DA307A – Datorgrafik & modellering

Rendering : Assignment 5 – Cube Mapping

Av: Robin Andblom

Lösning:

Jag började med att skapa fyra nya shader pekare, en vertex och pixelshader för skydomen och två motsvarande för reflektionsytan. Jag skapade även en shader resource view pekare samt en resource (DDS texture) pekare, samt en rasterizer state för skydomen. Efter detta deklarerade jag en ny OBJ-klass som kom att få motsvara själva skydome meshen. Jag instansierade upp detta OBJ objekt och kopplade en sfär som mesh till den, och laddade in min DDS textur genom att ange sökvägen till en cubemap dds fil, på följande vis:

```
DirectX::CreateDDSTextureFromFile(g_Device, g_DeviceContext,
L"../../assets/cubemaps/snowcube1024.dds", &SkyTex, &SkyTexSRV);
```

Med referenser till DDS textur och SRV pekarna. I metoden updateObjects i main skrev jag:

```
            MSkyDome = mat4f::scaling(100);
```

För att skala skydomens storlek (i varje frame). I renderObjects metoden sätter jag mina shaders för skydomen och reflektionsytan (kallad reflection),mappar dem mot matrixbuffern och renderar ut dem, på följande vis:

```
g_DeviceContext->PSSetShaderResources(5, 1, &SkyTexSRV);
            g_DeviceContext->RSSetState(g_RasterStateSkyDome);

            skyDome->MapMatrixBuffers(g_DeviceContext, g_MatrixBuffer, MSkyDome,
Mview, Mproj);
            skyDome->renderObj(g_DeviceContext, material_buffer);

            g_DeviceContext->VSSetShader(g_VertexShaderReflection, nullptr, 0);
            g_DeviceContext->PSSetShader(g_PixelShaderReflection, nullptr, 0);
            g_DeviceContext->RSSetState(g_RasterState);
```

Uträkningen i **skydomens vertex shader** ger:

```
struct VSIn
{
    float3 Pos : POSITION;
    float3 Normal : NORMAL;
    float3 Binormal : BINORMAL;
    float3 Tangent : TANGENT;
    float2 TexCoord : TEX;
};

struct PSIn
{
    float4 pos : SV_Position;
    float3 texCord : TEXCOORD0;
};

PSIn main( VSIn input )
{
```

```
    PSIn psin;

    matrix MV = mul(WorldToViewMatrix, ModelToWorldMatrix);
            // model-to-projection
    matrix MVP = mul(ProjectionMatrix, MV);

    psin.pos = mul(MVP, float4(input.Pos, 1));
    psin.texCord = input.Pos;

            return psin;
}
```

I koden ovan skapar jag en matris transformation som tar modellen till världen, och sen världen till kameravyn och till sist till projection space. Jag multiplicerar projection space matrisen med en ny 4 dimensionell vektor (3D-koordinat, 1).

Uträkningen i **skydomens pixel shader** ger:

```
TextureCube tex : register(t5);
sampler samp : register(s0);

float4 main(PSIn input) : SV_TARGET
{
    return tex.Sample(samp, input.texCord);
}
```

I koden ovan gör jag en texture cube och en vanlig sampler och returnerar texturen med sampler-variablen samt input textur koordinaterna.

Uträkningen i **reflektions ytans vertex shader** ger:

```
struct VSIn
{
    float3 Pos : POSITION;
    float3 Normal : NORMAL;
    float3 Binormal : BINORMAL;
    float3 Tangent : TANGENT;
    float2 TexCoord : TEX;
};

struct PSIn
{
    float4 pos : SV_Position;
    float3 worldPos : WORLDPOS;
    float3 normal : NORMAL;
};

PSIn main(VSIn input)
{
    PSIn psin;
    matrix MV = mul(WorldToViewMatrix, ModelToWorldMatrix);
            // model-to-projection
    matrix MVP = mul(ProjectionMatrix, MV);

    psin.pos = mul(MVP, float4(input.Pos, 1));
    psin.worldPos = mul(ModelToWorldMatrix, float4(input.Pos, 1)).xyz;
    psin.normal = mul(ModelToWorldMatrix, input.Normal);
    return psin;
}
```

I koden ovan transormerar jag modellen till projection space som i skydomens vertex shader, multiplicerar med en ny 4-dimensionell vektor (MVP * float4(input.Pos, 1)) igen, samt att jag skriver världens position på samma form i en modeltoworld transformation, med skillnaden att jag lägger till xyz på slutet för att kasta bort w-komponenten (vi vill ha endast ha tre dimensioner i världen). Sedan skriver jag pixel shaderns normal som modeltoworld matrisen multiplicerat med den inskickade normalen.

Uträkningen i **reflektions ytans pixel shader** ger:

```
TextureCube tex : register(t5);
sampler samp : register(s0);

float4 main(PSIn input) : SV_TARGET
{
    float3 texCord;
    float3 V = normalize(input.worldPos - camera_pos.xyz);
    texCord = reflect(V, input.normal);
    return tex.Sample(samp, texCord);
}
```

I koden ovan gör jag en 3D-textur koordinat (texCord), en vektor V = B – A där A är kamerans position i x,y,z och B är den inlästa världspositionen. Vektorn V normaliseras och används sedan i nästa steg. I nästa steg använder jag min 3D-textur koordinat och gör en reflect-operation där jag stoppar in min vektor V och den inlästa normalen. Jag returnerar sedan en textur sampling med en sampler-variabel och min 3D-textur koordinat.

Resultat: