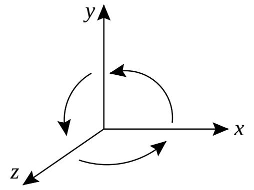
Outline

- 3D coordinate systems.
- Transformations in 3D.
- Transforming objects.
- Transformations as a change in coordinate system.

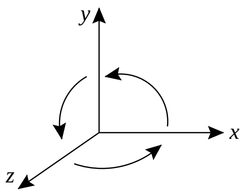
3D coordinate systems

- Although very similar to the 2D system we need to take some care in 3D.

Axis of rotation	Dir. of +ve rotation
x	$y \to z$
y	$z \to x$
z	$x \to y$



3D coordinate systems



- We assume a right handed coordinate system.
- It means that objects further away from you have a smaller z value.
- This may mean a more negative value ....

Transformations in 3D

- A general point $\boldsymbol{r} = [x, y, z]'$ will be represented in homogeneous coordinates by $\boldsymbol{r} = [x/w, y/w, z/w, 1]'$.
- translation:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = T\boldsymbol{r} \; ;$$

- scaling:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = S\boldsymbol{r} \; .$$

Transformations in 3D

- Rotation is a little more tricky, since we need to be careful about which axis we are rotating.
- Rotation about the z axis (seen this before):

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = R_z\boldsymbol{r} \; ,$$

- Rotation about the x axis:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = R_x\boldsymbol{r} \; ,$$

Transformations in 3D

- Finally rotation about the y axis:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = R_y\boldsymbol{r} \; .$$

- Recall OpenGL uses: `glRotate#(angle, x, y, z)`.
- Thus the messy matrix stuff is hidden from us.

Transformations in 3D

- A shear in (x, y) is given by:

$$\boldsymbol{r}^* = \begin{bmatrix} x^* \\ y^* \\ z^* \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & h_x & 0 \\ 0 & 1 & h_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = H_{x,y}\boldsymbol{r} \; .$$

- Included for completeness.

Transformations in 3D

- All the above transformation matrices have inverses.
- We can again combine the transformations to give a general transformation matrix $M =$

$$\begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R^* & \boldsymbol{t}^* \\ \boldsymbol{0} & 1 \end{bmatrix} ,$$

- $\boldsymbol{r}^* = M\boldsymbol{r}$ can be computed using:

$$\begin{bmatrix} x^* \\ y^* \\ z^* \end{bmatrix} = R^* \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \boldsymbol{t}^* \; .$$

Transforming lines and planes

- Lines are generally transformed by transforming the two end points separately.
- Three points define a plane and we can transform planes by transforming these three points.
- However, planes are often defined by the (implicit) equation for a plane, $f(x, y, z) = ax + by + cz + d = 0$.
- Define a column vector, $\boldsymbol{a} = [a, b, c, d]'$ then writing an arbitrary point as $\boldsymbol{p} = [x, y, z, 1]'$, points on the plane satisfy $\boldsymbol{a} \cdot \boldsymbol{p} = 0$ or $\boldsymbol{a}'\boldsymbol{p} = 0$.

Transforming planes

- If we transform all points $\boldsymbol{p}$, with a transformation matrix M ($\boldsymbol{p}^* = M\boldsymbol{p}$), this is equivalent to transforming $\boldsymbol{a}$ so that the condition $\boldsymbol{a}_n'\boldsymbol{p}^* = 0$ defines the transformed plane, where $\boldsymbol{a}_n = Q\boldsymbol{a}$ and Q is the transformation matrix for $\boldsymbol{a}$. Now:

$$(Q\boldsymbol{a})'(M\boldsymbol{p}) = 0 \,,$$

- Use the identity $(AB)' = B'A'$ to write $\boldsymbol{a}'Q'M\boldsymbol{p} = 0$.
- This is true if $Q'M = \alpha I$. Assuming $\alpha = 1$ leads us to:

$$Q = (M^{-1})' \,,$$

- Must ensure M^{-1} exists.

Transformations - useful identities

- One useful coordinate system transformation is given by:

$$M_{R\leftarrow L} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = M_{L\leftarrow R} \,,$$

  which transforms from a left handed to a right handed coordinate systems (and is its own inverse).
- When using multiple transformations we can use the matrix identity $(AB)^{-1} = B^{-1}A^{-1}$ to work out the inverse transformation when the transformation is composite.

Transformations: a change of coordinate system?

- There are two ways to regard any transformation:
  - as a change applied to the object in a fixed coordinate system;
  - as a change in the coordinate system of a fixed object.
- Which interpretation makes most sense will depend on the context.

Transformations in OpenGL

- Basic commands are:
  - `glTranslate#(dx, dy, dz)`
  - `glScale#(sx, sy, sz)`
  - `glRotate#(angle, x, y, z)`
- We use `glPushMatrix()` and `glPopMatrix()` to 'save' the matrix stack.
- The matrices are applied to the vertices in the opposite order they are specified.
- Can define our own matrices: `glLoadMatrix` and `glMultMatrix`.

Modelling in OpenGL

- Imagine we want to build a very simple robot in OpenGL with one arm, which can swing about its body.
- First we will create functions or display lists to draw the objects, which will start at the origin.
- Now we need to work out where to put the translations so that the model will animate as we want – this is not easy.
- We will need to use the OpenGL transformations and the matrix stack.

OpenGL example

```
/* Assume this is called from the display function */

glPushMatrix(); /* Store the current composite transformation matrix */

  glTranslate  the object to where we want to draw it
  drawRobotBody;
  glPushMatrix(); /* Store the current composite transformation matrix */

    glTranslate  the arm to the shoulder location
    glRotate  the arm to the desired angle
    glTranslate  the arm to the rotation point
    drawRobotArm;

  glPopMatrix(); /* Restore the previous matrix */

   now draw any other parts of the robot

glPopMatrix(); /* Restore the previous matrix */

 now draw any other objects -- probably another function
```

Summary

- Having finished this lecture you should:
  - understand the different 3D coordinate systems;
  - know how 3D transformations are applied;
  - be able to transform planes as well as points;
  - see transformations as changes in the coordinate system;
  - use transformations with OpenGL.