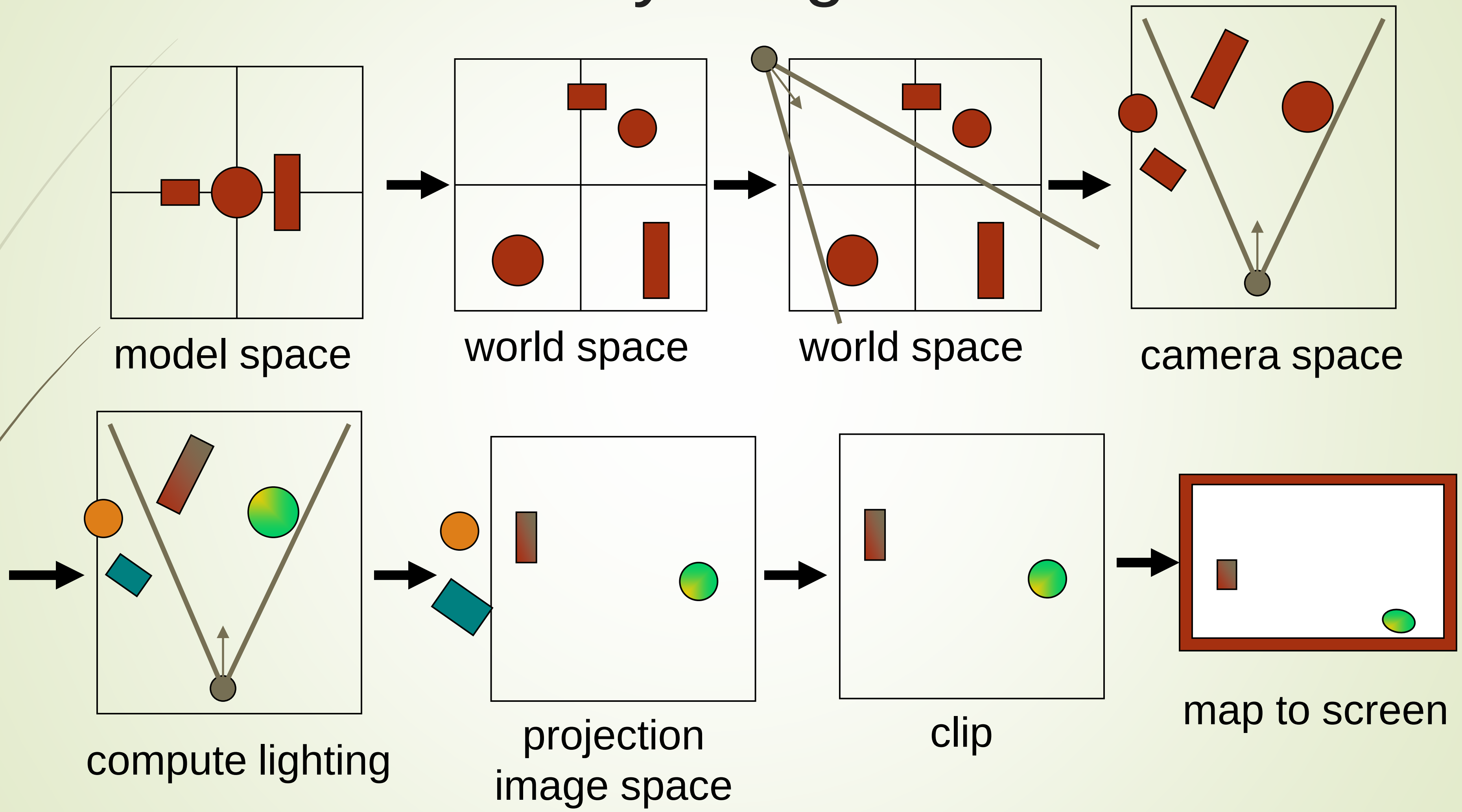
CS451 More Transforms

Quaternion/Dual Quaternion and Projections

Jyh-Ming Lien
Department of Computer SCience
George Mason University

Based on Tomas Akenine-Möller's lecture note

The Geometry stage

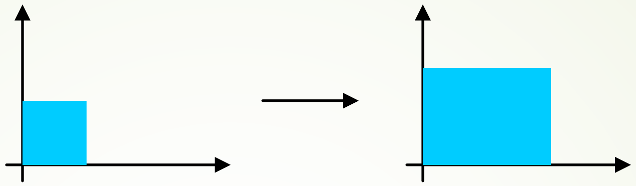


The Rasterization stage

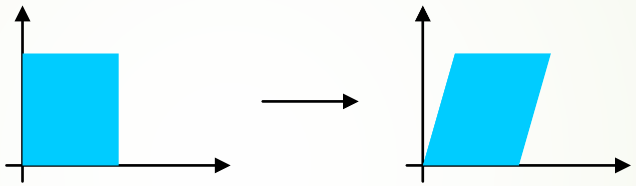
- Scan-conversion
  - Find out which pixels are inside the primitive
- Texturing
  - Put images on triangles
- Interpolation over triangle
- Z-buffering
  - Make sure that what is visible from the camera really is displayed
- Double buffering
- …

Review basic transforms

- Scaling



- Shear



Q1: How to scale long an arbitrary direction?

