# main.cpp

//#pragma warning(disable:4996)

#include <windows.h>

#include <d3d11.h>

#include <d3dx11.h>

#include <dxerr.h>

#include <dinput.h>

#define \_XM\_NO\_INTRINSICS\_

#define XM\_NO\_ALIGNMENT

#include <xnamath.h>

#include <stdio.h>

int (WINAPIV \* \_\_vsnprintf\_)(char \*, size\_t, const char\*, va\_list) = \_vsnprintf;

// My classes

#include "camera.h"

#include "text2D.h"

#include "input.h"

#include "model.h"

//////////////////////////////////////////////////////////////////////////////////////

// Global Variables

//////////////////////////////////////////////////////////////////////////////////////

HINSTANCE g\_hInst = NULL;

HWND g\_hWnd = NULL;

char g\_TutorialName[100] = "Philip Guy Individual Project AE2\0";

D3D\_DRIVER\_TYPE g\_driverType = D3D\_DRIVER\_TYPE\_NULL;

D3D\_FEATURE\_LEVEL g\_featureLevel = D3D\_FEATURE\_LEVEL\_11\_0;

ID3D11Device\* g\_pD3DDevice = NULL;

ID3D11DeviceContext\* g\_pImmediateContext = NULL;

IDXGISwapChain\* g\_pSwapChain = NULL;

ID3D11RenderTargetView\* g\_pBackBufferRTView = NULL;

ID3D11Buffer\* g\_pVertexBuffer; // Vertex buffer

ID3D11VertexShader\* g\_pVertexShader; // Vertex shader

ID3D11PixelShader\* g\_pPixelShader; // Pixel Shader

ID3D11InputLayout\* g\_pInputLayout;

ID3D11DepthStencilView\* g\_pZBuffer; // Z buffer

ID3D11ShaderResourceView\* g\_pTexture0; // Load texture

ID3D11SamplerState\* g\_pSampler0;

Text2D\* g\_2DText;

// Camera pointer

camera\* Camera;

float degrees = 0;

// keyboard

input Keyboard;

// Ambient and Directional Lighting

XMVECTOR g\_directional\_light\_shines\_from;

XMVECTOR g\_directional\_light\_colour;

XMVECTOR g\_ambient\_light\_colour;

// Define vertex structure

struct POS\_COL\_TEX\_NORM\_VERTEX

{

XMFLOAT3 Pos;

XMFLOAT4 Col;

XMFLOAT2 Texture0; // So that multiple sets of texture coords can be used at once

XMFLOAT3 Normal;

};

// Const buffer structs. Pack to 16 bytes. Don't let any single element cross a 16 byte boundary

struct CONSTANT\_BUFFER0

{

XMMATRIX WorldViewProjection; // 64 bytes ( Must be a multiple of 16)

XMVECTOR directional\_light\_vector; // 16 bytes

XMVECTOR directional\_light\_colour; // 16 bytes

XMVECTOR ambient\_light\_colour; // 16 bytes

// Total size = 112 bytes

};

ID3D11Buffer\* g\_pConstantBuffer0;

//////////////////////////////////////////////////////////////////////////////////////

// Forward declarations

//////////////////////////////////////////////////////////////////////////////////////

HRESULT InitialiseWindow(HINSTANCE hInstance, int nCmdShow);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

HRESULT InitialiseD3D();

void ShutdownD3D();

void RenderFrame(void);

HRESULT InitialiseGraphics(void);

//////////////////////////////////////////////////////////////////////////////////////

// Entry point to the program. Initializes everything and goes into a message processing

// loop. Idle time is used to render the scene.

//////////////////////////////////////////////////////////////////////////////////////

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

if (FAILED(InitialiseWindow(hInstance, nCmdShow)))

{

DXTRACE\_MSG("Failed to create Window");

return 0;

}

if (FAILED(Keyboard.InitialiseInput(g\_hInst, g\_hWnd)))

{

DXTRACE\_MSG("Failed to initialise keyboard");

return 0;

}

if (FAILED(InitialiseD3D()))

{

DXTRACE\_MSG("Failed to create Device");

return 0;

}

if (FAILED(InitialiseGraphics()))

{

DXTRACE\_MSG("Failed to initialise graphics");

return 0;

}

// Main message loop

MSG msg = { 0 };

while (msg.message != WM\_QUIT)

{

if (PeekMessage(&msg, NULL, 0, 0, PM\_REMOVE))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

else

{

// Movement of

// Movement of camera

RenderFrame();

}

}

ShutdownD3D();

return (int)msg.wParam;

}

//////////////////////////////////////////////////////////////////////////////////////

// Register class and create window

//////////////////////////////////////////////////////////////////////////////////////

HRESULT InitialiseWindow(HINSTANCE hInstance, int nCmdShow)

{

// Give your app window your own name

char Name[100] = "Hello World\0";

// Register class

WNDCLASSEX wcex = { 0 };

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.hInstance = hInstance;

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

// wcex.hbrBackground = (HBRUSH )( COLOR\_WINDOW + 1); // Needed for non-D3D apps

wcex.lpszClassName = Name;

if (!RegisterClassEx(&wcex)) return E\_FAIL;

// Create window

g\_hInst = hInstance;

RECT rc = { 0, 0, 640, 480 };

AdjustWindowRect(&rc, WS\_OVERLAPPEDWINDOW, FALSE);

g\_hWnd = CreateWindow(Name, g\_TutorialName, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, rc.right - rc.left,

rc.bottom - rc.top, NULL, NULL, hInstance, NULL);

if (!g\_hWnd)

return E\_FAIL;

ShowWindow(g\_hWnd, nCmdShow);

return S\_OK;

}

//////////////////////////////////////////////////////////////////////////////////////

// Called every time the application receives a message

//////////////////////////////////////////////////////////////////////////////////////

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

/\*case WM\_KEYDOWN:

if (wParam == VK\_ESCAPE)

DestroyWindow(g\_hWnd);

return 0;\*/

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

//////////////////////////////////////////////////////////////////////////////////////

// Create D3D device and swap chain

//////////////////////////////////////////////////////////////////////////////////////

HRESULT InitialiseD3D()

{

HRESULT hr = S\_OK;

RECT rc;

GetClientRect(g\_hWnd, &rc);

UINT width = rc.right - rc.left;

UINT height = rc.bottom - rc.top;

UINT createDeviceFlags = 0;

#ifdef \_DEBUG

createDeviceFlags |= D3D11\_CREATE\_DEVICE\_DEBUG;

#endif

D3D\_DRIVER\_TYPE driverTypes[] =

{

D3D\_DRIVER\_TYPE\_HARDWARE, // comment out this line if you need to test D3D 11.0

// functionality on hardware that doesn't support it

D3D\_DRIVER\_TYPE\_WARP, // comment this out also to use reference device

D3D\_DRIVER\_TYPE\_REFERENCE,

};

UINT numDriverTypes = ARRAYSIZE(driverTypes);

D3D\_FEATURE\_LEVEL featureLevels[] =

{

D3D\_FEATURE\_LEVEL\_11\_0,

D3D\_FEATURE\_LEVEL\_10\_1,

D3D\_FEATURE\_LEVEL\_10\_0,

};

UINT numFeatureLevels = ARRAYSIZE(featureLevels);

DXGI\_SWAP\_CHAIN\_DESC sd;

ZeroMemory(&sd, sizeof(sd));

sd.BufferCount = 1;

sd.BufferDesc.Width = width;

sd.BufferDesc.Height = height;

sd.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

sd.BufferDesc.RefreshRate.Numerator = 60;

sd.BufferDesc.RefreshRate.Denominator = 1;

sd.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

sd.OutputWindow = g\_hWnd;

sd.SampleDesc.Count = 1;

sd.SampleDesc.Quality = 0;

sd.Windowed = true;

for (UINT driverTypeIndex = 0; driverTypeIndex < numDriverTypes; driverTypeIndex++)

{

g\_driverType = driverTypes[driverTypeIndex];

hr = D3D11CreateDeviceAndSwapChain(NULL, g\_driverType, NULL,

createDeviceFlags, featureLevels, numFeatureLevels,

D3D11\_SDK\_VERSION, &sd, &g\_pSwapChain,

&g\_pD3DDevice, &g\_featureLevel, &g\_pImmediateContext);

if (SUCCEEDED(hr))

break;

}

if (FAILED(hr))

return hr;

// Get pointer to back buffer texture

ID3D11Texture2D \*pBackBufferTexture;

hr = g\_pSwapChain->GetBuffer(0, \_\_uuidof(ID3D11Texture2D),

(LPVOID\*)&pBackBufferTexture);

if (FAILED(hr)) return hr;

// Use the back buffer texture pointer to create the render target view

hr = g\_pD3DDevice->CreateRenderTargetView(pBackBufferTexture, NULL,

&g\_pBackBufferRTView);

pBackBufferTexture->Release();

if (FAILED(hr)) return hr;

// Ceate a Z buffer texture

D3D11\_TEXTURE2D\_DESC tex2dDesc;

