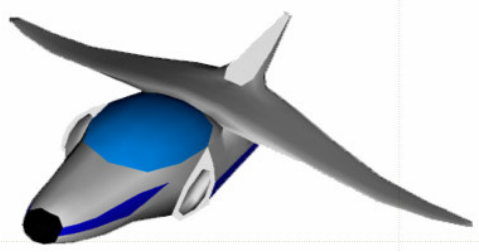


XNA Tutorials

Utah State University
Association for Computing Machinery
XNA Special Interest Group
RB Whitaker
17 October 2007

Texturing

Tutorial 10

Overview

We are now in a place where we can take a look at one of the heavyweights of today's computer graphics: texturing. We will discuss what it is, and how it works. Then we will use it to make a backdrop for our scene.

Texturing

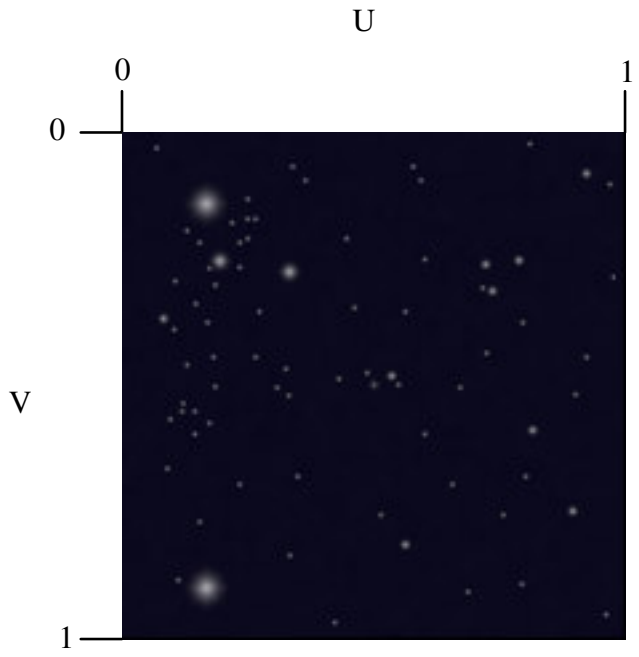
So what exactly is texturing? So far, we've been using only colors to make our models look nice. Texturing is simply sticking an image onto our triangles as we draw them. Imagine how much cooler a spaceship model will look if we put metal-looking images on the triangles, rather than just coloring it gray. And imagine if we wanted to see the veins on a leaf that we are modeling. We could make a model that has bumps all along it, or, for far fewer polygons, we could just stick an image on our leaf and let the image take care of the veins. Texturing will save us a lot of polygons, which means we can have more stuff on the screen and still run just as fast.

Texturing is used in virtually every game these days, and it will probably continue to be the best choice for quite some time.

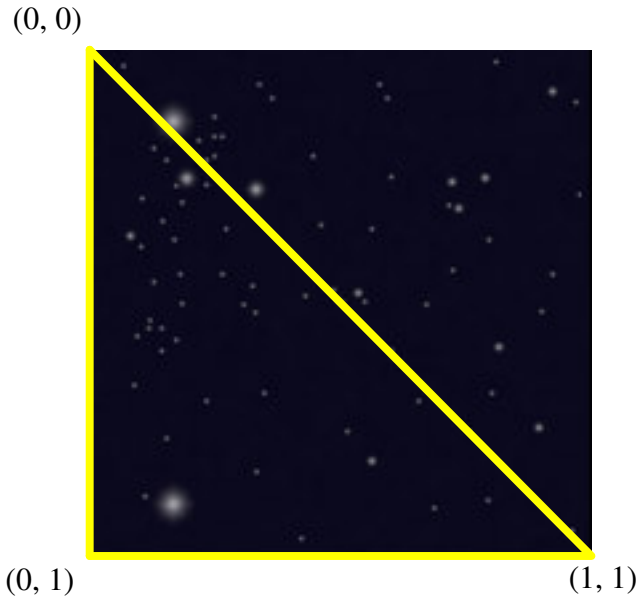
UV-Coordinates

So now that we know what texturing is, let's take a look at how it works. When we define our triangle, in addition to defining the x-, y-, and z-coordinate of each vertex, we will also specify where in the texture our vertex is.

The way we do this is by using uv-coordinates. A uv-coordinate indicates a location in the texture image that we want to use. U is the location horizontally, and V is the location vertically, as shown in the image below. Also, rather than specifying a location in pixels (i.e. "50 pixels over and 20 pixels down"), we specify our coordinate as a fraction of how far to go across the entire image. The top left corner is (0, 0) and the bottom right corner is (1, 1).



All we have to do for our triangle is assign it u- and v-coordinates for each of its vertices. For example, we could create a triangle like the one shown in the picture below, and the bottom corner of the image will be drawn on our triangle. We could use a second triangle for the other half of the image, which is what we will be doing in this tutorial.

