CPI 411 Graphics for Games

Lab 2

This lab is to introduce you to the basic of 3D graphics programming in the high level shader language (HLSL). We are going to display a triangle with several different views by customizing the SimpleTexture.fx file, which was studied in the previous lab.

A. 2D Rotation & Key Control

Before starting 3D programming, we review the key board control in XNA/MonoGame. Without changing the view, we cannot check if a 3d object is correctly displayed or not. We use the same triangle in the previous lab. The triangle moves in the X and Y position corresponding to the keys. Do you have the solution? If you have, do it yourself. Otherwise, following the instruction below.

It is not fun to move a triangle in the X and Y direction because we did a lot of similar exercises in the other courses. I am going to rotate the triangle referring to the center of screen. In order to implement the control, it is required to make a global variable to keep the angle. Once the user presses the right/left arrow, then the angle is updated.

```
float angle; // a global variable

-- in the Update() method --
...
if (Keyboard.GetState().IsKeyDown(Keys.Left))
{
   angle += 0.02f;
   Vector3 offset =  new Vector3(
             (float)System.Math.Cos(angle),
             (float)System.Math.Sin(angle),
              0);
   effect.Parameters["offset"].setValue(offset);
}
```

The next step is to send the offset (cosA, sinA, 0) to the shader side to update position of each vertex. Make a new shader file named SimpleRotate.fx. Add a global variable offset (this is in HLSL so the type is not Vector3 but float3), and update the XYZ position with the offset value.

```
-- in HLSL --
float3 offset;

-- in the vertex shader
input.Position.xyz += offset;
return input;
```

B. First 3D Shader

Now let's start the first 3D shader programming. The objective is to understand the pipeline of 3D shader programming. Make a new shader file named Simple3D.fx.

The first step is to understand the matrix to convert vertices from 3D space to screen space. Look at the course materials if you don't have any idea to implement 3D transformation.

Add three matrices, World, View, and Projection. In 3D computer graphics, we use 4x4 matrixes for each transform. These are from the game program side, so declare the global variables as followings.

```
float4x4  World;
float4x4  View;
float4x4  Projection;
```

The Vertex data should be same as before. We have only to convert each position by multiplying each matrix above. In HLSL, **`mul(A,B)`** function is used to multiply vector A and matrix B. Therefore, the vertex shader should be like below.

```
-- in the vertex shader
VertexPositionTexture output;
float4 worldPos = mul(input.Position, World);
float4 viewPos = mul(worldPos, View);
output.Position = mul(viewPos, Projection);
output.TextureCoordinate = input.TextureCoordinate;
return output;

*) Other option : mul(A,B) can be used for matrix * matrix.
output.Position = mul(worldPos, mul(View, Projection));
```

That is all! It is very simple. Next step is to make the matrices and update the View matrix corresponding to the angle, which is controlled by a user.

C. Game Program for 3D

The XNA/MonoGame API provides many useful methods for 3D graphics programming. In order to become a good CG programmer, you are asked to remember the core methods one by one. This time, memorize the following methods.

1. Making a world matrix: We use only one triangle, so it should be identity matrix (I). Use the following statement.

   ```
   Matrix world = Matrix.Identity;
   ```

2. Making a view matrix: A camera view is defined with Position, Traget, and Up directions. The following statement is an example to make the view matrix, where the position is (1, 0, 1), the target is (0, 0, 0), and the Up direction is (0, 1, 0).

```
Matrix view = Matrix.CreateLookAt (
      new Vector3(1, 0, 1),
      new Vector3 (),
      new Vector (0, 1, 0));
```

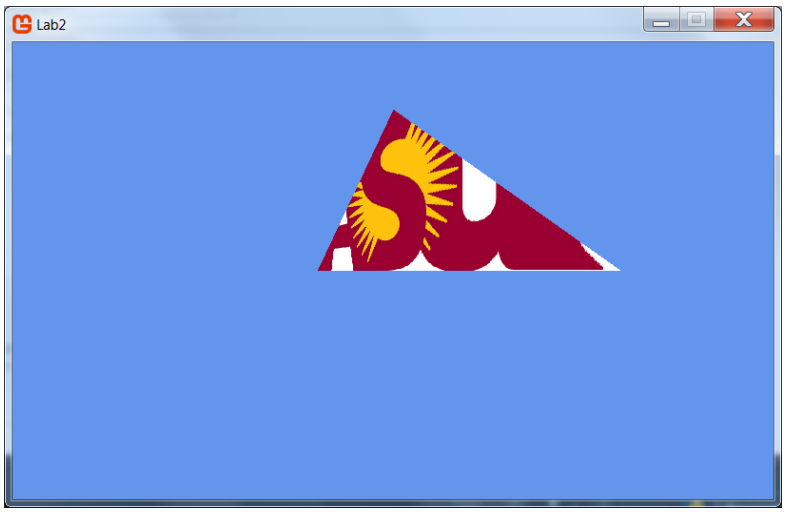
3. Make a projection matrix: A projection is a matrix to set the visible space. It is defined with the field of view (visible range), proportion of screen, and nearest and farthest plane (distances). For example, the following statement is to initialize the projection in which the visible range is 90 degree, the proportion from the current viewport, and the nearest and furthest plane distances are 0.1 and 100. In other words, any object whose the distance to camera is less than 0.1 or larger than 100, is not displayed.

```
Matrix projection = Matrix.CreatePerspectiveFieldOfView(
      MathHelper.ToRadians(90),
      GraphicsDevice.Viewport.AspectRatio,
      0.1f, 100);
```

Once the matrices are created, send the data to the shader (effect) side.

```
effect.Parameters["World"].SetValue(world);
effect.Parameters["View"].SetValue(view);
effect.Parameters["Projection"].SetValue(projection);
```

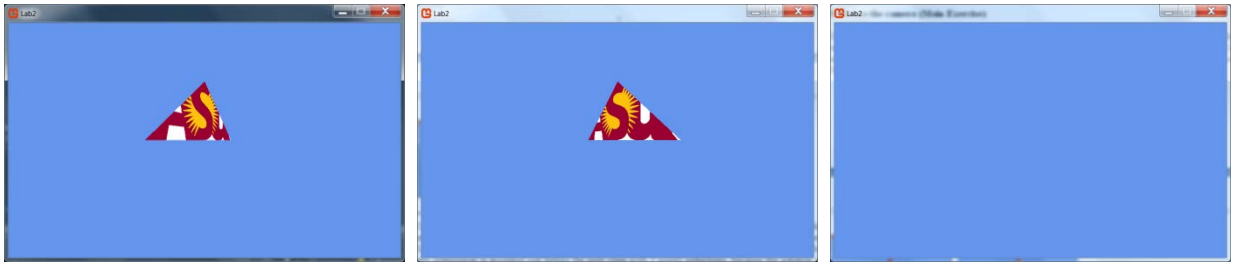
You see the output like below.



D. Rotate the camera (Main Exercise)

Today's exercise is to add the key controls (left and right arrow keys) to rotate the camera. Based on the section A, update the view **matrix** corresponding to the key input instead of the offset. The following is an example to update the camera position using the angle value. The triangle is crated in XY plane, so the camera should be on XZ plane.

```
Vector3 cameraPosition = distance * new Vector3(
          (float)System.Math.Sin(angle), 0, (float)System.Math.Cos(angle));
```



If the camera is backside of triangle, then it is disappeared.
Add other key controls such as changing the distance to the triangle. (Up/Down keys)

***** 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. The submission item is located 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.