Advanced Graphics Programming

Report

Dylan Gijsbertsen
Studentnumber: 500687199
Class: IVGT1
Assessor: Richard de Koning
3-4-2016

Inhoud

Assignment 1 Primitives

I created a sphere with the GeometryGenerator that will change at run time when the user press a key.

```
GeometryGenerator::MeshData sphere;

GeometryGenerator geoGen;
geoGen.CreateSphere(0.5f, dynamicVertex, dynamicVertex, sphere);
```

The createSpere method needs four parameters to create one. The first is the radius of how big the sphere has to be, second and third is for the indices and vertices, I created for this a variable called dynamicVertex, with this variable I will change the indices and vertices at run time. And the last parameter is the mesh for a sphere.

Next I have two variables for the vertex and index offset from the sphere. When the sphere is going to be drawn it will know where to start with the index and vertex of this object. These values are now zero because there are no other objects that is going to been drawn.

```
// Cache the vertex offsets to each object in the concatenated vertex buffer.
mSphereVertexOffset = 0;

// Cache the index count of each object.
mSphereIndexCount = sphere.Indices.size();

// Cache the starting index for each object in the concatenated index buffer.
mSphereIndexOffset = 0;

UINT totalVertexCount = sphere.Vertices.size();

UINT totalIndexCount = mSphereIndexCount;
```

I also need the total amount of indices and vertices for the index buffer and vertex buffer, so the sphere can be drawn.

```
std::vector<UINT> indices;
indices.insert(indices.end(), sphere.Indices.begin(), sphere.Indices.end())

D3D11_BUFFER_DESC ibd;
ibd.Usage = D3D11_USAGE_IMMUTABLE;
ibd.ByteWidth = sizeof(UINT) * totalIndexCount;
ibd.BindFlags = D3D11_BIND_INDEX_BUFFER;
ibd.CPUAccessFlags = 0;
ibd.MiscFlags = 0;
D3D11_SUBRESOURCE_DATA iinitData;
iinitData.pSysMem = &indices[0];
HR(md3dDevice->CreateBuffer(&ibd, &iinitData, &mIB));
```

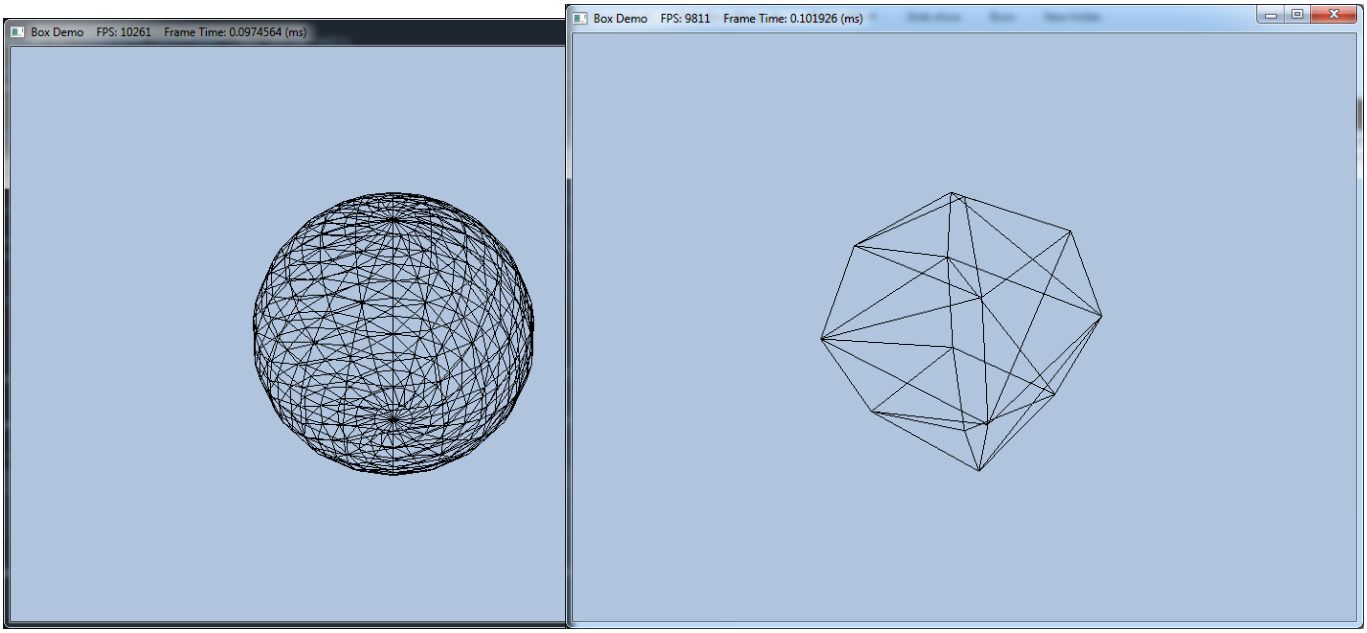
```
D3D11_BUFFER_DESC vbd;
vbd.Usage = D3D11_USAGE_IMMUTABLE;
vbd.ByteWidth = sizeof(Vertex) * totalVertexCount;
vbd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
vbd.CPUAccessFlags = 0;
vbd.MiscFlags = 0;
D3D11_SUBRESOURCE_DATA vinitData;
vinitData.pSysMem = &vertices[0];
HR(md3dDevice->CreateBuffer(&vbd, &vinitData, &mVB));
```

