CS451 Transforms

Jyh-Ming Lien
Department of Computer SCience
George Mason University

Based on Tomas Akenine-Möller's lecture note

Why transforms?

- We want to be able to **animate/deform** objects
  - Translations
  - Rotations
  - Shears
  - And more…
- We want to be able to use **projection** transforms

- Part of this lecture is a refresher

How to implement transforms?

- Matrices!
- Can you really do everything with a matrix?
- Not everything, but a lot!
- We use 3x3 and 4x4 matrices

$$\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \qquad \mathbf{M} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

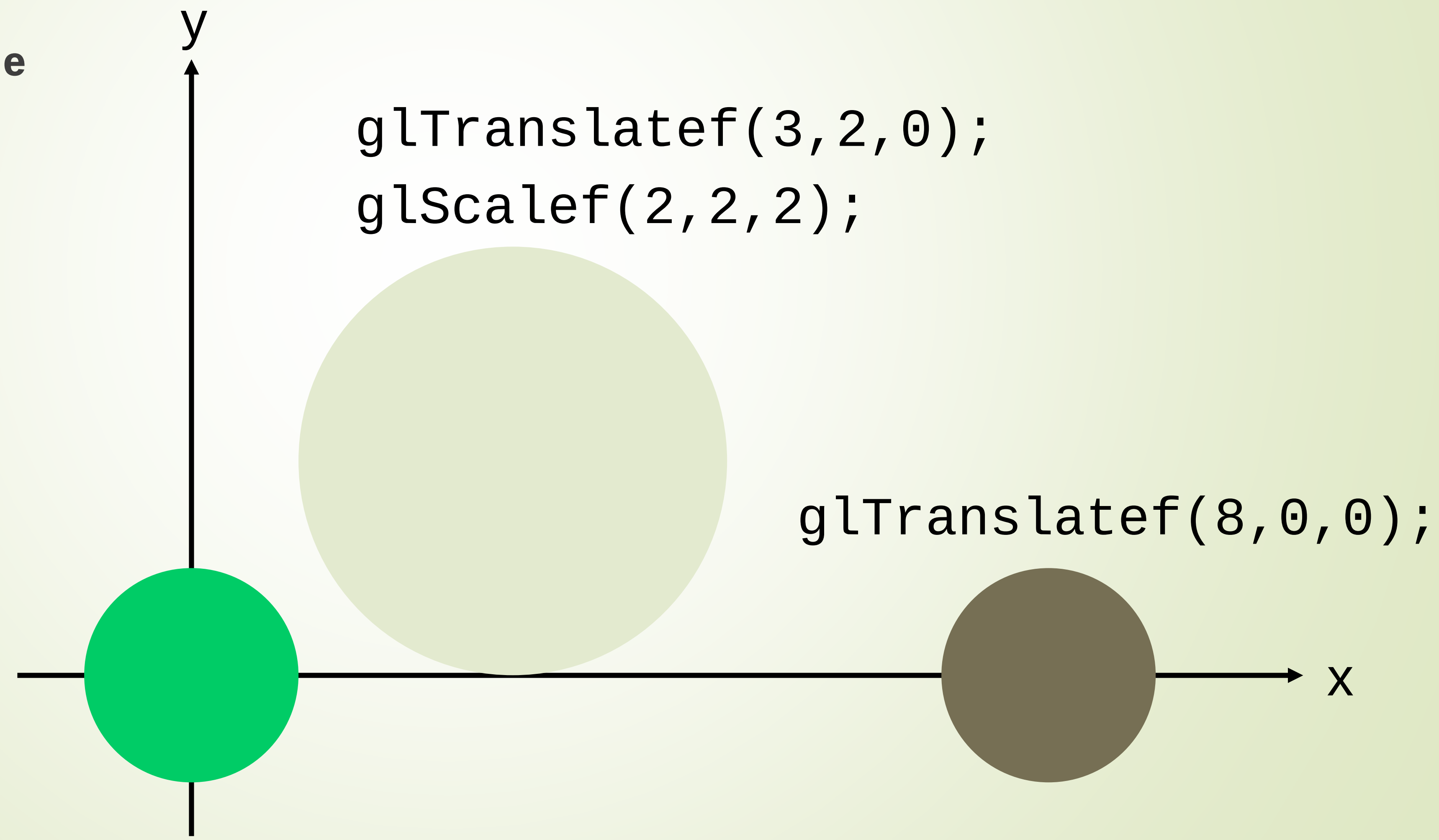
How do I use transforms practically?

