P3_Tutorial3 pt. I

This tutorial aims to take you through the stages of creating a Direct3D that loads a bitmap in to the background.
You can find bitmap resources for this program online at activehelix.com/prog.html or they will be distributed in class.

This uses information in the surfaces notes.
There are also some additional functions that have not been seen before; these can be easily referenced by looking at the msdn website and performing an advanced search: msdn.microsoft.com/

Don't forget we need to alter our Visual Studio settings and set our project dependencies.

The Basic Functionality

To Start with we need all the usually Win32 code initialising.

We need to specify our header files:

```
////////// Header Files //////////
#include <windows.h>
#include <d3d9.h>
```

Then we need to declare our function prototypes

```
////////// Function Prototypes //////////
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam);
void RegisterWindowClass(HINSTANCE hInstance);
void CreateAppWindow(HINSTANCE hInstance);
WPARAM StartMessageLoop();
HRESULT InitFullScreenDirect3D();
void Render();
void CleanUpDirect3D();
```

Then we add any global variables

```
////////// Global Variables //////////
HWND g_hWnd;
// Declare pointer to IDirect3D9 interface -
// called g_pDirect3D because it is a global pointer to the interface
IDirect3D9 *g_pDirect3D = NULL;
// pointer to IDirect3DDevice9 interface - initialised to NULL
IDirect3DDevice9* g_pDirect3DDevice = NULL;
```

Next is our WinMain() function, where everything originates from. This has been stripped down with code we normally see added to functions that are called from WinMain().
WinMain() is therefore only acting to call other functions.

```
//////////////////////////////////////////////////////
// Function: WinMain()
// Performs: The entry point of our application
//////////////////////////////////////////////////////
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, INT)
{
      // The program starts here. We just call all the other
functions
    RegisterWindowClass(hInstance);
    CreateAppWindow(hInstance);
    ShowWindow(g_hWnd, SW_SHOWDEFAULT);
    UpdateWindow(g_hWnd);
    HRESULT hResult = InitFullScreenDirect3D();
    if (SUCCEEDED(hResult))
      {
            WPARAM result = StartMessageLoop();
      }
    CleanUpDirect3D();
    return 0;
}
```

Next we add a function to handle any windows messages, as this is platform we are operating one it is required.

```
//////////////////////////////////////////////////////
// Function: WndProc()
// Performs: This handles any incoming Windows messages and sends any
//        that aren't handled to DefWindowProc for Windows to handle.
//////////////////////////////////////////////////////
LRESULT WINAPI WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM
lParam)
{
    switch(msg)
    {
    case WM_CREATE:
        return 0;

    case WM_DESTROY:
        PostQuitMessage( 0 );
        return 0;

    case WM_PAINT:
        ValidateRect(g_hWnd, NULL);
        return 0;
    case WM_KEYDOWN:
        switch(wParam)
        {
            case VK_ESCAPE:
                PostQuitMessage(WM_QUIT);
                break;
        }
    }
    return DefWindowProc(hWnd, msg, wParam, lParam);
}
```

The next code to add is the code to register our window, this function is called from WinMain() and registers our window with the Windows operating system

```
//////////////////////////////////////////////////////
// Function: RegisterWindowClass()
// Performs: registers our window
//////////////////////////////////////////////////////
void RegisterWindowClass(HINSTANCE hInstance)
{
    WNDCLASSEX wc;
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wc.lpfnWndProc = WndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = (HCURSOR)LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = "Direct3DApp";
    wc.hIconSm = NULL;

    RegisterClassEx(&wc);
}
```

This function is again called by WinMain() and this creates our window

```
//////////////////////////////////////////////////////
// Function: CreateAppWindow()
// Performs: creates our window
//////////////////////////////////////////////////////
void CreateAppWindow(HINSTANCE hInstance)
{
    g_hWnd = CreateWindowEx(
        NULL,
        "Direct3DApp",
        "Basic Direct3D Application",
        WS_OVERLAPPEDWINDOW,
        100,
        100,
        648,
        514,
        GetDesktopWindow(),
        NULL,
        hInstance,
        NULL);
}
```

This function starts to see more Directx. In other tutorials or code you may see this defined as the game loop. It is where the processing happens when the program is running. We handle messages here and render (draw) to the window.

```
//////////////////////////////////////////////////////
// Function: StartMessageLoop()
// Performs: message processing for our program
//////////////////////////////////////////////////////
WPARAM StartMessageLoop()
{     //Enter the game loop
    MSG msg;
```

```
    while(1)
    {
        if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {   //Process message
            if (msg.message == WM_QUIT)
                break;

            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            // Use idle time here.
                  // No message to process, so render the current
scene
            Render();
        }
    }
    return msg.wParam;
}
```

This is where we are starting to get into our DirectX coding.
We will define and code functions (see the prototypes previously) that call the relevant DirectX objects and methods necessary for our program.

```
////////// DirectX Functions //////////
```

The first function is there to initialise Direct3D, setting its parameters and to check that our chosen display mode is available.

```
//////////////////////////////////////////////////
// Function: InitFullScreenDirect3D()
// Performs: Creates a Direct3D full screen window
//////////////////////////////////////////////////
HRESULT InitFullScreenDirect3D()
{
      // This is called to obtain a pointer to the Direct3D object
    g_pDirect3D = Direct3DCreate9(D3D_SDK_VERSION);

      // check to see that the call to Direct3DCreate9() didn't fail
      if(!g_pDirect3D)
      {     // If the call failed, handle the error here

            // here we handle the error by returning a DirectX
constant
            // it represents a failure of call to Direct3DCreate9()
            return E_FAIL;
      }

      // check to see if chosen display mode is available
    HRESULT hResult = g_pDirect3D ->
CheckDeviceType(D3DADAPTER_DEFAULT,
        D3DDEVTYPE_REF, D3DFMT_X8R8G8B8, D3DFMT_X8R8G8B8, FALSE);

    if (hResult != D3D_OK)
    {
        MessageBox(g_hWnd,
            "Sorry. This program won't\nrun on your system.",
```

```
            "DirectX Error", MB_OK);
        return E_FAIL;
    }

    D3DPRESENT_PARAMETERS D3DPresentParams;

    ZeroMemory(&D3DPresentParams, sizeof(D3DPRESENT_PARAMETERS));

    D3DPresentParams.Windowed = FALSE;
    D3DPresentParams.BackBufferCount = 1;
    D3DPresentParams.BackBufferWidth = 1024;
    D3DPresentParams.BackBufferHeight = 768;
    D3DPresentParams.BackBufferFormat = D3DFMT_X8R8G8B8;
    D3DPresentParams.SwapEffect = D3DSWAPEFFECT_DISCARD;
    D3DPresentParams.hDeviceWindow = g_hWnd;

      // call Direct3DDevice createdevice() method
    hResult = g_pDirect3D -> CreateDevice(D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL, g_hWnd, D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    &D3DPresentParams, &g_pDirect3DDevice);

    if (FAILED(hResult))
    {
        return E_FAIL;
      }

    return D3D_OK;
}
```

This function cleans up any pointers and memory used by DirectX and is called before the program closes.

```
//////////////////////////////////////////////////////
// Function: CleanUpDirect3D()
// Performs: Releases all interface pointers
//////////////////////////////////////////////////////
void CleanUpDirect3D()
{
    if (g_pDirect3DDevice)
    {
        g_pDirect3DDevice->Release();
      }

    if (g_pDirect3D)
    {
        g_pDirect3D->Release();
      }
}
```

This function is where the actual drawing goes, it is called from the game loop (StartMessageLoop() function).

```
//////////////////////////////////////////////////////
// Function: Render()
// Performs: Clears the screen  and then presents the results.
//         If we were doing any real drawing, it would go in this
function between
//         the BeginScene() & EndScene().
```

```
//////////////////////////////////////////////////
void Render()
{
    g_pDirect3DDevice->Clear(0, 0, D3DCLEAR_TARGET,
        D3DCOLOR_XRGB(0,0,0), 0, 0);

    g_pDirect3DDevice->Present(NULL, NULL, NULL, NULL);
}
```

This so far is the very basic information necessary to initialise and load Direct3D.

Adding Bitmaps as a Background

We now start adding additional functionality. To begin with we are going to add a bitmap as background.

To use this additional functionality we need to include more header files.

Add the following header files to the code in the relevant location:

```
#include <d3dx9tex.h>
#include <dxerr9.h>
```

You will also need to add these as project dependencies in Visual Studio. To do this see the notes supplied last week, titled: **Direct32 Introduction**

As we are dealing with surfaces and bitmaps we are going to need to add a few more global variables.

```
// Global variables.
HWND g_hWnd = NULL;
IDirect3D9 *g_pDirect3D = NULL;
IDirect3DDevice9 *g_pDirect3DDevice = NULL;
IDirect3DSurface9 *g_pBitmapSurface = NULL;
HRESULT g_hResult = D3D_OK;
char g_szErrorMsg[256];
```

We are going to need at add a little more code to WinMain().
Add this between `CleanUpDirect3D();` and `return 0;`
This code is to help us detect and deal with any problems returned by `InitFullScreenDirect3D();`

```
    CloseWindow(g_hWnd);

    if (g_hResult != D3D_OK)
    {
        DXTRACE_ERR(g_szErrorMsg, g_hResult);
      }
```

