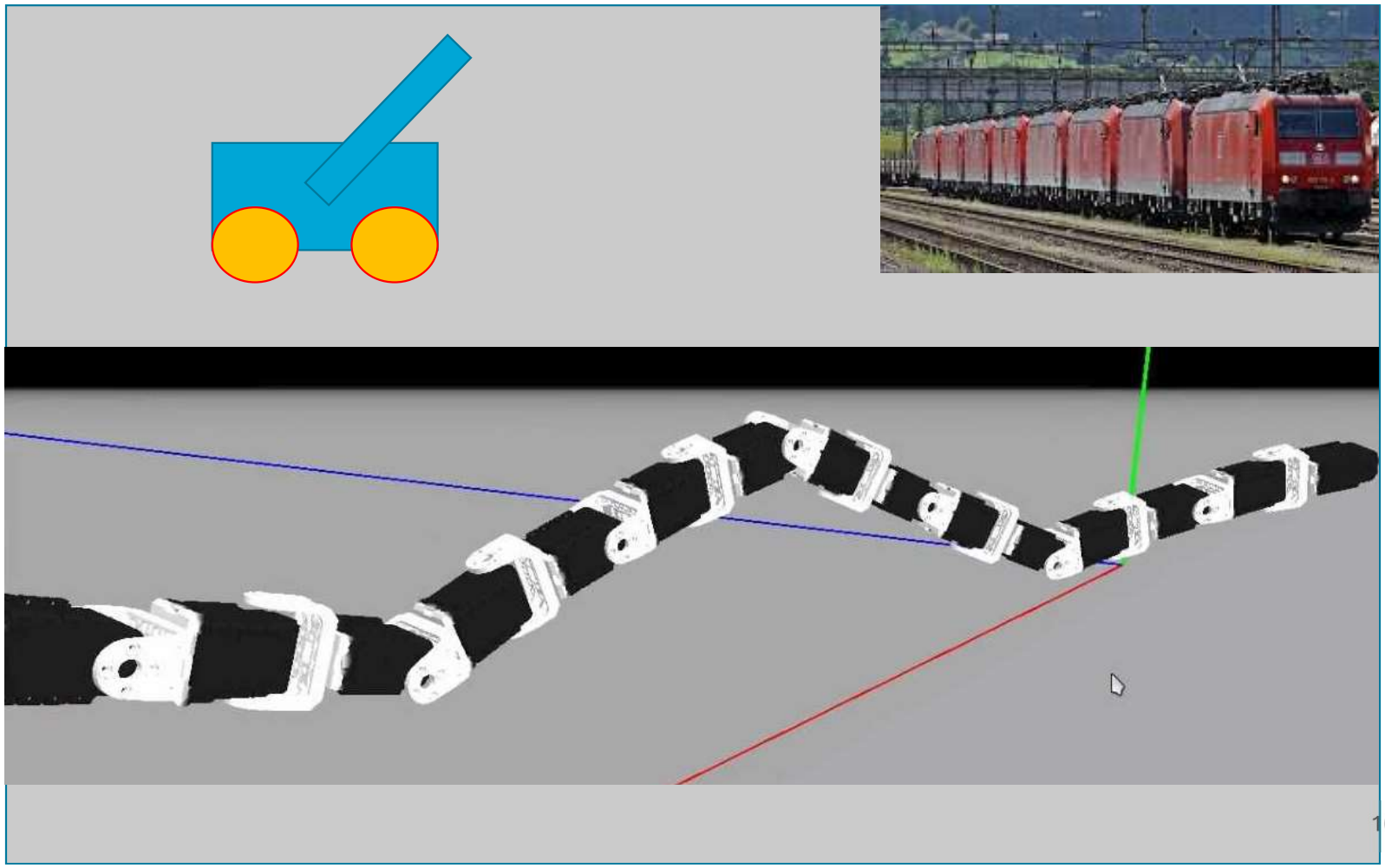
Questions?

Before 3D Computer Graphics

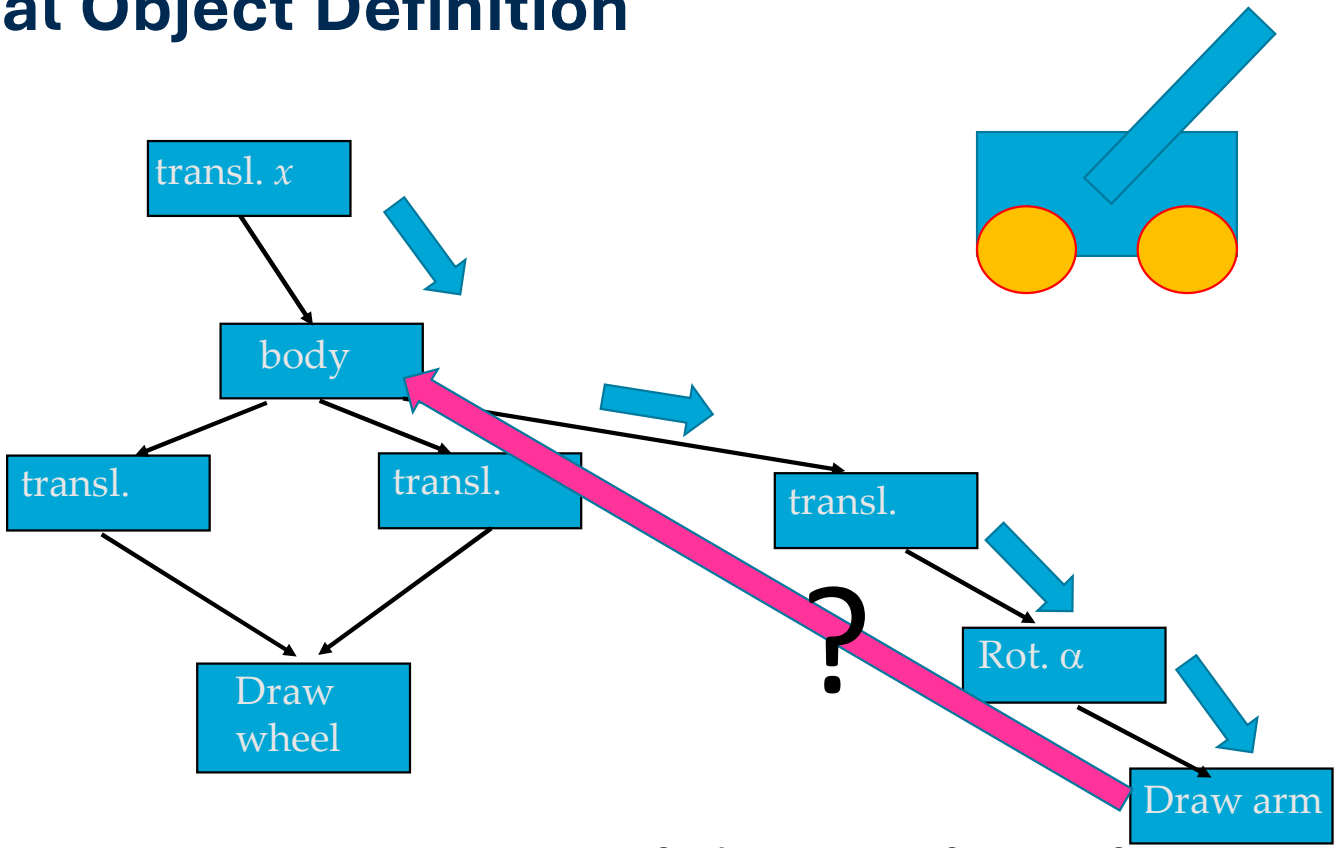


After 3D Computer Graphics

Other examples?



Hierarchical Object Definition



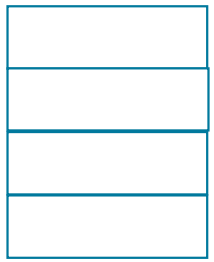
Common representation of objects in form of a tree

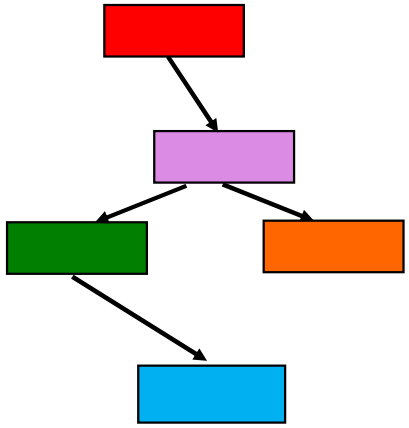
Geometry is reused (e.g., only one wheel is stored)

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:



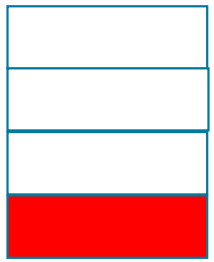


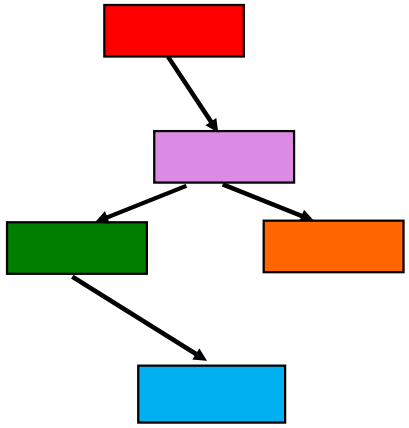
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:





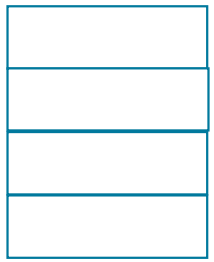
stack.push(root)

