真彩 BMP 图象的 TVGA 显示

郑玉锋　（天津大学　300072）

摘要：本文介绍一种在 TVGA 视频卡上显示真彩色 BMP 图象的实用方法，采用直接写屏技术，在 0.3 秒内即可显示一幅 640×480×16M 彩色的图象。

在普通的计算机上显示真彩色图象是图象处理爱好者的梦想。如果你备有 TVGA 9400 或是 9440 视频卡，笔者可以帮助你实现这一夙愿。所谓真彩色图象，是指每一象素用 24 位的颜色模式来描述，即 R、G、B 分量分别用 8 位来表示。这样，在计算机的屏幕上你可以同时看到 16M 种颜色丰富的彩色图象，真可谓赏心悦目！

TVGA 9400 卡的 oxbc 模式对应于真彩显示工作方式，此时相应的屏幕分辨率是 640×480，用户无须自行设置调色板，只需将图象数据（象素值）顺序写入 VRAM 区即可。注意：颜色分量的写入顺序必须 B、G、R！

和其他许多图象格式类似，BMP 格式图象有一个文件头，用以描述图象的属性。详细的 BMP 图象格式说明可参阅文献[1]，这里只简单说明一下其中几个重要的参数（请参阅源程序 TCBMP.C）：

1.象素彩色位数（bits）—可以是 1、4、8 或 24，对应的颜色分辨率分别是 2、16、256、16M。

2.图象宽高（Width，depth）—如 Width = 640，depth = 480。

3.图象大小（bisizeImage）—bisizeImage = width × depth × bits/8。

4.文件头大小（headersize）—当 bits < 24 且 headsize > 54 时，说明文件头中包含一个彩色表，用户必须根据彩色表的内容设置调色板。彩色表中每 4 个字节（分别是 B、G、R、O）顺序对应调色板中的一个索引值。注意：真彩图象中无彩色表定义。

5.文件大小（filesize）—filesize = headersize + bisizeImage。

值得注意的是 BMP 格式图象文件一般是按非压缩格式存放图象数据，而且文件中图象数据的第一行对应于屏幕图象的最后一行。对于真彩色图象（bits = 24），每个象素用三个字节表示，依次是 B、G、R 值；每行如不在 4 字节边界上则填 0。

文末（附在下页）给出了用 MSC6.0 编写的完整的源程序（TCBMP.C），并在通用 486 微机（配有 TVGA 9400 视频卡）上调试通过。考虑程序的通用性，该程序除了能显示真彩 BMP 图象外，还可用于显示 2、16、256 色彩色/灰度 BMP 图象。程序采用直接写屏方法显示图象，速度较快，只需 0.3 秒即可显示一幅 640×480×16M 色的真彩 BMP 图象。

编译参数：/STACK:4096。

运行环境：在 DOS 提示符下键入 TCBMP ZYF 回车即可。在当前目录下应有执行文件 TCBMP.EXE 和图象文件 ZYF.bmp。

讨论：

(1)如要显示其他格式图象文件（如 .pcx，.tga，.tif，.gif 等），读者可以借助现成的图象格式转换软件（如 Hijaak，Gmaphic Worushop 等）将其转换为已知的 BMP 格式后再显示之。

(2)如要显示大尺寸图象（如 512×512），可考虑将原图象缩小显示，或采用图象屏幕漫游的方法。

(3)如要显示自定义格式图象（如用大恒生产的 VC32 图象采集卡所获得的真彩图象是以 R、G、B 分量分别存储的三个数据文件，如 ZYF.R，ZYF.G，ZYF.B），此时读者须自己编制图象格式转换程序（即 .r + .g + .b⇒.bmp）。

参考文献

[1] 潘志庚编，《Windows 环境下图形图象程序设计》，北京清华大学出版社，1995。

（接第 42 页）重要的功能，它帮我们检查语法和任何用 CSP/AD 定义的应用程序的逻辑设计以及模拟执行应用程序，并可观察每个语句的处理过程和结果。对于 SQL 应用程序而言，它还要检查用户是否取得对所用的 SQL 数据库的授权以及所授权限制的级别。这里有一个问题需要特别指出：SQL 应用程序测试功能不能模拟分段执行方式。分段方式会影响 SQL 处理，因为在第一段的结尾，所有对数据库的更改都会自动落实到数据库中，这种落实也会释放所有封锁和失去 SCAN（扫描）位置。为了解决这个问题，我们可以通过在应用程序顶部（即主进程语句定义的预处理语句部分）把 EZECNVCM 字设置为 1（这个字是用来在非 DPPX/SP 系统上控制每个 CONVERSE 对话的数据自动落实），强制模拟分段方式的作用，以要求测试功能在第一个 CONVERSE 进程期间落实 SQL 资源。