Next thing that needs altering is the `InitFullScreenDirect3D();` as this where any additional initialisation is occurring.
I have included the full code for the new function.
It calls upon a few more new functions such as `CreateOffscreenPlainSurface`
These functions can be referenced from the msdn website.

```
//////////////////////////////////////////////////////
// InitFullScreenDirect3D()
//////////////////////////////////////////////////////
HRESULT InitFullScreenDirect3D()
{
    g_pDirect3D = Direct3DCreate9(D3D_SDK_VERSION);

    if (g_pDirect3D == NULL)
    {
        MessageBox(g_hWnd,
            "Couldn't create DirectX object.",
            "DirectX Error", MB_OK);
        return E_FAIL;
    }

    HRESULT hResult = g_pDirect3D-
>CheckDeviceType(D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL, D3DFMT_X8R8G8B8, D3DFMT_X8R8G8B8, FALSE);

    if (hResult != D3D_OK)
    {
        MessageBox(g_hWnd,
            "Sorry. This program won't\nrun on your system.",
            "DirectX Error", MB_OK);
        return E_FAIL;
    }

    D3DPRESENT_PARAMETERS D3DPresentParams;
    ZeroMemory(&D3DPresentParams, sizeof(D3DPRESENT_PARAMETERS));
    D3DPresentParams.Windowed = FALSE;
    D3DPresentParams.BackBufferCount = 1;
    D3DPresentParams.BackBufferWidth = 640;
    D3DPresentParams.BackBufferHeight = 480;
    D3DPresentParams.BackBufferFormat = D3DFMT_X8R8G8B8;
    D3DPresentParams.SwapEffect = D3DSWAPEFFECT_DISCARD;
    D3DPresentParams.hDeviceWindow = g_hWnd;

    hResult = g_pDirect3D->CreateDevice(D3DADAPTER_DEFAULT,
    D3DDEVTYPE_HAL, g_hWnd, D3DCREATE_SOFTWARE_VERTEXPROCESSING,
    &D3DPresentParams, &g_pDirect3DDevice);

    if (FAILED(hResult))
    {
        MessageBox(g_hWnd,
            "Failed to create Direct3D device.",
            "DirectX Error", MB_OK);
        return E_FAIL;
    }

    g_hResult = g_pDirect3DDevice->CreateOffscreenPlainSurface(640,
480,
        D3DFMT_X8R8G8B8, D3DPOOL_SYSTEMMEM, &g_pBitmapSurface, NULL);

    if (FAILED(g_hResult))
```

```
    {
        strcpy(g_szErrorMsg, "Error creating bitmap surface.");
        PostQuitMessage(WM_QUIT);
    }

    g_hResult = D3DXLoadSurfaceFromFile(g_pBitmapSurface, NULL, NULL,
        "image.bmp", NULL, D3DX_DEFAULT, 0, NULL);

    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Couldn't load bitmap file.");
        PostQuitMessage(WM_QUIT);
    }

    return D3D_OK;
}
```

We now need to release one more component in the function. This is the bitmap surface. To do this use the following code.

```
  if (g_pBitmapSurface)
  {
    g_pBitmapSurface->Release();
  }
```

Render is the other function that is to under go a large change. As render is performing the drawing it now needs to load the surface into the back buffer, with DirectX 9.0 this has become part of a swap chain that is managed by DirectX. This then needs blitting to the screen (or front buffer), using the function UpdateSurface.

```
//////////////////////////////////////////////////////
// Render()
//////////////////////////////////////////////////////
void Render()
{
    IDirect3DSurface9* pBackBuffer = NULL;

    g_hResult = g_pDirect3DDevice->GetBackBuffer(0,
            0, D3DBACKBUFFER_TYPE_MONO, &pBackBuffer);

    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Error getting back buffer.");
        PostQuitMessage(WM_QUIT);
    }

    g_hResult = g_pDirect3DDevice->UpdateSurface(g_pBitmapSurface,
        NULL,pBackBuffer,NULL);

    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Error copying image buffer.");
        PostQuitMessage(WM_QUIT);
    }

    g_pDirect3DDevice->Present(NULL, NULL, NULL, NULL);
```

```
    if (pBackBuffer)
    {
        pBackBuffer->Release();
      }
}
```

The code should now work and will load up a bitmap called image.bmp, which is located in the source folder, as a background.
Although the image doesn't need to have a width of 640 and a height of 480 (or whatever we want our resolution to be), the numbers in the function `CreateOffscreenPlainSurface` need to be set to the same numbers specified by our `D3DPresentParams`.

See `D3DXLoadSurfaceFromFile` and `CreateOffscreenPlainSurface` in `InitFullScreenDirect3D()`

You made need to quickly find an image or make one in paint shop. It doesn't matter what it looks like, just make sure it is a bitmap and give it a size of 640x480.

There are some improvements that could be made to the code such as making the bitmap filename a const char gc_szFILNAME[] = //our filename
Or even using the string class