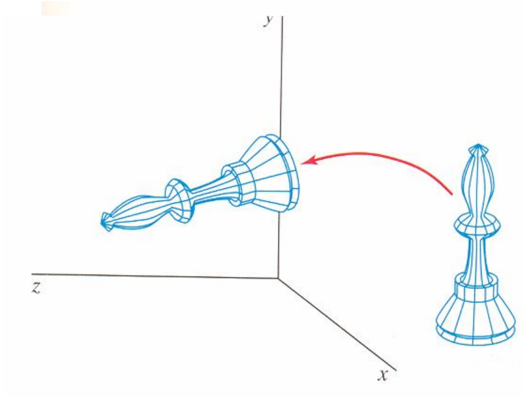
Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

3D Transformations



Computação Visual

Beatriz Sousa Santos, J. Madeira

Overview

- 3D Translation
- 3D Scaling
- 3D Rotations
- Other Transformations: Simmetry / Shearing
- Concatenating Transformations
- Transformations in OpenGL / WebGL
- Application Examples

3D TRANSFORMATIONS

• 3D transformations are a generalization of the known 2D transformations

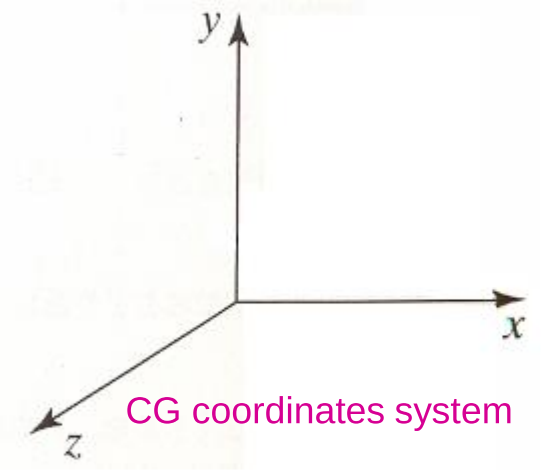
• In 2D, rotations were carried out on the XOY plane (i.e., around axes perpendicular to that plane)

• In 3D, rotations can be carried out around any 3D axis

• A point defined in the 3D space is represented, in homogeneous coordinates, by a column-vector with 4 elements

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

$$P_h = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

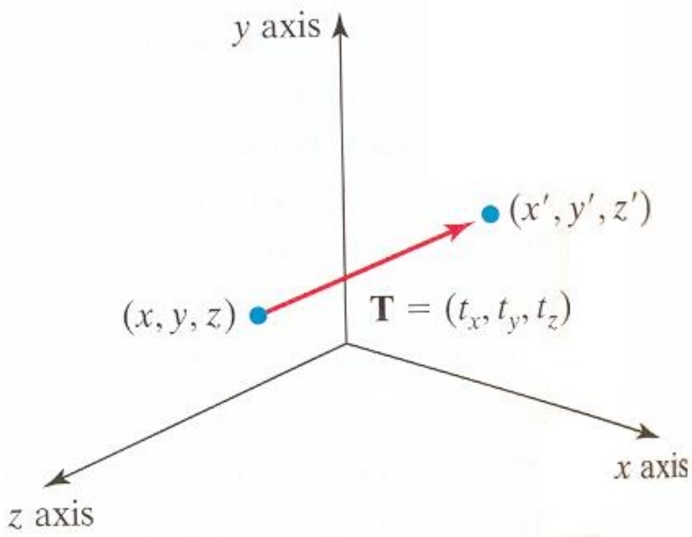


CG coordinates system

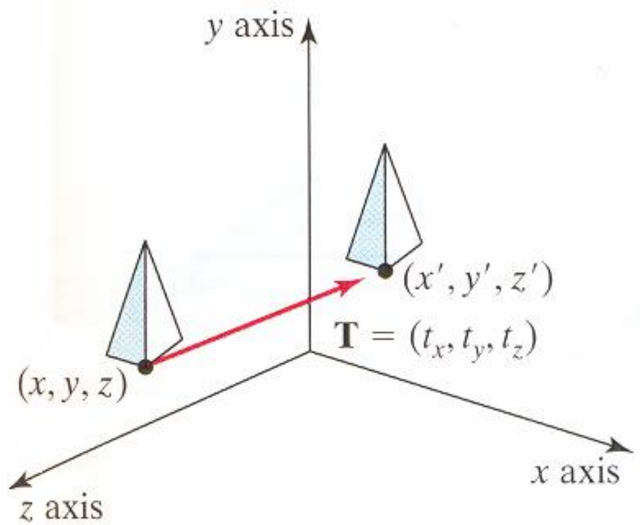
3D TRANSLATION

Translation

$$x' = x + t_x \qquad y' = y + t_y \qquad z' = z + t_z$$



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



$$\mathbf{P'} = \mathbf{T} \cdot \mathbf{P}$$

3D SCALING

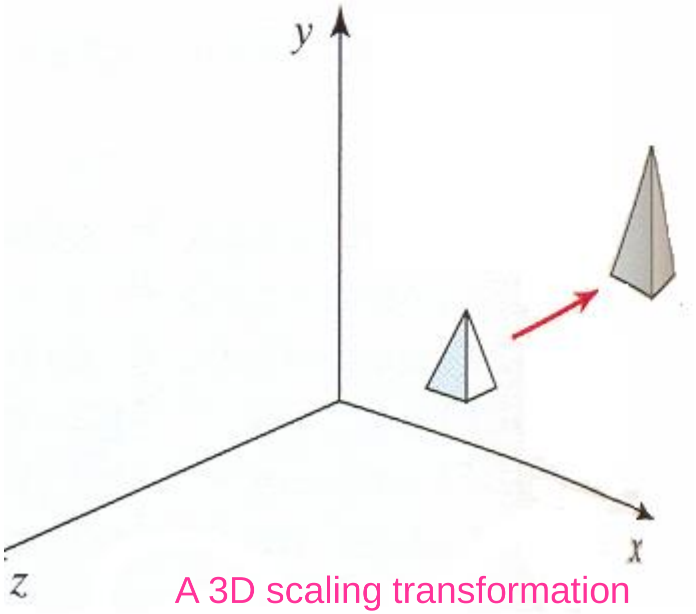
Scaling

- The 3D scaling matrix is a generalization of the 2D scaling matrix

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

$$\mathbf{P}' = \mathbf{S} \cdot \mathbf{P}$$

$$x' = s_x \,.\, x \;,\quad y' = s_y \,.\, y \;,\quad ´z' = s_z \,.\, z$$



A 3D scaling transformation might change the position of the resulting, transformed object.

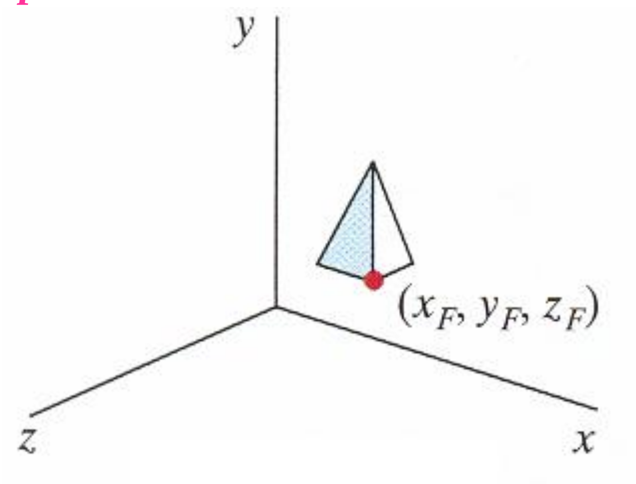
Scaling relative to a fixed point (x_p, y_p, z_p)

- A 3D scaling transformation, relative to a fixed point, is made up of the following sequence of transformations:

