

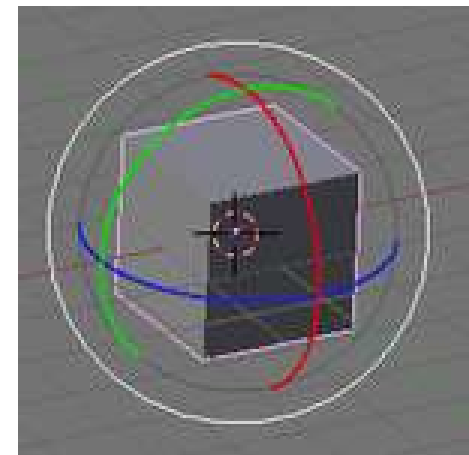
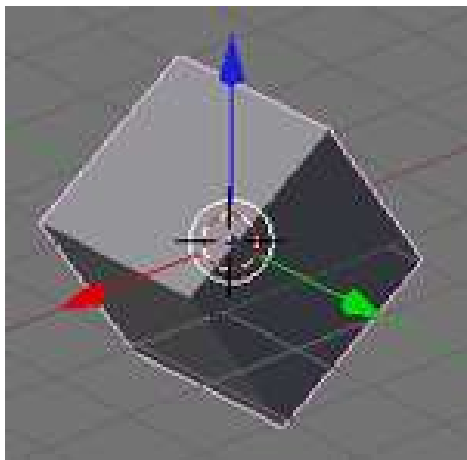
# ICS370 Computer Graphics

Instructor: Toby Daniel

Lecture 9

# 3D Transformations

- A 3D Transform is a mathematical operation that transforms a shape to a new form or position within a three dimensional user space.
- Same idea as 2D transformations
  - defined coordinates (x,y,z)
  - can be expressed as a matrix

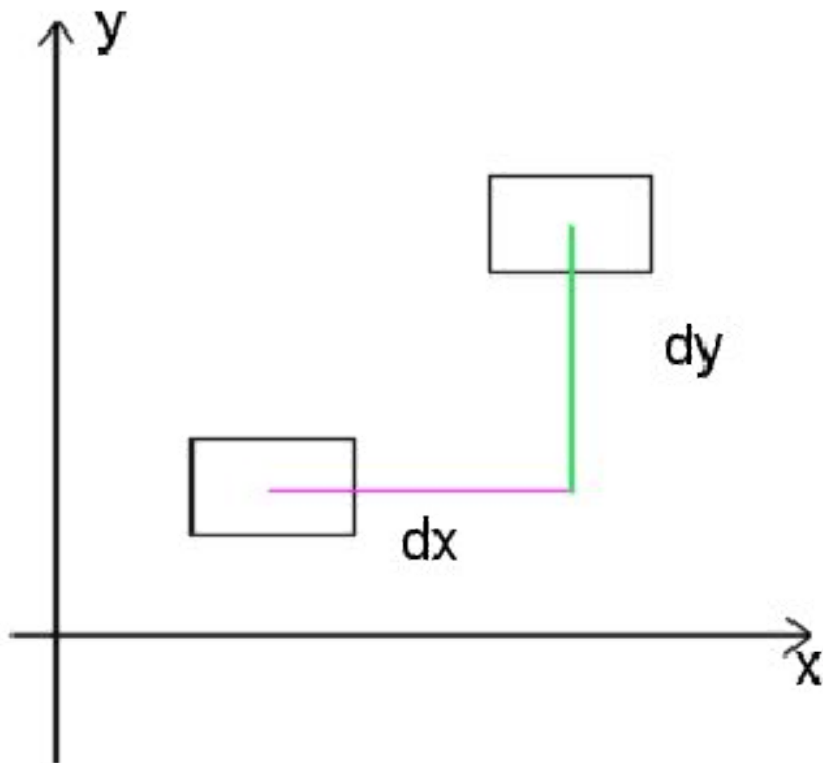


# Translation

- When you move an object in 3D space it's called “translation”.
- If we forget about matrices for the time, translation is very simple.
- If you have a point  $(x,y)$  and want to move it  $dx$  in the x-axis and  $dy$  in the y-axis you do this:

$$x' = x + dx$$

$$y' = y + dy$$



# Translation

- Expanding this into 3 dimensions should be straightforward (just add a Z coordinate).
- Now to put those numbers into a standard matrix is quite easy too, it's just matrices representing column vectors:

$$\begin{bmatrix} dx \\ dy \\ dz \end{bmatrix} + \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ z + dz \end{bmatrix} = \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix}$$

# Composing Translations

- Now, what happens if you move an object in one direction and move it in another direction afterwards?
- That would be expressed in matrices like this:

