Programming 3 Tutorial 3

Triangle Drawing

In this we are going to code a triangle and then colour that triangle in.

You can find bitmap and basic Direct3D application resources for this program online at www.activehelix.com/prog3.html
So obtain these by downloading tutorial3code.zip from the above link.

Representing our Vertex

In your code we can represent a vertex as a structure, this means that you can customise and alter the structure to have member data based on what is required.
In this tutorial we will represent a vertex with the following structure:

```
struct CUSTOMVERTEX
{
    FLOAT x, y, z, rhw;
    DWORD colour;
};
```

It would be a good idea to put this structure in to a separate header file.

Message Loop

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.

```
//////////////////////////////////////////////////
// StartMessageLoop()
//////////////////////////////////////////////////
WPARAM StartMessageLoop()
{
    MSG msg;
    while(1)
    {
        if (PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
        {
            if (msg.message == WM_QUIT)
                break;
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
        else
        {
            // Use idle time here.
            Render();
        }
    }
    return msg.wParam;
}
```

Rendering

Our code is pretty much going to look very similar to the basic Direct3D application, with one exception the code needed to render to the vertices. We are putting this into a function called render.

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

```
//////////////////////////////////////////////////
// Render()
//////////////////////////////////////////////////
void Render()
{
      // An array to hold information on our vertices
      // this one is set to draw a triangle
    CUSTOMVERTEX triangleVertices[] =
    {
        {400.0f, 180.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(255,0,0),},
        {500.0f, 380.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(0,255,0),},
        {300.0f, 380.0f, 0.0f, 1.0f, D3DCOLOR_XRGB(0,0,255),},
    };

      // create and interface object for our vertices
    IDirect3DVertexBuffer9* pVertexBuf = NULL;

      // create a vertex buffer....
    HRESULT hResult = g_pDirect3DDevice->CreateVertexBuffer(
        3*sizeof(CUSTOMVERTEX), 0, D3DFVF_XYZRHW | D3DFVF_DIFFUSE,
        D3DPOOL_DEFAULT, &pVertexBuf, NULL);

    if(FAILED(hResult))
    {
        DXTRACE_ERR("Error creating vertex buffer", hResult);
        return;
    }

      // Fill the buffers with the triangle data.
    VOID* pVertices;
    hResult = pVertexBuf->Lock(0, 0, (void**)&pVertices, 0);
    if(FAILED(hResult))
    {
        DXTRACE_ERR("Error locking vertex buffer", hResult);
        return;
    }

    memcpy(pVertices, triangleVertices, sizeof(triangleVertices));
    pVertexBuf->Unlock();

    g_pDirect3DDevice->Clear(0, NULL, D3DCLEAR_TARGET,
        D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );
    g_pDirect3DDevice->SetStreamSource(0, pVertexBuf, 0,
        sizeof(CUSTOMVERTEX));

      // set the way the vertex is formatted, our vertex
      // contains a position and diffuse colour property
```

```
      g_pDirect3DDevice->SetFVF(D3DFVF_XYZRHW | D3DFVF_DIFFUSE);

    g_pDirect3DDevice->BeginScene();
      // Draw one triangle
    g_pDirect3DDevice->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 1);
    g_pDirect3DDevice->EndScene();

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

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

The code isn't as functional as it could be many of the functions (especially the Win32 functions) we use throughout these tutorials.
Putting these in a header file along with definitions for additional global constants (const unsigned int) for attributes used more than once, height and width of windows would be a way to improve code reduce and make altering things such as resolution easier, just change to values in the global constants and it takes effect throughout the code.

Exercises

1) Modify the program so it displays a completely red triangle
2) Modify the program to display the triangle in the lower-left corner
3) Modify the program so that it displays three triangles on screen (in different locations of the screen)
4) Modify the program so that is displays a wire frame triangle (no colour)

Tutorial Part 2 – Mapping a Texture

Then we need to declare our vertex structure.
Notice our vertex structure is different for this program. The reason is that this time we need texture coordinates as we will be loading texture coordinates. We don't need colour value as the texture them selves are coloured.

Each vertex that is defined from this data type will have its own 3D coordinates and texture coordinates. These work together to tell Direct3D where to draw the primitive and what part of the texture to apply to the primitive.

```
struct CUSTOMVERTEX
{
    float x, y, z, rhw;
      // texture coordinates
    float tu,tv;
};
```

When coding these next bits pay careful attention to any additional information, this is where we will be learning to display images using vertices.

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

```
//////////////////////////////////////////////////////
// InitFullScreenDirect3D()
//////////////////////////////////////////////////////
HRESULT InitFullScreenDirect3D()
{
    g_hResult = InitDirect3D();
    if (FAILED(g_hResult))
        return g_hResult;
    g_hResult = CreateSurfaces();
    if (FAILED(g_hResult))
        return g_hResult;
    g_hResult = CreateVertices();
    if (FAILED(g_hResult))
        return g_hResult;
    return D3D_OK;
}
```

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

```
//////////////////////////////////////////////////////
// CleanUpDirect3D()
//////////////////////////////////////////////////////
void CleanUpDirect3D()
{
    if (g_pBitmapSurface)
        g_pBitmapSurface->Release();
    if (g_pTexture)
        g_pTexture->Release();
    if (g_pVertexBuf)
        g_pVertexBuf->Release();
    if (g_pDirect3DDevice)
        g_pDirect3DDevice->Release();
    if (g_pDirect3D)
        g_pDirect3D->Release();
}
```

This function is where the actual drawing goes, now it calls to functions to carry out the relevant aspects of the drawing. It is called from the game loop (StartMessageLoop() function).