5.生成应用程序

这是建立 SQL 应用程序的最后一个步骤。当 SQL 应用程序经反复修改、测试，确认无误后，便可生成我们需要的应用程序执行代码文件，并通过填写应用装载文件名称，将所生成的应用程序装载到指定的 ALF 中去，供用户使用。在生成输出的模块中，除了一般用户程序所具有的用户装载模块（若未建立 CSP 用户表则无此模块）、映象装载模块、应用程序映象模块、数据装载模块、处理装载模块和数据特性半载模块之外，还包括 SQL 语句准备装载模块和 SQL 语句执行装载模块。这两大模块分别给出了识表（SSIT）和缩短了的准备 SQL 进程表。

如果用户是 PC 机用户，那么切勿在主机上进行生成，而要将 MSL 中的源程序取出（EXPORT），经传递接收（RECEIVE）至 PC 硬盘上，然后在 PC 机上启动 EZ－PREP 软件生成一个 EZ－RUN 应用程序，方可在 PC 机上运行。关于 EZPEP 和 EZRUN 的工作原理和方法，在此就不再赘叙。

```
/*
 *      ***TCBMP.C***           ###/96-5-24/###
 *      DISPLAY THE COLOR BITMAP IMAGE BY WRITING VIDEO MEMORY
 *      !!! DISPLAY IMAGE ON SCREEN WITH RATIO 1:1 (Width/Height)
 *      // Set TVGA Mode: [0x005d] = 640x480x256 Color levels
 *      // Set TVGA Mode: [0x006c] = 640x480x16M Color levels (True Color)
 *      AUTHOR === ZHENG YU FENG
 */

#include "graph.h"
#include "stdio.h"
#include "dos.h"
#include "stdlib.h"
#include "string.h"
#include <conio.h>
#include "filetype.h"

#pragma intrinsic( outp, inp )

void main (int argc, char *argv[])
{
 register int i,j;
 union REGS inregs,outregs;
 char flnm[60],arg[5];
 FILE *fp;
 int row=512,col=512;
 unsigned char _far *vp=(char _far *)0xa0000000;
 unsigned char fb[640*3],bitpix,index=0;
 long add;
 int m_b=-1,seg,flag_mode=0;
 BMPHEAD bmph;
 int disp_width;

 printf("\n  ********** TRUE COLOR BITMAP DISPLAYING PROGRAM **********  \n");
 if(argc>1)
   strcpy(flnm,argv[1]);
 else
 {
 printf("\n  Input the Bitmap file name [.bmp]: ");
 gets(flnm);
 }
 REDISP:
 if(!strchr(flnm,'.'))
   strcat(flnm,".bmp");
 if((fp=fopen(flnm,"rb"))==NULL)
   {
    printf("\n Open file failure!! \a\a\n");
    printf("\n Please Check following files whether exist:");
    printf("\n %s",flnm);
    printf("\n\n Note: The file extension name is appended automatically,");
    printf("\n       such as [.bmp]!");
    exit(1);
   }

 index=0;
 /* Read the Bitmap file Header Information */
 fread(bmph  . id              ,sizeof(char),2,fp);
 fread(&bmph . filesize        ,sizeof(long),1,fp);
 fread(bmph  . reserved        ,sizeof(int ),2,fp);
 fread(&bmph . headersize      ,sizeof(long),1,fp);
 fread(&bmph . infoSize        ,sizeof(long),1,fp);
 fread(&bmph . width           ,sizeof(long),1,fp);
 fread(&bmph . depth           ,sizeof(long),1,fp);
 fread(&bmph . biPlanes        ,sizeof(int ),1,fp);
 fread(&bmph . bits            ,sizeof(int ),1,fp);
 fread(&bmph . biCompression   ,sizeof(long),1,fp);
 fread(&bmph . biSizeImage     ,sizeof(long),1,fp);
 fread(&bmph . biXPelsPerMeter ,sizeof(long),1,fp);
 fread(&bmph . biYPelsPerMeter ,sizeof(long),1,fp);
 fread(&bmph . biClrUsed       ,sizeof(long),1,fp);
 fread(&bmph . biClrImportant  ,sizeof(long),1,fp);

 if(bmph . biCompression!=0)
 {
   _setvideomode(_DEFAULTMODE);
   printf("\n Sorry, Cannot Display Compressed Bitmap (RLE)! \a\a\n");
   exit(-1);
 }

 if(bmph . bits==24)                  // True Color Mode
 {
 inregs.x.ax=0x006c;                  // Set TVGA Mode: 640x480x16M  levels
 int86(0x10,&inregs,&outregs);
 }
 else
 {
 inregs.x.ax=0x005d;                  // Set TVGA Mode: 640x480x256  levels
 int86(0x10,&inregs,&outregs);
 }

 if(bmph . bits<24)                   // Not True Color Mode
 {