- Say you have a circle with origin at (0,0,0) and with radius 1 – unit circle

- `glTranslatef(8,0,0);`
- `RenderCircle();`
- `glTranslatef(3,2,0);`
- `glScalef(2,2,2);`
- `RenderCircle();`

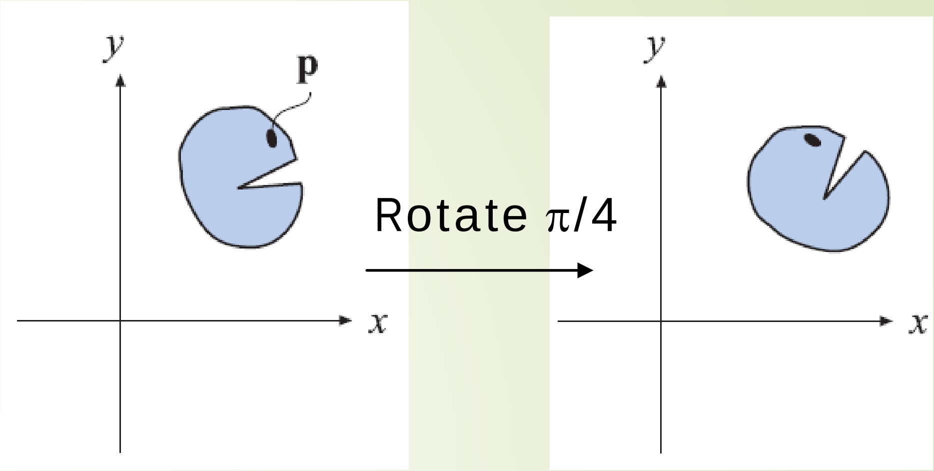
Cont'd from previous slide
A simple 2D example

- A circle in **model space**

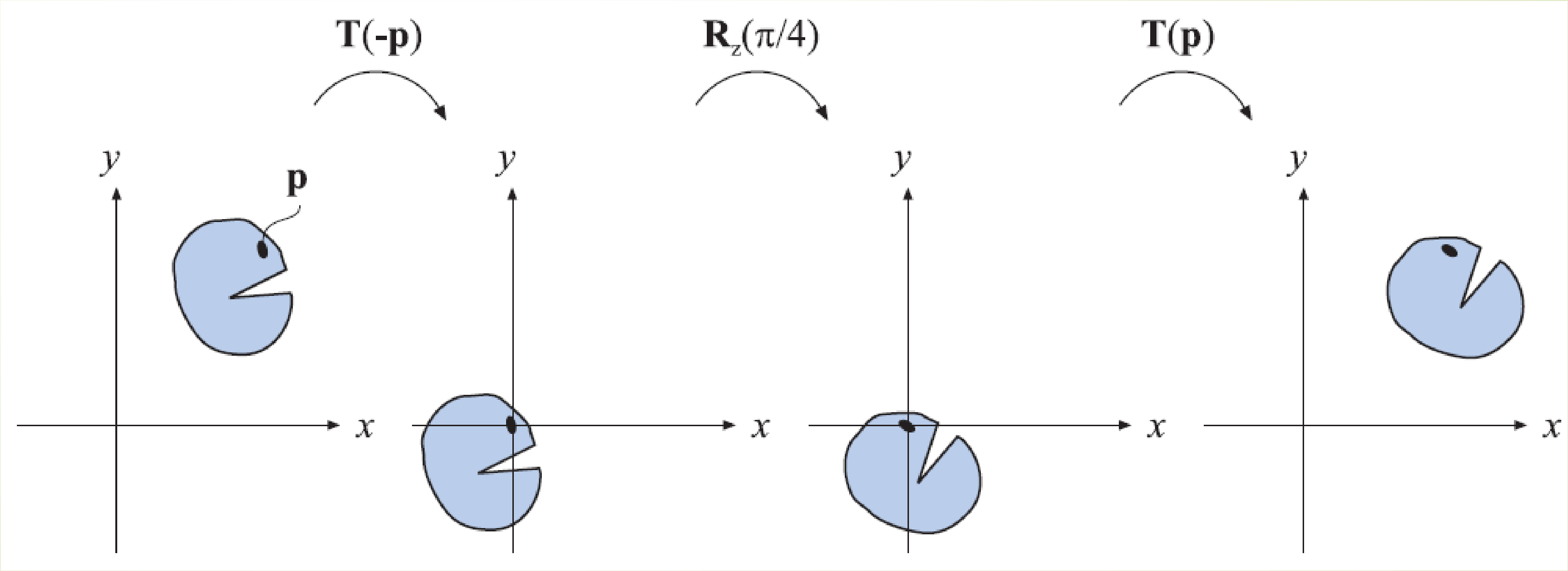


Another Example

- How do you code OpenGL to do this?



