Transformations & Matrices

OpenGL

Learning Outcomes

- Have a deeper insight into the functioning of the Transformation Pipeline
- Have a clearer understanding of glTranslate, glRotate and glScale functions in this context
- Understand usage of the identity matrix
- Be able to appreciate the difference between the projection and modelview matrices.

The Role of Matrices

- An exceptionally powerful mathematical tool that greatly simplifies the process of solving one or more equations with variables that have complex relationships to each other.

- E.g. a point in space represented by x, y, and z coordinates, and need to compute where that point is if you rotate it a number of degrees around some arbitrary point and orientation.

- Use Matrices because the new x coordinate depends not only on the old x coordinate and the other rotation parameters, but also on the y and z coordinates.

The Matrix

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix} \quad \begin{bmatrix} 0 & 42 \\ 1.5 & 0.877 \\ 2 & 14 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

- A set of numbers arranged in uniform rows and columns—in programming terms, a two-dimensional array.

- Doesn’t have to be square, but each row or column must have the same number of elements as every other row or column in the matrix.

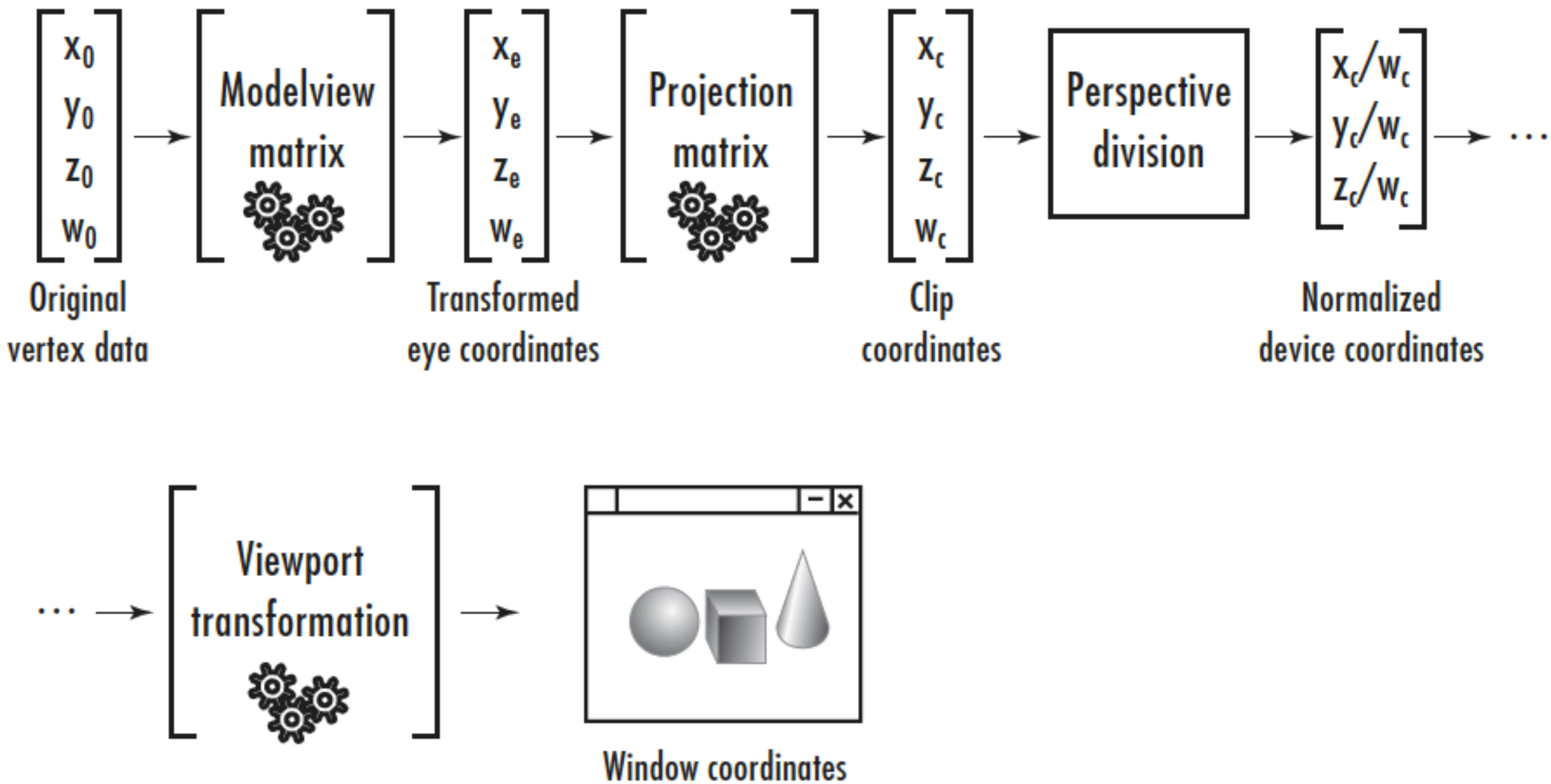
- It is valid for a matrix to have a single column or row.