I use here the Immutable which means I use the GPU to read my buffer.

Resource Usage	Default	Dynamic	Immutable	Staging
GPU-Read	yes	yes	yes	yes[1]
GPU-Write	yes			yes[1]
CPU-Read				yes[1]
CPU-Write		yes		yes[1]

```
if (GetAsyncKeyState('A'))
{
    dynamicVertex -= .1f;
    BuildGeometryBuffers();
}
else if (GetAsyncKeyState('D'))
{
    dynamicVertex += .1f;
    BuildGeometryBuffers();
}

if (dynamicVertex <= 5)
    dynamicVertex = 5;
```

In the update method I created these simple if statements to change the sphere. If the user press the 'A' key the sphere will get less indices and vertices and if the user press 'D' the sphere will get more indices and vertices.



Assignment 3 Blending & Stenciling

I created a couple of triangles and use the blending modes.

```
//
// TransparentBS
//

D3D11_BLEND_DESC transparentDesc = { 0 };
transparentDesc.AlphaToCoverageEnable = false;
transparentDesc.IndependentBlendEnable = false;

transparentDesc.RenderTarget[0].BlendEnable = true;
transparentDesc.RenderTarget[0].SrcBlend = D3D11_BLEND_SRC_ALPHA;
transparentDesc.RenderTarget[0].DestBlend = D3D11_BLEND_INV_SRC_ALPHA;
transparentDesc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
transparentDesc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
transparentDesc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
transparentDesc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
transparentDesc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;

HR(device->CreateBlendState(&transparentDesc, &AlphaBlending));

//
// AdditiveBlending
//

D3D11_BLEND_DESC AdditiveBlendingDesc = { 0 };
AdditiveBlendingDesc.AlphaToCoverageEnable = false;
AdditiveBlendingDesc.IndependentBlendEnable = false;

AdditiveBlendingDesc.RenderTarget[0].BlendEnable = true;
AdditiveBlendingDesc.RenderTarget[0].SrcBlend = D3D11_BLEND_ONE;
AdditiveBlendingDesc.RenderTarget[0].DestBlend = D3D11_BLEND_ONE;
AdditiveBlendingDesc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
AdditiveBlendingDesc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
AdditiveBlendingDesc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
AdditiveBlendingDesc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
AdditiveBlendingDesc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;

HR(device->CreateBlendState(&transparentDesc, &AdditiveBlending));
```

In the picture above and on the next page you see the 4 different blend modes.
To get these different modes you should change the srcBlend and DestBlend. And I did with help from the picture below.

Blend type	Blend settings
Alpha Blending	(*source* × Blend.SourceAlpha) + (*destination* × Blend.InvSourceAlpha)
Additive Blending	(*source* × Blend.One) + (*destination* × Blend.One)
Multiplicative Blending	(*source* × Blend.Zero) + (*destination* × Blend.SourceColor)
2X Multiplicative Blending	(*source* × Blend.DestinationColor) + (*destination* × Blend.SourceColor)

```
//
// MultiplicativeBlending
//

D3D11_BLEND_DESC MultiplicativeBlendingDesc = { 0 };
MultiplicativeBlendingDesc.AlphaToCoverageEnable = false;
MultiplicativeBlendingDesc.IndependentBlendEnable = false;

MultiplicativeBlendingDesc.RenderTarget[0].BlendEnable = true;
MultiplicativeBlendingDesc.RenderTarget[0].SrcBlend = D3D11_BLEND_ZERO;
MultiplicativeBlendingDesc.RenderTarget[0].DestBlend = D3D11_BLEND_SRC_COLOR;
MultiplicativeBlendingDesc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
MultiplicativeBlendingDesc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
MultiplicativeBlendingDesc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
MultiplicativeBlendingDesc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
MultiplicativeBlendingDesc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;

HR(device->CreateBlendState(&transparentDesc, &MultiplicativeBlending));

//
// MultiplicativeBlending X2
//

D3D11_BLEND_DESC MultiplicativeBlendingX2Desc = { 0 };
MultiplicativeBlendingX2Desc.AlphaToCoverageEnable = false;
MultiplicativeBlendingX2Desc.IndependentBlendEnable = false;

MultiplicativeBlendingX2Desc.RenderTarget[0].BlendEnable = true;
MultiplicativeBlendingX2Desc.RenderTarget[0].SrcBlend = D3D11_BLEND_DEST_COLOR;
MultiplicativeBlendingX2Desc.RenderTarget[0].DestBlend = D3D11_BLEND_SRC_COLOR;
MultiplicativeBlendingX2Desc.RenderTarget[0].BlendOp = D3D11_BLEND_OP_ADD;
MultiplicativeBlendingX2Desc.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
MultiplicativeBlendingX2Desc.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ZERO;
MultiplicativeBlendingX2Desc.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_ADD;
MultiplicativeBlendingX2Desc.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;

HR(device->CreateBlendState(&transparentDesc, &MultiplicativeBlendingX2));
```

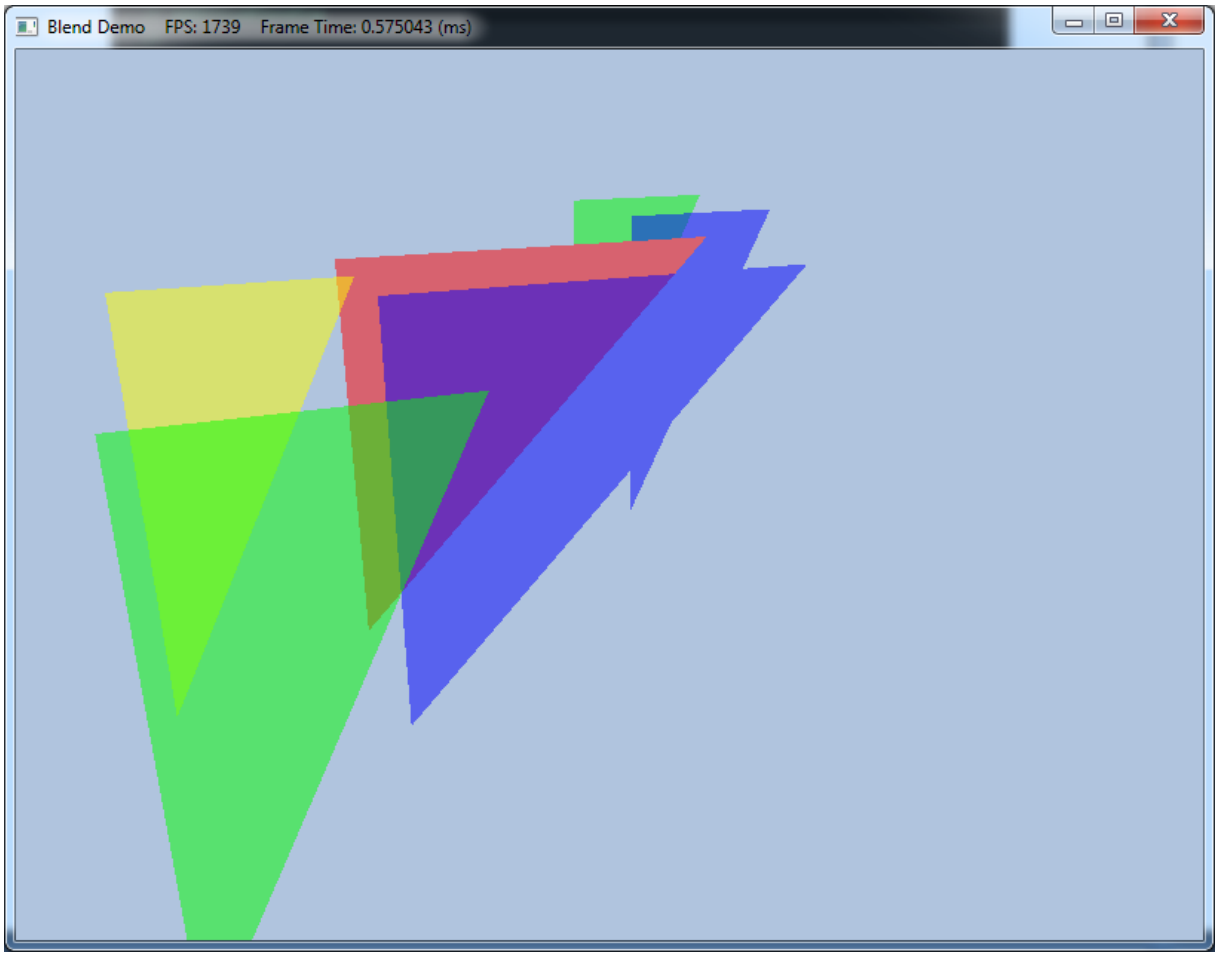
I call this blend types in the draw function with the OMSetBlendState method.

```
ID3DX11EffectPass* pass = mTech->GetPassByIndex(p);

mTech->GetPassByIndex(p)->Apply(0, md3dImmediateContext);

md3dImmediateContext->OMSetBlendState(RenderStates::AlphaBlending, blendFactor, 0xffffffff);
//md3dImmediateContext->OMSetBlendState(RenderStates::AdditiveBlending, blendFactor, 0xffffffff);
//md3dImmediateContext->OMSetBlendState(RenderStates::MultiplicativeBlending, blendFactor, 0xffffffff);
//md3dImmediateContext->OMSetBlendState(RenderStates::MultiplicativeBlendingX2, blendFactor, 0xffffffff);

pass->Apply(0, md3dImmediateContext);
```

The alpha blending technique allows for the combination of two colors allowing for a transparency effect so you could make something like glass or water.

Additive blending will give you in result a brighter image.

Multiplicative blending makes the images thicker.