ZeroMemory(&tex2dDesc, sizeof(tex2dDesc));

tex2dDesc.Width = width;

tex2dDesc.Height = height;

tex2dDesc.ArraySize = 1;

tex2dDesc.MipLevels = 1;

tex2dDesc.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;

tex2dDesc.SampleDesc.Count = sd.SampleDesc.Count;

tex2dDesc.BindFlags = D3D11\_BIND\_DEPTH\_STENCIL;

tex2dDesc.Usage = D3D11\_USAGE\_DEFAULT;

ID3D11Texture2D \*pZBufferTexture;

hr = g\_pD3DDevice->CreateTexture2D(&tex2dDesc, NULL, &pZBufferTexture);

if (FAILED(hr)) return hr;

// Create the Z buffer

D3D11\_DEPTH\_STENCIL\_VIEW\_DESC dsvDesc;

ZeroMemory(&dsvDesc, sizeof(dsvDesc));

dsvDesc.Format = tex2dDesc.Format;

dsvDesc.ViewDimension = D3D11\_DSV\_DIMENSION\_TEXTURE2D;

g\_pD3DDevice->CreateDepthStencilView(pZBufferTexture, &dsvDesc, &g\_pZBuffer);

pZBufferTexture->Release();

// Set the render target view

g\_pImmediateContext->OMSetRenderTargets(1, &g\_pBackBufferRTView, g\_pZBuffer);

// Set the viewport

D3D11\_VIEWPORT viewport;

viewport.TopLeftX = 0;

viewport.TopLeftY = 0;

viewport.Width = width;

viewport.Height = height;

viewport.MinDepth = 0.0f;

viewport.MaxDepth = 1.0f;

g\_pImmediateContext->RSSetViewports(1, &viewport);

//HUD

g\_2DText = new Text2D("assets/font1.bmp", g\_pD3DDevice, g\_pImmediateContext);

return S\_OK;

}

//////////////////////////////////////////////////////////////////////////////////////

// Clean up D3D objects

//////////////////////////////////////////////////////////////////////////////////////

void ShutdownD3D()

{

if (g\_pConstantBuffer0) g\_pConstantBuffer0->Release();

if (g\_pVertexBuffer) g\_pVertexBuffer->Release();

if (g\_pInputLayout) g\_pInputLayout->Release();

if (g\_pVertexShader) g\_pVertexShader->Release();

if (g\_pPixelShader) g\_pPixelShader->Release();

//Textures

if (g\_pTexture0) g\_pTexture0->Release();

if (g\_pSampler0) g\_pSampler0->Release();

if (g\_pBackBufferRTView) g\_pBackBufferRTView->Release();

if (g\_pSwapChain) g\_pSwapChain->Release();

if (g\_pImmediateContext) g\_pImmediateContext->Release();

if (g\_pD3DDevice) g\_pD3DDevice->Release();

if (g\_pD3DDevice) g\_pD3DDevice->Release();

delete Camera;

delete g\_2DText;

}

//////////////////////////////////////////////////////////////////////////////////////

// Init graphics

//////////////////////////////////////////////////////////////////////////////////////

HRESULT InitialiseGraphics() //03-01

{

HRESULT hr = S\_OK;

// Define vertices of a cube - screen coordinates -1.0 to +1.0

POS\_COL\_TEX\_NORM\_VERTEX vertices[] =

{

// back face

{ XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

{ XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

{ XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, 1.0f) },

// front face

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

{ XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

{ XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

{ XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

{ XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 0.0f, -1.0f) },

// left face

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(-1.0f, 0.0f, 0.0f) },

// right face

{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

// bottom face

{ XMFLOAT3(1.0f, -1.0f, -1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

{ XMFLOAT3(1.0f, -1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, -1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, -1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, -1.0f, 0.0f) },

// top face

{ XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

{ XMFLOAT3(1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, 1.0f), XMFLOAT4(1.0f, 0.0f, 0.0f, 1.0f), XMFLOAT2(0.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

{ XMFLOAT3(1.0f, 1.0f, 1.0f), XMFLOAT4(0.0f, 1.0f, 0.0f, 1.0f), XMFLOAT2(1.0f, 0.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) },

{ XMFLOAT3(-1.0f, 1.0f, -1.0f), XMFLOAT4(0.0f, 0.0f, 1.0f, 1.0f), XMFLOAT2(0.0f, 1.0f), XMFLOAT3(0.0f, 1.0f, 0.0f) }

};

// Set up and create vertex buffer

D3D11\_BUFFER\_DESC bufferDesc;

ZeroMemory(&bufferDesc, sizeof(bufferDesc));

bufferDesc.Usage = D3D11\_USAGE\_DYNAMIC; // Used by CPU and GPU

bufferDesc.ByteWidth = sizeof(vertices); // Total size of buffer

bufferDesc.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER; // Use as a vertex buffer

bufferDesc.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE; // Allow CPU access

hr = g\_pD3DDevice->CreateBuffer(&bufferDesc, NULL, &g\_pVertexBuffer); // Create the buffer

// Set up and create constant buffer

D3D11\_BUFFER\_DESC constant\_buffer\_desc;

ZeroMemory(&constant\_buffer\_desc, sizeof(constant\_buffer\_desc));

constant\_buffer\_desc.Usage = D3D11\_USAGE\_DEFAULT; // Can use UpdateSubresource() to update

constant\_buffer\_desc.ByteWidth = 112; // MUST be a multiple of 16, calculate from CB struct

constant\_buffer\_desc.BindFlags = D3D11\_BIND\_CONSTANT\_BUFFER; // Use as a constant buffer

hr = g\_pD3DDevice->CreateBuffer(&constant\_buffer\_desc, NULL, &g\_pConstantBuffer0);

if (FAILED(hr)) // return error code on failure

{

return hr;

}

// Copy the vertices into the buffer

D3D11\_MAPPED\_SUBRESOURCE ms;

// Lock the buffer to allow writing

g\_pImmediateContext->Map(g\_pVertexBuffer, NULL, D3D11\_MAP\_WRITE\_DISCARD, NULL, &ms);

// Copy the data

memcpy(ms.pData, vertices, sizeof(vertices));

// Unlock the buffer

g\_pImmediateContext->Unmap(g\_pVertexBuffer, NULL);

// Load and compile pixel and vertex shaders - use vs\_5\_0 to target DX11 hardware only

ID3DBlob \*VS, \*PS, \*error;

hr = D3DX11CompileFromFile("shaders.hlsl", 0, 0, "VShader", "vs\_4\_0", 0, 0, 0, &VS, &error, 0);

if (error != 0) // Check for shader compilation error

{

OutputDebugStringA((char\*)error->GetBufferPointer());

error->Release();

if (FAILED(hr)) // Don't fail if error is just a warning

{

return hr;

};

}

hr = D3DX11CompileFromFile("shaders.hlsl", 0, 0, "PShader", "ps\_4\_0", 0, 0, 0, &PS, &error, 0);

if (error != 0) // Check for shader compilation error

{

OutputDebugStringA((char\*)error->GetBufferPointer());

error->Release();

if (FAILED(hr)) // Don't fail if error is just a warning

{

return hr;

};

}

// Create shader objects

hr = g\_pD3DDevice->CreateVertexShader(VS->GetBufferPointer(), VS->GetBufferSize(), NULL, &g\_pVertexShader);

if (FAILED(hr))

{

return hr;

}

hr = g\_pD3DDevice->CreatePixelShader(PS->GetBufferPointer(), PS->GetBufferSize(), NULL, &g\_pPixelShader);

if (FAILED(hr))

{

return hr;

}

// Set the shader object as active

g\_pImmediateContext->VSSetShader(g\_pVertexShader, 0, 0);

g\_pImmediateContext->PSSetShader(g\_pPixelShader, 0, 0);

// Create and set the input layout object

D3D11\_INPUT\_ELEMENT\_DESC iedesc[] =

{

{ "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

{ "COLOR", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0, D3D11\_APPEND\_ALIGNED\_ELEMENT, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

{ "TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, D3D11\_APPEND\_ALIGNED\_ELEMENT, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{ "NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, D3D11\_APPEND\_ALIGNED\_ELEMENT, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0}

};

hr = g\_pD3DDevice->CreateInputLayout(iedesc, ARRAYSIZE(iedesc), VS->GetBufferPointer(), VS->GetBufferSize(), &g\_pInputLayout);

if (FAILED(hr))

{

return hr;

}

// Loading player texture

D3DX11CreateShaderResourceViewFromFile(g\_pD3DDevice, "assets/texture.bmp", NULL, NULL, &g\_pTexture0, NULL);

D3D11\_SAMPLER\_DESC sampler\_desc;

ZeroMemory(&sampler\_desc, sizeof(sampler\_desc));

sampler\_desc.Filter = D3D11\_FILTER\_MIN\_MAG\_MIP\_LINEAR;

sampler\_desc.AddressU = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampler\_desc.AddressV = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampler\_desc.AddressW = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampler\_desc.MaxLOD = D3D11\_FLOAT32\_MAX;

g\_pD3DDevice->CreateSamplerState(&sampler\_desc, &g\_pSampler0);

