Objectives

1. To learn how to associate texture coordinates to vertex positions.
2. To understand how to create texture objects using DirectX 11 API.
3. To learn how to specify textures as a shader resource.
4. To understand how different texture sampling technique works.
5. To learn how to create and specify sampling states.
6. To understand how multiple textures can be combined to create different graphics effects.

All the exercises described below can be done directly based on what you have achieved in the lab on lighting. If you would like to see an example on how to load a texture from a file and use it in the fragment shader, you can have a look at the Tutorial07 C++ source code from the Tutorial 07 project.

Exercise 1. A wooden cube

The first thing you need to do is to associate to each vertex a texture coordinate. This can be done in the same way as we associate the normal vectors to vertices for lighting.

1. In the structure D3D11_INPUT_ELEMENT_DESC layout[ ], add a new element corresponding to texture coordinates:

```
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 24, D3D11_INPUT_PER_VERTEX_DATA, 0 },
};
```

2. In the vertex structure, add a new element “TexCoord”

```
struct SimpleVertex
{
    XMFLOAT3 Pos;
    XMFLOAT3 Normal;
    XMFLOAT2 TexCoord;
};
```

3. In the vertex list, specify texture coordinate for each vertex position:

```
SimpleVertex vertices[] =
{
    //Top face
    { XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ), XMFLOAT2(0.0f, 1.0f) },
    { XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ), XMFLOAT2(1.0f, 1.0f) },
    … …
```

4. To create a texture object as a shader resource, two objects need to be created: the shader resource view and sampler state for each texture to be used in a pixel shader to specify how the texture is to be used. To create the two objects, declare two global variables:

```
ID3D11ShaderResourceView*           wood_TextureRV = nullptr;
ID3D11SamplerState*                 wood_Sampler = nullptr;
```

5. Load an image to create a texture. We will use the DirectXTK CreateDDSTextureFromFile( ) method to create a texture.
    a. Copy from project Tutorial07 the following two DirectXTK files to your project directory: “DDSTextureLoader.cpp” and “DDSTextureLoader.h”.
    b. Add the files to your project and include the header file to your c++ program.
    c. In in the InitDevice() method, call the following function to load a DDS image to create the texture object:

```
hr = CreateDDSTextureFromFile( g_pd3dDevice, L"Wood.dds", nullptr, &wood_TextureRV );
```

6. Next, set the texture as a pixel shade resource in the Render( ) method:

```
g_pImmediateContext->PSSetShaderResources(0, 1, &wood_TextureRV);
```

7. Now create a sampler state to specify how the texture is to be used

```
D3D11_SAMPLER_DESC sampDesc;
ZeroMemory(&sampDesc, sizeof(sampDesc));
sampDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
sampDesc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
sampDesc.ComparisonFunc = D3D11_COMPARISON_NEVER;
hr = g_pd3dDevice->CreateSamplerState(&sampDesc, &wood_Sampler);
```

8. In the Render( ) method, specify the sampler state to be used:

```
g_pImmediateContext->PSSetSamplers( 0, 1, &g_pSamplerLinear );
```

9. Now open the pixel shader and add variables regarding the texture and sampler objects you created in your c++ program:

```
Texture2D txWoodColor : register(t0);
SamplerState txWoodsamSampler : register(s0);
```

10. Modify the vertex shader input data structure corresponding to the vertex layout you have created in your application to input vertex data into the vertex shader:

```
struct VS_INPUT
{
    float4 Pos : POSITION;
    float3 Norm : NORMAL;
    float2 Tex : TEXCOORD;
};
```

11. Modify the vertex shader output data structure to pass texture coordinates to the pixel shader:

```
struct PS_INPUT
{
    float4 Pos : SV_POSITION;
    float2 Tex : TEXCOORD0;
    float3 Norm : TEXCOORD1;
    … …
};
```

12. In vertex shader, specify the texture coordinate for each vertex:

```
output.Tex = input.Tex;
```

13. In pixel shader, read the texture colour using HLSL Sampler( ) function:

```
float4 woodColor = txWoodColor.Sample(txSoosamSampler, input.Tex);
```

14. Download the DDS "Wood.dds" texture and put it to your project directory.
15. You can then use the woodColor to colour the cube in various ways to specify the pixel colour, for example,

```
return woodColor;
```
or
```
return woodColor * lightColor;
```

Exercise 2. Texture wrapping mode

Create a texture-mapped cube using the coin texture “Coin.dds”, such that different faces of the cube has different number of coin patterns.

Exercise 3. Mipmapping

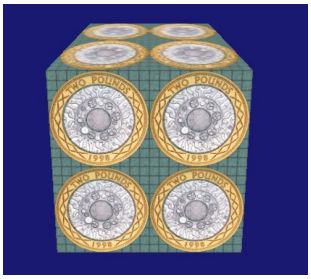
Create mipmaps of a loaded texture and use HLSL sampleLevel( ) to map the cube with different level of mipmaps.

Exercise 4. Texture filtering techniques.

Scale the cube along the view direction to create a long rectangular object. Using different filtering techniques to deal with the minification and magnification issues on texture mapping and observe how visual quality of the rendered image is being changed.

Exercise 5. Multiple texturing

Use a coin texture and a tile texture to create the following effect.



Exercise 5. An open box

Create the following open box effects using the wood.dds and rock.dds textures.