Advanced Graphics Programming

Report

Dylan Gijsbertsen
Studentnumber: 500687199
Class: IVGT1
Assessor: Richard de Koning
8-5-2016

Content

Assignment 1 Primitives

I created two methods to generate a cylinder by looking at the GeometryGenerator.cpp file and take from there code that was needed to generate a cylinder. The methods are GenerateCylinder and GenerateCylinderCap. First it has to create two caps (circles) before to connect them to one another.

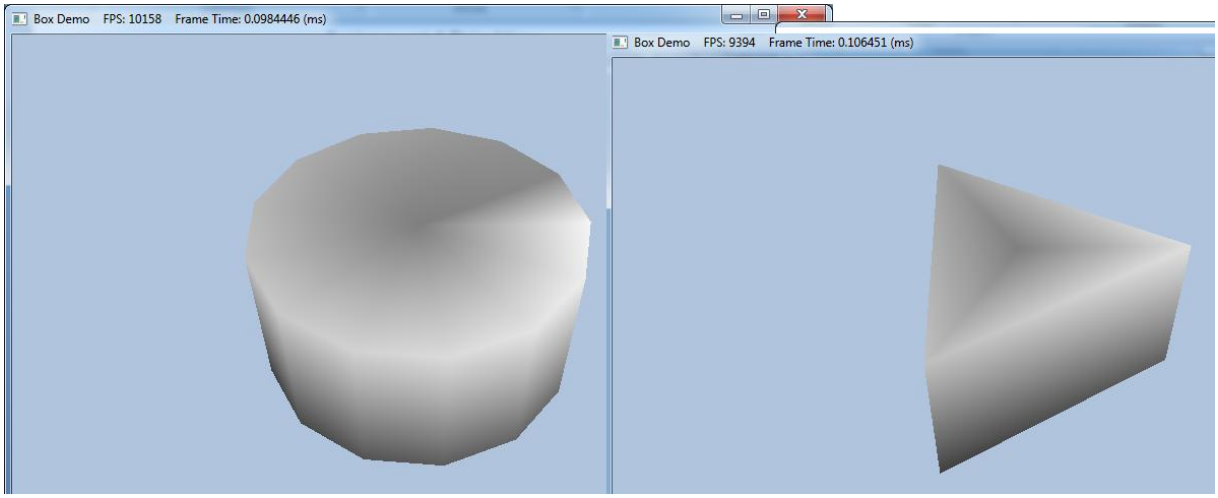
```
void GenerateCylinder(GeometryGenerator::MeshData &meshData, int sliceCount, float radius, float height, float yPos)
{
    GenerateCylinderCap(meshData, sliceCount, radius, yPos);
    GenerateCylinderCap(meshData, sliceCount, radius, height);
    int lastVertex = 1;
    for (size_t i = 0; i < sliceCount; i++)
    {
        meshData.Indices.push_back(lastVertex);
        meshData.Indices.push_back(lastVertex + sliceCount + 1);
        if (i == sliceCount - 1) {
            meshData.Indices.push_back(1);
        }
        else {
            meshData.Indices.push_back(lastVertex + 1);
        }
        lastVertex++;
    }
    lastVertex = sliceCount + 2;
    for (size_t i = 0; i < sliceCount; i++)
    {
        meshData.Indices.push_back(lastVertex);
        if (i == sliceCount - 1) {
            meshData.Indices.push_back(sliceCount + 2);
            meshData.Indices.push_back(lastVertex - (sliceCount * 2));

        }
        else {
            meshData.Indices.push_back(lastVertex + 1);
            meshData.Indices.push_back(lastVertex - sliceCount);
        }
        lastVertex++;
    }
}
```

```
void GenerateCylinderCap(GeometryGenerator::MeshData &meshData, int sliceCount, float radius, float yPos)
{
    float theta = 2.0f*XM_PI / sliceCount;
    for (size_t i = 0; i < sliceCount + 1; i++)
    {
        GeometryGenerator::Vertex vertex;
        if (i == 0) {
            vertex.Position = XMFLOAT3(0.0f, yPos, 0.0f);
        }
        else {
            vertex.Position = XMFLOAT3(radius * cosf(i*theta), yPos, radius * sinf(i*theta));
        }
        meshData.Vertices.push_back(vertex);
    }

    int lastVertex = (meshData.Vertices.size() - sliceCount);
    for (size_t i = 0; i < sliceCount; i++)
    {
        meshData.Indices.push_back(lastVertex);
        meshData.Indices.push_back((meshData.Vertices.size() - sliceCount) - 1);
        if (i == sliceCount - 1) {
            meshData.Indices.push_back((meshData.Vertices.size() - sliceCount));
        }
        else {
            meshData.Indices.push_back(lastVertex + 1);
        }
        lastVertex++;
    }
}
```

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]

I used simple if statements to make the cylinder more dynamic, by pressing 'Q' or 'E' it will have more or less vertices. See the result below.



