Real Time Rendering Techniques

# 3D Scene Using DirectX

## Sky box

The sky box is made using environmental mapping methods. The sky box is made using the Box class, this creates a cube for texture mapping a coordinate to each side of the cube.

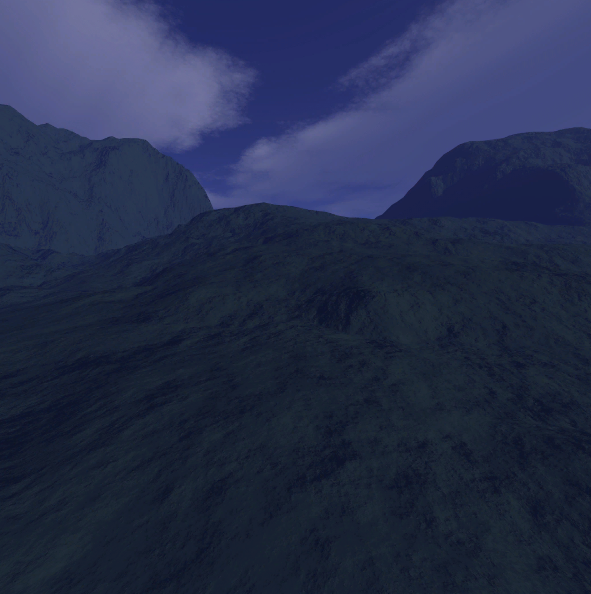


Figure 1 - Sky Box Example

To achieve the full sky box, we scale the cube to 1000 times itself, so the camera is essentially inside the box.

// The box class is derived from the BaseModel class

box = new Box(device, skyBoxEffect, NULL, 0, skyBoxTextureArray, 1);

box->setWorldMatrix(box->getWorldMatrix()\*XMMatrixScaling(1000, 1000, 1000));

box->update(context);

## Terrain

The terrain is made using the Terrain class. This class receives the device, context, height, width, normal map texture, height map texture, material array, number of materials, terrain texture array, number of textures.

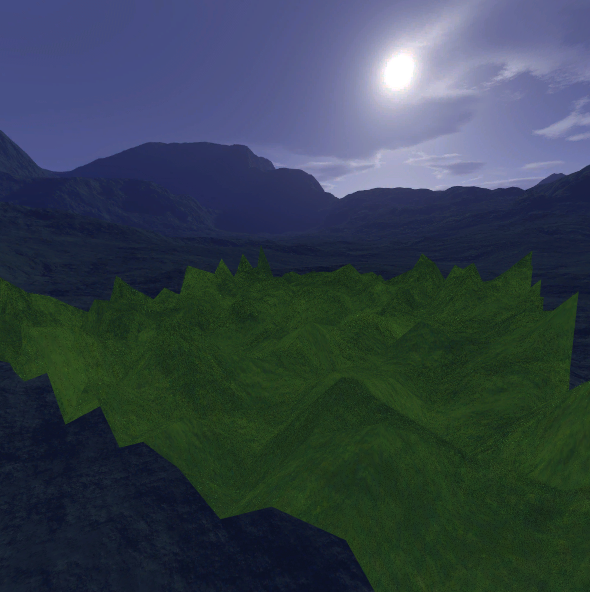


Figure 2 - Terrain Example

To see the heightmap properly the terrain y axis was scaled to 20.

//Terrain Object

ground = new Terrain(device, context, 25, 25, heightMap->getTexture(), normalMap->getTexture(), grassEffect, NULL, 0, grassTextureArray, 2);

ground->setWorldMatrix(ground->getWorldMatrix()\*XMMatrixTranslation(-3, 0, -6)\*XMMatrixScaling(4, 20, 4)\*XMMatrixRotationY(XMConvertToRadians(45)));

ground->update(context);

The terrain had a grass effect applied to it. The grass effect is generated using the Grid class. The grid then has random spots for the texture of the grass to be applied. The height of the grass is determined in the shader. The grass is then layered, like normal grass, the bottom is generally wider than the top. The grid is applied to the terrain to create grass around the terrain. The shader moves the top of the grass to a wind direction. The grass can be adjusted in the shader.



