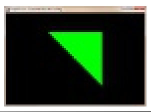
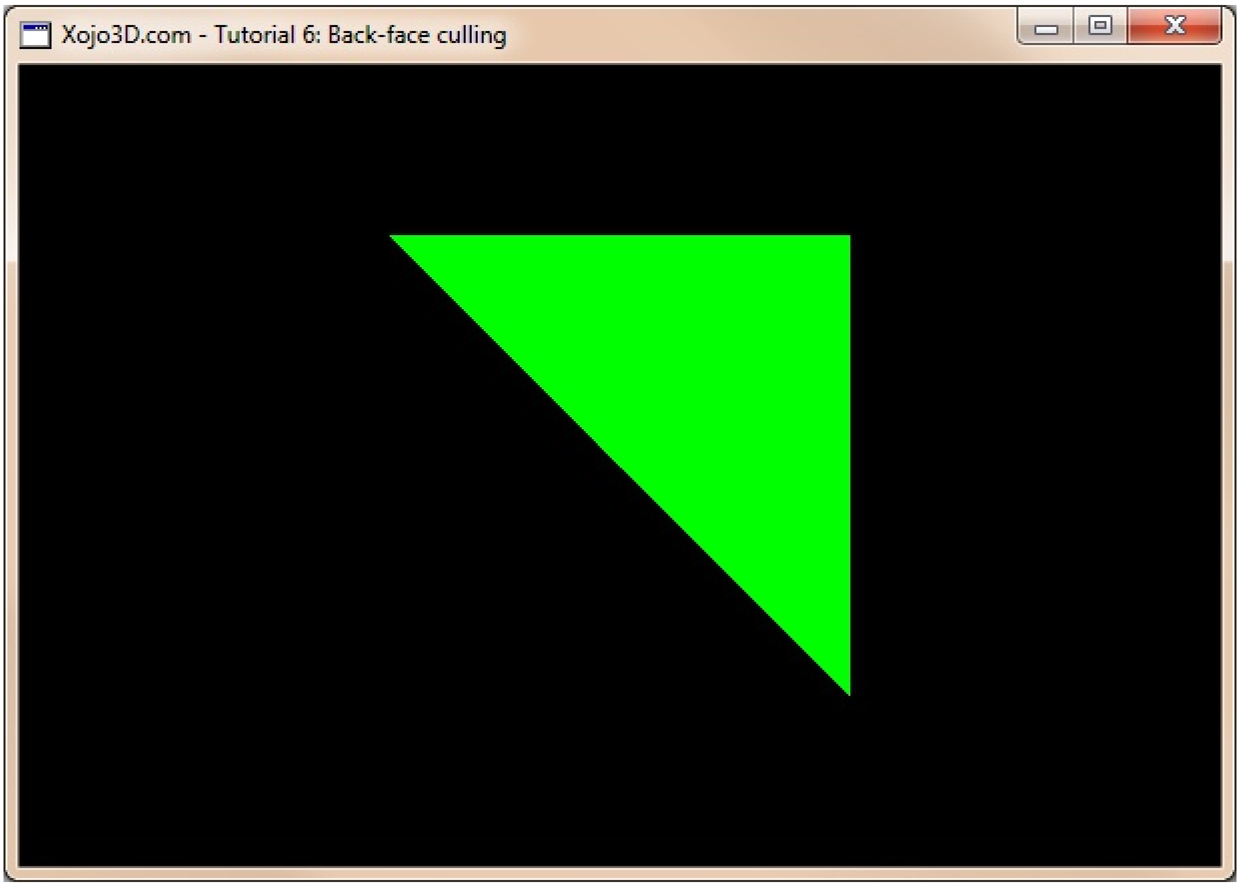
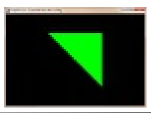


Tutorial 6: Back-face culling

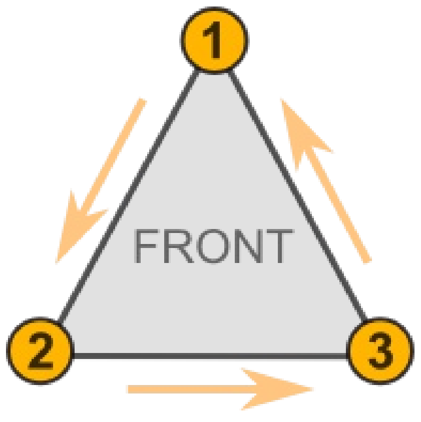
Back-face culling is a technique used to be more resourceful with your available processing power. In this tutorial you will learn how to activate back-face culling in OpenGL.



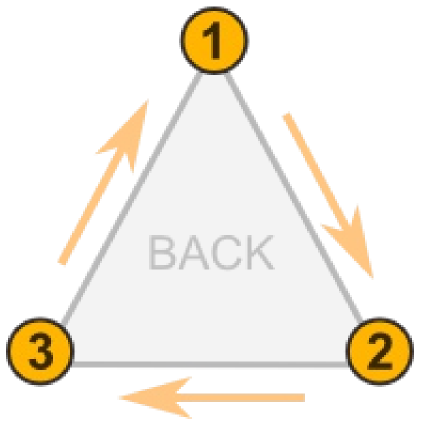


Theory

The vertices of a polygon is always defined in an **anti-clockwise** order, so that the computer knows which way the polygon is facing.