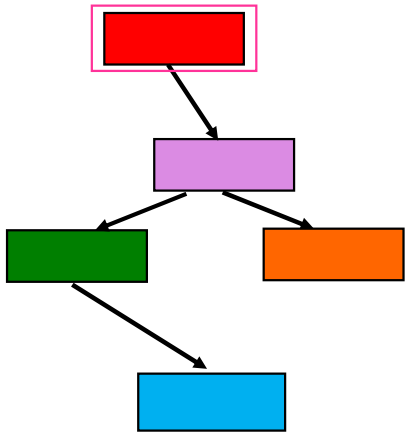
```
while (!stack.empty())
{
        node=stack.pop()
        process(node)//do what you need to do
        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:



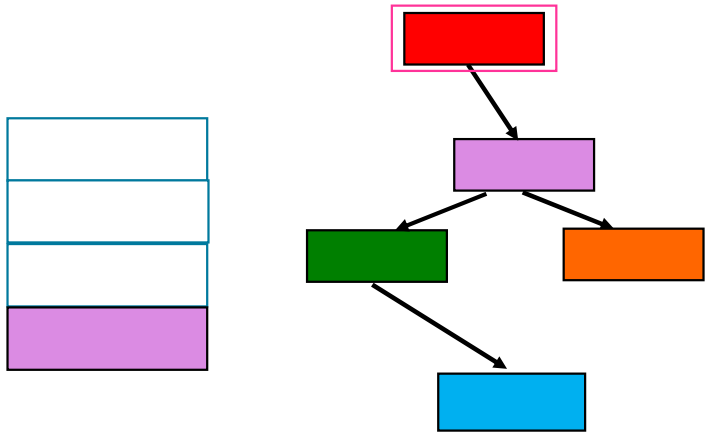


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

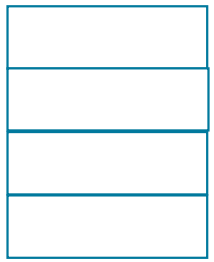


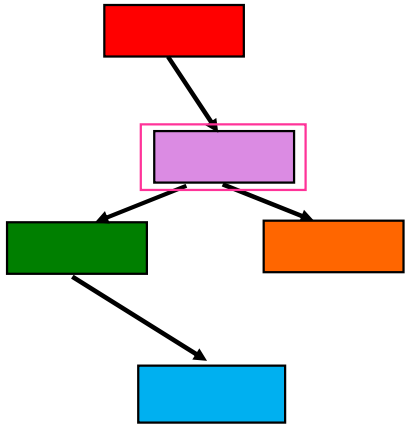
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:



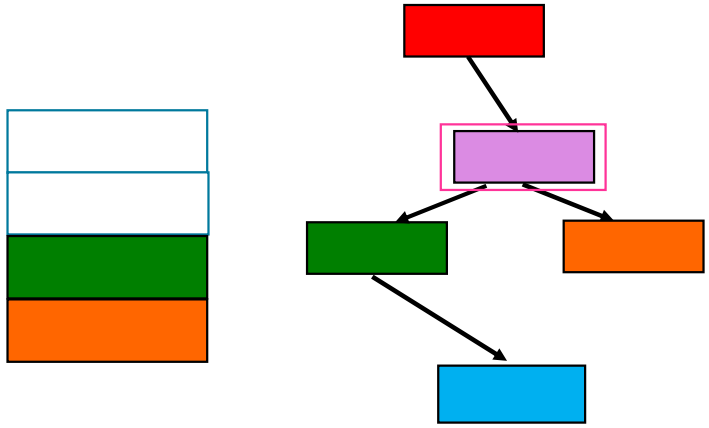


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

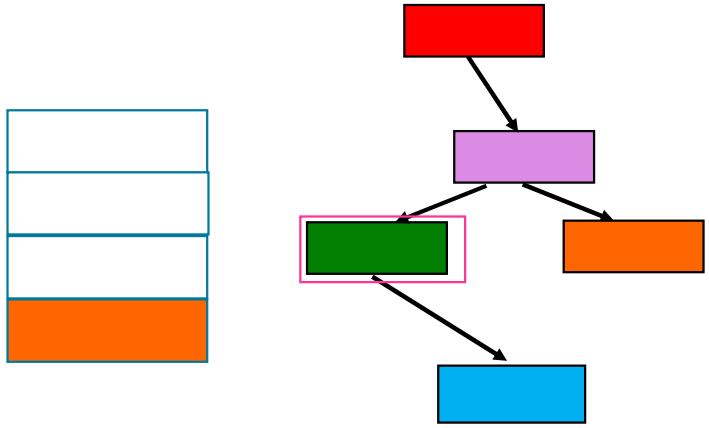


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

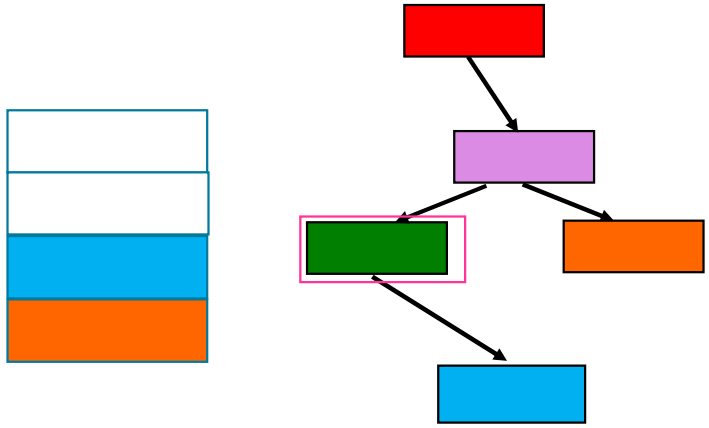


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

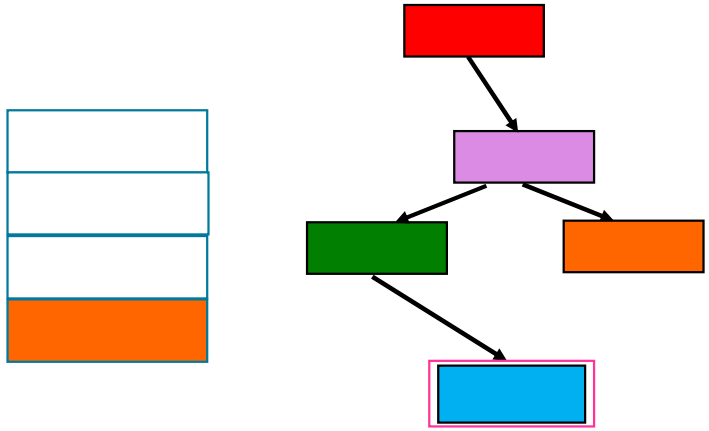


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

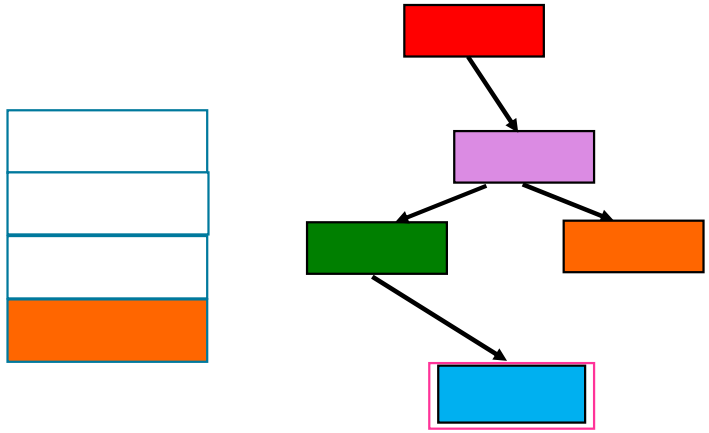


```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:

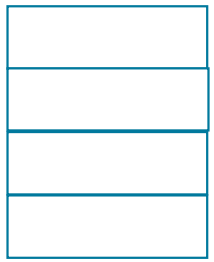


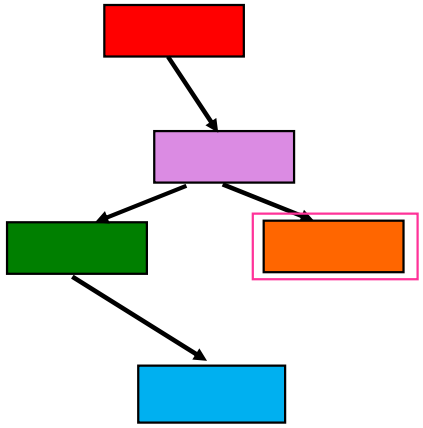
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:



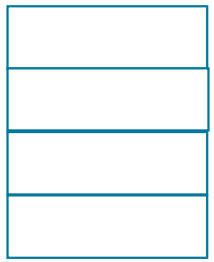


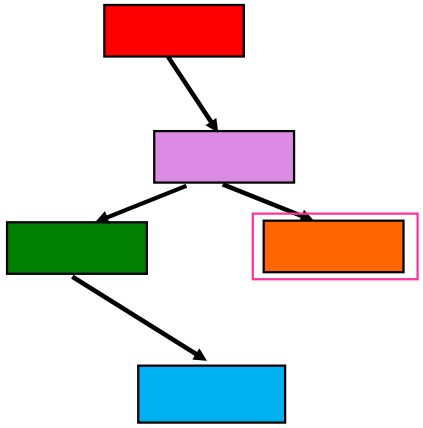
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:



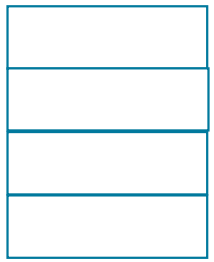


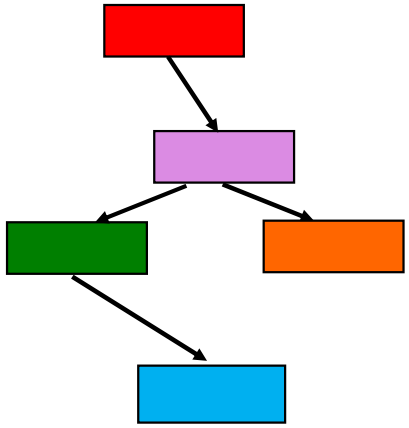
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Excursion: Walking over a tree

Depth-first tree traversal

using a stack:





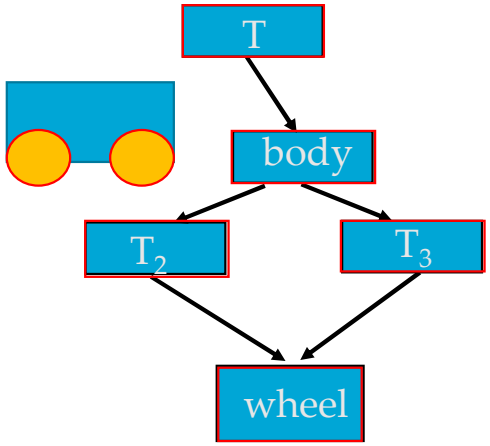
```
stack.push(root)

while (!stack.empty())
{
        node=stack.pop()

        process(node)//do what you need to do

        if (node.hasChildren()) stack.push(node.children())
}
```

Hierarchical Objects

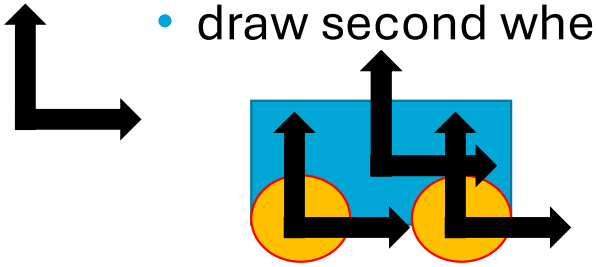
- Walk over tree using depth-first traversal

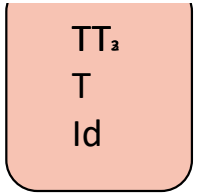
- Next to the “traversal” stack, keep a “matrix” stack that maintains the concatenations (multiplications) of matrices along the way through the tree

Matrix Stack



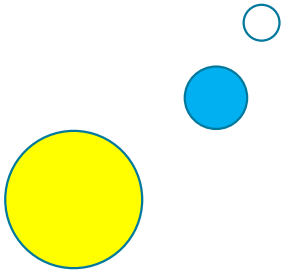
- Parallel to walking over the tree:
- Keep previous matrices on a stack
  - **T** (translation by x) pushMatrix idT=T
  - draw robot body
  - **T_2** (translation to center of 1st wheel) pushMatrix TT_2
  - draw first wheel as circle of center (0,0)
  - return to **T:** popMatrix
  - **T_3** (T_3 translation to center of 2nd wheel) pushMatrix TT_3
  - draw second wheel as circle of center (0,0)



