CPI 411 Graphics for Games

Lab 1

This lab is intended to introduce you to the basic of shader programming in the high level shader language (HLSL). We are going to display a triangle with several different colors by customizing the SimplestShader.fx file.

A. SimplestShader

The first step is to understand the minimum HLSL code. It requires three blocks. The first block is the vertex shader. The second block is the pixel shader, and the last block is the main function to set the vertex and pixel shaders.

```
float4 MyVertexShader(float4 position: POSITION) : POSITION
{
	return position;
}

float4 MyPixelShader(): COLOR
{
	return float4(1, 1, 1, 1);
}

technique MyTechnique
{
	pass Pass1
	{
		VertexShader = compile vs_4_0 MyVertexShader();
		PixelShader = compile ps_4_0 MyPixelShader();
	}
}
```

This sample is to read each vertex and assign white color. Before going to the details, the game program to call the shader effect is shown below.

B. Test Program (Lab1.cs)

To run the simplest shader, we use the MonoGame /XNA framework. The shader is initialized as an instance of Effect object. This test program is to create a triangle (three vertices) to send to the shader. A list of vertices are sent as a VertexElement struct data. There are several options of VertexElement in XNA.

- `VertexPositionColor:` `Position, Color`
- `VertexPositionColorTexture:` `Position, Color, Texture(UV)`
- `VertexPositionNormalTexture:` `Position, Normal, Texture(UV)`
- `VertexPositionTexture:` `Position, Texture(UV)`

This program uses the VertexPositionColor to store the triangle. Look at the constructor by yourself in the reference document.

The shader effect is simply loaded using Content.Load() method in the Load() method similar to the other content.

To display the triangle, we use the GraphicsDevice.DrawUserPrimitives<VertexPositionColor>(...) method. The XNA has four methods to draw polygons.

- DrawPrimitives: Using a VertexBuffer
- DrawIndexedPrimitives: Using a IndexBuffer
- DrawUserPrimitives: Using a struct of vertex.
- DrawUserIndexedPrimitives: Using a struct of index and vertex.

This case, we use the DrawUserPrimitives with an array of built-in VertexPositionColor. The following is the definision.

```
DrawUserPrimitives(
    < Primitive Type>,
    <array of vertex>,
    <start index>,
    <# of polygon>);
```

Now read the test program.

```
public class Lab1 : Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Effect effect;
    VertexPositionColor[] vertices =
    {
        new VertexPositionColor(new Vector3(0, 1, 0), Color.White),
        new VertexPositionColor(new Vector3(1, 0, 0), Color.Blue),
        new VertexPositionColor(new Vector3(-1, 0, 0), Color.Red)
    };

    public Lab1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = "Content";
        // ***** From MonoGame3.6 Need this statement
        graphics.GraphicsProfile = GraphicsProfile.HiDef;
        // ********************************************
    }

    protected override void Initialize(){base.Initialize();}
    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        effect = Content.Load<Effect>("SimplestShader");
    }

    protected override void UnloadContent(){}

    protected override void Update(GameTime gameTime)
    {
        ...
```

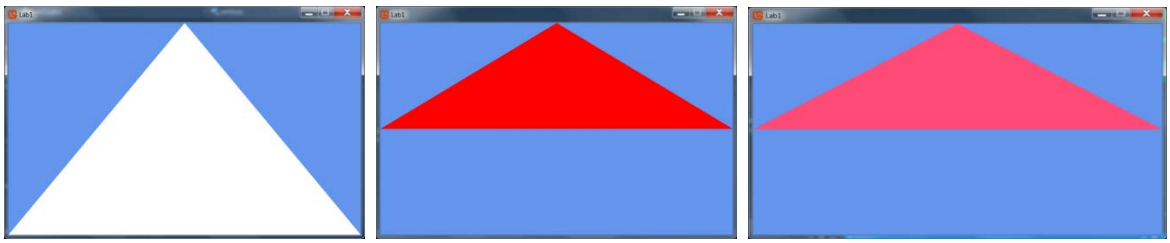
```
        }
        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            GraphicsDevice.BlendState = BlendState.AlphaBlend;

            foreach (var pass in effect.CurrentTechnique.Passes)
            {
                pass.Apply();

                GraphicsDevice.DrawUserPrimitives<VertexPositionColor>
                    (
                        PrimitiveType.TriangleList,
                        vertices,
                        0,
                        vertices.Length / 3
                    );
            }
        }
    }
```