// Primary and secondary camera creation

camera \*mainCamera = new camera(0.0, 0.0, -0.5, 0);

camera \*secondaryCamera = new camera(0.0, 1.0, 0, 0);

g\_pImmediateContext->IASetInputLayout(g\_pInputLayout);

return S\_OK;

}

// Render frame

void RenderFrame(void)

{

// User Controls

Keyboard.ReadInputStates();

if (Keyboard.IsKeyPressed(DIK\_ESCAPE)) DestroyWindow(g\_hWnd); // Exit game

if (Keyboard.IsKeyPressed(DIK\_UP)) Camera->Forward(0.2f); // Forward

if (Keyboard.IsKeyPressed(DIK\_DOWN)) Camera->Forward(-0.2f); // Backwards

if (Keyboard.IsKeyPressed(DIK\_LEFT)) Camera->Rotate(2.0f); // Look left

if (Keyboard.IsKeyPressed(DIK\_RIGHT)) Camera->Rotate(2.0f); // Look right

if (Keyboard.IsKeyPressed(DIK\_SPACE)) Camera->Up(1.0f); // Jump

Camera->Up(-1.0f); // Gravity, keeps the player on the ground if not jumping

// Clear the back buffer - choose a colour you like

float rgba\_clear\_colour[4] = { 0.1f, 0.2f, 0.6f, 1.0f };

g\_pImmediateContext->ClearRenderTargetView(g\_pBackBufferRTView, rgba\_clear\_colour);

// Clear the Z buffer

g\_pImmediateContext->ClearDepthStencilView(g\_pZBuffer, D3D11\_CLEAR\_DEPTH | D3D11\_CLEAR\_STENCIL, 1.0f, 0);

// Lighting

g\_directional\_light\_shines\_from = XMVectorSet(0.0f, 0.0f, -1.0f, 0.0f);

g\_directional\_light\_colour = XMVectorSet(1.0f, 1.0f, 1.0f, 0.0f);

g\_ambient\_light\_colour = XMVectorSet(0.1f, 0.1f, 0.1f, 1.0f);

// RENDER HERE

// Set vertex buffer

UINT stride = sizeof(POS\_COL\_TEX\_NORM\_VERTEX);

UINT offset = 0;

g\_pImmediateContext->IASetVertexBuffers(0, 1, &g\_pVertexBuffer, &stride, &offset);

// Constant Buffers

XMMATRIX transpose;

CONSTANT\_BUFFER0 cb0\_values;

// Set view matrix before creating WorldViewMatrix

Camera->GetViewMatrix();

// World, view, projection transformations

XMMATRIX projection, world, view;

// Matrix concatenation + backface culling

world = XMMatrixRotationX(XMConvertToRadians(degrees));

world \*= XMMatrixTranslation(2, 0, 5);

projection = XMMatrixPerspectiveFovLH(XMConvertToRadians(45.0), 640.0 / 480.0, 1.0, 100.0);

view = XMMatrixIdentity();

cb0\_values.WorldViewProjection = world \* view \* projection;

transpose = XMMatrixTranspose(world); // model world matrix

cb0\_values.directional\_light\_vector = g\_directional\_light\_colour;

cb0\_values.ambient\_light\_colour = g\_ambient\_light\_colour;

cb0\_values.directional\_light\_vector = XMVector3Transform(g\_directional\_light\_shines\_from, transpose);

cb0\_values.directional\_light\_vector = XMVector3Normalize(cb0\_values.directional\_light\_vector);

// TODO: degrees varies based on keypress!!!

// TODO: camera varies based on keypress!!!

// Upload the new values for the constant buffer

g\_pImmediateContext->UpdateSubresource(g\_pConstantBuffer0, 0, 0, &cb0\_values, 0, 0);

// Set the constant buffer to be active

g\_pImmediateContext->VSSetConstantBuffers(0, 1, &g\_pConstantBuffer0);

// Setting textures and sample states

g\_pImmediateContext->PSSetSamplers(0, 1, &g\_pSampler0);

g\_pImmediateContext->PSSetShaderResources(0, 1, &g\_pTexture0);

// Select which primitive type to use //03-01

g\_pImmediateContext->IASetPrimitiveTopology(D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST);

// Draw the vertex buffer to the back buffer

g\_pImmediateContext->Draw(36, 0);

// Render the HUD

g\_2DText->AddText("HUD", -1.0, +1.0, 0.2);

g\_2DText->RenderText();

// Display what has just been rendered

g\_pSwapChain->Present(0, 0);

}

# Camera.h

// Header file - camera.h

#pragma once

class camera

{

private:

// Member variables

float m\_x;

float m\_y;

float m\_z;

float m\_dx;

float m\_dz;

float m\_camera\_rotation;

XMVECTOR m\_position;

XMVECTOR m\_lookat;

XMVECTOR m\_up;

public:

// Constructor

camera(float x, float y, float z, float rotation);

~camera();

void Rotate(float degrees);

void Forward(float distance);

void Up(float distance);

XMMATRIX GetViewMatrix();

};

# Camera.cpp

// Implementation class - camera.cpp

#include "camera.h"

#include <D3D11.h>

#define \_XM\_NO\_INTRINSICS\_

#define XM\_NO\_ALIGNMENT

#include <xnamath.h>

#include <math.h>

camera::camera(float x, float y, float z, float rotation)

{

m\_x = x;

m\_y = y;

m\_z = z;

m\_camera\_rotation = rotation;

m\_dx = sin(rotation \* (XM\_PI / 180));

m\_dz = cos(rotation \* (XM\_PI / 180));

}

void camera::Rotate(float degrees)

{

m\_camera\_rotation += degrees;

m\_dx = sin(m\_camera\_rotation \* (XM\_PI / 180));

m\_dz = cos(m\_camera\_rotation \* (XM\_PI / 180));

}

void camera::Forward(float distance)

{

m\_x \*= distance;

m\_z \*= distance;

}

void camera::Up(float distance)

{

m\_y \*= distance;

}

XMMATRIX camera::GetViewMatrix()

{

m\_position = XMVectorSet(m\_x, m\_y, m\_z, 0.0);

m\_lookat = XMVectorSet(m\_x + m\_dx, m\_y, m\_z + m\_dz, 0.0);

m\_up = XMVectorSet(0.0, 1.0, 0.0, 0.0);

return XMMatrixLookAtLH(m\_position, m\_lookat, m\_up);

}

# Shaders.hlsl

Texture2D texture0;

SamplerState sampler0;

cbuffer CBuffer0

{

matrix WVPMatrix; // 64 bytes

float4 directional\_light\_vector; // 16 bytes

float4 directional\_light\_colour; // 16 bytes

float4 ambient\_light\_colour; // 16 bytes

// total 112 bytes

};

struct VOut

{

float4 position : SV\_POSITION;

float4 color : COLOR;

float2 texcoord : TEXCOORD;

};

VOut VShader(float4 position : POSITION, float4 color : COLOR, float2 texcoord : TEXCOORD, float3 normal : NORMAL)

{

VOut output;

// mul() multiplies a vertex model space position by a matrix

output.position = mul(WVPMatrix, position);

float diffuse\_amount = dot(directional\_light\_vector, normal);

diffuse\_amount = saturate(diffuse\_amount);

output.color = ambient\_light\_colour + (directional\_light\_colour \* diffuse\_amount);

output.texcoord = texcoord;

return output;

}

float4 PShader(float4 position : SV\_POSITION, float4 color : COLOR, float2 texcoord : TEXCOORD) : SV\_TARGET

{

return color \* texture0.Sample(sampler0, texcoord);

}

# Input.h

#pragma once

class input

{

private:

~input();

public:

HRESULT InitialiseInput(HINSTANCE g\_hInst, HWND g\_hWnd);

void ReadInputStates();

bool IsKeyPressed(unsigned char DI\_keycode);

};

# Input.cpp

#include "input.h"

#include <dinput.h>

IDirectInput8\* g\_direct\_input;

IDirectInputDevice8\* g\_keyboard\_device;

unsigned char g\_keyboard\_keys\_state[256]; // This will store the state of all keyboard keys

input::~input()

{

{

if (g\_keyboard\_device)

g\_keyboard\_device->Unacquire();

g\_keyboard\_device->Release();

}

if (g\_direct\_input) g\_direct\_input->Release();

}

// Create keyboard device

HRESULT input::InitialiseInput(HINSTANCE g\_hInst, HWND g\_hWnd)

{

HRESULT hr;

ZeroMemory(g\_keyboard\_keys\_state, sizeof(g\_keyboard\_keys\_state));

hr = DirectInput8Create(g\_hInst, DIRECTINPUT\_VERSION, IID\_IDirectInput8, (void\*\*)&g\_direct\_input, NULL);

if (FAILED(hr))

return hr;

hr = g\_direct\_input->CreateDevice(GUID\_SysKeyboard, &g\_keyboard\_device, NULL);

if (FAILED(hr))

return hr;

hr = g\_keyboard\_device->SetDataFormat(&c\_dfDIKeyboard);

if (FAILED(hr))

return hr;

hr = g\_keyboard\_device->SetCooperativeLevel(g\_hWnd, DISCL\_FOREGROUND | DISCL\_NONEXCLUSIVE);

if (FAILED(hr))

return hr;

hr = g\_keyboard\_device->Acquire();

if (FAILED(hr)) return hr;

return S\_OK;

