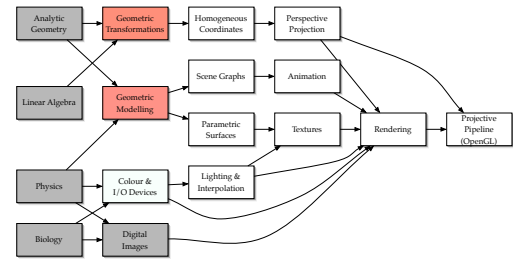


Transformations & Matrix Manipulation

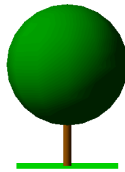
COMP 3003
Autumn 2005

Where We Are



Description

- Draw a sphere
 - radius 10 m
 - centred 7 m above ground
 - colour light green
- Draw a cylinder
 - radius 2 m, height 15 m
 - bottom face on ground
 - colour brown



Composite Modelling

- We build objects from *primitives*
 - e.g. points, lines, triangles
- Can also use *bigger* primitives
 - e.g. spheres, cones, cylinders
- Specify location, &c. with *transformations*

Object Description

- Objects have:
 - shape (what type of primitive)
 - size
 - location

Spheres

- Spheres are easy to move / resize:

A sphere of radius 1 at the origin:

$$x^2 + y^2 + z^2 = 1$$

A sphere of radius r at (x_0, y_0, z_0) :

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

- What about orientation?

Spheres

- Spheres are *symmetric*
 - they are the same in every orientation
 - so we don't have to worry
- But what about *cylinders*?
 - Moving / scaling isn't hard
 - Orientation (rotation) is hard

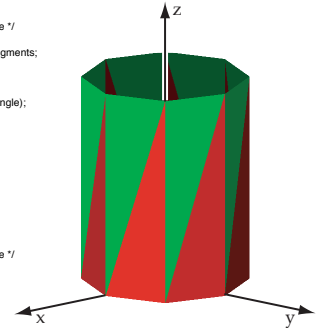
A Vertical Cylinder

```
for (float i = 0.0; i < nSegments; i += 1.0)
{ /* a loop around circumference of a tube */
float angle = PI * i * 2.0 / nSegments;
float nextAngle = PI * (i + 1.0) * 2.0 / nSegments;

/* compute sin & cosine */
float x1 = sin(angle), y1 = cos(angle);
float x2 = sin(nextAngle), y2 = cos(nextAngle);

/* draw top triangle */
drawTriangle( x1, y1, 0.0,
             x1, y1, 1.0,
             x2, y2, 1.0 );

/* draw bottom triangle */
drawTriangle( x1, y1, 0.0,
             x1, y1, 1.0,
             x2, y2, 1.0 );
} /* a loop around circumference of a tube */
```



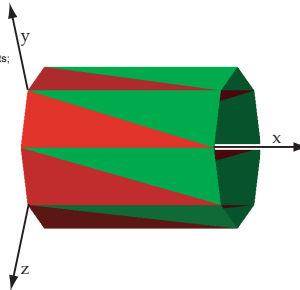
A Horizontal Cylinder

```
for (float i = 0.0; i < nSegments; i += 1.0)
{ /* a loop around circumference of a tube */
float angle = PI * i * 2.0 / nSegments;
float nextAngle = PI * (i + 1.0) * 2.0 / nSegments;

/* compute sin & cosine */
float y1 = sin(angle), z1 = cos(angle);
float y2 = sin(nextAngle), z2 = cos(nextAngle);

/* draw top triangle */
drawTriangle( 0.0, y1, z1,
             1.0, y1, z1,
             1.0, y2, z2 );

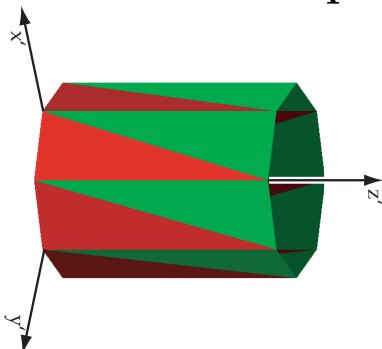
/* draw bottom triangle */
drawTriangle( 0.0, y1, z1,
             1.0, y1, z1,
             1.0, y2, z2 );
} /* a loop around circumference of a tube */
```



Drawbacks

- We need code for *each* cylinder
 - although they're essentially the *same*
- So how can we *reuse* code?
 - draw a standard cylinder
 - and move it around easily

A Different Viewpoint



New Axes for Old

- Tilt your head to the right
- Now "up" in your vision is changed
- The cylinder's coordinates haven't
- It's in a different *coordinate system*
- Described by a new *basis*
 - consisting of axes x' , y' , z'

Changing Systems

$$p' = p_x \bar{x}' + p_y \bar{y}' + p_z \bar{z}'$$

$$= p_x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + p_y \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} + p_z \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

- Let $\bar{x}', \bar{y}', \bar{z}'$ be axes
- P is a weighted sum
- Express as a matrix

$$= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ \underline{0} & \underline{-1} & \underline{0} \\ \bar{x}' & \bar{y}' & \bar{z}' \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix}$$

Basis Vectors

- Axes can be any set of 3 *independent* vectors
 - Call this set a *basis*
- Are (1, 0, 0), (0, 1, 0), (7, 13, -25) a basis?
- Are (1, 1, 2), (0, 1, 2), (1, 0, 1) a basis?
- Are (1, 1, 4), (1, -1, 3), (0, 2, 1) a basis?

