

Different Techniques for Bitmap Fading Using the MB86276

▶ Introduction

Overview of GDCs and the Fujitsu MB86276

Fujitsu's Graphic Display Controller (GDC) product line is currently in its third generation. It consists of the high-end MB86297, the middle end MB86296, and the low-end MB86276. The first two have a PCI interface and the last one has an SRAM-type interface for the host controller. The MB86276's low cost, small form factor, and relatively low power consumption are its most attractive features. This GDC is very well-suited for low-end applications in automotive navigation systems, dashboard clusters, handheld navigation systems and the like.

Overview of the MB86276's Feature Set

The MB86276 is a 2D GDC without a geometry processing unit but with a Z-buffer for 3D drawing. As mentioned before, this graphics display controller has an SRAM type of

interface for the host processor. The memory interface for the SDRAMs and FCRAMs serves to buffer drawing and display frames, display lists, bitmaps, texture maps, etc. The MB86276 also supports video capturing in RGB and ITU-656 formats and dual-video output in digital RGB format. The feature-rich rendering engine supports different drawing primitives, bit blt, texture mapping, alpha blending, anti-aliasing, shading, and the like.

This application note will focus on the MB86276's ability to execute the special effect of bitmap fading. This can be very useful in presenting good-looking animation. The functionality can be implemented using multiple techniques by using bitmap drawing, blt (block transfer) fill, alpha blending, CLUTs (Color Lookup Tables) or palettes, and alpha plane.

▶ Bitmap Fading

Description

Bitmap fading is a gradual appearance or disappearance (degree of transparency) of a bitmap on the display screen. One example of its use is in posting welcome messages when an application begins. Implementation can get complicated if multiple bitmaps need to be faded on the same graphics layer. (A layer is a plane, an area in graphics memory, which contains different drawing objects. Fujitsu GDCs allow the use of multiple layers by overlaying them.). Fortunately, the MB86276 has several functions that make the implementation very easy. These functions are hardware based and offload the host processor significantly. It should be noted that bitmap fading can be implemented in a similar fashion with the rest of Fujitsu GDCs.

Techniques for Implementing Bitmap Fading

The versatility of the MB86276 allows implementing this special effect using three different techniques.

1. Layer Alpha Blending
2. Palette Animation in Indirect Color Mode
3. Alpha Plane

Different Techniques for Bitmap Fading Using the MB86276

► Techniques for Implementing Bitmap Fading

Bitmap Fading Using Layer Alpha Blending

This technique is carried out by first blting the bitmap on the respective graphics layer. At least two layers should be overlaid in the blending mode, using a constant alpha-blend ratio. By gradually varying this ratio between the layers, the bitmap can be faded in the desired fashion.

Sample code (based on the Fujitsu GDC API V02L03) for this technique is shown below.

While this method is very simple, it has an obvious drawback – the inability to fade two or more bitmaps on the same layer and in different fading modes (amount and direction of fading). Also, at least two layers are required to implement this technique.

```
DrawBitmap(L1ADR); //bitmap drawn on Layer 1

XGdcFlushEx(&drvctx, 0); //Display List flushed

printf("Press Enter to start fade...\n");
getchar();

blend = 0;
for(i=0;i<256;i++)
{
    GdcLayerOverlayBlend(GDC_DISP_LAYER_L0, GDC_BLEND_CURRENT_RATIO,
                        GDC_BLEND_NO_CORRECT, GDC_BLEND_RATIO_CONSTANT,
                        blend);

    ++blend;
    GdcWait(11); }
}
```

Bitmap Fading Using Palette Animation

Indirect color mode (8 bits per pixel) uses the CLUT or palette to reduce the graphics memory usage. In effect, this also limits the amount of data to be transferred from the host processor to the graphics memory and improves overall efficiency. Palette animation is based on using the CLUT specified in the indirect color mode. Instead of modifying the bitmap data or changing the alpha-blending ratio, this technique just changes the palette entries that, in turn, change the way the bitmap appears.