}

void input::ReadInputStates()

{

HRESULT hr;

hr = g\_keyboard\_device->GetDeviceState(sizeof(g\_keyboard\_keys\_state), (LPVOID)&g\_keyboard\_keys\_state);

if (FAILED(hr))

{

if ((hr == DIERR\_INPUTLOST) || (hr == DIERR\_NOTACQUIRED))

{

g\_keyboard\_device->Acquire();

}

}

}

bool input::IsKeyPressed(unsigned char DI\_keycode)

{

return g\_keyboard\_keys\_state[DI\_keycode] & 0x80;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* DirectInput keyboard scan codes

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define DIK\_ESCAPE 0x01

#define DIK\_1 0x02

#define DIK\_2 0x03

#define DIK\_3 0x04

#define DIK\_4 0x05

#define DIK\_5 0x06

#define DIK\_6 0x07

#define DIK\_7 0x08

#define DIK\_8 0x09

#define DIK\_9 0x0A

#define DIK\_0 0x0B

#define DIK\_MINUS 0x0C /\* - on main keyboard \*/

#define DIK\_EQUALS 0x0D

#define DIK\_BACK 0x0E /\* backspace \*/

#define DIK\_TAB 0x0F

#define DIK\_Q 0x10

#define DIK\_W 0x11

#define DIK\_E 0x12

#define DIK\_R 0x13

#define DIK\_T 0x14

#define DIK\_Y 0x15

#define DIK\_U 0x16

#define DIK\_I 0x17

#define DIK\_O 0x18

#define DIK\_P 0x19

#define DIK\_LBRACKET 0x1A

#define DIK\_RBRACKET 0x1B

#define DIK\_RETURN 0x1C /\* Enter on main keyboard \*/

#define DIK\_LCONTROL 0x1D

#define DIK\_A 0x1E

#define DIK\_S 0x1F

#define DIK\_D 0x20

#define DIK\_F 0x21

#define DIK\_G 0x22

#define DIK\_H 0x23

#define DIK\_J 0x24

#define DIK\_K 0x25

#define DIK\_L 0x26

#define DIK\_SEMICOLON 0x27

#define DIK\_APOSTROPHE 0x28

#define DIK\_GRAVE 0x29 /\* accent grave \*/

#define DIK\_LSHIFT 0x2A

#define DIK\_BACKSLASH 0x2B

#define DIK\_Z 0x2C

#define DIK\_X 0x2D

#define DIK\_C 0x2E

#define DIK\_V 0x2F

#define DIK\_B 0x30

#define DIK\_N 0x31

#define DIK\_M 0x32

#define DIK\_COMMA 0x33

#define DIK\_PERIOD 0x34 /\* . on main keyboard \*/

#define DIK\_SLASH 0x35 /\* / on main keyboard \*/

#define DIK\_RSHIFT 0x36

#define DIK\_MULTIPLY 0x37 /\* \* on numeric keypad \*/

#define DIK\_LMENU 0x38 /\* left Alt \*/

#define DIK\_SPACE 0x39

#define DIK\_CAPITAL 0x3A

#define DIK\_F1 0x3B

#define DIK\_F2 0x3C

#define DIK\_F3 0x3D

#define DIK\_F4 0x3E

#define DIK\_F5 0x3F

#define DIK\_F6 0x40

#define DIK\_F7 0x41

#define DIK\_F8 0x42

#define DIK\_F9 0x43

#define DIK\_F10 0x44

#define DIK\_NUMLOCK 0x45

#define DIK\_SCROLL 0x46 /\* Scroll Lock \*/

#define DIK\_NUMPAD7 0x47

#define DIK\_NUMPAD8 0x48

#define DIK\_NUMPAD9 0x49

#define DIK\_SUBTRACT 0x4A /\* - on numeric keypad \*/

#define DIK\_NUMPAD4 0x4B

#define DIK\_NUMPAD5 0x4C

#define DIK\_NUMPAD6 0x4D

#define DIK\_ADD 0x4E /\* + on numeric keypad \*/

#define DIK\_NUMPAD1 0x4F

#define DIK\_NUMPAD2 0x50

#define DIK\_NUMPAD3 0x51

#define DIK\_NUMPAD0 0x52

#define DIK\_DECIMAL 0x53 /\* . on numeric keypad \*/

#define DIK\_OEM\_102 0x56 /\* <> or \| on RT 102-key keyboard (Non-U.S.) \*/

#define DIK\_F11 0x57

#define DIK\_F12 0x58

#define DIK\_F13 0x64 /\* (NEC PC98) \*/

#define DIK\_F14 0x65 /\* (NEC PC98) \*/

#define DIK\_F15 0x66 /\* (NEC PC98) \*/

#define DIK\_KANA 0x70 /\* (Japanese keyboard) \*/

#define DIK\_ABNT\_C1 0x73 /\* /? on Brazilian keyboard \*/

#define DIK\_CONVERT 0x79 /\* (Japanese keyboard) \*/

#define DIK\_NOCONVERT 0x7B /\* (Japanese keyboard) \*/

#define DIK\_YEN 0x7D /\* (Japanese keyboard) \*/

#define DIK\_ABNT\_C2 0x7E /\* Numpad . on Brazilian keyboard \*/

#define DIK\_NUMPADEQUALS 0x8D /\* = on numeric keypad (NEC PC98) \*/

#define DIK\_PREVTRACK 0x90 /\* Previous Track (DIK\_CIRCUMFLEX on Japanese keyboard) \*/

#define DIK\_AT 0x91 /\* (NEC PC98) \*/

#define DIK\_COLON 0x92 /\* (NEC PC98) \*/

#define DIK\_UNDERLINE 0x93 /\* (NEC PC98) \*/

#define DIK\_KANJI 0x94 /\* (Japanese keyboard) \*/

#define DIK\_STOP 0x95 /\* (NEC PC98) \*/

#define DIK\_AX 0x96 /\* (Japan AX) \*/

#define DIK\_UNLABELED 0x97 /\* (J3100) \*/

#define DIK\_NEXTTRACK 0x99 /\* Next Track \*/

#define DIK\_NUMPADENTER 0x9C /\* Enter on numeric keypad \*/

#define DIK\_RCONTROL 0x9D

#define DIK\_MUTE 0xA0 /\* Mute \*/

#define DIK\_CALCULATOR 0xA1 /\* Calculator \*/

#define DIK\_PLAYPAUSE 0xA2 /\* Play / Pause \*/

#define DIK\_MEDIASTOP 0xA4 /\* Media Stop \*/

#define DIK\_VOLUMEDOWN 0xAE /\* Volume - \*/

#define DIK\_VOLUMEUP 0xB0 /\* Volume + \*/

#define DIK\_WEBHOME 0xB2 /\* Web home \*/

#define DIK\_NUMPADCOMMA 0xB3 /\* , on numeric keypad (NEC PC98) \*/

#define DIK\_DIVIDE 0xB5 /\* / on numeric keypad \*/

#define DIK\_SYSRQ 0xB7

#define DIK\_RMENU 0xB8 /\* right Alt \*/

#define DIK\_PAUSE 0xC5 /\* Pause \*/

#define DIK\_HOME 0xC7 /\* Home on arrow keypad \*/

#define DIK\_UP 0xC8 /\* UpArrow on arrow keypad \*/

#define DIK\_PRIOR 0xC9 /\* PgUp on arrow keypad \*/

#define DIK\_LEFT 0xCB /\* LeftArrow on arrow keypad \*/

#define DIK\_RIGHT 0xCD /\* RightArrow on arrow keypad \*/

#define DIK\_END 0xCF /\* End on arrow keypad \*/

#define DIK\_DOWN 0xD0 /\* DownArrow on arrow keypad \*/

#define DIK\_NEXT 0xD1 /\* PgDn on arrow keypad \*/

#define DIK\_INSERT 0xD2 /\* Insert on arrow keypad \*/

#define DIK\_DELETE 0xD3 /\* Delete on arrow keypad \*/

#define DIK\_LWIN 0xDB /\* Left Windows key \*/

#define DIK\_RWIN 0xDC /\* Right Windows key \*/

#define DIK\_APPS 0xDD /\* AppMenu key \*/

#define DIK\_POWER 0xDE /\* System Power \*/

#define DIK\_SLEEP 0xDF /\* System Sleep \*/

#define DIK\_WAKE 0xE3 /\* System Wake \*/

#define DIK\_WEBSEARCH 0xE5 /\* Web Search \*/

#define DIK\_WEBFAVORITES 0xE6 /\* Web Favorites \*/

#define DIK\_WEBREFRESH 0xE7 /\* Web Refresh \*/

#define DIK\_WEBSTOP 0xE8 /\* Web Stop \*/

#define DIK\_WEBFORWARD 0xE9 /\* Web Forward \*/

#define DIK\_WEBBACK 0xEA /\* Web Back \*/

#define DIK\_MYCOMPUTER 0xEB /\* My Computer \*/

#define DIK\_MAIL 0xEC /\* Mail \*/

#define DIK\_MEDIASELECT 0xED /\* Media Select \*/

/\*

\* Alternate names for keys, to facilitate transition from DOS.