1. Translation of the fixed point to the coordinates' origin

2. Scaling

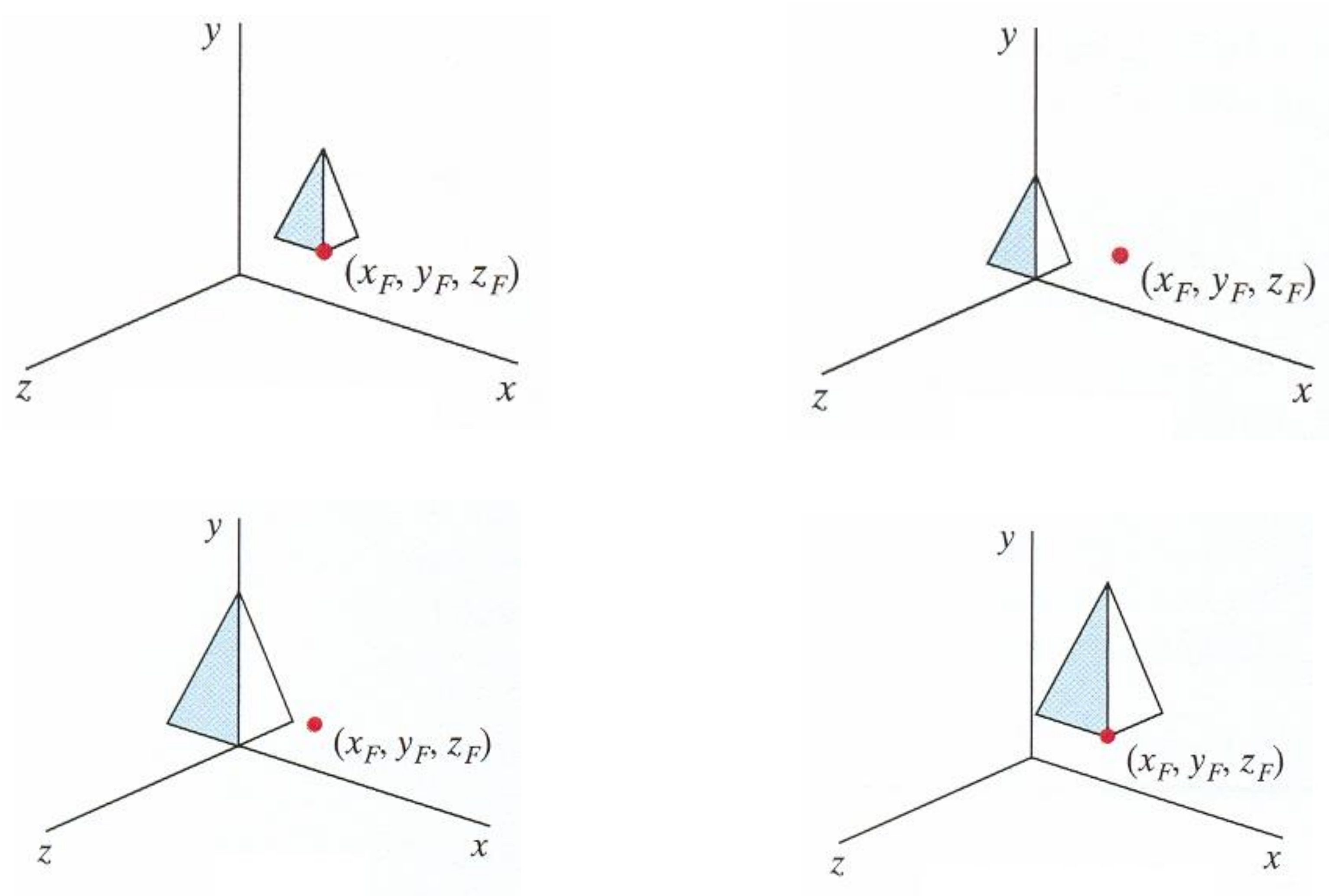
3. Translation of the fixed point back to its original position



After the last transformation, the fixed point is back at its original position

$$\mathbf{T}(x_f, y_f, z_f)\cdot\mathbf{S}(s_x, s_y, s_z)\cdot\mathbf{T}(-x_f, -y_f, -z_f) = \begin{bmatrix} s_x & 0 & 0 & (1-s_x)x_f \\ 0 & s_y & 0 & (1-s_y)y_f \\ 0 & 0 & s_z & (1-s_z)z_f \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling relative to a fixed point (x_p, y_p, z_p)



3D ROTATIONS

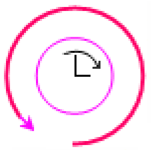
Rotations

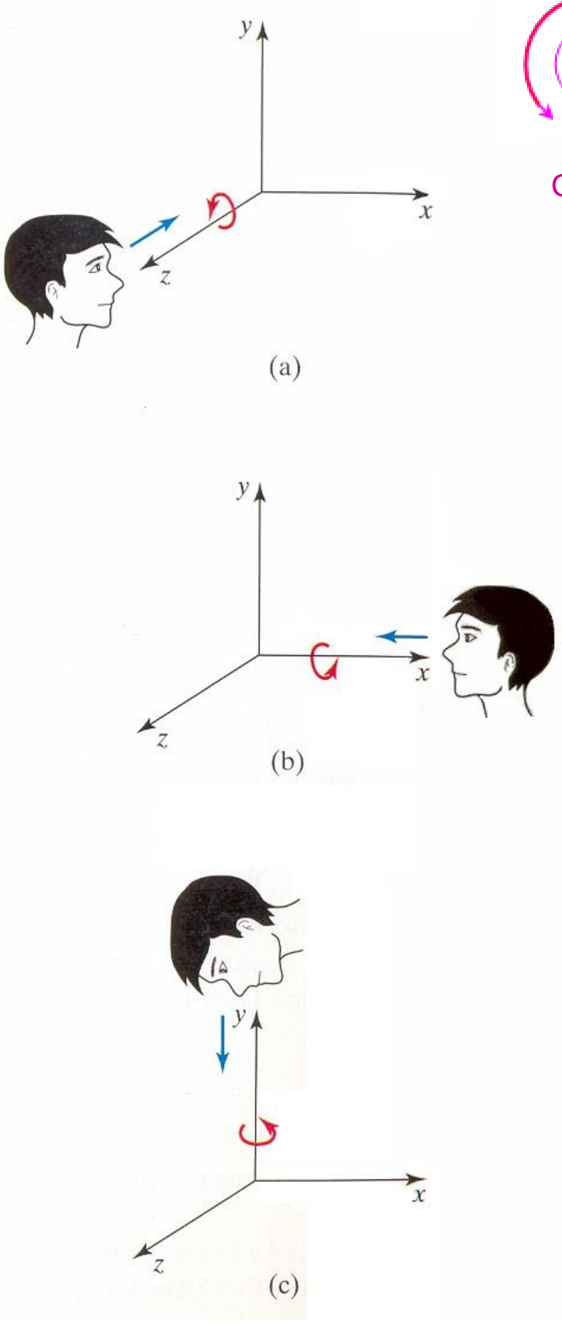
- A model can be rotated around any 3D axes, but rotations around one of the coordinate axes are simpler

- Convention:

  rotation angle > 0 → CCW rotation

- It is the same convention as established for 2D (on plane XOY, around the origin)





Rotations around the coordinate axes

• The 2D rotation transformation on the XOY plane, around the origin, is easily generalized to 3D