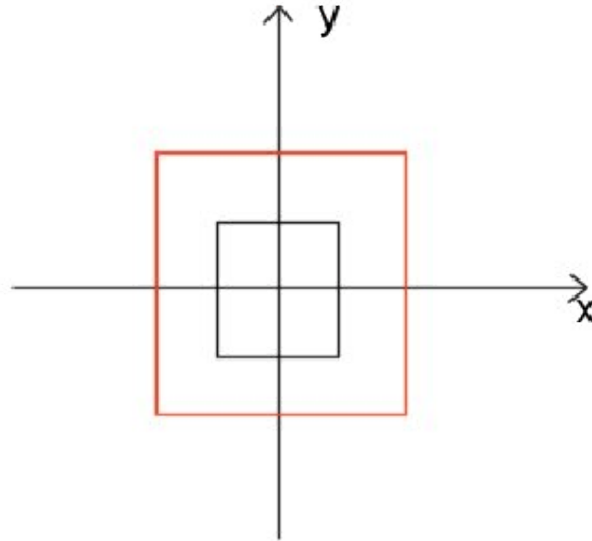
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} + \begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \end{bmatrix} = \begin{bmatrix} x + d_x + d_{x1} \\ y + d_y + d_{y1} \\ z + d_z + d_{y1} \end{bmatrix}$$

- The two translations can be composed into a single matrix looking like this:

$$\begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix} + \begin{bmatrix} d_{x1} \\ d_{y1} \\ d_{z1} \end{bmatrix} = \begin{bmatrix} d_x + d_{x1} \\ d_y + d_{y1} \\ d_z + d_{y1} \end{bmatrix}$$

# Scaling

- Scaling is used to increase or decrease the size of an object.



- If we have a line segment with a length of 1 and want to make it  $n$  times as long we multiply the length with  $n$ . That applies to all the axes and gives us the following equations for scaling:

$$x' = x \cdot n$$

$$y' = y \cdot n$$

# Scaling

- This can also be expressed using a matrix. The scaling matrix is:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

- The previous scale factor  $n$  has been changed for a scale factor for each axis ( $S_x, S_y, S_z$ )
- Scaling can therefore be non-uniform!

Confused? Forgotten how multiplication of matrices works?

# Matrix Multiplication

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \cdot s_x + y \cdot 0 + z \cdot 0 \\ x \cdot 0 + y \cdot s_y + z \cdot 0 \\ x \cdot 0 + y \cdot 0 + z \cdot s_z \end{bmatrix} = \begin{bmatrix} x \cdot s_x \\ y \cdot s_y \\ z \cdot s_z \end{bmatrix}$$

- This scaling works relative to the origin.
- If your object isn't centered around the origin you must translate the object to the origin and scale it.
- Or, you could also use a local coordinate system which is what happens in more advanced 3D programs.

# Multiple Scaling

- Scaling matrices can be composed just like the translation matrices

$$\begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & s_{z1} \end{bmatrix} \times \left( \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \times \begin{bmatrix} x \\ y \\ z \end{bmatrix} \right) = \begin{bmatrix} x \cdot s_x \cdot s_{x1} \\ y \cdot s_y \cdot s_{y1} \\ z \cdot s_z \cdot s_{z1} \end{bmatrix}$$

- Now let's see if we can combine the two scaling matrices before multiplying with the x,y,z point.

$$\begin{bmatrix} s_{x1} & 0 & 0 \\ 0 & s_{y1} & 0 \\ 0 & 0 & s_{z1} \end{bmatrix} \times \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} = \begin{bmatrix} s_x \cdot s_{x1} & 0 & 0 \\ 0 & s_y \cdot s_{y1} & 0 \\ 0 & 0 & s_z \cdot s_{z1} \end{bmatrix}$$

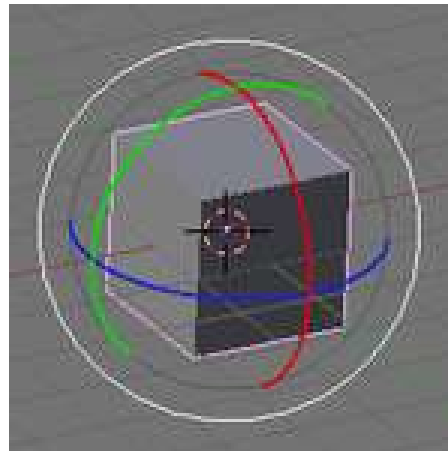
# Rotation

- The last of the 3D transformations which we'll look at is rotation. The mathematical definition is this:

$$x' = x \cdot \cos(V) - y \cdot \sin(V)$$

$$y' = x \cdot \sin(V) + y \cdot \cos(V)$$

- Expressing this as a matrix will depend on how we rotate, through the x, y or z axis.



