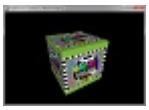
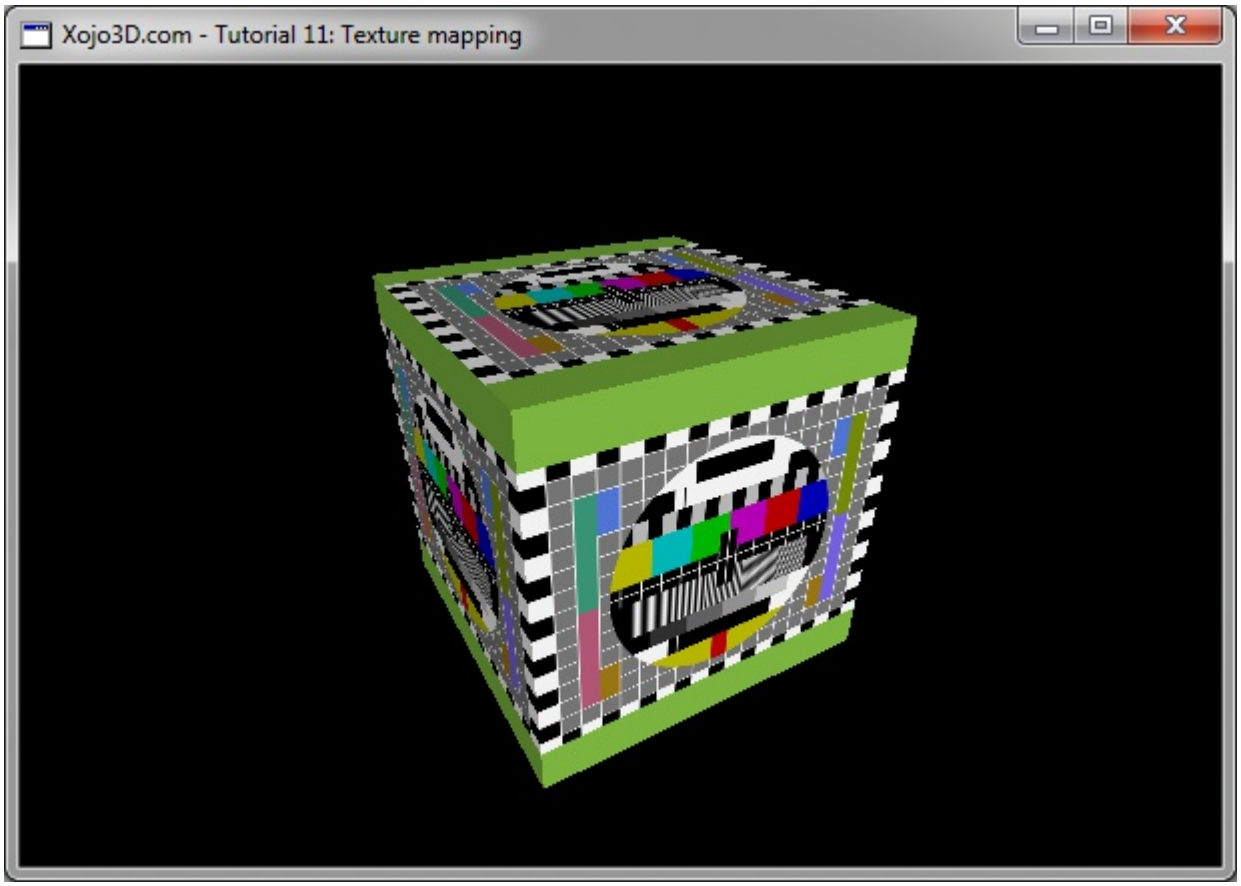


