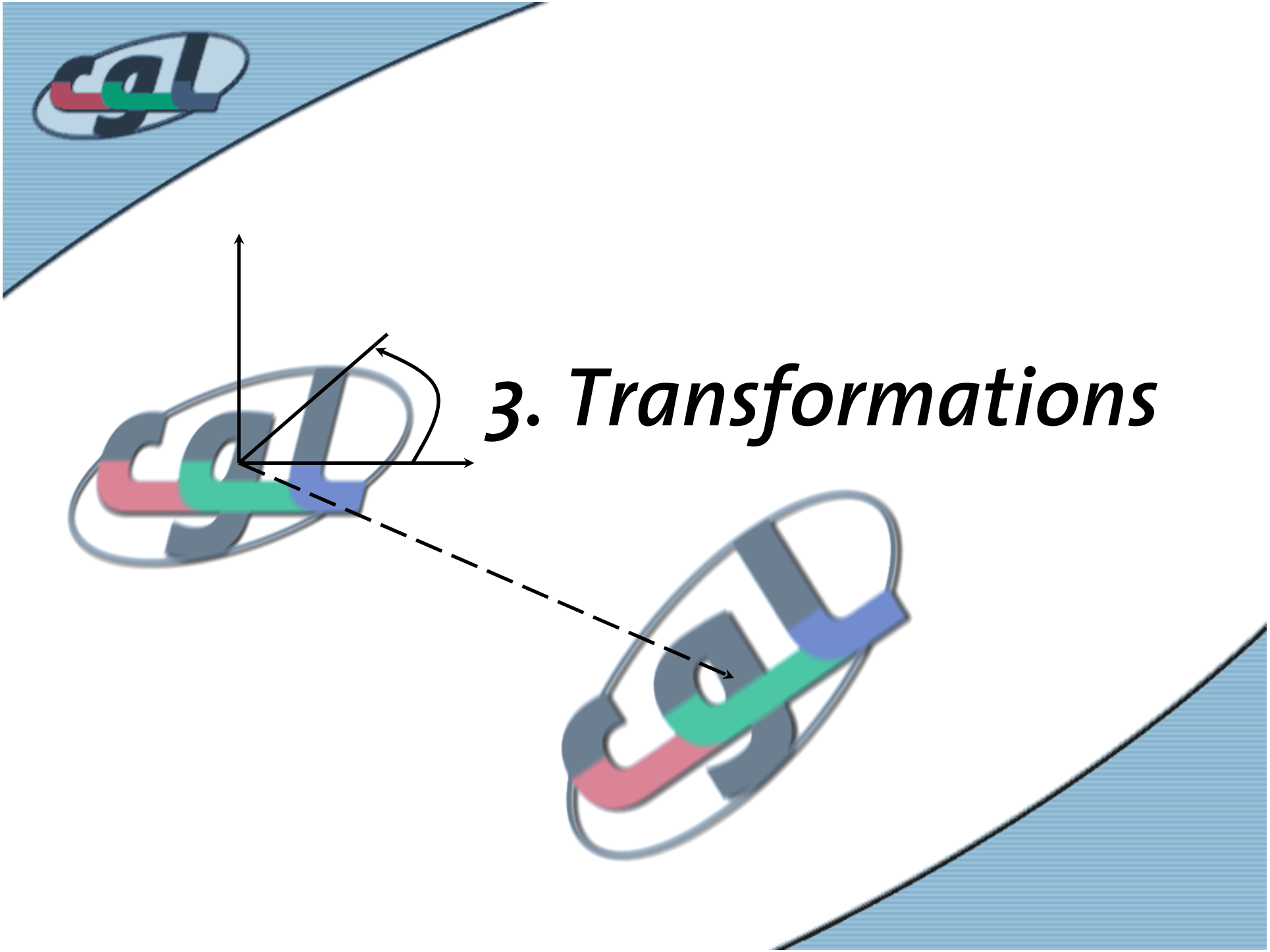

3. Transformations

Linear and Affine Mappings

- a map A: $\boldsymbol{x} \Rightarrow \boldsymbol{x}'$ is called ***linear***, if

$$A(\alpha \boldsymbol{x} + \beta \boldsymbol{y}) = \alpha A(\boldsymbol{x}) + \beta A(\boldsymbol{y})$$

- a map A: $\boldsymbol{x} \Rightarrow \boldsymbol{x}'$ is called ***affine***, if

$$\boldsymbol{x}' = A(\boldsymbol{x}) + \boldsymbol{t} = B(\boldsymbol{x})$$

- Linear transforms are represented by matrices, i.e. $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}'$

2D Transforms

- Translation

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{P'} = \boldsymbol{P} + \boldsymbol{T}$$

- Scaling

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{P'} = \boldsymbol{S} \times \boldsymbol{P}$$

2D Transforms

- Rotation of a point ***P*** along some angle θ

$$x' = x \cdot cos\theta - y \cdot sin\theta$$

$$y' = x \cdot sin\theta + y \cdot cos\theta$$

- Matrix form

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} \quad \Rightarrow \quad \boldsymbol{P'} = \boldsymbol{R} \times \boldsymbol{P}$$