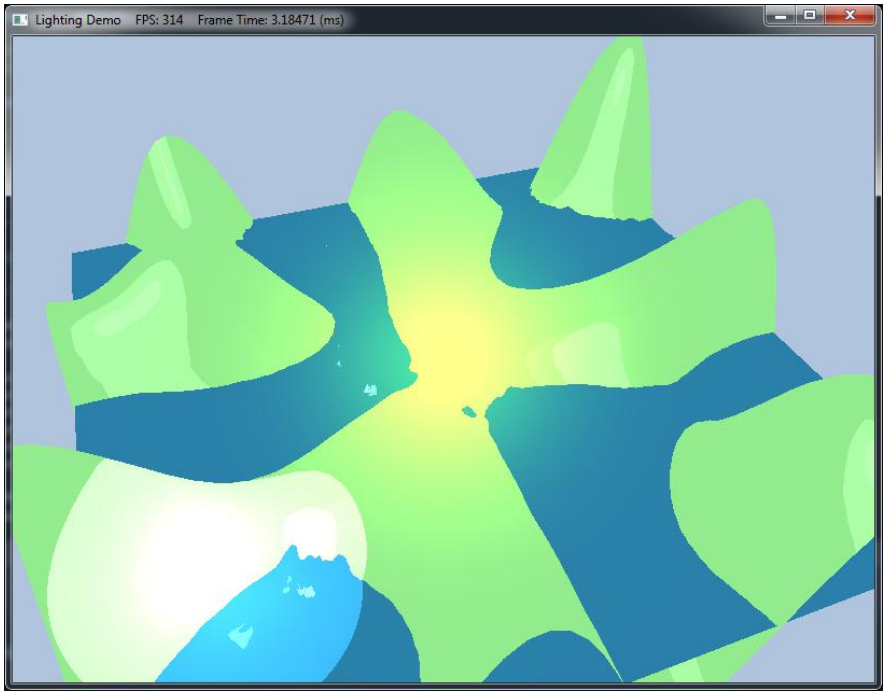
This is the result of the blending assignment.

Assignment 4 Lighting

For this assignment I use toon shading by applying these if statements below.

```
float diffuseFactor = dot(lightVec, normal);

if (diffuseFactor > -10000 && diffuseFactor <= 0.0f)
    diffuseFactor = 0.4f;
if (diffuseFactor > 0.0f && diffuseFactor <= 0.5f)
    diffuseFactor = 0.6f;
if (diffuseFactor > 0.5f && diffuseFactor <= 1.0f)
    diffuseFactor = 1.0f;

// Flatten to avoid dynamic branching.
[flatten]
if (diffuseFactor > 0.0f)
{
    float3 v = reflect(-lightVec, normal);
    float specFactor = pow(max(dot(v, toEye), 0.0f), mat.Specular.w);

    if (specFactor >= 0.0f &&specFactor <= 0.1f)
        specFactor = 0.0f;
    if (specFactor > 0.1f &&specFactor <= 0.8f)
        specFactor = 0.5f;
    if (specFactor > 0.8f &&specFactor <= 1.0f)
        specFactor = 0.8f;

    diffuse = diffuseFactor * mat.Diffuse * L.Diffuse;
    spec = specFactor * mat.Specular * L.Specular;
}
```

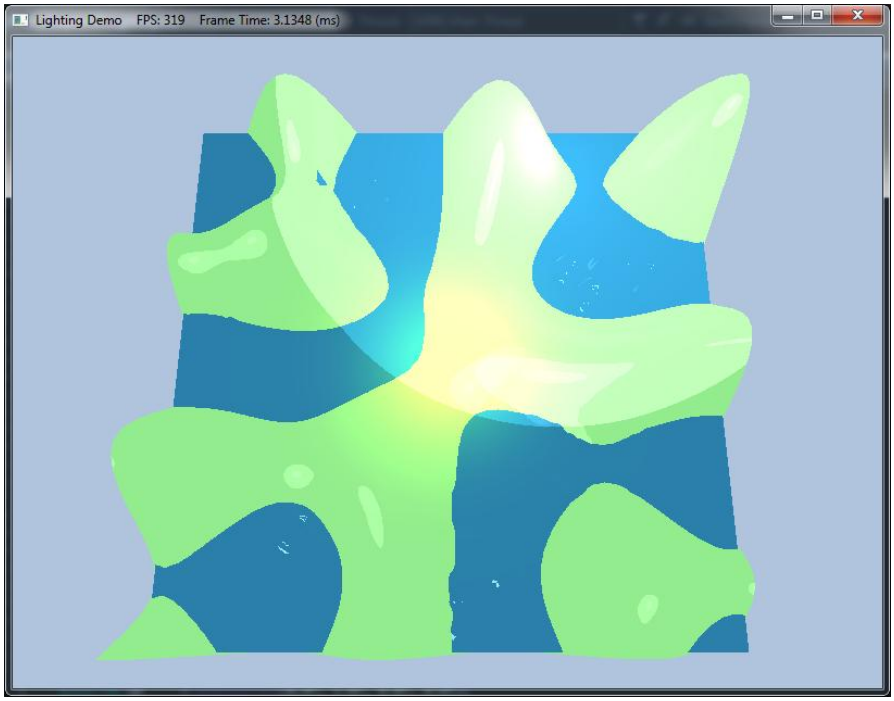
I applied these six if statements to all three lights (Directional, spot and point light). This will lead to the result below.



