2: Homogeneous Transformations

Dr. Hamish Carr & Dr. Rafael Kuffner dos Anjos

Agenda

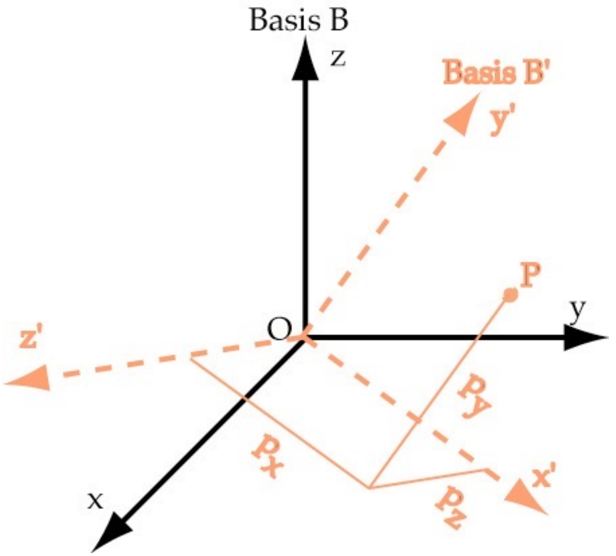
- Matrix transformations
- Homogeneous coordinates
- Projection
- Projections in OpenGL

Changing Bases

- Assume P is in B'

$$p = p_x \vec{e}_{x'} + p_y \vec{e}_{y'} + p_z \vec{e}_{z'}$$

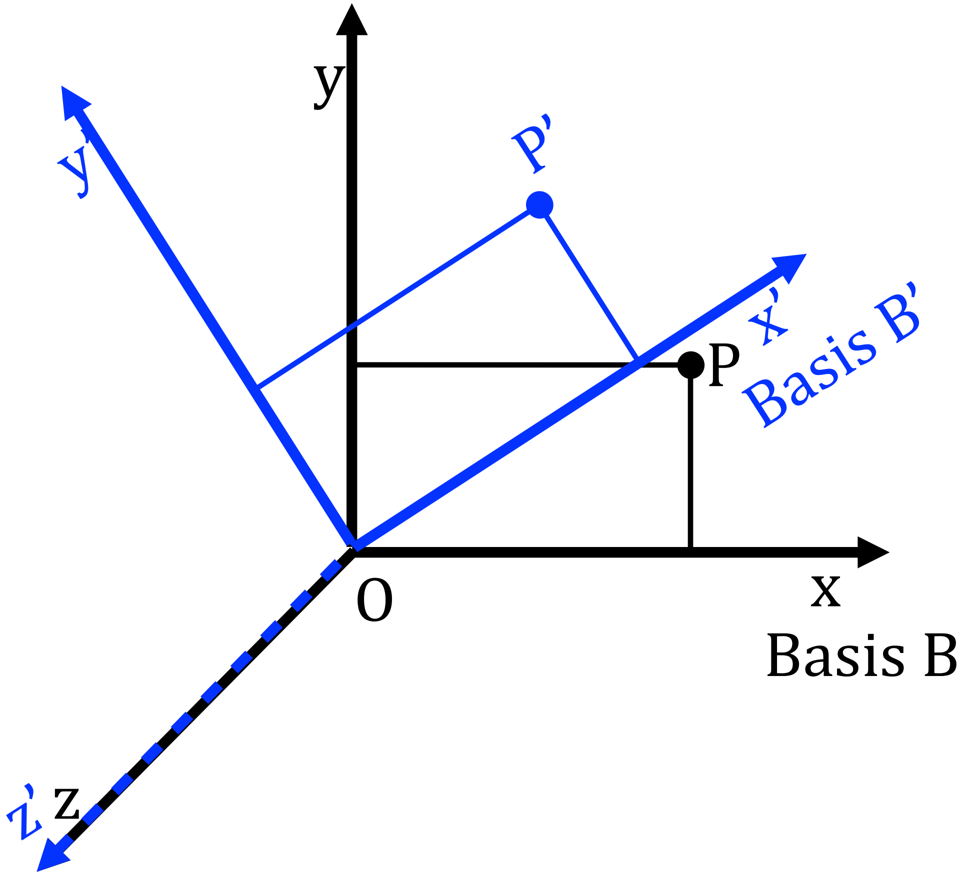
- Where is P in B?
- Depends on $\vec{e}_{x'}, \vec{e}_{y'}, \vec{e}_{z'}$
  - the basis of B'



$$\begin{aligned}
p' &= p_x \vec{e}_{x'} + p_y \vec{e}_{y'} + p_z \vec{e}_{z'} \\
&= p_x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + p_y \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} + p_z \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \\
&= Rp
\end{aligned}$$

Rotation Matrices

- Given a point P in basis B, rotate θ CCW around z
- Which we do by finding a rotated basis B
- The coordinates of P in B, are the coordinates of P' in B'