Homogenous Coordinates

- Matrix form of 3 fundamental transforms

$$P' = R \times P \qquad P' = S \times P \qquad P' = T + P$$

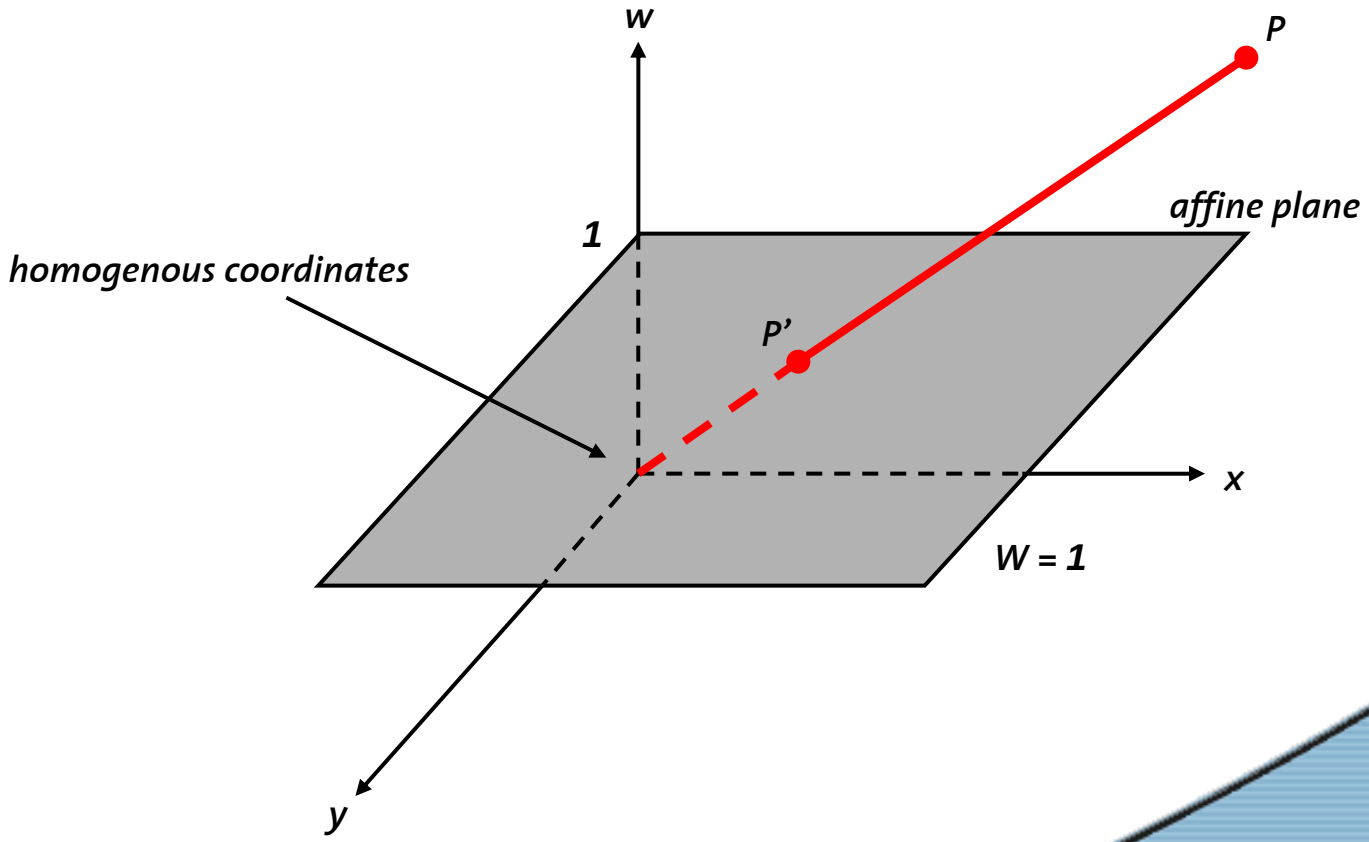
- Translation as additive component
- Take point ***P*** in homogenous coordinates by adding additional weight:

$$P = (x, y, W)$$

A point P in 2D has infinitely many homogenous coordinates: (2,1) can be written as (2,1,1), (4,2,2) or (-4,-2,-2)... Division by w -> projection

Homogenous Coordinates

- Point ***P'*** = (*x*, *y*, ***1***) as line (*wx*, *wy*, *w*) in 3D



Translation and Scaling

- Representation by 3x3 matrix

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \times \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \Rightarrow \quad \boldsymbol{P'} = \boldsymbol{T} \times \boldsymbol{P}$$

- Scaling

$$\boldsymbol{S}(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation and Shearing

- Rotation Matrix

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

- Shears along x- and y-axis

$$\boldsymbol{SH}_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \boldsymbol{SH}_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combinations

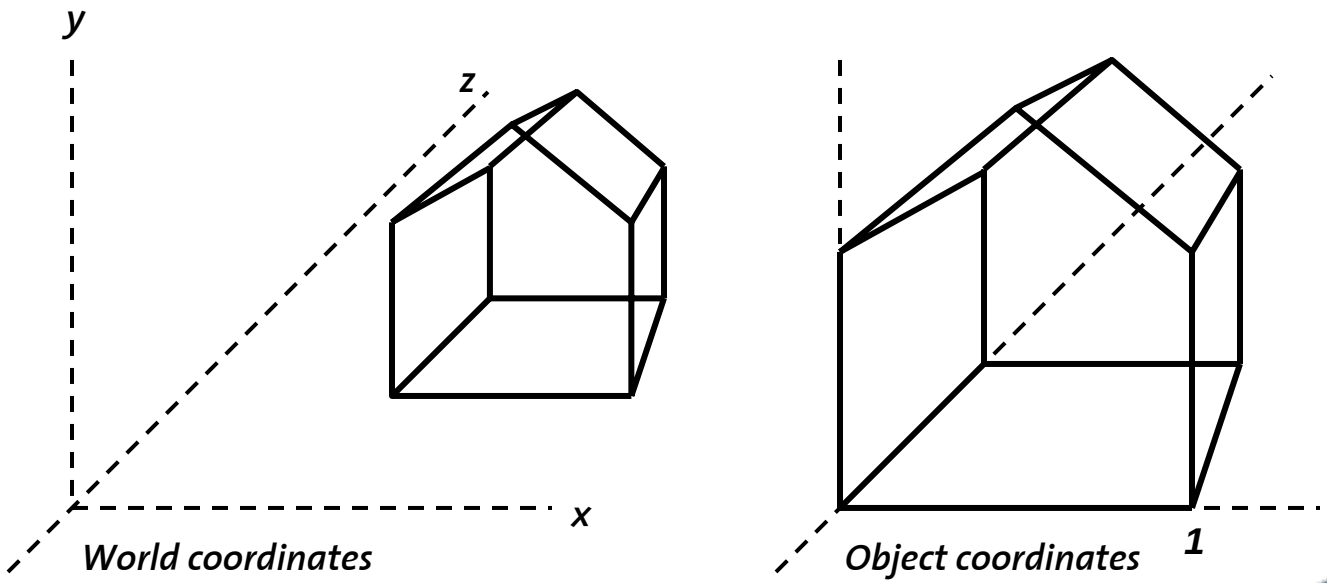
- Stack transforms using matrix multiplication
- Example: Translation and Rotation $\boldsymbol{T} \times \boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$
- Commutativity of Matrices ***M1***, ***M2*** given if

Matrix M1	*Matrix M2*
Translation	*Translation*
Rotation	*Rotation*
Scaling	*Scaling*
Scaling	*Rotation*

2D only!