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.

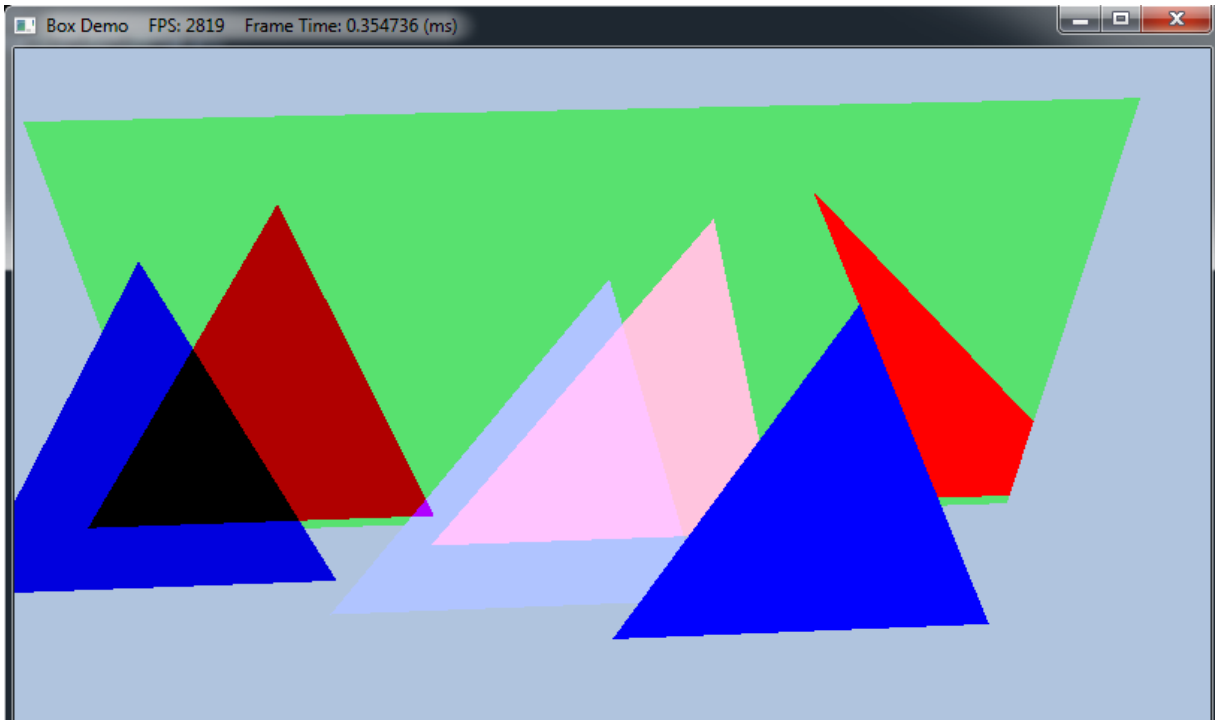
```
ID3DX11EffectPass* pass = mTech->GetPassByIndex(p);

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.

In the picture below you see all the blending techniques and stenciling. For this I created a Quadviewer that is back to the buffer and a triangle that reflects to the quadviewer. Just like in the mirror project.



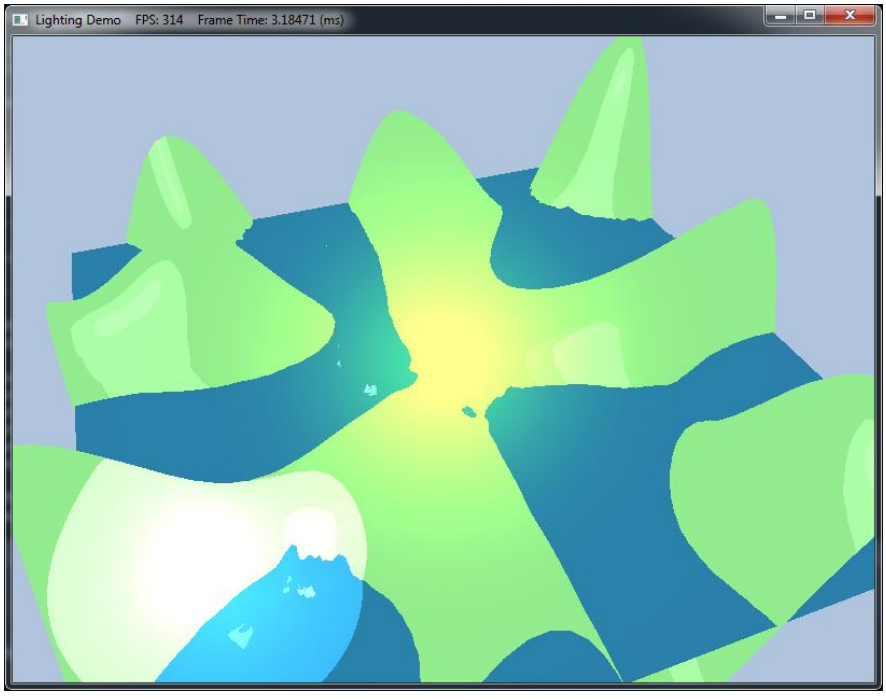
This is the result of the blending assignment.

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;
}
```

