

Welcome To the Windows Graphics Trainer By *DenthoroF Asphyxia*

Part 2

Introduction

Hello everyone :) I am recently returned from a university ceremony up in Pretoria (way behind the Borewors Curtain) ... most of it was in Afrikaans, so I got to know the English pamphlet pretty well :)

Anyway, I'm back, and I bring with me part two of the Windows Graphics Trainer. Response was mixed for the first one... one mail bomb, a few flames taking about my "betrayal" for coding in Windows... and many from interested parties telling me to keep going. What, me shy away from controversy? It worked for Madonna, didn't it... ;)

Hornet ate my upload, so GIN Internet services and EzE found room for me on one of their sites.

This one is on bitmaps... you are going to hate it, because it leaves you with more questions then when you started, but you have to start small.

The sample code has been written and compiled in MSVC++ v1.52c 16-bit, and will run fine under Win 3.1 ... the code should run without much modification under any Windows C compiler.

What is a "Device Context"?

Okay, so I mentioned Device Context's (DC's) last time, but it's important.

A DC is how Windows sees a surface it can draw on. A DC can be a printer, your screen, or a piece of memory pretending to be anything.

Most of the time, you can only draw to the screen DC from the same DC or one pretending to be like the screen DC in memory. This means that for our purposes, we want to create a DC in memory like that of the screen. We do this as follows :

```
HDC    tempDC;
tempDC = CreateCompatibleDC (NULL);
//.....
DeleteDC (tempDC);
```

Using NULL as a parameter gives us a DC like that of the systems screen. A new DC defaults to having a monochrome bitmap as its background. *If you select in another bitmap to your DC, always remember to select in the original bitmap when done, otherwise you will lose system resources.*

You must be very careful when using DC's ... in Win3.1, you can have a maximum of five DC's active at once. If you forget to delete your DC directly after you have used it, Windows will run out of resources with which to update

the screen and the system will crash.

Okay, enough about DC's for now, we'll get back to them in a bit.

Bitmaps and Resource Files

The easiest way to get a bitmap into your application is to store it in your resource file. In VC++ you have easy access to this via the Tools/App Studio option, and as I recall from my Borland C++ days it is the same there too. If you use a compiler without a DevStudio, don't worry, you can write the equivalent as a text file :)

The problem with this is blatantly obvious. Most of these horrible things only store the bitmap as a 16 color image! Eeeew.

Anyway, for now we will live with this, because palette manipulation is something I'll be covering in another tut (probably the next one)

If you have a bitmap resource, all you need to load it up is

```
HBITMAP hbmPicture = LoadBitmap(hInst, MAKEINTRESOURCE(nIDResource));
```

where hInst is the HINSTANCE of your application (given in your WinMain function) and nIDResource is the identifier for your bitmap (something like IDB_MYPICTURE ... you specify when you add it to the resource list)

One of the plusses of this is that the bitmap is included in your .exe file, so you just distribute one thing.

As you may have guessed, anything starting with an H is like a pointer.. HBITMAP is a Handle to a bitmap.

Okay, so now you have your very own DC and your very own bitmap. How do you draw it to screen?

Drawing bitmaps to Screen

As I said earlier, the easiest way to get information to the screen DC of your application is to copy it from a memory DC.

Let's assume we get a WM_PAINT message and want to draw a bitmap to screen.

On creation we have said :

```
HBITMAP hbmPicture = LoadBitmap(hInst, MAKEINTRESOURCE(nIDResource));
```

then we get the message :

```
...
case WM_PAINT :
{
    HDC pDC = GetDC(hWnd); // Get the screen DC
    HDC tempDC;
    tempDC = CreateCompatibleDC (NULL);
```

```

        //      Create the memory DC

HBITMAP bOld = (HBITMAP)SelectObject (tempDC, hbmPicture);
        //      Select in our bitmap
BitBlt (pDC, 0, 0, 100, 100, tempDC, 0, 0, SRCCOPY);
SelectObject (tempDC, bOld);
        //      Select back in old bitmap so we don't lose GDI //
        resources

DeleteDC (tempDC);           //      Delete the memory DC
ReleaseDC (hWnd, pDC);       //      Release the screen DC
    }
    break;
...

```

If you do the above, you will have your own little 16 color bitmap in the top left hand of your applications' window.

Note that while you have your bitmap in the memory HDC, GetPixel and SetPixel will work.. *and the changes will stay on the bitmap even after it has been selected out of the DC.*

The sample application does a bit more then this, but it is all based upon the same easy concept. Let me know if any of you have problems with the resource file with your particular compiler, I would like to know the differences.

In Closing

Okay, so you have mastered the above (easy) concepts, picked through the source file and understand it, and are about ready to strangle me.

Sure, I hear you say, tell us about these things and then leave us in the lurch. What about 256 colors, transparency, having a solid background and all those other neat things?

Sheesh, you people. Wasteland was brilliant and only used 16 colors. You are all just too spoilt with your fancy monitors and... okay, okay, just kidding :) Getting everything done in one trainer would be just a bit too much to chew off in one sitting. Be patient, I'll get to all of that stuff and more.

Visit my web page! :-) it's at <http://www.eze.co.za/denthor> ... it's more fun then MODULA2.

See you next time!

Byeeee.....
Denthor

03/04/97