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

- Rigid-body: rotation then translation

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

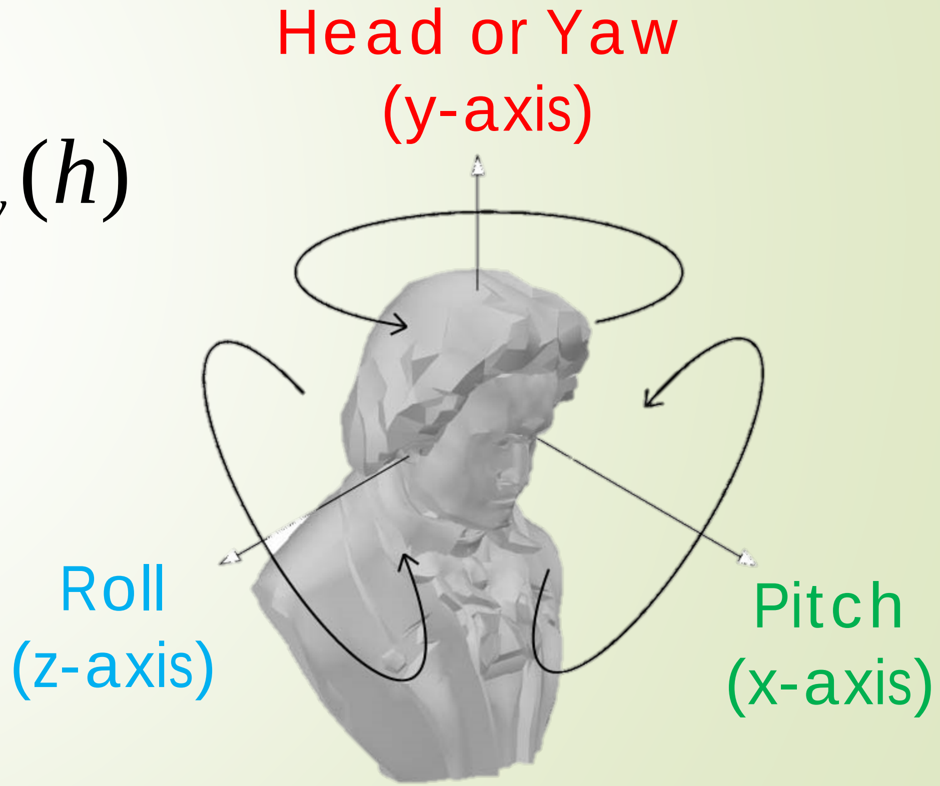
- Concatenation of matrices
  - Not commutative, i.e., $\mathbf{RT} \neq \mathbf{TR}$
  - In $\mathbf{X} = \mathbf{TR}$, the rotation is done first