# Rotation - about the z axis

- The matrix for rotating about the z-axis can be thought of as a basic 2D rotation, the z value for the object never changes.
- We see that in computing the x-coordinate we must multiply x with the cosine to the angle and subtract y multiplied by the sine to the angle. this can be shown as:

$$x' = (x \cdot \cos(V)) + (-y \cdot \sin(V)) + 0 \cdot z$$

- You should now be able to see the connection to the matrix multiplication algorithm and derive the matrix. The same is applied to the y coordinate giving the following matrix for rotation about the z-axis.

$$R_z = \begin{bmatrix} \cos(V) & -\sin(V) & 0 \\ \sin(V) & \cos(V) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

# Rotation - about the y axis

- When you want to rotate about the y axis you do virtually the same as you did when rotating about the z axis.
- Discard the y coordinates so the rotation matrix is:

$$R_y = \begin{bmatrix} \cos(V) & 0 & \sin(V) \\ 0 & 1 & 0 \\ -\sin(V) & 0 & \cos(V) \end{bmatrix}$$

# Rotation - about the x axis

## Question

- Can you derive the matrix for a rotation around the x axis?
  
- It is important to remember that the position of the object relative to the origin plays an important role in the rotation procedure.
- Just like the scaling problem you have to translate the object to the origin or use a local coordinate system.

# Combining 3D transformations

- It would be very useful to combine the three different types of transformations into one.
- Since scaling and rotation matrices both are 3x3 matrices they can be combined together without any problems ... however ...
- since translation matrices are added together we can't directly combine them with scaling and rotation.
- The solution to this is *homogeneous coordinates*.
- The trick is to make all the transformations into 4x4 matrices instead.

# Expanding the Rotation and Scaling Matrices

- Making the rotation and scaling matrices into 4x4 matrices is very easy, you just add another column and another row.

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(V) & -\sin(V) & 0 \\ 0 & \sin(V) & \cos(V) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- If you use 4x4 matrices you must store a point in a 4x1 matrix too, our new point looks like this:

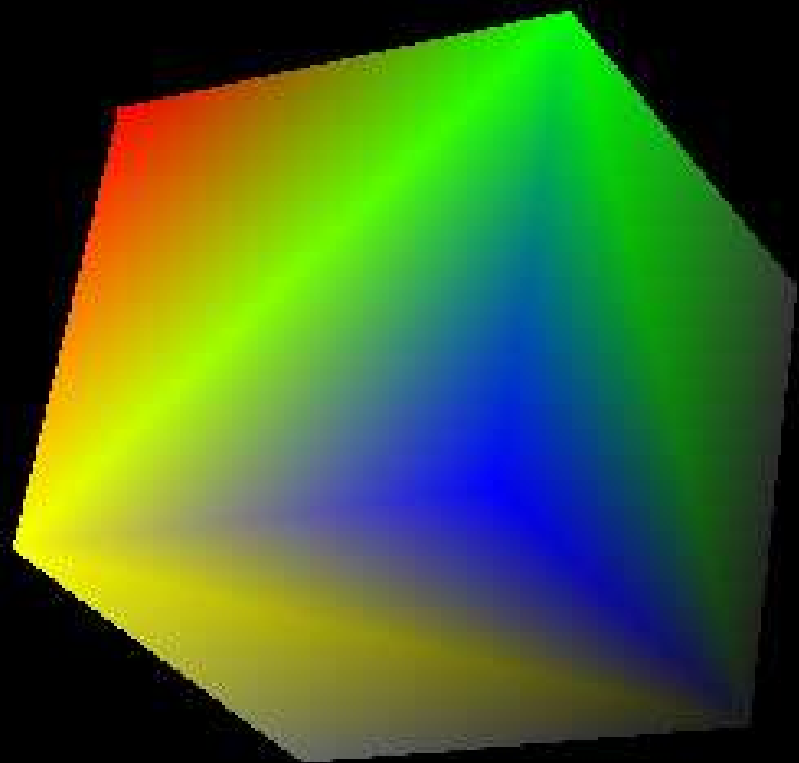
$$\begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix}$$

# Expanding the Translation Matrix

- Expanding the translation matrix to our new 4x4 size is a bit tricky
- We don't want to multiply our point with a scalar, we want to add something to it. So we must preserve the (x,y,z,W) components. That's done by adding 1's in a diagonal.
- Next we want to add a scalar to the coordinates

$$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Now we have all the transformations in the same form, we can use them in combination!