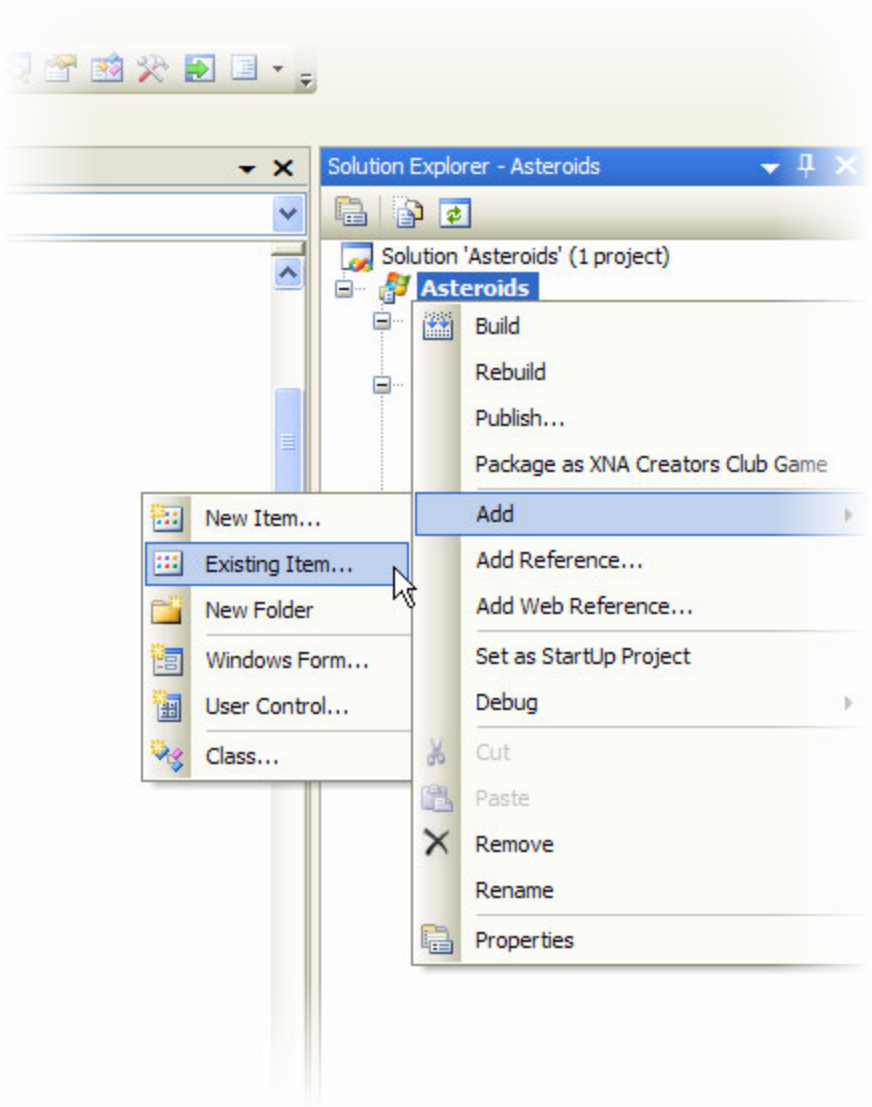


Loading the Texture Image

The first thing we need to do is to load the texture image into our program. But in order to get it into our program, first we need an image. You can use any image you want, or you could download the image that I will be using in this tutorial from

http://cc.usu.edu/~rbwhitaker/images/stars.jpg. Copy this image (or one of your own) into your program's root directory (where we put the .fx file).

Next, we need to bring it into our program. In the Solution Explorer window, right click on your project to bring up the menu. Under "Add", select "Existing Item". Find the image and click "Add". The image will now appear in your list of resources.



Finally, let's load the image in our program as a texture. First, next to the rest of your member variables, add the following line:

```
private Texture2D texture;
```

This variable will store our texture information for us. Now all we need to do is load the image into this Texture2D object. Find the method `LoadGraphicsContent()` and add the following line:

```
texture = content.Load<Texture2D>("stars");
```

This will load the image from our content pipeline to our texture. Our texture is now loaded, and we are ready to move on.

Setting up the Background

The next thing we need to do is set up the triangles that will represent the background. The first thing we need to do is create an array to hold our triangles, so go back to where we defined our other member variables and add the line below.

```
private VertexPositionTexture[] backgroundVertices;
```

The `VertexPositionTexture` data type stores the information needed for a vertex that has both a position, and a texture coordinate.