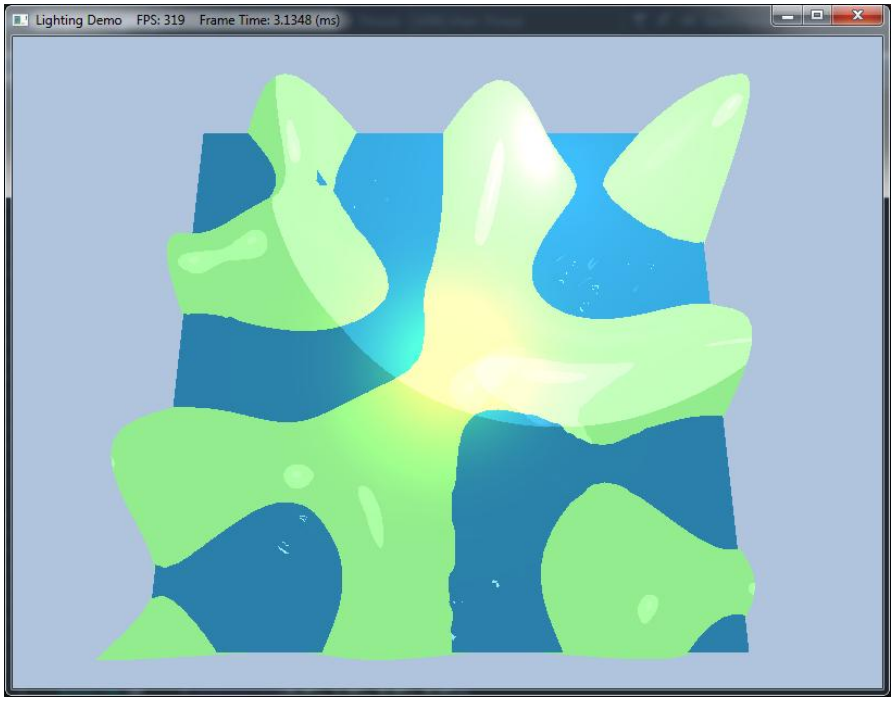
I applied these six if statements to all three lights (Directional, spot and point light). This will lead to the result below.



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.