Answer:



Derivation of rotation matrix in 2D

$$\mathbf{n} = \mathbf{R}_z \mathbf{p} \quad \text{what is } \mathbf{R}_z?$$

$$\begin{pmatrix} n_x \\ n_y \end{pmatrix} = \underbrace{\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}}_{\mathbf{R}_Z} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Rotations in 3D

- Same as in 2D for Z-rotations, but with a 3x3 matrix

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \Rightarrow \mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- For X

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

- For Y

$$\mathbf{R}_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

Translations must be simple?

Translation

$$\begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix}\mathbf{p} = \mathbf{p} + \mathbf{t}$$

Rotation

$$\mathbf{Rp} = \mathbf{n}$$

- Rotation is matrix multiplication, translation is addition
- Would be nice if we could only use matrix multiplications...
- Turn to **homogeneous coordinates**
- Add a new component to each vector

Homogeneous notation

- A point: $\mathbf{p} = \begin{pmatrix} p_x & p_y & p_z & 1 \end{pmatrix}^T$
- Translation becomes:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}(\mathbf{t})} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}$$

- A vector (direction):
- Translation of vector: $\mathbf{d} = \begin{pmatrix} d_x & d_y & d_z & 0 \end{pmatrix}^T$
- Also allows for projections (later)

$$\mathbf{Td} = \mathbf{d}$$

Rotations in 4x4 form

- Just add a row at the bottom, and a column at the right:

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Similarly for X and Y
- det(R)=1 (for 3x3 matrices)
- Trace(R)=1+2cos(alpha) (for any axis,3x3)

Scaling

- Uniform scaling

$$S(s) = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S'(s) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/s \end{pmatrix}$$

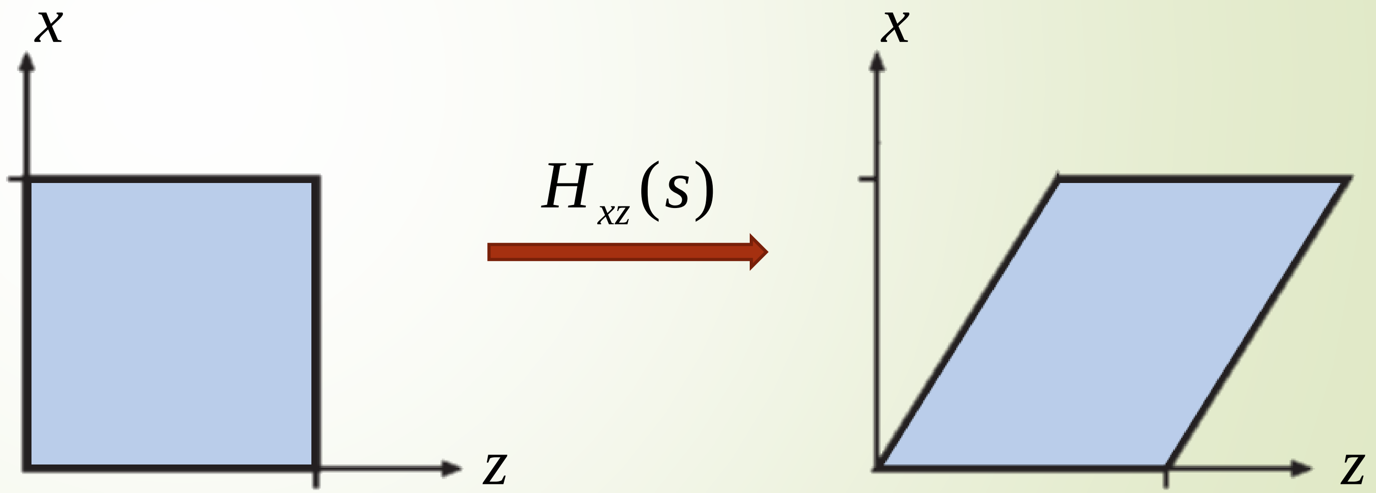
- Non-uniform scaling

$$S(s,t,u) = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & t & 0 & 0 \\ 0 & 0 & u & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Shearing

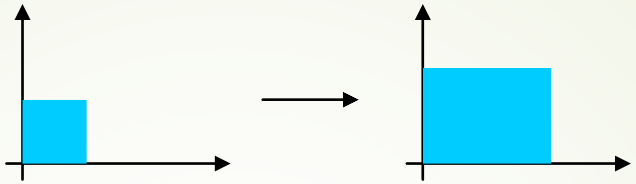
- Shearing in XZ plane
  - Using the Z coordinate to change the X coordinate

$$H_{xz}(s) = \begin{pmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

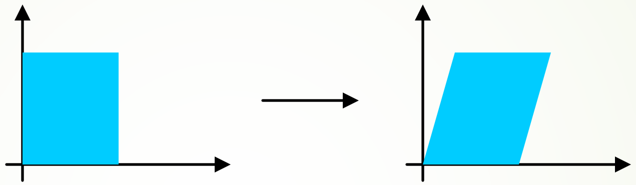


Review basic transforms

- Scaling



- Shear



Q1: How to scale along an arbitrary direction?

Q2: How do you scale a translated, rotated shape?

- Rigid-body: rotation then translation

$$\mathbf{X} = \mathbf{TR}$$

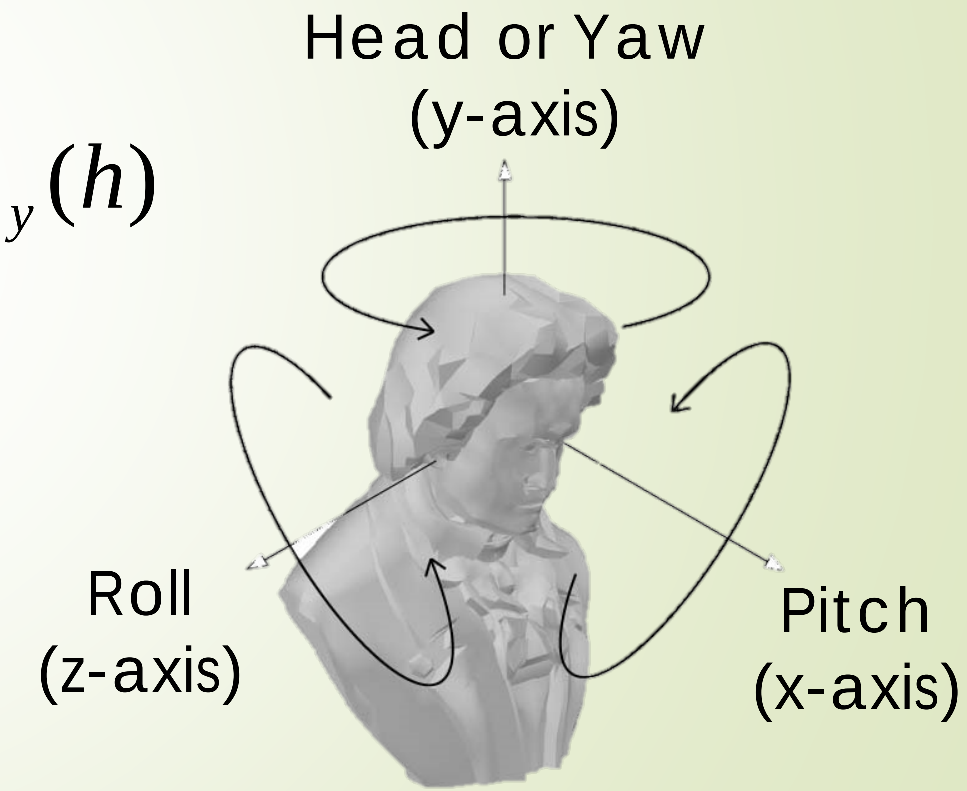
- Concatenation of matrices
  - Not commutative, i.e., $\mathbf{RT} \neq \mathbf{TR}$
  - In $\mathbf{X} = \mathbf{TR}$, the rotation is done first
- Inverses and rotation about arbitrary axis

The Euler Transform

- Assume the view looks down the negative z-axis, with up in the y-direction, x to the right

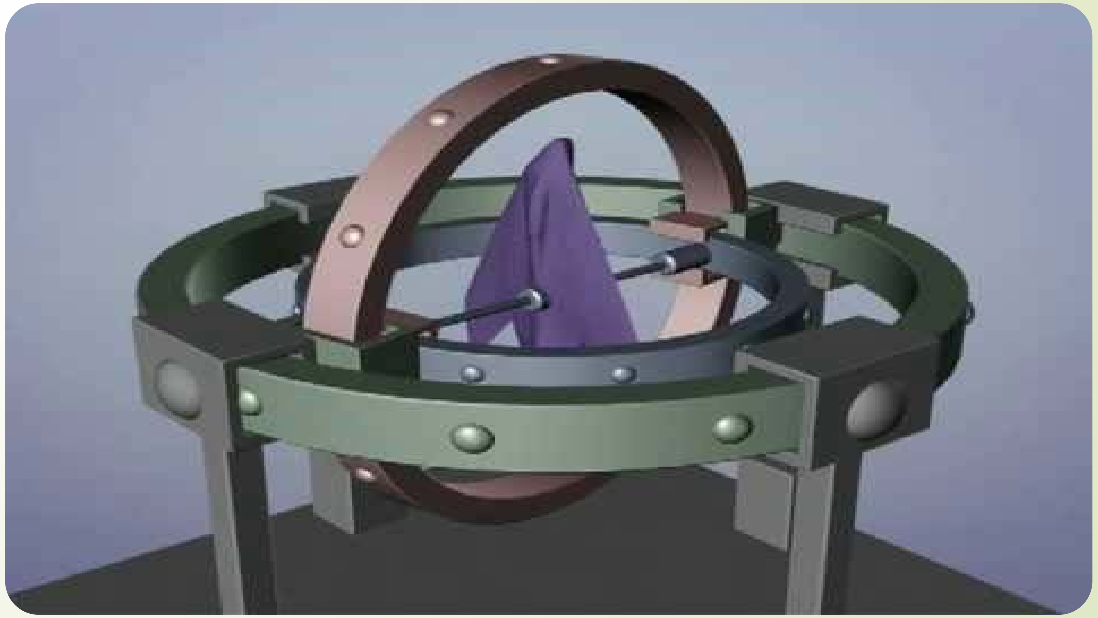
$$\mathbf{E}(h, p, r) = \mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h)$$

- h=head (yaw)
- p=pitch
- r=roll



Gimbal Lock

- Euler Transform is a hierarchical system
  - XYZ or ZYX or ... Indicates the order of rotation
- Gimbal lock can occur
  - The top and the bottom of the hierachy overlaps
  - looses one degree of freedom
- Can also be explained using Matrix



By The Guerrilla CG Project

Quaternions

$$\mathbf{q} = (q_w, \mathbf{q}_v) = (q_w, q_x, q_y, q_z)$$

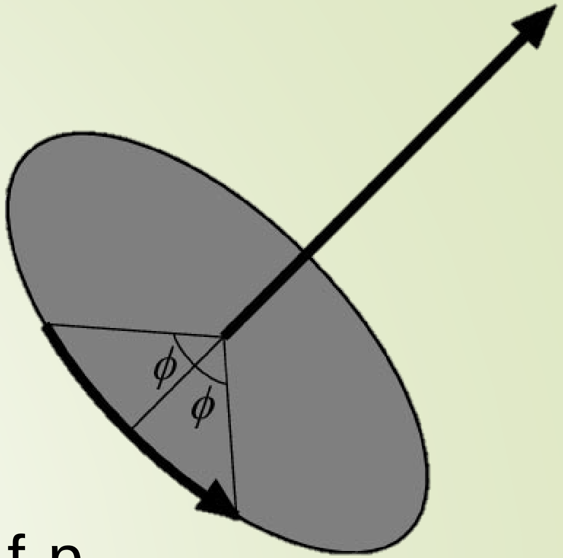
- Extension of imaginary numbers
- Avoids *gimbal lock* that the Euler could produce
- Focus on unit quaternion:

$$n(\mathbf{q}) = q_x^2 + q_y^2 + q_z^2 + q_w^2 = 1$$

- A unit quaternion is:

$$\mathbf{q} = (\cos\phi, \sin\phi\mathbf{u}_q) \quad \text{where } \|\mathbf{u}_q\| = 1$$

Unit quaternions are perfect for rotations! $\mathbf{q} = (\cos\phi, \sin\phi \bullet \mathbf{u}_q)$



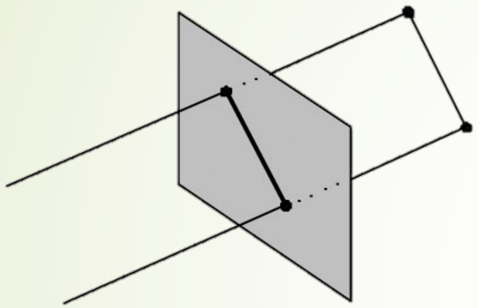
- Compact (4 components)
- Can show that $\hat{\mathbf{q}}\hat{\mathbf{p}}\hat{\mathbf{q}}^{-1}$
- ...represents a rotation of 2ϕ radians around u_q of p

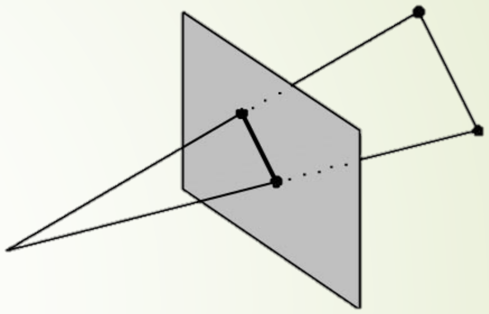
- That is: a unit quaternion represent a rotation as a rotation axis and an angle
  - In OpenGL: glRotatef(ux,uy,uz,angle);

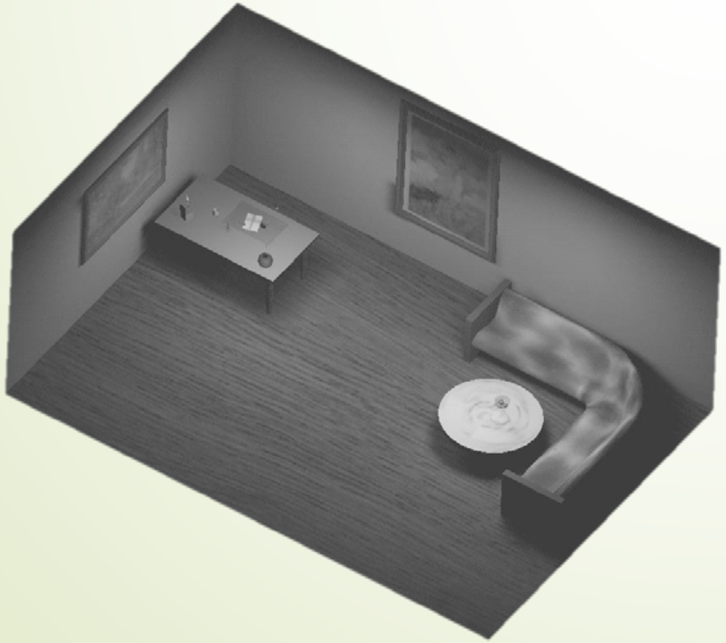
- Read the quaternion code from PA1 for more details
  - Mathtool/quaternion.h