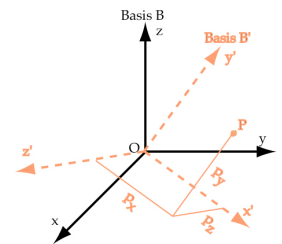
Orthonormal Basis

- The “best” bases are *orthonormal*
 - vectors are mutually perpendicular
 - length 1
- Cartesian coordinates are orthonormal
 - that’s why they’re so useful

Changing Bases

- Assume P is in B'
- Where is P in B?
- Find x', y', z' in B
- Multiply by

$$M = [\bar{x}' \quad \bar{y}' \quad \bar{z}']$$
- To reverse, use M^{-1}



Transformations

- Changing basis is a *transformation*
- An operation on vectors, points, &c.
- An *affine* transformation preserves lines
 - Lines before are still lines after
 - Angles and lengths may change
- Cartesian matrices are *linear* transformations
 - and are always *affine*

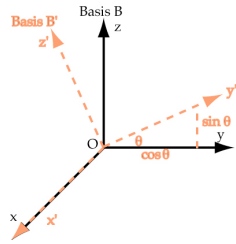
What can Bases Do?

- *Rotation* (any)
- *Scaling*, including
 - *Reflection*
 - *Perpendicular Projection*
- *Shearing*
- BUT not *Translation* or *Perspective Projection*

Rotation Matrices

- Rotate CW around x-axis by angle θ :
 - from B to B'

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



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 z
- Any orthonormal basis is a rotation
 - How can we find the axis?

Inverse Rotations

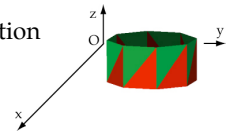
- Inverse rotation given by *transpose*
 - only works for (orthonormal) rotations

$$\begin{aligned} RR^T &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & \sin \theta \\ 0 & -\sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta \\ 0 & \sin \theta & \cos \theta \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos^2 \theta + \sin^2 \theta & -\cos \theta \sin \theta + \sin \theta \cos \theta \\ 0 & -\cos \theta \sin \theta + \sin \theta \cos \theta & \sin^2 \theta + \cos^2 \theta \end{bmatrix} \\ &= I \\ R^{-1} &= R^T \end{aligned}$$

Scaling

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

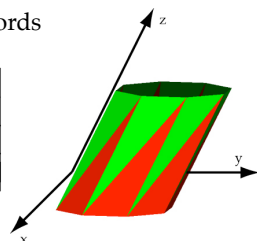
$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.25 \end{bmatrix}$$



Shearing

- Slide the top sideways
 - adds $0.5z$ to y coords

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



Normal Vectors

- OpenGL transforms:
 - vertices
 - normal vectors
- Non-uniform scaling distorts normals
 - as does shearing
 - uniform scaling has OpenGL problems

Distorted Normals

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

$$\begin{aligned}
 n \cdot p - c &= 0 \\
 \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \cdot \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} - c &= \begin{bmatrix} 2n_x \\ n_y \\ n_z \end{bmatrix} \cdot \begin{bmatrix} 2p_x \\ p_y \\ p_z \end{bmatrix} - c \\
 &= 4n_x p_x + n_y p_y + n_z p_z - c \\
 &= 3n_x p_x + (n_x p_x + n_y p_y + n_z p_z) - c \\
 &= 3n_x p_x + n \cdot p - c \\
 &= 3n_x p_x
 \end{aligned}$$

Solutions

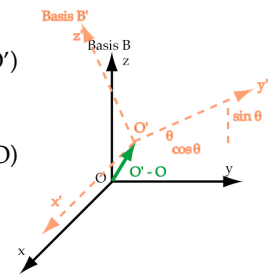
- Calculate normals *after* scaling / shearing
 - more work for the programmer
 - but often done in modelling software
- *Avoid* scaling and shearing
 - e.g. specify cylinder *height* and *radius*

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
 - We'll see a way around this next week
 - For now, *assume* that there is a matrix

Arbitrary Rotation

- Translate by (O - O')
- Rotate at O
- Translate by (O' - O)



Meaning of Transformations

- Two possible interpretations:
 - transformation is *applied* to an object
 - used when *animating* an object
 - transformation *resets* working coordinates
 - i.e. specifies new *basis* for drawing
 - used when *modelling* an object