UNIVERSITY OF LEEDS

Constructing Rotation

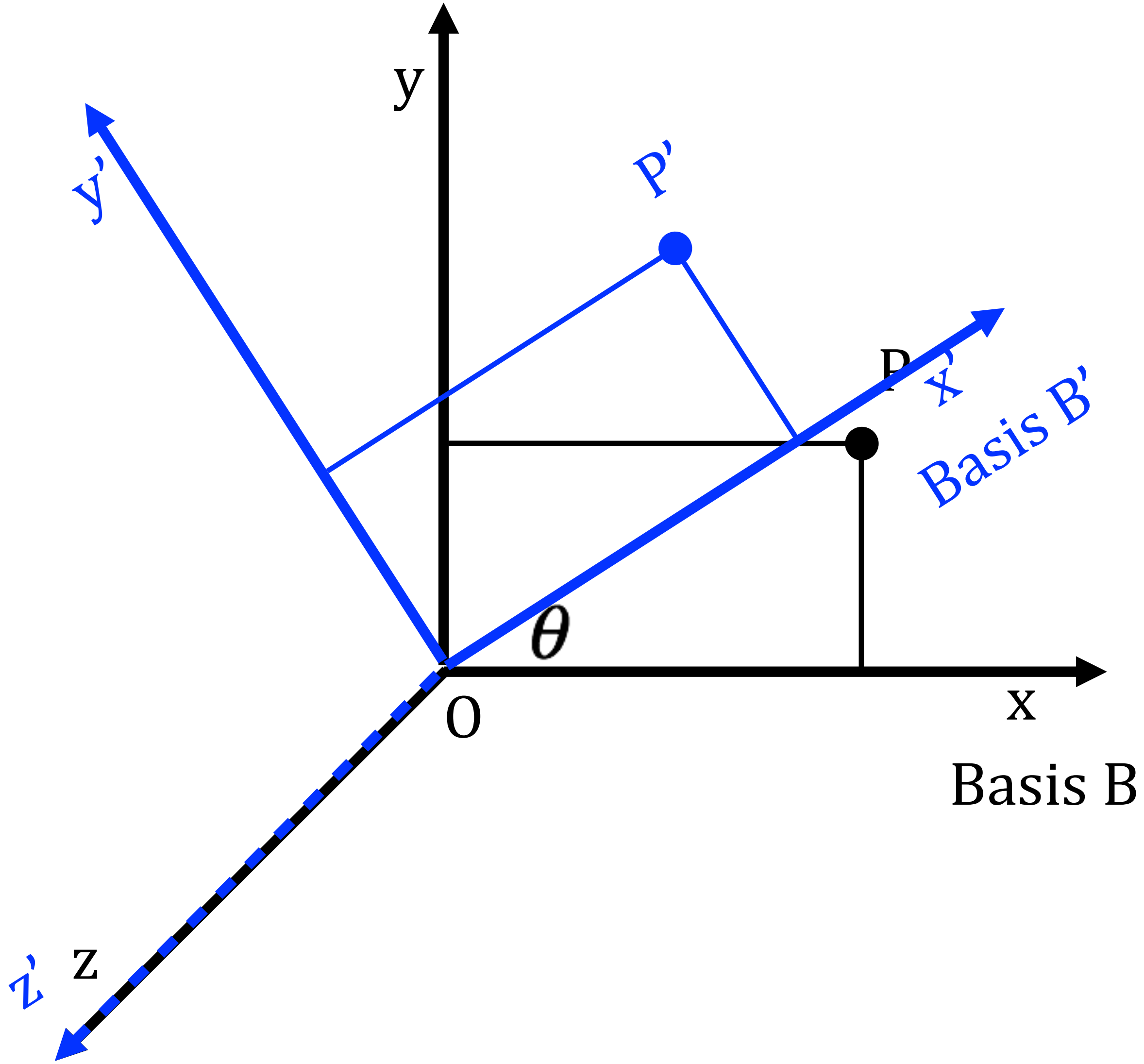
$$\vec{e}_{x'} = (\cos\theta, \sin\theta, 0)$$

$$\vec{e}_{y'} = (-\sin\theta, \cos\theta, 0)$$

$$\vec{e}_{z'} = (0, 0, 1)$$

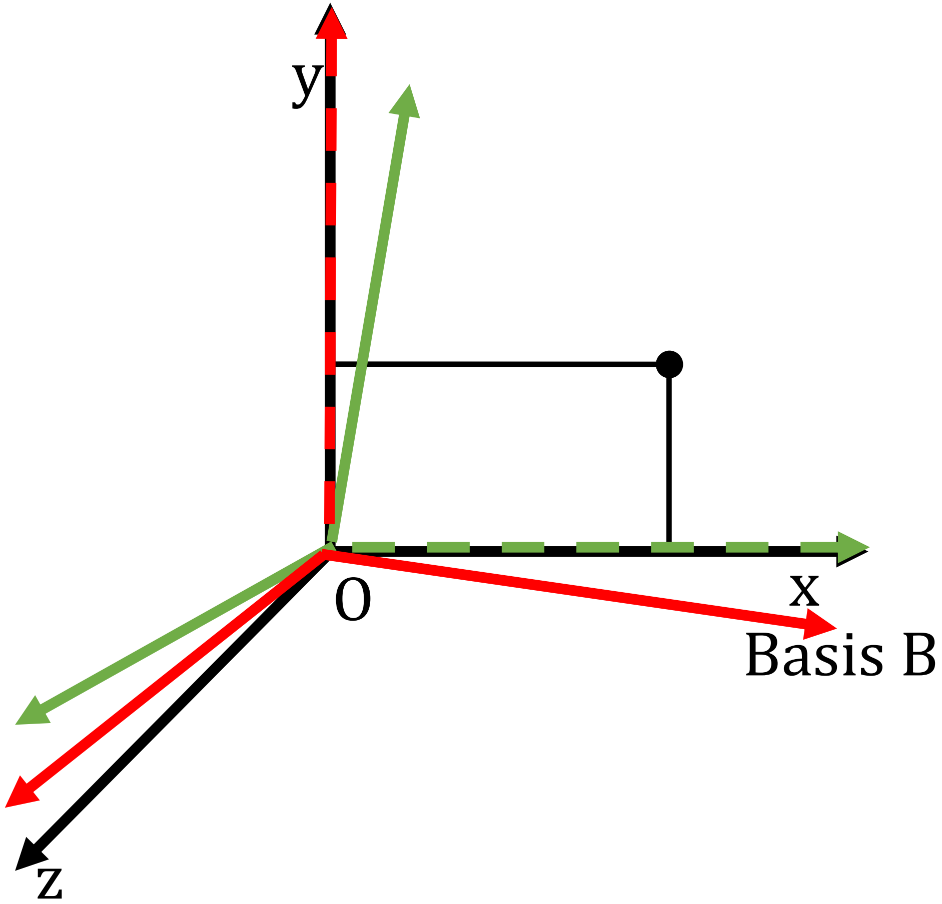
$$R_\theta = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = R_\theta P$$



More Rotations

- Stand at the end of the axis of rotation
  - face in, and rotation is CCW
- Find the matrix for a rotation around y
- Find the matrix for a rotation around x
- Any orthonormal basis is a rotation
  - How can we find the axis?