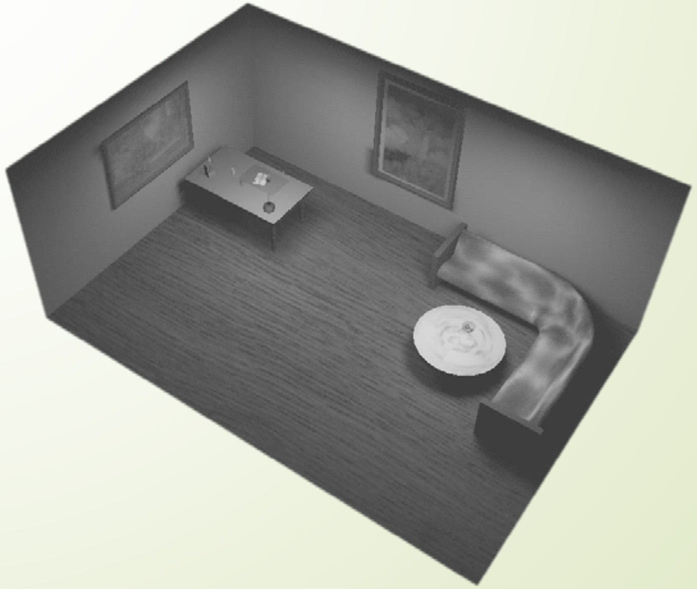
Projections

- Orthogonal (parallel) and Perspective

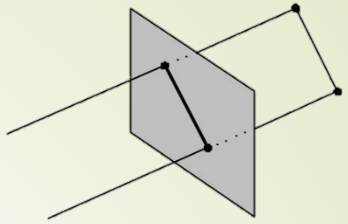






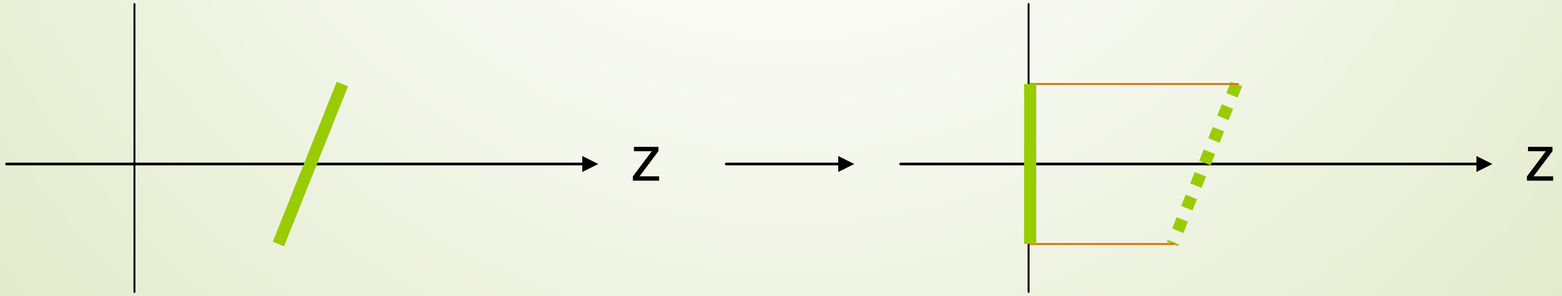


Orthogonal projection

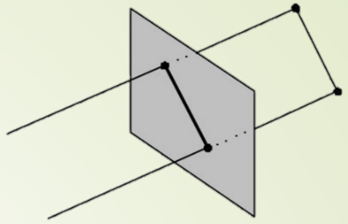


- Simple, just skip one coordinate
  - Say, we're looking along the z-axis
  - Then drop z, and render

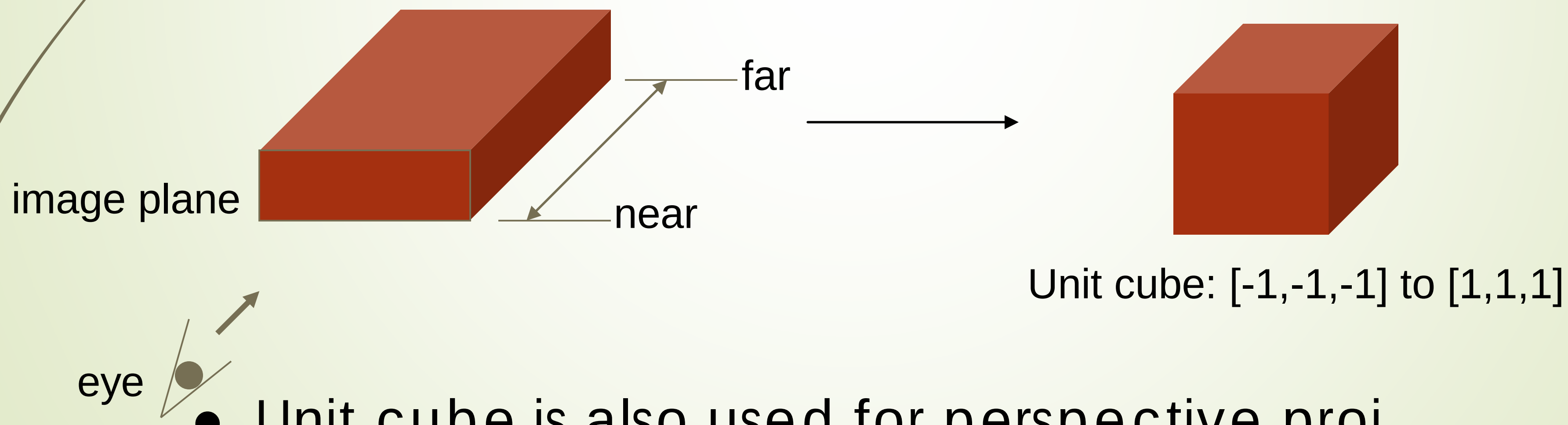
$$\mathbf{M}_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow \mathbf{M}_{ortho} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ 0 \\ 1 \end{pmatrix}$$



Orthogonal projection



- Not invertible! (determinant is zero)
- For Z-buffering
  - It is not sufficient to project to a plane
  - Rather, we need to ”project” to a box



- Unit cube is also used for perspective proj.
- Simplifies clipping

What about those homogenenous coordinates?