The Euler Transform

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

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

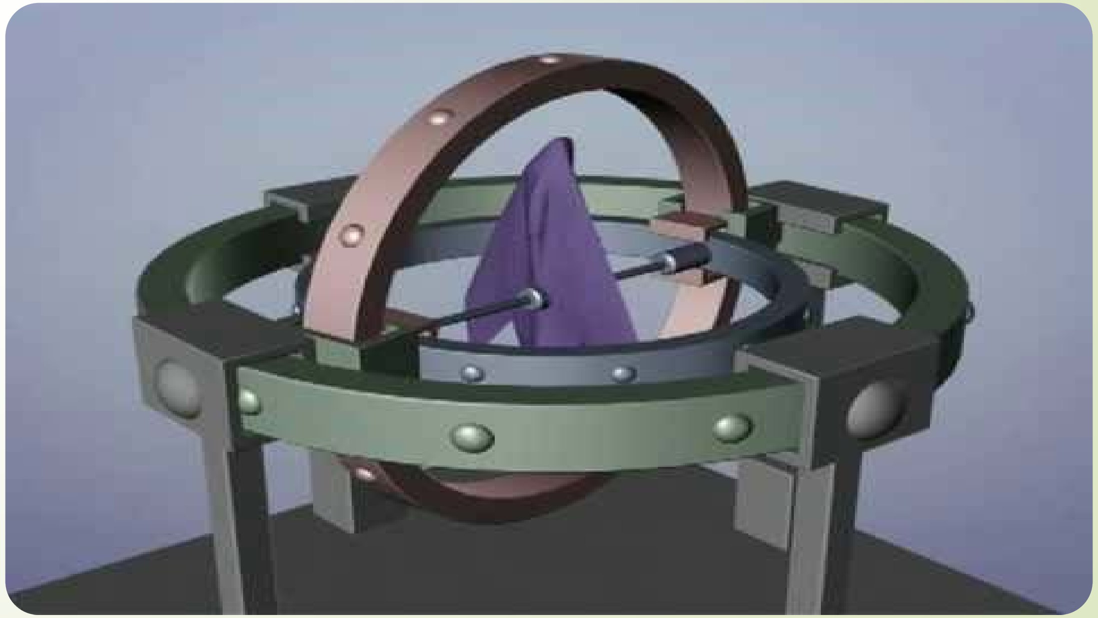
- h=head (yaw)
- p=pitch
- r=roll



Gimbal Lock

- Euler Transform is a hierarchical system
  - XYZ or ZYX or ... Indicates the order of rotation
  - there are 6 different conbinations (or 12 if something like XYX is allowed)
- 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

$$\mathbf{q} = (q_w, \mathbf{q}_v) = (w, x\mathbf{i}, y\mathbf{j}, z\mathbf{k})$$

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

$$n(\mathbf{q}) = w^2 + x^2 + y^2 + z^2 = 1$$

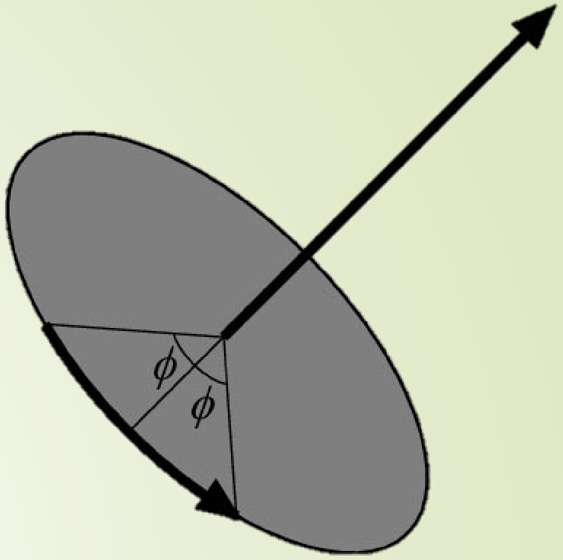
- A example of unit quaternion is:

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

Quaternions Basic Operations

- Given a quaternion $\mathbf{q}=(w,\mathbf{v})$
- Scalar multiplication $s\mathbf{q} = (sw, s\mathbf{v})$
- Addition $\mathbf{q_1}+ \mathbf{q_2} = (w_1+ w_2 , \mathbf{v_1}+ \mathbf{v_2})$
- Multiplication $\mathbf{q_1 q_2} = ((w_1 w_2 - \mathbf{v_1 v_2}), w_1\mathbf{v_2}+w_2\mathbf{v_1}+(\mathbf{v_1}\times\mathbf{v_2}))$
- Conjugate $\mathbf{q}^* = (w, -\mathbf{v})$
- Norm $||\mathbf{q}||= \mathbf{qq}^*$
- Normalization $\mathbf{q_n} = \mathbf{q}/||\mathbf{q}||$

Rotation Quaternions



- Represents a rotation of ϕ radians around **v**

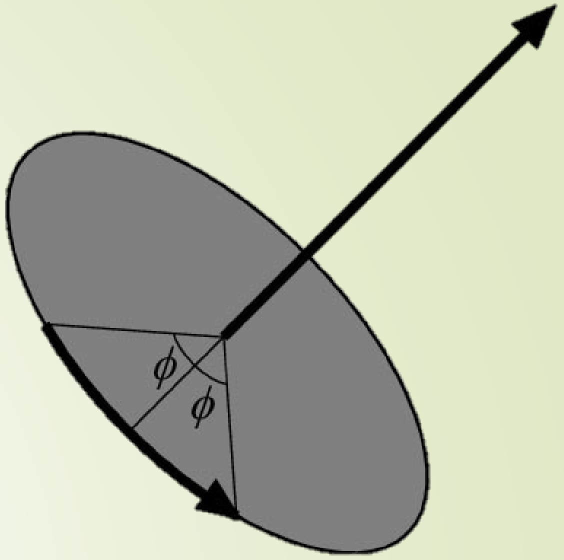
$$\mathbf{q} = (\cos\frac{\phi}{2}, \sin\frac{\phi}{2} \bullet \mathbf{v}\ )$$

- Compact (4 components)
- Can show that $(0, p') = q(0, p)q^*$

- Rotation can be applied in the way similar to matrix
  - $(0, p') = \ q_3 q_2 q_1 (0, p) q_1^* q_2^* q_3^*$

- Try to rotate vector (1,1,1) around Y-axis for 180 degree

Rotation Quaternions



- Represents a rotation of ϕ radians around **v**

$$\mathbf{q} = (\cos\tfrac{\phi}{2}, \sin\tfrac{\phi}{2} \bullet \mathbf{v} \ )$$

- 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 PA03 for more details
  - mathtool/quaternion.h

Rotation Matrix vs. Rotation Quaternion

- Why should (or should not) you use matrix?