\*/

#define DIK\_BACKSPACE DIK\_BACK /\* backspace \*/

#define DIK\_NUMPADSTAR DIK\_MULTIPLY /\* \* on numeric keypad \*/

#define DIK\_LALT DIK\_LMENU /\* left Alt \*/

#define DIK\_CAPSLOCK DIK\_CAPITAL /\* CapsLock \*/

#define DIK\_NUMPADMINUS DIK\_SUBTRACT /\* - on numeric keypad \*/

#define DIK\_NUMPADPLUS DIK\_ADD /\* + on numeric keypad \*/

#define DIK\_NUMPADPERIOD DIK\_DECIMAL /\* . on numeric keypad \*/

#define DIK\_NUMPADSLASH DIK\_DIVIDE /\* / on numeric keypad \*/

#define DIK\_RALT DIK\_RMENU /\* right Alt \*/

#define DIK\_UPARROW DIK\_UP /\* UpArrow on arrow keypad \*/

#define DIK\_PGUP DIK\_PRIOR /\* PgUp on arrow keypad \*/

#define DIK\_LEFTARROW DIK\_LEFT /\* LeftArrow on arrow keypad \*/

#define DIK\_RIGHTARROW DIK\_RIGHT /\* RightArrow on arrow keypad \*/

#define DIK\_DOWNARROW DIK\_DOWN /\* DownArrow on arrow keypad \*/

#define DIK\_PGDN DIK\_NEXT /\* PgDn on arrow keypad \*/

# Text2D.h

#pragma once

#define \_XM\_NO\_INTRINSICS\_

#define XM\_NO\_ALIGNMENT

#include <d3d11.h>

#include <d3dx11.h>

#include <dxerr.h>

#include <xnamath.h>

#include <vector>

using namespace std;

// store a sting with x and y coords (-1.0 to +1.0), and size (0.0+)

struct string\_2d

{

string s;

float x;

float y;

float size;

};

// Define vertex structure

struct POS\_TEX\_VERTEX

{

XMFLOAT3 Pos;

XMFLOAT2 Texture;

};

// increase if more characters requird

const int MAX\_CHARACTERS = 10000;

class Text2D

{

private:

vector<string\_2d> s2d; // stores list of strings with coords

ID3D11Device\* pD3DDevice;

ID3D11DeviceContext\* pImmediateContext;

ID3D11ShaderResourceView\* pTexture;

ID3D11SamplerState\* pSampler;

POS\_TEX\_VERTEX vertices[MAX\_CHARACTERS \* 6]; // system memory vertex list, to be copied to vertex buffer

ID3D11Buffer\* pVertexBuffer;

ID3D11VertexShader\* pVShader;

ID3D11PixelShader\* pPShader;

ID3D11InputLayout\* pInputLayout;

ID3D11DepthStencilState\* pDepthEnabledStencilState; // state to turn on Z buffer

ID3D11DepthStencilState\* pDepthDisabledStencilState; // state to turn off Z buffer

public:

Text2D(string filename, ID3D11Device\* device, ID3D11DeviceContext\* context); // pass in filename of font, device and context

~Text2D(void);

// add a string with position and size to the list

// positions are from -1.0 to +1.0 for x and y, represents top left of string on screen

// size is fraction of screen size

void AddText(string s, float x, float y, float size) ;

// render all strings at once

void RenderText();

};

# Text2D.cpp

#include "Text2D.h"

// constructor does not handle fails gracefully for simplicity, simply exits

// track errors by looking at debug output or using debugger

// scope for more advanced code to improve error handling

Text2D::Text2D(string filename, ID3D11Device\* device, ID3D11DeviceContext\* context)

{

pD3DDevice = device;

pImmediateContext = context;

// Set up and create vertex buffer

D3D11\_BUFFER\_DESC bufferDesc;

ZeroMemory(&bufferDesc, sizeof(bufferDesc));

bufferDesc.Usage = D3D11\_USAGE\_DYNAMIC; // Used by CPU and GPU

bufferDesc.ByteWidth = sizeof(vertices); // Total size of buffer,

bufferDesc.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER; // Use as a vertex buffer

bufferDesc.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE; // Allow CPU access

HRESULT hr = pD3DDevice->CreateBuffer(&bufferDesc, NULL, &pVertexBuffer); // Create the buffer

if(FAILED(hr)) exit(0);

// Load and compile pixel and vertex shaders - use vs\_5\_0 to target DX11 hardware only

ID3DBlob \*VS, \*PS, \*error;

hr = D3DX11CompileFromFile("text2d\_shaders.hlsl", 0, 0, "TextVS", "vs\_4\_0", 0, 0, 0, &VS, &error, 0);

if(error != 0)

{

OutputDebugStringA((char\*)error->GetBufferPointer());

error->Release();

if(FAILED(hr))exit(0);

}

hr = D3DX11CompileFromFile("text2d\_shaders.hlsl", 0, 0, "TextPS", "ps\_4\_0", 0, 0, 0, &PS, &error, 0);

if(error != 0)

{

OutputDebugStringA((char\*)error->GetBufferPointer());

error->Release();

if(FAILED(hr)) exit(0);

}

// Create shader objects

hr = pD3DDevice->CreateVertexShader(VS->GetBufferPointer(), VS->GetBufferSize(), NULL, &pVShader);

if(FAILED(hr)) exit(0);

hr = pD3DDevice->CreatePixelShader(PS->GetBufferPointer(), PS->GetBufferSize(), NULL, &pPShader);

if(FAILED(hr)) exit(0);

// Create and set the input layout object

D3D11\_INPUT\_ELEMENT\_DESC iedesc[] =

{

{"POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{"TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, D3D11\_APPEND\_ALIGNED\_ELEMENT, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

};

hr = pD3DDevice->CreateInputLayout(iedesc, ARRAYSIZE(iedesc), VS->GetBufferPointer(), VS->GetBufferSize(), &pInputLayout);

if(FAILED(hr)) exit(0);

// Load in the font texture from given filename

hr = D3DX11CreateShaderResourceViewFromFile(pD3DDevice, filename.c\_str(),NULL, NULL, &pTexture,NULL);

if(FAILED(hr)) exit(0);

// Create sampler for texture

D3D11\_SAMPLER\_DESC sampler\_desc;

ZeroMemory(&sampler\_desc, sizeof(sampler\_desc));

sampler\_desc.Filter = D3D11\_FILTER\_MIN\_MAG\_MIP\_POINT;

sampler\_desc.AddressU = D3D11\_TEXTURE\_ADDRESS\_CLAMP;

sampler\_desc.AddressV = D3D11\_TEXTURE\_ADDRESS\_CLAMP;

sampler\_desc.AddressW = D3D11\_TEXTURE\_ADDRESS\_CLAMP;

sampler\_desc.MaxLOD = D3D11\_FLOAT32\_MAX;

hr = pD3DDevice->CreateSamplerState(&sampler\_desc, &pSampler);

// Create 2 depth stencil states to turn Z buffer on and off

D3D11\_DEPTH\_STENCIL\_DESC depthStencilDesc;

ZeroMemory(&depthStencilDesc, sizeof(depthStencilDesc));

depthStencilDesc.DepthEnable = false;

depthStencilDesc.DepthWriteMask = D3D11\_DEPTH\_WRITE\_MASK\_ALL;

depthStencilDesc.DepthFunc = D3D11\_COMPARISON\_LESS;

depthStencilDesc.StencilEnable = true;

depthStencilDesc.StencilReadMask = 0xFF;

depthStencilDesc.StencilWriteMask = 0xFF;

depthStencilDesc.FrontFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

depthStencilDesc.FrontFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_INCR;

depthStencilDesc.FrontFace.StencilPassOp = D3D11\_STENCIL\_OP\_KEEP;

depthStencilDesc.FrontFace.StencilFunc = D3D11\_COMPARISON\_ALWAYS;

depthStencilDesc.BackFace.StencilFailOp = D3D11\_STENCIL\_OP\_KEEP;

depthStencilDesc.BackFace.StencilDepthFailOp = D3D11\_STENCIL\_OP\_DECR;

depthStencilDesc.BackFace.StencilPassOp = D3D11\_STENCIL\_OP\_KEEP;

depthStencilDesc.BackFace.StencilFunc = D3D11\_COMPARISON\_ALWAYS;

hr = device->CreateDepthStencilState(&depthStencilDesc, &pDepthDisabledStencilState);

if(FAILED(hr)) exit(0);

depthStencilDesc.DepthEnable = true;

hr = device->CreateDepthStencilState(&depthStencilDesc, &pDepthEnabledStencilState);

if(FAILED(hr)) exit(0);

}

// add a string with position and size to the list

// positions are from -1.0 to +1.0 for x and y, represents top left of string on screen

// size is fraction of screen size

void Text2D::AddText(string s, float x, float y, float size)