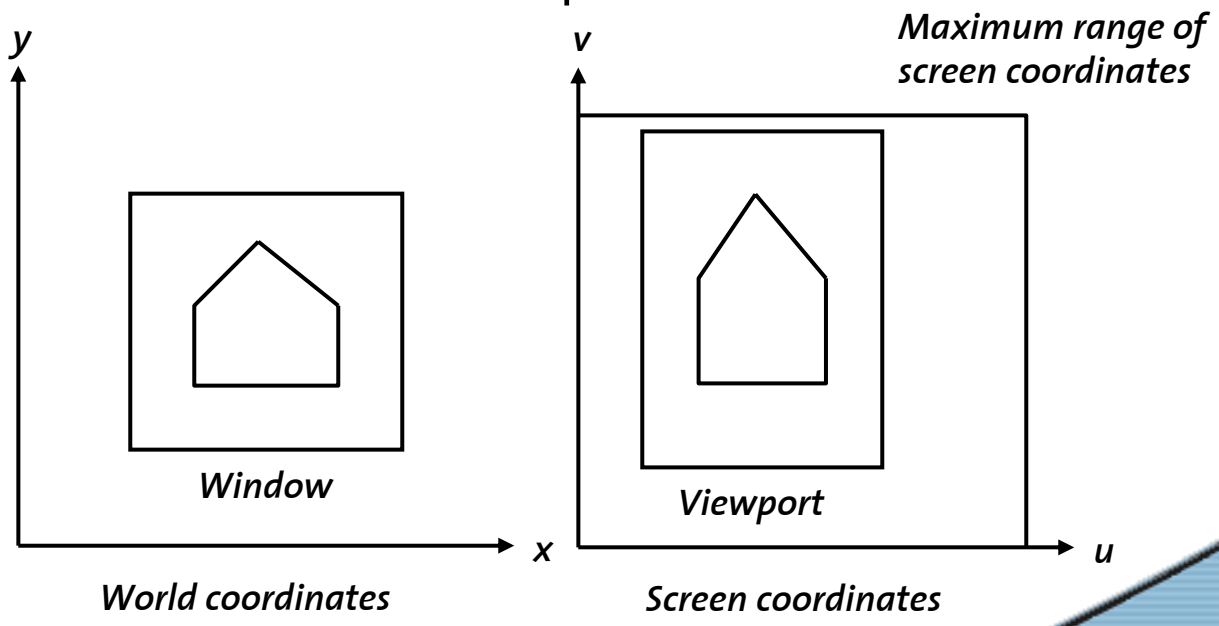
Coordinate Systems

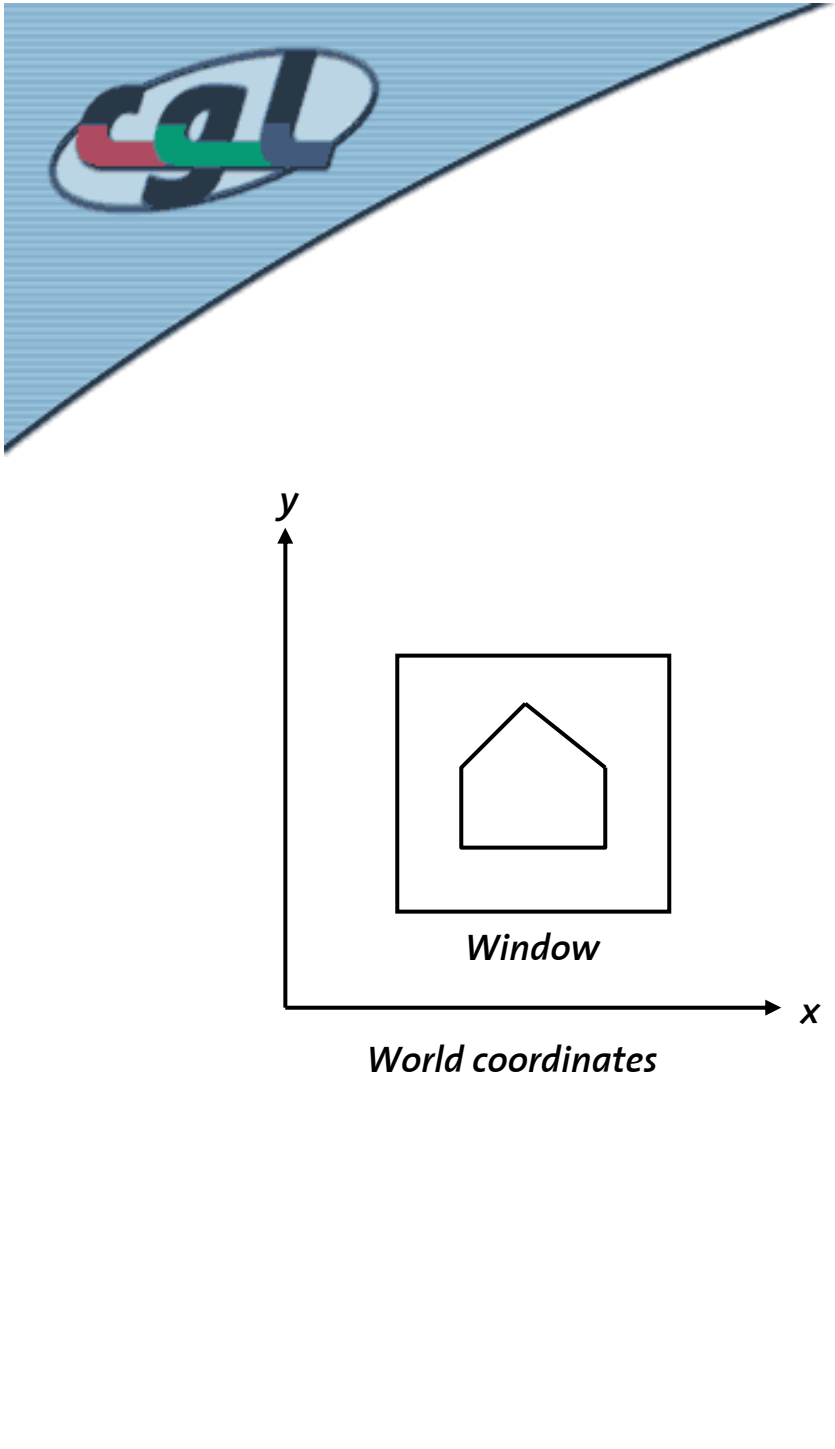
- ***World coordinates***: reference system to describe scenery
- ***Object coordinates***: internal representation



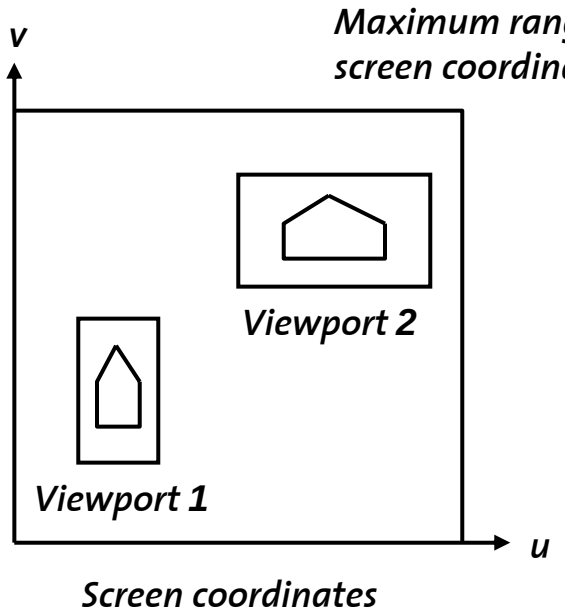
Windows and Viewports

- ***Window*** in world coordinates to represent camera
- ***Viewport*** in device coordinates to define sub-window of output device



