CS 428: Fall 2010

Introduction to Computer Graphics

Geometric Transformations (continued)

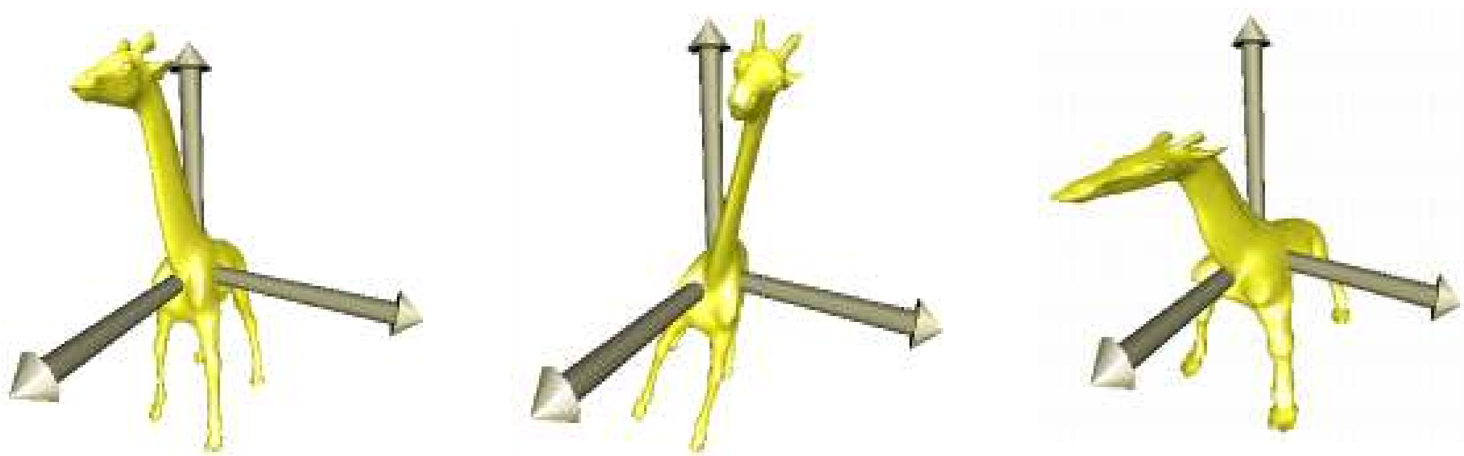
Translation

- Translations are affine transformations
  - The linear part is the identity matrix
  - The 4x4 matrix for the translation by vector $(x_0, y_0, z_0)^t$ is given as

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_0 \\ y + y_0 \\ z + z_0 \\ 1 \end{bmatrix}$$

Scaling, shearing and rotation

- Affine transformations scaling, shearing and rotation leave the origin invariant
  - Their translation component is zero
  - These are purely linear transformations
  - 3x3 matrices would suffice, if we were only interested in these



Scaling, shearing and rotation

- Homogeneous form

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- The images of the basis vectors $(1,0,0)^t$, $(0, 1, 0)^t$, $(0, 0, 1)^t$ define the linear transformation $A : \Re^3 \mapsto \Re^3$

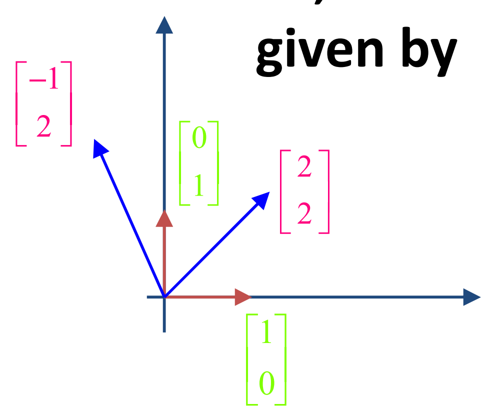
- As a simplification, vectors are written transposed in the text $(x, y, z)^t = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

Scaling, shearing and rotation

- Multiplying the canonical coordinate axes from the right shows the images of the basis vectors in the columns of the matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{31} \end{bmatrix} \qquad \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_{13} \\ a_{23} \\ a_{33} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{12} \\ a_{22} \\ a_{32} \end{bmatrix}$$



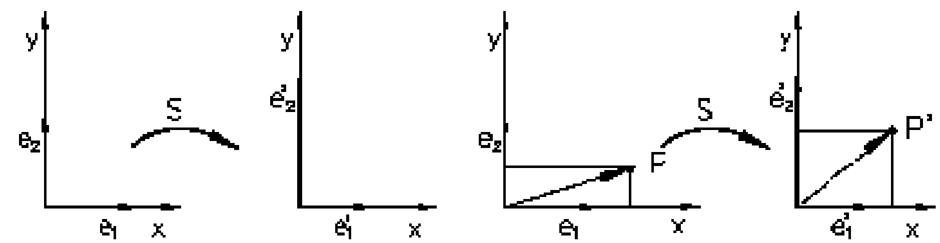
So, this linear transformation is given by

