MASTER ELECTRONIC DESIGN

Homework 5( )

Miguel Tlapa Juárez

19/05/2014

This document describes the system architecture and design about the body controller module, it's have block diagram and flowchart to describe software and hardware architecture.

Revision History

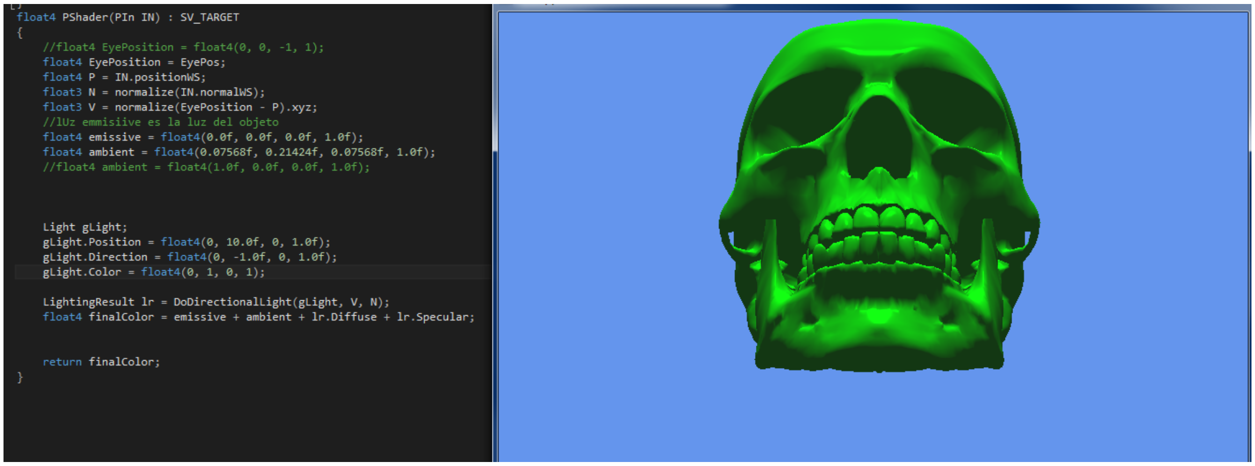
Date	Revision Number	Author/Editor	Modifications
June2014	0.1	Miguel Tlapa	Created file