Inverse Rotations

- Inverse rotation given by transpose
  - only works for (orthonormal) rotations

$$RR^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos^2\theta + \sin^2\theta & -\cos\theta\sin\theta + \sin\theta\cos\theta \\ 0 & -\cos\theta\sin\theta + \sin\theta\cos\theta & \sin^2\theta + \cos^2\theta \end{bmatrix}$$

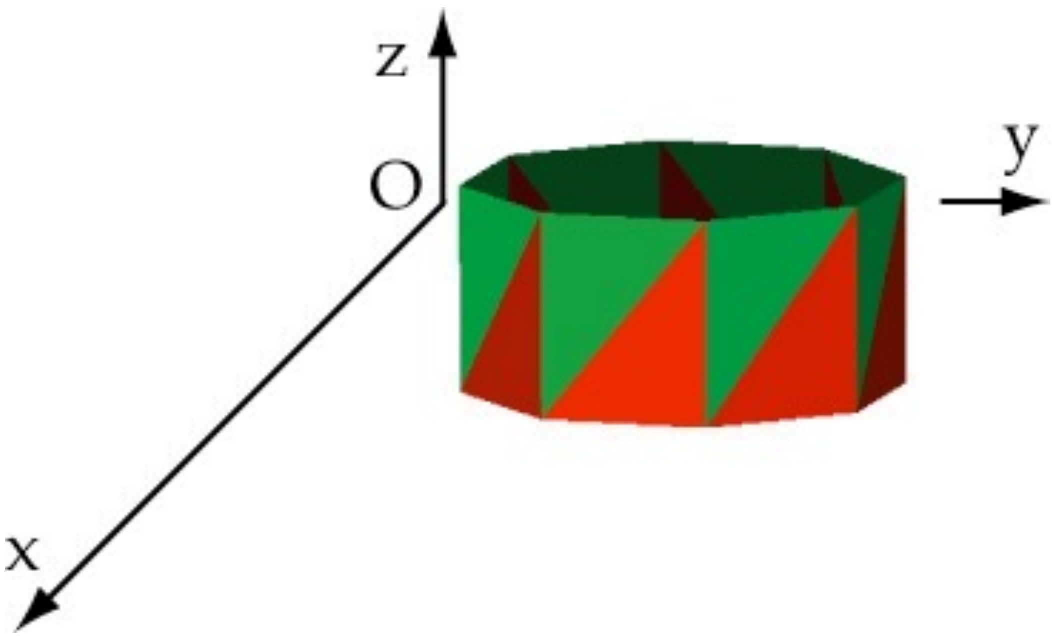
$$= I$$

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

UNIVERSITY OF LEEDS

Scaling

- Shrink or grow one coordinate, but not others
- Negative scale is reflection

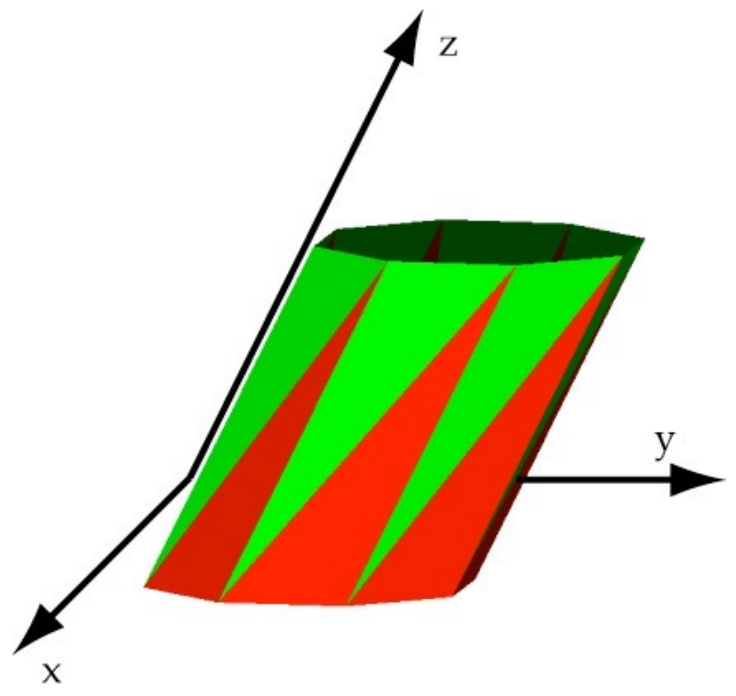


$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$$

UNIVERSITY OF LEEDS

Shearing

- Slide the top sideways
  - by a vector multiplied by z



$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.5 \\ 0 & 0 & 1 \end{bmatrix}$$

Transforming Normals

- If a surface rotates, the normal must also do so
  - We will apply the rotation to the normal too
- Scaling distorts normals
  - as does shearing
- So mostly used in modelling software
  - where they can be fixed without time issues

Distorted Normals

- Not a problem for rotation / translation
  - BIG problem for scaling / shearing

$$n \cdot p - c = 0$$

$$\left(\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \right) \cdot \left(\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \right) - c = \begin{bmatrix} 2n_x \\ n_y \\ n_z \end{bmatrix} \cdot \begin{bmatrix} 2p_x \\ p_y \\ p_z \end{bmatrix} - c$$

$$= 4n_x p_x + n_y p_y + n_z p_z - c$$

$$= 3n_x p_x + \left(n_x p_x + n_y p_y + n_z p_z \right) - c$$

$$= 3n_x p_x + n \cdot p - c$$

$$= 3n_x p_x$$

Translation

- Translation moves an object
  - in the direction given by a vector
  - add the vector to each vertex

$$p' = p + \vec{v}$$

- Can't do it with Cartesian matrix multiplication

Applying Transformations

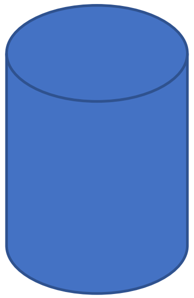
- Rotate a cylinder, then translate it:

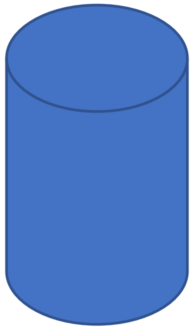
$$p' = \vec{v} + Rp$$

- Translate a cylinder, then rotate it:

$$p' = R(\vec{v} + p)$$

- Specify transformations in reverse order.
- Closest to point, first to be applied.



