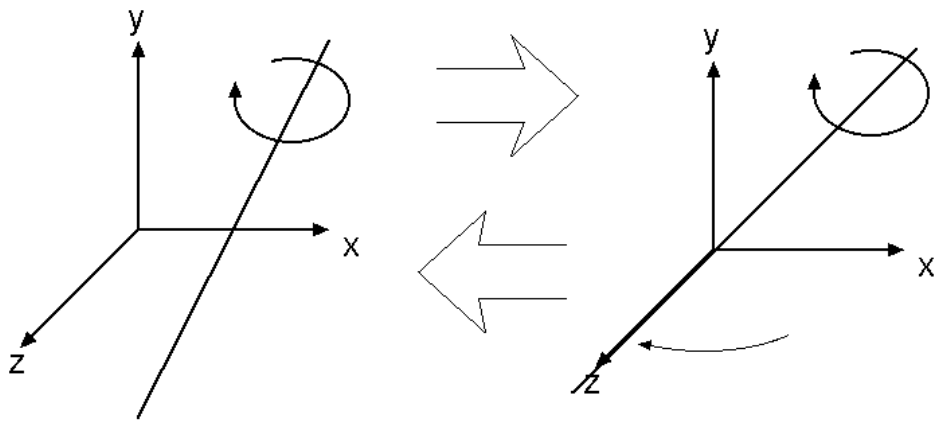
Rotation around arbitrary axis

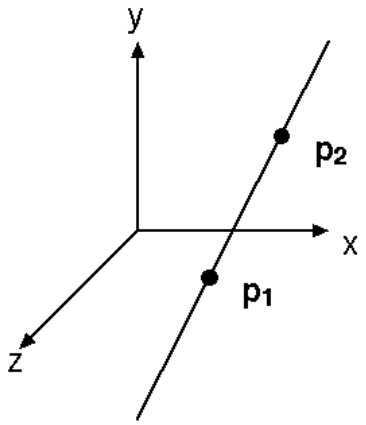
Two methods

Rotation around arbitrary axis



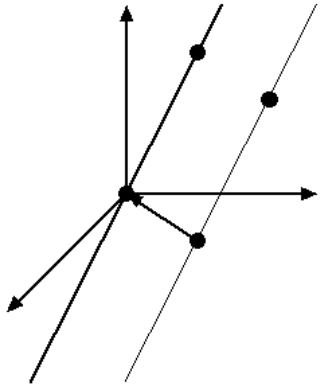
Transform to align axis with the Z axis, rotate, and transform back.

Definition of the rotation axis



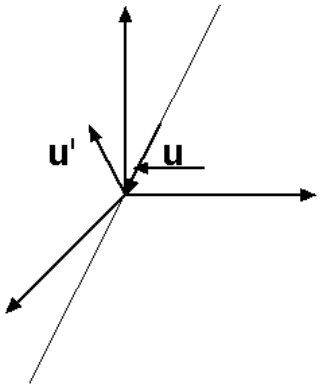
p_1 and p_2 define the rotation axis

Translate to origin



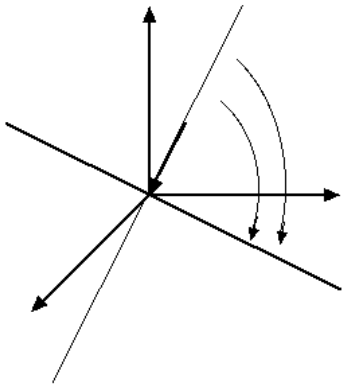
$T(-p_1)$

Finding an angle to rotate around X



Project u on the yz plane = $(0, u_y, u_z)$

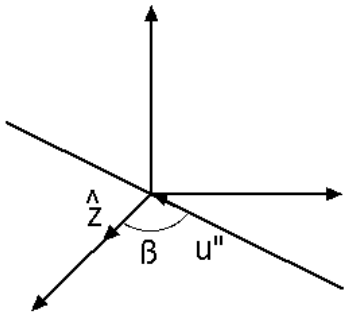
Rotate around X



$R_X(\alpha)$

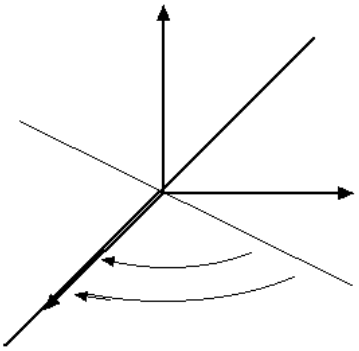
Information Coding / Computer Graphics, ISY, LiTH

Finding an angle to rotate around Y



u" and $\hat{z}$ gives the angle ß in the xz plane

Rotate around Y



$R_y(\beta)$

Rotation around arbitrary axis, summary:

The axis to rotate around is given as two points, $\mathbf{p}_1$ and $\mathbf{p}_2$.

$\mathbf{v} = \mathbf{p}_2 - \mathbf{p}_1$

$\mathbf{u} = \mathbf{v} / |\mathbf{v}| = (u_x, u_y, u_z)$ Normalized!

