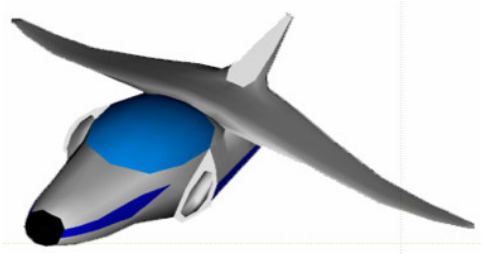

XNA Tutorials

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

Effects and Effect Files

Tutorial 4

Overview

There is still one more significant thing that we need to discuss before we can actually start drawing: effects. In this tutorial, we will first discuss what effects are, and why we need to worry about them here, before we start actually drawing. Then we will discuss how to write the code to handle effects.

Effects and XNA

Back in the day, when 3D graphics were first started, people had ideas about what should be available in your program, including coloring, lighting, and texturing. These were all different types of effects. Many older graphics APIs supplied specific functions for doing things like turning on lights, placing lights, and setting the colors of objects.

There is an advantage and a disadvantage to this idea. The advantage is that all of the work needed to perform the predefined effects, like lighting, was done for you. The disadvantage is that you could *only* do the things that were predefined for you.

XNA uses an alternative solution. With XNA, can program an effect to be exactly the way you want it, and then send the effect over to the graphics card for use. This means that we can do any effect imaginable, opening up a huge world of possibilities. The problem this presents is that now there are no built-in effects for us to use by default.

What's worse is that it is not easy to create an effect on our own. Effects are written in a language called HLSL, which is substantially more complicated than C#.

Fortunately, there is a solution. Scattered throughout the Internet are effect files that are available to use.

Getting Our Effect File

The effect file that we will be using through the next few tutorials is provided by Reimer's XNA Tutorials, and can be found at http://users.pandora.be/riemer/files/effects.fx Download this file and put it in the directory where your code is located at. Feel free to open it up in a text editor to get an idea of what HLSL looks like. Don't worry, HLSL is not something we need to worry

about for a while. For now, we are just going to use what has been defined for us in this file.

Loading the Effect File

The next thing we want to do is to tell our program where to find our effect file, and what to do with it. The first thing we will need is an object within our Asteroids class that can store the information in the effect file. So let's start by defining one. Put the following line of code in your Asteroids class, next to the other member variables.

```
private Effect effect;
```

Go back to the method we created in the last tutorial, `SetupGraphicsDevice()`. We now want to add the following two lines to this method:

```
CompiledEffect compiledEffect =
      Effect.CompileEffectFromFile("@/../../../../effects.fx", null,
      null, CompilerOptions.None, TargetPlatform.Windows);

effect = new Effect(device, compiledEffect.GetEffectCode(),
      CompilerOptions.None, null);
```

Essentially, the first line takes our effect file (/../../../../effects.fx is the path from the .exe file to where we put effects.fx) and compiles it. For now, we are not going to discuss what the other parameters do. The second line takes what was compiled and gives us access to it in our `effect` object. Now we have an effect that we can use. Our `SetupGraphicsDevice()` should now look like this:

```
protected void SetupGraphicsDevice()
{
    device = graphics.GraphicsDevice;

    graphics.PreferredBackBufferWidth = 600;
    graphics.PreferredBackBufferHeight = 600;
    graphics.ApplyChanges();

    Window.Title = "Asteroids 3D";

    CompiledEffect compiledEffect =

    Effect.CompileEffectFromFile("@/../../../../effects.fx",
          null, null, CompilerOptions.None,
          TargetPlatform.Windows);
    effect = new Effect(device, compiledEffect.GetEffectCode,
          CompilerOptions.None, null);
}
```

More about the Effect Object

Now that we have an Effect object, it is probably time that we discuss what it includes. An effect in XNA is essentially simply being a collection of the techniques that we can choose from to do our rendering. A technique is a specific way of drawing objects. For example, do we want lighting? Do we want a texture on our object? Do we

want shadows? A technique will define how to handle these things, and more. When we are about to draw something, we must tell the graphics card "I want to use the technique labeled 'colored'". The graphics card will then find the technique that we called "colored" and use it.

Techniques can also be made up of multiple "passes." For example, we want to put a combination of two textures on our object. Rather than having to merge the two textures in a image editing tool, we can create a separate pass for each texture, and then run each pass on its own. For now, don't worry too much about this. It just means that in our code, we are going to have to take into account that there may be multiple passes in the technique we want to use.

Code for Effects

We will need to add several more lines of code to tell the graphics card what technique in the Effect object we want to use. Since this is preparing for actual drawing, we want to put this code in our `Draw()` method. First of all, we need to tell the graphics card what technique we want to use.

```
effect.CurrentTechnique = effect.Techniques["Pretransformed"];
effect.Begin();
```

These two lines say "I want to use the technique called 'Pretransformed'" and then tells the graphics card, start using this effect now.

The pretransformed technique is a little bit of a cheat. It lets us skip over a number of things so that we can start drawing stuff immediately. It won't be long before we will discard this technique, and use something more real.

Finally, since our technique may be made up of multiple passes, we need to go down the list of passes, and do each one individually. The code below will do just this, and should be added in the `Draw()` method, just below the code we just put in.

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

    // We will draw in here

    pass.End();
}
```

Finally, we need to say that we're done with the current effect. Add the line below to your code just after the foreach loop.

```
effect.End();
```

Our `Draw()` method should now look like this:

```
protected override void Draw(GameTime gameTime)
{
      graphics.GraphicsDevice.Clear(Color.Black);
```

```
        effect.CurrentTechnique = effect.Techniques["Pretransformed"];
        effect.Begin();

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

              // We will draw in here

              pass.End();
        }

        effect.End();

        base.Draw(gameTime);
}
```

In the very near future, we will move away from the “Pretransformed” technique, and on to something more complicated, but more powerful. Also, at some point in the future, when we know XNA better, we will want to start making our own effects from scratch. But for now, let’s go on and start doing some drawing.