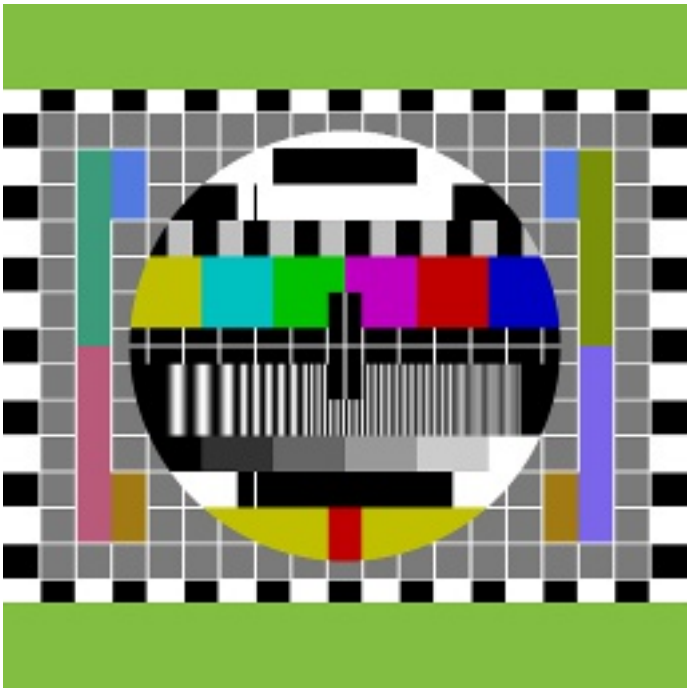
Tutorial 11: Texture mapping

Mapping textures to polygons is a technique that gives 3D models a realistic look, by making use of 2D images. Learn the basics of texture mapping with this tutorial.



Theory

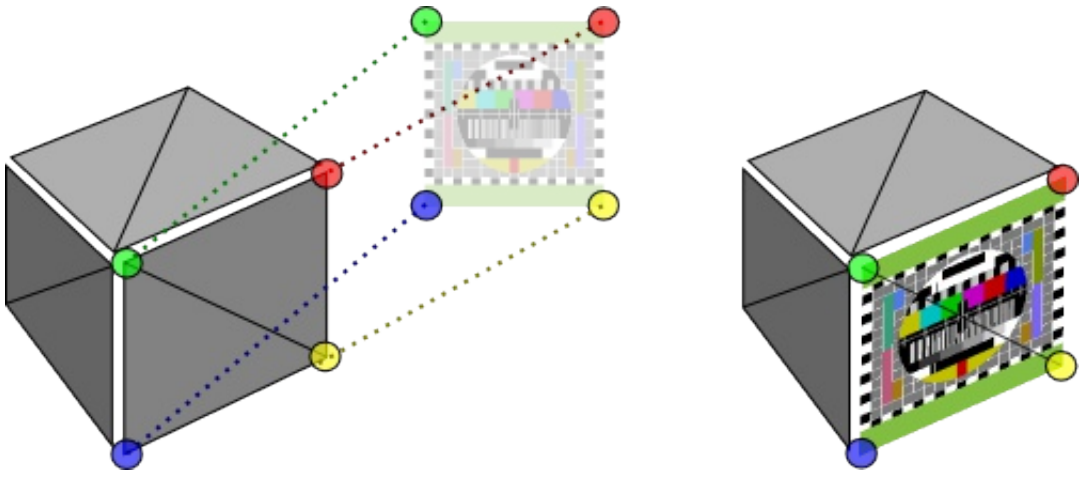
Texture mapping is a technique used to "paint" an image, or part thereof, onto a polygon. Consider the following image that could potentially be used as a texture bitmap.

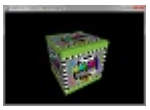


The PNG file format is the recommended format for storing texture bitmaps. PNG supports loss-less data compression (high quality images) and has full alpha channel support (transparency).

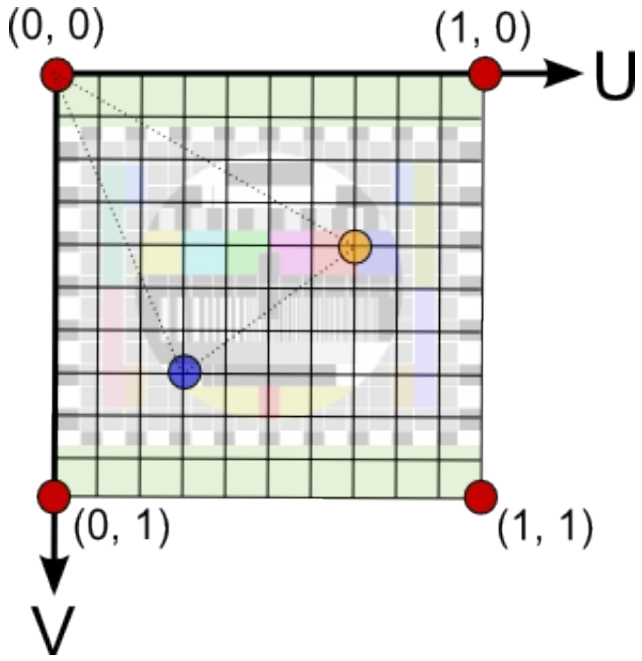
It is best practice to ensure that the dimensions of your textures are power-of-two (e.g. 32 by 32 pixels or 128 by 128 pixels). Power-of-two dimensions ensure backward compatibility with older hardware and in certain cases improve the rendering speed.