C. VertexPositionColor

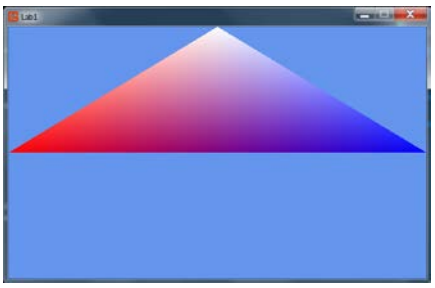
We get the result showing a white triangle on the screen as shown below.

Exercise-1: Make the following outputs by changing the code. The first one is to change the positions of vertex. The second is to try to paint in different color. The last one is to use the transparent value to 0.5. In order to use the alpha value, go to the Draw() method of test program and add the following code.

```
GraphicsDevice.BlendState = BlendState.AlphaBlend;
```



Exercise-2: So far the color information of vertex is not sent to the shader side. Let's try to show the following output.



First, it needs to add a **`struct`** for the shader file in order to get the position and color from the test program.

```
struct VertexPositionColor {
      float4 Position: POSITION;
      float4 Color: COLOR;
};
```

Second, modify the input and output data from `float4` to the `VertexPositionColor`. Add the float4 color: COLOR for the input of pixel shader. Since the input has the COLOR semantic, so the VertexPositionColor's Color is sent to the pixel shader.

Once it is done, test the following code in the pixel shader.

- `color.r = 0;`
- `color.rb= 0;`
- `color.rb = color.br;`
- `color = 1 – color;`
- `color += 0.3f`
- `if (color.r %0.1 < 0.05f) return float4(1, 1, 1, 1); else return color;`

D. VertexPositionTexture (Main Exercise)

The next step is to use texture (image data). In the shader side, it needs to define a sampler in order to use the image data.

```
texture MyTexture;

sampler mySampler = sampler_state{
      Texture = <MyTexture> ;
};
```

The sampler is to define the resource data and several attributes for texture mapping. We have to modify several parts of shader as we studies in the previous exercise.

Make a new **`struct`** named VertexPositionTexture with the position and UV coordinate to import the data from the main program. The struct should have the following statements.

```
float4 Position: POSITION;
```

```
float2 TextureCoordinate : TEXCOORD;
```

Based on this change, update the data type of input and output parameters in the shader functions. For example, the pixel shader is supposed to have the float2 texture coordinate with TEXTCOORD semantic.

The pixel shader returns the pixel information using the tex2D function such as the statement below. The mySampler is the texture data, and textureCoorinate is the UV position. Check the section 3.3.3 Built-in-functions at http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter03.html (Cg Tutorial Chapter 3).

```
return tex2D(mySampler, textureCoodinate);
```

Once the shader is modified, we need to modify two parts of test program. First, change the data type of vertices[ ] from VertexPositionColor to VertexPositionTexture. The constructor is explained at (https://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphics.vertexpositiontexture.vertexpositiontexture.aspx) Assign a good UV value for each vertex.

The other part is to set the parameter (global variable) of shader side. The shader is created as an instance of Effect object. Use the SetValue() method of the effect parameters. The following is the syntax.

```
effect.Parameters["<shader's variable>"].SetValue(<data>);
```

This case is to update the MyTexture, so use the following statement to set the "logo_mg" image file.

```
effect.Parameters["MyTexture"].SetValue(Content.Load<Texture2d>("logo_mg"));
```

***** IMPORTANT *****

Complete the exercise in D section, and submit a zipped file including the solution (.sln) file and the project folders to course online site in the "**Quiz and Lab**" section. Each lab has **10 points**. If you complete the exercise in class time, the full points will be assigned. The late submission is accepted just before the next class with 2 points reductions, because the solution is demonstrated in the next class.