{

string\_2d temp;

temp.s = s ;

temp.x = x;

temp.y = y;

temp.size = size;

s2d.push\_back(temp);

}

// render all text at given positions

// scope for improvement to add alpha blended text

void Text2D::RenderText(void)

{

int current\_char = 0; // keep track of number of characters so far

// loop through all the added string\_2d

for(int i=0; i < s2d.size(); i++)

{

float tempx = s2d[i].x;

float tempy = s2d[i].y;

float temps = s2d[i].size;

// loop through each character

for(int j = 0;j < s2d[i].s.length(); j++)

{

// create 6 vertices with given size and xy position

vertices[current\_char\*6].Pos.x = tempx;

vertices[current\_char\*6].Pos.y = tempy;

vertices[current\_char\*6+1].Pos.x = tempx + temps;

vertices[current\_char\*6+1].Pos.y = tempy;

vertices[current\_char\*6+2].Pos.x = tempx + temps;

vertices[current\_char\*6+2].Pos.y = tempy - temps;

vertices[current\_char\*6+3].Pos.x = tempx;

vertices[current\_char\*6+3].Pos.y = tempy;

vertices[current\_char\*6+4].Pos.x = tempx + temps;

vertices[current\_char\*6+4].Pos.y = tempy - temps;

vertices[current\_char\*6+5].Pos.x = tempx;

vertices[current\_char\*6+5].Pos.y = tempy - temps;

// set all z to 1.0 to avoid being clipped

vertices[current\_char\*6].Pos.z = 1.0;

vertices[current\_char\*6+1].Pos.z = 1.0;

vertices[current\_char\*6+2].Pos.z = 1.0;

vertices[current\_char\*6+3].Pos.z = 1.0;

vertices[current\_char\*6+4].Pos.z = 1.0;

vertices[current\_char\*6+5].Pos.z = 1.0;

// this code assumes 26 characters across and 4 lines in texture, containing lower, upper, number, symbols

const int NUMLINES = 4;

char c = s2d[i].s[j]; // get current character

float texy, texx; // temp tex coords

// determine texture coord to begin at, based on character

// 1/26th added to x texture coord for each subsequent letter in alhpabet

if(c >= 'a' && c <= 'z') // lowercase

{

texy= 0.0; // first line

texx = (c-'a') \* 1.0 / 26.0;

}

else if(c >= 'A' && c <= 'Z') // uppercase

{

texy= 1.0/NUMLINES; //second line

texx = (c-'A') \*1.0 /26.0;

}

else if(c >= '0' && c <= '9') // numbers

{

texy= 2.0/NUMLINES; // third line

texx = (c-'0') \*1.0 /26.0;

}

else // add any symbol code here

{

texx=0;

texy= 3.0/NUMLINES;

//symbols to display can go here

}

// set correct texture coordinates for letter

vertices[current\_char\*6].Texture.x = texx;

vertices[current\_char\*6].Texture.y = texy;

vertices[current\_char\*6+1].Texture.x = texx + 1.0 /26.0;

vertices[current\_char\*6+1].Texture.y = texy;

vertices[current\_char\*6+2].Texture.x =texx + 1.0 /26.0;

vertices[current\_char\*6+2].Texture.y = texy+1.0/NUMLINES;

vertices[current\_char\*6+3].Texture.x = texx;

vertices[current\_char\*6+3].Texture.y = texy;

vertices[current\_char\*6+4].Texture.x = texx + 1.0 /26.0;

vertices[current\_char\*6+4].Texture.y = texy+1.0/NUMLINES;

vertices[current\_char\*6+5].Texture.x = texx;

vertices[current\_char\*6+5].Texture.y = texy+1.0/NUMLINES;

current\_char++;

tempx += temps; // position next character along in x

}

}

// clear out the vector every frame, otherwise will grow forever

s2d.clear();

// Copy the vertices into the buffer

D3D11\_MAPPED\_SUBRESOURCE ms;

pImmediateContext->Map(pVertexBuffer, NULL, D3D11\_MAP\_WRITE\_DISCARD, NULL, &ms); // Lock the buffer to allow writing

memcpy(ms.pData, vertices, sizeof(vertices[0])\*current\_char\*6); // Copy the data - only upload those that are used

pImmediateContext->Unmap(pVertexBuffer, NULL);

// set all rendering states

pImmediateContext->PSSetSamplers(0, 1, &pSampler);

pImmediateContext->PSSetShaderResources(0, 1, &pTexture);

pImmediateContext->VSSetShader(pVShader, 0, 0);

pImmediateContext->PSSetShader(pPShader, 0, 0);

pImmediateContext->IASetInputLayout(pInputLayout);

UINT stride = sizeof(POS\_TEX\_VERTEX);

UINT offset = 0;

pImmediateContext->IASetVertexBuffers(0, 1, &pVertexBuffer, &stride, &offset);

pImmediateContext->IASetPrimitiveTopology( D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST );

// turn off Z buffer so text always on top

pImmediateContext->OMSetDepthStencilState(pDepthDisabledStencilState, 1);

// draw all added characters

pImmediateContext->Draw(current\_char\*6, 0);

// turn on Z buffer so other rendering can use it

pImmediateContext->OMSetDepthStencilState(pDepthEnabledStencilState, 1);

}

Text2D::~Text2D(void)

{

if(pDepthDisabledStencilState) pDepthDisabledStencilState->Release();

if(pDepthEnabledStencilState) pDepthEnabledStencilState->Release();

if(pTexture) pTexture->Release();

if(pSampler) pSampler->Release();

if(pVertexBuffer) pVertexBuffer->Release();

if(pVShader) pVShader->Release();

if(pPShader) pPShader->Release();

if(pInputLayout) pInputLayout->Release();

}

# Text2D\_Shaders.hlsl

Texture2D texture0;

SamplerState sampler0;

struct VOut

{

float4 position : SV\_POSITION;

float2 texcoord :TEXCOORD;

};

VOut TextVS(float4 position : POSITION, float2 texcoord : TEXCOORD)

{

VOut output;

output.position = position;

output.texcoord = texcoord;

return output;

}

float4 TextPS(float4 position : SV\_POSITION, float2 texcoord : TEXCOORD) : SV\_TARGET

{

return texture0.Sample(sampler0, texcoord);

}

# Model.h

#pragma once

#include "objfilemodel.h"

ID3D11Device\* m\_pD3DDevice;

ID3D11DeviceContext\* m\_pImmediateContext;

ObjFileModel\* m\_pObject;

ID3D11VertexShader\* m\_pVShader;

ID3D11PixelShader\* m\_pPShader;

ID3D11InputLayout\* m\_pInputLayout;

ID3D11Buffer\* m\_pConstantBuffer;

float m\_x, m\_y, m\_z;

float m\_xangle, m\_zangle, m\_yangle;

float m\_scale;

class Model

{

private:

float m\_bounding\_sphere\_centre\_x;

float m\_bounding\_sphere\_centre\_y;

float m\_bounding\_sphere\_centre\_z;

float m\_bounding\_sphere\_radius;

void CalculateModelCentrePoint();

void CalculateBoundingSphereRadius();

public:

Model(ID3D11Device\* g\_pD3DDevice, ID3D11DeviceContext\* g\_pImmediateContext);

~Model();

int LoadObjModel(char\* filename);

void Draw(XMMATRIX view, XMMATRIX projection);

XMVECTOR GetBoundingSphereWorldSpacePosition();

float GetBoundingSphereRadius();

bool CheckCollision(Model\* model);

};

# Model.cpp

#include "Model.h"

struct MODEL\_CONSTANT\_BUFFER

{

XMMATRIX WorldViewProjection; // 64 bytes

// Total size = 64 bytes

};

MODEL\_CONSTANT\_BUFFER model\_cb\_values;

model\_cb\_values.WorldViewProjection = world\*(\*view)\*(\*projection);

VSSetConstantBuffer();

Model::Model(ID3D11Device\* g\_pD3DDevice, ID3D11DeviceContext\* g\_pImmediateContext)

{

m\_pD3DDevice = g\_pD3DDevice;

m\_pImmediateContext = g\_pImmediateContext;

m\_x = 0.0f;

m\_y = 0.0f;

m\_z = 0.0f;

m\_xangle = 0.0f;

m\_zangle = 0.0f;

m\_yangle = 0.0f;

m\_scale = 1.0f;

}

int Model::LoadObjModel(char\* filename)

{

ObjFileModel \*pObject;

pObject = new ObjFileModel(filename, m\_pD3DDevice, m\_pImmediateContext);

if (pObject->filename == "FILE NOT LOADED") return S\_FALSE;

// Collision

CalculateModelCentrePoint();

CalculateBoundingSphereRadius();

}

void Model::Draw(XMMATRIX view, XMMATRIX projection)

{

XMMATRIX world, translation, rotation, scale;

rotation = XMMatrixRotationX(XMConvertToRadians(45.0));

scale = XMMatrixScaling(m\_x, m\_y, m\_z);

translation = XMMatrixTranslation(0, 0, 5);

world \*= XMMatrixScaling(m\_scale, m\_scale m\_scale);

}

void Model::CalculateModelCentrePoint()

