33 Reading BMP Files

“Love is not rude, is not selfish, and does not get upset with others.”

When you look at the BMP file format closely, you can find that BMP stores palette information in it. So in order to display BMP files, we must load that palette information. When we read a BMP file in mode 13h we have two restrictions: maximum color of BMP must be 256 (BMP files can be of 16, 256 or 2^{24} colors!) and file size must be less than 64KB. The following program by **Alexander Russell** reads 256 colors BMP file. It clips images larger than 320x200. It reads the whole thing into memory, and then displays it directly to video memory.

33.1 Programs

```
#include <stdio.h>
#include <io.h>
#include <conio.h>
#include <malloc.h>
#include <string.h>
#include <dos.h>

#pragma -mm /* force to compile in medium memory model */

#pragma inline

#define _64k 65300u

#define BM_TYPE 19778u

#define BI_RGB      0L
#define BI_RLE8     1L
#define BI_RLE4     2L

typedef unsigned int WORD;
typedef unsigned long DWORD;
typedef unsigned char BYTE;

typedef struct tagBITMAPFILEHEADER {
        WORD    bfType;
        DWORD   bfSize;
        WORD    bfReserved1;
        WORD    bfReserved2;
        DWORD   bfOffBits;
} BITMAPFILEHEADER;

typedef struct tagBITMAPINFOHEADER{
   DWORD  biSize;
   DWORD  biWidth;
   DWORD  biHeight;
   WORD   biPlanes;
   WORD   biBitCount;
   DWORD  biCompression;
   DWORD  biSizeImage;
   DWORD  biXPelsPerMeter;
   DWORD  biYPelsPerMeter;
   DWORD  biClrUsed;
   DWORD  biClrImportant;
} BITMAPINFOHEADER;

typedef struct tagRGBQUAD {
   BYTE    rgbBlue;
   BYTE    rgbGreen;
   BYTE    rgbRed;
   BYTE    rgbReserved;
} RGBQUAD;

typedef struct tagBITMAPINFO {
   BITMAPINFOHEADER    bmiHeader;
   RGBQUAD             bmiColors[1];
} BITMAPINFO;

static BYTE old_mode;

#define INPUT_STATUS_1  03dah   /* Input Status 1 register */

/* ---------------------------------------------------
       SaveVideoMode - save the vid mode so
            we can restore it on exit     */

void SaveVideoMode( void )
{
   /* save current mode */
   asm {
      mov   ah, 0fh
      int   10h
      mov   old_mode, al
      }
} /*--SaveVideoMode( )------*/
```

```
/* ----------------------------------------------
      SetGraph - set graphics mode to
           mode BIOS 0x13, 320x200 256 color    */

short SetGraph( void )
{

   asm {
      /* set new mode */
      xor   ah, ah
      mov   al, 013h
      int   10h
      }

   return(0);
} /*--SetGraph( )--------*/

/* ----------------------------------------------
       RestoreVideoMode - restore old video
                        mode        */

void RestoreVideoMode( void )
{
   asm {
      xor   ah, ah
      mov   al, old_mode
      int   10h
      }
} /*--RestoreVideoMode( )-------*/

/*-------------------------------------------------------
      SetUpVGAPalette -  set all 256 colours of the
        palette, wait for vert sync to avoid flashing */

void SetUpVGAPalette( char *p )
{

   /* wait for vert sync */
   asm   {
         mov     dx,INPUT_STATUS_1
         }
WaitVS:
   asm   {
         in      al,dx
         test    al,08h
         jz      WaitVS  /* vertical sync is active high (1 = active) */
         }
   asm   {
         .386

/*       this sets the default palette register mask, don't need to do
         this unless it gets changed

         mov   dx, 03c6h
         mov   al, 0ffh
         out   dx, al
*/
         /* set palette, using auto-increment feature */
         xor   al, al
         mov   dx, 03c8h
         out   dx, al
         mov   cx, 768
         mov   si, p
         mov   dx, 03c9h
         rep   outsb
         }
} /*--SetUpVGAPalette( )----------*/

/*-----------------------------------------------
      FarFread - returns number of bytes read
   I compiled this in medium model, so fread
   expects a near pointer.