- Why should (or should not) you use quaternion?

Dual Quaternion

- Dual number $z = r + d\varepsilon$ where $\varepsilon \neq 0$ but $\varepsilon^2 = 0$
  - Add $z_A + z_B = (r_A + d_A\varepsilon) + (r_B + d_B\varepsilon) = (r_A + r_B) + (d_A + d_B)\varepsilon$
  - Multiply $z_A z_B = (r_A + d_A\varepsilon)(r_B + d_B\varepsilon) = (r_A r_B) + (r_B d_A + r_A d_B)\varepsilon + (d_A d_B)\varepsilon^2$
- Dual quaternion $\mathbf{Q} = \mathbf{q_r} + \mathbf{q_d}\varepsilon$
  - Represent both rotation $\mathbf{q_r}$ and translation $\mathbf{q_d}$
  - Scalar Multiplication $s\mathbf{Q} = s\mathbf{q_r} + s\mathbf{q_d}\varepsilon$
  - Addition $\mathbf{Q_1} + \mathbf{Q_2} = (\mathbf{q_{r1}} + \mathbf{q_{r2}}) + (\mathbf{q_{d1}} + \mathbf{q_{d2}})\varepsilon$
  - Multiplication $\mathbf{Q_1 Q_2} = (\mathbf{q_{r1}}\, \mathbf{q_{r2}}) + (\mathbf{q_{r1}}\, \mathbf{q_{d2}} + \mathbf{q_{d1}} \mathbf{q_{r2}})\varepsilon$
  - Conjugate $\mathbf{Q}^* = \mathbf{q_r}^* + \mathbf{q_d}^*\varepsilon$
  - Norm $||\mathbf{Q}|| = \mathbf{QQ}^*$

Dual Quaternion (Cont.)

Question: Can you show that $\mathbf{Q}_r$, $\mathbf{Q}_d$ and $\mathbf{Q}$ are unit dual quaternion?

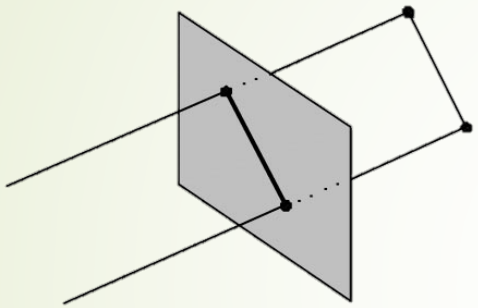
- Transformation dual quaternion
  - Given a rotation quaternion $\mathbf{q}_r = (\cos(\theta/2), \sin(\theta/2)\mathbf{v})$
  - Given a translation vector $\mathbf{t} = (t_x, t_y, t_z)$
  - Rotation only dual quaternion $\mathbf{Q}_r = \mathbf{q}_r + (0,0,0,0)\varepsilon$
  - Translation only dual quaternion $\mathbf{Q}_d = (1,0,0,0) + (0, t_x/2, t_y/2, t_z/2)\varepsilon$
  - Combine both Rotation and Translation $\mathbf{Q} = \mathbf{Q}_d\,\mathbf{Q}_r$
- Using dual quaternion to perform rigid transform
  - Similar to using quaternion, we use dual quaternion as
    - $p' = \mathbf{Q}\, p\, \mathbf{Q}^*$
    - Here p is first rotated and then translated to produce p'