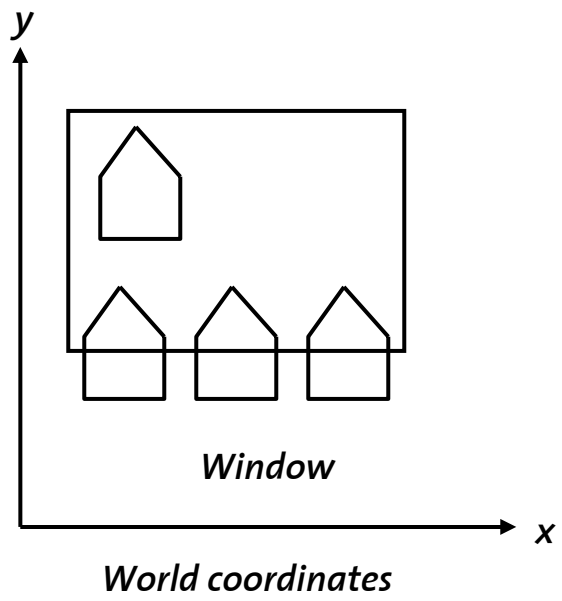


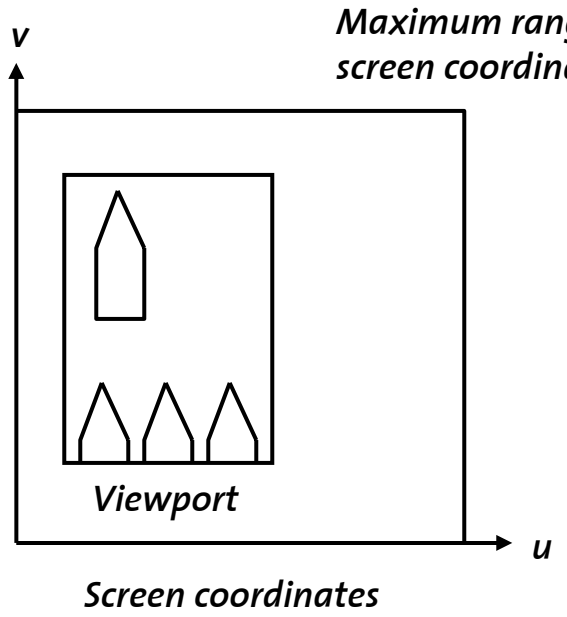
Multiple Viewports



Clipping Windows

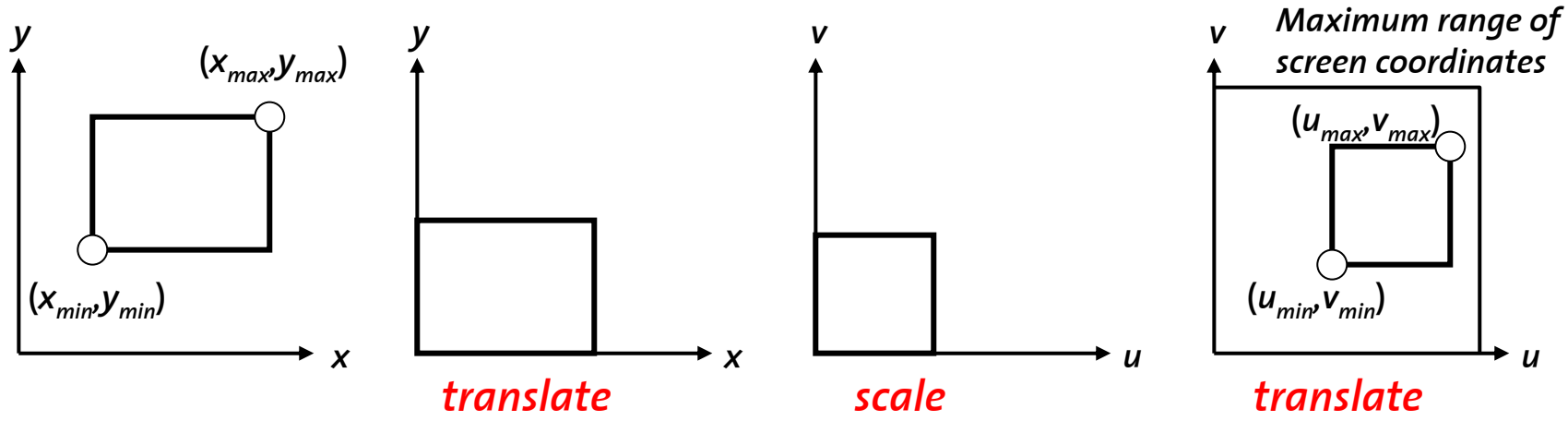
- Transform affects clipping and ***aspect ratio***





Window-Viewport Transform

- 3 step sequence of translate-scale-translate



$$M_{WV} = T(u_{min}, v_{min}) \times S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \times T(-x_{min}, -y_{min})$$

Window-Viewport Transform

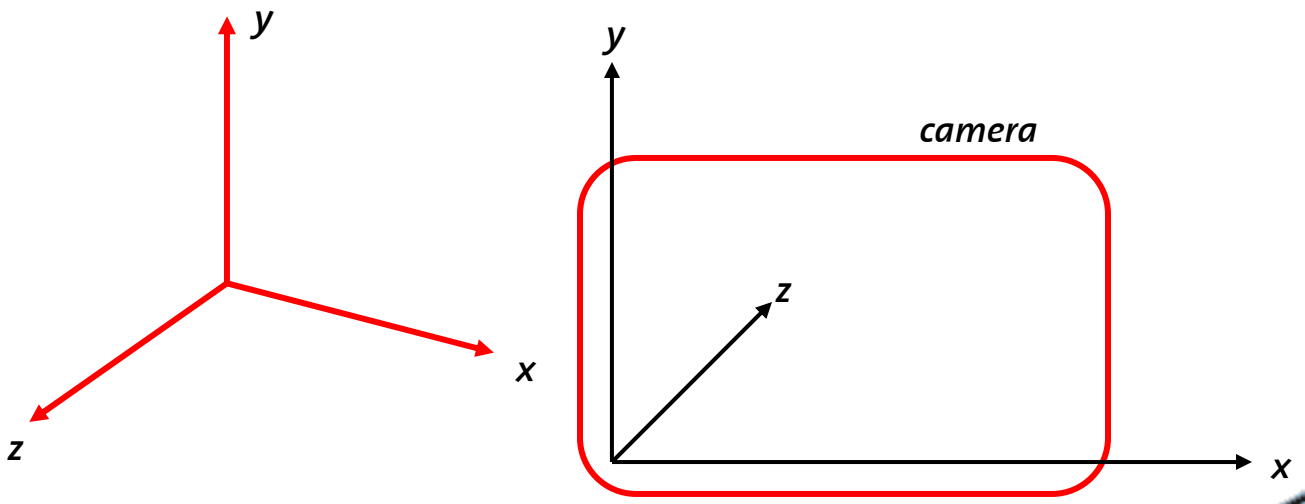
- Homogenous coordinates

$$= \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & -x_{min} \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & -y_{min} \cdot \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

3D Transforms

- Homogenous coordinates -> 4x4 matrices
- Project point $\boldsymbol{p} = (x, y, z, w)$ onto hyperplane $(x/w, y/w, z/w, \boldsymbol{1})$
- Left-handed versus right-handed systems