- A single row or column of numbers is also more simply called a vector

Scalars, Vectors & Matrices

- A scalar is just an ordinary single number used to represent magnitude or a specific quantity
- Matrices can be multiplied and added together, but they can also be multiplied by vectors and scalar values.
- Multiplying a point (a vector) by a matrix (a transformation) yields a new transformed point (a vector).

The Transformation Pipeline



1. The vertex is converted to a 1×4 matrix in which the first three values are the x, y, and z coordinates. The fourth number is a scaling factor that you can apply manually by using the vertex functions that take four values. This is the w coordinate, usually 1.0 by default

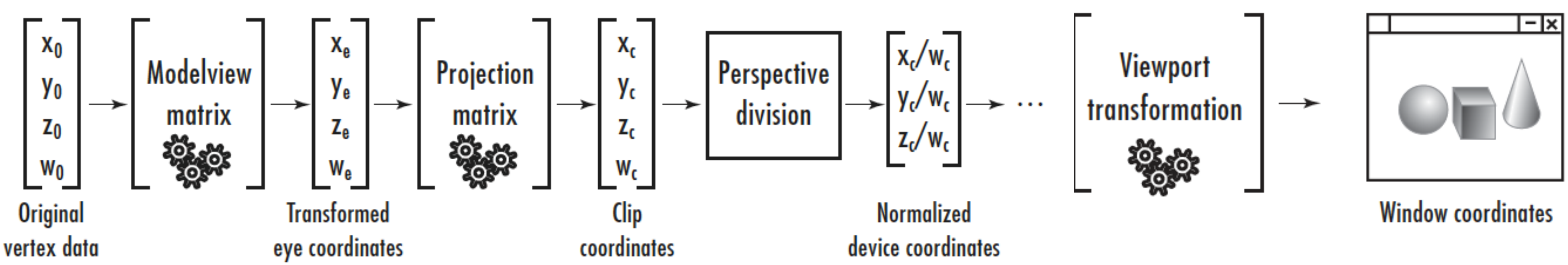
---

2. The vertex is then multiplied by the modelview matrix, which yields the transformed eye coordinates.

3. The eye coordinates are then multiplied by the projection matrix to yield clip coordinates - eliminating all data outside this clipping space

4. The clip coordinates are then divided by the w coordinate to yield normalized device coordinates. The w value may have been modified by the projection matrix or the modelview matrix, depending on the transformations that occurred

5. The coordinate triplet is mapped to a 2D plane by the viewport transformation.



Specifying the Projection Matrix

```
void retupRC()
{
  //...
  glMatrixMode(GL_PROJECTION);
  glOrtho (-20.0f, 20.0f, -20.0f, 20.0f, -20.0f, 20.0f);
  //...
}
```

- “Load” the projection matrix by setting the MatrixMode to PROJECTION
- Specify Orthographic projection parameters

The Modelview Matrix

- A a 4×4 matrix that represents the transformed coordinate system you are using to place and orient your objects.

- The vertices you provide for your primitives are used as a single-column matrix and multiplied by the modelview matrix to yield new transformed coordinates in relation to the eye coordinate system.

