#include "Direct3D.h"

#include <string>

**namespace**

{

Direct3D \*g\_pDirect3D = nullptr;

//Model \*testModel = nullptr;

}

LRESULT CALLBACK lpfnWndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

**if**(g\_pDirect3D)

**return** g\_pDirect3D->MsgProc(hWnd, msg, wParam, lParam);

**else**

**return** DefWindowProc(hWnd, msg, wParam, lParam); // if no, we still need to return the message to something. so default will do.

}

Direct3D::Direct3D(HINSTANCE hInstance)

{

m\_hInstance = hInstance;

m\_hwndAppWindow = NULL;

m\_nClientHeight = GetSystemMetrics(SM\_CYSCREEN);

m\_nClientWidth = GetSystemMetrics(SM\_CXSCREEN);

m\_chAppTitle = "-----";

m\_dwWindowStyle = WS\_EX\_TOPMOST | WS\_POPUP; // maximize, minimize, etc... (standard window)

g\_pDirect3D = **this**;

}

**int** Direct3D::Run()

{

MSG msg = { 0 }; // init all members of struct to zero.

**while**((msg.wParam != VK\_ESCAPE) || (msg.message == WM\_QUIT)) // (msg.wParam != VK\_ESCAPE) ||

{

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

**else**

{

// update

Update(0.01f);

// render

Render(0.0f);

}

}

**return** **static\_cast**<**int**>(msg.wParam);

}

**bool** Direct3D::Init()

{

**if**(!InitializeWindow())

**return** **false**;

**else**

{

**return** **true**;

}

}

**bool** Direct3D::InitializeWindow()

{

WNDCLASSEX wcex;

ZeroMemory(&wcex, **sizeof**(wcex));

wcex.cbClsExtra = 0; //no extra bytes

wcex.cbWndExtra = 0; //no extra bytes

wcex.cbSize = **sizeof**(WNDCLASSEX); //set size in bytes

wcex.style = CS\_HREDRAW | CS\_VREDRAW; //Basically states that window should be redrawn both HORIZ. and VERT (when stretched)

wcex.hInstance = m\_hInstance; //Set handle to application instance;

wcex.lpfnWndProc = (WNDPROC)lpfnWndProc; //Set message procedure to our globally defined one

wcex.hIcon = LoadIcon(NULL, IDI\_APPLICATION); //Set window icon (standard application icon)

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW); //Set window arrow (standard windows arrow)

wcex.hbrBackground = (HBRUSH)GetStockObject(BLACK\_BRUSH); //Set clear background

wcex.lpszClassName = "WIN32WINDOWCLASS"; //Name it w.e you like.

wcex.lpszMenuName = NULL; //We are not using a menu at this time.

wcex.hIconSm = LoadIcon(NULL, IDI\_APPLICATION); //Set small window icon (standard application icon)

**if**(!RegisterClassEx(&wcex))

{

OutputDebugString("\nFAILED TO REGISTER CLASS (WCEX)");

**return** **false**;

}

m\_hwndAppWindow = CreateWindow("WIN32WINDOWCLASS", m\_chAppTitle, m\_dwWindowStyle, 0, 0,

m\_nClientWidth, m\_nClientHeight, NULL, NULL, m\_hInstance, NULL);

**if**(!m\_hwndAppWindow)

{

OutputDebugString("\nFAILED TO CREATE WINDOW (HWND)");

**return** **false**;

}

ShowWindow(m\_hwndAppWindow, SW\_SHOW);

**return** **true**;

}

// here we implement what we actually DO when a message comes from the window

// in lparam and wpram are stored important messages like mouse position

// and which key is down or up.

LRESULT Direct3D::MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

**switch**(msg)

{

**case** WM\_DESTROY:

PostQuitMessage(0); // sends this to the Run() procedure!