```
//////////////////////////////////////////////////////
// Render()
//////////////////////////////////////////////////////
void Render()
```

```
{
    g_pDirect3DDevice->Clear(0, NULL, D3DCLEAR_TARGET,
        D3DCOLOR_XRGB(0,0,0), 1.0f, 0 );
    PaintBackground();
    PaintTexture();
}
```

This function loads in the background bitmap

```
//////////////////////////////////////////////////////
// CreateSurfaces()
//////////////////////////////////////////////////////
HRESULT CreateSurfaces()
{
      g_hResult = g_pDirect3DDevice->CreateOffscreenPlainSurface(800,
600,
    D3DFMT_X8R8G8B8, D3DPOOL_SYSTEMMEM, &g_pBitmapSurface, NULL);

    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Error creating bitmap surface.");
        PostQuitMessage(WM_QUIT);
        return g_hResult;
    }
    g_hResult = D3DXLoadSurfaceFromFile(g_pBitmapSurface, NULL, NULL,
        "image02.bmp", NULL, D3DX_DEFAULT, 0, NULL);
    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Couldn't load bitmap file.");
        PostQuitMessage(WM_QUIT);
        return g_hResult;
    }

      // this function loads a file in memory as a texture
    g_hResult = D3DXCreateTextureFromFileEx(g_pDirect3DDevice,
"Texture.bmp",
        D3DX_DEFAULT, D3DX_DEFAULT, 1, 0, D3DFMT_A8R8G8B8,
        D3DPOOL_MANAGED, D3DX_DEFAULT, D3DX_DEFAULT, 0xFF000000,
        NULL, NULL, &g_pTexture);
    if(FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Couldn't load texture file.");
        PostQuitMessage(WM_QUIT);
        return g_hResult;
    }

    return D3D_OK;
}
```

This function loads our brick wall texture into two triangles. Two triangles make a rectangle…

```
//////////////////////////////////////////////////////
// CreateVertices()
//////////////////////////////////////////////////////
```

```
HRESULT CreateVertices()
{
      // uses two triangles two define a rectangle
    CUSTOMVERTEX triangleVertices[] =
    {
        {250.0f, 400.0f, 0.0f, 1.0f, 0.0f, 1.0f,},
        {250.0f, 200.0f, 0.0f, 1.0f, 0.0f, 0.0f,},
        {550.0f, 200.0f, 0.0f, 1.0f, 1.0f, 0.0f,},
        {250.0f, 400.0f, 0.0f, 1.0f, 0.0f, 1.0f,},
        {550.0f, 200.0f, 0.0f, 1.0f, 1.0f, 0.0f,},
        {550.0f, 400.0f, 0.0f, 1.0f, 1.0f, 1.0f,},
    };

      // create a vertex buffer....
      // notice number is 6* due to number of vertices...
    HRESULT hResult = g_pDirect3DDevice->CreateVertexBuffer(
        6*sizeof(CUSTOMVERTEX), 0, D3DFVF_XYZRHW | D3DFVF_TEX1,
        D3DPOOL_DEFAULT, &g_pVertexBuf, NULL);

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

    VOID* pVertices;
    g_hResult = g_pVertexBuf->Lock(0, 0, (void**)&pVertices, 0);
    if (FAILED(g_hResult))
    {
        strcpy(g_szErrorMsg, "Error locking vertex buffer.");
        PostQuitMessage(WM_QUIT);
        return g_hResult;
    }
    memcpy(pVertices, triangleVertices, sizeof(triangleVertices));
    g_pVertexBuf->Unlock();

    return D3D_OK;
}
```

This function renders the background bitmap

```
//////////////////////////////////////////////////
// PaintBackground()
//////////////////////////////////////////////////
void PaintBackground()
{
    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);
    }
}
```

This function renders the textures on the primitive and sets transparency for the black areas of the bitmap so that the background shows through

```
//////////////////////////////////////////////////
// PaintTexture()
//////////////////////////////////////////////////
void PaintTexture()
{
      // these are to do with the texture layers
    g_pDirect3DDevice->SetTextureStageState(0,
        D3DTSS_COLOROP, D3DTOP_SELECTARG1);
    g_pDirect3DDevice->SetTextureStageState(0,
        D3DTSS_COLORARG1, D3DTA_TEXTURE);
    g_pDirect3DDevice->SetTextureStageState(0,
        D3DTSS_ALPHAOP, D3DTOP_SELECTARG1);
    g_pDirect3DDevice->SetTextureStageState(0,
        D3DTSS_ALPHAARG1, D3DTA_TEXTURE);
      // these are to do with source and destination
      // blending operations in terms of colour blending
      // and setting the transparency
    g_pDirect3DDevice->SetRenderState(D3DRS_SRCBLEND,
        D3DBLEND_SRCALPHA);
    g_pDirect3DDevice->SetRenderState(D3DRS_DESTBLEND,
        D3DBLEND_INVSRCALPHA);
    g_pDirect3DDevice->SetRenderState(D3DRS_ALPHABLENDENABLE, TRUE);

      g_pDirect3DDevice->SetFVF(D3DFVF_XYZRHW|D3DFVF_TEX1);
      g_pDirect3DDevice->SetStreamSource(0, g_pVertexBuf, 0,
        sizeof(CUSTOMVERTEX));

    g_pDirect3DDevice->SetTexture(0, g_pTexture);

      // this actually draws the textured primitive
      // easy because of all the defining already done
    if(SUCCEEDED(g_pDirect3DDevice->BeginScene()))
    {
        g_pDirect3DDevice->DrawPrimitive(D3DPT_TRIANGLELIST,0,2);
        g_pDirect3DDevice->EndScene();
        g_pDirect3DDevice->Present( NULL, NULL, NULL, NULL );
    }
}
```

Exercises

1) Modify the program so that it uses only have the texture, both horizontally and vertically.
2) Modify the program so that it displays two rectangles. What happens with regards to the texture?