Figure 3 - Grass blowing in the wind

## Castle Model

The castle was added using the Model class. The texture is mapped in the shader by attaching the coordinates from the image file to the model. The image file is then wrapped around the model applying the perPixelLightingEffect.

//Castle Object

Material castleMat;

castleMat.setSpecular(XMCOLOR(1, 1, 1, 1));

Material \*castleMatArray[] = { &castleMat };

castle = new Model(device, wstring(L"Resources\\Models\\castle.3ds"), perPixelLightingEffect, castleMatArray, 1, castleTextureArray, 1);

castle->setWorldMatrix(castle->getWorldMatrix()\*XMMatrixTranslation(8.3, 2, 0)\*XMMatrixScaling(5, 5, 5));

castle->update(context);

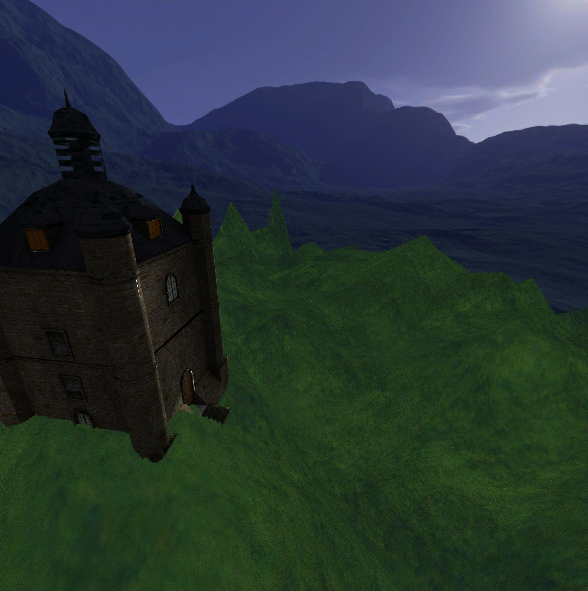


Figure 4 - The castle on the terrain

## Water

The water is achieved using the Grid class. A tile for the water texture is scaled and repeated, the ocean shaders determine how the tiles are moved, this is a repeated process until a wave effect is created. A normal map for the water is applied to make the water look realistic. The wave displacement is modified in the shader.

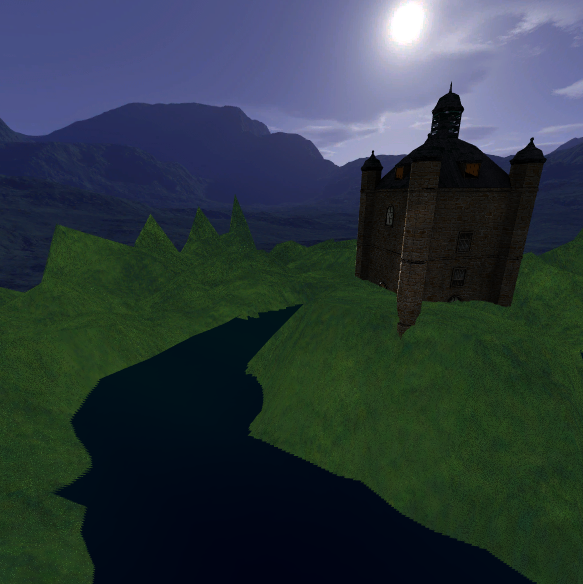


Figure 5 - Water in the moat around the castle

## Tree

The tree effect is achieved using the Model class. The model has the tree texture wrapped around it, then a group of leaves is applied from the texture to various branches on the tree. The alpha blending states are used to make transparency on the tree. The tree then is modified by the shader to give the illusion that the tree is being moved with the wind.

