P3_Tutorial4 pt.1

This tutorial aims to take you through the basics of Direct3D animation. These basics break down into 4 specifics:

- How all types of animation work
- The general process needed to perform computer animation
- How to use Direct3D to create an animation sequence
- Timing animation sequences

Animation Mechanics

The world is filled with different kinds of animation – movies, flip books, games. All these things have one thing in common: the appearance of motion comes from rapidly projecting a series of images.

DirectX Display Details

We already know how to draw (or display) the animation sequence. The tutorials last week covered the necessary stages:

1) Load animation images into memory
2) Draw animation image to back buffer
3) Display the back buffer
4) Repeat stages 2 and 3

Frames and Lingo

Just to mention that the technical term for images that are part of an animation sequence is “frame”. So that is what I will be referring to when I use the word frame – A single image that is part of an animation sequence

Coding the Animation Sequence

There are two ways to code the animation sequence:

- Using the DirectX defined Sprite and series of RECTS
- Load an series of surfaces

Often in animation, certainly professional animation, bitmap frames are all part of one image. Maths is then used to determine which part of the bitmap to display for each frame.
In DirectX we use a RECTS structure to specify what part of the image to use; this is similar to grabbing animation frames as shown in Programming 2. The 2D code example “Explosions” shows an example of this.

Note on the Tutorial Code

This follows the same principles of the other tutorials; it includes all the code for building a working application of the type we wish to demonstrate.
Comments are included to highlight the areas we need our attention drawing to. Remember though that a lot of the code remains relatively static, such as the message loop. It is therefore ideal to put such code it in separate source files

with prototypes in the headers. The header can then be added to the source file that defines **WinMain()** and the other necessary functions.

Using Surfaces
This tutorial is going to focus on loading a series of surfaces that give the appearance of animation.

We need to specify our header files, global variables and function prototypes.

```
#include <windows.h>
#include <d3d9.h>
#include <d3dx9tex.h>
#include <dxerr9.h>

// 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();
BOOL IsTimeForNextFrame();

// Global variables.
HWND g_hWnd;
IDirect3D9* g_pDirect3D = NULL;
IDirect3DDevice9* g_pDirect3DDevice = NULL;
// Declare Array of 8 surfaces to hold bitmaps
IDirect3DSurface9* g_pBitmapSurfaces[8];
HRESULT g_hResult = D3D_OK;
char g_szErrorMsg[256];
```

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 initialise some of our variables and to call other functions.

```
//////////////////////////////////////////////////////
// WinMain()
//////////////////////////////////////////////////////
INT WINAPI WinMain(HINSTANCE hInstance, HINSTANCE, LPSTR, INT)
{
      // initialise our array of surfaces to NULL
      for (int x=0; x<8; ++x)
            g_pBitmapSurfaces[x] = NULL;

    RegisterWindowClass(hInstance);
    CreateAppWindow(hInstance);
    ShowWindow(g_hWnd, SW_SHOWDEFAULT);
```

```
    UpdateWindow(g_hWnd);
    HRESULT hResult = InitFullScreenDirect3D();

    if (SUCCEEDED(hResult))
        WPARAM result = StartMessageLoop();

    CleanUpDirect3D();
    CloseWindow(g_hWnd);

    if (g_hResult != D3D_OK)
        DXTRACE_ERR(g_szErrorMsg, g_hResult);

    return 0;
}
```

Next we add a function to handle any windows messages, as this is platform we are operating one it is required - `WndProc()`

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 - `RegisterWindowClass()`

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

The next code to add is the message handler - `StartMessageLoop()`

These functions have not changed; therefore the code is not included!

The next function coded is `InitDirect3D` to initialise the elements of our program. It checks for errors and returns relevant error code.

```
//////////////////////////////////////////////////////
// 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 = 800;
    D3DPresentParams.BackBufferHeight = 600;
    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;
    }

      /* This loop loads the surfaces with images, which are the
      8 animation frames. */
    for (int x=0; x<8; ++x)
    { // this section of code creates a surface for each frame
        g_hResult = g_pDirect3DDevice->CreateOffscreenPlainSurface(400,
400, D3DFMT_X8R8G8B8, D3DPOOL_SYSTEMMEM, &g_pBitmapSurfaces[x], NULL);

        if (FAILED(g_hResult))
        {
            strcpy(g_szErrorMsg, "Error creating bitmap surface.");
            PostQuitMessage(WM_QUIT);
            break;
        }

      // this section of code loads the surfaces with a bitmaps frame
        char filename[15];    // where we store our filename
      // dyanamically create our filename
        wsprintf(filename, "%s%d%s", "Frame0", x+1, ".bmp");

        g_hResult = D3DXLoadSurfaceFromFile(g_pBitmapSurfaces[x], NULL,
NULL, filename, NULL, D3DX_DEFAULT, 0, NULL);

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

```
    g_pDirect3DDevice->Clear(0, NULL, D3DCLEAR_TARGET,
        D3DCOLOR_XRGB(0,0,0), 1.0f, 0);

    return D3D_OK;
}
```

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

```
//////////////////////////////////////////////////////
// CleanUpDirect3D()
//////////////////////////////////////////////////////
void CleanUpDirect3D()
{
    for (int x=0; x<8; ++x)
    {
        if (g_pBitmapSurfaces[x])
            g_pBitmapSurfaces[x]->Release();
    }
    if (g_pDirect3DDevice)
        g_pDirect3DDevice->Release();
    if (g_pDirect3D)
        g_pDirect3D->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.

```
//////////////////////////////////////////////////////
// Render()
//////////////////////////////////////////////////////
void Render()
{     // call the timer function - when it returns
      // true it is time to display our next frame
    if (IsTimeForNextFrame())
    { // update our frame counter
        static frameNum = 0;
        frameNum = frameNum + 1;
        if (frameNum == 8)
            frameNum = 0;

            // access the backbuffer
        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);
            return;
        }
```

```
      // Clears the viewport, or a set of rectangles in the viewport,
      // to a specified RGBA color, clears the depth buffer,
      // and erases the stencil buffer.
        g_pDirect3DDevice->Clear(0, NULL, D3DCLEAR_TARGET,
            D3DCOLOR_XRGB(0,0,0), 1.0f, 0);

      // initialise a RECT
        RECT SrcRect;
        SrcRect.left = 0; SrcRect.right = 399;
        SrcRect.top = 0; SrcRect.bottom = 399;
      // initialise a POINT
        POINT DstPoint;
        DstPoint.x = 200;
        DstPoint.y = 100;

      // Copies rectangular subsets of pixels from one surface to
      // another.
      // In our case we are copying the bitmap to the backbuffer
        g_hResult = g_pDirect3DDevice->UpdateSurface(
            g_pBitmapSurfaces[frameNum],
            &SrcRect, pBackBuffer, &DstPoint);

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

      // present the backbuffer to the screen
        g_pDirect3DDevice->Present(NULL, NULL, NULL, NULL);

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

The last function is a very basic timer that just uses a for loop, it is called from the render function so it knows when to update and display the next frame:

```
//////////////////////////////////////////////////
// IsTimeForNextFrame()
//////////////////////////////////////////////////
BOOL IsTimeForNextFrame()
{
      // a very basic timer that uses a for loop
      // professional timers will be covered later
    static DWORD count = 0;
    count = count + 1;
    if (count > 100000)
    {
        count = 0;
        return TRUE;
    }
    else
```

```
        {
          return FALSE;
        }
}
```

Exercises

1) Modify the code so that the animation runs at half speed
2) Modify the program so that it displays only a 200x200 image from the centre of each frame
3) Move the display so that the animation runs in the lower left corner of the screen