Transformations in OpenGL

- OpenGL has the following built in:
 - `glTranslatef(x,y,z);`
 - `glRotatef(degrees,axisX,axisY,axisZ);`
 - `glScalef(xScale,yScale,zScale);`
- Which makes life a lot easier (for now)

Applying Transformations

- Rotate a cylinder, then translate it:

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

- Translate a cylinder, then rotate it:

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

- Specify transformations in *reverse order*
 - we'll see why next week

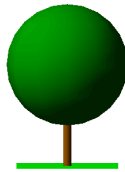
Reversing Transformations

- Once invoked, a transformation *persists*
 - to get rid of it, apply it's inverse
- If you have multiple transformations
 - invert in *opposite* order
 - i.e. always invert the *most recent*
 - we'll see another way next week

Modelling a Tree

```
void RenderTree()
{ /* RenderTree()
  /* ground */
  setColour(greenColour);
  glScalef(5.0,5.0,1.0);
  drawSquare();
  glScalef(0.2,0.2,1.0);
  /* trunk */
  setColour(brownColour);
  glScalef(0.2,0.2,40);
  glTranslatef(0.0,0.0,0.5);
  glRotatef(90.0,1.0,0.0,0.0);
  drawCylinder(12);
  glRotatef(-90.0,1.0,0.0,0.0);
  glTranslatef(0.0,0.0,-0.5);
  glScalef(5.0,5.0,0.25);

  /* foliage */
  setColour(dkgreenColour);
  glTranslatef(0.0,0.0,5.0);
  glScalef(3.0,3.0,3.0);
  drawSphere(12,12);
} /* RenderTree()
```

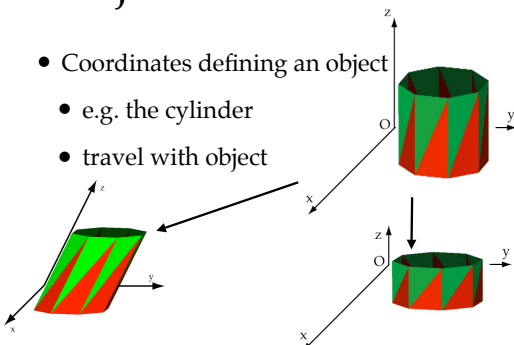


Some Standard Coordinate Systems

- OCS is the *Object Coordinate System*
- WCS is the *World Coordinate System*
- VCS is the *View Coordinate System*
- More next week

Object Coordinates

- Coordinates defining an object
 - e.g. the cylinder
 - travel with object



World Coordinates

- Arbitrary coordinate system
- Where is the origin?
 - The Earth?
 - The Sun?
 - Greenwich meridian?
- Used to keep track of other systems

View Coordinates

- Belong to the *eye* or *camera*
- Tilt your head 90 degrees right
 - “Up” (y) is now to the right
 - “Backward” (z) hasn’t changed
- We can use cross-product to get x

Perpendicular Projection

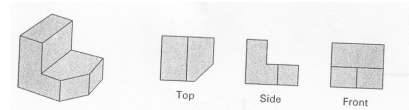
- We’ve seen this already
 - Projection of a point to a line
 - Projection of a point to a plane
 - Projection of a line to a plane
- But *always* along a perpendicular
- Also known as *parallel* projection

Parallel Projection

- We don’t always want perspective
 - Especially for engineering
- We want parallel lines to stay parallel
- And we want to measure distances
- So we use *parallel* projection
 - *Orthographic* or *Orthogonal*
 - *Oblique* (a slanted view)

Orthographic Projection

- Projection perpendicular to view plane
- Common in science & engineering

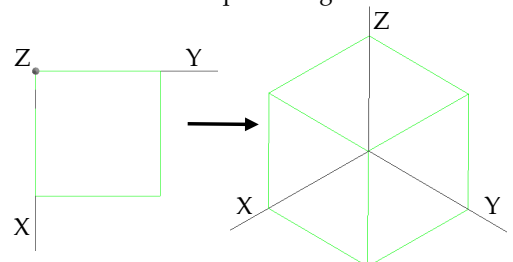


Orthographic Projection

- Projection parallel to y-axis:
 - discard y coordinates
$$P_y = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
- Orthographic projection:
 - rotate it first
 - then project along y-axis
$$P_y \vec{v} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} = \begin{bmatrix} v_x \\ v_z \end{bmatrix}$$

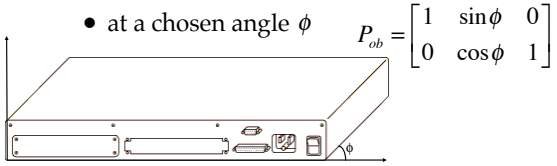
Exercise

- What matrix is required to get this view?



Oblique Projection

- Shows a slanted view of an object
- Slants lines perpendicular to plane
- at a chosen angle ϕ



Tomorrow

- Animation