Translation and Scaling

- Representation by 4x4 matrix

$$\boldsymbol{T}(t_x, t_y, t_z) = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scaling

$$\boldsymbol{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Rotation

- Rotation matrices for x-,y-, and z-axis

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

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

Rotation about Arbitrary Axis

- Normalized axis $\mathbf{u}$, angle θ

$$\mathbf{R}(\mathbf{u},\theta) = \begin{bmatrix} u_x^2 + \cos\theta(1-u_x^2) & u_x u_y(1-\cos\theta) - u_z \sin\theta & u_x u_z(1-\cos\theta) + u_y \sin\theta & 0 \\ u_x u_y(1-\cos\theta) + u_z \sin\theta & u_y^2 + \cos\theta(1-u_y^2) & u_y u_z(1-\cos\theta) + u_x \sin\theta & 0 \\ u_x u_z(1-\cos\theta) - u_y \sin\theta & u_y u_z(1-\cos\theta) + u_x \sin\theta & u_z^2 + \cos\theta(1-u_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear

- Shearing parallel to principal planes

$$\mathbf{SH_{xy}}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{SH_{xz}}(sh_x, sh_z) = \begin{bmatrix} 1 & sh_x & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & sh_z & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{SH_{yz}}(sh_y, sh_z) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ sh_y & 1 & 0 & 0 \\ sh_z & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transform of Normal Vector

- Given a plane in implicit form

$$Ax + By + Cz + D = 0$$

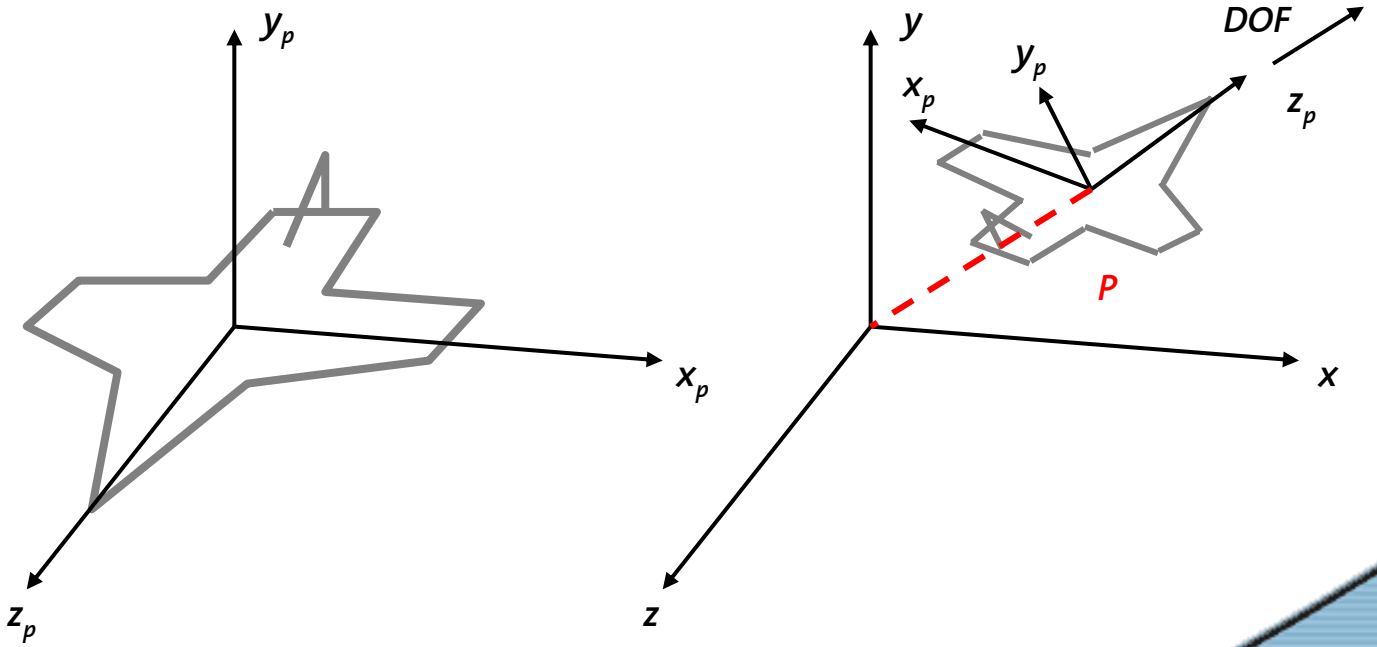
- Normal $\boldsymbol{n} = (A, B, C, D)$ and $\boldsymbol{P} = (x, y, z, 1)$
- Transformed normal $\boldsymbol{n'}$ computed by

$$\boldsymbol{n}' = (\boldsymbol{M}^{-1})^T \boldsymbol{n}$$

Verification by insertion into plane equation and some algebra !

Compound 3D Transforms

- Object-to-world coordinate transform
- Example: Visual simulation



Compound 3D Transforms

1. Compute a new orthogonal basis by
   - $\mathbf{r}_1 = \boldsymbol{y} \times \boldsymbol{DOF}$
   - $\mathbf{r}_2 = \boldsymbol{DOF} \times (\boldsymbol{y} \times \boldsymbol{DOF})$ (orthogonal)
   - $\mathbf{r}_3 = \boldsymbol{DOF}$
2. Corresponding rotation matrix is

$$\mathbf{R} = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & 0 \\ r_{1y} & r_{2y} & r_{3y} & 0 \\ r_{1z} & r_{2z} & r_{3z} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Compound 3D Transforms

2. Translate into $\boldsymbol{P} = (p_x, p_y, p_z, 1)$

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

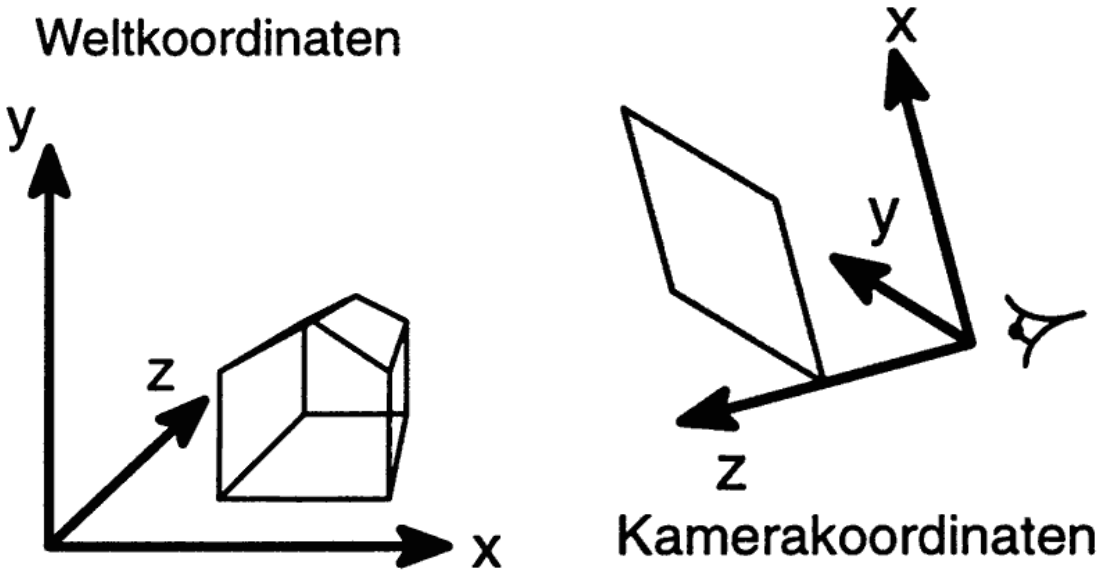
3. Compound model matrix

$$\mathbf{M} = \mathbf{TR} = \begin{bmatrix} r_{1x} & r_{2x} & r_{3x} & p_x \\ r_{1y} & r_{2y} & r_{3y} & p_y \\ r_{1z} & r_{2z} & r_{3z} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Model-View Transform

Model and View transforms are inverse to each other !

