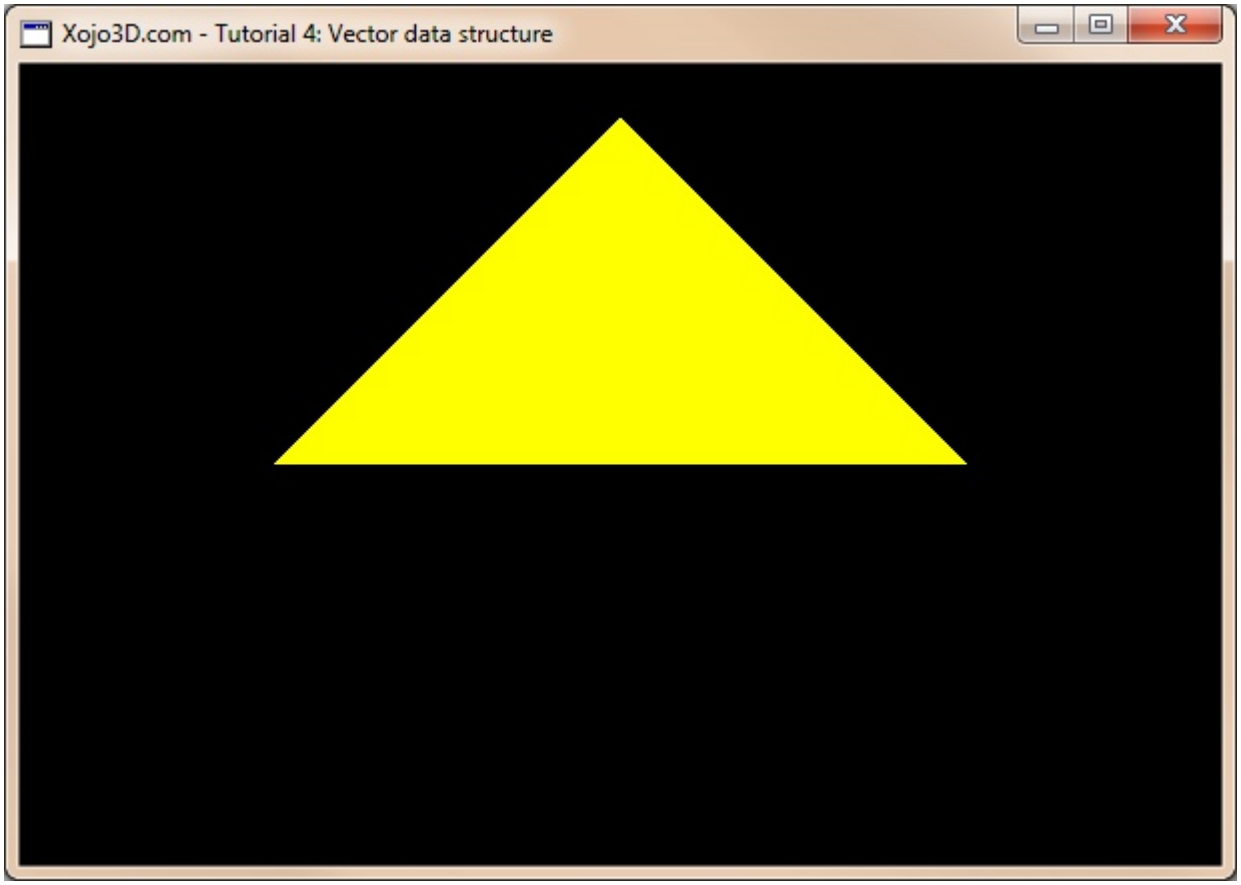
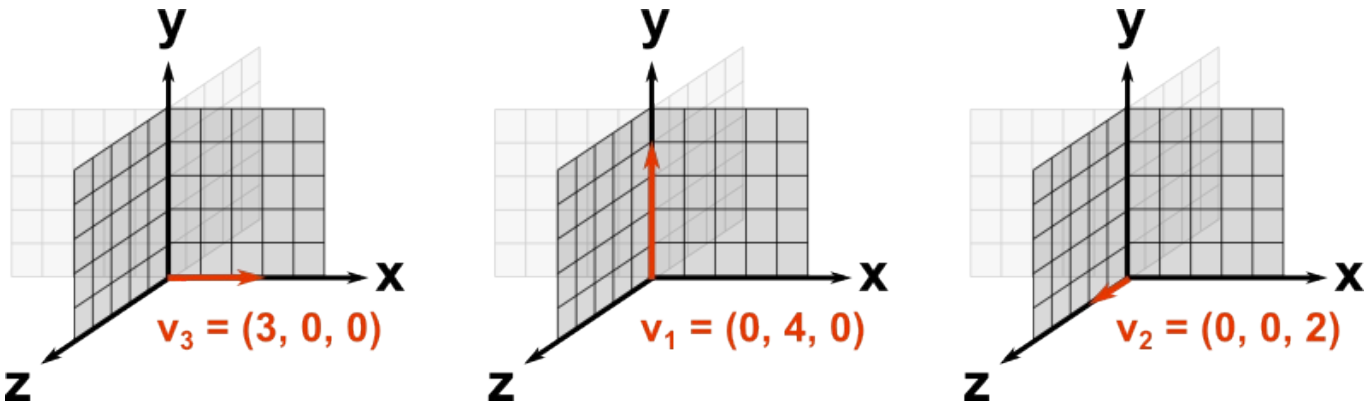
Tutorial 4: Vector data structure

Vectors are the foundation for 3D graphics and animation. In this tutorial we design, and learn how to use, a 3D vector class to represent the vertices of a polygon.



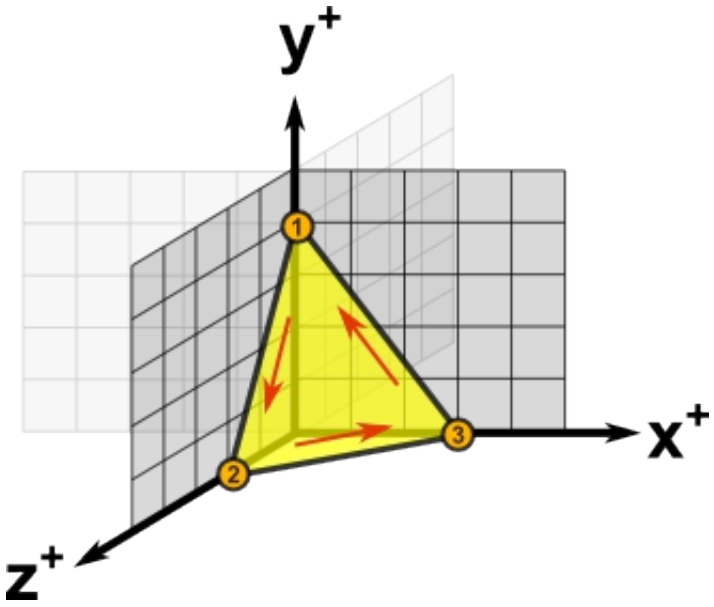
Theory

A **vector** is a quantity that has both magnitude (size) and direction. In 3D graphics a vector is represented by three values known as **X**, **Y** and **Z**.



In the illustrations above, it is easy to see how X, Y and Z values are used to specify the end point of a vector. The starting points of all the vectors are at the origin (point in 3D space where X, Y and Z is zero).

Building a polygon from vectors is simply a matter of storing the required vectors in a sequential list. If we use the vectors above to build the list [v1, v2, v3], then the final polygon will look as follow:



Our vector data structure is a class that has properties to store the X, Y and Z values of a vector.

Tutorial Steps

1. Open Xojo.
2. In the Project Chooser select Desktop.
3. Enter "Tutorial004" as the Application Name, and click OK.
4. Save your project.
5. Configure the following controls:

Control	Name	DoubleBuffer	Left	Top	Maximize Button
Window	SurfaceWindow	-	-	-	ON
OpenGLSurface	Surface	ON	0	0	-

6. Position and size *Surface* to fill the whole window, and set its locking to left, top, bottom and right.

7. Add the following code to the *SurfaceWindow.Open* event handler:

```
Self.MouseCursor = System.Cursors.StandardPointer
```

8. Add the following code to the *SurfaceWindow.Paint* event handler:

```
Surface.Render
```

9. Import the X3Core module, created in the previous tutorial.
   You can download the module from http://www.xojo3d.com/tutorials/tut004/x3core.zip.

10. Add the following code to the *Surface.Resized* event handler:

```
X3_SetPerspective Surface
```

11. Add a new class named "X3Vector" to module *X3Core*.
12. Add the following properties to *X3Vector*:

Name	Type
X	Double
Y	Double
Z	Double

13. Add the following method to class *X3Vector*:

```
Sub Constructor(initX As Double, initY As Double, initZ As Double)
  X = initX
  Y = initY
  Z = initZ
End Sub
```

14. Add the following code to the *Surface.Render* event handler:

```
Dim i As Integer
Dim vertex() As X3Core.X3Vector

vertex.Append new X3Core.X3Vector(0, 1, 0)
vertex.Append new X3Core.X3Vector(-1, 0, 0)
vertex.Append new X3Core.X3Vector(1, 0, 0)

OpenGL.glPushMatrix

OpenGL.glClearColor(0, 0, 0, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT)

OpenGL.glTranslatef 0.0, 0.0, -2.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

OpenGL.glColor3d(1, 1, 0)
for i = 0 to vertex.Ubound
  OpenGL.glVertex3d vertex(i).X, vertex(i).Y, vertex(i).Z
next i

OpenGL.glEnd

OpenGL.glPopMatrix
```

15. Save and run your project.

Analysis

The three properties added to the new X3Vector class is used to store the X, Y and Z values of the vector. These properties are used to set up the positions of the vertices of a polygon.

X3Vector.Constructor:

```
Sub Constructor(initX As Double, initY As Double, initZ As Double)
  X = initX
  Y = initY
  Z = initZ
End Sub
```

The constructor method makes it easy to instantiate a new vector with predefined values. We simply set the X, Y and Z properties of the vector equal to the given values.

Surface.Render:

```
Dim i As Integer
Dim vertex() As X3Core.X3Vector

vertex.Append new X3Core.X3Vector(0, 1, 0)
vertex.Append new X3Core.X3Vector(-1, 0, 0)
vertex.Append new X3Core.X3Vector(1, 0, 0)

OpenGL.glPushMatrix

OpenGL.glClearColor(0, 0, 0, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT)

OpenGL.glTranslatef 0.0, 0.0, -2.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

OpenGL.glColor3d(1, 1, 0)
for i = 0 to vertex.Ubound
  OpenGL.glVertex3d vertex(i).X, vertex(i).Y, vertex(i).Z
next i

OpenGL.glEnd

OpenGL.glPopMatrix
```

The vertex() array is used to store a list of X3Vector objects that represent the vertices of our polygon.

We use the vertex.Append instruction to append new vector objects to the array. Note how we use the constructor of X3Vector to initialize each vector with its required values.

Drawing the polygon is very similar to how we did it in the previous tutorials. The only difference is that we now use a "for" loop to iterate through the vertices stored in our vector array. The advantage of this method is that we can now add as many triangular polygons as we need to our array, without having to change our rendering code, e.g. if we want to add a second polygon, we simply add three more vector objects to our existing vertex array.