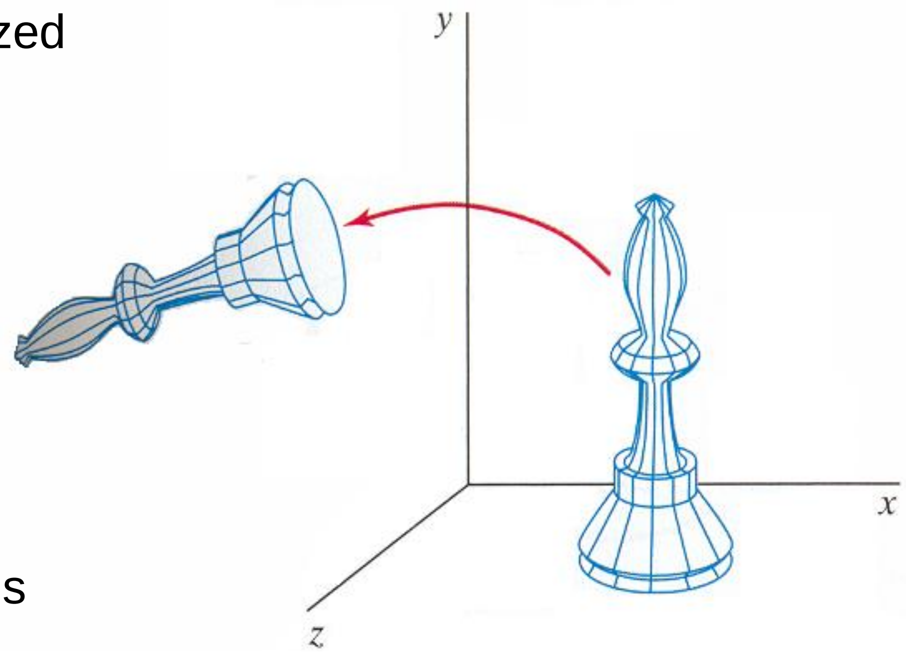
$$x' = x \cos \theta - y \sin \theta$$

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

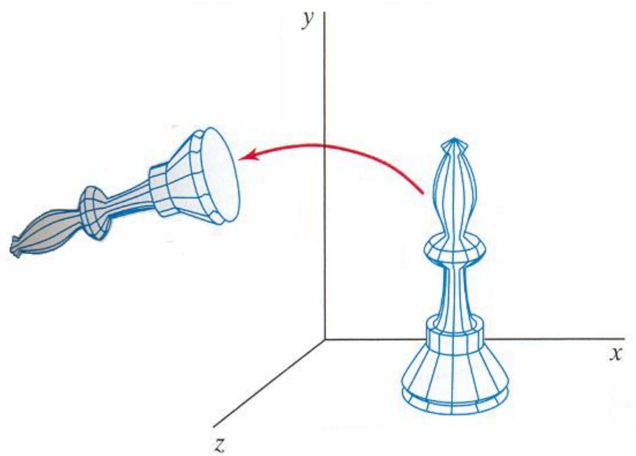
$$z' = z$$

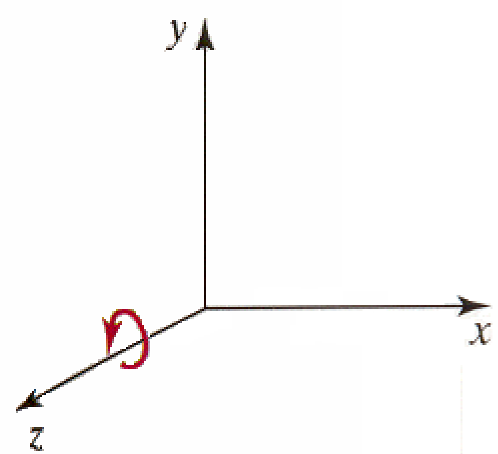
• θ is the rotation angle around the ZZ' axis

• For any rotated point, the z coordinate remains unchanged



Rotation around the *ZZ'* axis





- Generalization to 3D:

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

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

$$z' = z$$

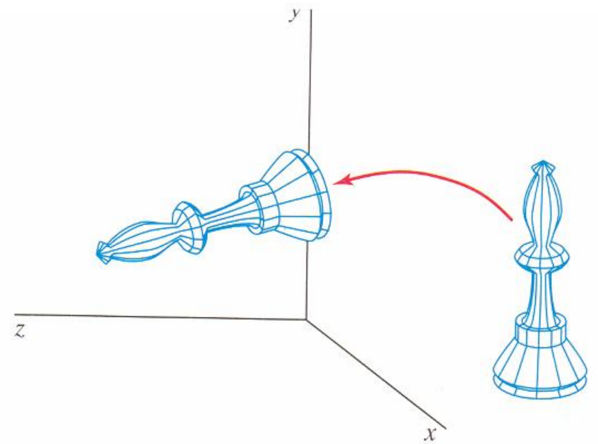
- Matricial representation:

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

$$\mathbf{P}' = \mathbf{R}_z(\theta) \cdot \mathbf{P}$$

z coordinates
remain unchanged

Rotation around the *XX'* axis



- The equations / matrices for the rotations around the other coordinate axes can be obtained through a cyclic permutation of ***x***, ***y*** and ***z***:

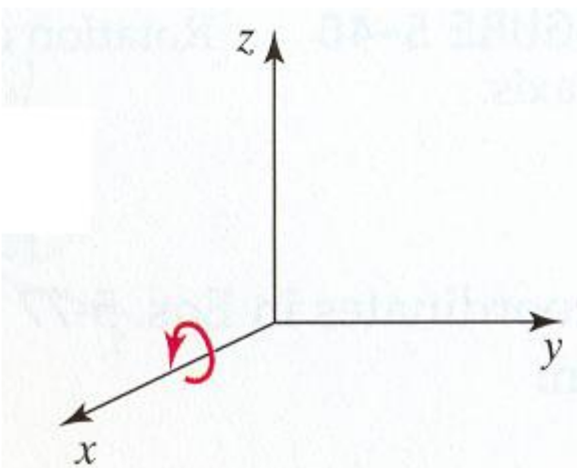
$$x \rightarrow y \rightarrow z \rightarrow x$$

- For the rotation around the axis *XX'* axis:

$$y' = y \cos\theta - z \sin\theta$$

$$z' = y \sin\theta + z \cos\theta$$

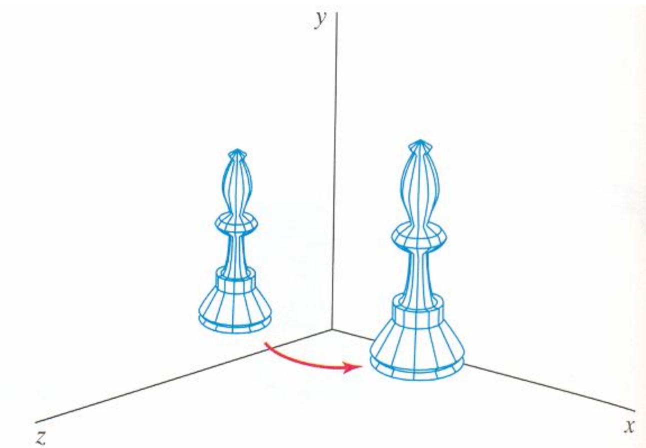
$$x' = x$$



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

x coordinates remain unchanged

Rotation around the ***YY'*** axis

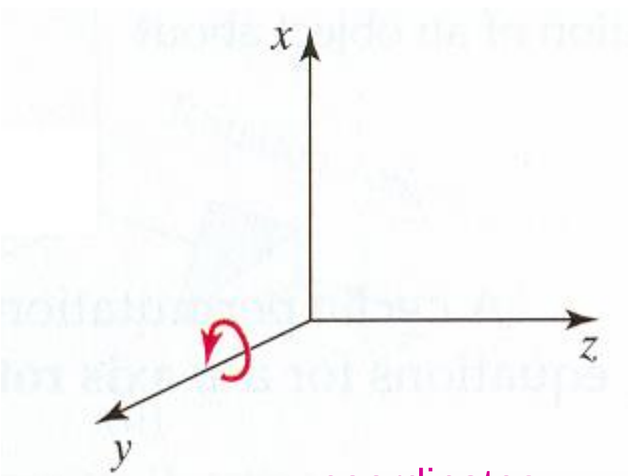


- With a similar permutation, we obtain the rotation around the ***YY'*** axis:

$$z' = z\cos\theta - x\sin\theta$$

$$x' = z\sin\theta + x\cos\theta$$

$$y' = y$$

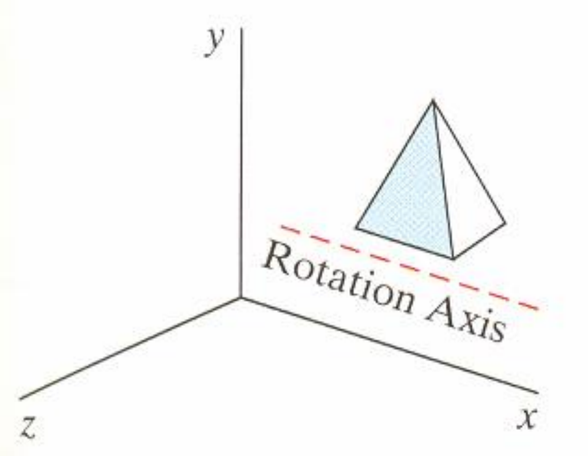


$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

y coordinates
remain unchanged

General 3D rotations

- The matrix representing the rotation around an axis parallel to a coordinate axis corresponds to the following sequence of transformations:



