DirectX 11 Tutorial 14 – Box Class

Part 1 – Box Class

Using the project provided and the lecture notes add a new class “Box”. The box will require and index buffer with 36 indices:

// Create the indices

UINT indices[] = {

// front face OK

0, 1, 2,

0, 2, 3,

// back face OK

4, 6, 5,

4, 7, 6,

// left face

//4, 5, 1,4, 1, 0,

8, 9, 10,

8, 10, 11,

// right face

//3, 2, 6,3, 6, 7,

12, 13, 14,

12, 14, 15,

// top face

//1, 5, 6,1, 6, 2,

16, 17, 18,

16, 18, 19,

// bottom face

//4, 0, 3, 4, 3, 7

20, 21, 22,

20, 22, 23,

};

And should be rendered with the DrawIndexed method (context->DrawIndexed(36, 0, 0);) as opposed to the Draw method for the Triangle.

If you get stuck review the solution to posted on BB

Part 2

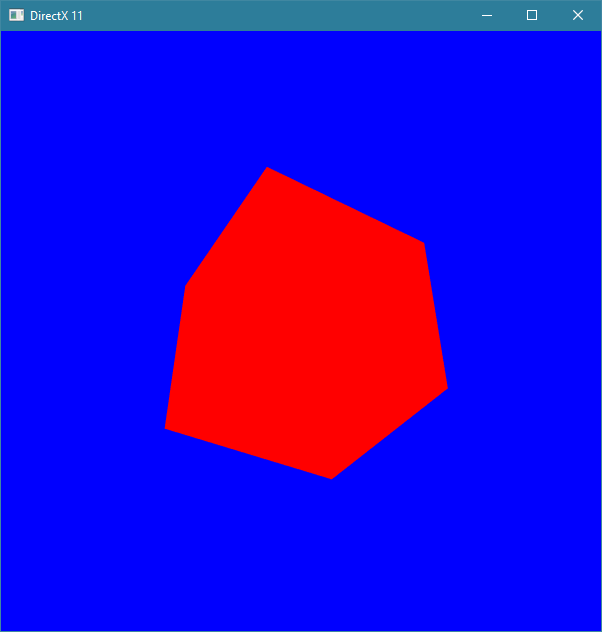
Using your solution to Part 1 and the lecture notes add a red “Box” to your scene and render it.

Notes

Create a Cbuffer containing WVPMatrix, WorldITMatrix (Normal Matrix) and a light vector.

Create a new vertex shader (basic\_lighting\_vs.hlsl) with the appropriate CBuffer and vertex input and output declarations you can use the basic\_colour\_ps for the pixel shader so make sure the output of the vertex shader matches the in put of the pixel shader.

Add anew Effect (using basic\_lighting\_vs and basic\_colour\_ps).



If you get stuck review the Additional Notes (below) and solution to posted on BB

If you are stuck:

* define the CBufferStructure in basic\_lighting\_vs:

cbuffer basicCBuffer : register(b0) {

row\_major float4x4 worldViewProjMatrix;

row\_major float4x4 worldITMatrix;

float3 lightDir;

};

Setting up the CPU side CBuffer and creating the GPU resource CBuffer

Add the following structure to CBufferStructures.h:

// CBufferBasic struct

\_\_declspec(align(16)) struct CBufferBasic {

DirectX::XMMATRIX WVPMatrix;

DirectX::XMMATRIX worldITMatrix;

DirectX::XMFLOAT3 lightDir;

};

* In Scene.h add a pointer to a CBufferBasic struct:
* CBufferBasic \*cBufferBasicCPU;
* and a pointer to an ID3D11Buffer:
* ID3D11Buffer \*cBufferBasicGPU;
* Also make sure you have added a pointer to a box and a pointer to a Box

Add code to Scene.cpp

In Scene.cpp (initialiseSceneResources)

* Create a new Effect using the basic\_lighting\_vs and the basic\_colour\_pixel shader from last week.
* Create a new Box.
* Allocate memory and initialise the CPU side CBuffer:

//Allocate 16 byte aligned block of memory for "main memory" copy of cBufferBasic

cBufferBasicCPU = (CBufferBasic\*)\_aligned\_malloc(sizeof(CBufferBasic), 16);

// Fill out cBufferBasicCPU

XMVECTOR pos = XMVectorSet(-3, -5, -5, 1.0f);

XMVECTOR target = XMVectorZero();

XMVECTOR up = XMVectorSet(0.0f, 1.0f, 0.0f, 1.0f);

XMMATRIX viewMatrix = XMMatrixLookAtLH(pos, target, up);

XMMATRIX projMatrix = XMMatrixPerspectiveFovLH(0.25f\*3.14 , 1.0, 1.0f, 1000.0f);

XMMATRIX worldMatrix = XMMatrixRotationY(XMConvertToRadians(0));

cBufferBasicCPU->WVPMatrix = worldMatrix\*viewMatrix\*projMatrix;

XMVECTOR det=XMMatrixDeterminant(worldMatrix);

cBufferBasicCPU->worldITMatrix = XMMatrixInverse(&det, XMMatrixTranspose(worldMatrix));

cBufferBasicCPU->lightDir = XMFLOAT3(0.5, -0.5, 1.0);

* Initialise CbufferGPU description:

// Create GPU resource memory copy of cBufferBasic

// fill out description (Note if we want to update the CBuffer we need D3D11\_CPU\_ACCESS\_WRITE)

D3D11\_BUFFER\_DESC cbufferDesc;

D3D11\_SUBRESOURCE\_DATA cbufferInitData;

ZeroMemory(&cbufferDesc, sizeof(D3D11\_BUFFER\_DESC));

ZeroMemory(&cbufferInitData, sizeof(D3D11\_SUBRESOURCE\_DATA));

cbufferDesc.ByteWidth = sizeof(CBufferBasic);

cbufferDesc.Usage = D3D11\_USAGE\_DYNAMIC;

cbufferDesc.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE;

cbufferDesc.BindFlags = D3D11\_BIND\_CONSTANT\_BUFFER;

cbufferInitData.pSysMem = cBufferBasicCPU;// Initialise GPU CBuffer with data from CPU CBuffer

* Create the ID3D11Buffer CBuffer

HRESULT hr = device->CreateBuffer(&cbufferDesc, &cbufferInitData,

&cBufferBasicGPU);

* Update the GPU CBuffer with the data from the CPU CBuffer.

// We dont strictly need to call map here as the GPU CBuffer was initialised from the CPU CBuffer at creation.

// However if changes are made to the CPU CBuffer during update the we need to copy the data to the GPU CBuffer

// using the mapCbuffer helper function provided the in Util.h mapCbuffer(context, cBufferBasicCPU, cBufferBasicGPU, sizeof(CBufferBasic));

* Apply the CBuffer to the Vertex shader using the context:

// Attach CBufferGPU to to register b0 for the vertex shader

context->VSSetConstantBuffers(0, 1, &cBufferBasicGPU):

* Render the Box

Refer to the lecture notes for additional information.