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

Scaling

anisotropic

- Scaling **S** modifies the basis vectors as
  - **S**$((1,0,0,)^t) = (s_1, 0, 0)^t$
  - **S**$((0, 1, 0)^t) = (0, s_2, 0)^t$
  - **S**$((0, 0, 1)^t) = (0, 0, s_3)^t$



- Resulting in the following 3x3 linear and 4x4 homogeneous transformation

$$\begin{pmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & s_3 \end{pmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Scaling

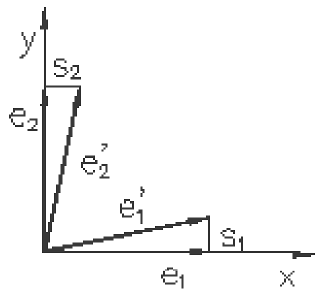
isotropic

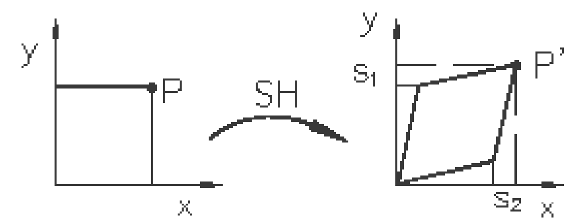
- The special case $s_1 = s_2 = s_3 = s$ means equal (isotropic) scaling for all coordinate axes
- The homogeneous matrix has the form

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{bmatrix}$$

Shearing

- Shearing **SH** modifies the basis vectors as
  - **SH**($(1, 0 , 0)^t$) = $(1, s_1, s_3)^t$
  - **SH**($(0, 1, 0)^t$) = $(s_2, 1, s_4)^t$
  - **SH**($(0, 0, 1)^t$) = $(s_5, s_6, 1)^t$





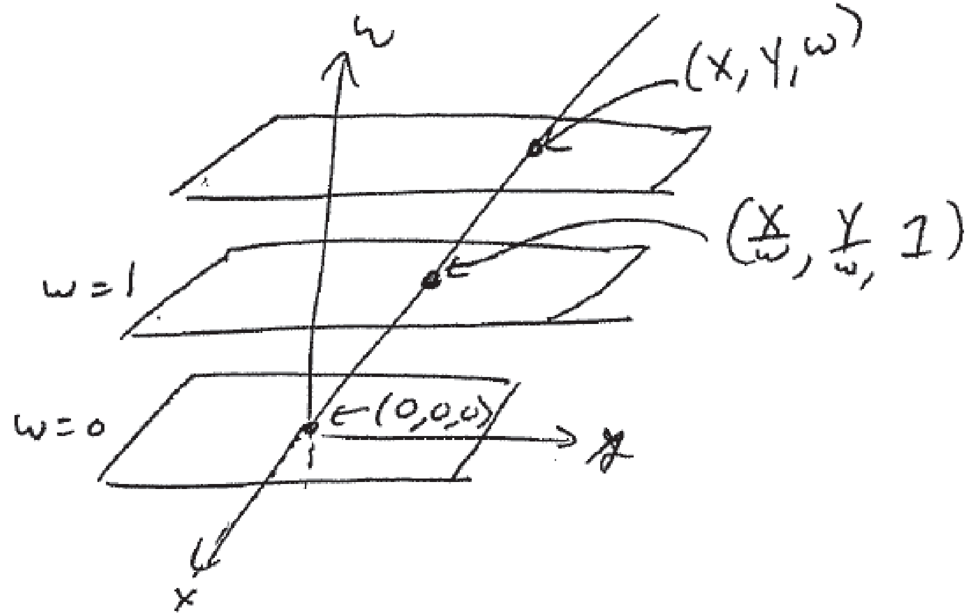
- Resulting in the following 3x3 linear and 4x4 homogeneous transformation

$$\begin{pmatrix} 1 & s_2 & s_5 \\ s_1 & 1 & s_6 \\ s_3 & s_4 & 1 \end{pmatrix} \qquad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & s_2 & s_5 & 0 \\ s_1 & 1 & s_6 & 0 \\ s_3 & s_4 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Homogeneous coordinates