Next, let's create a method called `protected void SetupBackground()` in our `Asteroids` class, where we can do all of the work for setting up our background triangles. Add the following code to this method:

```
// How far back the background is
float depth = -3;

// The size from the center of the background to the edge
float size= 4;

// We need 6 vertices for the background.
backgroundVertices = new VertexPositionTexture[6];

// The first triangle
// The top left corner
backgroundVertices[0].Position = new Vector3(-size, size, depth);
backgroundVertices[0].TextureCoordinate.X = 0;
backgroundVertices[0].TextureCoordinate.Y = 0;

// The bottom left corner
backgroundVertices[1].Position = new Vector3(-size, -size, depth);
backgroundVertices[1].TextureCoordinate.X = 0;
backgroundVertices[1].TextureCoordinate.Y = 1;

// The bottom right corner
backgroundVertices[2].Position = new Vector3(size, -size, depth);
backgroundVertices[2].TextureCoordinate.X = 1;
backgroundVertices[2].TextureCoordinate.Y = 1;

// The second triangle
// The top left corner
backgroundVertices[3].Position = new Vector3(-size, size, depth);
backgroundVertices[3].TextureCoordinate.X = 0;
backgroundVertices[3].TextureCoordinate.Y = 0;

// The bottom right corner
backgroundVertices[4].Position = new Vector3(size, -size, depth);
backgroundVertices[4].TextureCoordinate.X = 1;
backgroundVertices[4].TextureCoordinate.Y = 1;

// The top right corner
```

```
backgroundVertices[5].Position = new Vector3(size, size, depth);
backgroundVertices[5].TextureCoordinate.X = 1;
backgroundVertices[5].TextureCoordinate.Y = 0;
```

This code does a number of things. First, we set up the information about where in 3D space our background is going to be. We are looking down the z-axis (towards negative infinity), so this is basically the z-coordinate of the plane that our background is going to be on. If we made this something like -15, our texture would appear farther away in the background. The `size` variable says how big our background will be. You can play around with these two variables later if you want to. These values, though, will give us a nice background.

The third line defines our backgroundVertices array to have 6 vertices in it, which we will need since we are drawing two triangles. We could use index buffers again if we wanted to, but for now, this will be simpler.

The rest of the code defines each vertex, one at a time. We give it a position in the first line, and then the last two lines are the u- and v- coordinate in the image.

Lastly, let's add a call to the Initialize method to run this method when the program is getting initialized. Add the line below:

```
SetupBackground();
```

Drawing the Background

Now all we need to do is draw the background. Drawing textured vertices is not really all that different from colored vertices. But before we make a big mess of our Draw method, let's do a couple of things to make our code still manageable. First, create a method identical to the code below:

```
protected void DrawObject()
{
}
```

Now move all of the code in the Draw method *except* the first line (`graphics.GraphicsDevice.Clear(Color.Black);`) and the last line (`base.Draw(gameTime);`) over to this method. Our Draw method will now contain only the two lines we left, and our DrawObject method should look something like this:

```
protected void DrawObject()
{
      effect.CurrentTechnique = effect.Techniques["Colored"];
      effect.Begin();

      Matrix rotationMatrix = Matrix.CreateRotationY(angle);
      Matrix rotation2 = Matrix.CreateRotationX(angle / 2);
      Matrix translationMatrix = Matrix.CreateTranslation(new
                        Vector3(0, 1, 0));
      Matrix worldMatrix = rotation2 * rotationMatrix;
      effect.Parameters["xWorld"].SetValue(worldMatrix);

      device.RenderState.CullMode = CullMode.None;
```

```
        foreach (EffectPass pass in effect.CurrentTechnique.Passes)
        {
                pass.Begin();

                device.VertexDeclaration = new VertexDeclaration(device,
                        VertexPositionColor.VertexElements);

                device.Vertices[0].SetSource(vertexBuffer, 0,
                        VertexPositionColor.SizeInBytes);
                device.Indices = indexBuffer;
                device.VertexDeclaration = new VertexDeclaration(device,
                        VertexPositionColor.VertexElements);

                device.DrawIndexedPrimitives(PrimitiveType.TriangleList, 0,
                        0, 12, 0, 20);

                pass.End();
        }

        effect.End();
}
```

Add the two lines below to your Draw method:

```
                DrawBackground();
                DrawObject();
```

DrawBackground() is a method that we are about to create in a minute, and DrawObject will call the code for drawing our Icosahedron. Now create a new method using the code below:

```
protected void DrawBackground()
{
        Matrix worldMatrix = Matrix.Identity;
        effect.CurrentTechnique = effect.Techniques["Textured"];
        effect.Parameters["xWorld"].SetValue(worldMatrix);
        effect.Parameters["xTexture"].SetValue(texture);

        effect.Begin();

        foreach (EffectPass pass in effect.CurrentTechnique.Passes)
        {
                pass.Begin();

                device.VertexDeclaration = new VertexDeclaration(device,
                                VertexPositionTexture.VertexElements);
                device.DrawUserPrimitives(PrimitiveType.TriangleList,
                                backgroundVertices, 0, 2);

                pass.End();
        }

        effect.End();
}
```

Much of this code should be familiar. When we are drawing our background, we don't want to do any transformations, or our background will be moving around, so we use the identity matrix as our transformation matrix.

Also notice that we are using a new effect called "Textured". This effect let's us do texturing. We use the line `effect.Parameters["xTexture"].SetValue(texture);` to give the texture to the effect on the graphics card. Also note that for our vertex declaration, we use VertexPositionTexture.VertexElements, instead of VertexPositionColor.VertexElements, since that is what we are going to be using here.

The rest of this code should look familiar to you, so I won't bother explaining it.

We should now be able to run our program again, and see our textured background. Our result should look like the image below: