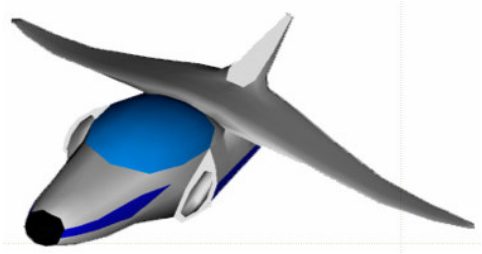


XNA Tutorials

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

Triangles

Tutorial 5

Overview

Finally we are ready to begin doing some drawing. We will first cover why we are going to learn how to draw triangles before anything else. This tutorial will also cover a data structure for storing a triangle, and then how to actually draw a triangle.

Why Triangles?

You might be thinking, "After five tutorials, all I'll be able to draw is triangles?!" However, in the world of graphics, everything that you see is made of triangles. The model used in the image in the upper left corner of this page was built out of about 1200 triangles.

So even though triangles may seem very primitive, they are what are used to draw everything else. And, of course, we need to be able to draw one triangle before we can draw a 1200 triangle model.

Storing Our Triangle

What exactly is it going to take to store a triangle? For starters, we are going to want to store an x-, y-, and z-coordinate for every vertex. But it would also nice to store a color for the vertex as well.

Fortunately, XNA has a built in data structure for storing the information we need. It is the VertexPositionColor struct. This struct can handle the information for one vertex of our triangle.

Since we are going to be drawing a triangle with three vertices, we need to create an array of these. Let's add the following member variable to our Asteroids class. Next to our declaration of our effect and graphics device, just before the constructor, add the following line of code:

```
private VertexPositionColor[] vertices;
```

Next, let's create a method where we can define what information is in the vertices. Create a method in your class that looks like the code below:

```
protected void SetupVertexArray()
{
    vertexArray = new VertexPositionColor[3];

    vertexArray[0].Position = new Vector3(-.5f, -.5f, 0);
    vertexArray[0].Color = Color.Red;

    vertexArray[1].Position = new Vector3(0, .5f, 0);
    vertexArray[1].Color = Color.Blue;

    vertexArray[2].Position = new Vector3(.5f, -.5f, 0);
    vertexArray[2].Color = Color.Green;
}
```

This will give us a nice colored triangle, when we're done. Now we just have to call this method so that the array gets set up. You can call this from the constructor, since it doesn't require the graphics device to already be set up. Add the line `SetupVertexArray();` to the constructor.

Drawing Our Triangle

Now we can draw our first triangle. Go to your `Draw()` method. Between the `pass.Begin()` and `pass.End()` is a spot for us to draw our triangle. With two lines we can draw our triangle. Add the two lines below to your code in this spot.

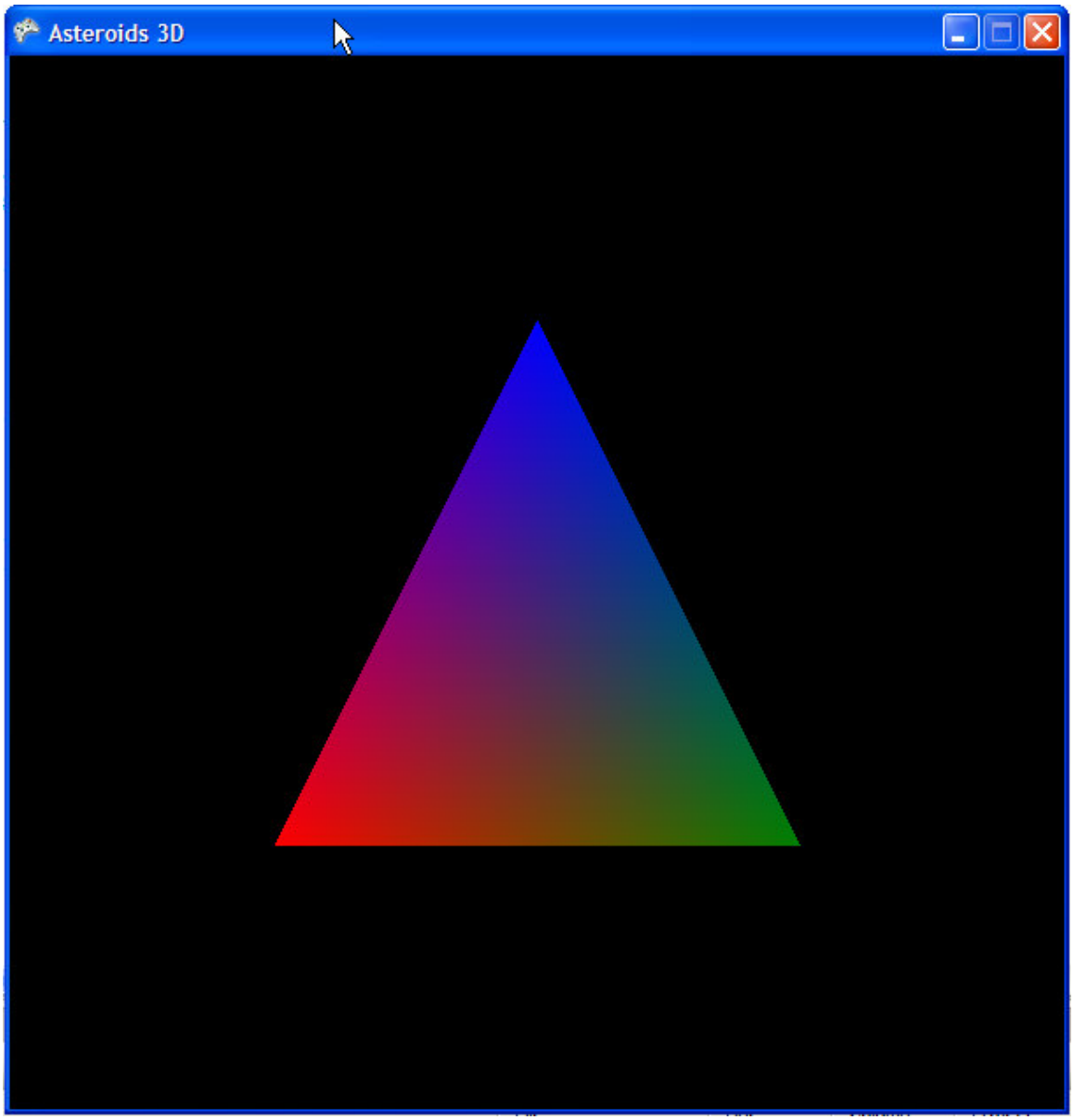
```
device.VertexDeclaration = new VertexDeclaration(device,
            VertexPositionColor.VertexElements);

device.DrawUserPrimitives(PrimitiveType.TriangleList, vertexArray, 0,
            1);
```

The first line tells the graphics card what type of data is about to be sent to it. There are lots of different types we could send to it, and so we want to make sure it knows how to interpret what it receives.

The second line sends the actual information over to the graphics card, and tells it to draw it. The first parameter, `PrimitiveType.TriangleList`, tells the graphics card that we are sending a list of triangles (as opposed to a triangle strip or triangle fan). The second parameter contains our actual data. The third parameter is what index in the array to start at. This time, we want to start at the beginning of the list (index 0), but this is not always the case. The fourth parameter tells the graphics card how many triangles we want to draw. In our case, we only have one to draw, so we give it 1.

Now run the program again. You should see something like the screen shot below.


Asteroids 3D