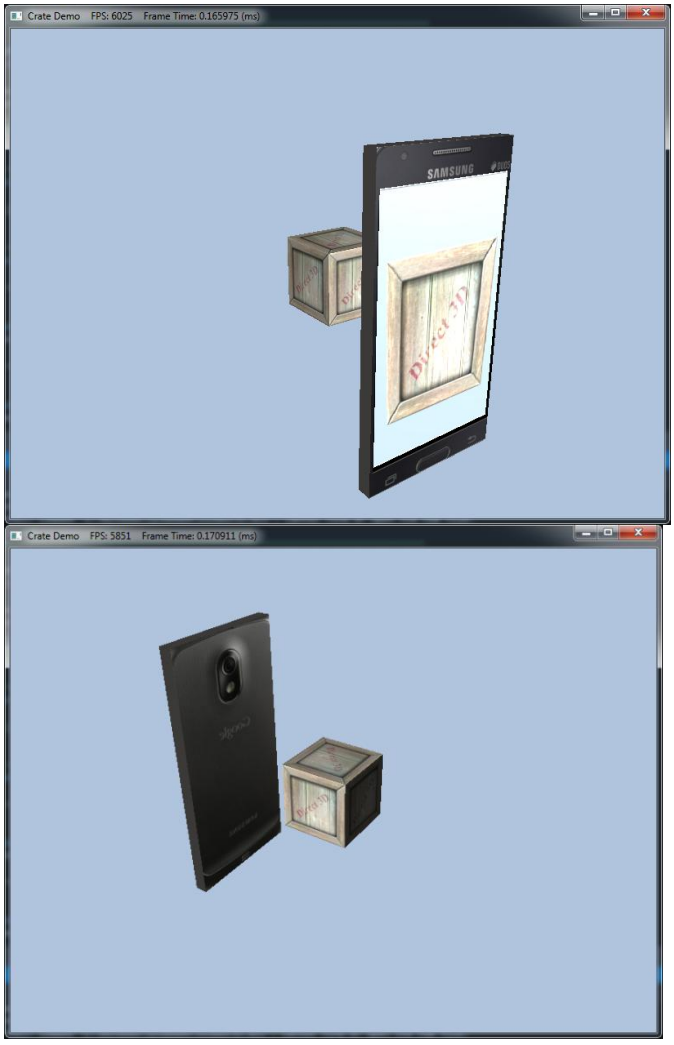
```
void LightingApp::UpdateScene(float dt)
{

    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 and render a mobile phone with a texture in its display.



First I created the mobile phone by setting all the vertices myself hardcoded, but know I used the GeometryGenerator to create the mobile phone and the screen of the phone, and to change the texture UV's I grabbed the specific vertices and set them to the right value to get the right part of the texture.

```
geoGen.CreateBox(1.9f, 3.2f, 0.1f, phoneScreen);
mphoneScreenVertexOffset = 0;
mPhoneScreenIndexCount = phoneScreen.Indices.size();
mPhoneScreenIndexOffset = 0;
UINT totalPhoneScreenVertexCount = phoneScreen.Vertices.size();
UINT totalPhoneScreenIndexCount = mPhoneScreenIndexCount;

std::vector<Vertex::Basic32> vertices3(totalPhoneScreenVertexCount);
UINT j = 0;

for (size_t i = 0; i < phoneScreen.Vertices.size(); ++i, ++j)
{
    //edit: change position of the phone
    XMFLOAT3 p = phoneScreen.Vertices[i].Position;

    p.z = -2.0001f;

    XMFLOAT2 t = phoneScreen.Vertices[i].TexC;
    if (i == 0 || i == 1)
        t.x = 0.35f;
    else if (i == 2 || i == 3)
        t.x = 0.66f;

    vertices3[j].Pos = p;
    vertices3[j].Normal = phoneScreen.Vertices[i].Normal;
    vertices3[j].Tex = t;
}
```

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 coordinates is between 0 and 1 so three different pictures would be a third.

Assignment 6 Shading

For this assignment I used the Flame shader from shadertoy.com and refactor the code from glsl to hlsl.

this shader needed a global time to animate the flame shader. To get the time I created a variable mTimeScalar from the type see below.

```
ID3DX11EffectScalarVariable * mTimeScalar;
```

In the drawScene method I give the mTimeScalar the value of TotalTime from mTimer.

```
mTimeScalar->SetFloat(mTimer.TotalTime()); // set time
```

In the BuildFX method a link the mTimeScalar with the shader as shown below.

```
// link variables to shader
mTech = mFX->GetTechniqueByName("ColorTech");
mfxWorldViewProj = mFX->GetVariableByName("gWorldViewProj")->AsMatrix();
mTimeScalar = mFX->GetVariableByName("sTime")->AsScalar();
```

```
static float iGlobalTime;
cbuffer cbPerObject
{
    float4x4 gWorldViewProj;
    float sTime;
};
```

Now that sTime has a value I give that value to IGlobalTime to animate the flame.

```
float flame(float3 p)
{
    float d = sphere(p*float3(1., .5, 1.), float4(.0, -1., .0, 1.));
    return d + (noise(p + float3(.0, iGlobalTime*2., .0)) + noise(p*3.)*.5)*.25*(p.y);
}
```

The shader is upside down because directX uses left handed coordinate system and openGL uses right handed coordinate system.