**return** 0;

**default**:

DefWindowProc(hWnd, msg, wParam, lParam);

}

}

**void** Direct3D::drawInfo()

{

D3DXFONT\_DESC FontDesc = {24,

0,

400,

0,

**false**,

DEFAULT\_CHARSET,

OUT\_TT\_PRECIS,

CLIP\_DEFAULT\_PRECIS,

DEFAULT\_PITCH,

"Arial"};

RECT FontPosition;

FontPosition.top = 0;

FontPosition.left = 0;

FontPosition.right = GetSystemMetrics(SM\_CXSCREEN);

FontPosition.bottom =GetSystemMetrics(SM\_CYSCREEN);

D3DXCreateFontIndirect(m\_pDevice,&FontDesc,&g\_Font);

FontPosition.top = 0; //position

g\_Font->DrawText(NULL,

"An investigation into 3D",

-1,

&FontPosition,

DT\_CENTER,

D3DCOLOR\_XRGB(255, 0, 0)); //draw text

}

**void** Direct3D::statistics(**int** rotID)

{

D3DXFONT\_DESC FontDesc = {24,

0,

400,

0,

**false**,

DEFAULT\_CHARSET,

OUT\_TT\_PRECIS,

CLIP\_DEFAULT\_PRECIS,

DEFAULT\_PITCH,

"Arial"};

RECT FontPosition;

FontPosition.left = 900;

FontPosition.top = 40;

FontPosition.right = GetSystemMetrics(SM\_CXSCREEN);

FontPosition.bottom = GetSystemMetrics(SM\_CYSCREEN);

D3DXCreateFontIndirect(m\_pDevice,&FontDesc,&g\_Font);

**const** **char** \*szText;

**switch**(rotID)

{

**case** 1: szText = "Rotating in X by 0.05 radians.";

**break**;

**case** 2: szText = "Rotating in Y by 0.05 radians.";

**break**;

**case** 3:

szText = "Translating in X by 0.1.";

**break**;

**case** 4:

szText = "Translating in Y by 0.1";

**break**;

**case** 5:

szText = "Translating in Z by 0.1";

**break**;

}

g\_Font->DrawText(NULL,

szText,

-1,

&FontPosition,

DT\_CENTER,

D3DCOLOR\_XRGB(0, 0, 0)); //draw text

}

**int** Direct3D::getFPS()

{

}

**void** Direct3D::drawMenu()

{

D3DXFONT\_DESC FontDesc = {24,

0,

400,

0,

**false**,

DEFAULT\_CHARSET,

OUT\_TT\_PRECIS,

CLIP\_DEFAULT\_PRECIS,

DEFAULT\_PITCH,

"Arial"};

RECT FontPosition;

FontPosition.left = 0;

FontPosition.top = 40;

FontPosition.right = GetSystemMetrics(SM\_CXSCREEN);

FontPosition.bottom = GetSystemMetrics(SM\_CYSCREEN);

D3DXCreateFontIndirect(m\_pDevice,&FontDesc,&g\_Font);

g\_Font->DrawText(NULL,

"Using the up and down arrows demonstrate x-rotation (rotation about the x-axis). \nLeft and right demonstrate y-rotation. The left and right arrow click will demonstrate panning in and out, achieved via camera displacement. \n\nPressing the O key will change what is called the object render state, possible states are wireframe and solid - wireframe\n demonstrates how triangles are combined into quads to build 3D shapes. \n\n Using the ESDF keys will translate the object across the screen via a Translation Matrix in the direction desired. \n\nTo automate rotation about x, y and z press the O key. \n\nWhen you are ready, press the space bar to start the demonstration.",

-1,

&FontPosition,

DT\_CENTER,

D3DCOLOR\_XRGB(0, 0, 0)); //draw text

}

**bool** Direct3D::InitializeD3D()

{

HRESULT hr;

m\_pDirect3D = Direct3DCreate9(D3D\_SDK\_VERSION);

**if**(!m\_pDirect3D)

{

MessageBoxA(NULL, "FAILED", "Failed to create d3d", MB\_ICONERROR);

OutputDebugString("\nFailed to create D3D version.");