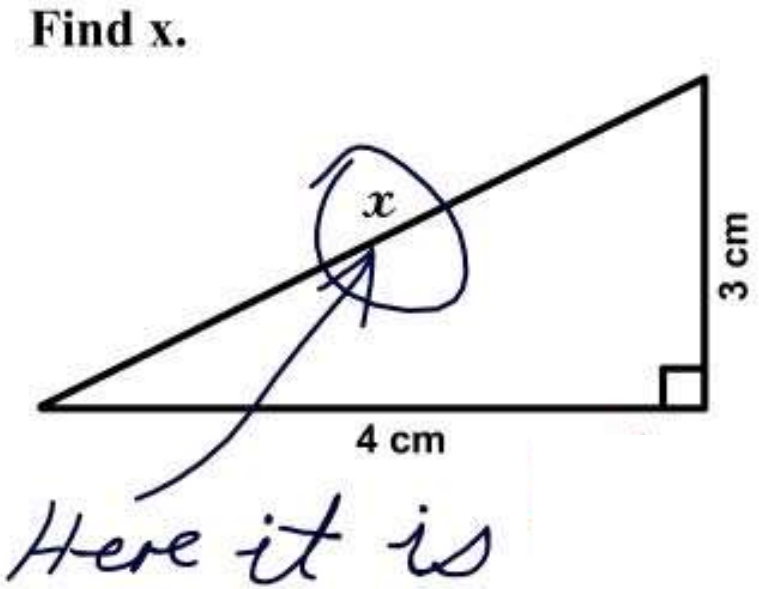


In the practical…

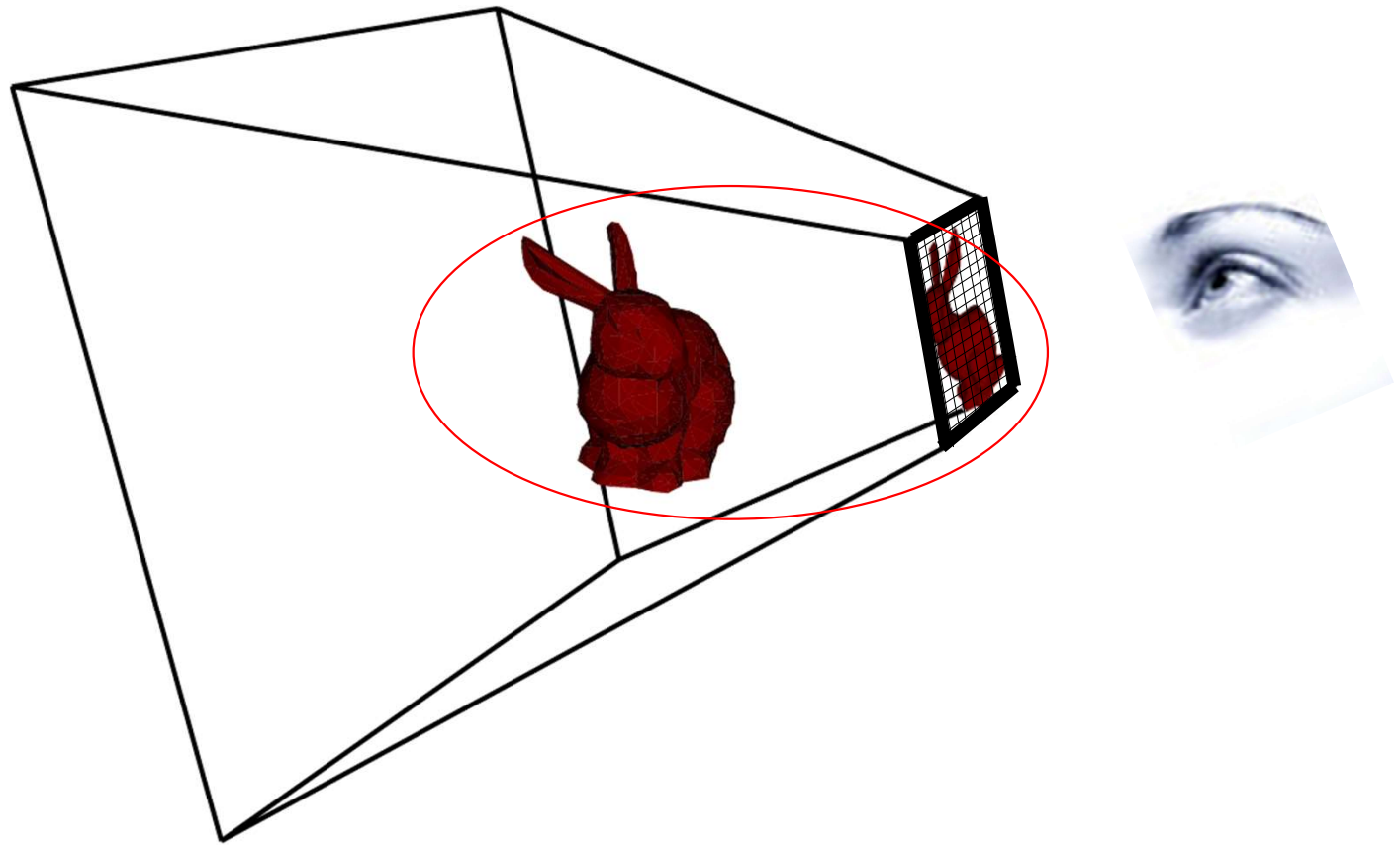
- Define the hierarchy for a solar system:
- Earth rotating around sun
- Moon rotating around earth



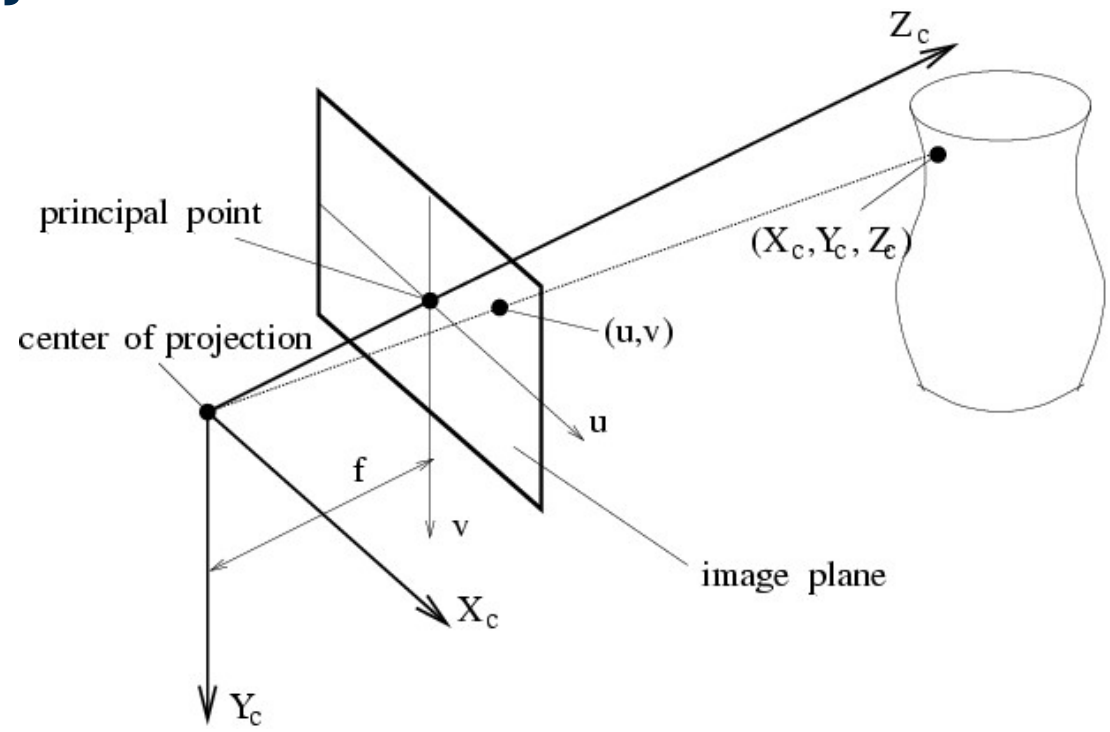
Questions?



One pipeline step is still missing...

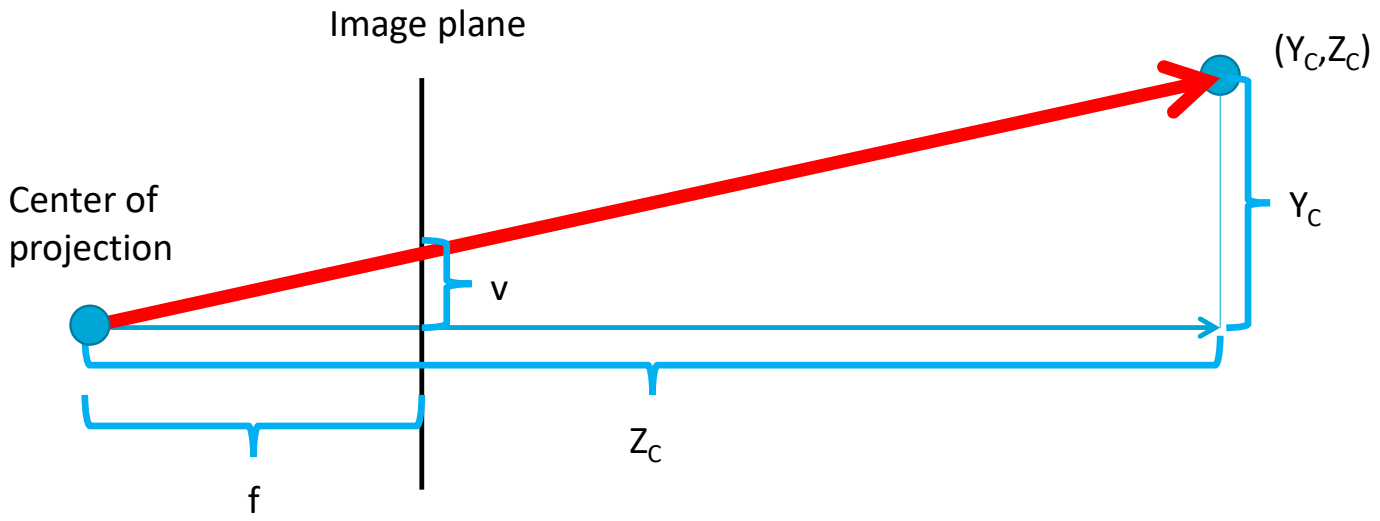


Perspective Projection



Perspective Projection

- sideview: Formula is simple if **camera is at the origin**

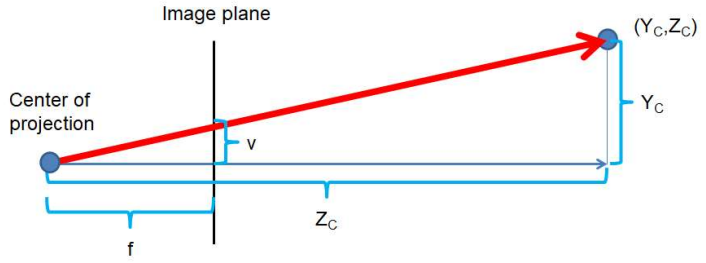


Similar triangles: $v / f = Y_C / Z_C$

Perspective Projection

- Embed the point P in the projective space

$$P = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \Rightarrow \tilde{P} = \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$



$$u = f\frac{X_c}{Z_c} \quad v = f\frac{Y_c}{Z_c}$$

- Look for M such that:

$$M\tilde{P} = \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix}$$

Impossible in standard R^n

What is the problem?

$$M\widetilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

What is the problem?

$$M\widetilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ \dots \\ \\ \end{bmatrix}$$

What is the problem?

$$M\tilde{P} = \begin{bmatrix} a & b & c & d \\ e & q & g & h \\ i & j & m & n \\ k & l & o & p \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} ax + by + cz + dw \\ \dots \end{bmatrix} \Rightarrow \frac{fx}{z} ?$$

$$u = f \frac{X_C}{Z_C}$$

Perspective Projection

- Hint: Think projective!

$$M\widetilde{P} = \begin{pmatrix} fX / Z \\ fY / Z \\ 1 \end{pmatrix}$$

Perspective Projection

- Hint: Think projective!

$$M\widetilde{P} = \begin{pmatrix} fX/Z \\ fY/Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