{

float min\_x = 0;

float max\_x = 0;

float min\_y = 0;

float max\_y = 0;

float min\_z = 0;

float max\_z = 0;

// Determine the minimum and maximum x, y, and z values

for (int i = 0; i < m\_pObject->numverts; i++)

{

// x values

if ((m\_pObject->vertices[i].Pos.x) < min\_x)

{

min\_x = m\_pObject->vertices[i].Pos.x;

}

if ((m\_pObject->vertices[i].Pos.x) > max\_x)

{

max\_x = m\_pObject->vertices[i].Pos.x;

}

// y values

if ((m\_pObject->vertices[i].Pos.y) < min\_y)

{

min\_y = m\_pObject->vertices[i].Pos.y;

}

if ((m\_pObject->vertices[i].Pos.y) > max\_y)

{

max\_y = m\_pObject->vertices[i].Pos.x;

}

// z values

if ((m\_pObject->vertices[i].Pos.z) < min\_z)

{

min\_z = m\_pObject->vertices[i].Pos.z;

}

if ((m\_pObject->vertices[i].Pos.z) > max\_z)

{

max\_z = m\_pObject->vertices[i].Pos.z;

}

}

// Set centre points for sphere

m\_bounding\_sphere\_centre\_x = max\_x - min\_x;

m\_bounding\_sphere\_centre\_y = max\_y - min\_y;

m\_bounding\_sphere\_centre\_z = max\_z - min\_z;

}

void Model::CalculateBoundingSphereRadius()

{

float highest\_distance\_x = 0.0f;

float highest\_distance\_y = 0.0f;

float highest\_distance\_z = 0.0f;

for (int i = 0; i < m\_pObject->numverts; i++)

{

// x value

if ((m\_pObject->vertices[i].Pos.x - m\_bounding\_sphere\_centre\_x) > highest\_distance\_x)

{

highest\_distance\_x = m\_pObject->vertices[i].Pos.x;

}

// y value

if ((m\_pObject->vertices[i].Pos.y - m\_bounding\_sphere\_centre\_y) > highest\_distance\_y)

{

highest\_distance\_y = m\_pObject->vertices[i].Pos.y;

}

// z value

if ((m\_pObject->vertices[i].Pos.z - m\_bounding\_sphere\_centre\_z) > highest\_distance\_z)

{

highest\_distance\_z = m\_pObject->vertices[i].Pos.z;

}

}

// Compare each highest vertex to find the furthest vertex from the centre

if (highest\_distance\_x >= highest\_distance\_y && highest\_distance\_z)

m\_bounding\_sphere\_radius = highest\_distance\_x;

if (highest\_distance\_y >= highest\_distance\_x && highest\_distance\_z)

m\_bounding\_sphere\_radius = highest\_distance\_y;

if (highest\_distance\_z >= highest\_distance\_y && highest\_distance\_x)

m\_bounding\_sphere\_radius = highest\_distance\_z;

}

XMVECTOR Model::GetBoundingSphereWorldSpacePosition()

{

XMMATRIX world, translation, rotation, scale;

scale = XMMatrixScaling(1, 1, 2);

rotation = XMMatrixRotationX(XMConvertToRadians(45.0));

translation = XMMatrixTranslation(0, 0, 5);

world = scale \* rotation \* translation;

XMVECTOR offset;

offset = XMVectorSet(m\_bounding\_sphere\_centre\_x, m\_bounding\_sphere\_centre\_y, m\_bounding\_sphere\_centre\_z, 0.0);

offset = XMVector3Transform(offset, world);

return offset;

}

float Model::GetBoundingSphereRadius()

{

return m\_bounding\_sphere\_radius;

}

bool Model::CheckCollision(Model\* model)

{

if (model == this->LoadObjModel) // returns false if the model passed in is the same as the current model

return false;

XMVECTOR current\_model\_position;

GetBoundingSphereWorldSpacePosition();

current\_model\_position = model->GetBoundingSphereWorldSpacePosition();

float x1 = XMVectorGetX(current\_model\_position);

float x2 = XMVectorGetX(GetBoundingSphereWorldSpacePosition());

float y1 = XMVectorGetY(current\_model\_position);

float y2 = XMVectorGetY(GetBoundingSphereWorldSpacePosition());

float z1 = XMVectorGetZ(current\_model\_position);

float z2 = XMVectorGetZ(GetBoundingSphereWorldSpacePosition());

float distance\_squared = pow((x1 - x2), 2) + pow((y1 - y2), 2) + pow((z1 - z2), 2);

if (distance\_squared < pow((GetBoundingSphereRadius() + model->GetBoundingSphereRadius()), 2)) // Collision!

return true;

return false; // else collision is false

}

# Model\_Shaders.hlsl

cbuffer CB0

{

matrix WVPMatrix; // 64 bytes

// Total size = 64 bytes

};

Texture2D texture0;

SamplerState sampler0;

struct VOut

{

float4 position : SV\_POSITION;

float4 color : COLOR;

float2 texcoord : TEXCOORD;

};

VOut ModelVS(float4 position : POSITION, float2 texcoord : TEXCOORD, float3 normal : NORMAL)

{

VOut output;

float4 default\_color = { 1, 1, 1, 1 };

output.position = mul(WVPMatrix, position);

output.texcoord = texcoord;

output.color = default\_color;

return output;

}

float4 ModelPS(float4 position : SV\_POSITION, float4 color: COLOR, float2 texcoord : TEXCOORD) : SV\_TARGET

{

return texture0.Sample(sampler0, texcoord) \* color;

}

# Objfilemodel.h

#pragma once

#define \_XM\_NO\_INTRINSICS\_

#define XM\_NO\_ALIGNMENT

#include <d3d11.h>

#include <d3dx11.h>

#include <dxerr.h>

#include <xnamath.h>

#include <stdio.h>

#include <string>

#include <vector>

using namespace std;

class ObjFileModel

{

private:

ID3D11Device\* pD3DDevice;

ID3D11DeviceContext\* pImmediateContext;

//////////////////////////////////////////////////

int loadfile(char\* fname);

char\* fbuffer;

long fbuffersize; // filesize

size\_t actualsize; // actual size of loaded data (can be less if loading as text files as ASCII CR (0d) are stripped out)

//////////////////////////////////////////////////

void parsefile();

bool getnextline() ;

bool getnexttoken(int& tokenstart, int& tokenlength);

unsigned int tokenptr;

//////////////////////////////////////////////////

bool createVB();

ID3D11Buffer\* pVertexBuffer;

public:

struct xyz { float x, y, z; }; //used for vertices and normals during file parse

struct xy { float x, y; }; //used for texture coordinates during file parse

// Define model vertex structure

struct MODEL\_POS\_TEX\_NORM\_VERTEX

{

XMFLOAT3 Pos;

XMFLOAT2 TexCoord;

XMFLOAT3 Normal;

};

string filename;

ObjFileModel(char\* filename, ID3D11Device\* device, ID3D11DeviceContext\* context);

~ObjFileModel();

void Draw(void);

vector <xyz> position\_list; // list of parsed positions

vector <xyz> normal\_list; // list of parsed normals

vector <xy> texcoord\_list; // list of parsed texture coordinates

vector <int> pindices, tindices, nindices; // lists of indicies into above lists derived from faces

MODEL\_POS\_TEX\_NORM\_VERTEX\* vertices;

unsigned int numverts;

};

# Objfilemodel.cpp

// turn off fopen warnings

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "ObjFileModel.h"

// draw object

void ObjFileModel::Draw(void)

{

UINT stride = sizeof(MODEL\_POS\_TEX\_NORM\_VERTEX);

UINT offset = 0;

pImmediateContext->IASetVertexBuffers(0, 1, &pVertexBuffer, &stride, &offset);

pImmediateContext->Draw(numverts, 0);

}

// load object from obj file in constructor

ObjFileModel::ObjFileModel(char\* fname, ID3D11Device\* device, ID3D11DeviceContext\* context)

{

pD3DDevice = device;

pImmediateContext = context;

if(loadfile(fname)==0)

{

// file not loaded, check debug output;

filename="FILE NOT LOADED";

return;

}

filename = fname;

parsefile();

createVB();

delete[] fbuffer; // delete file buffer created in loadfile()

}

// load wavefront object file. adds terminating \n so last line of file can be correctly parsed as a 'line' later

// basic loader - only deals with vertices v, texcoords vt, normals vn

// - only copes with triangular meshes (no quads)

// - doesn't deal with textures or materials

int ObjFileModel::loadfile(char\* fname)

{

FILE\* pFile;

pFile = fopen(fname , "r"); // if changed to bin format will read carriage return \r (0d) as well as \n (0a) into fbuffer, may need to add \r checks(but seemed to work with basic test)

if (pFile==NULL) { DXTRACE\_MSG("Failed to open model file");DXTRACE\_MSG(fname); return 0 ;}

// get file size

fseek(pFile, 0, SEEK\_END);

fbuffersize = ftell(pFile);

rewind(pFile);

