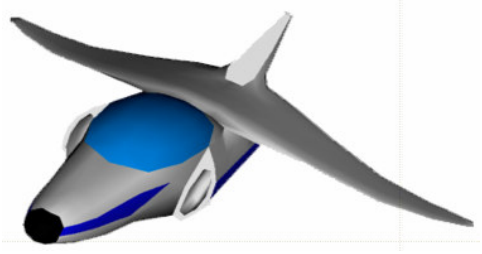


XNA Tutorials

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

More Texturing

Tutorial 11

Overview

Now that we have done some basic texturing for our background, let's try doing something a little bit harder. Let's try texturing our icosahedron as an asteroid. Of course, if we were going to really make asteroids, we would want a better model than an icosahedron, but we will leave that for another tutorial. This tutorial will mostly consist of making changes to our code to accommodate this texturing, but very little new material will be covered.

Loading the Asteroid Texture

We now want a new texture for our asteroid. Again, you are more than welcome to use any texture image you want, but I will be using one called "asteroid.jpg", which you can download from http://cc.usu.edu/~rbwhitaker/images/asteroid.jpg.

Let's create a new Texture2D object as another member variable, in the same place that we created the texture object for our background. Use the following line to create it:

```
private Texture2D asteroidTexture;
```

Now load the texture image into your program, via the Solution Explorer like we did before. Add the following line to your `LoadGraphicsContent()` method:

```
asteroidTexture = content.Load<Texture2D>("asteroid");
```

Now the texture has been loaded, like we needed it to be.

The Vertex Buffer

There are three changes we need to make in order to load our textured vertices into the vertex buffer. First, we want to change the type of our vertex array to VertexPositionTexture. Change the line in your SetupVertexBuffer method that says:

```
VertexPositionColor[] vertexArray = new VertexPositionColor[12];
```

to say:

```
VertexPositionTexture[] vertexArray = new VertexPositionTexture[12];
```

Now instead of defining the colors of the vertices, we will need to define the uv-coordinates of each vertex. We can now go ahead and delete the code we made for making colors in the SetupVertexBuffer() method. Replace it with the code below (or something like it):

```
Random random = new Random();
for (int i = 0; i < 12; i++)
{
   vertexArray[i].TextureCoordinate.X = (float)random.NextDouble() / 8;
   vertexArray[i].TextureCoordinate.Y = (float)random.NextDouble() / 8;
}
```

This code will pick a random u- and v-coordinate for every vertex. We divide by 8, because that will make it look a little nicer, but it is actually an arbitrary decision. In a few tutorials, we will find a much nicer way of getting a hold of texture coordinates.

The final thing we must do in this method is change one of the attributes in our assignment to the vertexBuffer object. Change the line that says:

```
vertexBuffer = new VertexBuffer(device, sizeof(float) * 4 *
                   vertexArray.Length, ResourceUsage.WriteOnly,
                   ResourceManagementMode.Automatic);
```

to say:

```
vertexBuffer = new VertexBuffer(device, sizeof(float) * 5 *
                   vertexArray.Length, ResourceUsage.WriteOnly,
                   ResourceManagementMode.Automatic);
```

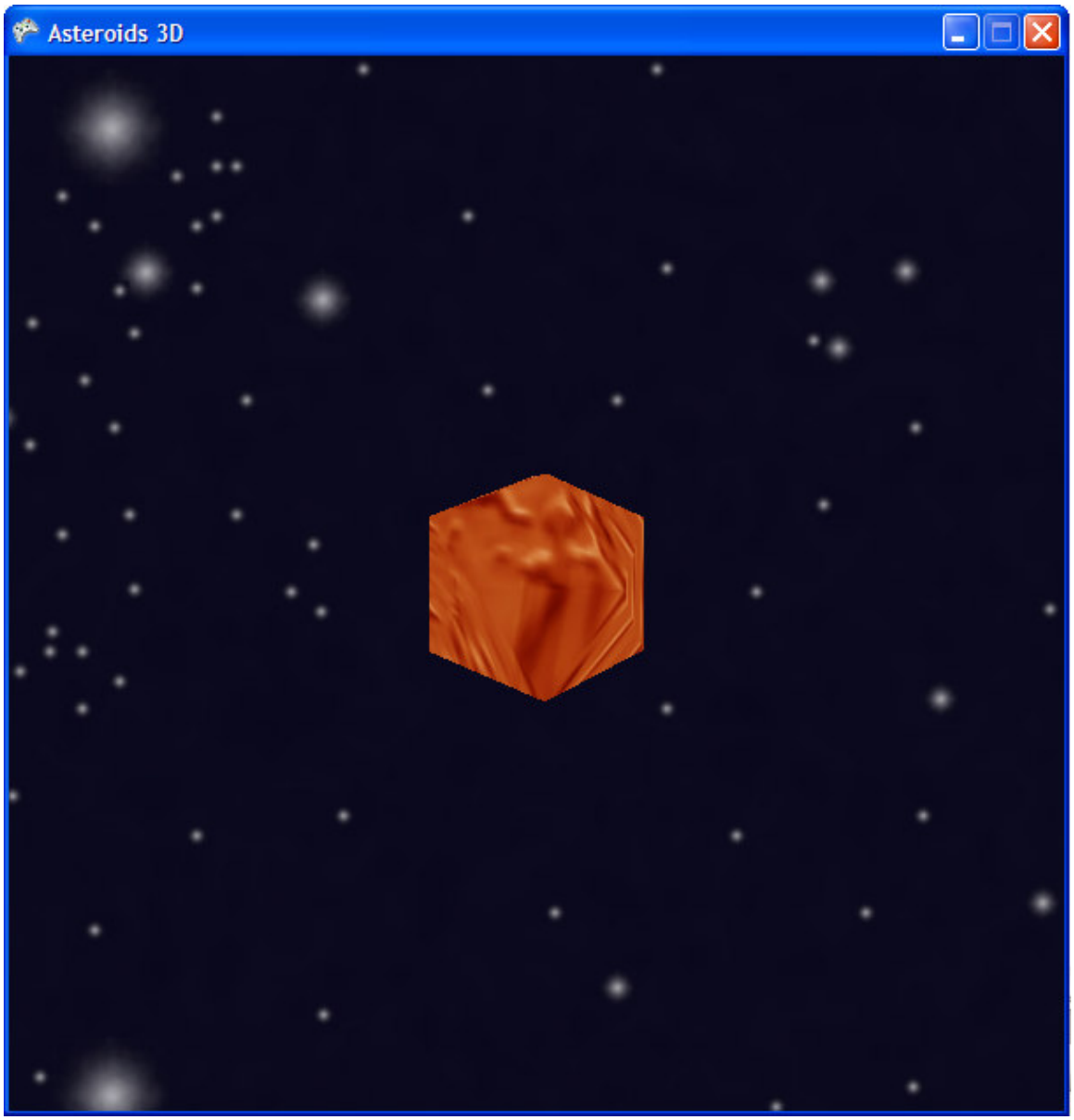
We change this value from 4 to 5 because now our data structure has 5 fields instead of 4.

Drawing the Textured Asteroid

We now need to make a few changes to the way we are drawing our "asteroid" in order to be able to draw it correctly. Go to the DrawObject method. Change the technique from "Colored" to "Textured" like we did with the background image. Just after we set the "xWorld" parameter, set the texture to be the asteroid texture using the following line of code:

```
effect.Parameters["xTexture"].SetValue(asteroidTexture);
```

Inside the foreach loop, there are three references to `VertexPositionColor`. Change each of these to `VertexPositionTexture`. Now we are ready to run our program again. The output should look like the image below:


Asteroids 3D