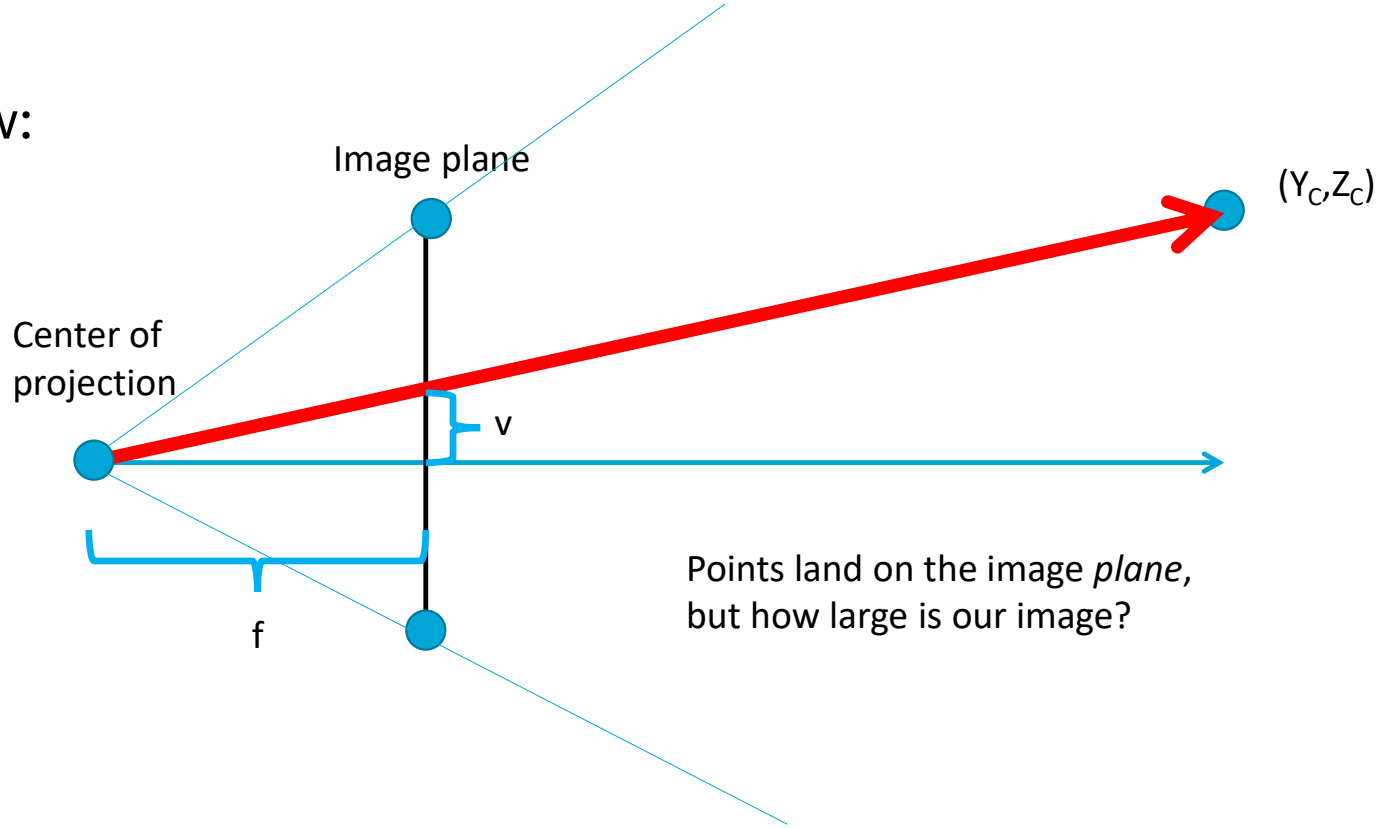
Perspective Projection

- Solution:

$$M = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

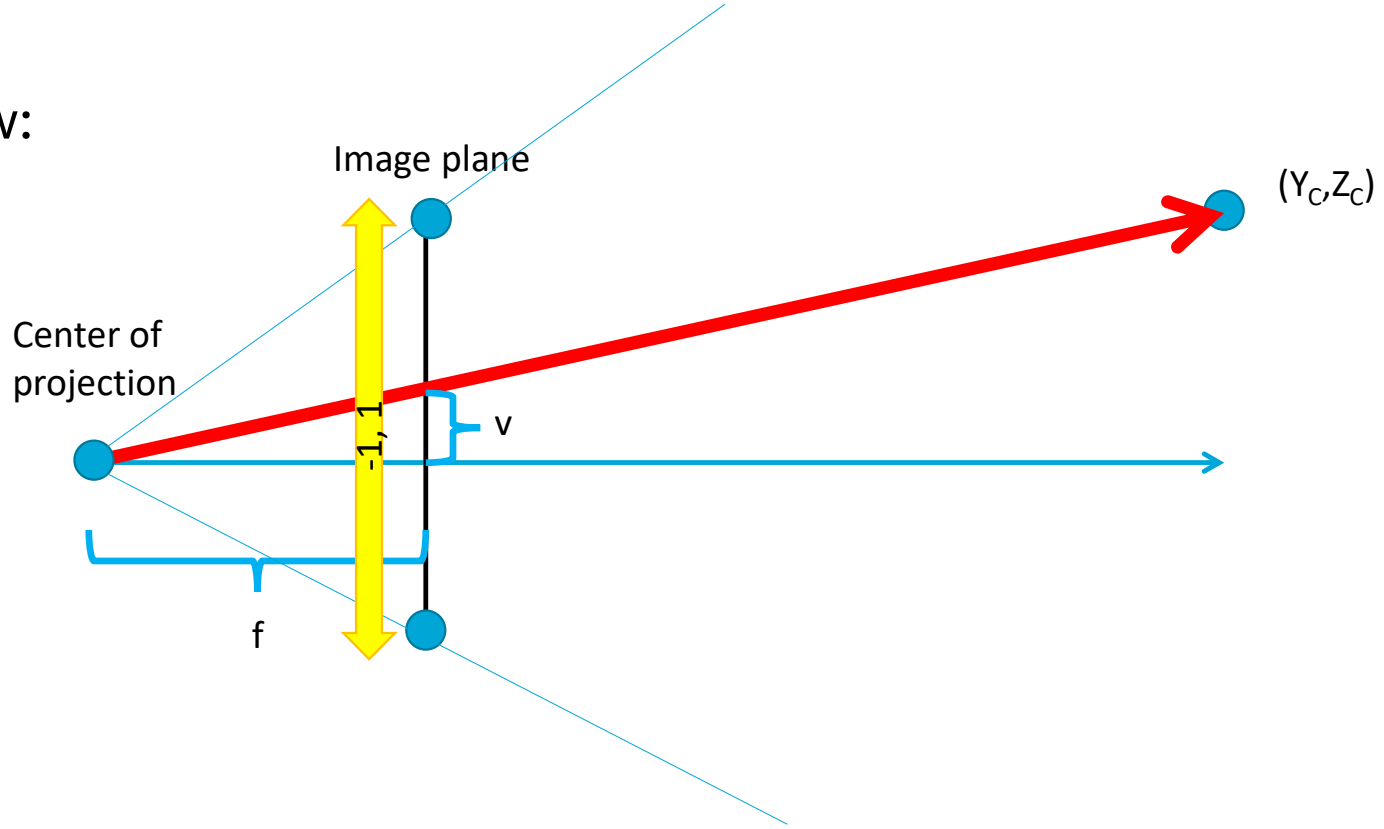
Virtual Camera Model

- sideview:



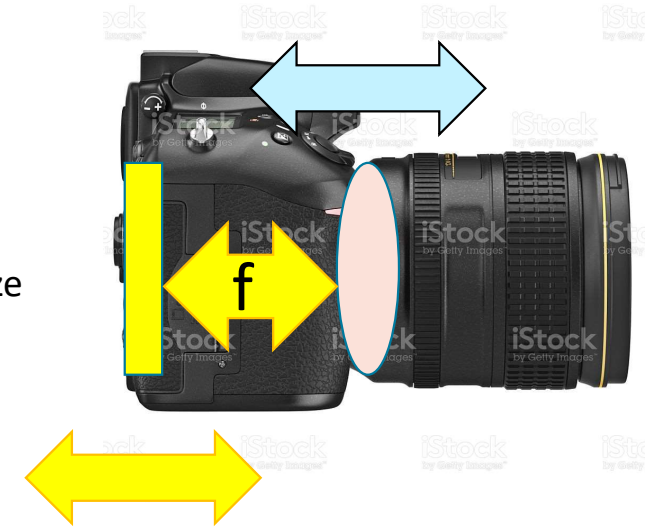
Virtual Camera Model

- sideview:



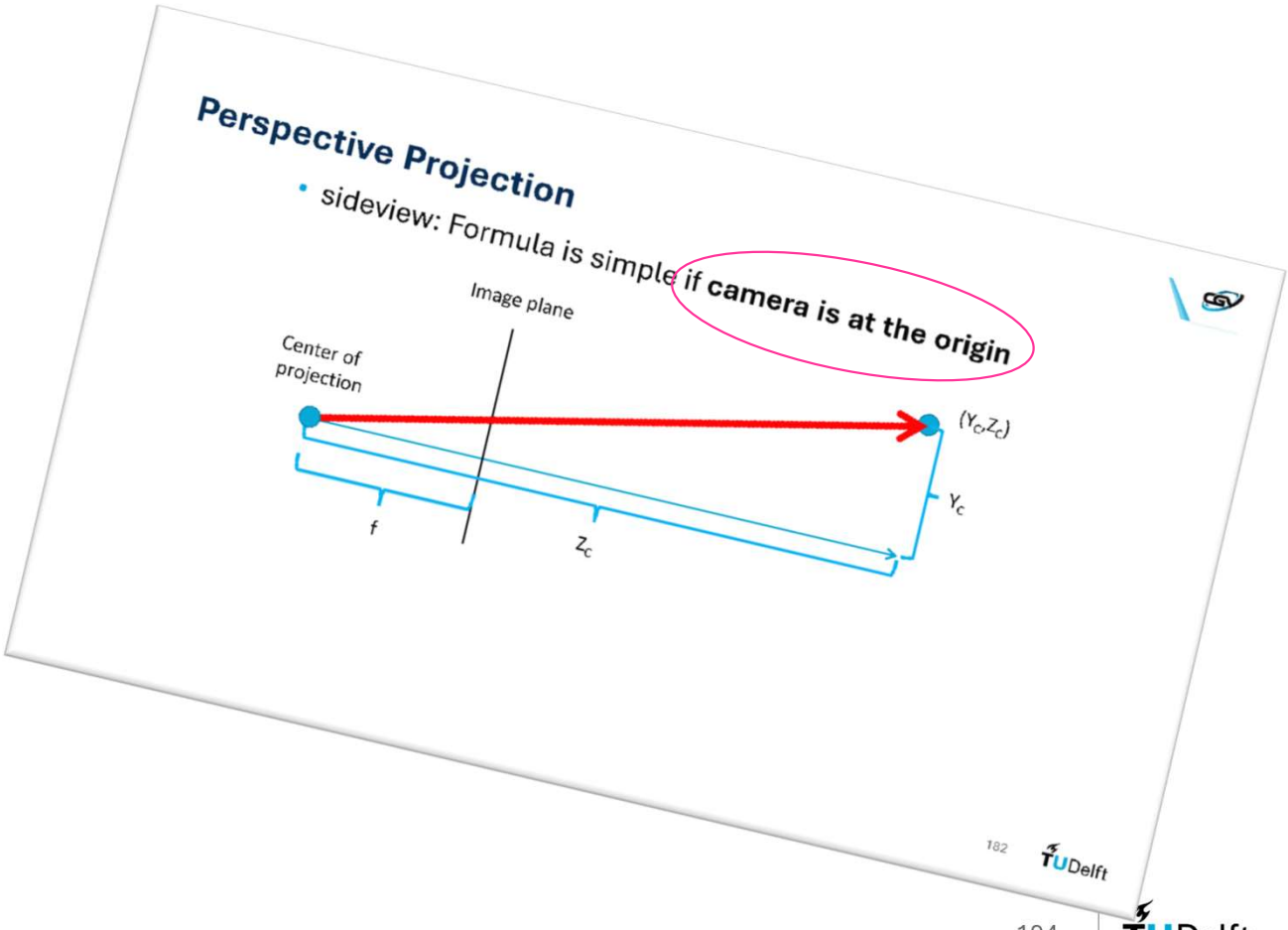
Real camera

Sensor has a fixed size
“[-1,1]”