1. Left-Right transform
2. Rotate into new basis (camera)
3. Translate
4. Invert compound matrix



Modeling Transform – Forward

$$M_T \times M_R \times M_{LR} = \begin{bmatrix} 1 & 0 & 0 & P_x \\ 0 & 1 & 0 & P_y \\ 0 & 0 & 1 & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_{x_1} & h_{y_1} & h_{z_1} & 0 \\ h_{x_2} & h_{y_2} & h_{z_2} & 0 \\ h_{x_3} & h_{y_3} & h_{z_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} h_{x_1} & h_{y_1} & -h_{z_1} & P_x \\ h_{x_2} & h_{y_2} & -h_{z_2} & P_y \\ h_{x_3} & h_{y_3} & -h_{z_3} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Viewing Transform – Inverse

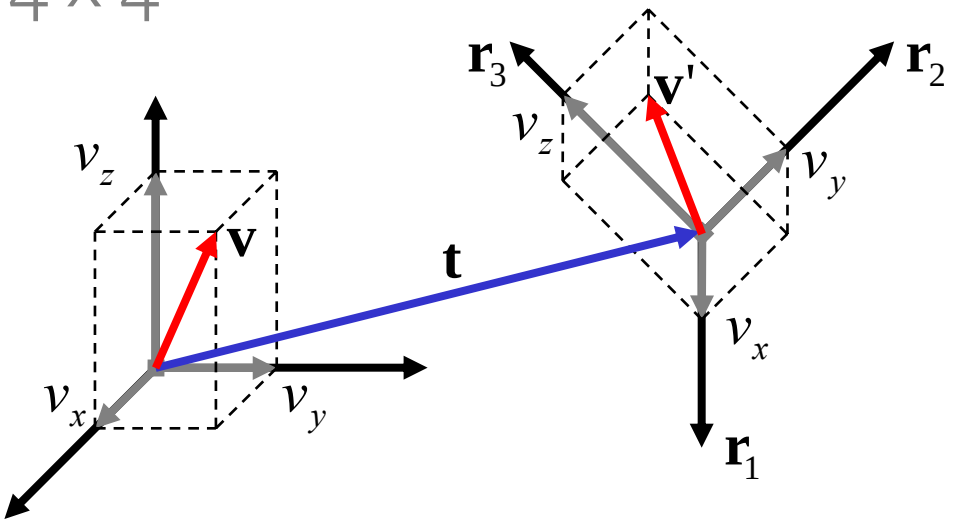
$$M_{LR}^{-1} \times M_R^{-1} \times M_T^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_{x_1} & h_{x_2} & h_{x_3} & 0 \\ h_{y_1} & h_{y_2} & h_{y_3} & 0 \\ h_{z_1} & h_{z_2} & h_{z_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note the orthogonality of M_R-> transpose is inverse !

Duality of Modeling Transform and Viewing Transform (OpenGL)

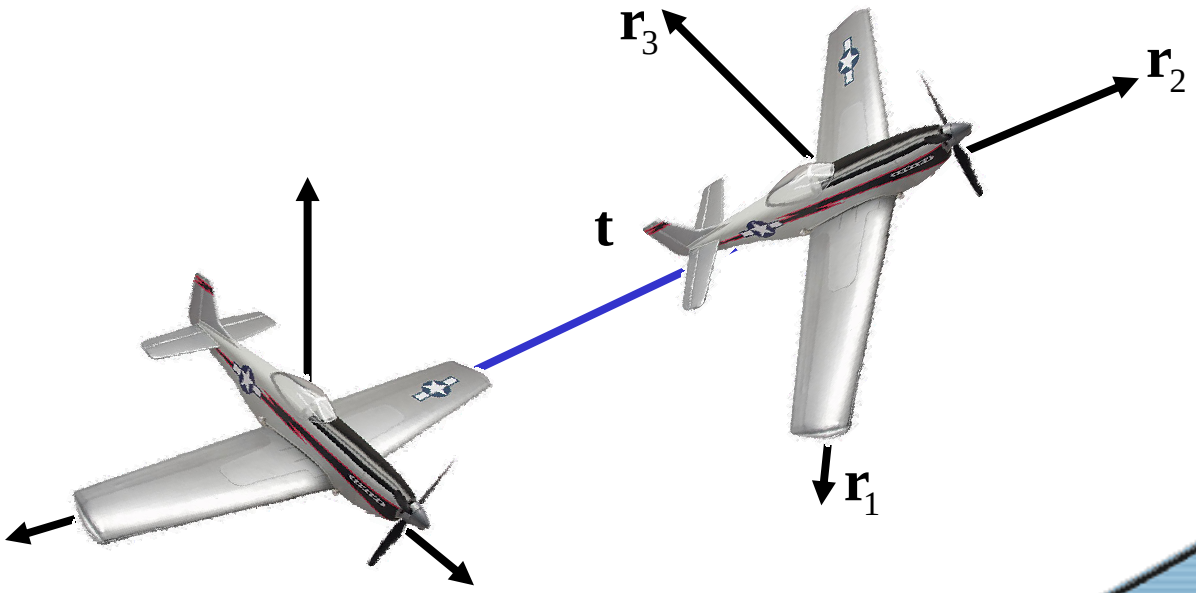
Transformation between Coordinate Systems

$$\underbrace{\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{4 \times 4} \cdot \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{t} + v_x\mathbf{r}_1 + v_y\mathbf{r}_2 + v_z\mathbf{r}_3 \\ 1 \end{bmatrix}$$



Model to World

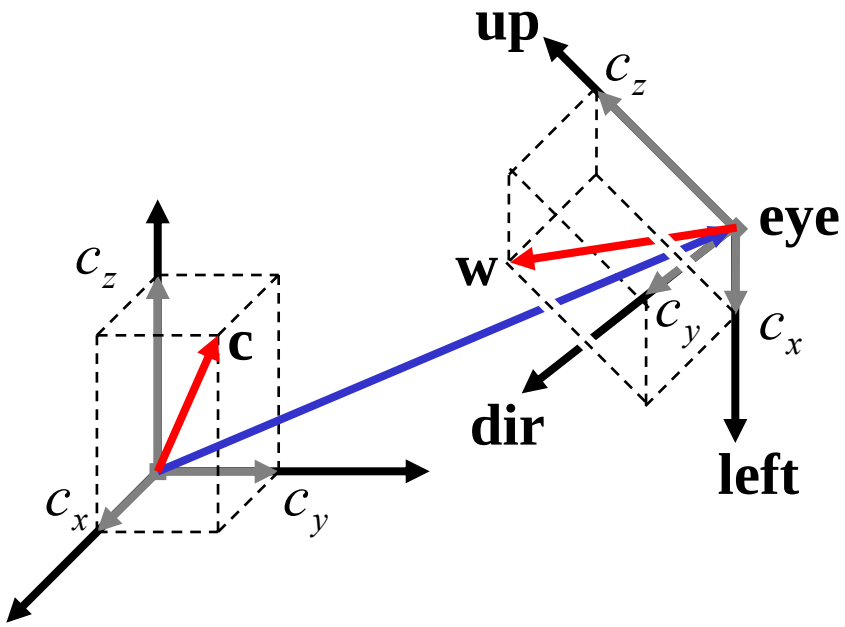
$$\begin{bmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} m_x \\ m_y \\ m_z \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{t} + v_x\mathbf{r}_1 + v_y\mathbf{r}_2 + v_z\mathbf{r}_3 \\ 1 \end{bmatrix} = \begin{bmatrix} w_x \\ w_y \\ w_z \\ 1 \end{bmatrix}$$



