Objectives

1. To understand different techniques of generating visual bumps on geometric surfaces.
2. To understand how to transform light direction and view direction to tangent space.
3. To discover how to use normal maps.
4. To learn how to create visual bumpy effects use only height maps.
5. To learn how to implement normal mapping, height mapping and parallax mapping using HLSL shaders.

All the exercises described below can be done directly based on what you have achieved in the last two labs on lighting and texture mapping.

Exercise 1. Normal mapped bumpy 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 two new elements corresponding to the tangent and binormal vectors:

```
D3D11_INPUT_ELEMENT_DESC layout[] =
{
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TEXCOORD", 0, DXGI_FORMAT_R32G32_FLOAT, 0, 12, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 20, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "TANGENT", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 32, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    { "BINORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 44, D3D11_INPUT_PER_VERTEX_DATA, 0 },
};
```

2. In the vertex structure, add a new element “Tangent” and “Binormal”

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

3. In the vertex list, specify the values of tangent vector and binormal vectors for each vertex position. You can start with only one face of the cube, say, the top face or the front face. Once successful, you can extend the vertex list to include all six faces.

```
SimpleVertex verPosTex[] =
    {

        { XMFLOAT3( -1.0f, 1.0f, -1.0f), XMFLOAT2( 1.0f, 0.0f ),
          XMFLOAT3(0.0f, 1.0f, 0.0f),    XMFLOAT3(0.0f, 0.0f, 1.0f),
          XMFLOAT3(-1.0f, 0.0f, 0.0f) },
        … …
        … …
       }
```

4. Create and set vertex buffer to pass vertex data as input to vertex shader.
5. Load and create two texture objects: one for object’s surface colour, the other for surface’s normal vectors.
6. Bind the two texture objects to pixel shader by setting the two textures as pixel shade resources.

7. In vertex shader, calculate light direction and view direction and transform the two vectors into tangent space, then output them into pixel shder:

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

struct PS_INPUT
{
       float4 Pos : SV_POSITION;
       float2 Tex : TEXCOORD0;
       float3 viewDirInTang : TEXCOORD1;
       float3 lightDirInTang : TEXCOORD2;
};


PS_INPUT VS( VS_INPUT input )
{

       PS_INPUT output = (PS_INPUT)0;

       … …

       float3 viewDirW =  eyePosition – input.Pos;
       float3 lightDirW = lightPosition - input.Pos;

       float3 N = normalize(input.Norm);
       float3 T = normalize(input.Tang);
       float3 B = normalize(input.Binorm);

       float3x3 mat2Tang= float3x3 (T, B, N);

       output.viewDirInTang= mul( mat2Tang, viewDirW);
       output.lightDirInTang= mul(mat2Tang, lightDirW);

       … …

}
```

8. In pixel shader, specify normal vector at each pixel using a normal map and use it to calculate the diffuse light and specular light:

```
Texture2D txStoneColor : register(t0);
Texture2D txStoneBump : register(t1);
SamplerState txStoneSampler : register(s0);

… …

float4 PS( PS_INPUT input) : SV_Target
{
    float4 finalColor = 0;
    float4 stoneCol = txStoneColor.Sample(txStoneSampler, input. Tex);
    float4 stoneNormal = txStoneBump.Sample(txStoneSampler, input.Tex);
    float3 N=normalize(2.0* stoneNormal.xyz -1.0);
    … …
}
```

Exercise 2. Height map-based bumpy cube

Create a similar visual effect using a height map. Refer to my lecture notes on how the normal vector at each pixel is calculated using the HLSL intrinsic functions ddx( ) and ddy( ).

Exercise 3. Procedural mapping

Write a procedural normal map to create the following effect.

Exercise 4. Parallax mapping-based bump effect.

Improve the bump mapping techniques by using both a normal map and a height map to create high quality visual bumpy surface details.