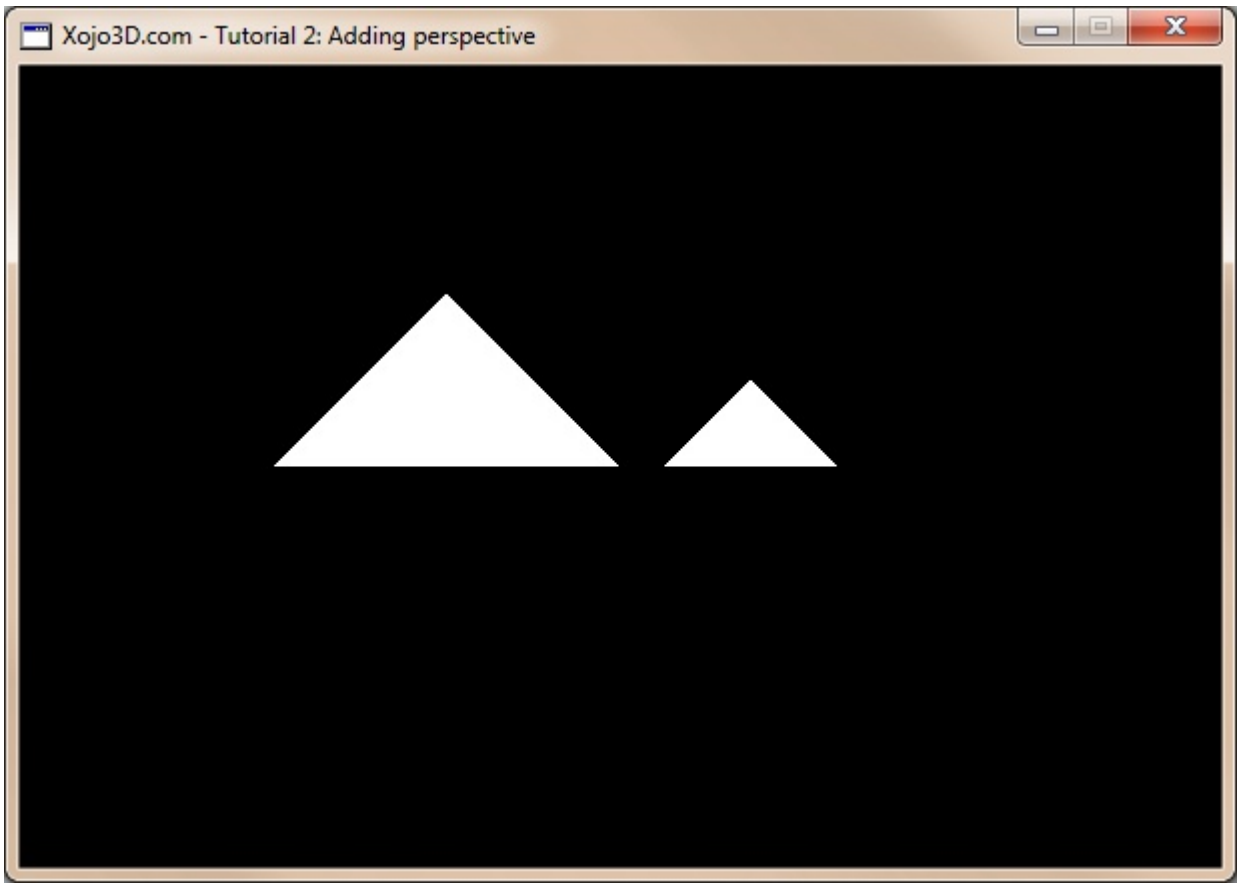
Tutorial 2: Perspective projection

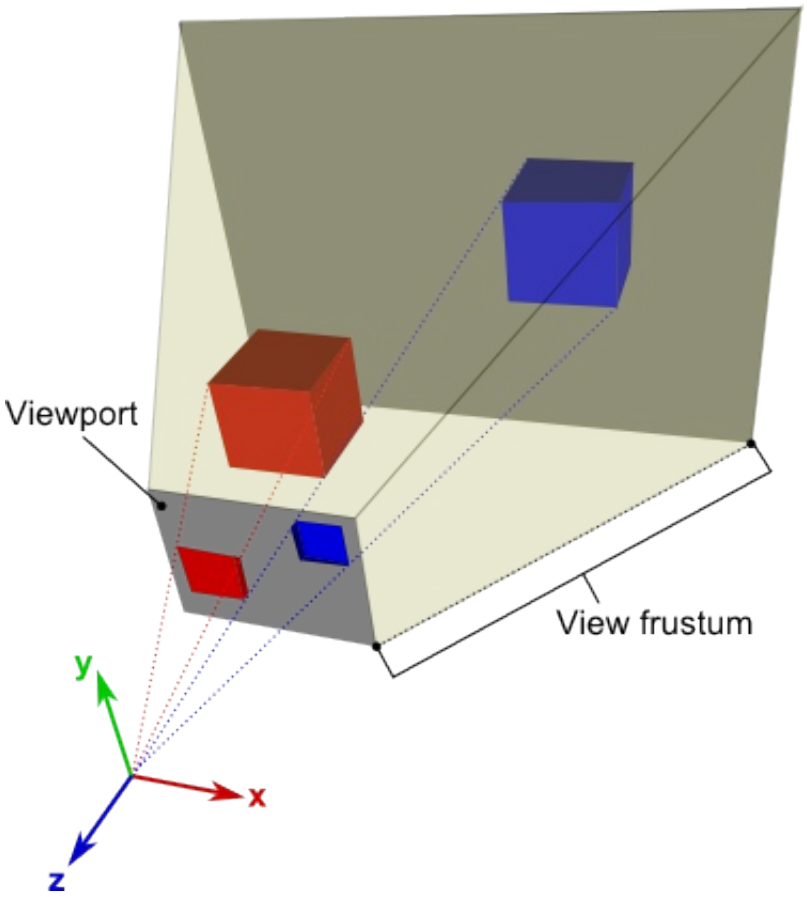
Drawing polygons with proper perspective settings makes your scenes look more realistic. In this tutorial you will learn about the basics of perspective projection.



Theory

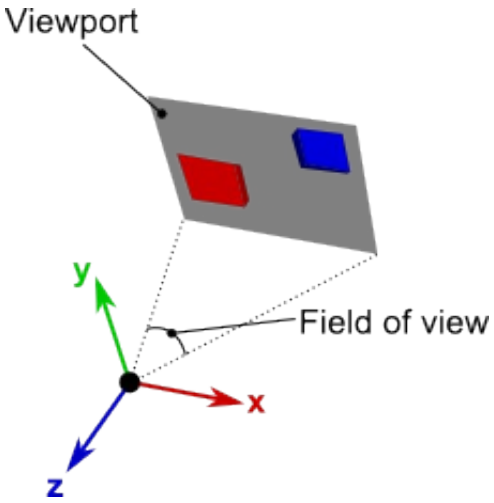
Perspective projection is concerned with transforming 3D points into 2D points, so that when the scene is rendered onto a 2D output device, objects closer to the viewer appears larger than objects in the distance.

A group of very smart computer scientists already took the time to design mathematical algorithms for