World to Camera

$$\begin{bmatrix} \mathbf{dir} & \mathbf{up} & -\mathbf{left} & \mathbf{eye} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} c_x \\ c_y \\ c_z \\ 1 \end{bmatrix} = \begin{bmatrix} w_x \\ w_y \\ w_z \\ 1 \end{bmatrix}$$

- Solve for **c**
- Invert Transformation



An OpenGL Example

- View Transform for entire scene

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0,0,2, 0,0,0, 0,1,0);  // eye, center, up
```

- Model Transform only for teapot

```
glMatrixMode(GL_MODELVIEW);
glPushMatrix();
glRotatef(30, 1.0, 1.0, 1.0);
glutSolidTeapot(0.5);
glPopMatrix();
```

Quaternions

- Elegant notion for the modeling of translation and rotation
- Compact representation
- Mathematical element with a set of operators (e.g. multiplication)
- Efficient implementation
- Widely used for animation

Definition

- A ***quaternion*** $\boldsymbol{q}$ is a "hypercomplex" number defined as: $\boldsymbol{q} = c + x\,i + y\,j + z\,k$
- c, y, x, z are real numbers
- i, j, k are imaginary
- Notation: $\boldsymbol{q} = c + \boldsymbol{u}$
- c: real part, $\boldsymbol{u}$: ***pure quaternion***

$$\boldsymbol{u} = x\,i + y\,j + z\,k$$

Basic Properties and Operators

- Addition

$$\boldsymbol{q}+\boldsymbol{q}'=(c+c')+(x+x')i+(y+y')j+(z+z')k$$

- Multiplication of complex operators

$$i^2=j^2=k^2=-1$$

$$ij=k,\quad ji=-k;\quad jk=i,\quad kj=-i;\quad ki=j,\quad ik=-j$$

- Quaternion multiplication of $\boldsymbol{q}$ and $\boldsymbol{q}'$

$$\boldsymbol{q}\boldsymbol{q}'=(c+u)(c'+u')$$
$$=(cc'-\boldsymbol{u}\cdot\boldsymbol{u}')+(\boldsymbol{u}\times\boldsymbol{u}'+<c\boldsymbol{u}'>+<c'\boldsymbol{u}>)$$

Quaternion multiplication is NOT commutative!

Basic Properties and Operators

- Inner product, scalar operator **<...>** and cross product **×** defined as:

$$\boldsymbol{u}\cdot\boldsymbol{u}' = x\,x' + y\,y' + z\,z'$$

$$<c\,\boldsymbol{u}> = c\,x\,i + c\,y\,j + c\,z\,k$$

$$\boldsymbol{u}\times\boldsymbol{u}' = (y\,z' - z\,y')i + (z\,x' - x\,z')j + (x\,y' - y\,x')k$$

- One-elements of addition and multiplication

$$0 = 0 + 0\,i + 0\,j + 0\,k$$

$$1 = 1 + 0\,i + 0\,j + 0\,k$$

Basic Properties and Operators

- Conjugate elements

$$q = c + u \quad ; \quad \bar{q} = c - u$$

- Absolute

$$q\bar{q} = c^2 + x^2 + y^2 + z^2$$

$$q\bar{q} = |q|^2$$

- Inverse elements of
  - addition $-q = -c - x\,i - y\,j - z\,k$
  - multiplication $q^{-1} = \frac{1}{|q|^2}\bar{q}$

Unit Quaternions

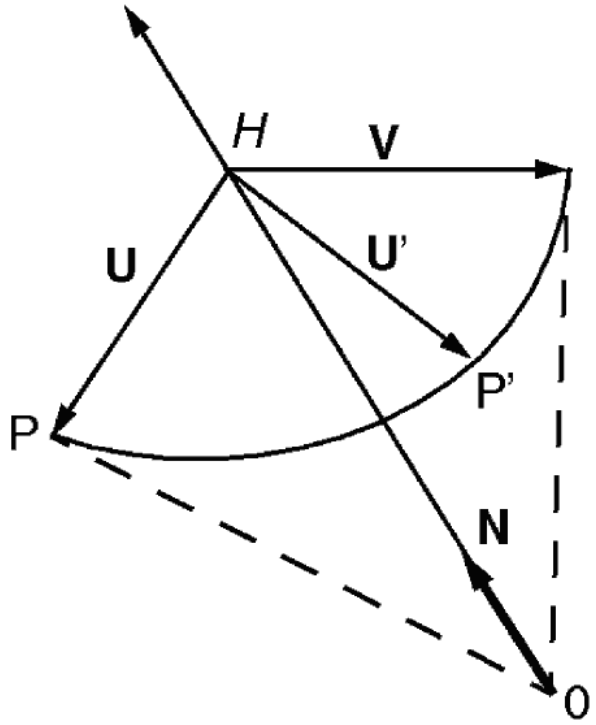
- Quaternions of length 1 are fundamental to encode transformations
- Using a unit quaternion.. $|\boldsymbol{q}|^2 = c^2 + \boldsymbol{u} \cdot \boldsymbol{u} = 1$
- .. and introducing .. $\boldsymbol{N} = [N_x, N_y, N_z]^T \quad \boldsymbol{I} = [i, j, k]^T$
- ..with.. $\boldsymbol{q} = c + \boldsymbol{u} \quad \boldsymbol{u} = s\boldsymbol{n} \quad \boldsymbol{n} = \boldsymbol{N}\boldsymbol{I}$