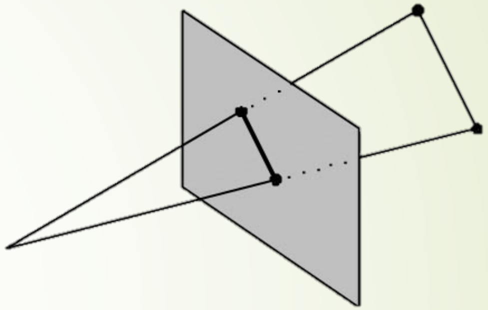
Create Dual Quaternion

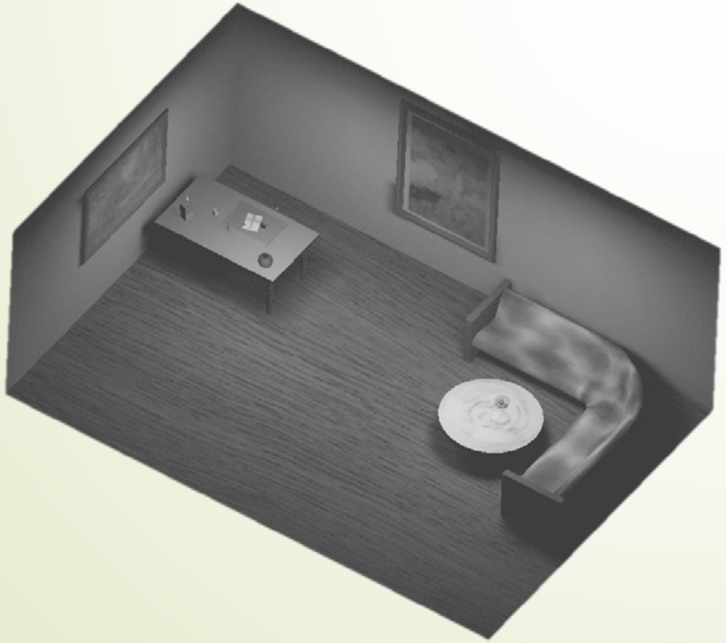
- Given a rotation angle θ and a rotation vector $\mathbf{v}$ and a translation vector $\mathbf{t}$
- We first construct a rotation quaternion
  - $\mathbf{q_r} = (\cos(\theta/2), \sin(\theta/2)\mathbf{v})$
- Then we construct the second quaternion to represent translation
  - $\mathbf{q_d} = (0, \mathbf{t}/2)\ \mathbf{q_r}$
  - Why?
- Finally, $\mathbf{Q} = (\mathbf{q_r}, \mathbf{q_d}\,\varepsilon)$
- Question: How do you convert dual quaternion $\mathbf{Q}$ to get the rotational quaternion and translational vector $\mathbf{t}$

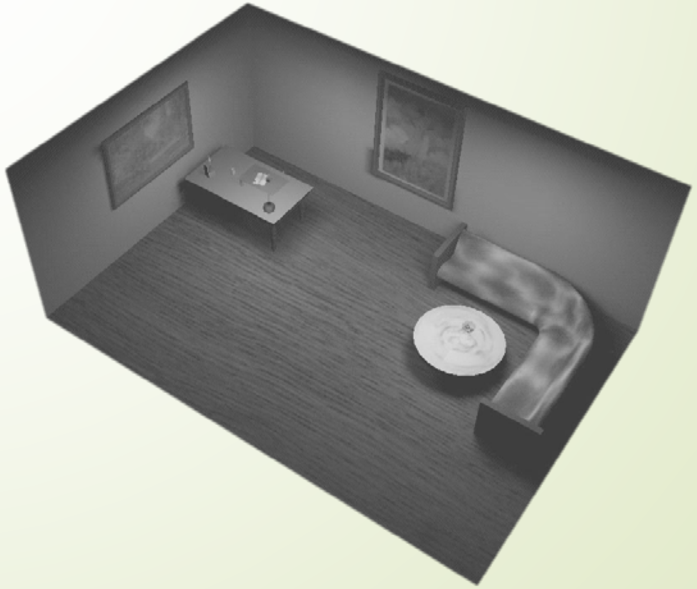
Projections

- Orthogonal (parallel) and Perspective

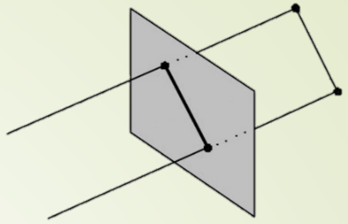






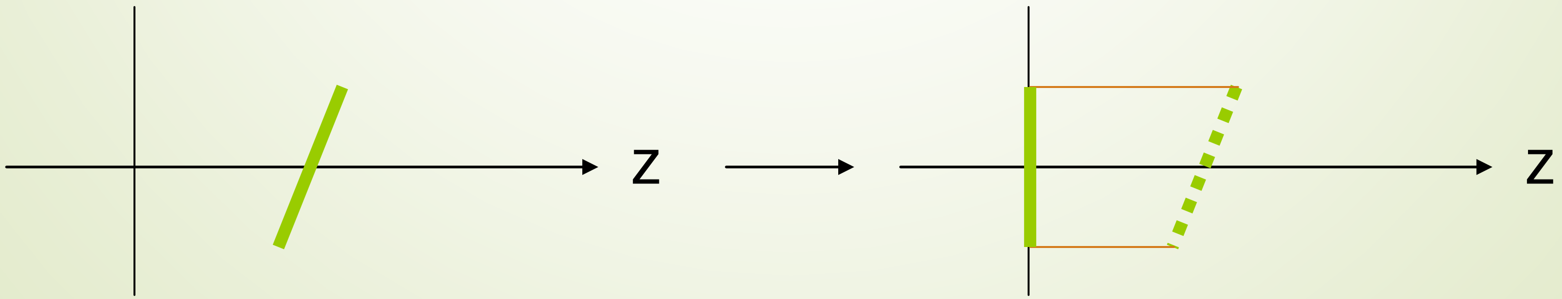


Orthogonal projection

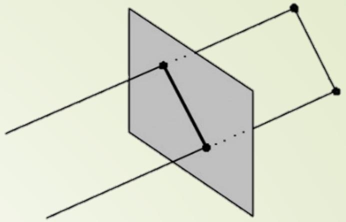


- Simple, just skip one coordinate
  - Say, we're looking along the z-axis
  - Then drop z, and render