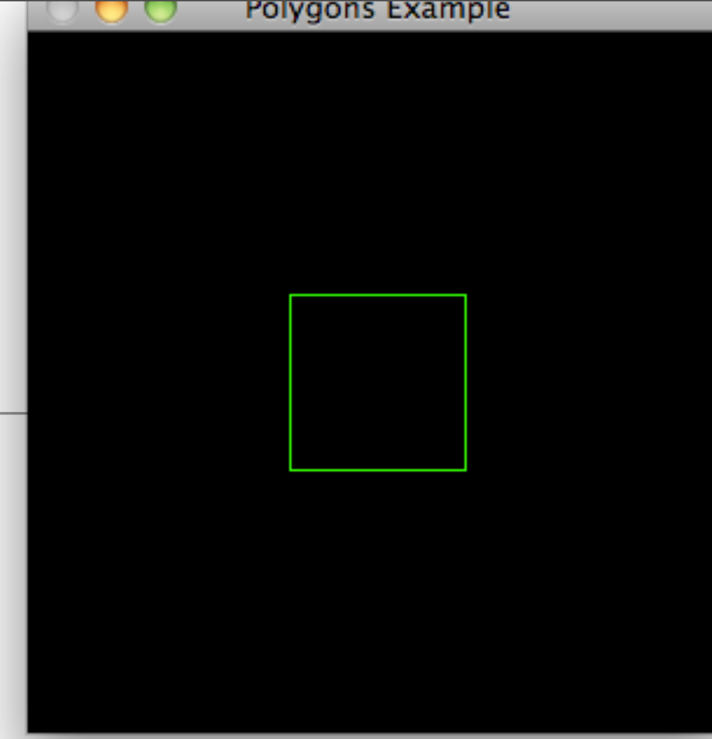
- E.g. A matrix containing data for a single vertex is multiplied by the modelview matrix to yield new eye coordinates. The vertex data is actually four elements with an extra value, w, that represents a scaling factor. This value is set by default to 1.0

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \begin{bmatrix} 4 \times 4 \\ M \end{bmatrix} = \begin{bmatrix} X_e \\ Y_e \\ Z_e \\ W_e \end{bmatrix}$$

Modelview Transformations

- Translation
- Rotation
- Scaling

Translation



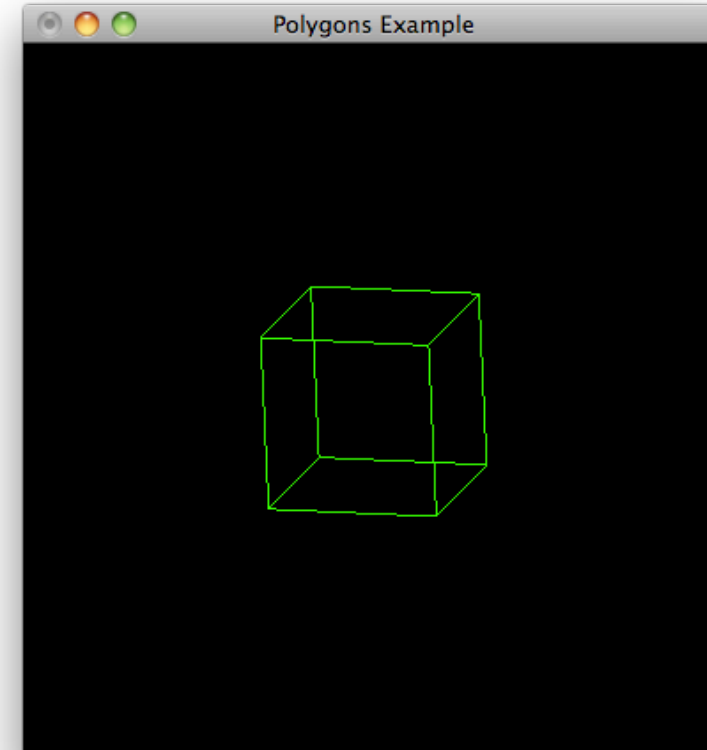
- Draw a cube using the GLUT library's glutWireCubefunction.

- A cube that measures 50 units on a side is then centered at the origin

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glutWireCube(50.0f);