Beside the toon shading I also created that if the user uses the 'W' or 'S' key the spot and point light will increase or decrease.

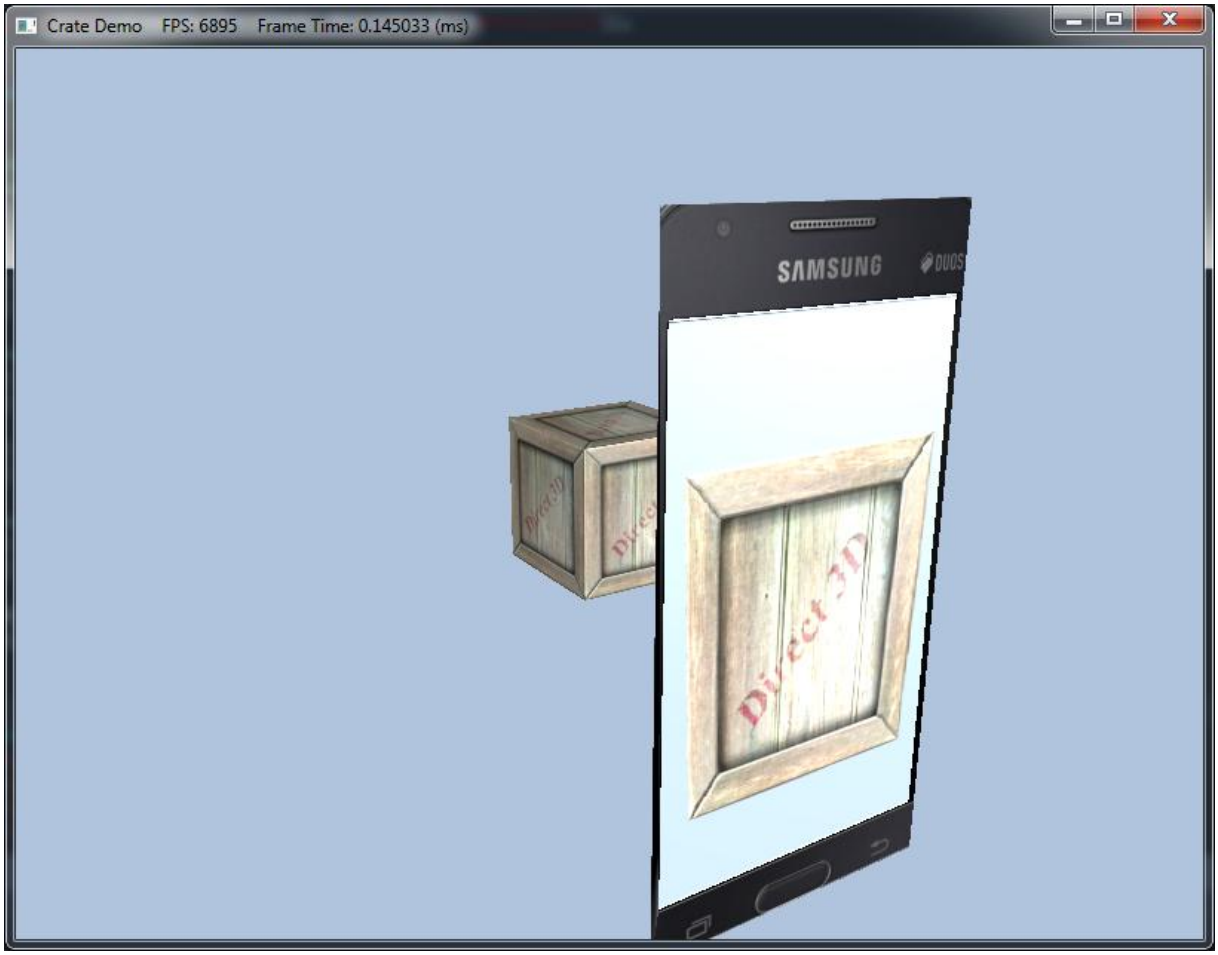
```
void LightingApp::UpdateScene(float dt)
{

    if (GetAsyncKeyState('W'))
    {
        mSpotLight.Spot += 2;
        mPointLight.Range += 1;
    }
    else if (GetAsyncKeyState('S'))
    {
        mSpotLight.Spot -= 2;
        mPointLight.Range -= 1;
    }
```