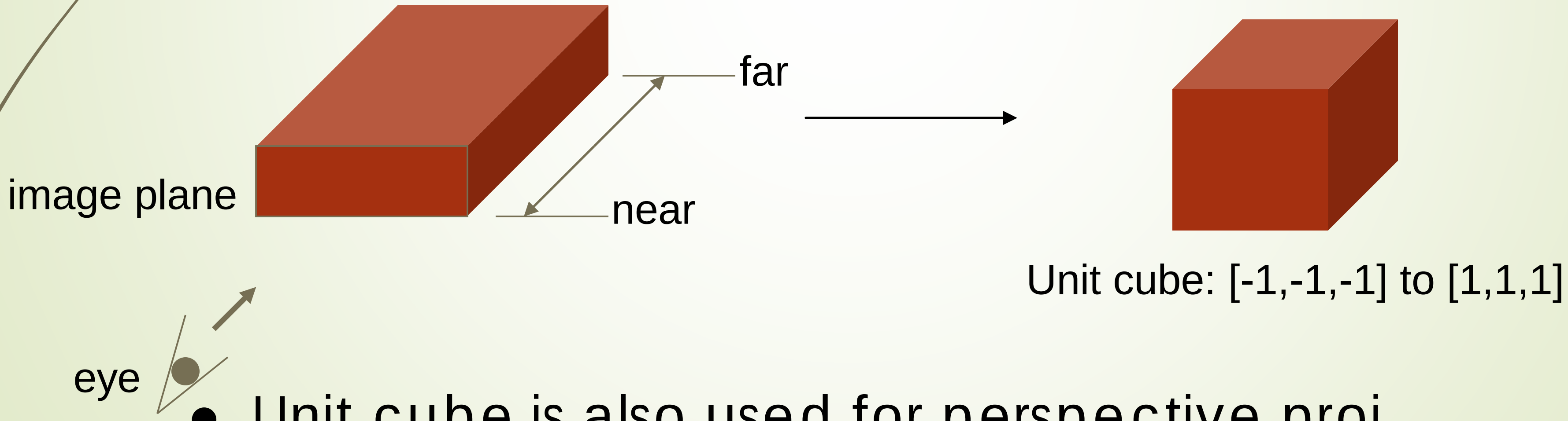
$$\mathbf{M}_{ortho} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad \mathbf{M}_{ortho} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ 0 \\ 1 \end{pmatrix}$$



Orthogonal projection

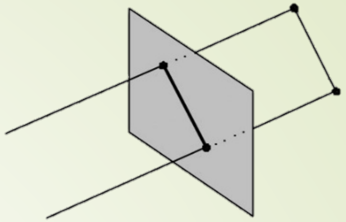


- Not invertible! (determinant is zero)
- For Z-buffering
  - It is not sufficient to project to a plane
  - Rather, we need to "project" to a box