// allocate memory for entire file size

fbuffer = new char[fbuffersize+1]; // 1 added to cope with adding a \n later in case file doesn't end with \n

if (fbuffer == NULL) {fclose(pFile); DXTRACE\_MSG("Failed allocate memory for model file");DXTRACE\_MSG(fname); return 0 ;}

// copy file into memory

actualsize = fread(fbuffer,1,fbuffersize,pFile); // actualsize may be less than fbuffersize in text mode as \r are stripped

if (actualsize == 0) {fclose(pFile); DXTRACE\_MSG("Failed to read model file");DXTRACE\_MSG(fname); return 0 ;}

// add a newline at end in case file does not, so can deal with whole buffer as a set of lines of text

fbuffer[actualsize] = '\n'; fclose(pFile);

return 1;

}

// uses concept of getting parsable tokens seperated by whitespace and '/'

// one line of file is parsed at a time, lines seperated by '\n'

void ObjFileModel::parsefile()

{

tokenptr=0; // token pointer points to first element of buffer

int tokenstart, tokenlength;

xyz tempxyz;

xy tempxy;

bool success;

int line=0;

do

{

line++; // keep track of current line number for error reporting

if(!getnexttoken(tokenstart, tokenlength)) continue; // get first token on line, go to next line if first token is \n

// ADD FURTHER KEYWORDS HERE TO EXTEND CAPABILITIES

if(strncmp(&fbuffer[tokenstart], "v ", 2)==0) // VERTEX POSITION - note the space in the string is needed (see vt, etc)

{

success=true; // used to see if correct number of tokens left on line for this type of attribute

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.x = (float) atof(&fbuffer[tokenstart]);

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.y = (float) atof(&fbuffer[tokenstart]);

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.z = (float) atof(&fbuffer[tokenstart]);

// if not correct number of tokens, display error in debug output

if(!success) {char s[100] = "ERROR: Badly formatted vertex, line : "; \_itoa(line, &s[strlen(s)], 10); strcat(s, " : "); strcat(s, filename.c\_str()); DXTRACE\_MSG(s); }

position\_list.push\_back(tempxyz); // add a new element to the list

}

else if(strncmp(&fbuffer[tokenstart], "vt", 2)==0) // TEXTURE COORDINATES

{

success=true;

success = success && getnexttoken(tokenstart, tokenlength);

tempxy.x = (float) atof(&fbuffer[tokenstart]);

success = success && getnexttoken(tokenstart, tokenlength);

tempxy.y = (float) atof(&fbuffer[tokenstart]);

if(!success) {char s[100] = "ERROR: Badly formatted texture coordinate, line : "; \_itoa(line, &s[strlen(s)], 10); strcat(s, " : "); strcat(s, filename.c\_str()); DXTRACE\_MSG(s); }

texcoord\_list.push\_back(tempxy);

}

else if(strncmp(&fbuffer[tokenstart], "vn", 2)==0) // NORMALS

{

success=true;

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.x = (float) atof(&fbuffer[tokenstart]);

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.y = (float) atof(&fbuffer[tokenstart]);

success = success && getnexttoken(tokenstart, tokenlength);

tempxyz.z = (float) atof(&fbuffer[tokenstart]);

if(!success) {char s[100] = "ERROR: Badly formatted normal, line : "; \_itoa(line, &s[strlen(s)], 10); strcat(s, " : "); strcat(s, filename.c\_str()); DXTRACE\_MSG(s); }

normal\_list.push\_back(tempxyz);

}

else if(strncmp(&fbuffer[tokenstart], "f ", 2)==0) // FACE - only deals with triangles so far

{

int tempptr = tokenstart + 2; // skip "f "

int forwardslashcount=0;

bool adjacentslash = false;

// this works out how many elements are specified for a face, e.g.

// f 1 2 3 -> 0 forward slashes = just position

// f 1/1 2/2 3/3 -> 3 slashes = position and texcoords

// f 1/1/1 2/2/2 3/3/3 -> 6 slashes = position, texcoords, normals

// f 1//1 2//2 3//3 -> 6 slashes with adjacent = position, normals

while(fbuffer[tempptr] != '\n')

{

if(fbuffer[tempptr] == '/')

{

forwardslashcount++;

if(fbuffer[tempptr-1] == '/') adjacentslash=true;

}

tempptr++;

}

success=true;

// Get 3 sets of indices per face

for(int i=0; i<3; i++)

{

// get vertex index

success = success && getnexttoken(tokenstart, tokenlength);

pindices.push\_back(atoi(&fbuffer[tokenstart]));

if(forwardslashcount>=3&& adjacentslash==false) // get texcoord index if required

{

success = success && getnexttoken(tokenstart, tokenlength);

tindices.push\_back(atoi(&fbuffer[tokenstart]));

}

if(forwardslashcount==6 || adjacentslash==true) // get normal index if required

{

success = success && getnexttoken(tokenstart, tokenlength);

nindices.push\_back(atoi(&fbuffer[tokenstart]));

}

}

if(!success) {char s[100] = "ERROR: Badly formatted face, line : "; \_itoa(line, &s[strlen(s)], 10); strcat(s, " : "); strcat(s, filename.c\_str()); DXTRACE\_MSG(s); }

}

} while(getnextline() == true);

}

// get next token. if \n is next token do not proceed, use getnextline() to resume

bool ObjFileModel::getnexttoken(int& tokenstart, int& tokenlength)

{

tokenstart = tokenptr;

tokenlength=1;

int tokenend;

while(fbuffer[tokenptr] == ' ' || fbuffer[tokenptr] == '\t' || fbuffer[tokenptr] == '/') tokenptr++; //skip whitespace and '/'

if(fbuffer[tokenptr] == '\n') { return false; } // keeps tokenptr pointing to \n as a token to indicate end of line

// doesn't point to next token, dealt with in getnextline()

tokenend=tokenptr+1;

while(fbuffer[tokenend] != ' ' && fbuffer[tokenend] != '\t' && fbuffer[tokenend] != '\n' && fbuffer[tokenend] != '/') tokenend++; // find length of token by finding next whitespace or '\n' or '/'

tokenlength = tokenend - tokenptr;

tokenstart = tokenptr;

tokenptr+=tokenlength; //ready for next token

return true;

}

// gets next line of buffer by skipping to next element after end of current line, returns false when end of buffer exceeded

bool ObjFileModel::getnextline()

{

// relies on getnexttoken()leaving tokenptr pointing to \n if encountered

while(fbuffer[tokenptr] != '\n' && tokenptr < actualsize) tokenptr++; // skip to end of line

tokenptr++; // point to start of next line

if (tokenptr >= actualsize) return false;

else return true;

}

// create Vertex buffer from parsed file data

bool ObjFileModel::createVB()

{

// create vertex array to pass to vertex buffer from parsed data

numverts = pindices.size();

vertices = new MODEL\_POS\_TEX\_NORM\_VERTEX[numverts]; // create big enough vertex array

for(unsigned int i = 0; i< numverts; i++)

{

int vindex = pindices[i]-1; // use -1 for indices as .obj files indices begin at 1

// set position data

vertices[i].Pos.x = position\_list[vindex].x;

vertices[i].Pos.y = position\_list[vindex].y;

vertices[i].Pos.z = position\_list[vindex].z;

if(tindices.size() > 0)

{

// if there are any, set texture coord data

int tindex = tindices[i]-1;

vertices[i].TexCoord.x = texcoord\_list[tindex].x;

vertices[i].TexCoord.y = texcoord\_list[tindex].y;

}

if(nindices.size() > 0)

{

// if there are any, set normal data

int nindex = nindices[i]-1;

vertices[i].Normal.x = normal\_list[nindex].x;

vertices[i].Normal.y = normal\_list[nindex].y;

vertices[i].Normal.z = normal\_list[nindex].z;

}

}

// Set up and create vertex buffer

D3D11\_BUFFER\_DESC bufferDesc;

ZeroMemory(&bufferDesc, sizeof(bufferDesc));

bufferDesc.Usage = D3D11\_USAGE\_DYNAMIC; // Used by CPU and GPU

bufferDesc.ByteWidth = sizeof(vertices[0])\*numverts; // Total size of buffer

bufferDesc.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER; // Use as a vertex buffer

bufferDesc.CPUAccessFlags = D3D11\_CPU\_ACCESS\_WRITE; // Allow CPU access

HRESULT hr = pD3DDevice->CreateBuffer(&bufferDesc, NULL, &pVertexBuffer); // Create the buffer

if(FAILED(hr))

{

return false;

}

// Copy the vertices into the buffer

D3D11\_MAPPED\_SUBRESOURCE ms;

pImmediateContext->Map(pVertexBuffer, NULL, D3D11\_MAP\_WRITE\_DISCARD, NULL, &ms); // Lock the buffer to allow writing

memcpy(ms.pData, vertices, sizeof(vertices[0])\*numverts); // Copy the data

pImmediateContext->Unmap(pVertexBuffer, NULL); // Unlock the buffer

return true;

}

ObjFileModel::~ObjFileModel()

{

// clean up memory used by object

if(pVertexBuffer) pVertexBuffer->Release();

delete [] vertices;

position\_list.clear();

normal\_list.clear();

texcoord\_list.clear();

}