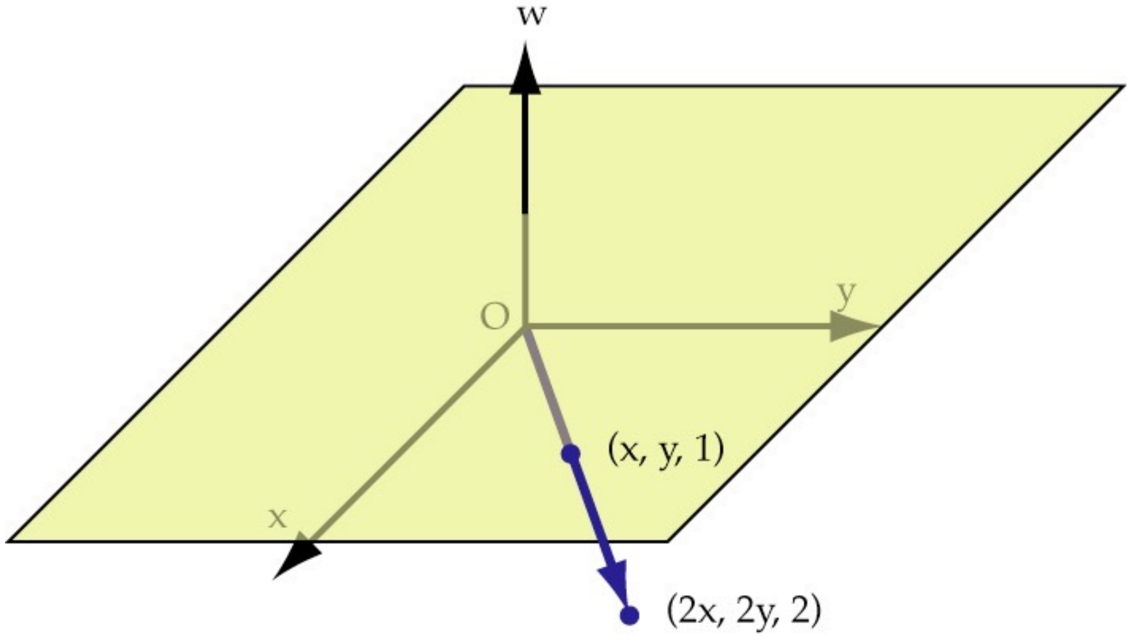


Three Problems

1. Represent translation in matrix form
2. Apply sequences of transformations efficiently
3. Represent perspective in matrix form

2D Homogeneous Coords

- Homogeneous coords exist in all dimensions
  - In 2D, (x, y) becomes (x, y, 1)
  - w is a *scale* factor: usually 1
  - (x, y, w) refers to the point
  - (1, 2, 1) is the same as (3, 6, 3)
- Each point becomes a line in space
  - H.c. can represent projection as well

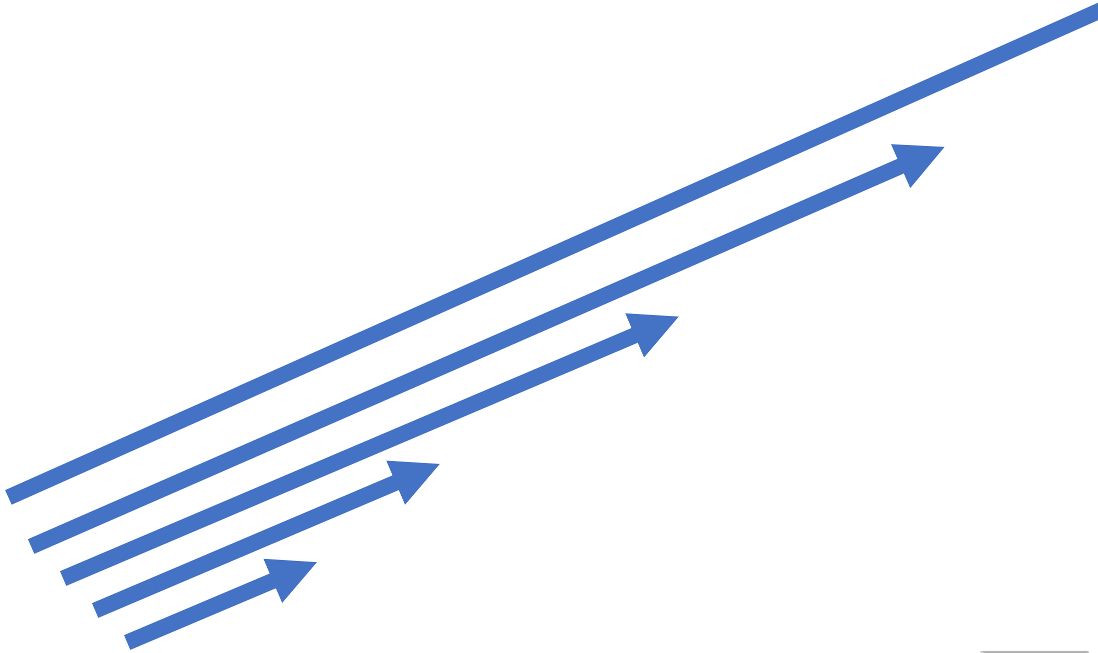


3D Homogeneous Coords.

- In 3D, homogeneous coordinates are (x, y, z, w)
  - x, y, z are the same as usual (almost)
  - w is the same as in 2D
- (x, y, z, w) refers to the point $(\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$
- (1, 2, 3, 1) is the same as (3, 6, 9, 3)

Homogeneous Vectors

- Vectors can be written as: (x, y, z, 0)
- Why?
  - Consider $\lim_{w \to 0} (\frac{x}{w}, \frac{y}{w}, \frac{z}{w})$
  - As $w \to 0$, the point travels outwards
  - So the vector (x, y, z) is (x, y, z, 0)
- Alternately, (x, y, z, 0) is infinitely far out



Rotations

- Transformation matrices add 1 row/col
- Result of the multiplication is the same

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

Scaling

- Again, pretty much the same

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \\ s_z z \\ w \end{bmatrix}$$

Shearing

$$\begin{bmatrix} s_x & s_{xy} & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} s_x x + s_{xy} y \\ s_y y \\ s_z z \\ w \end{bmatrix}$$

Translation

- To translate (x, y, z, w) by (a, b, c, 1):