**return** **false**;

}

ZeroMemory(&m\_pD3dpp, **sizeof**(m\_pD3dpp));

m\_pD3dpp.Windowed = **true**;

m\_pD3dpp.SwapEffect = D3DSWAPEFFECT\_DISCARD;

m\_pD3dpp.BackBufferFormat = D3DFMT\_R5G6B5;

m\_pD3dpp.BackBufferCount = 1;

m\_pD3dpp.BackBufferHeight = m\_nClientHeight;

m\_pD3dpp.BackBufferWidth = m\_nClientWidth;

m\_pD3dpp.hDeviceWindow = m\_hwndAppWindow;

m\_pD3dpp.AutoDepthStencilFormat = D3DFMT\_D16;

m\_pD3dpp.EnableAutoDepthStencil = 1;

m\_pD3dpp.FullScreen\_RefreshRateInHz = D3DPRESENT\_RATE\_DEFAULT;

m\_pD3dpp.PresentationInterval = D3DPRESENT\_INTERVAL\_DEFAULT;

//Create the Video Device

hr = m\_pDirect3D->CreateDevice( D3DADAPTER\_DEFAULT, //The default adapter is the primary display adapter

D3DDEVTYPE\_HAL, //the HAL (hardware accelerated layer) uses your 3d accelerator card

m\_hwndAppWindow,

D3DCREATE\_HARDWARE\_VERTEXPROCESSING, //sets the graphic card to do the hardware vertexprocessing

&m\_pD3dpp, //The present parameters we created above

&m\_pDevice

);

**if**( FAILED(hr)){

**return** **false**;

}

m\_pDevice->SetRenderState(D3DRS\_CULLMODE, D3DCULL\_CCW);

m\_pDevice->SetRenderState(D3DRS\_ZENABLE, TRUE);

m\_pDevice->SetRenderState(D3DRS\_NORMALIZENORMALS, TRUE); //this normalizes the normal values (this is important for how lighting effects your models)

// DRAW SHAPES, MODELS, ETC...

// setup view, light, camera

**return** **true**;

}

**void** Direct3D::BeginScene()

{

**if**(m\_pDevice == NULL)

**return**;

m\_pDevice->Clear(0, NULL, D3DCLEAR\_TARGET|D3DCLEAR\_ZBUFFER, D3DCOLOR\_XRGB(55,55,55), 1.0f, 0);

m\_pDevice->BeginScene();

}

**void** Direct3D::EndScene()

{

m\_pDevice->EndScene();

m\_pDevice->Present(0, 0, 0, 0);

}

**void** Direct3D::ViewSetup()

{

D3DXMATRIX projectionMatrix;

D3DXMATRIX viewMatrix;

D3DXVECTOR3 position = D3DXVECTOR3(0.0f, 0.0f, 30.0f); //the position of our camera

D3DXVECTOR3 target = D3DXVECTOR3(0.0f, 0.0f, 0.0f); //the lookat target of our camera

**float** aspect = 1.333f; // the aspect ratio of the screen

**float** nearClip = 1.0f; //nearest point at which the objects stop rendering

**float** farClip = 1000.0f; //farthest point at which the objects stop rendering

// set up projection matrix

D3DXMatrixPerspectiveFovLH(&projectionMatrix, D3DX\_PI / 4.0f, aspect, nearClip, farClip);

// set up projection

m\_pDevice->SetTransform(D3DTS\_PROJECTION, &projectionMatrix);

// view

D3DXMatrixLookAtLH(&viewMatrix,

&position, // cam (or eye)

&target, // target

&D3DXVECTOR3(0.0f, 1.0f, 0.0f)); // up direction

// view setup

m\_pDevice->SetTransform(D3DTS\_VIEW, &viewMatrix);

// lighting

m\_pDevice->SetRenderState(D3DRS\_LIGHTING, TRUE);

// how we create a light

D3DLIGHT9 light;

light.Type = D3DLIGHT\_POINT; //point light, lights in every direction.

light.Diffuse.r = light.Diffuse.g = light.Diffuse.b = 1.0f;

light.Specular.r = light.Specular.g = light.Specular.b = 0.0f;

light.Ambient.r = light.Ambient.g = light.Ambient.b = 0.3f;

light.Position = D3DXVECTOR3( 0.0f, 10.0f, 25.0f );

light.Attenuation0 = light.Attenuation1 = light.Attenuation2 = 0.0f;

light.Range = 60.0f;

// setup the light

m\_pDevice->SetLight(0, &light);

// turn on the light

m\_pDevice->LightEnable(0, TRUE);

}

Direct3D::~Direct3D()

{

}