The mapping of texture bitmaps onto polygons is a very straightforward process. We simply need to match the coordinates on a texture bitmap to the vertices of a polygon. OpenGL will automatically handle the drawing of the texture onto the polygon. The image below illustrates how the coordinates of a texture bitmap might be matched with the vertices of polygons.

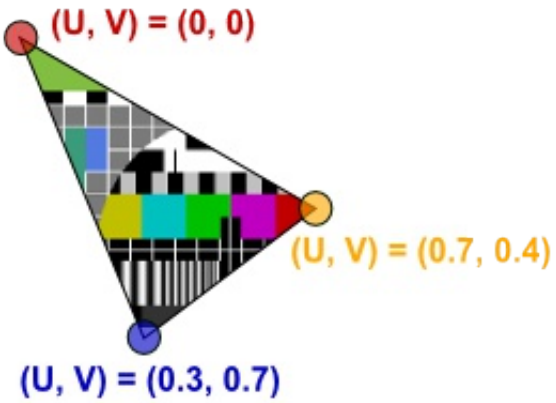


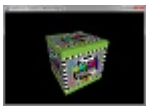


The diagram below illustrates the UV-coordinate system, or **UV Map**, that we use when mapping texture bitmaps to polygons.



Using the above coordinate system as a reference, we can now easily map a texture, or part thereof, to a polygon as shown below.





Tutorial Steps

1. Open Xojo.
2. In the Project Chooser select Desktop.
3. Enter "Tutorial011" 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.*Paint event handler:

```
Surface.Render
```

8. Import the X3Core module, created in the previous tutorial.
   You can download the module from http://www.xojo3d.com/tutorials/tut011/x3core.zip.

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

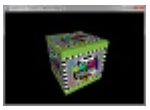
```
X3_Initialize

X3_EnableLight OpenGL.GL_LIGHT0, new X3Core.X3Light(0, 0, 1)
```

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

```
X3_SetPerspective Surface
```

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

```
Function X3_LoadRGBATexture(RGBABitmap As MemoryBlock, width As
Integer, height As Integer) As Integer
  // IMPORTANT: Image dimensions must be in power of 2 (e.g. 8x8,
16x16, 32x32, 64x64, ...)

  Dim idMB As MemoryBlock
  Dim oglName As Integer

// continue on next page
```



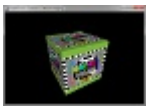
```
// continued from previous page

  idMB = new MemoryBlock(4)
  OpenGL.glGenTextures(1, idMB)
  oglName = idMB.Long(0)

  OpenGL.glBindTexture(OpenGL.GL_TEXTURE_2D, OGLName)

  OpenGL.glTexParameteri(OpenGL.GL_TEXTURE_2D,
     OpenGL.GL_TEXTURE_MIN_FILTER, OpenGL.GL_LINEAR)
  OpenGL.glTexParameteri(OpenGL.GL_TEXTURE_2D,
     OpenGL.GL_TEXTURE_MAG_FILTER, OpenGL.GL_LINEAR)

  OpenGL.glTexImage2d(OpenGL.GL_TEXTURE_2D, 0, 4, width, height,
     0, OpenGL.GL_RGBA, OpenGL.GL_UNSIGNED_BYTE, RGBABitmap)

  return oglName
End Function
```

12. Add a new class named "X3Texture" to module X3Core.
13. Add the following properties to *X3Texture*:

Name	Type
Height	Integer
Width	Integer
OGLName	Integer
RGBABitmap	MemoryBlock

14. Add the following method to *X3Texture*:

```
Sub Constructor(texture As Picture)
  Dim x, y, offset As Integer
  Dim textCol As Color
  Dim textMaskCol As Color
  Dim alpha As Byte

  Width = texture.Width
  Height = texture.Height

  RGBABitmap = new MemoryBlock(Height * Width * 4)

// continue on next page
```