  glutSwapBuffers();
}
```

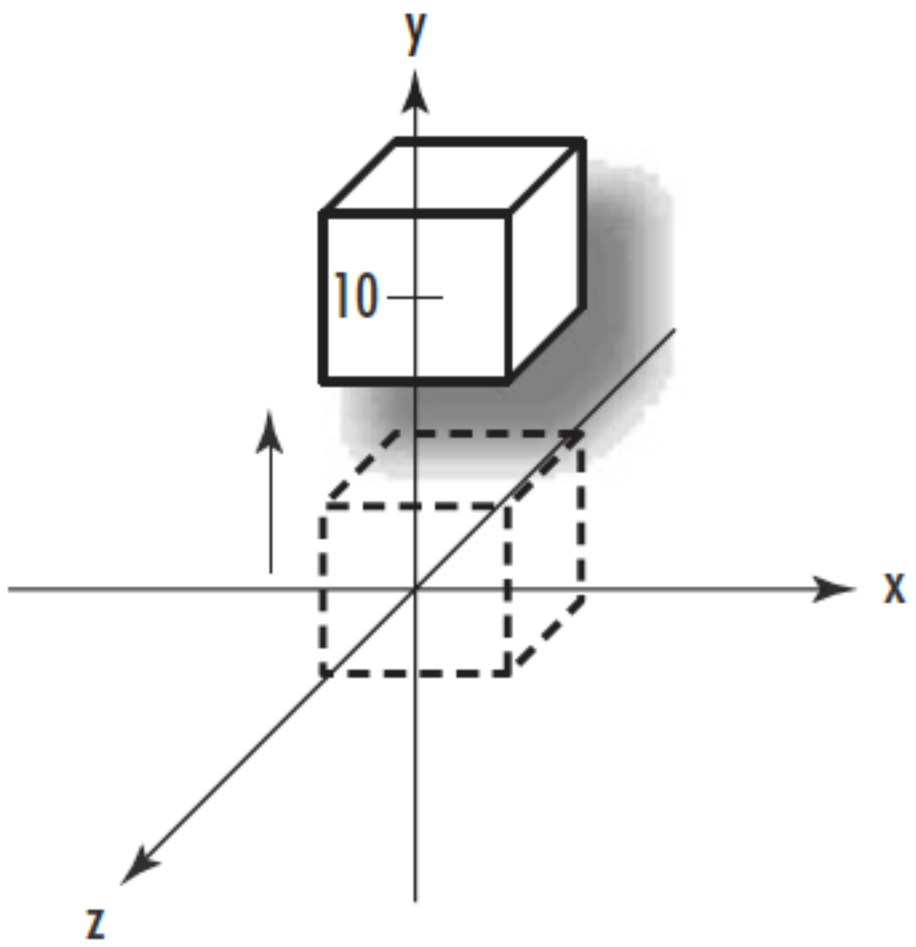


Move the Cube

- To move the cube up the y-axis by 10 units before drawing it,
    - multiply the modelview matrix by a matrix that describes a translation of 10 units up the y-axis
    - then do the drawing.

```
 // Construct a translation matrix for positive 10 Y
 ...
 // Multiply it by the modelview matrix
 ...
 // Draw the cube
glutWireCube(50.0f);
```

Translation in OpenGL



- OpenGL provides a high-level function that performs this task.

```
void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
```

- Takes as parameters the amount to translate along the x, y, and z directions.

- Constructs an appropriate matrix and multiplies it onto the current matrix stack

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glTranslatef(0.0f, 10.0f, 0.0f);
  glutWireCube(50.0f);

  glutSwapBuffers();
}
```

Rotation

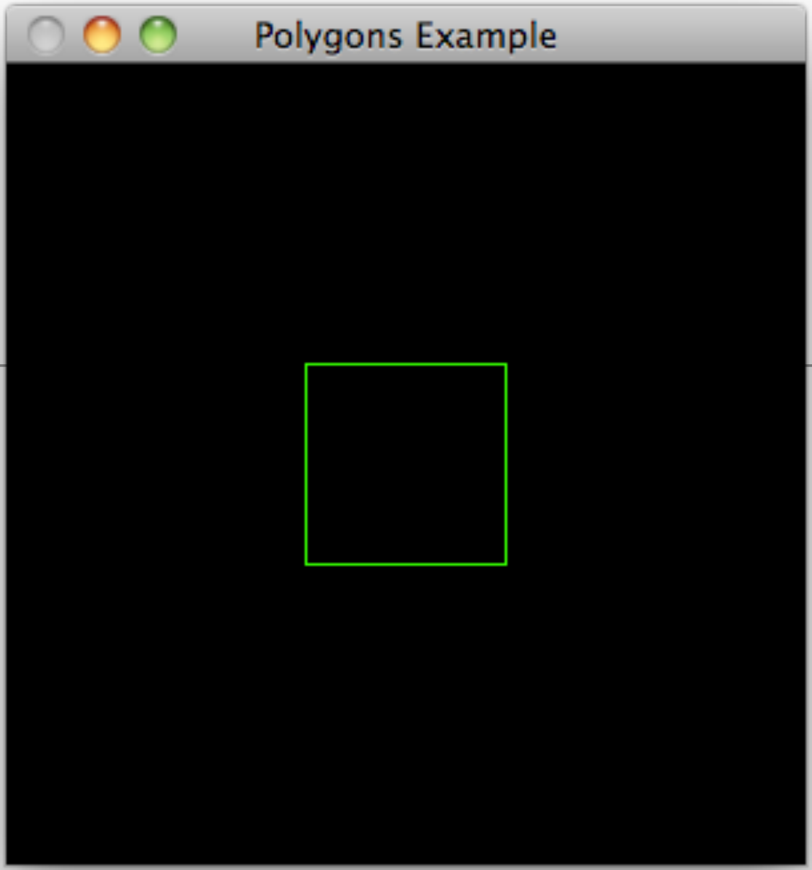
- To rotate an object about one of the three coordinate axes you have to devise a rotation matrix.

```
glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);
```

- Performs a rotation around the vector specified by the x,y, and z arguments.