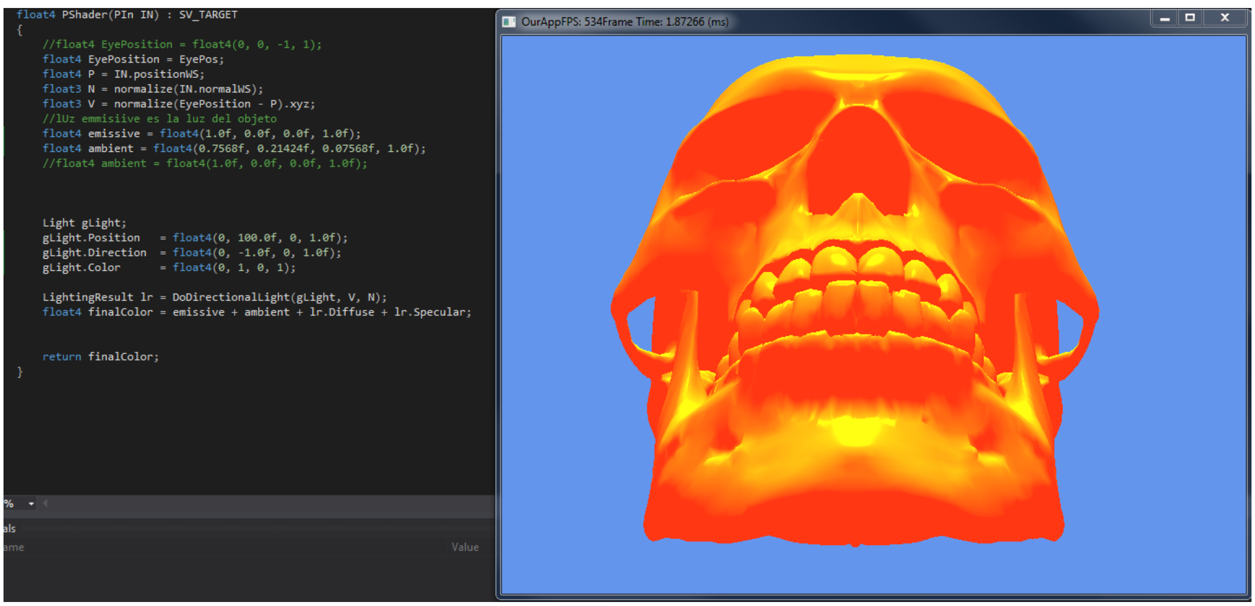
Disclaimers

EXERCISES

1. **Modify the values of each term in the light equation**

   Original Values





2. **Send the position of the eye to the pixel shader**

ModelApp.cpp

```
void ModelApp::Update(float dt)
```

```
XMFLOAT3 Epos = mCam.GetPosition();
XMFLOAT4 EyePos = XMFLOAT4(Epos.x, Epos.y, Epos.z,1.0f);
m_pImmediateContext->UpdateSubresource(mpConstantP[0], 0, nullptr, &EyePos, 0, 0);
```

ShaderPhongP.hlsl

```
cbuffer Ojo: register(b0)
{
    float4 EyePos:POSITION;

}
```

```
float4 PShader(PIn IN) : SV_TARGET
{
    //float4 EyePosition = float4(0, 0, -1, 1);
    float4 EyePosition = EyePos;
    float4 P = IN.positionWS;
    float3 N = normalize(IN.normalWS);
    float3 V = normalize(EyePosition - P).xyz;
    //luz emmisiive es la luz del objeto
    float4 emissive = float4(0.0f, 0.0f, 0.0f, 1.0f);
    float4 ambient = float4(0.07568f, 0.21424f, 0.07568f, 1.0f);
    //float4 ambient = float4(1.0f, 0.0f, 0.0f, 1.0f);



    Light gLight;
    gLight.Position = float4(0, 100.0f, 0, 1.0f);
    gLight.Direction = float4(0, -1.0f, 0, 1.0f);
    gLight.Color = float4(0, 1, 0, 1);

    LightingResult lr = DoDirectionalLight(gLight, V, N);
    float4 finalColor = emissive + ambient + lr.Diffuse + lr.Specular;


    return finalColor;
}
```

3.- Send the position of the light (float4) and implement a new function (DoPointLight) that considers the position of the light, call another function (DoLinearAttenuation) that reduce the light intensity as the distance increases. Tip: use a distance factor, specify a range of a valid distance. (In the next demo, we will implement a better approach).

ModelApp.cpp

```
struct EYE
{
    XMFLOAT4 EyePos;
    XMFLOAT4 EyeDir;
};
```

```
ID3D11Buffer* mpConstP;
```

```
void ModelApp::Update(float dt)
```

```
XMFLOAT3 Epos = mCam.GetPosition();
XMFLOAT4 EyePos = XMFLOAT4(Epos.x, Epos.y, Epos.z,1.0f);

// Son nuevas lineas
XMFLOAT3 Edir = mCam.GetLook();
XMFLOAT4 EyeDir = XMFLOAT4(Edir.x, Edir.y, Edir.z, 1.0f);

EYE eyeInfo;
eyeInfo.EyePos = EyePos;
eyeInfo.EyeDir = EyeDir;

m_pImmediateContext->UpdateSubresource(mpConstP, 0, nullptr, &eyeInfo, 0, 0);
```

```
void ModelApp::InitConstantBuffers(){
```

```
D3D11_BUFFER_DESC constantBufferDesc2;
ZeroMemory(&constantBufferDesc2, sizeof(D3D11_BUFFER_DESC));

constantBufferDesc2.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
constantBufferDesc2.ByteWidth = sizeof(EYE);
constantBufferDesc2.CPUAccessFlags = 0;
constantBufferDesc2.Usage = D3D11_USAGE_DEFAULT;
HR(m_pDevice->CreateBuffer(&constantBufferDesc2, nullptr, &mpConstP));
```

```
void ModelApp::Render(float dt)
{
```

```
m_pImmediateContext->PSSetConstantBuffers(0, 1, &mpConstP);
```

