Hogeschool van Amsterdam

Game Technology
Advanced Graphics Programming

Titelblad

Studentnaam	**:**	**Jan-Willem Jozic**
Studentnummer	**:**	**500623980**
Plaats	**:**	**Hoofddorp**
Datum van uitgave	**:**	**3-4-2016**
Opleiding instituut	**:**	**Hogeschool van Amsterdam**

Introduction

This report details the solutions for the Advanced Graphics Programming exercises

It will explain some crucial parts of the code used to complete the tasks. It motivates, how each chapter criteria is met and why.

Primitives

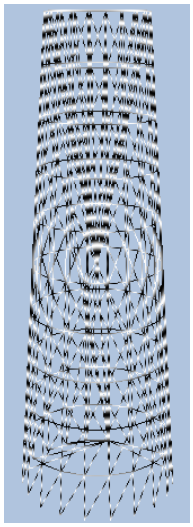
For rendering a Dynamic cylinder I applied a Geometry Generator that I then generated into the world

```
//Places the Cylinder into the world with 5 levels of layers
for (int i = 0; i < 5; ++i)
{
	world = XMLoadFloat4x4(&mCylWorld[i]);
	mfxWorldViewProj->SetMatrix(reinterpret_cast<float*>(&(world*viewProj)));
	mTech->GetPassByIndex(p)->Apply(0, md3dImmediateContext);
	md3dImmediateContext->DrawIndexed(mCylinderIndexCount, mCylinderIndexOffset,
mCylinderVertexOffset);
}



//The Geometry generator which creates a cylinder shape with the given XYZ sizes,
including how many per layer
GeometryGenerator::MeshData cylinder;

GeometryGenerator geoGen;

geoGen.CreateCylinder(0.5f, 0.3f, 3.0f, 20, 10, cylinder);

mCylinderIndexCount = cylinder.Indices.size();

UINT totalVertexCount = cylinder.Vertices.size();

UINT totalIndexCount = mCylinderIndexCount;




//This foor loop generates the position of the vertices and the color they apply
for (size_t i = 0; i < cylinder.Vertices.size(); ++i, ++k)
{
	vertices[k].Pos = cylinder.Vertices[i].Position;
	vertices[k].Color = white;
}
```

The intermediate level for this chapter was to generate a Cylinder that can have the height defined by an N-value I have done that in this case and would deserve 8 points for this criteria.

The final display isn't in white because this same project part is used for the 6th Chapter: Shading

The result can generate this Cylinder / Cone that can go on indefinitely, the vertices can be seen clearly at the bottom.



Models

<Due to time constraints and difficulty I was unable to make any progress on this exercise>

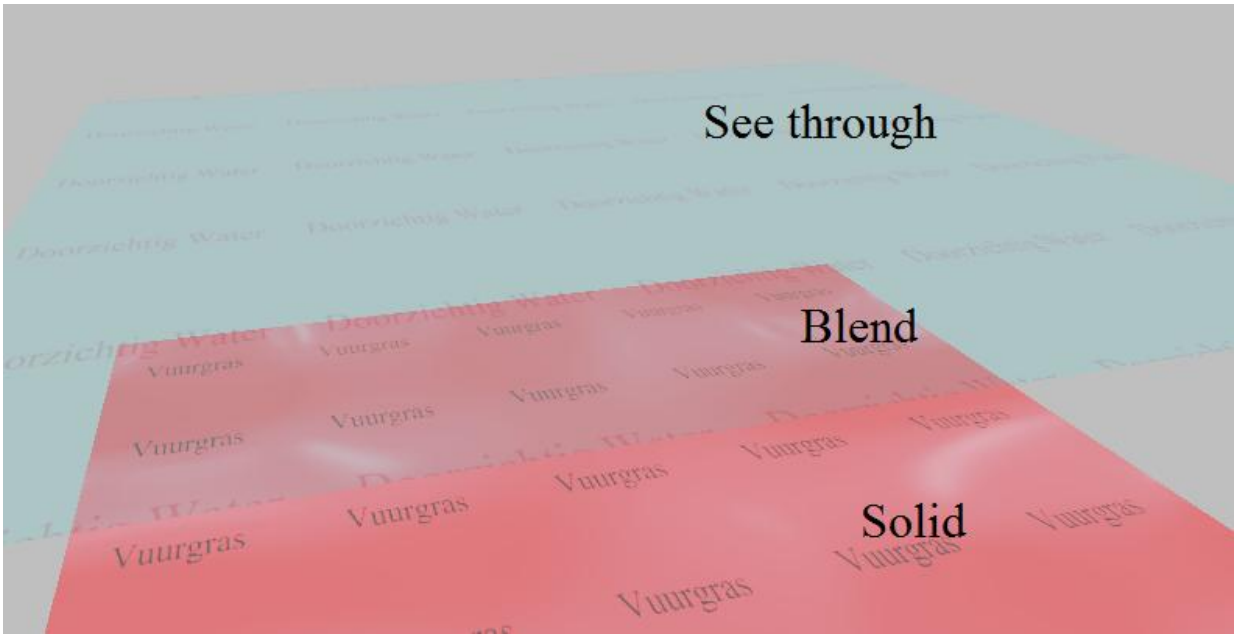
Blending

In a similar fashion to Chapter ones creation of shapes using a Geometry Generator,

In the Blending chapter I have made use of two Grid-shapes (Planes).

```
//Generates the Plane within the geometry generator
GeometryGenerator::MeshData grid;

GeometryGenerator geoGen;

geoGen.CreateGrid(20.0f, 200.0f, 30, 60, grid);


//Lowers the planes y position to place it below the first plane

//Altering the last float value between 0.0f and 1.0f sets the percentage of alpha
coloring, going below 1.0f makes the plane see through
mLandMat.Diffuse = XMFLOAT4(1.0f, 1.0f, 1.0f, 0.5f);

mWavesMat.Diffuse = XMFLOAT4(1.0f, 1.0f, 1.0f, 0.1f);
```

By lowering the alpha value below 1.0f the planes can become see through, the result is shown below:



This would fulfill the intermediate requirement of Blending, grading an 8.

Lighting

By altering the diffuse factor and specular factor within the LightHelper.fx file to match the given conditions in the Programming with DirectX 11 book:

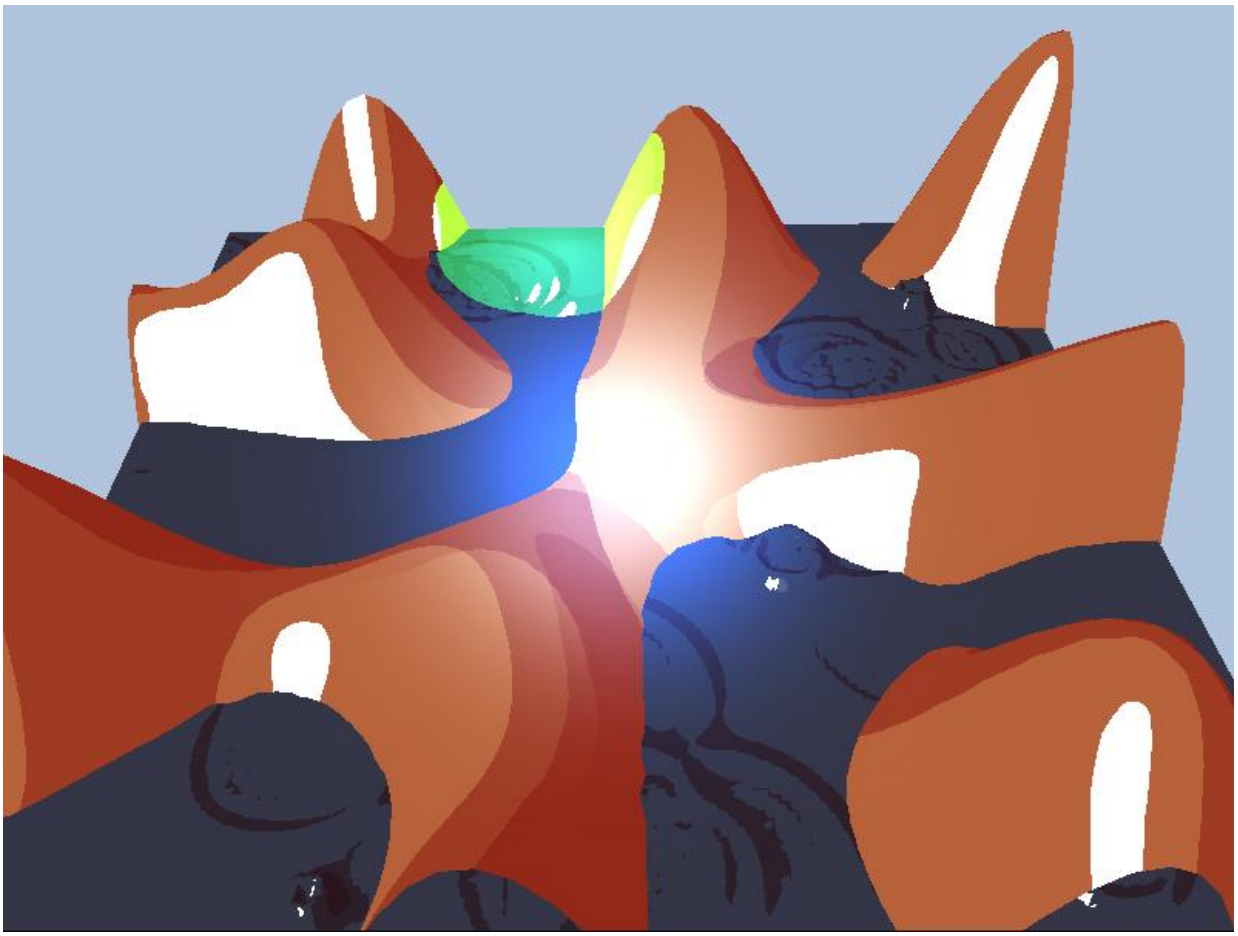
$$k_d' = f(k_d) = \begin{cases} 0.4 & \text{if } -\infty < k_d \le 0.0 \\ 0.6 & \text{if } 0.0 < k_d \le 0.5 \\ 1.0 & \text{if } 0.5 < k_d \le 1.0 \end{cases}$$

$$k_s' = g(k_s) = \begin{cases} 0.0 & \text{if } 0.0 \le k_s \le 0.1 \\ 0.5 & \text{if } 0.1 < k_s \le 0.8 \\ 0.8 & \text{if } 0.8 < k_s \le 1.0 \end{cases}$$

A cartoon lighting effect can be generated

```
float diffuseFactor = dot(lightVec, normal);
	if (diffuseFactor <= 0.0f&& diffuseFactor > -150000000000000000.0f){
		diffuseFactor = 0.4f;
	}
	else if (diffuseFactor <= 0.5f&& diffuseFactor > 0.0f){
			diffuseFactor = 0.6f;
	}
	else if (diffuseFactor <= 1.0f&& diffuseFactor > 0.5f){
				diffuseFactor = 1.0f;
	}

[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;
	}
	else if (specFactor >= 0.1f && specFactor <= 0.8f)
	{
		specFactor = 0.5f;
	}
	else if (specFactor >= 0.8f && specFactor <= 1.0f)
	{
		specFactor = 0.8f;
	}

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

The result with a bit of unrelated alterations of the lighting colors would generate a cartoon shader as seen in the following image:



This would fulfill the advanced category of the Lighting Criteria, giving the full 10 points.

Texturing

After generating a box with a texture like in the Luna example, I created a plane to place on top of the object with a different texture like in the intermediate criteria.