- The angle of rotation is in the counterclockwise direction measured in degrees and specified by the argument angle.

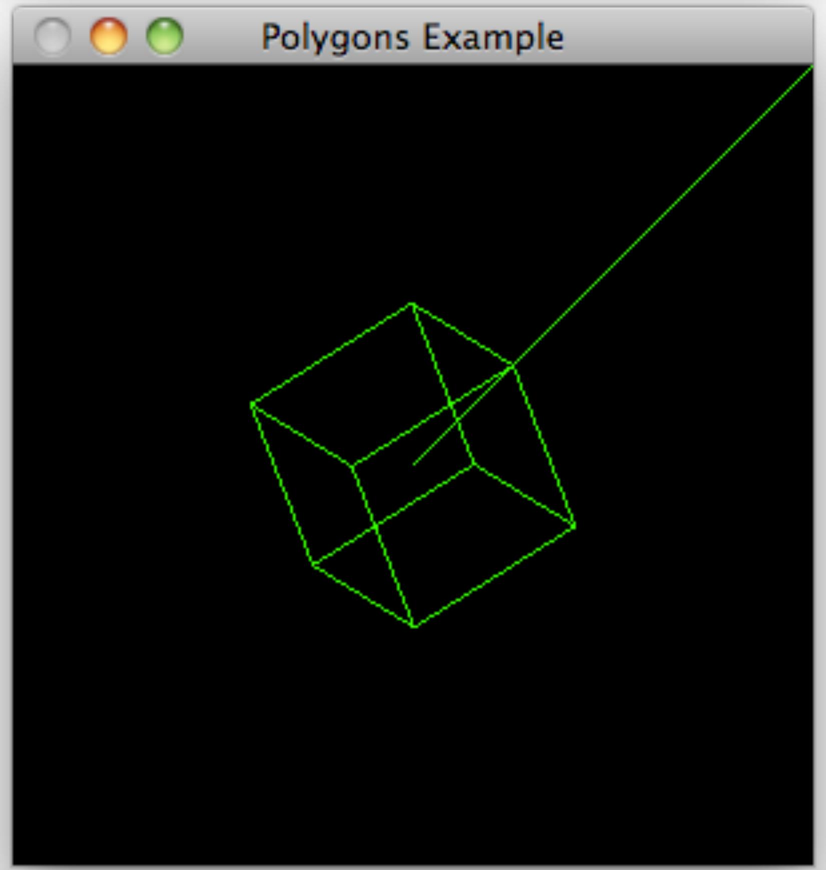
- To see the axis of rotation, you can just draw a line from the origin to the point represented by (x,y,z).




```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glBegin(GL_LINES);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(100.0f, 100.0f, 100.0f);
  glEnd();
  glRotatef(45.0f, 1.0f, 1.0f, 1.0f);
  glutWireCube(50.0f);

 glutSwapBuffers();
}
```



Scaling

- Changes the size of your object by expanding or contracting all the vertices along the three axes by the factors specified

```
glScalef(GLfloat x, GLfloat y, GLfloat z);
```