and see result.



Assignment 5 Texturing

For this assignment I created a render a mobile phone with a texture in it's display.



I did this by creating my phone with a Basic32 array. I set hard coded values for the position, normals and UV's coordinates. See next page.

```
void CrateApp::BuildPhoneGeometryBuffers()
{
    Vertex::Basic32 v[12];
    // phone
    // back
    v[0] = Vertex::Basic32(+1.0f, -2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f);
    v[1] = Vertex::Basic32(+1.0f, 2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f);

    v[2] = Vertex::Basic32(-1.0f, 2.0f, -2.0f, -1.0f, 0.0f, 0.0f, .69f, 0.0f);
    v[3] = Vertex::Basic32(-1.0f, -2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 0.69f, 1.0f);

    //front
    v[4] = Vertex::Basic32(+1.0f, 2.0f, -2.0f, -1.0f, 0.0f, 0.0f, .33f, 0.0f);
    v[5] = Vertex::Basic32(+1.0f, -2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 0.33f, 1.0f);

    v[6] = Vertex::Basic32(-1.0f, -2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f);
    v[7] = Vertex::Basic32(-1.0f, 2.0f, -2.0f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f);

    // screen
    v[8] = Vertex::Basic32(-.95f, -1.65f, -2.0001f, 0.0f, 0.0f, -1.0f, .35f, 1.0f);
    v[9] = Vertex::Basic32(-.95f, 1.5f, -2.0001f, 0.0f, 0.0f, -1.0f, .35f, 0.0f);

    v[10] = Vertex::Basic32(+.95f, 1.5f, -2.0001f, 0.0f, 0.0f, -1.0f, 0.66f, 0.0f);
    v[11] = Vertex::Basic32(+.95f, -1.65f, -2.0001f, 0.0f, 0.0f, -1.0f, 0.66f, 1.0f);



    D3D11_BUFFER_DESC vbd;
    vbd.Usage = D3D11_USAGE_IMMUTABLE;
    vbd.ByteWidth = sizeof(Vertex::Basic32) * 12;
    vbd.BindFlags = D3D11_BIND_VERTEX_BUFFER;
    vbd.CPUAccessFlags = 0;
    vbd.MiscFlags = 0;
    D3D11_SUBRESOURCE_DATA vinitData;
    vinitData.pSysMem = v;
    HR(md3dDevice->CreateBuffer(&vbd, &vinitData, &mPhoneCaseVB));
}
```

For the UV coordinates I use this 1 texture I created. It is a simple textures so hard coding the UV's is not very difficult. The coords is between 0 and 1 so three different pictures would be a third.