The MB86276 supports four different palettes, one per layer for four layers, each having 256 entries. Therefore, more than one bitmap on a single layer can be supported by dividing a

palette into two or more parts. For example, the 256-entry palette could be divided into 8 sub-palettes, each consisting of 32 entries. This means that on one 8-bpp layer, eight bitmaps can be simultaneously executed for the fading operation. Each bitmap can have a different transparency level and fading direction. The amount of data required to be transferred between the host processor and graphics memory is much less than if each bitmap were individually modified. Palette animation can be implemented using only one layer. All these factors make this technique very flexible and effective for fading implementations.

Sample code (based on Fujitsu GDC API V02L03) for this technique is shown on the next page.

```
XGdcBltdraw8(&drvctx, (WIN_WIDTH / 2) - (bmWidth/2),
             (WIN_HEIGHT/2) - (abs(bmHeight)/2),
             bmWidth, abs(bmHeight), BitmapBits);
XGdcFlushEx(&drvctx, 0);

printf("Press <Enter> to fade out...\n");
getchar();
//-----
// FADE OUT
// Now animate the palette to fade out
// This is done by writing to MB86276's palette registers directly
// Implemented by reducing the R, G, and B values of each palette entry
//-----
for(fade=63; fade>0; fade--)
{
    for(i=0; i<25; i++)
    {
        fadeRed = (LOPalette[i] & 0x00FC0000) >> 18;
        fadeRed = (fadeRed * fade) / 63;
        WrkPalette[i] = fadeRed << 18;

        fadeGreen = (LOPalette[i] & 0x0000FC00) >> 10;
        fadeGreen = (fadeGreen * fade) / 63;
        WrkPalette[i] |= fadeGreen << 10;

        fadeBlue = (LOPalette[i] & 0x000000FC) >> 2;
        fadeBlue = (fadeBlue * fade) / 63;
        WrkPalette[i] |= fadeBlue << 2;
    }
    GdcColorPalette(GDC_L0_LAYER_PALETTE, 0, 25, WrkPalette);
    GdcWait(10); //10 millisecond delay so you can see the fade...
}

printf("Press <Enter> to fade in...\n");
getchar();
//-----
// FADE IN
// Now animate the palette to fade in
// This is done by writing to MB86276's palette registers directly
// so this is a hardware assisted fade...very fast!
//-----
for(fade=1; fade<64; fade++)
{
    for(i=0; i<25; i++)
    {
        fadeRed = (LOPalette[i] & 0x00FC0000) >> 18;
        fadeRed = (fadeRed * fade) / 63;
        WrkPalette[i] = fadeRed << 18;

        fadeGreen = (LOPalette[i] & 0x0000FC00) >> 10;
        fadeGreen = (fadeGreen * fade) / 63;
        WrkPalette[i] |= fadeGreen << 10;

        fadeBlue = (LOPalette[i] & 0x000000FC) >> 2;
        fadeBlue = (fadeBlue * fade) / 63;
        WrkPalette[i] |= fadeBlue << 2;
    }
    GdcColorPalette(GDC_L0_LAYER_PALETTE, 0, 25, WrkPalette);
    GdcWait(10); //10 millisecond delay so you can see the fade...
}
}
```

Different Techniques for Bitmap Fading Using the MB86276

Bitmap Fading Using Alpha Plane

This technique uses layer 5 as alpha plane, which is in 8-bits-per-pixel mode. Each pixel of this layer acts as an 8-bit alpha-blending ratio for the corresponding pixel in the layer subjected to the alpha plane. By modifying the layer 5 in the correct location (which overlays with the bitmap in the other layer), the transparency can be varied and fading can be achieved.

This technique has a few advantages over palette animation. First, there is no limit to the number of bitmaps that can be faded. Second, the technique can also be used with the bitmap layer in direct color mode (16 bits per pixel). Just like palette animation, the alpha-plane method also allows having different fading degrees and direction for each bitmap.

However, the method has a disadvantage in that it requires at least three layers for implementation, whereas the palette animation requires only one layer. Bitmap fading using alpha plane also requires using a VSYNC interrupt so that each frame gets updated without the viewer's notice and the animation appears smooth. So additional manipulation is required due to the display list being flushed again and again. This was not required in the first two techniques.