Geometric interpretation

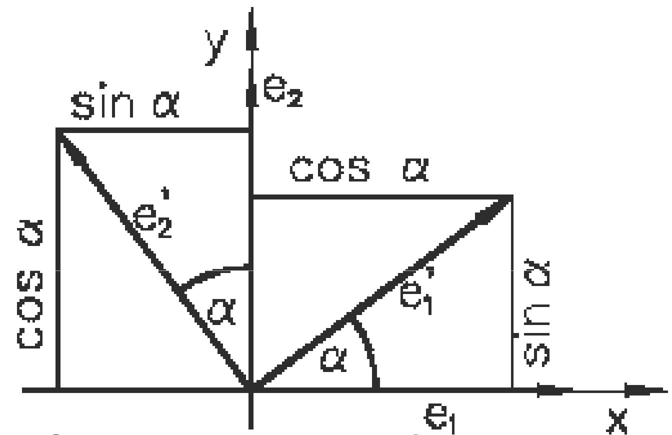
- Linear transformation in 3D can be used to compute affine transformation in 2D
- **Affine** translation in 2D becomes **linear** shear in 3D within the w = 1 plane (!)



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + x_0 \\ y + y_0 \\ 1 \end{bmatrix}$$

Rotation

- Rotation $\mathbf{R}_\alpha$ with angle α about the z-axis modifies the basis vectors as
  - $\mathbf{R}_\alpha((1, 0, 0)^t) = (\cos\alpha, \sin\alpha, 0)$
  - $\mathbf{R}_\alpha((0, 1, 0)^t) = (-\sin\alpha, \cos\alpha, 0)$
  - $\mathbf{R}_\alpha((0, 0, 1)^t) = (0, 0, 1)$

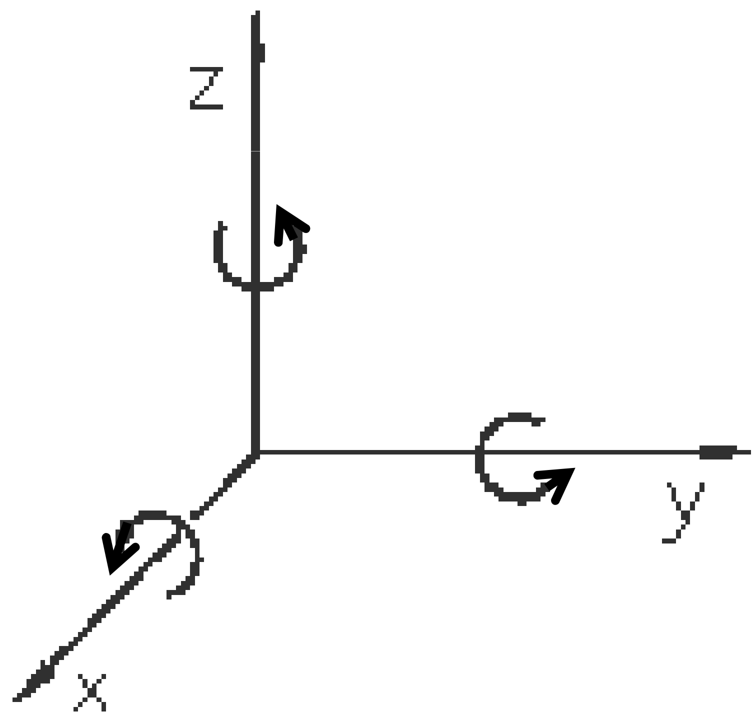


- Resulting in the following 3x3 linear and 4x4 homogeneous transformation

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation

- The following turning angles are positive in a right handed coordinate system



Rotation

- For rotations R_α about the x- and y-axis
  - Angle α about the x-axis

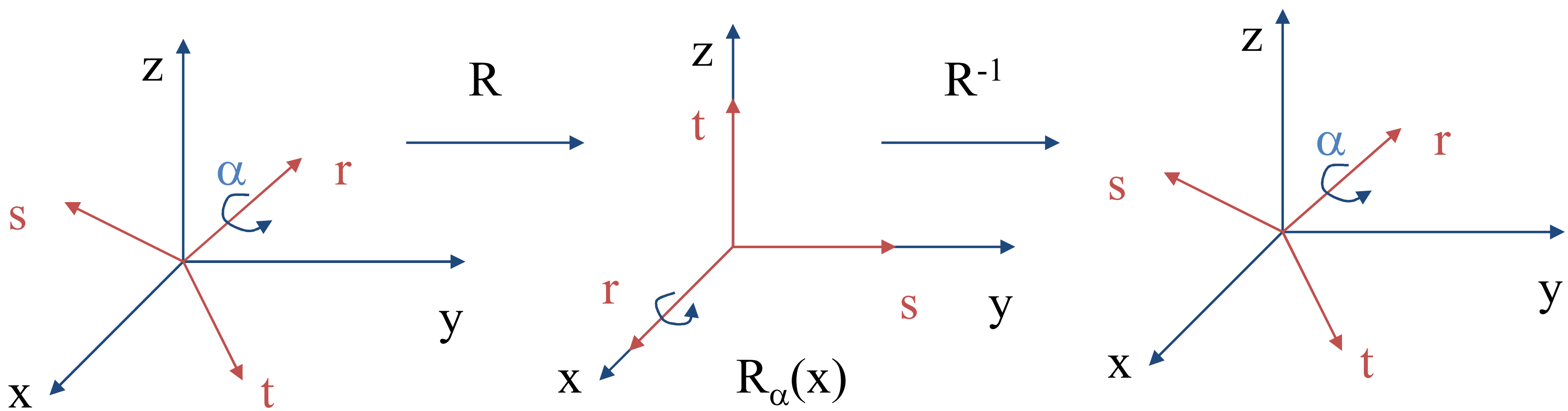
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