perspective projection. All we need to do, is to set up our desired projection settings and OpenGL will do the heavy lifting for us.



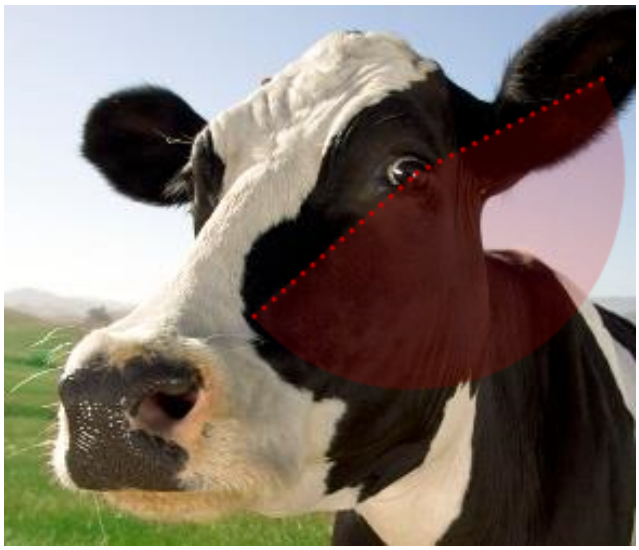
The **viewport** is the surface onto which our scene is projected (e.g. computer screen). The **view frustum** is the region of space in the virtual world that is projected onto the viewport.

Polygons rotate freely in 3D space (e.g., they can face towards you or face away from you). We therefore need to define the vertices in such a way that the computer knows which way the polygon is facing. The vertices of a polygon are always defined in an **anti-clockwise** order.



