**Appendix C – Code Samples**

**C++ with DirectX**

#include <windows.h>

#include <d3d10.h>

#include <d3dx10.h>

#include "resource.h"

HINSTANCE g\_hInst = NULL;

HWND g\_hWnd = NULL;

D3D10\_DRIVER\_TYPE g\_driverType = D3D10\_DRIVER\_TYPE\_NULL;

ID3D10Device\* g\_pd3dDevice = NULL;

IDXGISwapChain\* g\_pSwapChain = NULL;

ID3D10RenderTargetView\* g\_pRenderTargetView = NULL;

HRESULT InitWindow( HINSTANCE hInstance, int nCmdShow );

HRESULT InitDevice();

void CleanupDevice();

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

void Render();

int WINAPI wWinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow )

{

if( FAILED( InitWindow( hInstance, nCmdShow ) ) )

return 0;

if( FAILED( InitDevice() ) )

{

CleanupDevice();

return 0;

}

MSG msg = {0};

while( WM\_QUIT != msg.message )

{

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

{

TranslateMessage( &msg );

DispatchMessage( &msg );

}

else

{

Render();

}

}

CleanupDevice();

return ( int )msg.wParam;

}

HRESULT InitWindow( HINSTANCE hInstance, int nCmdShow )

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof( WNDCLASSEX );

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon( hInstance, ( LPCTSTR )IDI\_TUTORIAL1 );

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

wcex.hbrBackground = ( HBRUSH )( COLOR\_WINDOW + 1 );

wcex.lpszMenuName = NULL;

wcex.lpszClassName = L"TutorialWindowClass";

wcex.hIconSm = LoadIcon( wcex.hInstance, ( LPCTSTR )IDI\_TUTORIAL1 );

if( !RegisterClassEx( &wcex ) )

return E\_FAIL;

g\_hInst = hInstance;

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

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

g\_hWnd = CreateWindow( L"TutorialWindowClass", L"Direct3D 10 Tutorial 1: Direct3D 10 Basics", 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;

}

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;

default:

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

}

return 0;

}

HRESULT InitDevice()

{

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 |= D3D10\_CREATE\_DEVICE\_DEBUG;

#endif

D3D10\_DRIVER\_TYPE driverTypes[] =

{

D3D10\_DRIVER\_TYPE\_HARDWARE,

D3D10\_DRIVER\_TYPE\_REFERENCE,

};

UINT numDriverTypes = sizeof( driverTypes ) / sizeof( driverTypes[0] );

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 = D3D10CreateDeviceAndSwapChain( NULL, g\_driverType, NULL, createDeviceFlags,

D3D10\_SDK\_VERSION, &sd, &g\_pSwapChain, &g\_pd3dDevice );

if( SUCCEEDED( hr ) )

break;

}

if( FAILED( hr ) )

return hr;

ID3D10Texture2D\* pBackBuffer;

hr = g\_pSwapChain->GetBuffer( 0, \_\_uuidof( ID3D10Texture2D ), ( LPVOID\* )&pBackBuffer );

if( FAILED( hr ) )

return hr;

hr = g\_pd3dDevice->CreateRenderTargetView( pBackBuffer, NULL, &g\_pRenderTargetView );

pBackBuffer->Release();

if( FAILED( hr ) )

return hr;

g\_pd3dDevice->OMSetRenderTargets( 1, &g\_pRenderTargetView, NULL );

D3D10\_VIEWPORT vp;

vp.Width = width;

vp.Height = height;

vp.MinDepth = 0.0f;

vp.MaxDepth = 1.0f;

vp.TopLeftX = 0;

vp.TopLeftY = 0;

g\_pd3dDevice->RSSetViewports( 1, &vp );

return S\_OK;

}

void Render()

{

float ClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };

g\_pd3dDevice->ClearRenderTargetView( g\_pRenderTargetView, ClearColor );

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

}

void CleanupDevice()

{

if( g\_pd3dDevice ) g\_pd3dDevice->ClearState();

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

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

if( g\_pd3dDevice ) g\_pd3dDevice->Release();

}

**C# with XNA**

using System;

using System.Collections.Generic;

using System.Linq;

using Microsoft.Xna.Framework;

using Microsoft.Xna.Framework.Audio;

using Microsoft.Xna.Framework.Content;

using Microsoft.Xna.Framework.GamerServices;

using Microsoft.Xna.Framework.Graphics;

using Microsoft.Xna.Framework.Input;

using Microsoft.Xna.Framework.Media;

using Microsoft.Xna.Framework.Net;

using Microsoft.Xna.Framework.Storage;

namespace TEST

{

public class Game1 : Microsoft.Xna.Framework.Game

{

GraphicsDeviceManager graphics;

SpriteBatch spriteBatch;

public Game1()

{

graphics = new GraphicsDeviceManager(this);

Content.RootDirectory = "Content";

}

protected override void Initialize()

{

base.Initialize();

}

protected override void LoadContent()

{

spriteBatch = new SpriteBatch(GraphicsDevice);

}

protected override void UnloadContent()

{

}

protected override void Update(GameTime gameTime)

{

if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)

this.Exit();

base.Update(gameTime);

}

protected override void Draw(GameTime gameTime)

{

GraphicsDevice.Clear(Color.CornflowerBlue);

base.Draw(gameTime);

}

}

}