  - Angle α about the y-axis

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

Rotation about an arbitrary axis

- Rotation $R(x,y,z)$ about the normalized vector $\mathbf{r} = (x,y,z)^t$ with angle α



$$R_{(x,y,z)} = R^{-1} R_{\alpha}(x) R$$

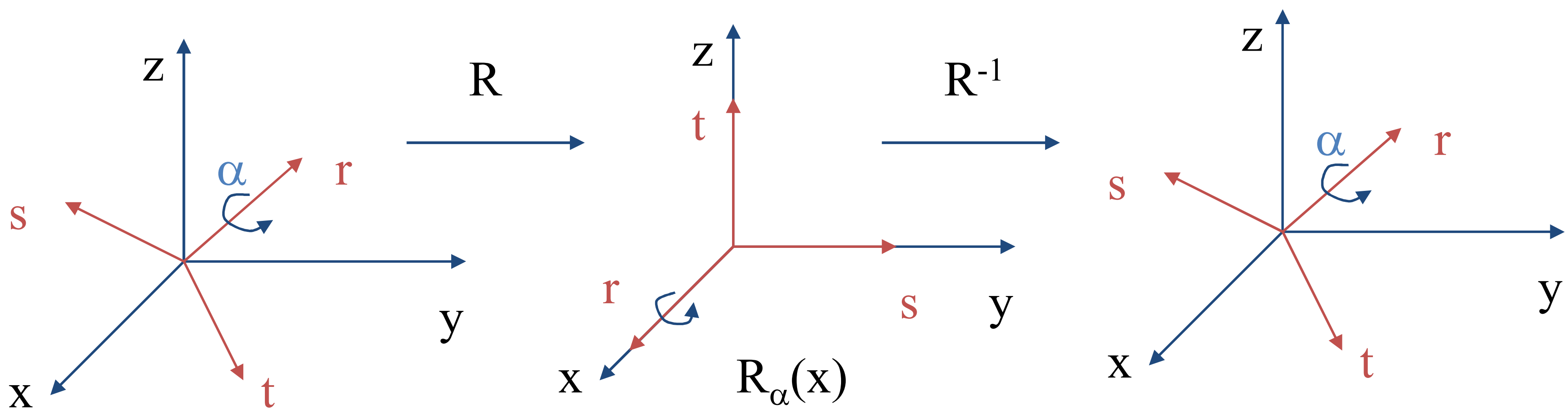
Rotation about an arbitrary axis

Computing **R**

- Define orthonormal basis $(\mathbf{r},\mathbf{s},\mathbf{t})$
  - First basis vector is $\mathbf{r}$
  - Second basis vector $\mathbf{s}$ is orthogonal to $\mathbf{r}$:

$$s = \frac{r \times e_x}{\left\| r \times e_x \right\|} \quad \text{or (if } r \| e_x \text{)} \quad s = \frac{r \times e_y}{\left\| r \times e_y \right\|}$$

  - Third basis vector $\mathbf{t} = \mathbf{r} \times \mathbf{s}$



Rotation about an arbitrary axis

Computing **R**

- Write vectors $(\mathbf{r},\mathbf{s},\mathbf{t})$ into the columns of the transformation matrix
- T-matrix is orthogonal and transforms
  - $\mathbf{e}_x \rightarrow \mathbf{r}$, $\mathbf{e}_y \rightarrow \mathbf{s}$, $\mathbf{e}_z \rightarrow \mathbf{t}$. (this is R^{-1})
  - For orthogonal matrices A the following holds $\mathrm{A}^{-1} = \mathrm{A}^{t}$
- Therefore: R is constructed by writing the vectors $(\mathbf{r},\mathbf{s},\mathbf{t})$ into the rows of the matrix