Field of view, or **fov**, is the angle that determines the extent of a scene that is projected onto the viewport.

In the real world, predators such as lions, whose eyes are facing forward, have a smaller field of view than grazers such as cows, whose eyes are on the side of their heads. The large fov that grazers have, helps them to spot predators in the wild.



Tutorial Steps

1. Open Xojo.
2. In the Project Chooser select Desktop.
3. Enter "Tutorial002" 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. Add the following code to the *Surface.Render* event handler:

```
OpenGL.glPushMatrix

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

OpenGL.glTranslatef 0.0, 0.0, -5.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

OpenGL.glVertex3d -1, 1, 1
OpenGL.glVertex3d -2, 0, 1
OpenGL.glVertex3d 0, 0, 1

// continue on next page
```

```
// continued from previous page

OpenGL.glVertex3d 1.5, 1, -3
OpenGL.glVertex3d 0.5, 0, -3
OpenGL.glVertex3d 2.5, 0, -3

OpenGL.glEnd

OpenGL.glPopMatrix
```

10. Add a new module named "X3Core" to your project.
11. Add the following method to module *X3Core*:

```
Sub X3_SetPerspective(Surface As OpenGLSurface,
                      fov As Double = 60.0,
                      zNear As Double = 1,
                      zFar As Double = 100)
  OpenGL.glViewport 0, 0, Surface.Width, Surface.Height

  OpenGL.glMatrixMode OpenGL.GL_PROJECTION
  OpenGL.glLoadIdentity
  OpenGL.gluPerspective fov, Surface.Width/Surface.Height, zNear, zFar

  OpenGL.glMatrixMode OpenGL.GL_MODELVIEW
  OpenGL.glLoadIdentity
End Sub
```

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

```
X3_SetPerspective Surface
```

13. Save and run your project.

Analysis

X3Core.X3_SetPerspective:

```
Sub X3_SetPerspective(Surface As OpenGLSurface,
                      fov As Double = 60.0,
                      zNear As Double = 1,
                      zFar As Double = 100)
  OpenGL.glViewport 0, 0, Surface.Width, Surface.Height

  OpenGL.glMatrixMode OpenGL.GL_PROJECTION
  OpenGL.glLoadIdentity
  OpenGL.gluPerspective fov, Surface.Width/Surface.Height, zNear, zFar

  OpenGL.glMatrixMode OpenGL.GL_MODELVIEW
  OpenGL.glLoadIdentity
End Sub
```

X3_SetPerspective is an OpenGL helper method to set up perspective projection settings.

Four parameters are passed to X3_SetPerspective:
Surface: The OpenGL Surface control used for OpenGL drawing.
fov: The field of view to use during rendering.
zNear: The z-value of the view frustum closest to us.
zFar: The z-value of the view frustum farthest from us.

First we use glViewport to set up OpenGL's viewport. Think of the viewport as a window through which you look into your virtual world.

OpenGL uses matrices to render our virtual 3D data onto a 2D surface. The two most common matrices are the GL_PROJECTION matrix and the GL_MODELVIEW matrix. The GL_PROJECTION matrix stores the projection settings used by our camera, while the GL_MODELVIEW matrix is used during rendering to translate, rotate and scale models.

We switch to the GL_PROJECTION matrix by calling glMatrixMode with the GL_PROJECTION constant. glLoadIdentity is then used to clear the current projection matrix so that we can set up our own custom projection settings.

Surface.Render:

```
OpenGL.glPushMatrix

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

OpenGL.glTranslatef 0.0, 0.0, -5.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

OpenGL.glVertex3d -1, 1, 1
OpenGL.glVertex3d -2, 0, 1
OpenGL.glVertex3d 0, 0, 1

OpenGL.glVertex3d 1.5, 1, -3
OpenGL.glVertex3d 0.5, 0, -3
OpenGL.glVertex3d 2.5, 0, -3

OpenGL.glEnd

OpenGL.glPopMatrix
```

glClearColor sets the color to use when clearing the OpenGL surface background. The four parameters, red, green, blue and alpha, makes up an RGBA color value. These values are floating point values in the ranging from 0 to 1.

Surface.Resized:

```
X3_SetPerspective Surface
```

Each time our OpenGL surface is resized, we update the perspective projection settings with our new X3_SetPerspective method. This ensures that our scences are rendered correctly, regardless of how the surface is resized.