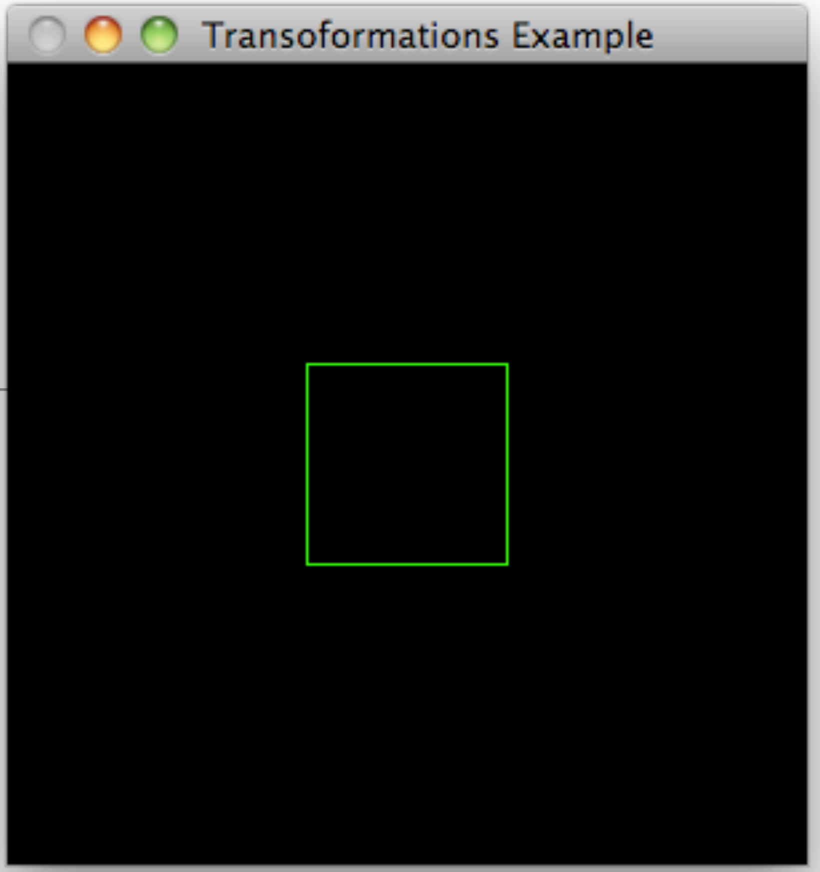
- The function multiplies the x,y, and z values by the scaling factors specified.

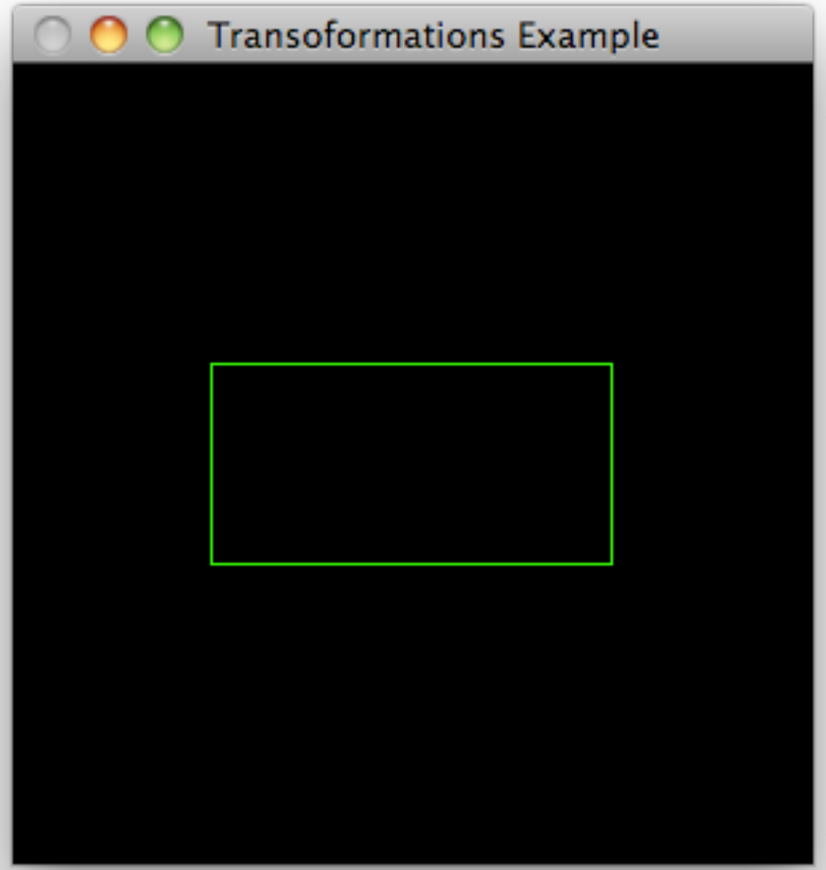
- E.g Produces a cube that is twice as large along the x- and z-axes, but still the same along the y-axis

```
void renderScene(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

  glScalef(2.0f, 1.0f, 2.0f);
  glutWireCube(50.0f);

   glutSwapBuffers();
}
```