Rotation about an arbitrary axis

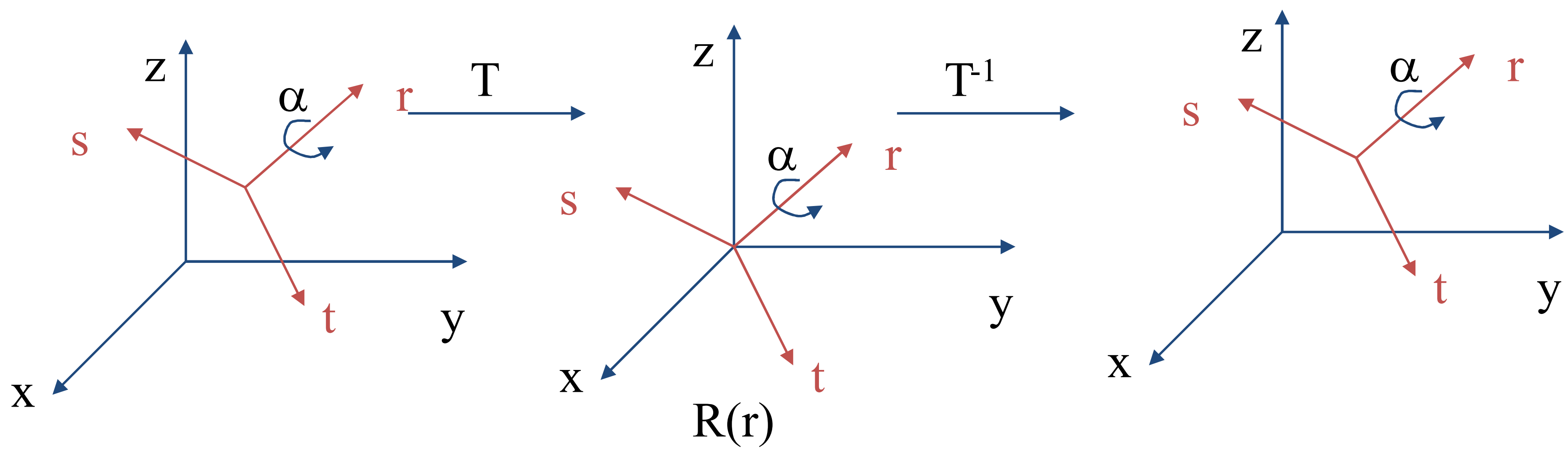
Computing **R**

- For clockwise rotation about the vector (x,y,z) by the angle α, using shorthands $s=\sin(\alpha)$, $c=\cos(\alpha)$ und $t=1\text{-}\cos(\alpha)$ the resulting matrix is given as

$$R_{(x,y,z)} = \begin{bmatrix} t\cdot x^2+c & t\cdot x\cdot y-s\cdot z & t\cdot x\cdot z+s\cdot y & 0 \\ t\cdot x\cdot y+s\cdot z & t\cdot y^2+c & t\cdot y\cdot z-s\cdot x & 0 \\ t\cdot x\cdot z-s\cdot y & t\cdot y\cdot z+s\cdot x & t\cdot z^2+c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

Rotation about an arbitrary point

- Axis of rotation through a point different from the origin
  - Move center of rotation to the origin
  - Perform rotation as previously described
  - Move center of rotation back



Rotation about an arbitrary point

- Example
  - Rotation in positive direction about an axis through the point (x_0, y_0, z_0) by angle α
  - The axis of rotation is the z-direction in this example

$$p' = \begin{bmatrix} 1 & 0 & 0 & x_0 \\ 0 & 1 & 0 & y_0 \\ 0 & 0 & 1 & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot p$$

Euler angles

orthogonal $R^T = R^{-1}$

pitch, yaw, roll

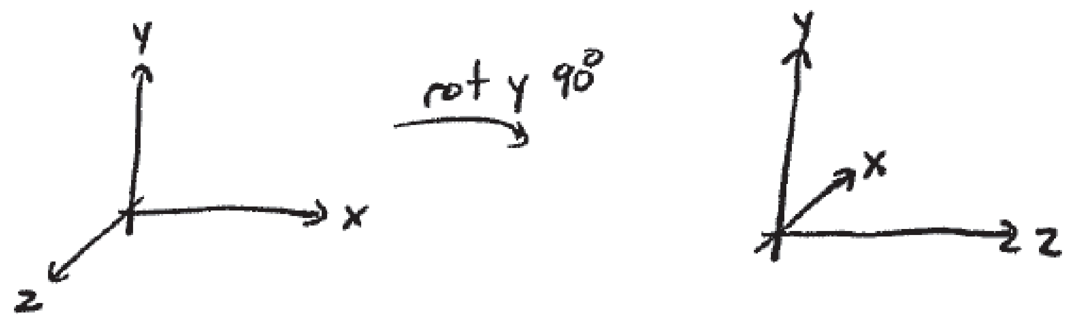
Euler angles: θ_x, θ_y, θ_z (in a particular order)

$R_z R_y R_x$ common

first

- Axis angle (previous slides) is preferred over Euler angles → Gimbal lock!