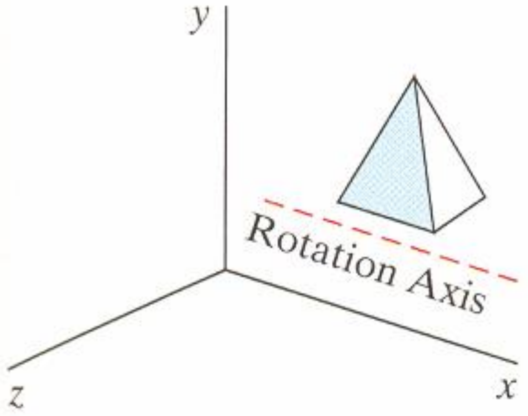
1. Translation to make the rotation axis coincide with the coordinate axis

2. Rotation around the coordinate axis

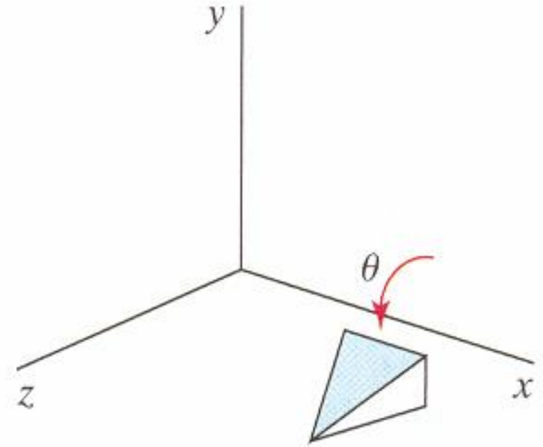
3. Inverse translation back to the original position of the rotation axis

$$\mathbf{P}' = \mathbf{T}^{-1} \cdot \mathbf{R}_x(\theta) \cdot \mathbf{T} \cdot \mathbf{P}$$

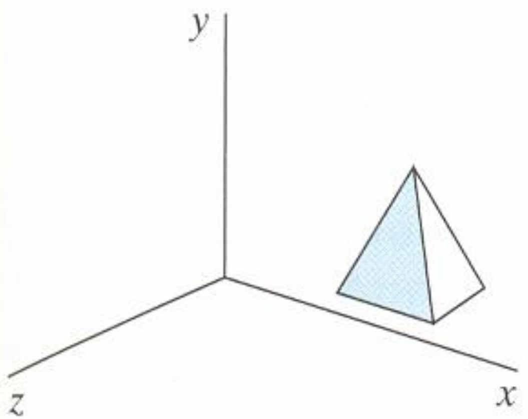
Rotation around an axis parallel to a coordinate axis



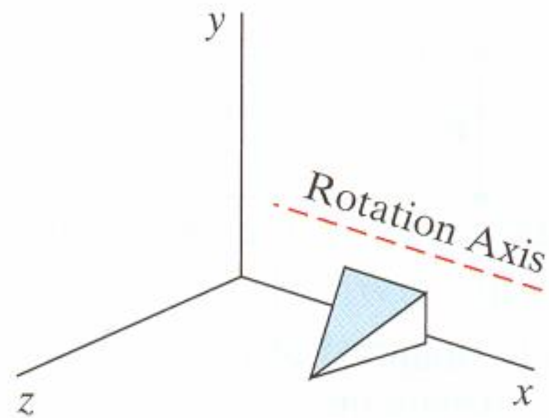
Initial position



2 - Rotation

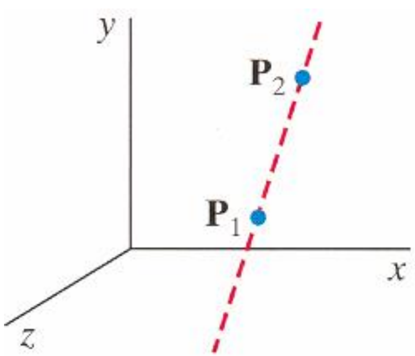


1 - Translation

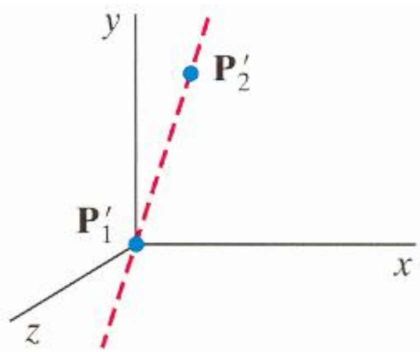


3 – Inverse translation

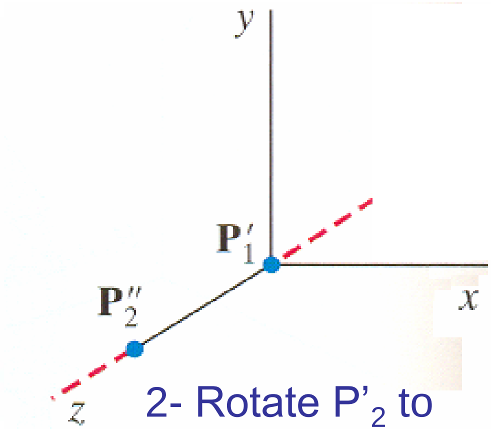
Rotation around an arbitrary axis



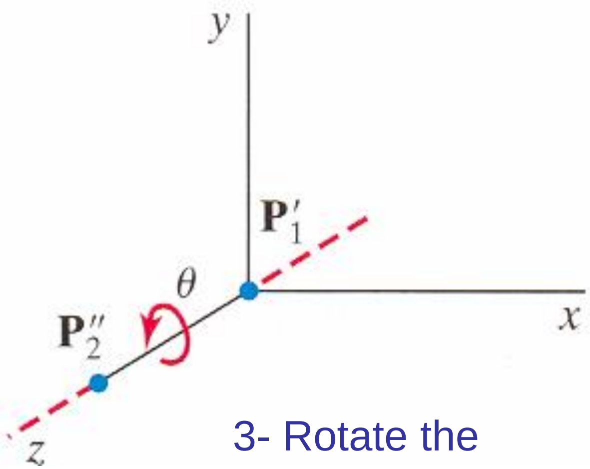
Initial position



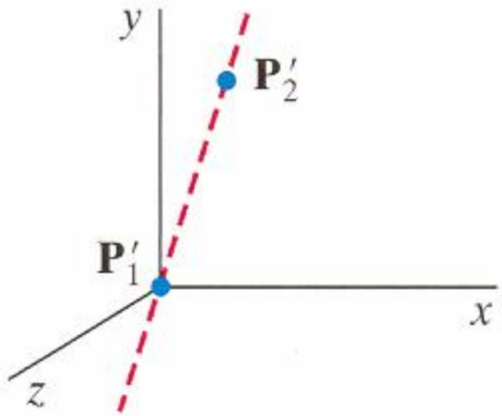
1- Translate P_1 to the origin



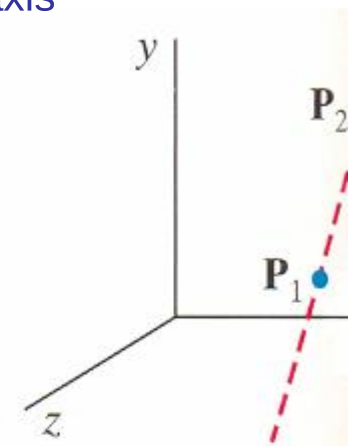
2- Rotate P'_2 to coincide with the ***ZZ'*** axis