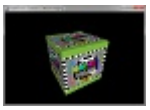


```
// continued from previous page

  offset = 0

  for y = 0 to Height - 1

      for x = 0 to Width - 1

      textCol = texture.RGBSurface.Pixel(x,y)
      textMaskCol = texture.Mask.RGBSurface.Pixel(x, y)

      alpha = 255 - (textMaskCol.Red + textMaskCol.Green +
               textMaskCol.Blue) / 3

      RGBABitmap.Byte(offset) = textCol.Red
      RGBABitmap.Byte(offset + 1) = textCol.Green
      RGBABitmap.Byte(offset + 2) = textCol.Blue
      RGBABitmap.Byte(offset + 3) = alpha

      offset = offset + 4

    next x

  next y

  OGLName = X3_LoadRGBATexture(RGBABitmap, Width, Height)
End Sub
```

15. Add the following method to *X3Texture*:

```
Sub Destructor()
  Dim texturePtr As MemoryBlock

  OpenGL.glFlush

  texturePtr = new MemoryBlock(4)

  texturePtr.Long(0) = OGLName

  OpenGL.glDeleteTextures(1, texturePtr)
End Sub
```

16. Add a new class named "X3UVCoordinate" to module *X3Core*.



17. Add the following properties to X3UVCoordinate:

Name	Type
U	Double
V	Double

18. Add the following method to X3UVCoordinate.

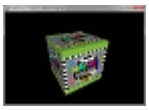
```
Sub Constructor(initU As Double, initV As Double)
  U = initU
  V = initV
End Sub
```

19. Add the following properties to *X3Polygon*:

Name	Type
Texture	X3Texture
UVMap()	X3UVCoordinate

20. Replace the code in the *X3Core.X3_RenderModel* method with the following:

```
Dim i, j As Integer
Dim poly As X3Core.X3Polygon

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to model.Polygon.Ubound

  poly = model.Polygon(i)

  OpenGL.glNormal3d poly.Normal.X, poly.Normal.Y, poly.Normal.Z

  if (poly.Texture <> nil) and (poly.UVMap.Ubound >=
poly.Vertex.Ubound) then

    OpenGL.glColor4d(1, 1, 1, 1)

    OpenGL.glBindTexture(OpenGL.GL_TEXTURE_2D, poly.Texture.OGLName)

    for j = 0 to poly.Vertex.Ubound

// continue on next page
```

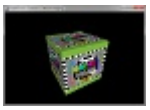


```
  // continued from previous page

        OpenGL.glTexCoord2d poly.UVMap(j).U, poly.UVMap(j).V
        OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
  poly.Vertex(j).Z

      next j

      OpenGL.glBindTexture(OpenGL.GL_TEXTURE_2D, 0)

    else

      if poly.FillColor <> nil then
        OpenGL.glColor4d(poly.FillColor.Red, poly.FillColor.Green,
            poly.FillColor.Blue, poly.FillColor.Alpha)
      else
        OpenGL.glColor4d(1, 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

    end if

    next i

  OpenGL.glEnd
```

21. Import the X3Test module into your project.
    You can download the module from http://www.xojo3d.com/tutorials/tut011/x3test.zip.
22. Add the following property to *SurfaceWindow*:

Name	Type
Model	X3Core.X3Model

23. Download texture.png from the link below and save it next to your project file.
    You can download the module from http://www.xojo3d.com/tutorials/tut011/texture.png.
24. Import the picture into your project and rename it to "imgTexture".



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

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

Model = X3Test_Cube4(imgTexture)
```

26. Add the following code to the end of the *X3Core.X3_Initialize* method:

```
OpenGL.glEnable OpenGL.GL_TEXTURE_2D
```

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

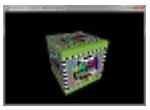
```
OpenGL.glClearColor(0, 0, 0, 1)
OpenGL.glClear(OpenGL.GL_COLOR_BUFFER_BIT +
OpenGL.GL_DEPTH_BUFFER_BIT)

OpenGL.glPushMatrix

OpenGL.glTranslatef 0, 0, -4.0
OpenGL.glRotated(30, 1, 0, 0)
OpenGL.glRotated(30, 0, 1, 0)
X3_RenderModel Model