ShaderPhongP.hlsl*

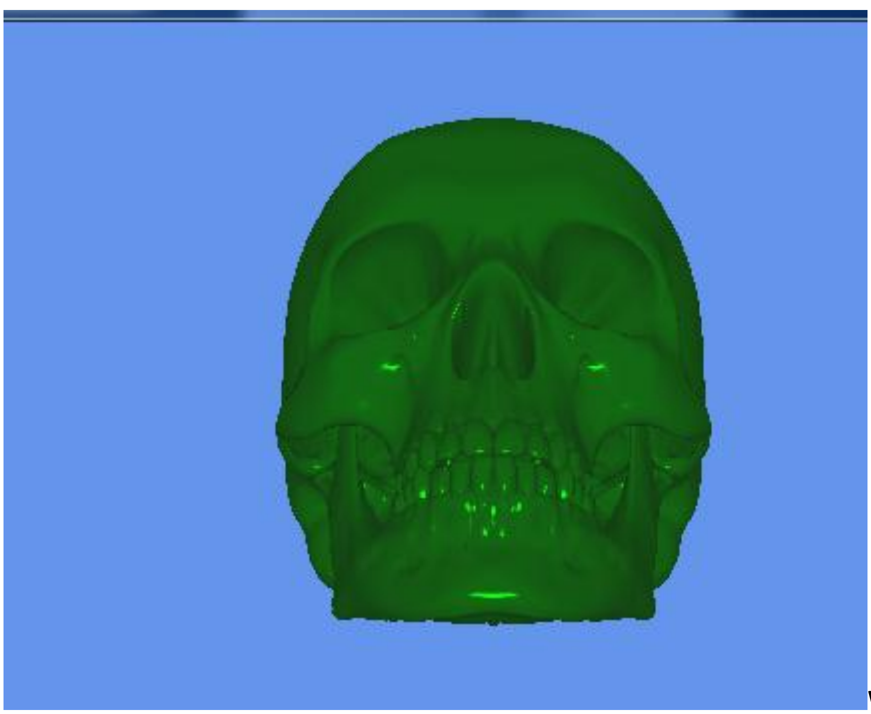
```
cbuffer Ojo: register(b0)
{
    float4 EyePos:POSITION;
    //---16BYTES
    float4 LDir: POSITION1;
    //---16BYTES

}
```

```
float4 PShader(PIn IN) : SV_TARGET
{
```

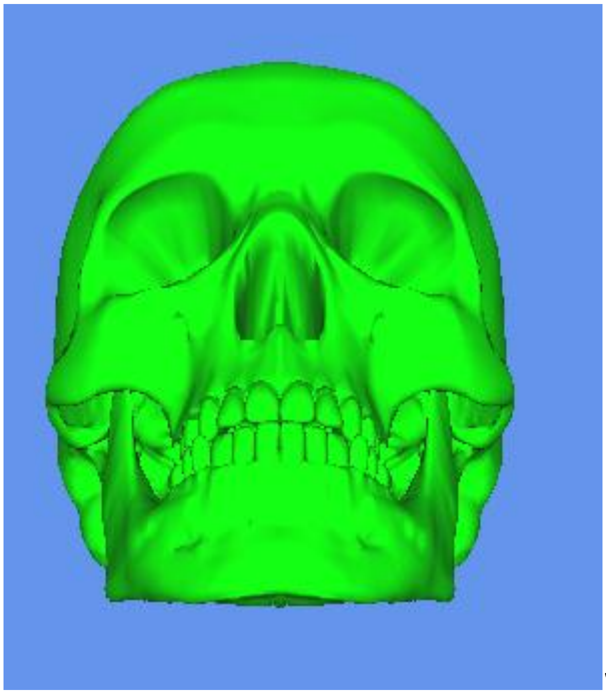
```
Light gLight;
gLight.Position = EyePos;
gLight.Direction = LDir; //  float4(0, -1.0f, 0, 1.0f);
gLight.Color = float4(0, 1, 0, 1);
```

```
float DoAttenuation(Light light, float4 P)
{

    float3 L              = (light.Position - P).xyz;
    float distance        = length(L);
    float ajuste;
    if (distance >= 5)
    {

        ajuste = 0.2;
    }


    return ajuste;
}
```

```
float ajuste = DoAttenuation(gLight,P);


LightingResult lr = DoDirectionalLight(gLight, V, N);
lr.Diffuse = lr.Diffuse*ajuste;
//LightingResult lr = DoDirectionalLight(attribute_light, V, N);
float4 finalColor = emissive + ambient + lr.Diffuse + lr.Specular;
```