Front View



Back View

From the above illustrations it is easy to see that when OpenGL renders a collection of polygons, all the polygons facing away from us can be ignored, since their front surface is not visible from where we are standing. The process of skipping over polygons that are facing away from us, to save processing time, is better known as **back-face culling**.

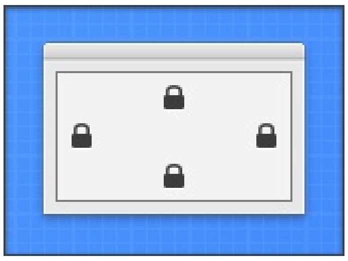
OpenGL has built-in support for back-face culling, so it is simply a matter of enabling back-face culling during our initialization routine.

Tutorial Steps

1. Open Xojo.
2. In the Project Chooser select Desktop.
3. Enter "Tutorial006" 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/tut006/x3core.zip.

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

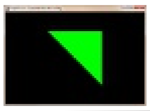
```
X3_SetPerspective Surface
```

11. Add the following method to module *X3Core*:

```
Sub X3_Initialize()
  OpenGL.glCullFace OpenGL.GL_BACK
  OpenGL.glEnable OpenGL.GL_CULL_FACE
End Sub
```

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

```
X3_Initialize
```



13. Add the following code to the Surface.Render event handler:

```
Dim i, j As Integer
Dim polygon() As X3Core.X3Polygon
Dim poly As X3Core.X3Polygon

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(0, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, -1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, 1, 0)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, -1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, -1, 0)
polygon.Append poly

OpenGL.glPushMatrix

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

OpenGL.glTranslatef 0.0, 0.0, -3.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to polygon.Ubound

  poly = polygon(i)

  if poly.FillColor <> nil then
    OpenGL.glColor3d(poly.FillColor.Red, poly.FillColor.Green,
poly.FillColor.Blue)
  else
    OpenGL.glColor3d(1, 1, 1)
  end if

// continue on next page
// continued from previous page

for j = 0 to poly.Vertex.Ubound
    OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
poly.Vertex(j).Z
  next j

next i

OpenGL.glEnd

OpenGL.glPopMatrix
```

14. Save and run your project.

Analysis

X3Core.X3_Initialize:

```
Sub X3_Initialize()
  OpenGL.glCullFace OpenGL.GL_BACK
  OpenGL.glEnable OpenGL.GL_CULL_FACE
End Sub
```

Enabling back-face culling is common to almost every OpenGL application, so we've created an initialization method that can be used in any OpenGL project.

First we set the culling mode to back-facing polygons with a call to glCullFace. We then enable face culling with a call to glEnable, passing the GL_CULL_FACE constant as a parameter. All polygons facing away from us will now be ignored by the OpenGL rendering routines.

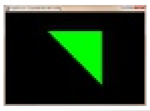
Surface.Open:

```
X3_Initialize
```

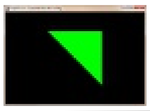
A call to X3_Initialize, in the Open event handler of the OpenGL surface, enables back-face culling.

Surface.Render:

```
Dim i, j As Integer
Dim polygon() As X3Core.X3Polygon
Dim poly As X3Core.X3Polygon

// continue on next page
```



```
poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(0, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, -1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, 1, 0)
polygon.Append poly

poly = new X3Core.X3Polygon()
poly.FillColor = new X3Core.X3Color(1, 0, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, 1, 0)
poly.Vertex.Append new X3Core.X3Vector(1, -1, 0)
poly.Vertex.Append new X3Core.X3Vector(-1, -1, 0)
polygon.Append poly

OpenGL.glPushMatrix

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

OpenGL.glTranslatef 0.0, 0.0, -3.0

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to polygon.Ubound

  poly = polygon(i)

  if poly.FillColor <> nil then
    OpenGL.glColor3d(poly.FillColor.Red, poly.FillColor.Green,
poly.FillColor.Blue)
  else
    OpenGL.glColor3d(1, 1, 1)
  end if

  for j = 0 to poly.Vertex.Ubound
    OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
poly.Vertex(j).Z
  next j

next i

OpenGL.glEnd

OpenGL.glPopMatrix
```



Notice how we defined two polygons in our rendering method, yet only one polygon is displayed when we run the program. This is because the second (red) polygon is facing away from us. Because we enabled back-face culling, the red polygon is ignored by OpenGL. To test the effect of back-face culling, comment out the X3_Initialize call in Surface.Open and run the program again. You will now see both polygons.

IMPORTANT: The vertices of a polygon is always defined in an anti-clockwise order, so that OpenGL knows which way the polygon is facing.