$d = \sqrt{u_y^2 + u_z^2}$

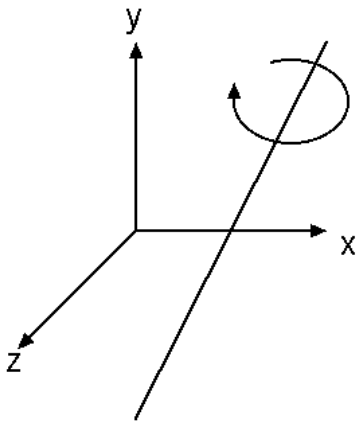
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & u_z/d & -u_y/d & 0 \\ 0 & u_y/d & u_z/d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} d & 0 & -u_x & 0 \\ 0 & 1 & 0 & 0 \\ u_x & 0 & d & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Total transformation:

$$R(\theta) = T(p_1) * R_x^T * R_y^T * R_z(\theta) * R_y * R_x * T(-p_1)$$

Rotation around arbitrary axis in OpenGL



Create matrices, multiply on CPU, upload to uniform matrices.

Rotation around arbitrary axis, using change of basis:

$\mathbf{v} = \mathbf{p}_2 - \mathbf{p}_1$

$\mathbf{u_z} = \mathbf{u} = \mathbf{v} / |\mathbf{v}| = (u_x, u_y, u_z)$ Normalized!

$\mathbf{u_y} = \mathbf{u} \times (u_x,0,0) / |\mathbf{u} \times (u_x,0,0)|$

$\mathbf{u_x} = \mathbf{u_y} \times \mathbf{u_z}$

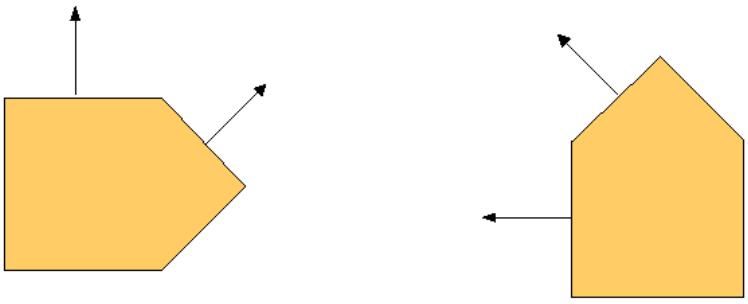
$$R = \begin{bmatrix} u_{x1} & u_{x2} & u_{x3} & 0 \\ u_{y1} & u_{y2} & u_{y3} & 0 \\ u_{z1} & u_{z2} & u_{z3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Total transformation:

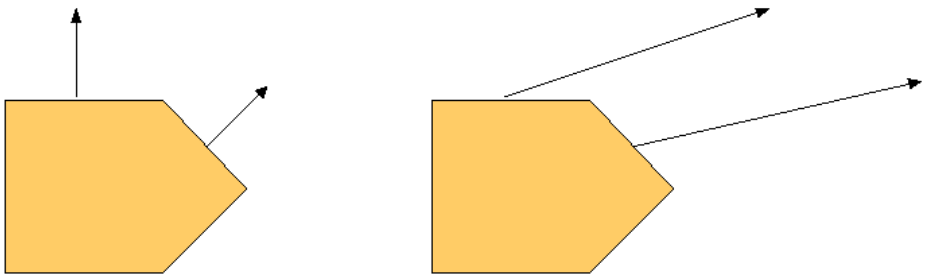
$$R(\theta) = T(p_1) * R^T * R_z(\theta) * R * T(-p_1)$$

The Normal matrix

When placing a model in the world, normals must be rotated...



but they must not be translated...



so we just cast to mat3, right?

$$\begin{bmatrix} r & r & r & t_x \\ r & r & r & t_y \\ r & r & r & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} r & r & r \\ r & r & r \\ r & r & r \end{bmatrix}$$

or we zero the translation part:

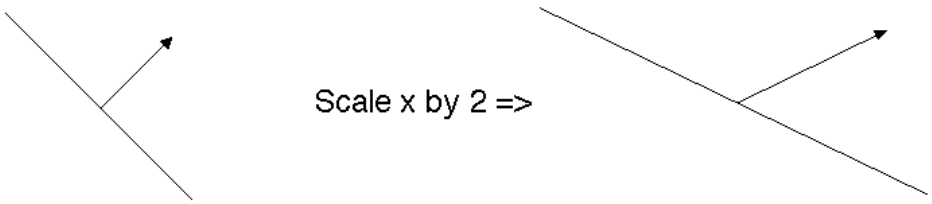
$$\begin{bmatrix} r & r & r & t_x \\ r & r & r & t_y \\ r & r & r & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} r & r & r & 0 \\ r & r & r & 0 \\ r & r & r & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

It worked in the lab... but...

Information Coding / Computer Graphics, ISY, LiTH

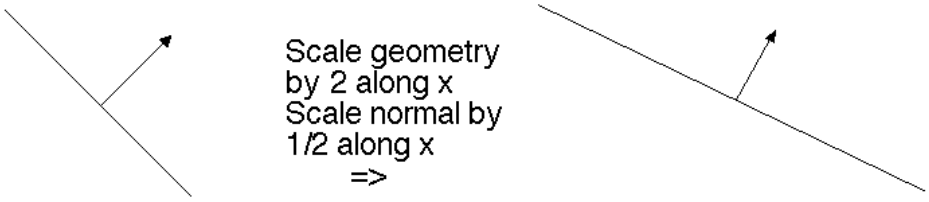
But wait!

For *non-uniform scaling*, this does not work!



The normal vector is no longer perpendicular to surface!

But what if we do the *opposite*...



Suddenly things look better...

but what happens if we mix in rotations?

Information Coding / Computer Graphics, ISY, LiTH

Normal matrix, full solution

Invert scaling, keep rotation

1) Invert to reverse both
2) Transpose to reverse rotation

=> Use *inverse transpose* of rotation part

$$N = (M^{-1})^T$$