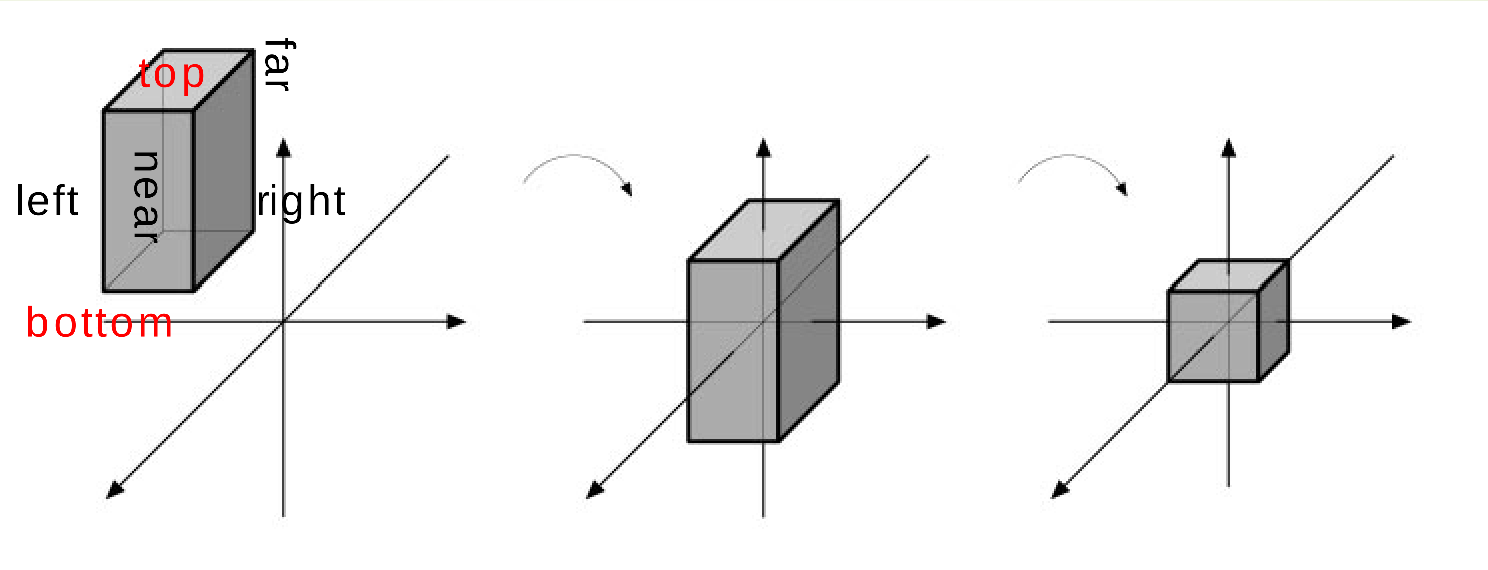


- Unit cube is also used for perspective proj.
- Simplifies clipping

Orthogonal projection



- The ”unitcube projection” is invertible
- Simple to derive
  - a translation followed by a scale



What about those homogenenous coordinates?

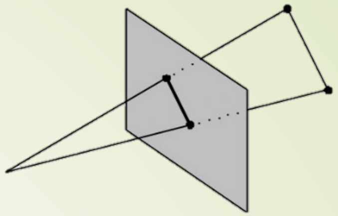
- pw=0 for vectors, and pw=1 for points
- What if pw is **not** 1 or 0?
- Solution is to divide all components by pw

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

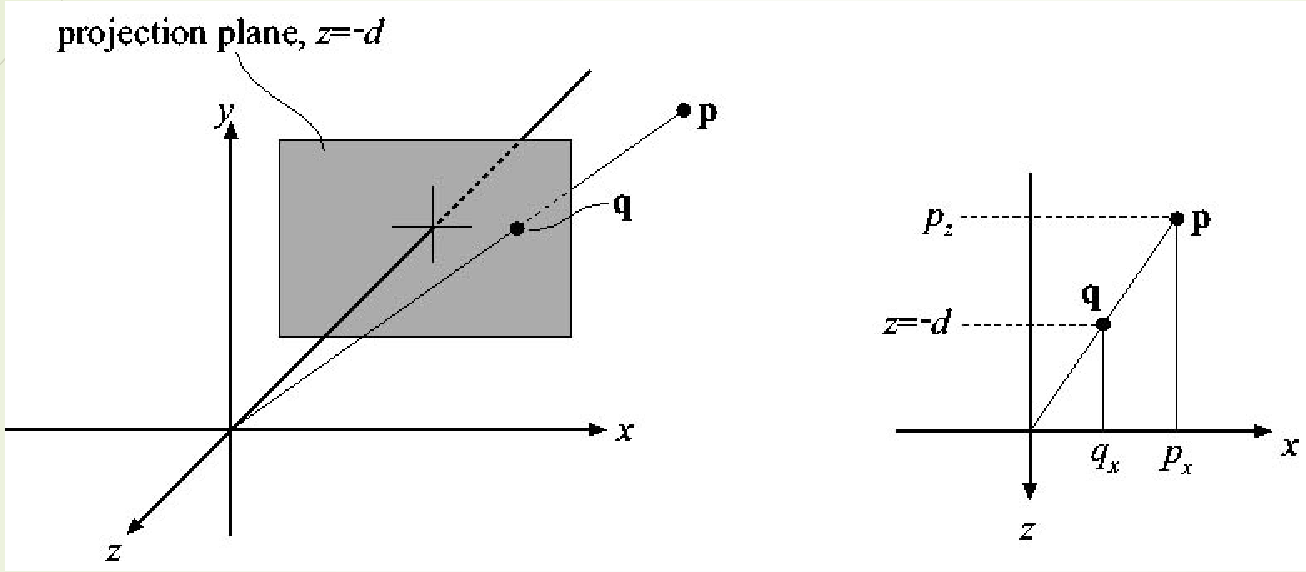
$$\mathbf{p} = \begin{pmatrix} p_x / p_w & p_y / p_w & p_z / p_w & 1 \end{pmatrix}^T$$