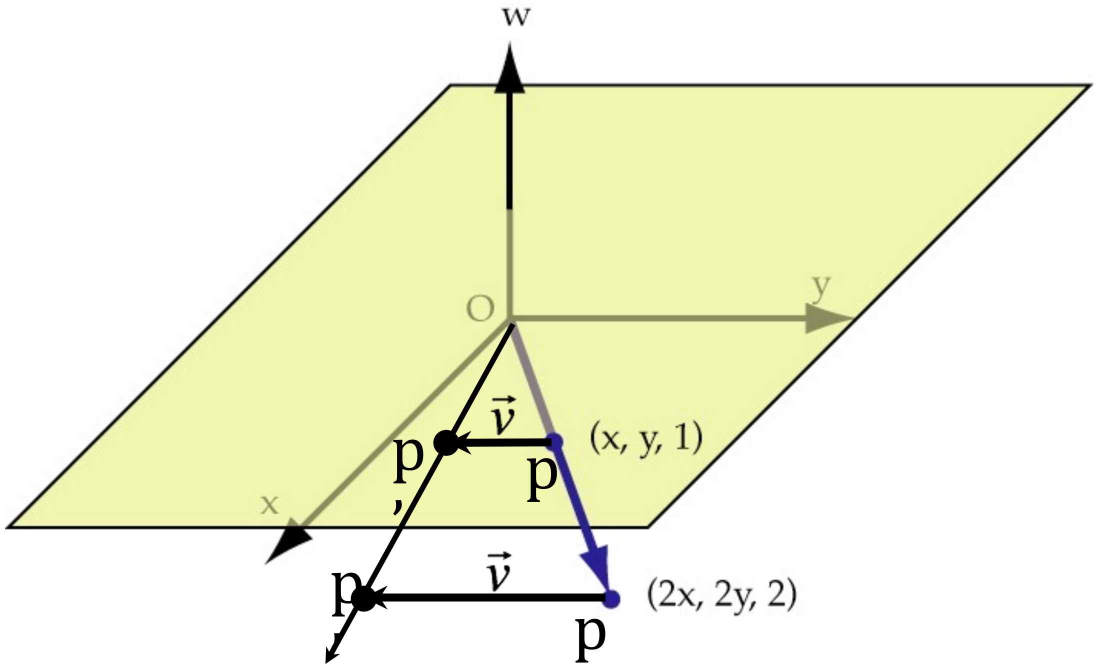
$$\begin{pmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x + aw \\ y + bw \\ z + cw \\ w \end{pmatrix} = \begin{pmatrix} \frac{x}{w} + a \\ \frac{y}{w} + b \\ \frac{z}{w} + c \end{pmatrix}$$

- Now we can do it by multiplication!

Why this Works (2D)

- In 2D, third column (w) is the same as z
- Which means we have a shear matrix
- p moves in direction v proportional to w
- And lands on the line representing p'

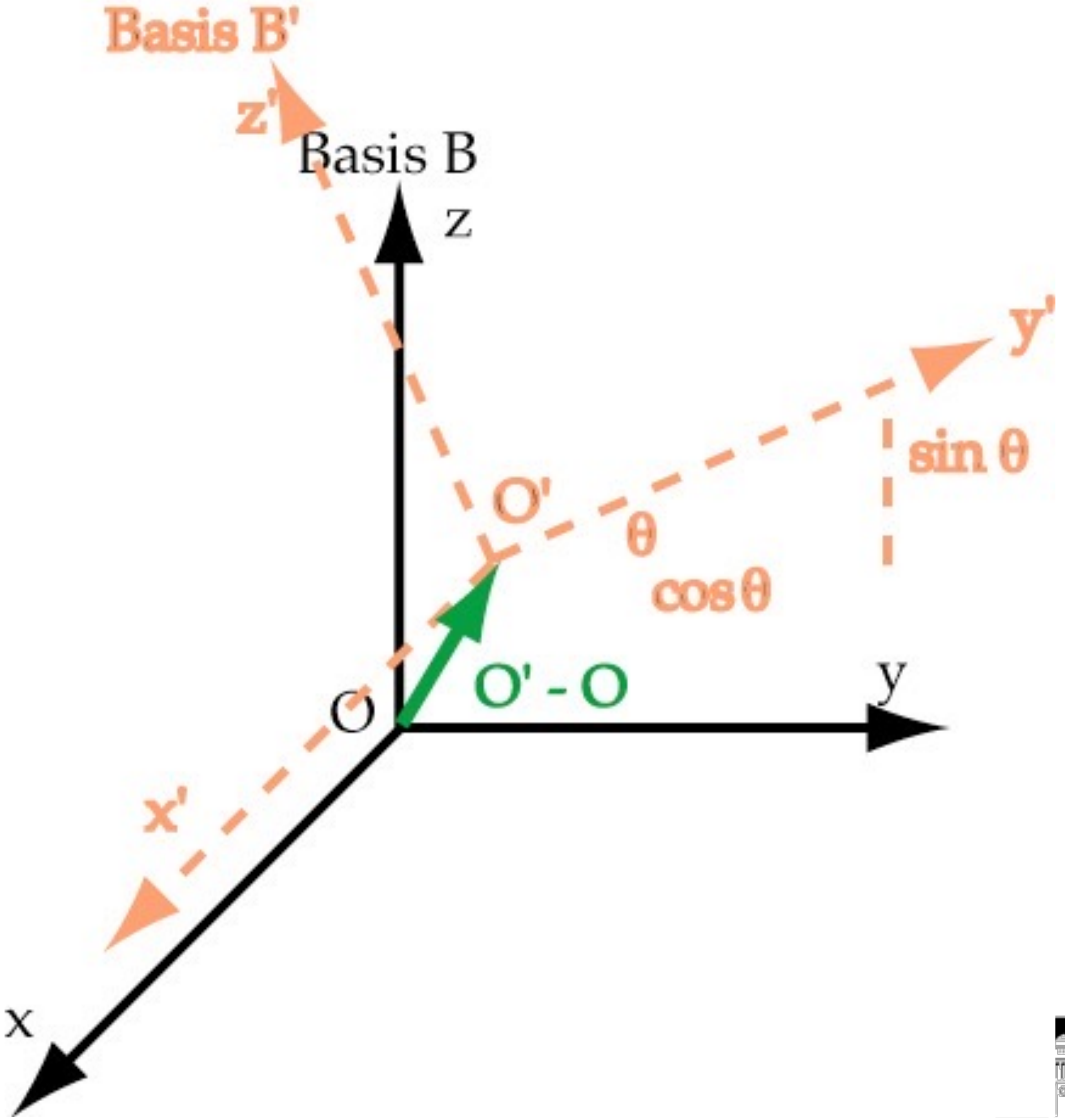
$$p' = \begin{bmatrix} 1 & 0 & v_x \\ 0 & 1 & v_y \\ 0 & 0 & 1 \end{bmatrix} p$$