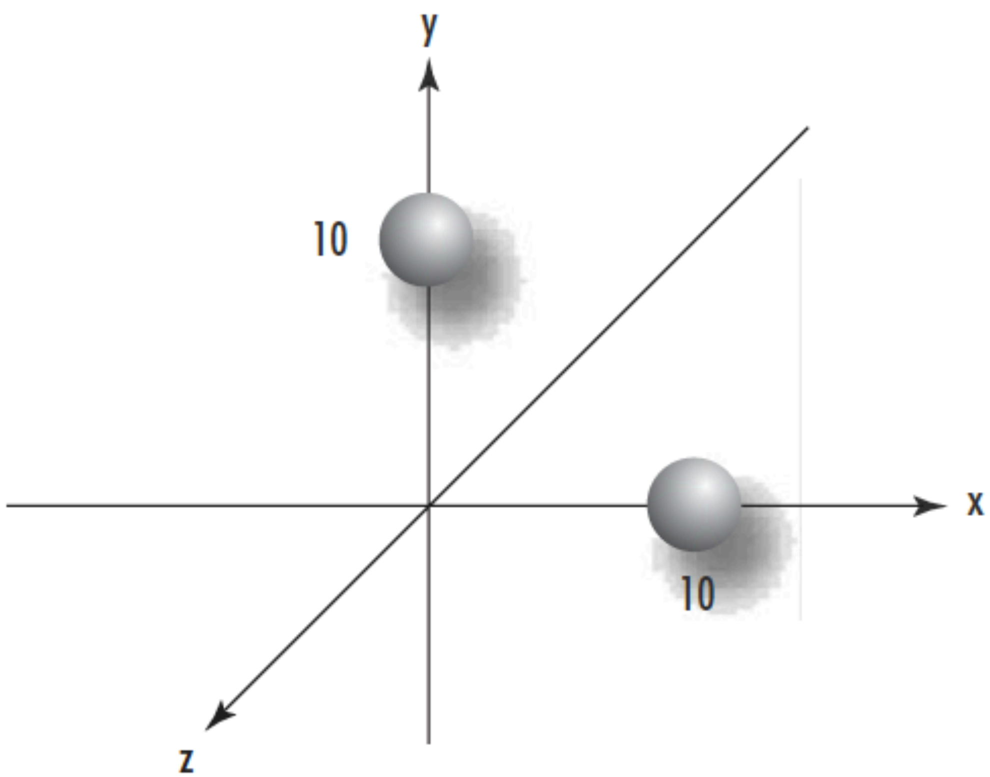
Current Modelview Matrix

- For each of these transformations, the appropriate matrix is constructed and multiplied by the current modelview matrix.

- The new matrix then becomes the current modelview matrix, which is then multiplied by the next transformation, and so on.

- Eg. Draw two spheres—one 10 units up the positive y-axis and one 10 units out the positive x-axis:

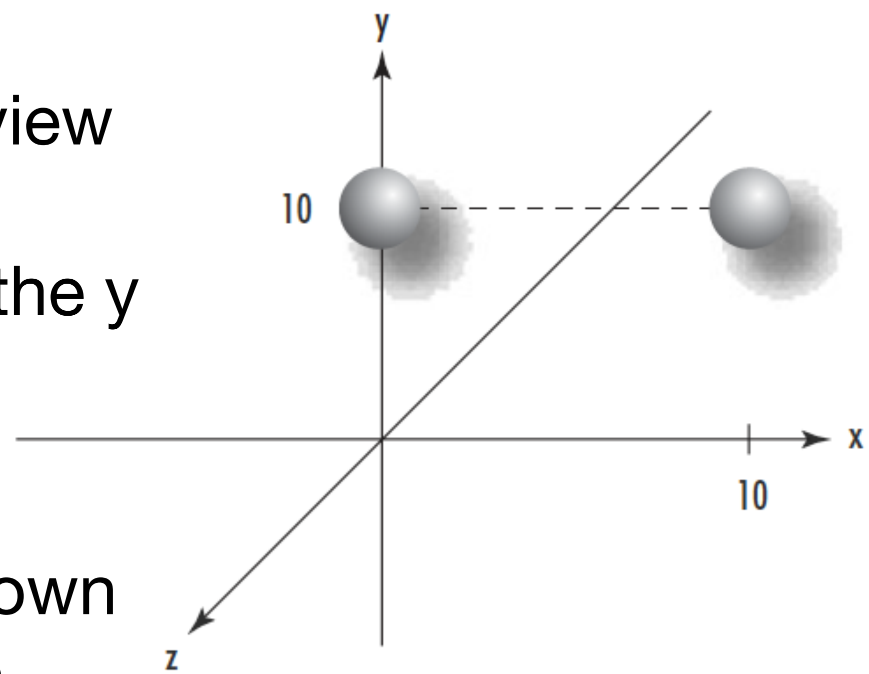
```
glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
```