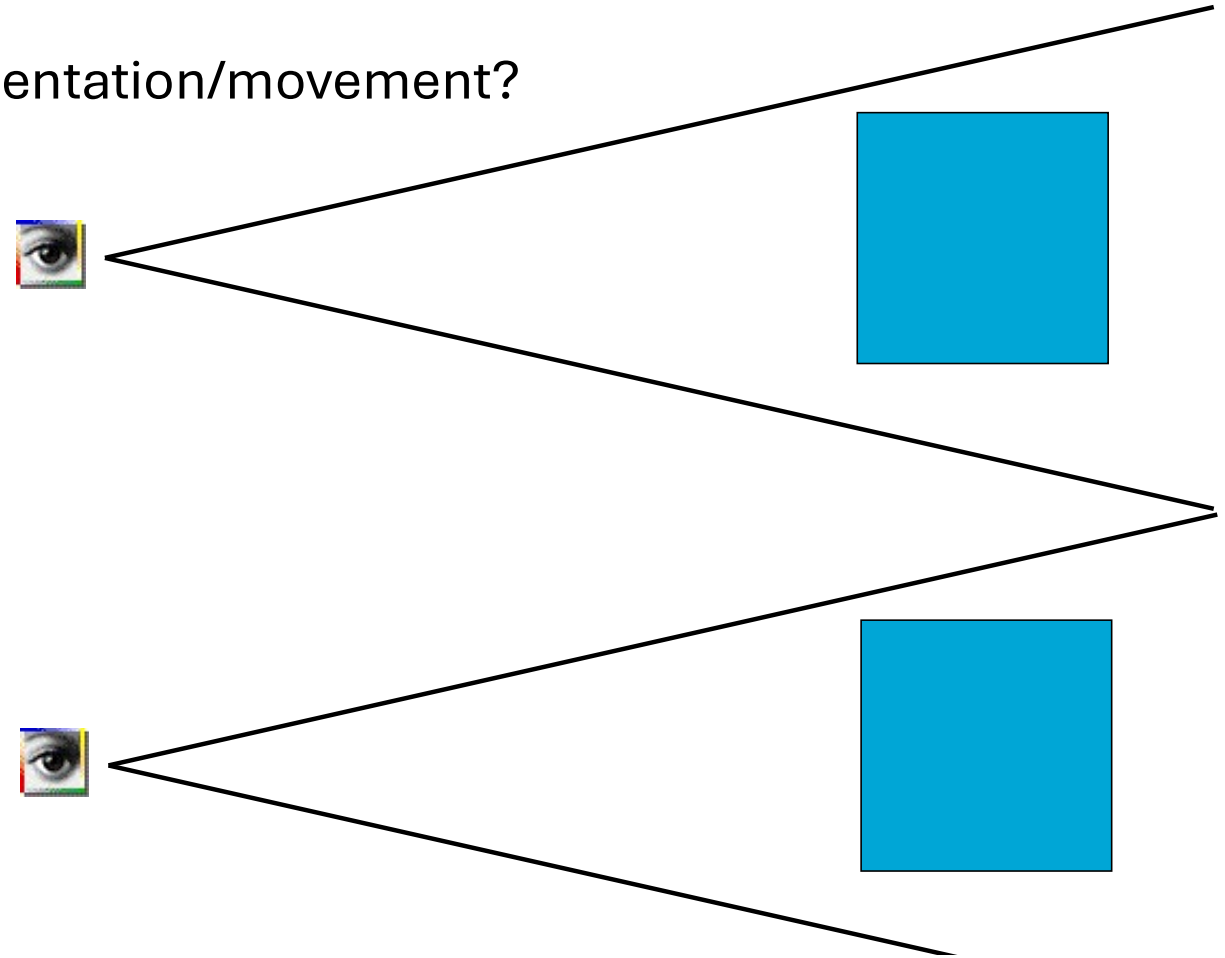
How to move/orient our virtual camera?

- Well, we don’t…



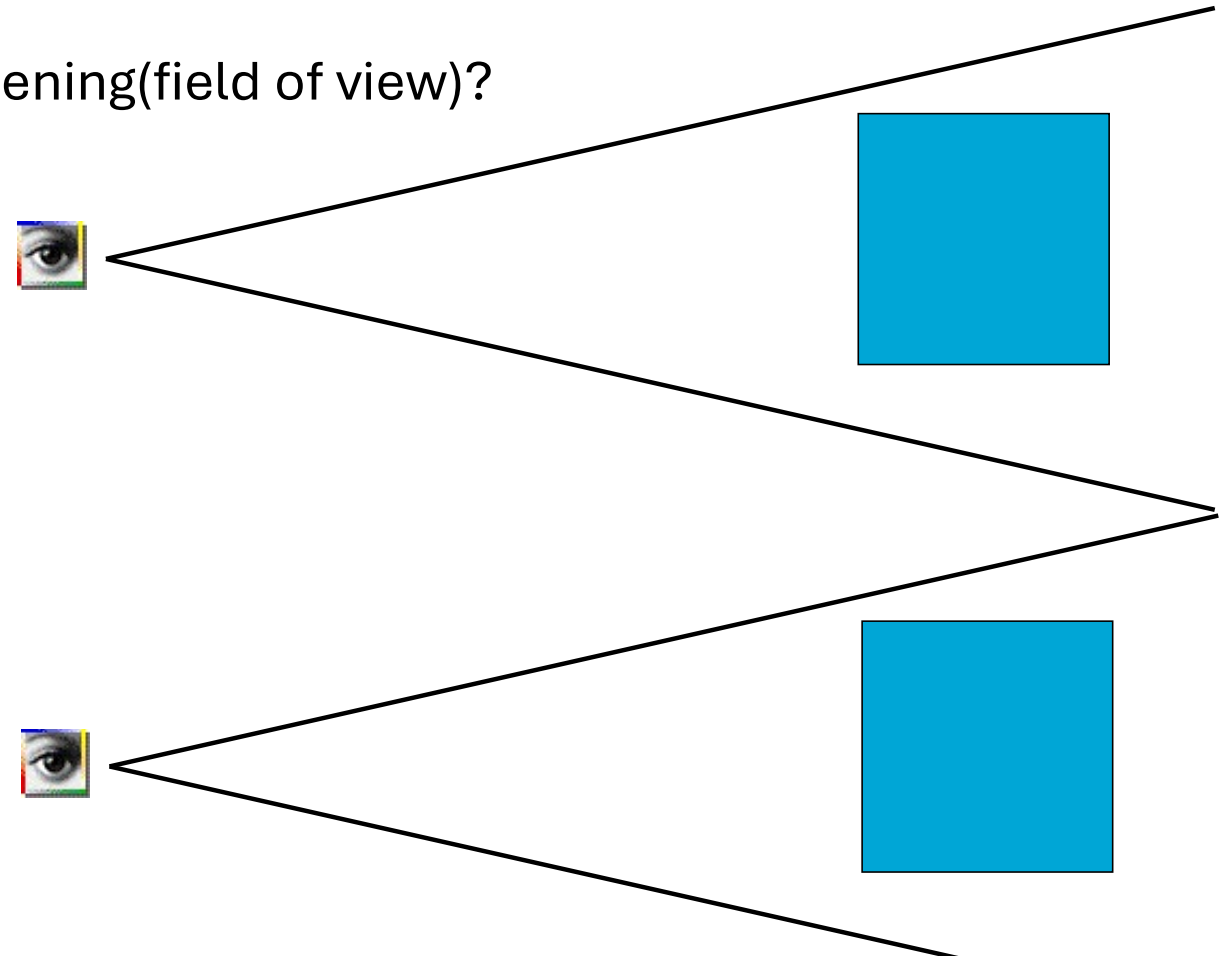
Virtual Camera Model

- Camera orientation/movement?



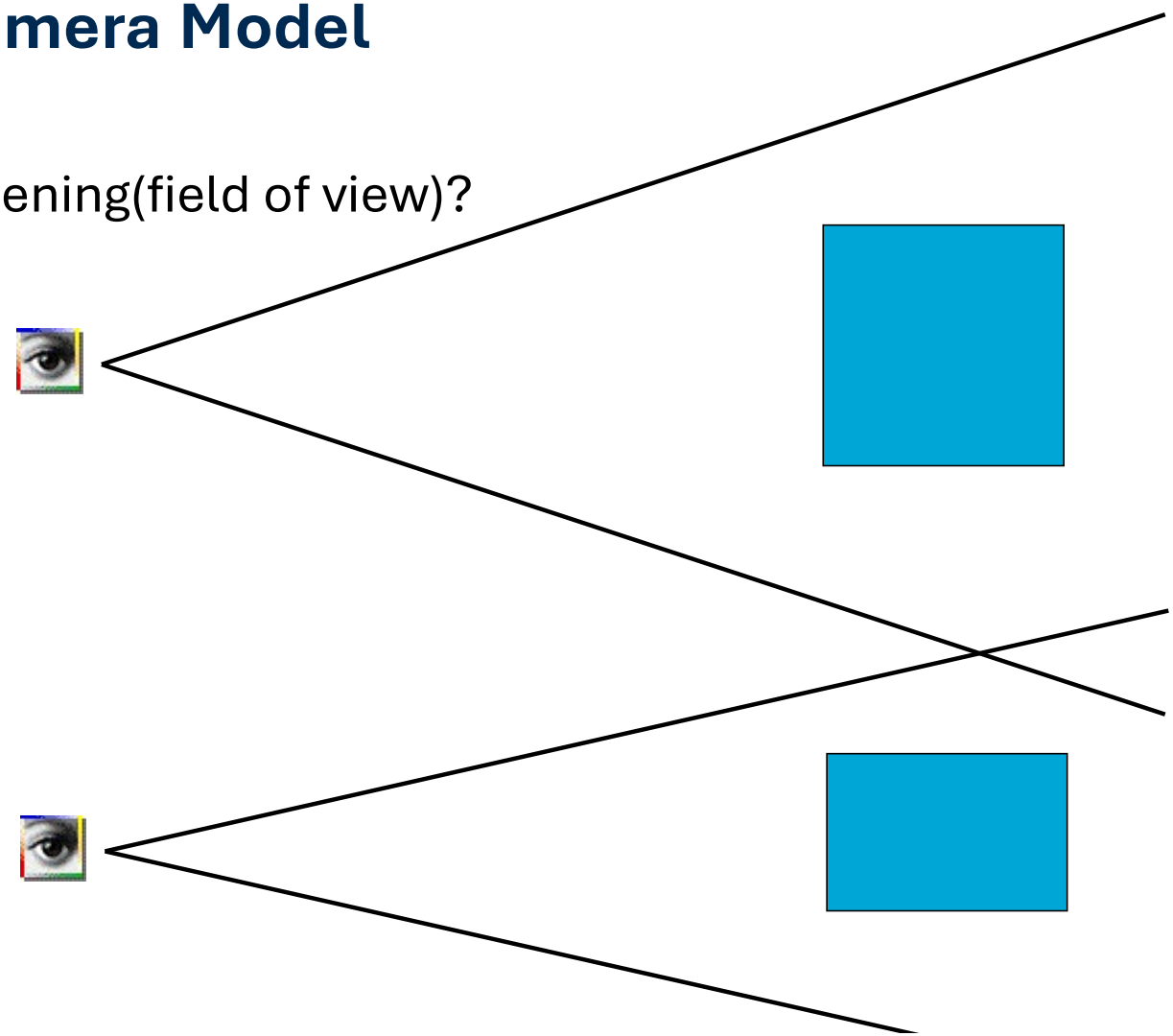
Virtual Camera Model

- Camera opening(field of view)?



Virtual Camera Model

- Camera opening(field of view)?



Virtual Camera Model

- The world literally revolves around the camera!
- Just deform the scene in the “right” way and we can always assume that
  - Camera centre (eye) is at the origin
  - Camera is oriented along the z-axis
  - Image aligned with x-y and has unit size $[-1,1]^2$
  - Helper functions map intuitive parameter to a matrix.