 for(i=1;i<=(int)(bmph . headersize-54)/col;i++)
 {
   fread(fb,sizeof(char),col,fp);
   for(j=0;j<col;j+=4)                // Remap TVGA Palette
   {
   inregs.x.ax=0x1010;
   inregs.x.bx=index++;                           // Index value
   inregs.h.cl=fb[j]/(unsigned char)4;             // Blue  value
   inregs.h.ch=fb[j+1]/(unsigned char)4;           // Green value    R,G,B=0~63
   inregs.h.dh=fb[j+2]/(unsigned char)4;           // Red   value
   int86(0x10,&inregs,&outregs);
   }
 }
 if((bmph . headersize-54)%col!=0)
 {
   fread(fb,sizeof(char),(size_t)((bmph . headersize-54)%col),fp);
   for(j=0;j<(int)((bmph . headersize-54)%col);j+=4)               // Remap TVGA Palette
   {
   inregs.x.ax=0x1010;
   inregs.x.bx=index++;                           // Index value
   inregs.h.cl=fb[j]/(unsigned char)4;             // Blue  value
   inregs.h.ch=fb[j+1]/(unsigned char)4;           // Green value    R,G,B=0~63
   inregs.h.dh=fb[j+2]/(unsigned char)4;           // Red   value
   int86(0x10,&inregs,&outregs);
   }
 }
 }

  switch(bmph . bits)
  {
    case 1:
        bmph . width/=8;
        break;
    case 4:
        bmph . width/=2;
        break;
  }

  disp_width=(int)bmph . width;             // For Displaying Bitmap
  for(i=0;i<3;i++)
    if(bmph . width%4!=0) bmph . width++;     // For Reading File

  for(i=(int)bmph . depth-1;i>=0;i--)
  {
    if(bmph . bits==24)                     // True Color Mode
    {
    add=(long)i*640*3;          /* compute the address */
    fread(fb,sizeof(unsigned char),(size_t)bmph . width*3,fp);
    }
    else
    {
    add=(long)i*640;            /* compute the address */
    fread(fb,sizeof(unsigned char),(size_t)bmph . width,fp);
    }
    for(j=0;j<disp_width;j++)
    {
      seg=(int)(add>>16)^0x02;
      if(seg!=m_b)
      {
      m_b=seg;
      outp(0x3c4,0x0e);
      outp(0x3c5,seg);
      }
      bitpix=fb[j];
      switch(bmph . bits)
      {
      case 1:
          vp[add++]=(unsigned char)((bitpix>>7)&0x01);
          vp[add++]=(unsigned char)((bitpix>>6)&0x01);
          vp[add++]=(unsigned char)((bitpix>>5)&0x01);
          vp[add++]=(unsigned char)((bitpix>>4)&0x01);
          vp[add++]=(unsigned char)((bitpix>>3)&0x01);
          vp[add++]=(unsigned char)((bitpix>>2)&0x01);
          vp[add++]=(unsigned char)((bitpix>>1)&0x01);
          vp[add++]=(unsigned char)(bitpix&0x01);
          break;
      case 4:
          vp[add++]=(unsigned char)((bitpix>>4)&0x0f);
          vp[add++]=(unsigned char)(bitpix&0x0f);
          break;
      case 8:
          vp[add++]=bitpix;           /* write the GREY pixel */
          break;
      case 24:
          vp[add++]=fb[j*3];          /* write the BLUE  pixel */
          vp[add++]=fb[j*3+1];        /* write the GREEN pixel */
          vp[add++]=fb[j*3+2];        /* write the RED   pixel */
          break;
      default:
          break;
      }
    }
  }
  printf("\a");
  getch();
  fclose(fp);

 if(argc>1) goto ENDP;
 _setvideomode(_DEFAULTMODE);
 printf("Display Another Bitmap (Y/[N])? ");
 gets(arg);
 if(!strcmp(arg,"Y")||!strcmp(arg,"y"))
 {
 printf("Input image file name: ");
 gets(flnm);
 goto REDISP;
 }
 ENDP:
 _setvideomode(_DEFAULTMODE);
```