3- Rotate the object around the ***ZZ'*** axis



4- Rotate axis back to initial orientation



5- Translate back to original position

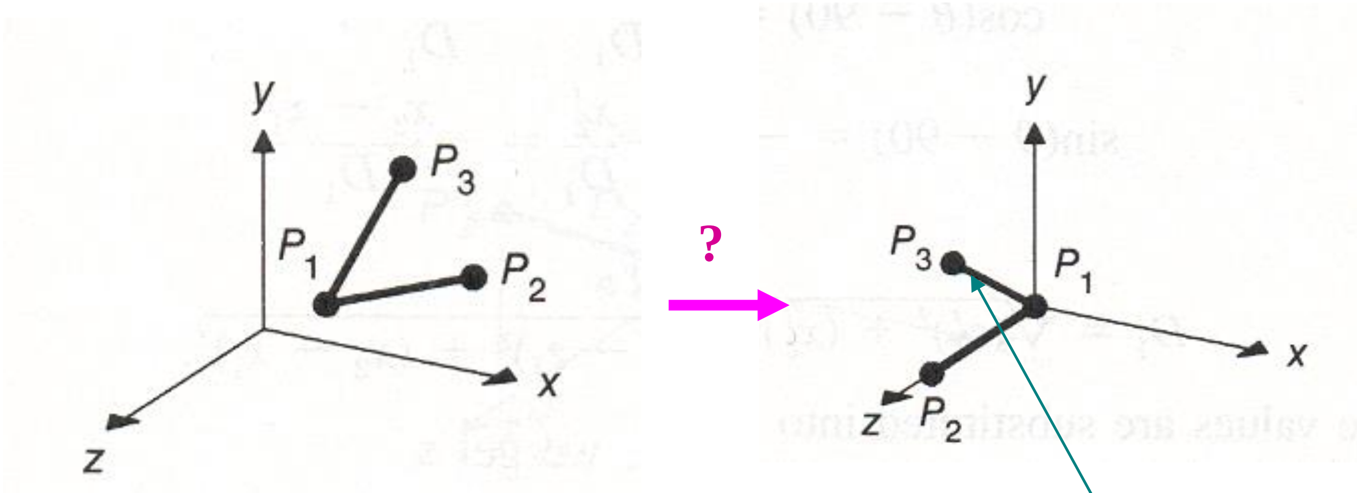
A MORE GENERAL EXAMPLE

A more complex rotation example

- Determine the transformation that moves the straight-line segments from position a) to position b)

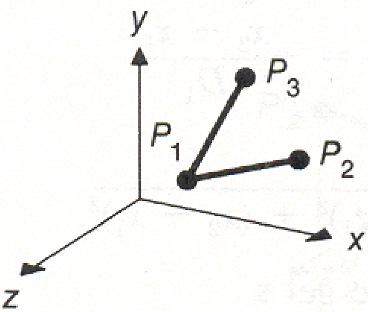
a) Initial position

b) Final position



$\boldsymbol{P_1P_3}$ belongs to the $\boldsymbol{YOZ}$ plane

- Decompose the problem into a sequence of simples problems, for which the transformation matrices are known:

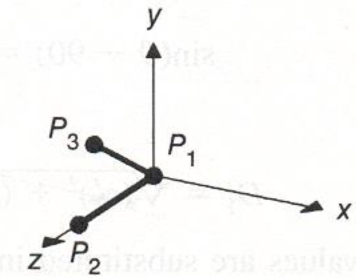


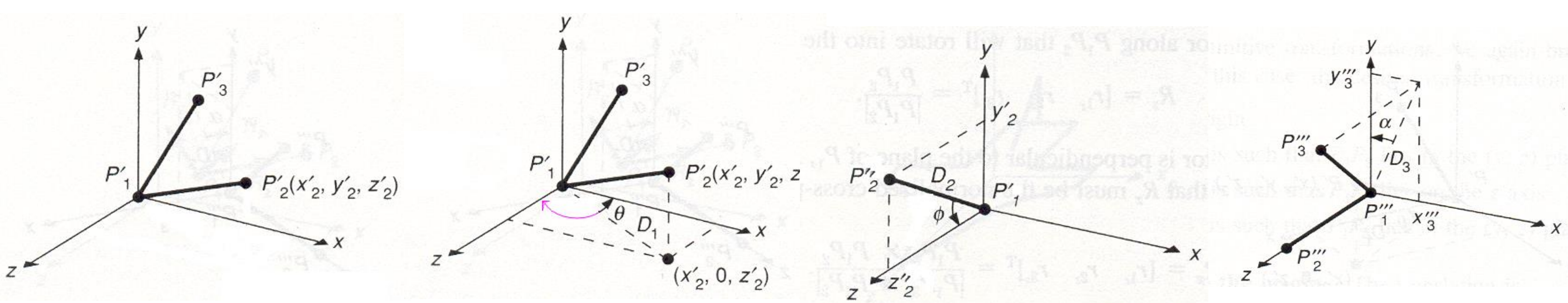
1- Translate P_1 to the origin of the coordinate axes

2- Rotate around *YY'* to move P_1 P_2 to the *YOZ* plane

3- Rotate around *XX'* to move P_1 P_2 to the *ZZ'* axis

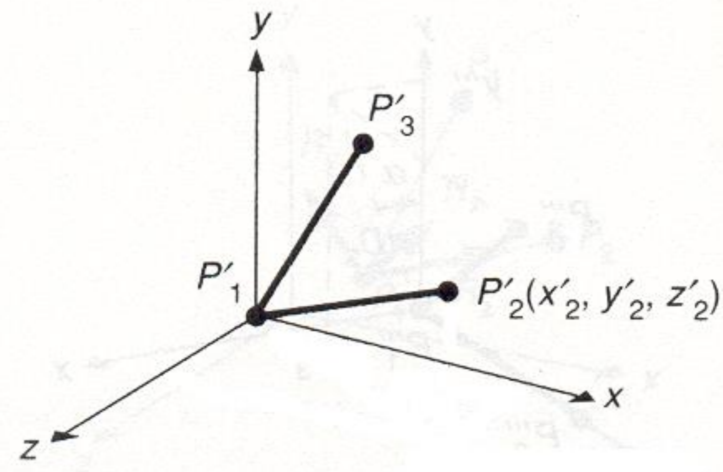
4- Rotate around *ZZ'* to move P_1 P_3 to the *YOZ* plane





1 **2** **3** **4**

1- Translate P_1 to the origin



$$T(-x_1, -y_1, -z_1) = \begin{bmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

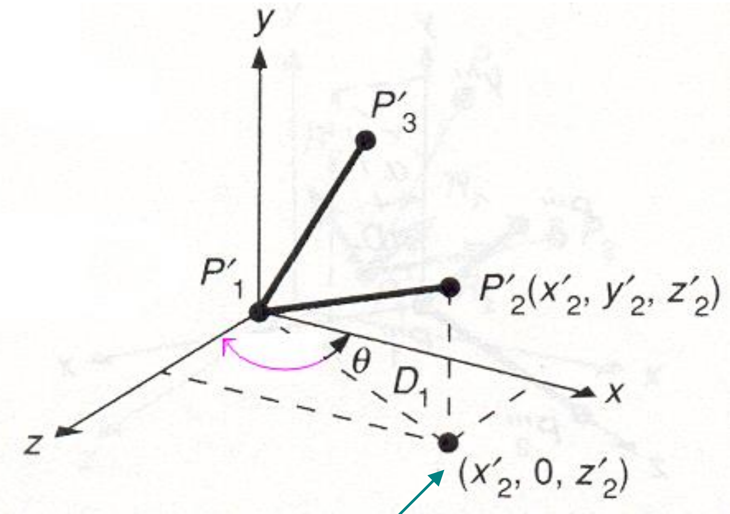
$$P'_1 = T(-x_1, -y_1, -z_1) \cdot P_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$P'_2 = T(-x_1, -y_1, -z_1) \cdot P_2 = \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \\ z_2 - z_1 \\ 1 \end{bmatrix}$$

$$P'_3 = T(-x_1, -y_1, -z_1) \cdot P_3 = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \\ z_3 - z_1 \\ 1 \end{bmatrix}$$