   This let’s me read the file into far memory. */

int FarFread( BYTE far *b, WORD size, FILE *fp )
{
   BYTE *t;
   unsigned int i;
   WORD read;

   t=malloc(1024);  // temp buffer
   if ( t )
      {
      read=0;
      i=0;
      // read into a near buffer, and then copy to the far buffer
      while ( size >= 1024 )
         {
         i=fread(t, 1, 1024, fp);
         read+=i;
         _fmemcpy(b, t, i);
         b+=i;
         size-=i;
```

```
            if ( i != 1024 )
               break;
            }

         i=fread(t, 1, size, fp);
         read+=i;
         _fmemcpy(b, t, i);

         free(t);
         }
      else
         read=0;

      return(read);
   } /*--FarFread( )-------*/

   /*-----------------------------------------------------------
         DecompressOneLineBMP
     decompress one line of a 256 colour bmp into line
     returns where we ended up in rp which is the raw image
    width is max line width, i_size is how much data we read in */

   BYTE far *DecompressOneLineBMP( BYTE far *rp,
                                    BYTE far *line,
                                    long *i_size, short width )
   {
      long size=0;
      BYTE num;
      short w=0;
      int odd;

      width+=3;  // just to make sure we don't over run line
                 // which would crash us, only a bad bmp would cause this
      while ( w < width )
         {
         if ( *rp )  /* first byte isn't zero,
                        so it is a run of identical pixels */
            {
            // RLE run
            num=*rp;
            rp++;
            size++;
            w+=num;
            while ( num )
               {
               *line++=*rp;
      num--;
      }
   rp++;
   size++;
   }
else
   {
   // zero, either escape sequence, or string of random pixels
   rp++;
   size++;
   switch ( *rp )
      {
      case 0:  // end of line, we are done
         rp++;
         size++;
         *i_size-=size;
         return rp;
         //break;

      case 1:  // end of bitmap
         rp++;
         *i_size=0;
         return rp;
         //break;

      case 2: // delta! - we do not handle this
              // this makes the x,y jump to a new place
         rp++;
         size++;
         break;

      default:  // string, 3 thru 0xff
                // a string of random pixels
         num=*rp;
         rp++;
         size++;
         size+=num;
         w+=num;
         odd=num & 1;  // pads odd runs
         while ( num )
            {
            *line++=*rp++;
            num--;
            }
         if ( odd ) // odd strings are padded to make them even
            {       // this skips the padding byte
            rp++;
```

```
                    size++;
                    }
                 break;
              }
           }
        }
     // should never get here actually, as each line ends with a EOL
     *i_size-=size;

     return(rp);
  } /*--DecompressOneLineBMP( )-----------*/

  /*----------------------------------------
        main - main of BMP            */

  int main( int argc, char *argv[] )
  {
     BITMAPFILEHEADER far *header;
     BITMAPINFOHEADER far *info;
     RGBQUAD far *rgb;
     FILE *fp;
     long size;
     long i_size, l1;
     short num_col;
     unsigned int m, w_copy;
     BYTE far *buff, far *rp, far *line;
     int i, adj;
     BYTE pal[768], *t1;
     BYTE far *video;

     if ( argc < 2 )
        printf( "Usge: BMP <bmpfile> \n\a" );
     else
        {
        fp=fopen(argv[1], "rb");
        if ( fp )
           {
           size=filelength(fileno(fp));
           if ( size > _64k )
              {
            printf( "DARN it! DOS SUCKS! file size greater"
                    "than %u bytes! - TRUNCATING!\n", _64k);
              size=_64k;
              }
           buff=farmalloc(size);
           if ( buff )
              {
```

```
m=FarFread(buff, size, fp); // read as much as we can into mem
  if ( m != size )
   printf("Error reading: %s\n", argv[1]);
else
   {
   // make header, and info point to the correct place
   header=buff;
   info=buff + sizeof(BITMAPFILEHEADER);

   /* this is demo code, so let's display all
              the header information. */
   printf("type   %u\n", header->bfType);
   printf("size   %lu\n", header->bfSize);
   printf("Offset %lu\n", header->bfOffBits);
   printf("Filesize %lu (%u indicates truncated)\n\n",
                                                size, _64k);

   printf("biSize          =%lu (%d)\n", info->biSize,
                                sizeof(BITMAPINFOHEADER));
   printf("biWidth         =%lu\n", info->biWidth);
   printf("biHeight        =%lu\n", info->biHeight);
   printf("biPlanes        =%u\n", info->biPlanes);
   printf("biBitCount      =%u\n", info->biBitCount);
   printf("biCompression   =%lu\n", info->biCompression);
   printf("biSizeImage     =%lu\n", info->biSizeImage);
   printf("biXPelsPerMeter =%lu\n", info->biXPelsPerMeter);
   printf("biYPelsPerMeter =%lu\n", info->biYPelsPerMeter);
   printf("biClrUsed       =%lu\n", info->biClrUsed);
   printf("biClrImportant  =%lu\n", info->biClrImportant);
   if ( header->bfType != BM_TYPE )
              printf("%s is not a bmp!\n", argv[1]);
   else
    {
    // lets display it!