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

Perspective projection



$d>0$

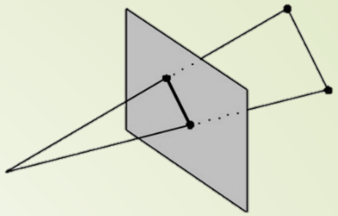


$$\frac{q_x}{p_x} = \frac{-d}{p_z} \quad \Rightarrow \quad q_x = -d\,\frac{p_x}{p_z}$$

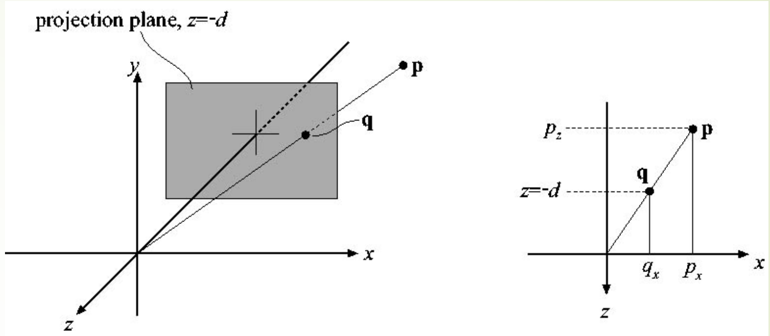
$$\text{For y:} \quad q_y = -d\,\frac{p_y}{p_z}$$

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

Perspective projection



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

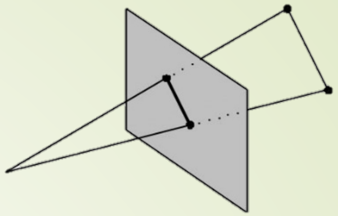


$$\mathbf{P}_p\mathbf{p} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{pmatrix}\begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x \\ p_y \\ p_z \\ -p_z/d \end{pmatrix} \Rightarrow \mathbf{q} = \begin{pmatrix} -dp_x/p_z \\ -dp_y/p_z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

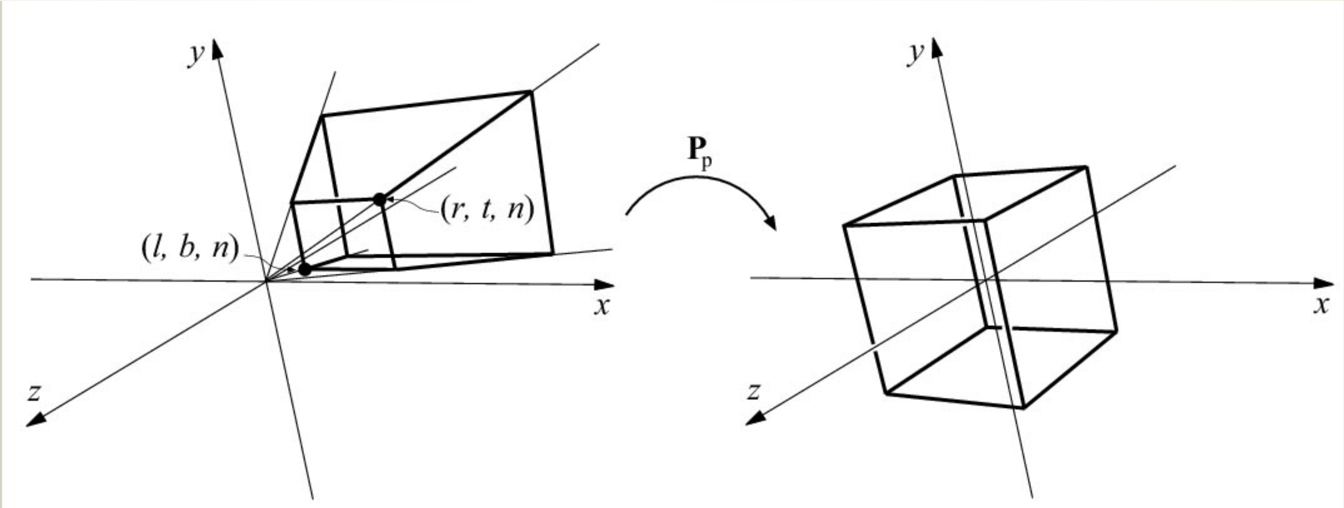
$$q_x = -d\frac{p_x}{p_z} \qquad q_y = -d\frac{p_y}{p_z}$$

- The ”arrow” is the homogenization process

Perspective projection



- Again, the determinant is 0 (not invertible)
- To make the rest of the pipeline the same as for orhogonal projection:
    - project into unit-cube



- Not much different from P_p
- Do not collapse z-coord to a plane