2- Rotate around ***YY'*** to move $P_1 P_2$ to the ***YOZ*** plane

Rotation angle: $-(90-\theta) = \theta - 90$



$$\cos(\theta - 90) = \sin\theta = \frac{z_2'}{D_1} = \frac{z_2 - z_1}{D_1}$$

$$\sin(\theta - 90) = -\cos\theta = -\frac{x_2'}{D_1} = -\frac{x_2 - x_1}{D_1}$$

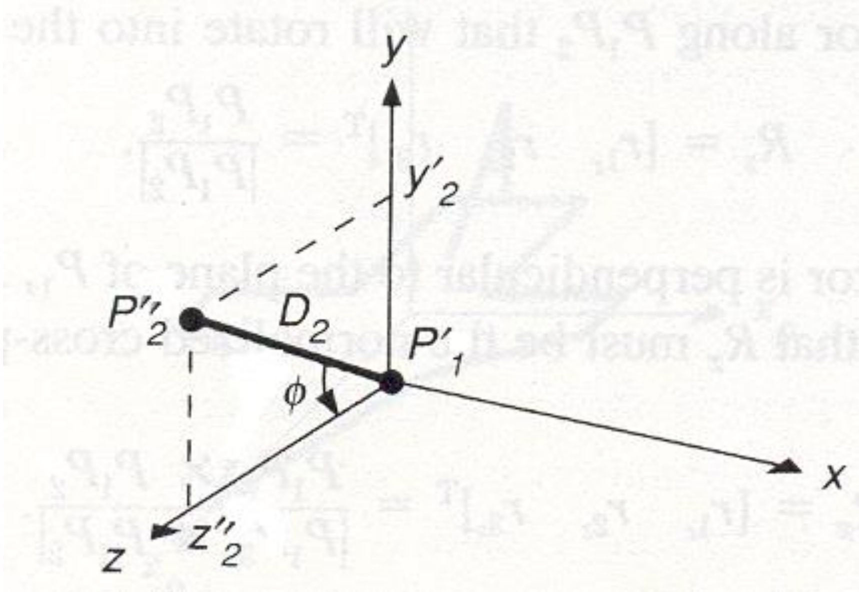
Projection of ***P₂'*** on ***XOZ*** plane

$$D_1 = \sqrt{(z_2')^2 + (x_2')^2} = \sqrt{(z_2 - z_1)^2 + (x_2 - x_1)^2}$$

$$P_2'' = R_y(\theta - 90) \cdot P_2' = [0 \quad y_2 - y_1 \quad D_1 \quad 1]^{\mathrm{T}}$$

3- Rotate around ***XX'*** to move $P_1 P_2$ to the ***ZZ'*** axis

$$\cos\phi = \frac{z_2''}{D_2}, \ \sin\phi = \frac{y_2''}{D_2},$$



$$D_2 = |P_1'' P_2''|$$

$$D_2 = |P_1'' P_2''| = |P_1 P_2| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

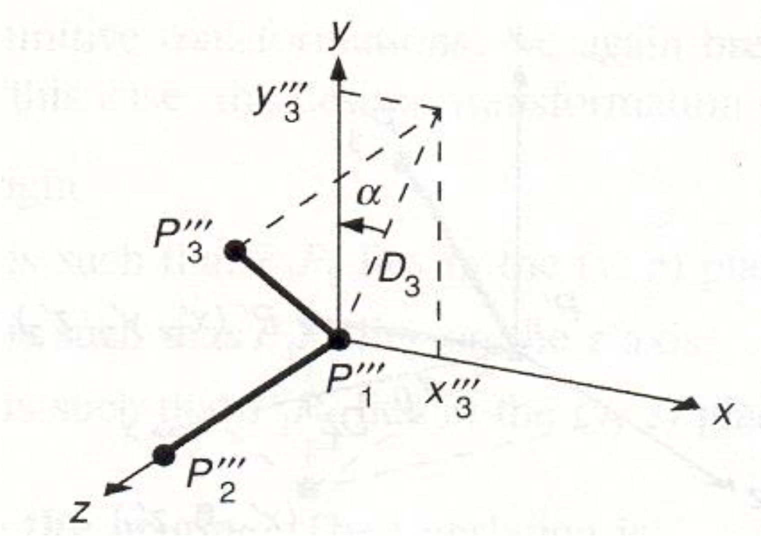
$$\begin{aligned} P_2''' &= R_x(\phi) \cdot P_2'' = R_x(\phi) \cdot R_y(\theta - 90) \cdot P_2' \\ &= R_x(\phi) \cdot R_y(\theta - 90) \cdot T \cdot P_2 = [0 \quad 0 \quad |P_1 P_2| \quad 1]^T \end{aligned}$$

4- Rotate around ***ZZ'*** to move $P_1\, P_3$ to the ***YOZ*** plane

$$\cos\alpha = y_3'''/D_3$$

$$\sin\alpha = x_3'''/D_3$$

$$D_3 = \sqrt{(x_3'''^2 + y_3'''^2)}$$



The global transformation is:

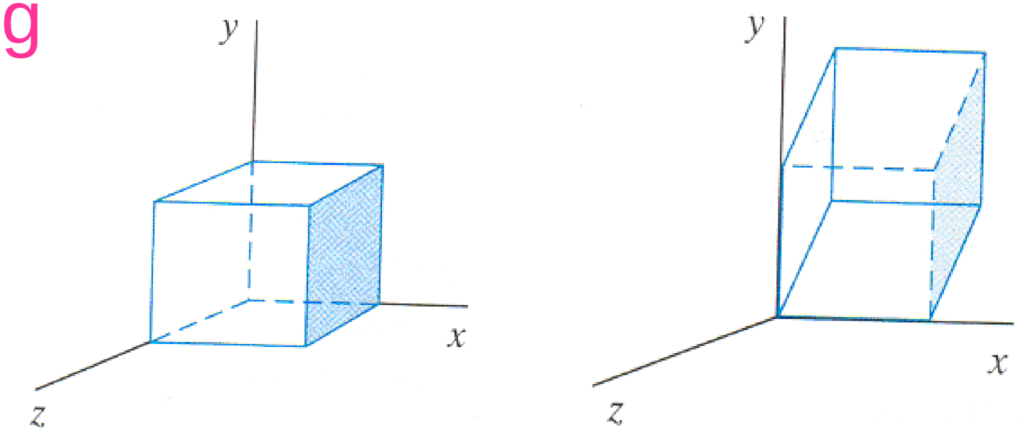
$$R_z(\alpha) \cdot R_x(\phi) \cdot R_y(\theta - 90) \cdot T(-x_1, -y_1, -z_1) = R \cdot T$$

SIMMETRY & SHEARING

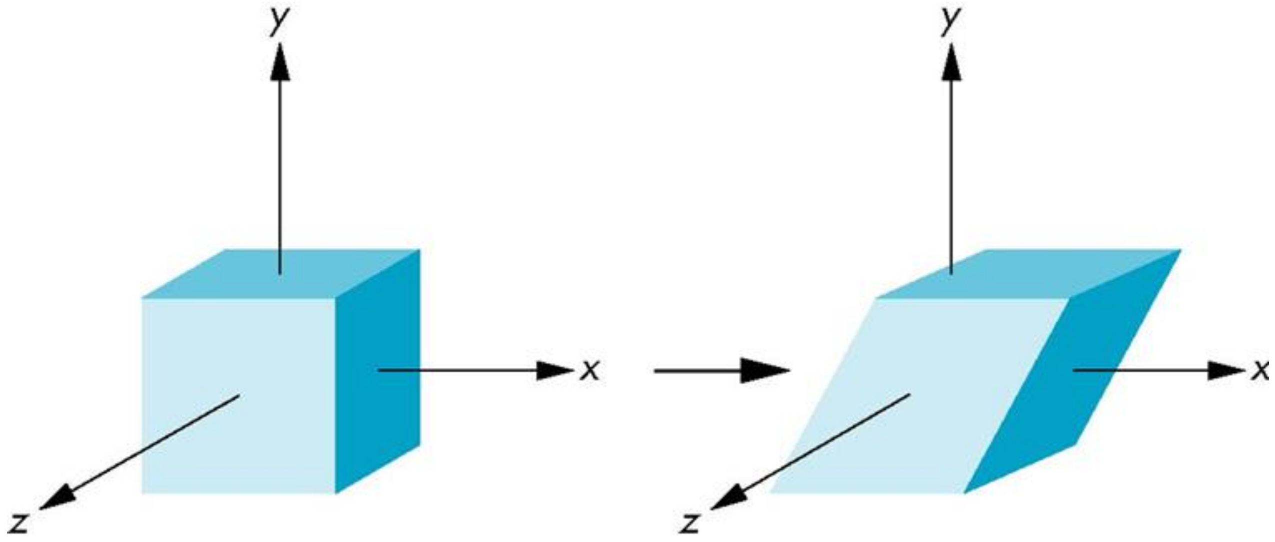
Other 3D transformations

- There are other useful 3D transformations:
  - *Shearings*
  - Symmetries

ZZ' shearing



Shearing

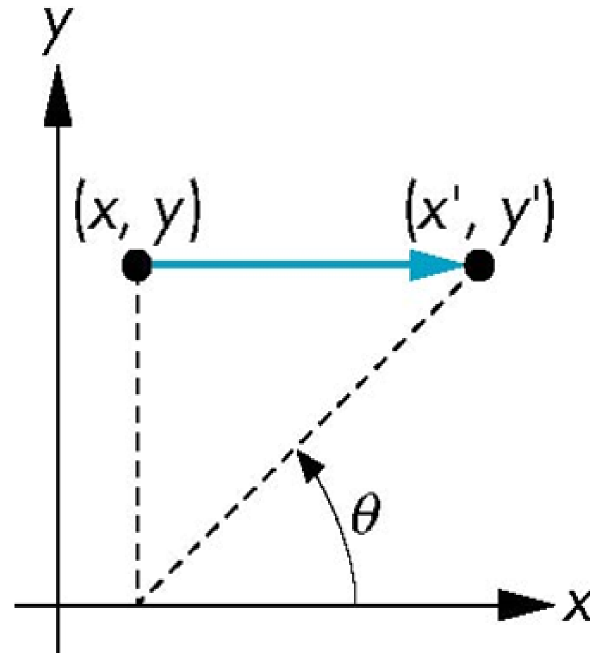


[Angel]