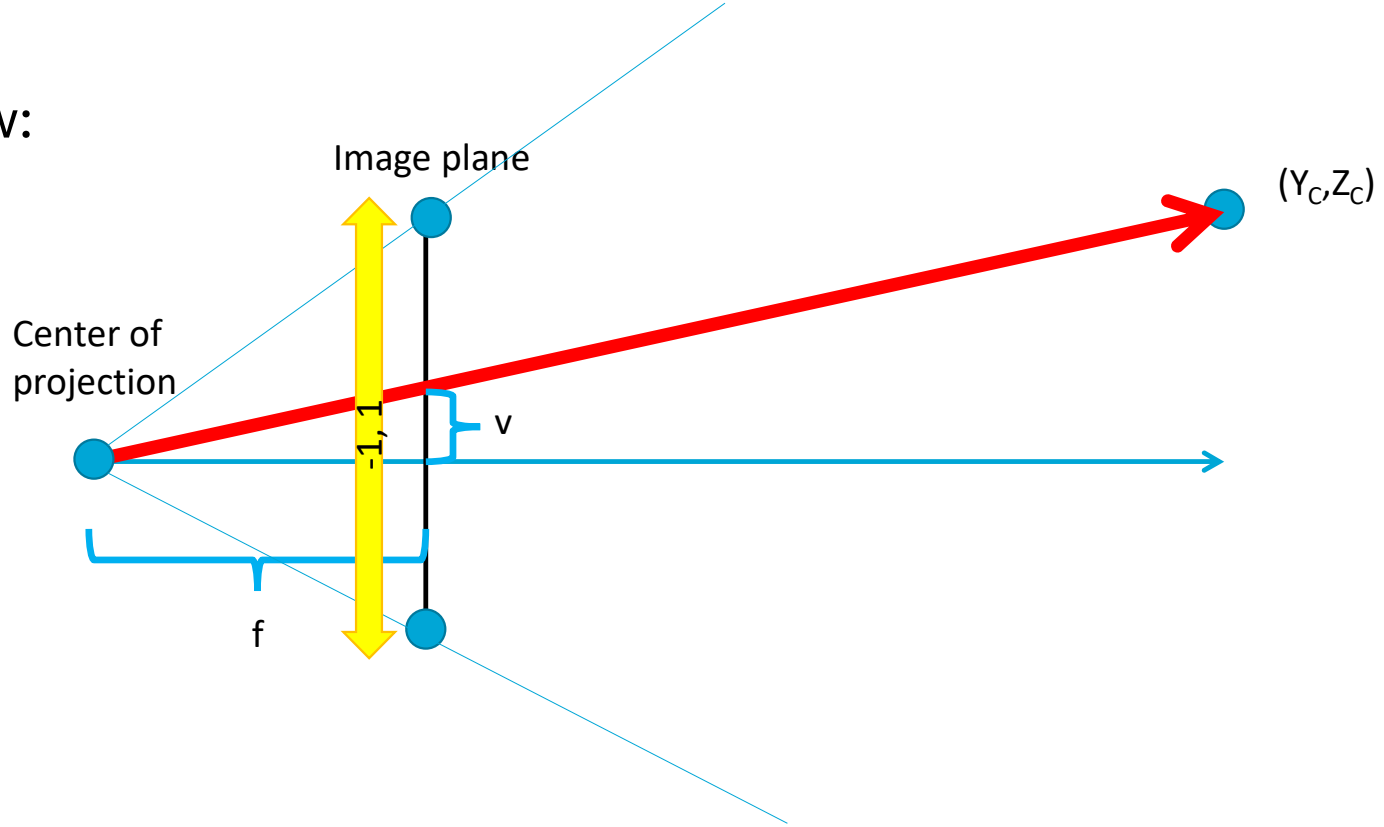
projection

orientation/location

$$\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

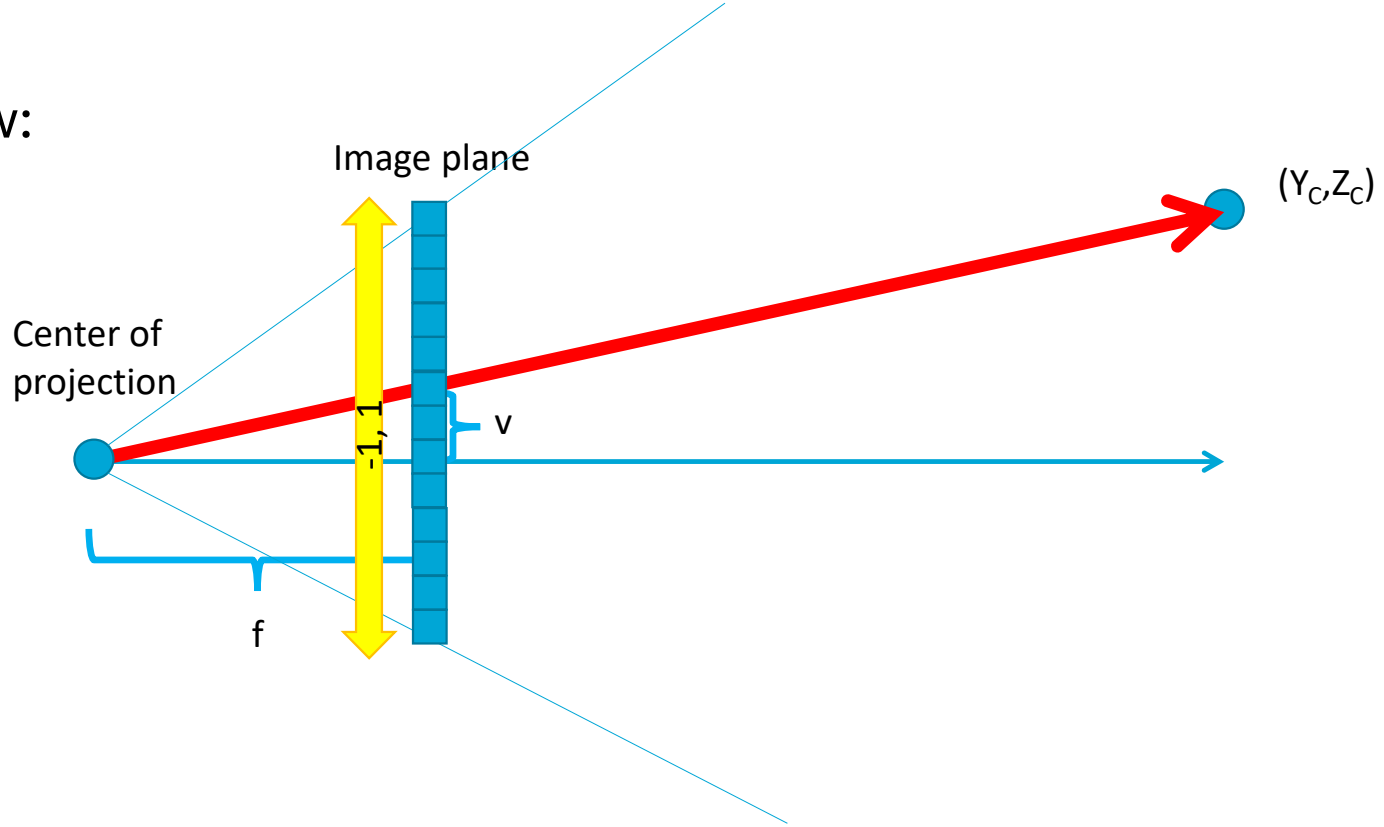
Virtual Camera Model mapping to pixels

- sideview:



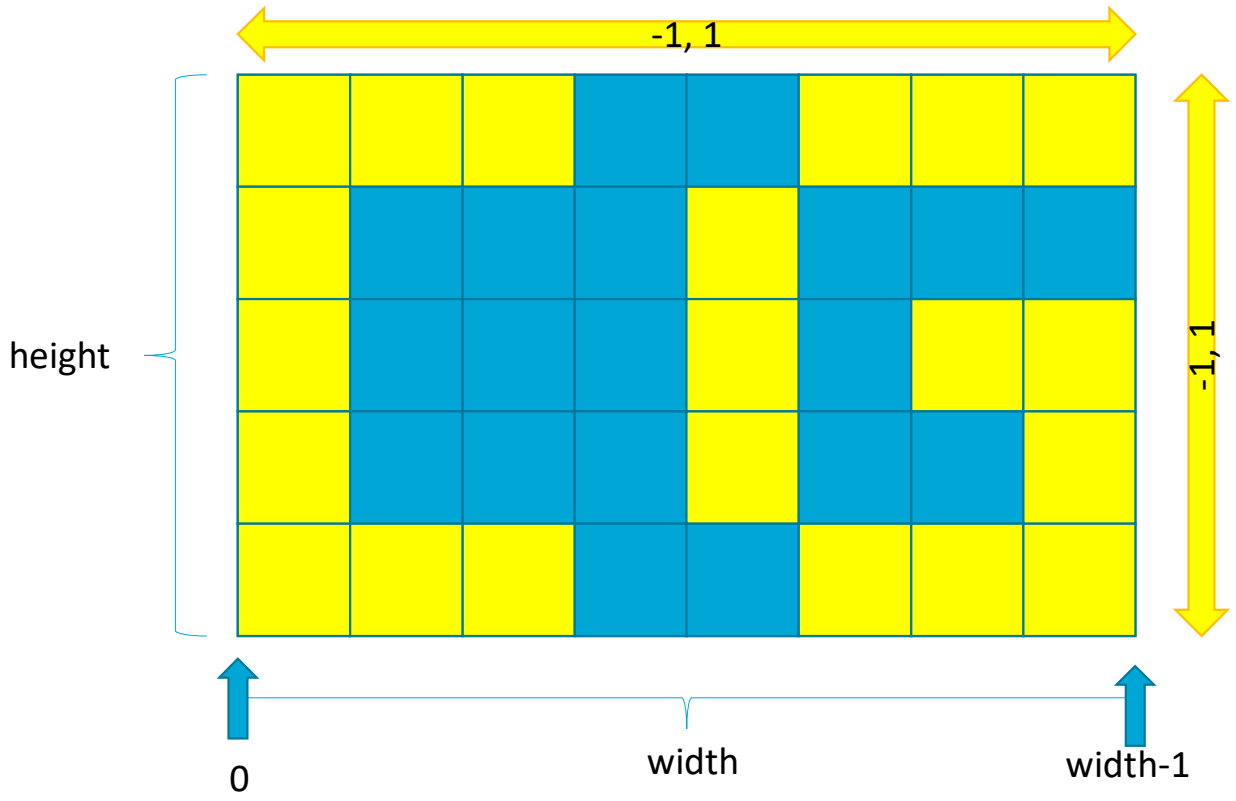
Virtual Camera Model mapping to pixels

- sideview:



Mapping to pixels: Viewport Transformation

- (-1,1) x (-1,1) ➡ (0, width-1)x(0,height-1)



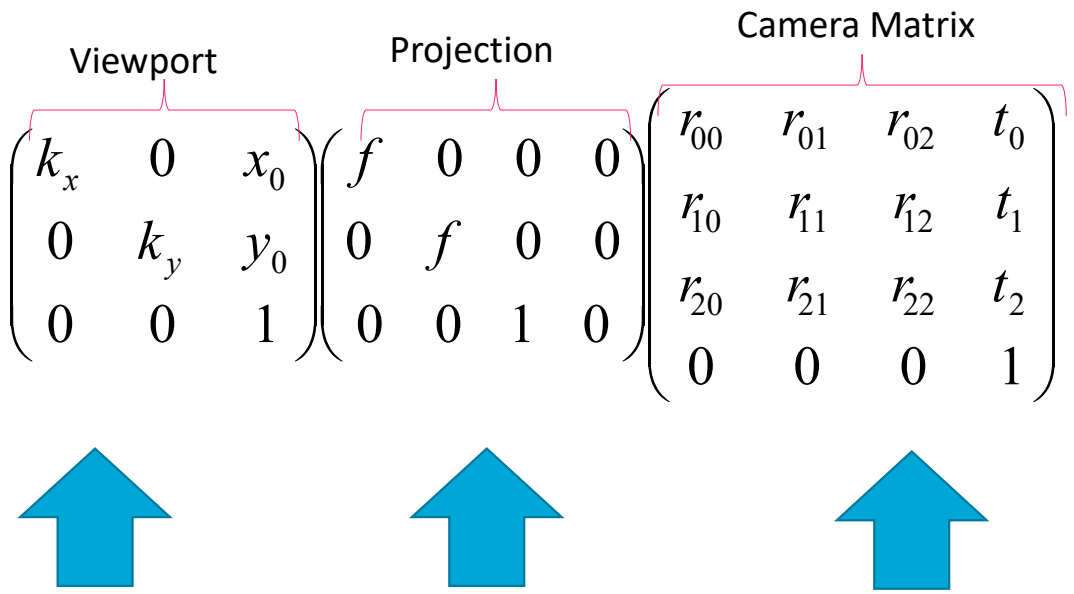
Homework:
Find a matrix to do this mapping