OpenGL.glPopMatrix
```

28. Save and run your project.

Analysis

The X3Core.X3_LoadRGBATexture helper method is used load RGBA texture bitmaps into OpenGL memory. X3Core.X3_LoadRGBATexture returns an integer that is used to bind to the texture during the mapping of the texture onto polygons.

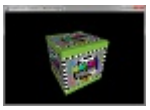
X3Texture is a class that makes working with textures easier. The Width and Height properties stores the dimensions of the texture. The OGLName property stores the Integer that OpenGL assigned to the texture. RGBABitmap is a memory block that stores the pixel color data of the texture.



X3Texture.Constructor:

```
Sub Constructor(texture As Picture)
  Dim x, y, offset As Integer
  Dim textCol As Color
  Dim textMaskCol As Color
  Dim alpha As Byte

  Width = texture.Width
  Height = texture.Height

  RGBABitmap = new MemoryBlock(Height * Width * 4)

  offset = 0

  for y = 0 to Height - 1

      for x = 0 to Width - 1

      textCol = texture.RGBSurface.Pixel(x,y)
      textMaskCol = texture.Mask.RGBSurface.Pixel(x, y)

      alpha = 255 - (textMaskCol.Red + textMaskCol.Green +
textMaskCol.Blue) / 3

      RGBABitmap.Byte(offset) = textCol.Red
      RGBABitmap.Byte(offset + 1) = textCol.Green
      RGBABitmap.Byte(offset + 2) = textCol.Blue
      RGBABitmap.Byte(offset + 3) = alpha

      offset = offset + 4

    next x

  next y

  OGLName = X3_LoadRGBATexture(RGBABitmap, Width, Height)
End Sub
```

The constructor of the X3Texture class takes a normal Picture object as a parameter. First the Width and Height properties of the X3Texture is initialized with the Width and Height properties of the picture object. The picture object is then transformed by the constructor into a MemoryBlock that is stored in the RGBABitmap property. Finally the texture bitmap is loaded into memory with the X3_LoadRGBATexture helper method.



X3UVCoordinate.Constructor:

```
Sub Constructor(initU As Double, initV As Double)
  U = initU
  V = initV
End Sub
```

The X3UVCoordinate object stores a single UV-coordinate, and when used in an array together with other X3UVCoordinate instances, forms an UV-map that maps a texture to a polygon. The constructor method makes it easy to instantiate a new coordinate with predefined values. We simply set the U and V properties of the coordinate equal to the given values.

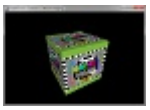
X3Core.X3_Initialize:

```
OpenGL.glEnable OpenGL.GL_TEXTURE_2D
```

The new instruction added to the end of the X3_Initialize method, enables OpenGL's texture mapping feature. We can now bind to textures during rendering, and map these textures onto polygons.

X3Core.X3_RenderModel:

```
Dim i, j As Integer
Dim poly As X3Core.X3Polygon

OpenGL.glBegin OpenGL.GL_TRIANGLES

for i = 0 to model.Polygon.Ubound

  poly = model.Polygon(i)

  OpenGL.glNormal3d poly.Normal.X, poly.Normal.Y, poly.Normal.Z

  if (poly.Texture <> nil) and (poly.UVMap.Ubound >=
     poly.Vertex.Ubound) then

    OpenGL.glColor4d(1, 1, 1, 1)

    OpenGL.glBindTexture(OpenGL.GL_TEXTURE_2D, poly.Texture.OGLName)

    for j = 0 to poly.Vertex.Ubound

      OpenGL.glTexCoord2d poly.UVMap(j).U, poly.UVMap(j).V

// continue on next page
```



```
// continued from previous page

      OpenGL.glVertex3d poly.Vertex(j).X, poly.Vertex(j).Y,
          poly.Vertex(j).Z

    next j

    OpenGL.glBindTexture(OpenGL.GL_TEXTURE_2D, 0)

  else

    if poly.FillColor <> nil then
      OpenGL.glColor4d(poly.FillColor.Red, poly.FillColor.Green,
          poly.FillColor.Blue, poly.FillColor.Alpha)
    else
      OpenGL.glColor4d(1, 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

  end if

  next i

OpenGL.glEnd
```

The new rendering routine distinguishes between two types of polygons, polygons that have a texture object assigned to them, and polygons without textures (only color).

When a polygon has a texture assigned to it, we bind to this texture using the glBindTexture function. The glTexCoord2d function is called with the UV-coordinates for each vertex, just before sending the vertex coordinates to OpenGL. Once we've completed the drawing of the vertices, we "unbind" from the texture using the glBindTexture function with a parameter value of 0.