With Value 0.2

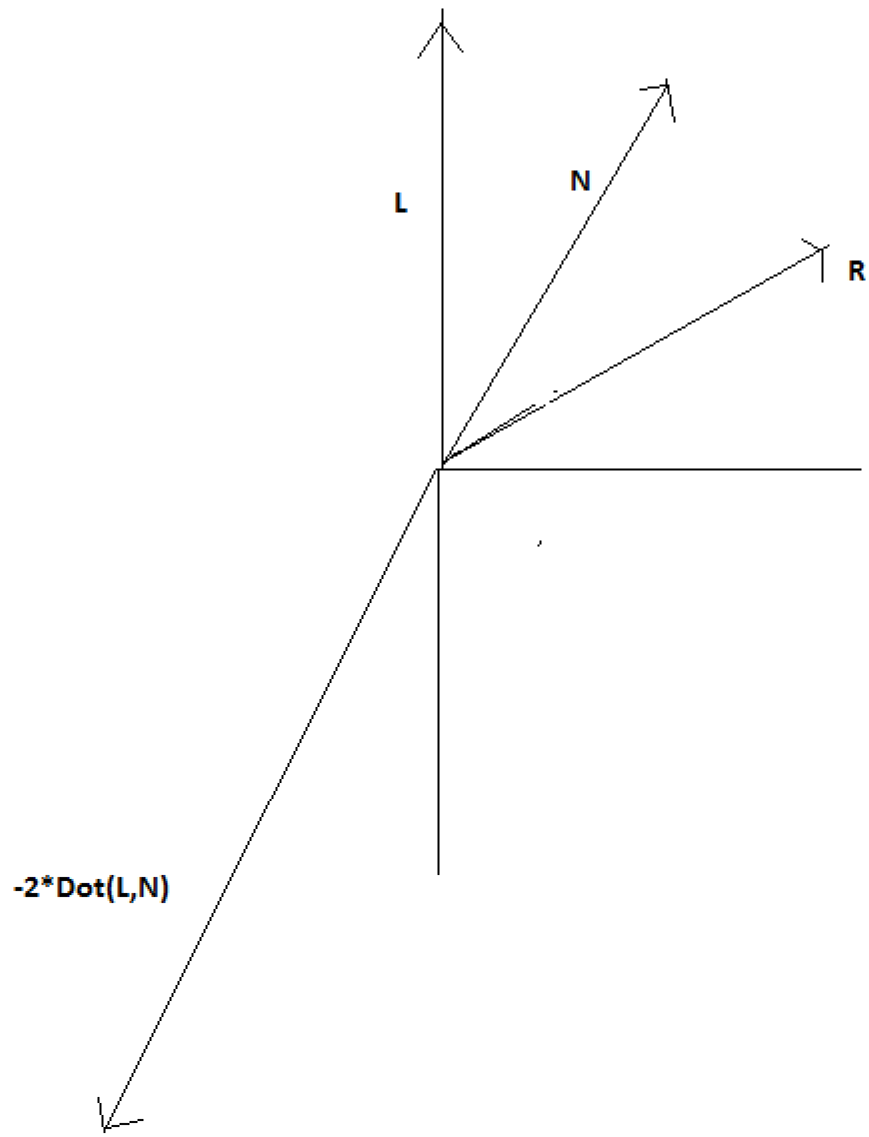

With Value 1.0

4.-

Understand the following equation and indicate if the reflected vector R is the same as the Phong lighting model presented before (explain how the R is been computed), implement this equation in your pixel shader:

R = −(L − 2(L · N) N)

	X	Y	Z
L	0.000000000000000	1.000000000000000	0.000000000000000
N	0.007739114000000	0.695679700000000	-0.718310500000000
Dot (L,N) =	0.695680000000000		
2*Dot(L,N) =	1.391360000000000		
2*Dot(L,N)*N =	0.010767893655040	0.967940907392000	-0.999428497280000
R = -[L - 2*Dot(L,N)*N]	-[-0.01076789365504, 0.032059092608,0.994284]		
	0.01076789365504, -0.032059092608,-0.994284]		


L
N
R
-2*Dot(L,N)

```
float4 DoSpecular(Light light, float3 V, float3 L, float3 N)
{
    //Phong lighting
    //float3 R    = normalize(reflect(-L, N));
    //float3 R = 2 * (dot(L, N)*N - L;
    float3 R = -(L - 2 * dot(L, N)*N);
    // RdotV = Calculo la distancia entre el Vector de Reflexion y el de la Camara
    float RdotV = max(0, dot(R, V));
    // Blinn-Phong lighting

    float3 H = normalize(L + V);
    float NdotH = max(0, dot(N, H));


    //NdotH
    return light.Color*pow(RdotV, SPECULAR_POWER);


}
```