```
//Imports the given texture
HR(D3DX11CreateShaderResourceViewFromFile(md3dDevice,
	L"Textures/Gooball.dds", 0, 0, &mOpplakTexSRV, 0));

HR(D3DX11CreateShaderResourceViewFromFile(md3dDevice,
	L"Textures/Phone.dds", 0, 0, &mDiffuseMapSRV, 0));

TexTech->GetDesc(&techDesc);
	for (UINT p = 0; p < techDesc.Passes; ++p)
	{
		md3dImmediateContext->IASetVertexBuffers(0, 1, &mGridVB, &stride,
&offset);
		md3dImmediateContext->IASetIndexBuffer(mGridIB, DXGI_FORMAT_R32_UINT, 0);

		// Set per object constants.
		XMMATRIX world = XMLoadFloat4x4(&mBoxWorld);
		XMMATRIX worldInvTranspose = MathHelper::InverseTranspose(world);
		XMMATRIX worldViewProj = world*view*proj;

		Effects::BasicFX->SetWorld(world);
		Effects::BasicFX->SetWorldInvTranspose(worldInvTranspose);
		Effects::BasicFX->SetWorldViewProj(worldViewProj);
		Effects::BasicFX->SetTexTransform(XMLoadFloat4x4(&mTexTransform));
		Effects::BasicFX->SetMaterial(mBoxMat);
		Effects::BasicFX->SetDiffuseMap(mOpplakTexSRV);

		TexTech->GetPassByIndex(p)->Apply(0, md3dImmediateContext);
		md3dImmediateContext->DrawIndexed(mTexGridCount, 0, 0);
	}

UINT k = 0;
for(size_t i = 0; i < box.Vertices.size(); ++i, ++k)
{
	vertices[k].Pos    = box.Vertices[i].Position;
	vertices[k].Normal = box.Vertices[i].Normal;
	vertices[k].Tex    = box.Vertices[i].TexC;
}

//specifies the part of the texture to display via the given vertices
//Voorkant
vertices[2].Tex.x = 0.5f;
vertices[3].Tex.x = 0.5f;

//Achterkant
vertices[4].Tex.x = 0.5f;
vertices[7].Tex.x = 0.5f;

//Bovenkant
vertices[8].Tex.x = 0.5f;
vertices[9].Tex.x = 0.5f;

//Onderkant
vertices[12].Tex.x = 0.5f;
vertices[15].Tex.x = 0.5f;

//Rechterkant
```

```
vertices[18].Tex.x = 0.5f;
vertices[19].Tex.x = 0.5f;

//Linkerkant
vertices[22].Tex.x = 0.5f;
vertices[23].Tex.x = 0.5f;
```

This would fulfill the intermediate category of the Texturing Criteria, giving 8 points.

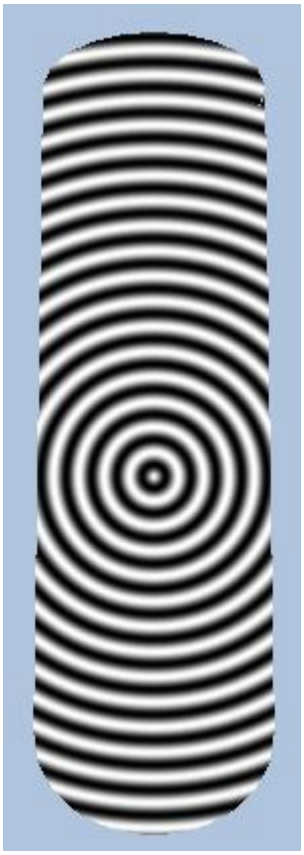
Shading

```
float4 PS(float4 pixCoords:SV_POSITION) : SV_Target
{
	float circleX = 800.0f / 2.0;
	float circleY = 600.0f / 2.0;
	float deltaX = pixCoords.x - circleX;
	float deltaY = pixCoords.y - circleY;
	float dist = sqrt(deltaX*deltaX + deltaY*deltaY);
	float light = 0.5*cos(dist*0.5 - gTime*80.0) + 0.5;



	return float4(light, light, light, 1.0);
}
```

Adding this to the Color.fx code and including

```
mfxTimeVar->SetFloat(mTimer.TotalTime());
```

This would fulfill the Moderate category of the Texturing Criteria, giving 5 points.



End Result

The end result would end up with:

Category 1: 8	= 8
Category 2: 0	= 0
Category 3: 8	= 8
Category 4: 10 * 2	= 20
Category 5: 8 * 2	= 16
Category 6: 5 * 2	= 10
Total	= 62

Giving the final grade a 6.2