# OpenGL



- OpenGL (**Open Graphics Library**) is a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics.
- The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives.
- OpenGL was developed by Silicon Graphics and is popular in the video games industry where it competes with Direct3D on Windows platforms.
- OpenGL is widely used in CAD, virtual reality, scientific visualization, information visualization, flight simulation and video game development.

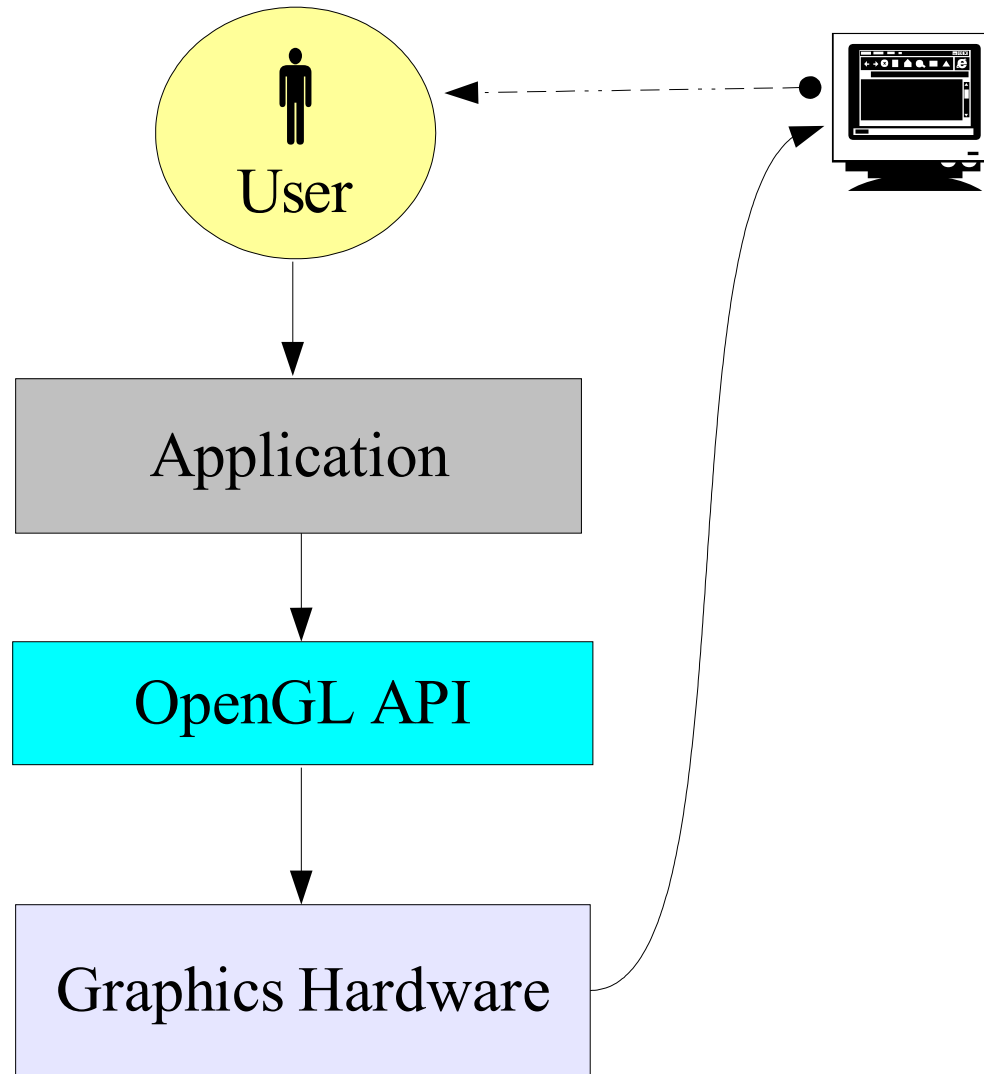


# OpenGL



- OpenGL is a specification, meaning it is simply a document that describes a set of functions and the precise behaviours that they must perform.
- From this specification, hardware vendors create implementations — libraries of functions created to match the functions stated in the OpenGL specification, making use of hardware acceleration where possible.
- Hardware vendors have to meet specific tests to be able to qualify their implementation as an OpenGL implementation.

# OpenGL





# OpenGL



- OpenGL serves two main purposes:
  - To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
  - To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).
- OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. This contrasts with descriptive APIs, where a programmer only needs to describe a scene and can let the library manage the details of rendering it.



# OpenGL



- The **Graphics Pipeline** accepts some representation of a three-dimensional scene as an input and results in a fully rendered 2D raster image as output.
- See the PDF file - OpenGL State Machine

# x-axis matrix answer

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(V) & -\sin(V) \\ 0 & \sin(V) & \cos(V) \end{bmatrix}$$