$R_z R_y R_x$:

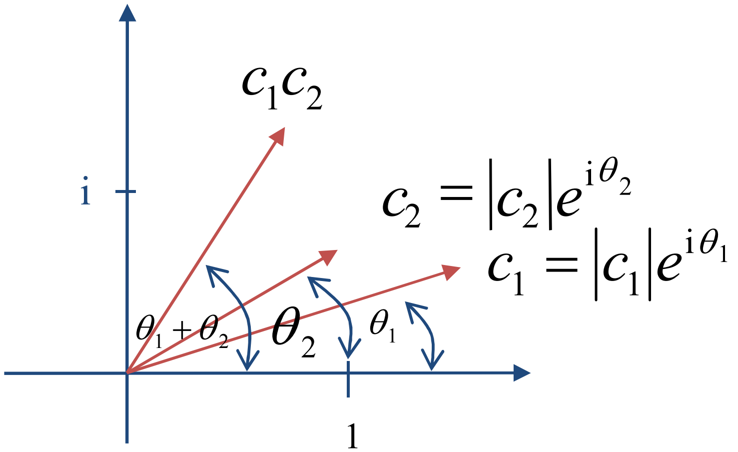


now, $R_x + R_z$ are redundant!

Excursion/aside: quaternions

- 4-dimensional analog to complex numbers
- Multiplication of complex numbers can describe orientation and rotation in 2D
- Complex numbers $c = a + ib = |c| \cdot e^{i\Theta}$
- Multiplication represents a similarity transformation

$$c_1 \cdot c_2 = |c_1| \cdot |c_2| \cdot e^{i(\Theta_1 + \Theta_2)}$$



Excursion/aside: quaternions

- Definition
  - Three imaginary numbers: i, j, k
  - $\mathbf{q} = a + bi + cj + dk$
  - Multiplication rules
    - $i^2 = j^2 = k^2 = -1$
    - $ij = -ji = k$
    - $jk = -kj = i$
    - $ki = -ik = j$
  - Careful: multiplication is not commutative!

Excursion/aside: quaternions

Properties

- Quaternions can be split into real and imaginary Parts

$$q = (s, \vec{v}) = s + v_1\,\mathrm{i} + v_2\,\mathrm{j} + v_3\,\mathrm{k}$$

- Multiplication

$$q_1 q_2 = (s_1 s_2 - \vec{v}_1 \cdot \vec{v}_2, s_1 \vec{v}_2 + s_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2)$$

- Conjugate

$$\overline{q} = (s, -\vec{v})$$

- Norm

$$\|q\| = \sqrt{s^2 + v_1{}^2 + v_2{}^2 + v_3{}^2}$$

Rotations and quaternions

- Points in space can be represented as purely imaginary quaternions

$$\mathbf{q}_p = (0, \mathbf{p}) = p_1\,\mathrm{i} + p_2\,\mathrm{j} + p_3\,\mathrm{k}$$

- Rotation of $\mathbf{p}$ about the origin
  - $\mathbf{q_p}' = \mathbf{q_r}\mathbf{q_p}\mathbf{q_r}^{-1}$, where $\mathbf{q}_i$ is a unit quaternion
- Inverse
  - For unit quaternions (as well as for complex numbers) $q^{-1} = \overline{q}$
  - The inverse of a unit quaternion is equal to its conjugate

Rotations and quaternions

- Unit quaternions are isomorph to orientations
- Unit quaternions can be expressed as