Cumulative Translations

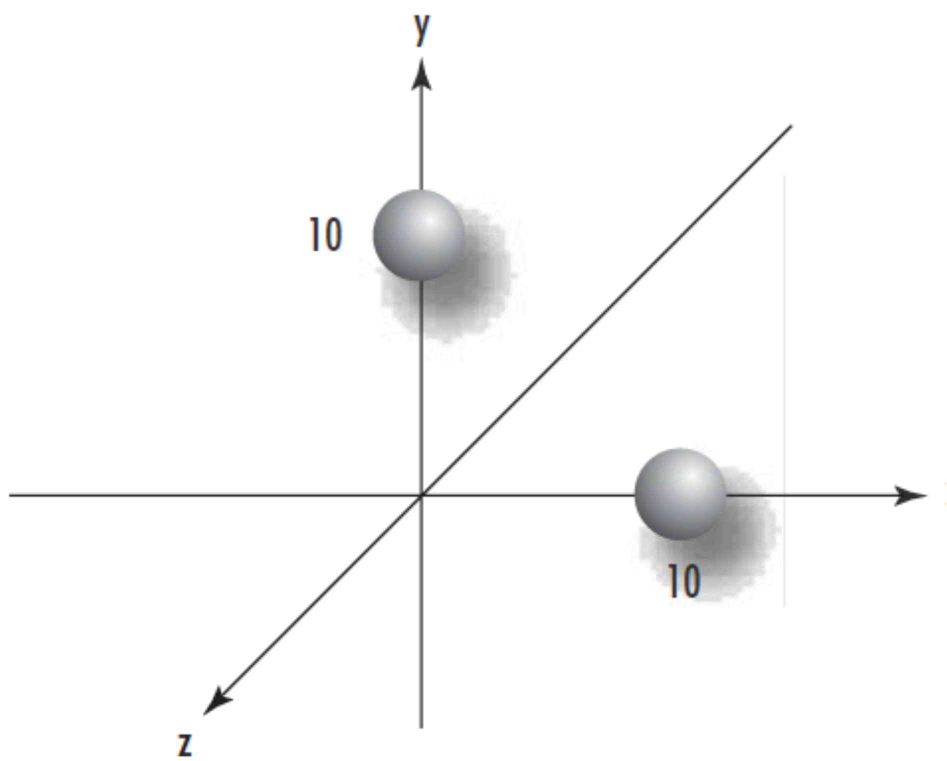
```
glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
```

- Each call to glTranslate is cumulative on the modelview matrix, so the second call translates 10 units in the positive x direction from the previous translation in the y direction.



- You can make an extra call to glTranslate to back down the y-axis 10 units in the negative direction, but this makes some complex scenes difficult to code and debug

```
glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
glTranslatef(0.0f, -10.0f, 0.0f);
glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
```



Identity Matrix

- Reset the origin by loading the modelview matrix with the identity matrix.
- The identity matrix specifies that no transformation is to occur, in effect saying that all the coordinates you specify when drawing are in eye coordinates.
- An identity matrix contains all 0s, with the exception of a diagonal row of 1s. When this matrix is multiplied by any vertex matrix, the result is that the vertex matrix is unchanged.

$$\begin{bmatrix} 8.0 \\ 4.5 \\ -2.0 \\ 1.0 \end{bmatrix} \begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 \\ 0 & 0 & 1.0 & 0 \\ 0 & 0 & 0 & 1.0 \end{bmatrix} = \begin{bmatrix} 8.0 \\ 4.5 \\ -2.0 \\ 1.0 \end{bmatrix}$$

- Loading the Identity Matrix is resetting the modelview matrix to the origin.

glMatrixMode & glLoadIdentity

- The first line specifies that the current operating matrix is the modelview matrix - this remains the active matrix until you change it.

- The second line loads the current matrix (in this case, the modelview matrix) with the identity matrix.

- The second call the glLoadIdentity() reset the modelview matrix again, to the final two calls operate with respect to the origin.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

glTranslatef(0.0f, 10.0f, 0.0f);
glutSolidSphere(1.0f,15,15);

glLoadIdentity();

glTranslatef(10.0f, 0.0f, 0.0f);
glutSolidSphere(1.0f,15,15);
```