Hint:
it has the form

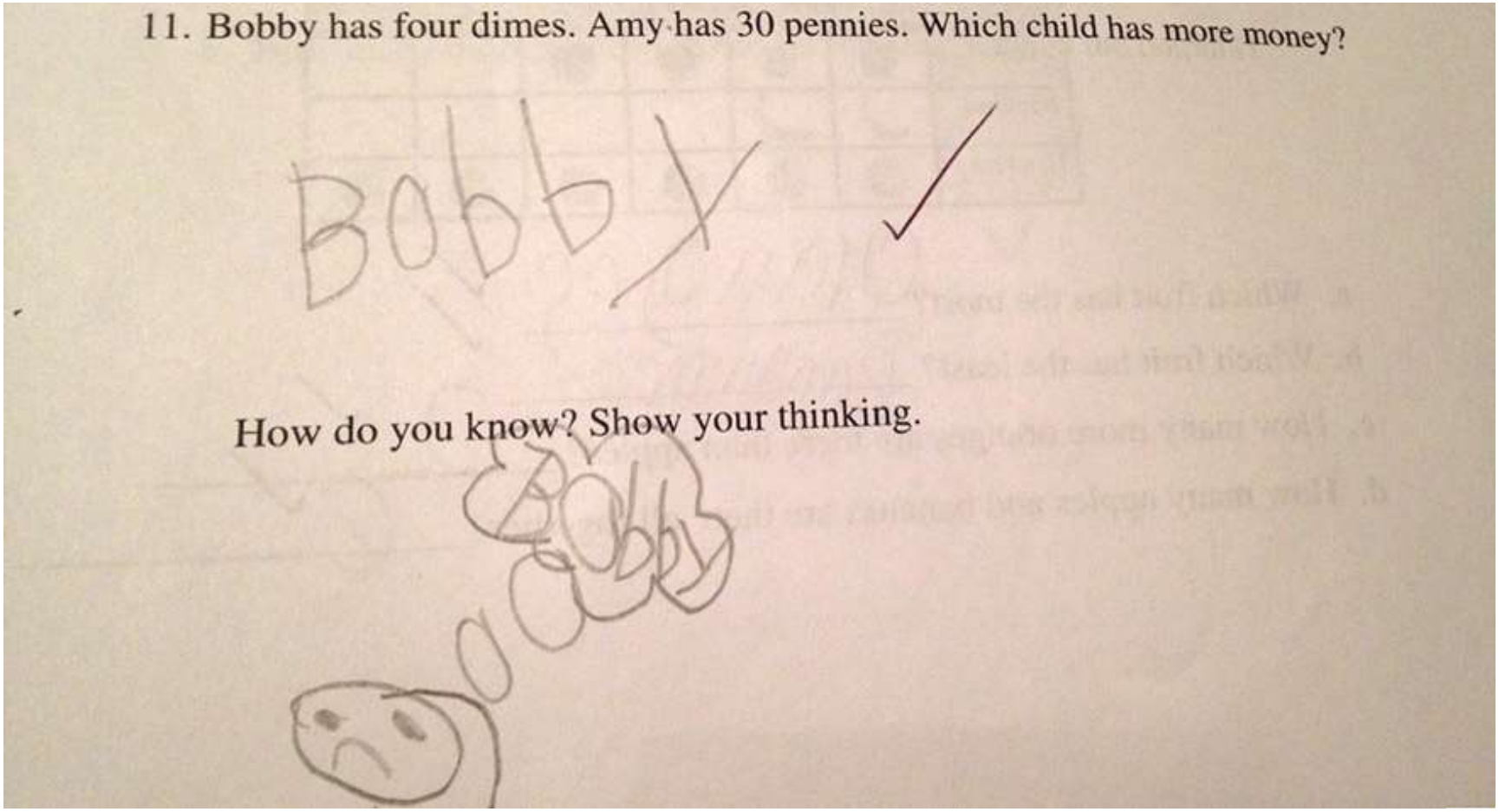
$$\begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Complete Camera Model

- Finally:



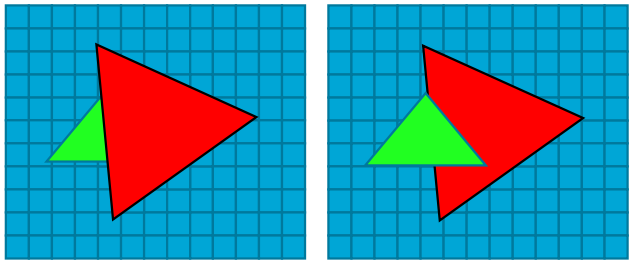
Questions?


11. Bobby has four dimes. Amy has 30 pennies. Which child has more money?
Bobby
How do you know? Show your thinking.

It could have been so simple...

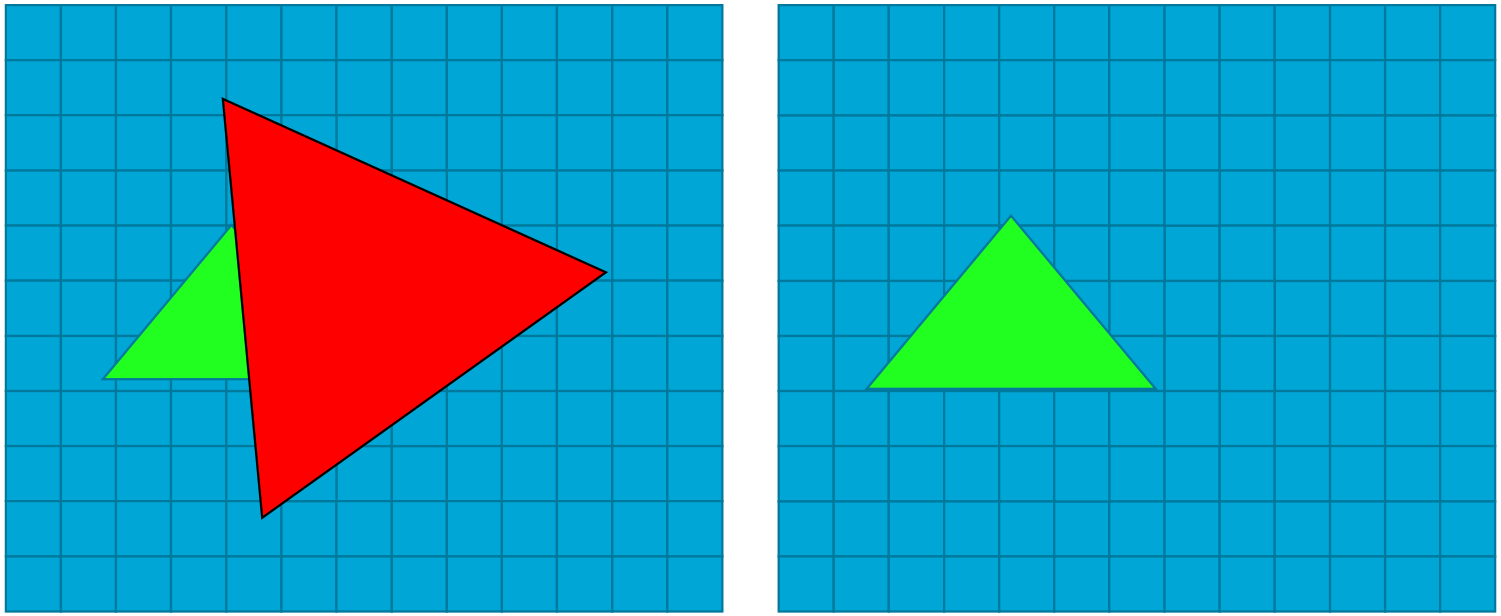
$$P = \underbrace{\begin{pmatrix} k_x & 0 & x_0 \\ 0 & k_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{image}} \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{\text{projection}} \underbrace{\begin{pmatrix} r_{00} & r_{01} & r_{02} & t_0 \\ r_{10} & r_{11} & r_{12} & t_1 \\ r_{20} & r_{21} & r_{22} & t_2 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{orientation/location}}$$

- What is the problem of this matrix for the Graphics Pipeline?



It could have been so simple…

- Catch: Triangle drawing order changes result

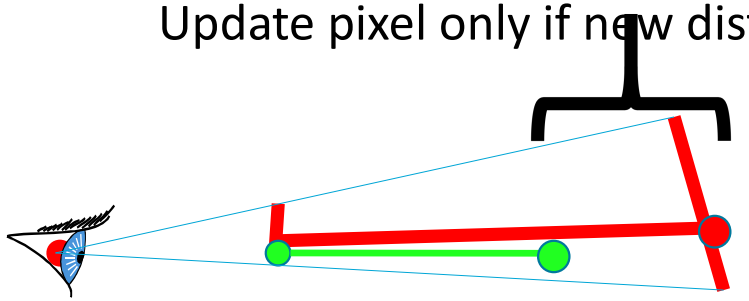


- We need a Z value per pixel!

It could have been so simple…

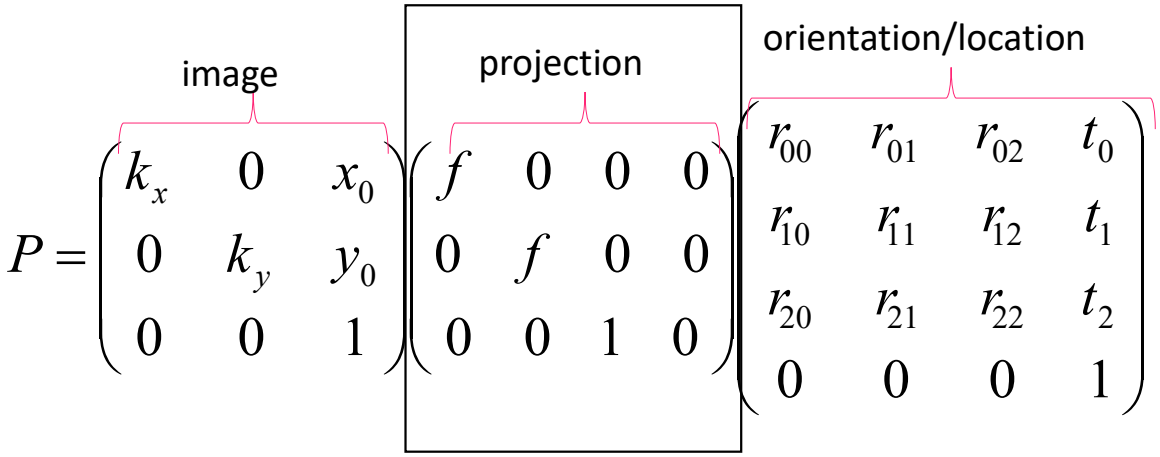
- We need to keep a Z coordinate!

Compare new distance to stored distance
Update pixel only if new distance is nearer



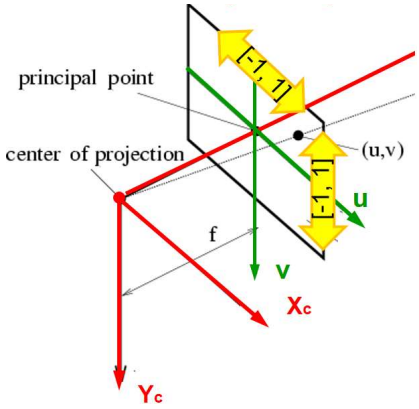
Depth value

It could have been so simple...