Shear matrix

$$x' = x + y \cot\theta$$
$$y' = y$$
$$z' = z$$

$$\mathbf{H}(\theta) = \begin{bmatrix} 1 & \cot\theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

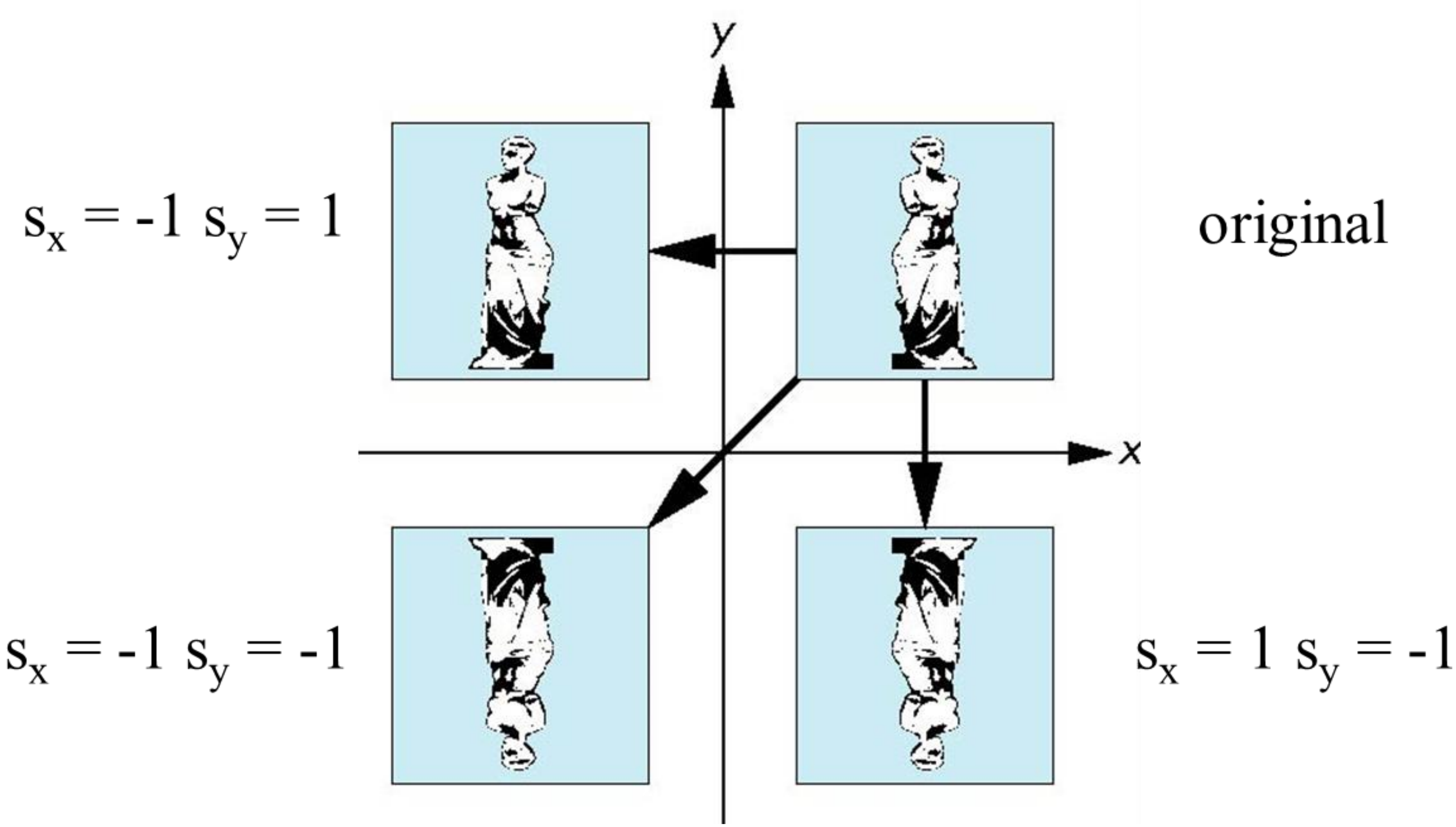


[Angel]

Symmetries

- Relative to the origin
- Relative to one of the coordinate planes
- Relative to particular planes
  - Parallel to one of the coordinate planes
  - Octant bissectors
  - …
- Relative to any plane

Symmetries



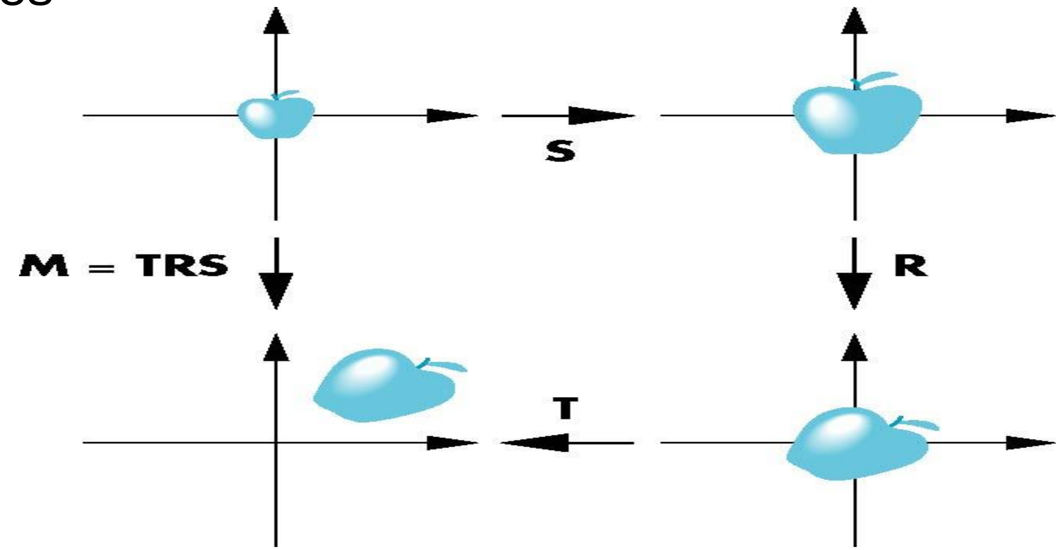
[Angel]

TRANSFORMATIONS IN OPENGL / WEBGL

Model instantiation

- Start with a model:
  - Centered at (0,0,0)
  - Oriented according to the coordinate axes
  - With a standard size

- Apply the global transformations Mi to create all required model instances



[Angel]

OpenGL (Pre-3.1)

- Concatenating of transformations
  - Multiply the existing global transformation matrix by the matrix corresponding to the next transformation to be applied

- `glTranslatef( x, y, z );`

- `glRotatef( angle, x, y, z );`
  - Rotation angle in degrees
  - (x, y, z) define the rotation axis that passes through (0, 0, 0)

- `glScalef( sx, sy, sz );`
  - Relative to (0, 0, 0) !!

OpenGL (Pre-3.1)

- Transformation order is important !!

```
glTranslatef( x1, y1, z1 );
glRotatef( 45, 0, 0, 1 );
glScalef( 2, 2, 2 );

glScalef( 2, 2, 2 );
glRotatef( 45, 0, 0, 1 );
glTranslatef( x1, y1, z1 );
```

- What is the difference ? What happens ?