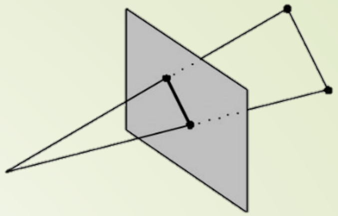
- pw=0 for vectors, and pw=1 for points
- What if pw is **not** 1 or 0?
- Solution is to divide all components by pw

$$\mathbf{p} = \begin{pmatrix} p_x & p_y & p_z & p_w \end{pmatrix}^T$$

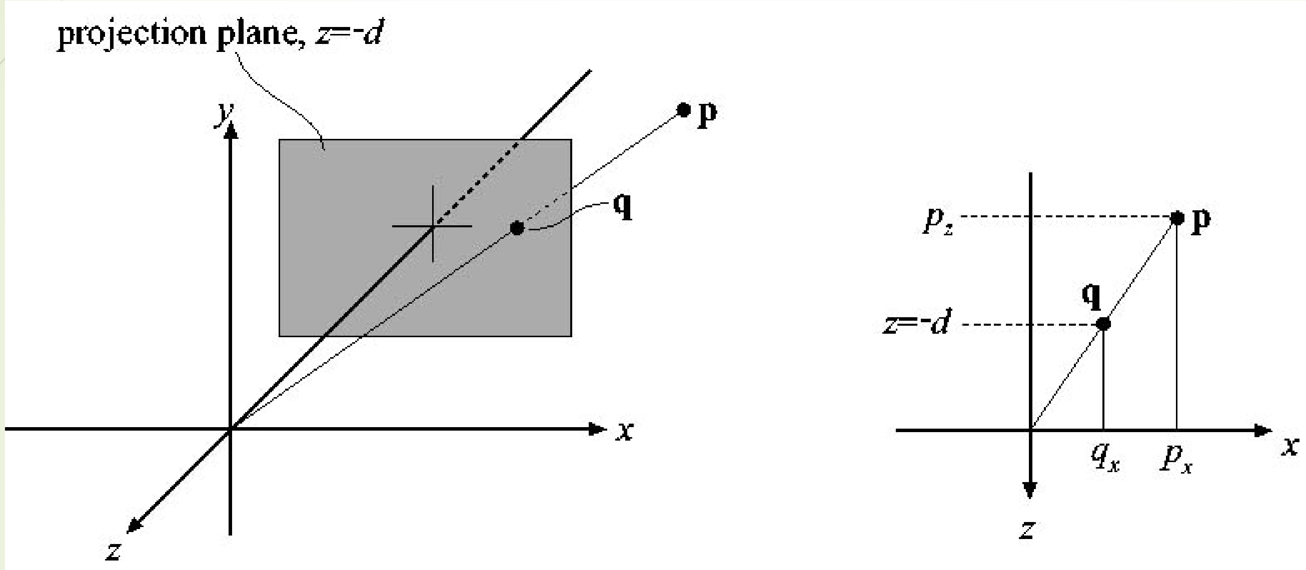
$$\mathbf{p} = \begin{pmatrix} p_x / p_w & p_y / p_w & p_z / p_w & 1 \end{pmatrix}^T$$

- Gives a point again!
- Can be used for projections, as we will see

Perspective projection



$d>0$

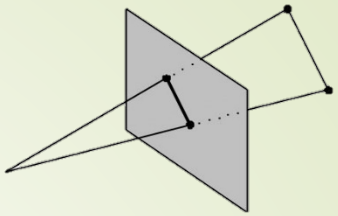


$$\frac{q_x}{p_x} = \frac{-d}{p_z} \quad \Rightarrow \quad q_x = -d\,\frac{p_x}{p_z}$$

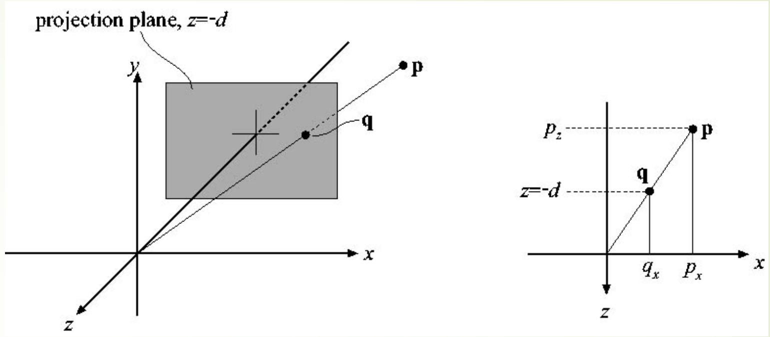
For y: $q_y = -d\,\frac{p_y}{p_z}$

$$\mathbf{P}_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}$$

Perspective projection



$$\mathbf{P}_p = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix} \qquad \mathbf{P}_p\mathbf{p} = ?$$



$$\mathbf{P}_p\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \mathbf{q} = \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -dp_z/p_z \\ 1 \end{pmatrix} = \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z \\ -d \\ 1 \end{pmatrix}$$

$$q_x = -d\frac{p_x}{p_z} \qquad q_y = -d\frac{p_y}{p_z}$$

- The ”arrow” is the homogenization process