Lab 3. HLSL programming

Objectives

1. Understand vertex shader and pixel shader basics, including their inputs and outputs
2. Understand different types of variables
3. How to use shaders in Direct3D 11
4. Understand how to compile a Shader, create a Shader Object and set the Shader Object

Exercise 1. Create your own vertex shaders

1. Open 'Tutorial04.fx', create a new vertex shader VS_main() by copying of the function VS( ):

```
VS_OUTPUT VS_main( float4 Pos : POSITION, float4 Color : COLOR )
{
   VS_OUTPUT output = (VS_OUTPUT)0;
   output.Pos = mul( Pos, World );
   … …
   return output;
}
```

2. Modify the new vertex shader, such that it transforms the input object first by a translation defined by a vector T=(1, 0.3, 1.0), then by a scaling transformation defined by a vector Scale=(0.2, 3, 3), and finally by a rotation around Y-axis. You may need first go to your C++ program, reset the world matrix g_World as the identity matrix, or replace

```
output.Pos = mul( Pos, World );
```

   with

```
float4 inPos = Pos;
```

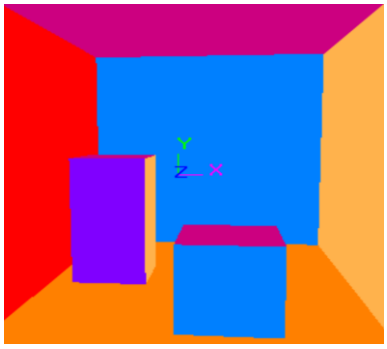
3. Create a new HLSL file VS2.HLSL by making a copy of Tutorial04_VS.hlsl.
4. Add the new vertex shader program VS2.HLSL to the project and set its properties properly.
5. Open your C++ program to create a vertex shader corresponding to the shader program **VS_main**:
   a. Declare a new `ID3D11VertexShader` variable, similar to g_pVertexShader, say,
      `ID3D11VertexShader*  g_pVertexShader_1 = nullptr;`
   b. Create `CreateVertexShader_1 using` VS_main(), by compiling the shader.
   c. Set the new vertex shader to render your object:
      `g_pImmediateContext->VSSetShader( g_pVertexShader_1, nullptr, 0 );`
   d. Release it once it has been created.
      `if( g_pVertexShader_1 ) g_pVertexShader_1->Release();`
6. Rebuild your project and run you program. Change the transformations used in your shader program and observe how the object input to GPU is changed.

Exercise 2. Create your own pixel shaders

Similar to Exercise 1 to create some pixel shaders of your own to draw, say, three cubes with different colours.

Exercise 3. Cornell box in vertex shaders

Create Cornell box by using THREE different vertex shaders.



Exercise 4. Define Model-View-Projection in vertex shader (optional)

1. Add three new `XMVECTOR element to struct ConstantBuffer corresponding to camera position, camera looking at point and camera up direction:`

```
XMVECTOR mEye;
XMVECTOR mLookAtPoint;
XMVECTOR mUp;
```

1. Initialize these variables in your C++ program.
2. `Replace View matrix in`

```
output.Pos = mul( output.Pos, View );
```

as a translation of eye position to the coordinate origin and rotation (refer to the lecture notes for details).

3. Define the projection matrix in the vertex shader based on the one at https://docs.microsoft.com/en-us/windows/win32/direct3d9/d3dxmatrixperspectivefovlh

```
float4x4 ProjMat=float4x4( … );
```

4. Replace Projection with ProjMat:

```
output.Pos = mul( output.Pos, ProjMat);
```

Exercise 5. Vertex shader point cloud (Optional)

Write a vertex shder in HLSL to create an effect shown at https://www.vertexshaderart.com/art/nL6YpkW8YvGKNEKtj