$$\mathbf{q}_r = \left(\cos(\alpha), \sin(\alpha)\mathbf{v}\right)$$

with unit vector $\vec{v}$

- $\mathbf{q_r}$ is equivalent to a rotation of angle 2α about the axis

$$\mathbf{q_{p'}} = \mathbf{q_r}\,\mathbf{q_p}\,\mathbf{q_r}^{-1}$$

Composition of transformations

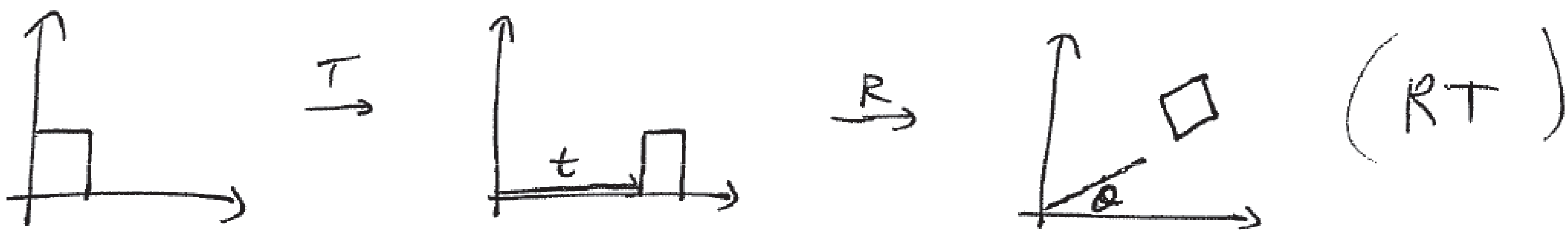
- We can compose the basic operations

$$M = M_1 M_2 \quad \text{(matrix mult.)}$$

$$TR_z S = \begin{bmatrix} s_x \cos\theta_z & -s_y \sin\theta_z & 0 & t_x \\ s_x \sin\theta_z & s_y \cos\theta_z & 0 & t_y \\ 0 & 0 & s_z & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

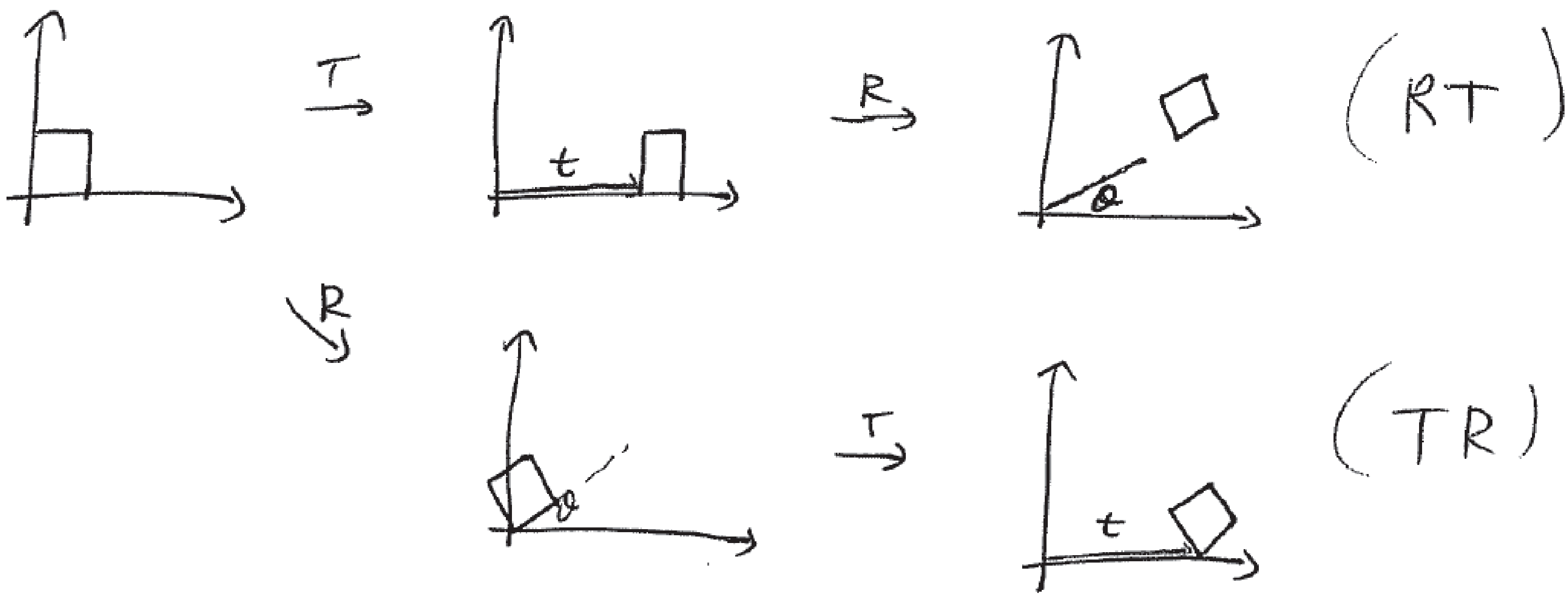
Composition of transformations

- In general, transformations do not commute!



Composition of transformations

- In general, transformations do not commute!