- ..it follows that $c^2 + s^2 = 1$
- Each unit quaternion can be rewritten as

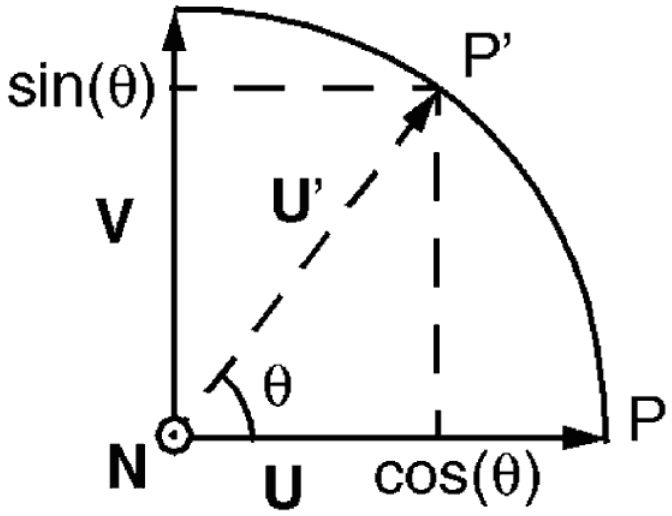
$$\boldsymbol{q} = \cos\theta + \sin\theta\, \boldsymbol{n}$$

$$\Rightarrow \boldsymbol{q}\boldsymbol{q}' = \cos(\theta + \theta') + \sin(\theta + \theta')\boldsymbol{n}$$

3D Rotation using Quaternions

- Rotation of a point ***P*** along an arbitrary axis ***N*** and angle θ





3D Rotation using Quaternions

- Find ***P'*** as a function of ***P*** and ***N***

$$\boldsymbol{P'} = \cos\theta\,\boldsymbol{P} + (1-\cos\theta)\,\boldsymbol{N}(\boldsymbol{N}\cdot\boldsymbol{P}) + \sin\theta\,(\boldsymbol{N}\times\boldsymbol{P})$$

- Corresponding rotation operator ***R*(θ, *N*)**

$$\boldsymbol{R}(\theta,\boldsymbol{N}) = \cos\theta\,\boldsymbol{I}_3 + (1-\cos\theta)\,\boldsymbol{N}^T\boldsymbol{N} + \sin\theta\,\boldsymbol{A}_{\boldsymbol{N}}$$

$$\boldsymbol{N} = \begin{pmatrix} N_1 \\ N_2 \\ N_3 \end{pmatrix}, \quad \boldsymbol{I}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \boldsymbol{A}_{\boldsymbol{N}} = \begin{bmatrix} 0 & N_3 & -N_2 \\ -N_3 & 0 & N_1 \\ N_2 & -N_1 & 0 \end{bmatrix}$$

3D Rotation using Quaternions

- Represent point **P** as a pure quaternion

$$\boldsymbol{P} = (x, y, z) \qquad \boldsymbol{p} = 0 + \boldsymbol{v} = xi + yj + zk$$

- Rotation using quaternion operators

$$\boldsymbol{R}_q(\boldsymbol{p}) = \boldsymbol{q}\,\boldsymbol{p}\,\bar{\boldsymbol{q}} \quad \boldsymbol{q} = c + \boldsymbol{u} = \cos\theta + \sin\theta\,\boldsymbol{n}$$

- Insertion and a little algebra reveals ..

$$\boldsymbol{R}_q(\boldsymbol{p}) = \begin{bmatrix} < \cos(2\theta)\boldsymbol{v} > + \\ < (1 - \cos(2\theta))(\boldsymbol{n}\cdot\boldsymbol{v})\boldsymbol{n} + < \sin(2\theta)(\boldsymbol{n}\times\boldsymbol{v}) > \end{bmatrix}$$

- Rotation of **P** along axis **N** by angle 2θ

Points and Rotation

- Recipe: Take $\boldsymbol{p}$ and $\boldsymbol{q}$ and compute

$$\boldsymbol{p}' = \boldsymbol{q}\,\boldsymbol{p}\,\bar{\boldsymbol{q}}$$

$$\boldsymbol{q} = \cos(\theta/2) + \sin(\theta/2)\,\boldsymbol{n}$$

$$\boldsymbol{n} = N_1\,i + N_2\,j + N_3\,k$$

- Elegant implementation using operator overloading

Points and Translation

- Translation vector as a pure quaternion

$$p' = p + t$$

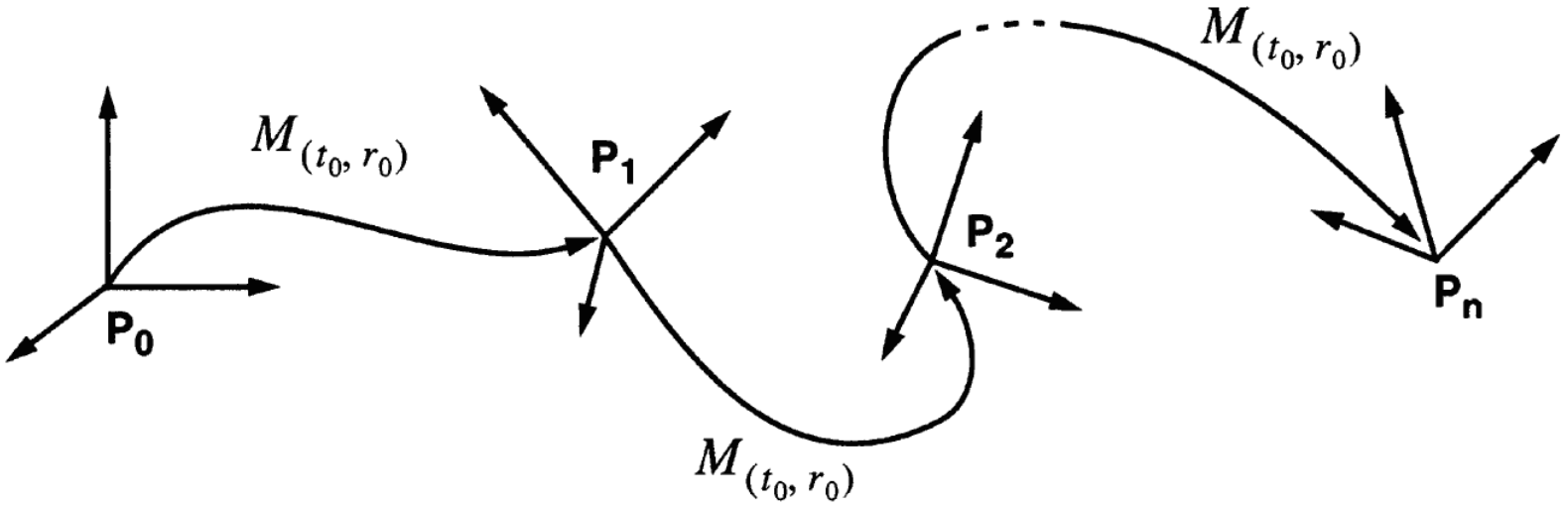
- Sequences of rotations r and translations t using a transformation operator M

$$p \rightarrow p' = M_{(t,r)}(p) = r\,p\,\bar{r} + t$$

- $M_{(t,r)}$ denotes rotation-translation-operator
- $M_{(0,r)}$ describes rotation and $M_{(t,1)}$ translation

$$M_{(t,r)} = M_{(0,r)} \circ M_{(t,1)}$$

Sequencing for Animation



$$(t, r) = (t_0, r_0) \circ (t_i, r_i)$$

$$(t, r) \circ (t', r') = (t + r\, t' \bar{r}, r\, r')$$

$$(t, r) = (t_0, r_0) \circ \ldots \circ (t_i, r_i) \circ \ldots \circ (t_n, r_n)$$