P3_Tutorial6 pt.2: Fonts and Text

Program Examples

See Collisions2D, D3D.h, D3D.cpp and ControlSprites.cpp (RenderSprites() method).
See also FontExample.cpp code online.

Displaying Text

When you display text in a game you may need text not just to display player information, but to display text as well. When you start programming game events a problem arises. You often need to display text that describes the outcome of the event. If a player opens a chest in an RPG you need to tell him what he has found, or that he needs a key to go through the locked door.

Displaying text in a Direct3D program isn't much more difficult than displaying text in any other widows program. However the process is a little different.

**Note: Although you can display text anywhere it is sometimes better to display the text in a text box. This is particularly true of game events.*

To display text you must carry out the following procedures, most of which mimic the standard model of DirectX:

- Create and ID3DXFont object
- *Optional: Calculate rectangle size to hold text*
- Call Direct3D device -> BeginScene()
- Draw the text
- Call Direct3D device -> EndScene()
- Release the ID3DXFont object

Create a Font Object

In order to display text on the screen you need to call the DrawText() method of the ID3DXFont object.
We need to do a few steps before we can do that though. The first is to declare a pointer to a ID3DXFont object.

```
LPD3DXFONT      g_pFont      = NULL;
```

After we have done this, we then have two methods of creating the font we want. Both of which rely somewhat on Win32.

The first method is using:

CreateFont

The CreateFont function creates a logical font with the specified characteristics. The logical font can subsequently be selected as the font for any device.

```
HFONT CreateFont(
  int nHeight,              // height of font
  int nWidth,               // average character width
  int nEscapement,          // angle of escapement
  int nOrientation,         // base-line orientation angle
  int fnWeight,             // font weight
  DWORD fdwItalic,          // italic attribute option
  DWORD fdwUnderline,       // underline attribute option
  DWORD fdwStrikeOut,       // strikeout attribute option
  DWORD fdwCharSet,         // character set identifier
  DWORD fdwOutputPrecision, // output precision
  DWORD fdwClipPrecision,   // clipping precision
  DWORD fdwQuality,         // output quality
  DWORD fdwPitchAndFamily,  // pitch and family
  LPCTSTR lpszFace          // typeface name
);
```

See: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/fontext_8fp0.asp

Example call:

```
HFONT hFont;
hFont = CreateFont( nHeight, 0, 0, 0, FW_BOLD, false, false, false,
      DEFAULT_CHARSET, OUT_DEFAULT_PRECIS,
CLIP_DEFAULT_PRECIS, DEFAULT_QUALITY,
      DEFAULT_PITCH | FF_DONTCARE, NULL );
```

This is the method used in the above examples. We saw this used during Programming 2.

After we have called this function we then need to create our ID3DXFont object. To do this we call D3DXCreateFont.

D3DXCreateFont Function

Creates a font object for a device and font.

```
HRESULT D3DXCreateFont(

    LPDIRECT3DDEVICE9 pDevice,
    HFONT hFont,
    LPD3DXFONT *ppFont
);
```

See: http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/d3dx/functions/miscellaneous/d3dxcreatefont.asp

Example call:

```
LPD3DXFONT pd3dxFontNew = NULL;
```

```
HRESULT hr;
hr = D3DXCreateFont( g_pd3dDevice, hFont, &pd3dxFontNew );
```

The second method:

LogFont
Is to initialise our own LogFont structure.
The LOGFONT structure defines the attributes of a font.

```
typedef struct tagLOGFONT {
  LONG lfHeight;
  LONG lfWidth;
  LONG lfEscapement;
  LONG lfOrientation;
  LONG lfWeight;
  BYTE lfItalic;
  BYTE lfUnderline;
  BYTE lfStrikeOut;
  BYTE lfCharSet;
  BYTE lfOutPrecision;
  BYTE lfClipPrecision;
  BYTE lfQuality;
  BYTE lfPitchAndFamily;
  TCHAR lfFaceName[LF_FACESIZE];
} LOGFONT, *PLOGFONT;
```

See: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdi/fontext_1wmq.asp

If you wanted to create an Arial Font with a size of 24 you would write this:

```
LOGFONT LogFont = {24, 0, 0, 0, FW_NORMAL, false, false, false,
DEFAULT_CHARSET, OUT_TT_PRECIS, CLIP_DEFAULT_PRECIS,
PROOF_QUALITY, DEFAULT_PITCH, “Arial”};
```

After we have called this function we then need to create our ID3DXFont object. To do this we call D3DXCreateFontIndirect.

D3DXCreateFontIndirect Function
Creates an font object indirectly for a device and font.

```
HRESULT D3DXCreateFontIndirect(

    LPDIRECT3DDEVICE9 pDevice,
    CONST LOGFONT *pLogFont,
    LPD3DXFONT *ppFont
);
```

See: http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c/directx/graphics/reference/d3dx/functions/miscellaneous/d3dxcreatefontindirect.asp

Example call:

```
LPD3DXFONT pd3dxFontNew = NULL;
HRESULT hr;
hr = D3DXCreateFontIndirect( g_pd3dDevice, &LogFont, &pd3dxFontNew );
```

Calculating the Text Rectangle

Next task to carry out is to determine the size of a rectangle that can contain the text. We have a few options at this point, we can:
Call the ID3DXFont object with the DT_CALCRECT flag. We then don't need our rectangle to be sized.
Or we can specify the size of our rectangle and use another flag, such as DT_CENTER. This centres the text in the middle of the Rectangle.
First we actually need to define a rectangle to hold the results of our calculations.

```
RECT r;
r.left = 0;
r.right = 0;
r.top = 0;
r.bottom = 0;
```

Actually Drawing the Text

Once we have our rectangle we call the DrwwText function mentioned aboce.

DrawText

```
INT DrawText(

    LPD3DXSPRITE pSprite,
    LPCTSTR pString,
    INT Count,
    LPRECT pRect,
    DWORD Format,
    D3DCOLOR Color
);
```

See: http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/directx9_c_Summer_04/directx/graphics/reference/d3dx/interfaces/id3dxfont/DrawText.asp
This definition also contains all the relevant Formats we can specify.

An actual call to DrawText looks something like:

```
pFont -> DrawText(pstrMsg, -1, &r, DT_CALCRECT, 0xff00000);
```

or

```
pFont->DrawText( "COLLISION", -1,  &textRect, DT_CENTER,
        D3DCOLOR_ARGB(255, 255, 0, 0) );
```

Dynamically Positioning the Text Rectangle

If we call DrawText with the DT_CALCRECT parameter then our RECT structure will contain the size of the rectangle needed to display the text. The left and top members will be as we set them, but the right and bottom sizes will have been adjusted to the appropriate sizes to contain the text. You can then get the width and height of the rectangle via the following calculation:

```
int iWidth = r.right – r.left;
int iHeight = r.bottom – r.top;
```

We can then use this information to help us dynamically position the text. If, for example, we want the text to appear in the centre of an 800x600 screen we could use the following calculation:

```
r.left = (800 - iWidth) / 2;
r.right = r.left + iWidth;
r.top = (600 - iHeight) / 2;
r.bottom = r.top + iHeight;
```

To use this method, we do not want to call the initial DrawText method, the one used to calculate the RECT structure, between BeginScene() and EndScene(). We only want to do this when we have worked out the best position for our text.

Releasing the Font Object

Don’t forget to release the font object, in the usual way, when you have finished.

```
pFont -> Release();
```

Updating for the DirectX 9.0c update

You now no long need to create a font object then pass that, you can just pass the parameters straight into the `D3DXCreateFont` function.

See Scaling and Rotation/Font code for examples.