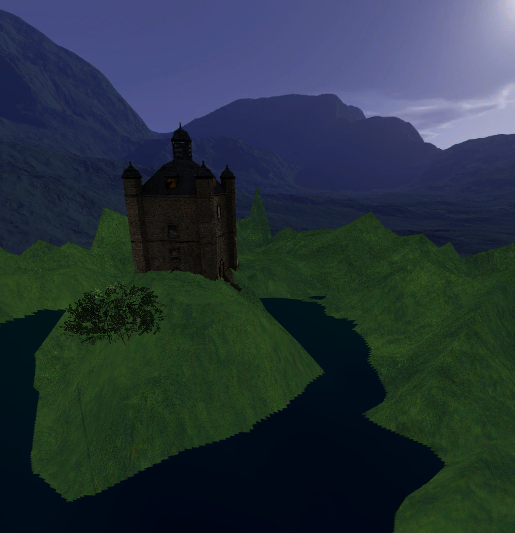


Figure 6- The tree next to the castle

## The Glowing Orb

The orb is produced using the Model class. The blurUtility is also applied using the blurUtility class. The glow is achieved by rendering a model with an increased and blurred model and applying it to the original model.

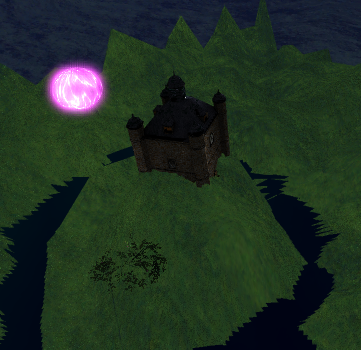


Figure 7 - A glowing orb above the castle

orb = new Model(device, wstring(L"Resources\\Models\\sphere.3ds"), orbEffect, NULL, 0, orbTextureArray, 1);

orb->setWorldMatrix(orb->getWorldMatrix()\* XMMatrixTranslation(50, 25, 0)\* XMMatrixScaling(1, 1, 1));

orb->update(context);

## Flares

The flare was achieved by drawing a line from the lightsource to the camera, therefore it can only be viewed if the light source is in the field of view. The amount of flares used can be adjusted as can the size. The texture can be modified also.

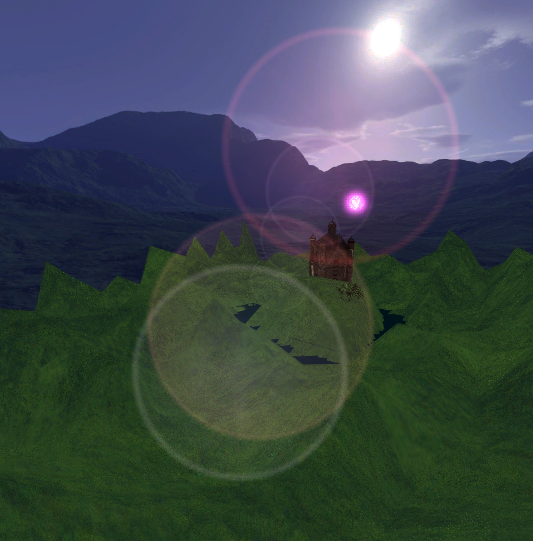


Figure 8 - The lens flares in the scene

for (int i = 0; i < numFlares; i++)

if (randM1P1() > 0)

flares[i] = new Flare(XMFLOAT3(-125.0, 60.0, 70.0), XMCOLOR(randM1P1()\*0.5 + 0.5, randM1P1()\*0.5 + 0.5, randM1P1()\*0.5 + 0.5, (float)i / numFlares), device, flareEffect, NULL, 0, flare1TextureArray, 1);

else

flares[i] = new Flare(XMFLOAT3(-125.0, 60.0, 70.0), XMCOLOR(randM1P1()\*0.5 + 0.5, randM1P1()\*0.5 + 0.5, randM1P1()\*0.5 + 0.5, (float)i / numFlares), device, flareEffect, NULL, 0, flare2TextureArray, 1);