Arbitrary Rotation

- Translate by (O - O')
- Rotate at O
- Translate by (O' - O)
- Compose the matrices:

$$M = TRT^{-1}$$



UNIVERSITY OF LEEDS

Composition

$$
\begin{aligned}
Mp &= TRT^{-1}p \\
&= \begin{bmatrix} 1 & 0 & 0 & -a \\ 0 & 1 & 0 & -b \\ 0 & 0 & 1 & -c \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & \cos\theta & -\sin\theta & b \\ 0 & \sin\theta & \cos\theta & c \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & \cos\theta & -\sin\theta & b\cos\theta - c\sin\theta - b \\ 0 & 0 & 1 & b\sin\theta + c\cos\theta - b \\ 0 & 0 & 0 & 1 \end{bmatrix}
\begin{bmatrix} p_x \\ p_y \\ p_z \\ p_w \end{bmatrix}
\end{aligned}
$$

Cost of Transformation

- Cartesian:
  - Add the vector, then rotate, then subtract
  - 3 adds, 9 multiplies + 6 adds, 3 adds
  - 12 adds, 9 multiplies total
- 3 rotations:
  - 36 adds, 27 multiplies
- For 100 vertices, 3,600 adds, 2,700 multiplies

- Homogeneous:
- 3 rotations:
  - 4,800 multiplies, 3,600 adds (4x4 matrices)
- But we can *pre-compute* a single matrix:
  - 128 multiplies & 96 adds
  - Then 1,600 multiplies, 1,200 adds
  - Total of 1,728 multiplies, 1,296 adds
  - Over 50% faster, and less book-keeping

Homogeneous Matrix

- Divides into
  - rotation (r)
    - also scale, shear
  - translation (t)
  - projection (p)
  - 1

$$\left[\begin{array}{ccc|c} r & r & r & t \\ r & r & r & t \\ r & r & r & t \\ \hline p & p & p & 1 \end{array}\right]$$

Projection

Orthographic Projection

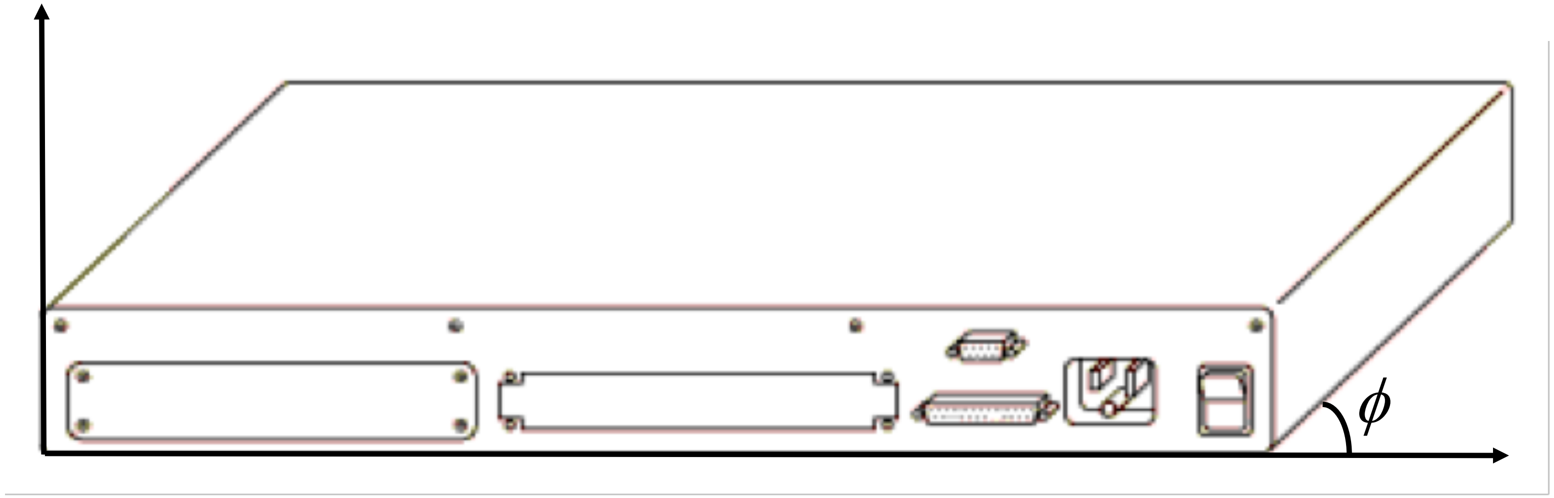
- Projects a point p in the world
  - to a point q in the image plane
  - essentially an identity matrix
  - Keeps z (we'll need it later)

$$\Pi_o = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad p' = \Pi_o p = p$$

Oblique Projection

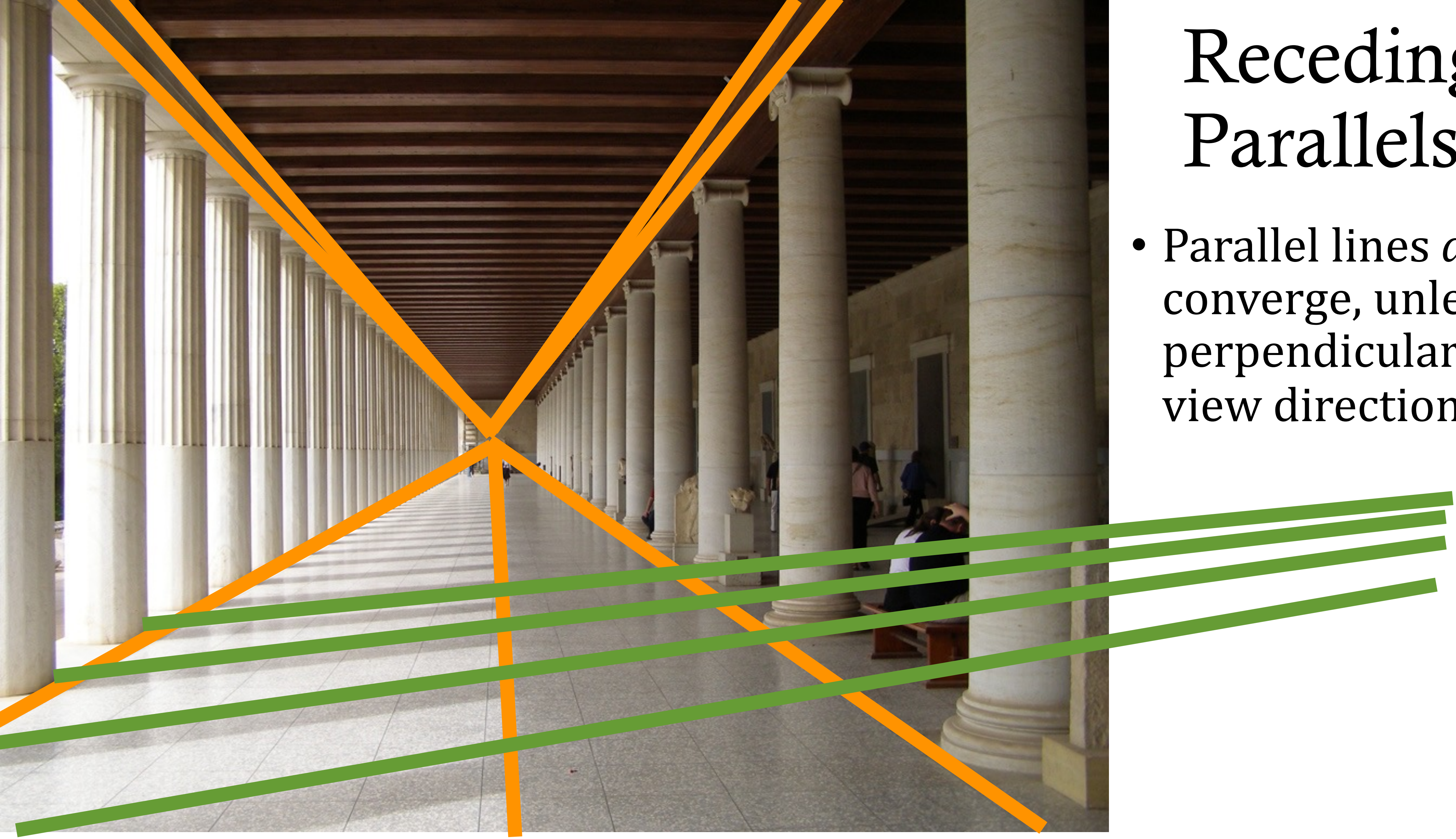
- Slants lines perpendicular to plane at a chosen angle ϕ

$$\Pi_{ob} = \begin{bmatrix} 1 & \sin\phi & 0 & 0 \\ 0 & \cos\phi & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Perspective Projection

- Lines are parallel in world but converge in image to a *vanishing point.*



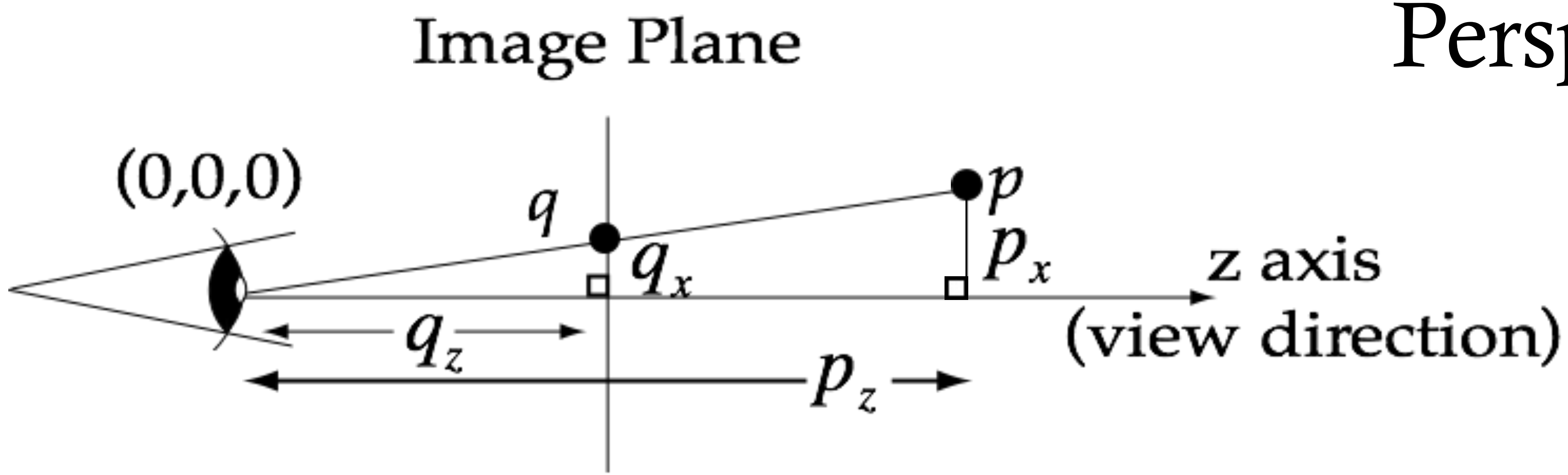
Receding Parallels

- Parallel lines *always* converge, unless perpendicular to view direction

UNIVERSITY OF LEEDS

Mathematical Perspective

- Project p from the world to q in image plane



- Use similar triangles to get coordinates

$$\frac{q_x}{q_z} = \frac{p_x}{p_z}$$

$$q_x = q_z \frac{p_x}{p_z} = d \frac{p_x}{p_z} = p_x \frac{d}{p_z}$$

d is distance from eye to image plane (usually 1)

All 3 Coordinates

$$
\begin{aligned}
(q_x, q_y, q_z) &= \left(p_x \cdot \frac{d}{p_z}, p_y \cdot \frac{d}{p_z}, d\right) \\
&= \left(p_x \cdot \frac{d}{p_z}, p_y \cdot \frac{d}{p_z}, p_z \cdot \frac{d}{p_z}\right) \\
&\cong \left(p_x, p_y, p_z, \frac{p_z}{d}\right) \text{ (homogeneous coordinates)} \\
&\cong \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix}
\end{aligned}
$$