OpenGL (Pre-3.1)

- It is possible to directly set transformation matrices

- And to save transformation matrices in a STACK
  - To apply hierarchical transformations
  - And animate / simulate the behavior of more complex models

- Distinguish between
  - PROJECTION mode
  - MODELING mode

OpenGL

- Concatenating transformations
  - Multiply the current (global) transformation matrix by another transformation matrix
- How to ?
  - Previous functions no longer exist !!
  - Create the (4x4) matrices
  - Multiply them in the correct order !!

OpenGL – Example

- Use auxiliary functions or a “math library”
- Create the identity matrix

```
mat4 m = Identity();
```

- Right-multiplication by a rotation matrix

```
mat4 r = Rotate(theta, vx, vy, vz);
m = m * r;
```

OpenGL - Example

- Similar for the other basic transformations

```
mat4 s = Scale( sx, sy, sz);

mat4 t = Translate(dx, dy, dz);

m = m * s * t;
```

OpenGL - Example

- Rotation of 30 degrees about an axis parallel to the ZZ' axis

```
mat 4 m = Identity();
m = Translate(1.0, 2.0, 3.0)*
    Rotate(30.0, 0.0, 0.0, 1.0)*
    Translate(-1.0, -2.0, -3.0);
```

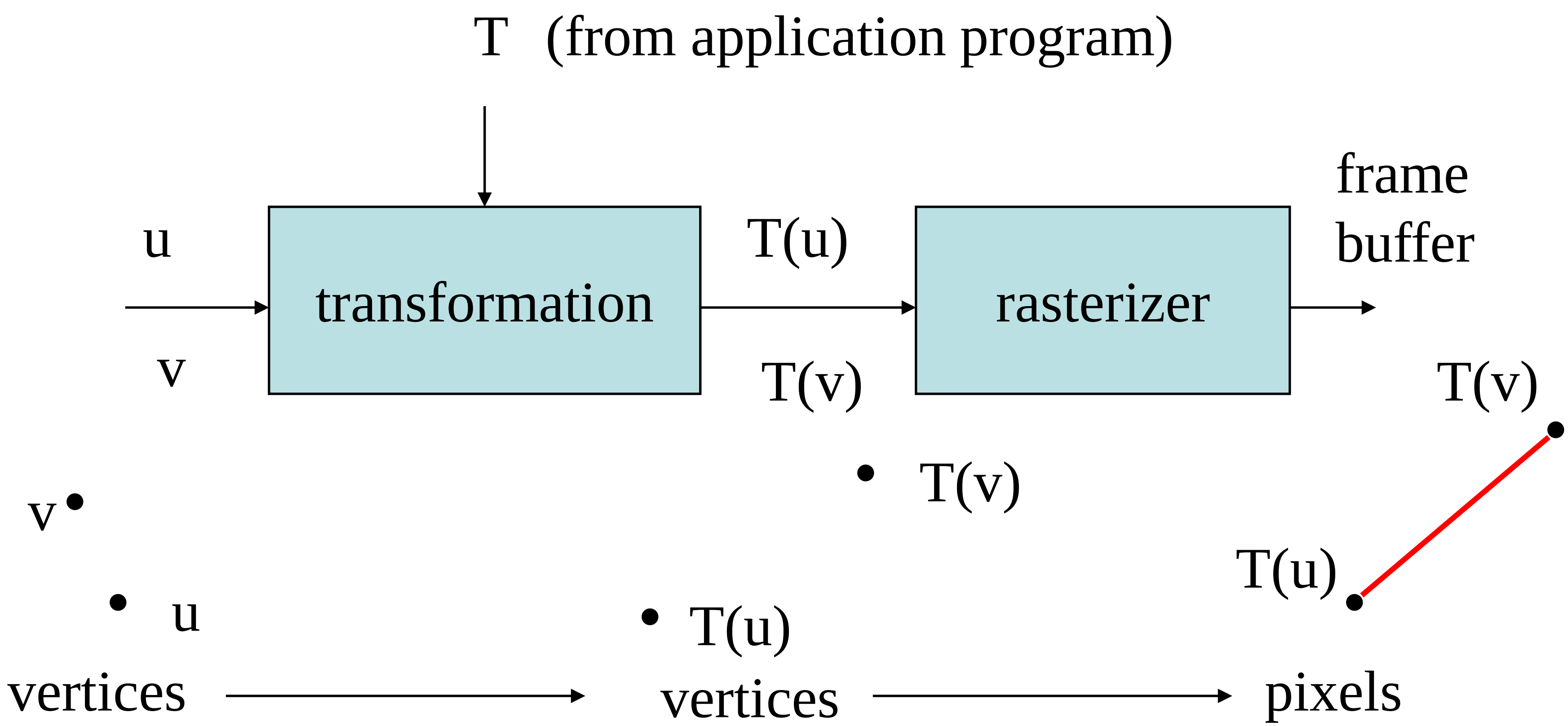
- The "last" matrix is the first one to be applied !!

OpenGL / WebGL

- Transformation matrices are stored as a 1D array with 16 elements.

- The elements of the 4 x 4 matrix are stored column-by-column.

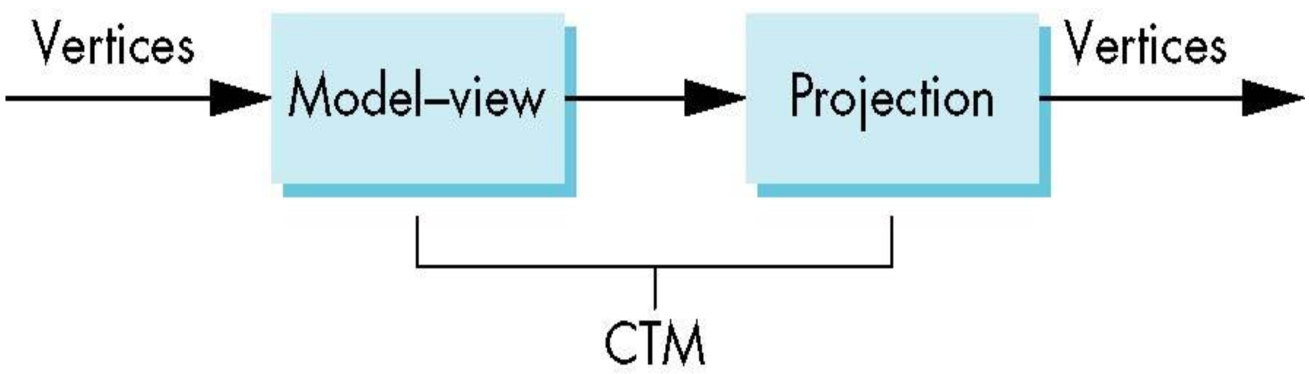
WebGL – Current transformation matrix



[Angel]

WebGL – Current transformation matrix

- Emulate the traditional OpenGL process
  - Model-View matrix
  - Projection matrix



[Angel]

Transformation matrices

- Identity matrix

```
var m = mat4();
```

- Concatenating transformations

```
var r = rotate(theta,vx,vy,vz);
m = mult(m,r);
var s = scale(sx,sy,sz);
m = mult(m,s);
```

Example

- Rotation about ZZ, by 30 degrees, with a fixed point (1.0,2.0,3.0)

```
var m = mult( translate(1.0,2.0,3.0),
              rotate(30,0.0,0.0,1.0) );
m = mult( m, translate(-1.0,-2.0,-3.0) );
```

- Multiplication order ?

TASK

Application problem (see PDF)

3- Consider the cube defined by the vertices:

V_1 (0, 0, 0)	V_2 (0, 1, 0)	V_3 (1, 1, 0)	V_4 (1, 0, 0)
V_5 (0, 0, 1)	V_6 (1, 0, 1)	V_7 (1, 1, 1)	V_8 (0, 1, 1)

Using *Homogeneous Coordinates*, determine the matrix that represents the transformation that is to be applied for the cube to rotate, by a 180 degrees angle, around the straight-line that passes through point (2, 0, 0) is parallel to the *YY'* axis.

REFERENCES

References

- D. Hearn and M. P. Baker, *Computer Graphics with OpenGL*, 3rd Ed., Addison-Wesley, 2004

- E. Angel and D. Shreiner, *Introduction to Computer Graphics,* 6th Ed., Pearson Education, 2012

- J. Foley et al., *Introduction to Computer Graphics*, Addison-Wesley, 1993