Sample code (using Fujitsu GDC API V02L03) for this technique is shown below. This was written for 16 bpp bitmaps. With a little modification, it can also be used with 8 bpp bitmaps.

```
//L5: Setting up the Alpha Plane Layer. Note that at least two more layers are required
// in addition to this layer for using the alpha plane.
  XGdcDrawDimension(&drvctx, GDC_8BPP_FORMAT, L5ADR0, WIN_WIDTH, WIN_HEIGHT);
  XGdcColor(&drvctx, 0xff);
  XGdcBltFill(&drvctx, 0, 0, WIN_WIDTH, WIN_HEIGHT);
  XGdcColor(&drvctx, 0x0);
  XGdcBltFill(&drvctx, 100, 100, 128, 96);
  XGdcColor(&drvctx, 0xff);
  XGdcBltFill(&drvctx, 300, 100, 128, 96);
  XGdcFlush(&drvctx);

//L0 - contains bitmaps
  XGdcDrawDimension(&drvctx, GDC_16BPP_FORMAT, L0ADR, WIN_WIDTH, WIN_HEIGHT); //BL

  XGdcColor(&drvctx, GDC_WHITE);
  XGdcBltFill(&drvctx, 0, 0, WIN_WIDTH, WIN_HEIGHT);
  XGdcBltDraw16(&drvctx, 100, 100, 128, 96, alpha_map_fade1);
  XGdcBltDraw16(&drvctx, 300, 100, 128, 96, alpha_map_fade2);
  XGdcFlush(&drvctx);

//L1 - background
  XGdcDrawDimension(&drvctx, GDC_16BPP_FORMAT, L1ADR, WIN_WIDTH, WIN_HEIGHT); //BL
  XGdcColor(&drvctx, GDC_BLUE);
  XGdcBltFill(&drvctx, 0, 0, WIN_WIDTH, WIN_HEIGHT);
  XGdcFlush(&drvctx);

printf("Press enter to start fading using Alpha Map\n");
getch();

  XGdcDrawDimension(&drvctx, GDC_8BPP_FORMAT, L5ADR0, WIN_WIDTH, WIN_HEIGHT);
  for(l = 0; l<256; l=l++)
  {
    XGdcVerticalSync(&drvctx); //Wait for VSync Interrupt
    XGdcColor(&drvctx, l);
    XGdcBltFill(&drvctx, 100, 100, 128, 96);
    XGdcColor(&drvctx, 255-l);
    XGdcBltFill(&drvctx, 300, 100, 128, 96);
    GdcWait(30);
  }
```

(continued on next page)

```
/*This section, including the XGdcVerticalSync call before,  
  ensures that the display frame is updated during VBI and therefore  
  no glitch occurs on the display*/  
XGdcInterrupt(&drvctx); //Request VSync interrupt  
num = XGdcQueryCurrentDLBuf(&drvctx);  
GdcSync(&drvctx, num);  
  
/*Clears the current DL in the DL buffer to ensure  
  that it doesn't get executed again*/  
  
XGdcCancelDisplayList(&drvctx);  
}
```

▶ Conclusion

The above discussion highlights the versatility of the MB86276. Bitmap fading may be a very small part of the entire application, yet this GDC has multiple ways to

implement it. The entire implementation adds little, if any, burden to the host processor. The MB86276 can be really intelligent in unexpected ways.

FUJITSU MICROELECTRONICS AMERICA, INC.

Corporate Headquarters
1250 E. Arques Avenue, M/S 333, Sunnyvale, CA 94085-5401
Tel: (800) 866-8608 Fax: (408) 737-5999
E-mail: inquiry@fma.fujitsu.com Web Site: <http://us.fujitsu.com/micro>

© 2006 Fujitsu Microelectronics America, Inc.
All company and product names are trademarks or
registered trademarks of their respective owners.
Printed in the U.S.A. GDC-AN-21233-12/2006