$TR \neq RT$

Composition of transformations

- In general, transformations do not commute!

$$TR = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

rotated $\begin{pmatrix} t_x \\ t_y \end{pmatrix}$

$$RT = \begin{bmatrix} \cos\theta & -\sin\theta & \cos\theta\, t_x - \sin\theta\, t_y \\ \sin\theta & \cos\theta & \sin\theta\, t_x + \cos\theta\, t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Composition of transformations

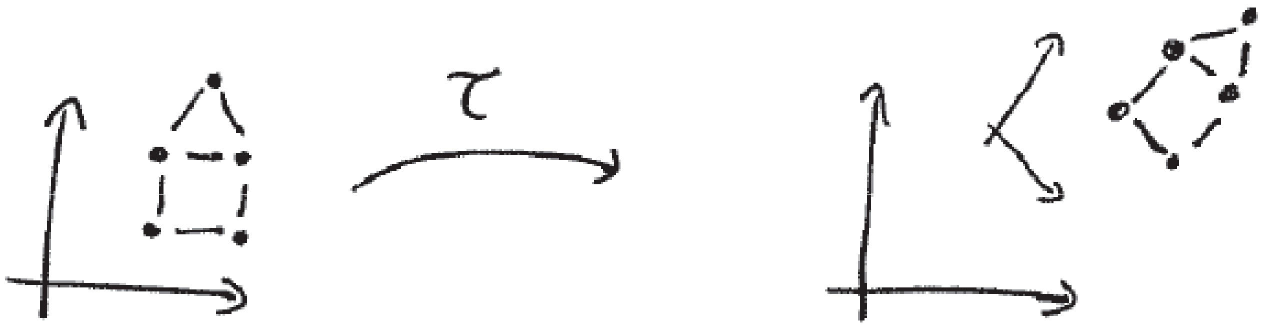
- Only commute in general
  - Any two translations
  - Two rotations **around the same axis**
  - Any two scales
  - Rotation and **uniform** scale

How is this implemented?

- Transform points + vectors
  - Original geometry (= positions in local coords) is left **unchanged!**
- Computations with transformed versions
- Use shape representations based on points and vectors
  - These are preserved under affine transformations

How is this implemented?

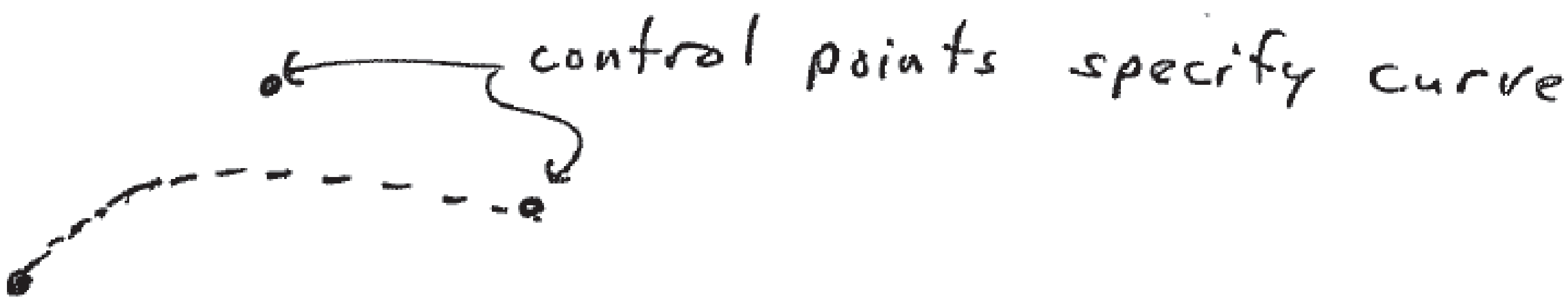
- Line segments



  - Affine transformations map lines to lines
  - So just transform the vertices (points) and connect the transformed points

How is this implemented?

- Curves and surfaces work too



  - Works since shape is built using multiple linear interpolations
(transformed curve = curve produced using transformed points)
- Some nonlinear deformations work this way

In OpenGL

- Maintain the "current" affine transformation
  - This is simply a single 4x4 matrix
  - All specified points (using `glVertex(…)`) are transformed by this matrix
  - OpenGL provides transformation functions for modifying this matrix

```
glLoadIdentity()

glTranslatef(x, y, z)
glRotatef(angle, x, y, z)
glScalef(x, y, z)
```

right multiplication

M_1

$M_1 M_2$

In OpenGL

- Maintain the "current" affine transformation
  - Matrix stack (incl. push and pop operations) to maintain a list of matrices
  - Top matrix is "current" modelview matrix

transform()
draw something()
:
push() transform again()
draw something else()
same
trans.
pop()
:
keep drawing something()