Lines are Lines

- A projected line is still a line (parametric form)
- Represent line in homogeneous coords
  - and apply the projection matrix

$$
\begin{aligned}
l &= p + \vec{v}t \\
p &= (p_x, p_y, p_z) \\
v &= (v_x, v_y, v_z)
\end{aligned}
$$

$$
\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{bmatrix}
\begin{bmatrix} p_x + v_x t \\ p_y + v_y t \\ p_z + v_z t \\ 1 \end{bmatrix}
= \begin{bmatrix} p_x + v_x t \\ p_y + v_y t \\ p_z + v_z t \\ \frac{p_z + v_z t}{d} \end{bmatrix}
\cong \begin{bmatrix} \frac{d(p_x + v_x t)}{p_z + v_z t} \\ \frac{d(p_y + v_y t)}{p_z + v_z t} \\ d \end{bmatrix}
$$

Simple Case $(v_z = 0, d = 1)$

- Line perp to view
  - Assume $p_z \neq 0$
- The line scaled by $\frac{1}{p_z}$

$$
\begin{aligned}
\begin{bmatrix} \frac{d(p_x+v_xt)}{p_z+v_zt} \\ \frac{d(p_y+v_yt)}{p_z+v_zt} \\ \frac{d(p_z+v_zt)}{p_z+v_zt} \end{bmatrix} &= \begin{bmatrix} \frac{1(p_x+v_xt)}{p_z+0t} \\ \frac{1(p_y+v_yt)}{p_z+0t} \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{p_x}{p_z} + \frac{v_x}{p_z}t \\ \frac{p_y}{p_z} + \frac{v_y}{p_z}t \\ 1 \end{bmatrix} \\
&= \begin{bmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \\ 1 \end{bmatrix} + \begin{bmatrix} \frac{v_x}{p_z} \\ \frac{v_y}{p_z} \\ 0 \end{bmatrix} t \\
&= \frac{1}{p_z} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} + \frac{1}{p_z} \begin{bmatrix} v_x \\ v_y \\ 0 \end{bmatrix} t \\
&= \frac{1}{p_z} p + \frac{1}{p_z} \vec{v} t \\
&= \frac{1}{p_z} (p + \vec{v} t)
\end{aligned}
$$

Hard Case ($v_z = 1, p_z = 0, d = 1$)

$$\begin{bmatrix} \frac{d(p_x+v_xt)}{p_z+v_zt} \\ \frac{d(p_y+v_yt)}{p_z+v_zt} \\ \frac{d(p_z+v_zt)}{p_z+v_zt} \end{bmatrix} = \begin{bmatrix} \frac{1(p_x+v_xt)}{0+1t} \\ \frac{1(p_y+v_yt)}{0+1t} \\ \frac{1(p_z+v_zt)}{0+1t} \end{bmatrix}$$

We substitute $u = \frac{1}{t}$
and remember that $v_z = 1, p_z = 0$

$$= \begin{bmatrix} \frac{p_x+v_xt}{t} \\ \frac{p_y+v_yt}{t} \\ \frac{p_z+v_zt}{t} \end{bmatrix} = \begin{bmatrix} v_x + p_xu \\ v_y + p_yu \\ v_z + p_zu \end{bmatrix}$$

$$= \begin{bmatrix} v_x + \frac{p_x}{t} \\ v_y + \frac{p_y}{t} \\ v_z + \frac{p_z}{t} \end{bmatrix} = v + \vec{p}u$$

- Line not perpendicular to view
- Line parameterised by *u=1/t*

UNIVERSITY OF LEEDS

And not just any line

- The point & vector have swapped
  - $p = (p_x, p_y, 0)$ is now the vector
  - $v = (v_x, v_y, 1)$ is now the point
- Two parallel lines have the same $v = (v_x, v_y, 1)$
  - their projections both pass through $(v_x, v_y, 1)$
  - $(v_x, v_y, 1)$ **IS** the *vanishing point*

Foreshortening

- Vertical spacing reduced further away
- One of the cues to depth of image.
- We assumed that z = 0, c = 1
- t is perpendicular distance to image plane
- What happens to evenly spaced points?
- P + 1V, P + 2V, P + 3V
- These map to V + 1 P, V + 1/2 P, V + 1/3 P
- No longer evenly spaced

OpenGL: how does this apply?

- For different projections, there are functions that give you the matrix that you need to use.
- Some in OpenGL, others GLUT (OpenGL utility toolkit).
- 4x4 matrices that can be used with homogeneous coordinates!

glOrtho()

- Given (Left,Right), (Top,Bottom), (Near,Far)

$$\begin{bmatrix} \frac{2}{R-L} & 0 & 0 & -\frac{R+L}{R-L} \\ 0 & \frac{2}{T-B} & 0 & -\frac{T+B}{T-B} \\ 0 & 0 & -\frac{2}{F-N} & -\frac{F+N}{F-N} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Scales & translates to centre view volume
- Projects along *negative* z-axis

glFrustum()

- Given (Left,Right), (Top,Bottom), (Near,Far)

$$\begin{bmatrix} \frac{2N}{R-L} & 0 & \frac{R+L}{R-L} & 0 \\ 0 & \frac{2N}{T-B} & \frac{T+B}{T-B} & 0 \\ 0 & 0 & -\frac{F+N}{F-N} & -\frac{2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Same as glOrtho() but with projection added
- Projection of translation gives shear

gluPerspective()

- Given (Field Of View), Aspect ratio, Near, Far

$$\begin{bmatrix} \frac{f}{Aspect} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & \frac{F+N}{N-F} & \frac{2FN}{N-F} \\ 0 & 0 & -1 & 0 \end{bmatrix}, where\ f = \cot(fov)$$

- Similar, but much simpler to set up

gluLookAt()

- Given eye e, centre c, up $\vec{u}$
- Computes $\vec{f} = c - e$, normalises $\vec{f}$ and $\vec{u}$
- Then computes $\vec{s} = \vec{f} \times \vec{u}$

$$\begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & s_y & s_z & 0 \\ u_x & u_y & u_z & 0 \\ -f_x & -f_y & -f_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Just rotation then translation

Homogeneous Coordinates

- Give *all* affine transformations in a 4x4 matrix:
  - Rotation, Scaling, Shearing
  - Translation
  - Projection (Orthographic, Oblique, Perspective)
- Also represent points & vectors differently
- *And* they are much more efficient in practice
  - Despite having more coordinates!