    // We only handle 256 colour types with this code!
    if ( info->biPlanes == 1 && info->biBitCount == 8 )
           {
       // get and set palette info
       // colour table
       rgb=(RGBQUAD far *)((BYTE far *)info + info->biSize);
           num_col=info->biClrUsed ? info->biClrUsed : 256;
           printf("num_col = %d\n", num_col);

           // have to shift because vga uses 6 bits only
           t1=pal;
```

```
for ( i=0; i <  num_col; i++ )
 {
       *t1++=(rgb[i].rgbRed)>>2;
       *t1++=(rgb[i].rgbGreen)>>2;
       *t1++=(rgb[i].rgbBlue)>>2;
 }

printf("Press a key to view image,"
     " then again to exit\n");
    getch();

SaveVideoMode();
SetGraph();
SetUpVGAPalette(pal);

/* get, de-compress, and display
 note, bmp stores the image 'upside down' */

// point to bottom of screen
video=MK_FP( 0xa000, 320u*199u );

rp=buff + header->bfOffBits; // Raw Pointer to image

// NOTE! if bisizeImage is zero, l1 must be used
i_size=info->biSizeImage;

// this is because we truncate large images
l1=size - (sizeof(BITMAPFILEHEADER) +
             sizeof(BITMAPINFOHEADER) + num_col*4);
    if ( i_size > l1 || i_size == 0 )
       i_size=l1;

    // clip width
    if ( info->biWidth <= 320 )
       w_copy=info->biWidth;
    else
       w_copy=320;

    if ( info->biCompression == BI_RLE8 )
       {
       // we will decompress one line at a time,
       // then clip and display it

       line=farmalloc(info->biWidth+4);
```

```
         if ( line )
               {
               for ( i=0; i < info->biHeight && i < 200
                                    && i_size > 0; i++ )
                  {
            rp=DecompressOneLineBMP(rp, line, &i_size,
                                          info->biWidth);
                  _fmemcpy(video, line, w_copy);
                  video-=320;
                  }

               farfree(line);
               }
            }

         else
            {
      // not compressed, simply copy to video mem
      //pads to multiple of 4 bytes
      adj=info->biWidth % 4;
            if ( adj )
               adj=4 - adj;
            if ( info->biCompression == BI_RGB )
               {
               for ( i=0; i < info->biHeight && i < 200
                                    && i_size > 319; i++ )
                  {
                  _fmemcpy(video, rp, w_copy);
                  video-=320;
                  rp+=info->biWidth;
                  rp+=adj;
                  i_size-=info->biWidth;
                  i_size-=adj;
                  }
               }
            }

         getch();
     RestoreVideoMode();
         }
      else
     printf("This code only does 256 colour BMP's\n");
  }
   }
farfree(buff);
}
```

```
            else
               printf("OUT of mem!\n");

            fclose(fp);
            }
         else
          printf("ERROR opening file: %s\n", argv[1]);
         }
      return(0);
   } /*--main( )--------*/
```