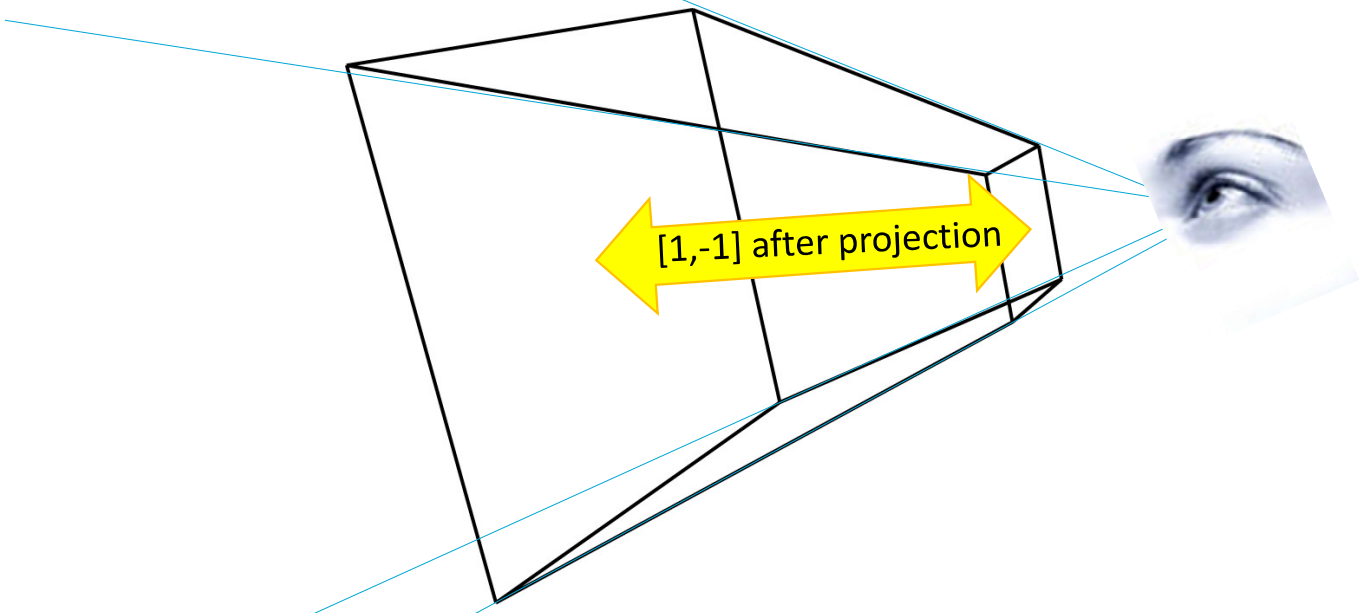


- It was eliminated in this projection matrix...
- To get it back, we need to extend this matrix

Problem:

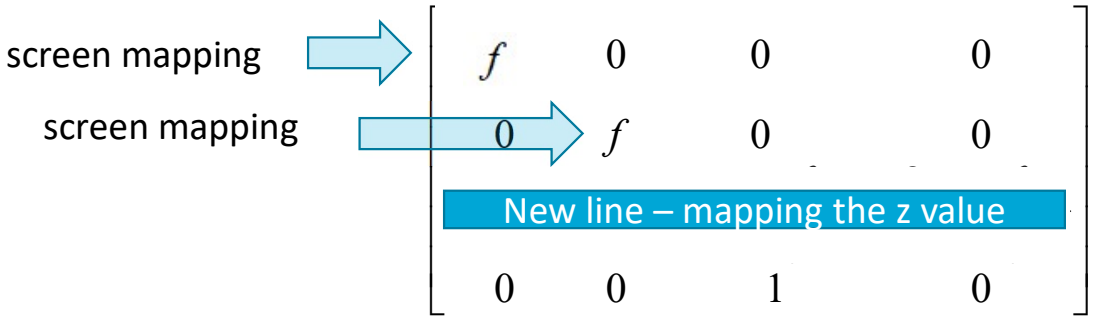


- A 3D scene is infinite…
- How do we represent Z?
- Solution add a **near and far** clipping plane!



The OpenGL Projection Matrix

- Projection matrix:



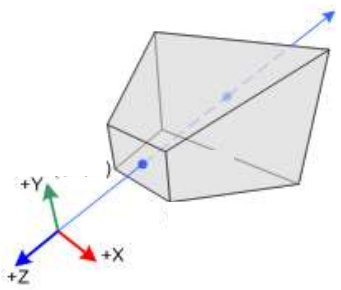
The OpenGL Projection Matrix

- Projection matrix:

screen mapping

screen mapping

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \text{New line – mapping the z value} & & & \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



By OpenGL convention, camera looks along the negative z-Axis
…don't bother too much about it…

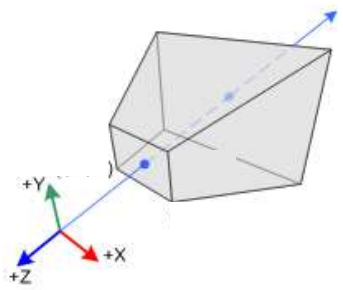
The OpenGL Projection Matrix

- Projection matrix:

screen mapping

screen mapping

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \text{New line – mapping the z value} & & & \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



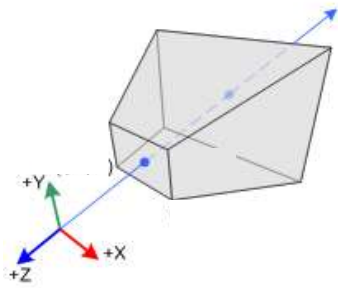
The OpenGL Projection Matrix

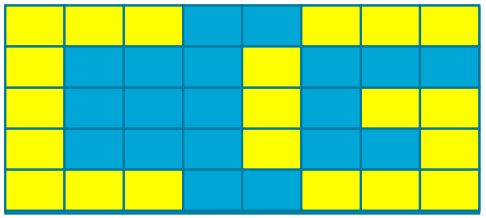
- Projection matrix:

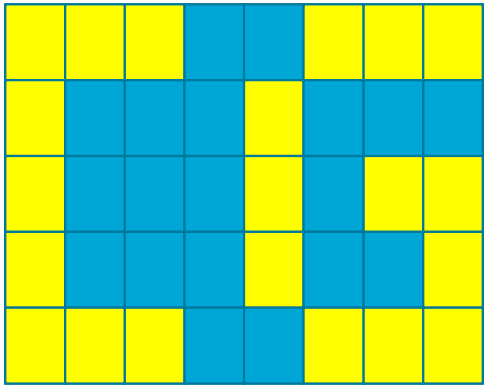
screen mapping

screen mapping

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ \text{New line – mapping the z value} & & & \\ 0 & 0 & -1 & 0 \end{bmatrix}$$







The OpenGL Projection Matrix

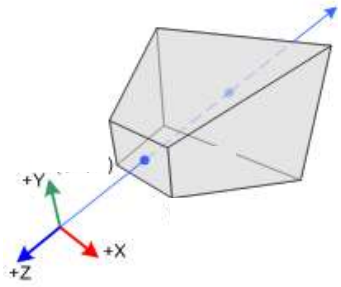
- Projection matrix:

screen mapping

screen mapping

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2\,near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Z mapping

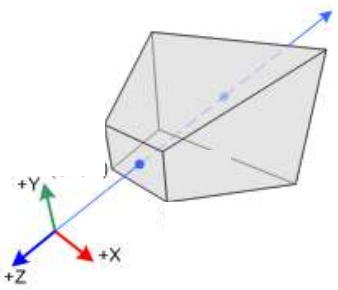


, where near and far are the distances of the planes to the origin.

The OpenGL Projection Matrix

- Projection matrix:

$$\begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



, where near and far are the distances of the planes to the origin.

- Verify:
  Point (0,0,-near,1) has a z-value of -1 after projection
  Point (0,0,-far,1) has a z-value of 1 after projection
  For near=2 and far =4 calculate the projection of (0,0,-2,1)

Traditional OpenGL Camera

- WE ARE DONE!

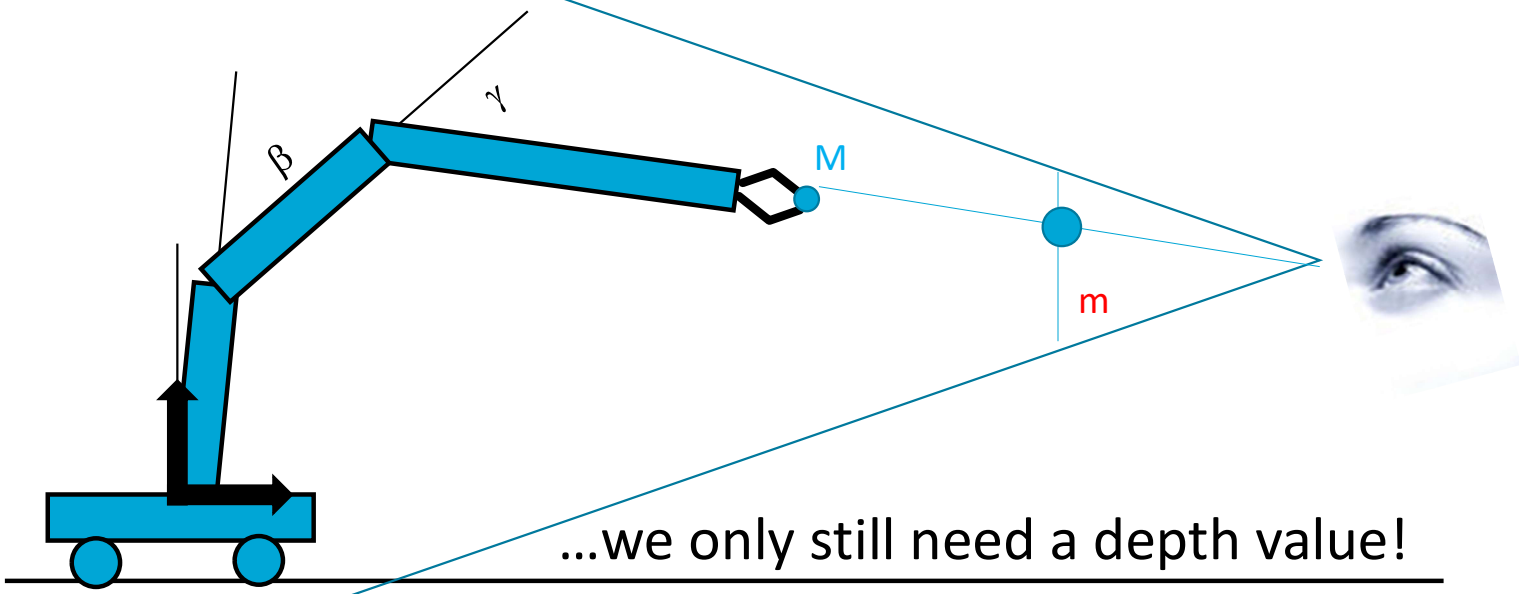
$$\begin{bmatrix} k_x & 0 & 0 & x_0 \\ 0 & k_y & 0 & y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{f}{aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{near+far}{near-far} & \frac{2near\,far}{near-far} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

4x4
ModelView Matrix

Viewport | Projection | ModelView

Today

Find matrix P such that the projected pixel position m of point M is PM.



…we only still need a depth value!

Conclusion

- In this lecture, you have seen how to:
  - Transformations with homogeneous coordinates
  - Creation of complex objects via local frames
  - Working towards a virtual camera expressed as a matrix

...(almost) all ingredients for the geometric graphics pipeline

Reading

- Chapter 7 in the book
  - Derivation is in a slightly different order:
    - First scene deformation with respect to a camera frustum
    - Optional: As preparation for next time
      - viewport matrix
      - projection matrix

Exercises

- Define a 45 degree rotation around a point and verify the matrix is correct with examples
- Check the given rotation matrices on the slides

Thank you very much!