図面保存プログラムの開発

○　渡辺昌明（いわき明星大院）
正　高三徳（いわき明星大理工）

1．緒　言

現代は画像パターン情報の時代といっても過言ではないだろう．我々がありとあらゆる活動分野，娯楽・放送・出版をはじめ産業・経済・社会・生活・医療・教育・芸術などに，画像・映像情報が深く浸透し，人間にとって最も直感的・印象的で理解しやすい情報メディアである．画像ファイルの一般的なフォーマットとしては次の種類がある[1].

（１）BMP(BITMAP)：Windows 標準の画像形式で，無圧縮方式のため画質が良いが，サイズが大きい.

（２）PICT(Picture)：Macintosh 標準の画像形式で，フルカラー画像に加え，ベクターデータ(クラリスワークスなどで描いた図形･線など)，テキストデータも含めて保存できる.圧縮効率は悪い.

（３）JPEG (Joint Photographics Experts Group)：24bit カラー (約 1677 万色)に対応する形式で，似た色を同じ色に書き換えることで，フルカラーの画像を小さなファイルサイズに圧縮できる．一度圧縮すると元に戻せない非可逆圧縮である．高圧縮率で圧縮すると画質を大きく損ねる．また，色数の少ないイラスト画像や文字の含まれた画像には不向きである．デジタルカメラのフォーマットでもある.

（４）GIF(Graphics Interchange Format)：8bit カラー (256 色) またはモノクロ 256 階調に対応した圧縮形式で，可逆性圧縮である．特定の色を透明化する「透過 GIF」や，データの読み込みとともに段階的に画像を表示する「インターレース GIF」，複数の画像を連続的に表示して動画を表現する「アニメーション GIF」などの機能がある.

（５）PNG(Portable Network Graphic)：画像をネットワークでやり取りするために考えられたファイル形式．48bit カラー (約 280 兆色) まで対応できる．インデックスカラー，グレースケール，トゥルーカラーのすべてに対応画像を劣化させずに高効率の可逆圧縮が可能で，透過 PNG やアルファチャンネルにも対応しており，アニメーション機能については MNG（Multiple-image Network Graphics）という動画形式で対応できる.

これらの画像形式の比較を表１に示す.

表１　画像ファイルのフォーマットの比較

画像形式	圧縮形式	色　　数	圧縮率	主 な 用 途
ＢＭＰ	無圧縮形式	24Bit ｶﾗｰ(16,777,216 色)	無圧縮	Windows 基本形式
ＰＩＣＴ	可逆圧縮		低い	Macintosh 基本形式
ＪＰＥＧ	非可逆圧縮	24Bit ｶﾗｰ(16,777,216 色)	指定可能	写真・インターネット 色数の多いもの
ＧＩＦ	可逆圧縮	8Bit ｶﾗｰ(256 色)	高い	インターネット 色数の少ないもの
ＰＮＧ		48Bit ｶﾗｰ(約 280 兆色)		ＣＧ

この他にもコンピュータ上での閲覧を目的に作られた形式であり印刷に適したＰＤＦ(Portable Document Format)形式やビットマップ画像をさまざまなコンピュータ間で交換することを目的として開発された形式である TIFF (Tagged-Image File Format) など多くの画像形式が存在する．

画像ファイルにはさまざま形式があるが，画像を扱うアプリケーションの多くは操作できる画像ファイル形式が特定されているので，画像ファイルの形式を変換する必要性がある場合がある．そのような場合は画像形式を変換するためのソフトを使用しなければならない．

画面を保存するための機能として，キーボードには「PrintScreen」というキーがあり，デスクトップ画面全体またはアクティブウィンドウの画面表示を BMP としてクリップバッファーにコピーすることが可能である．しかし，アニメーションする画面を保存するとき等は，ユーザーが「PrintScreen」キーを押すタイミングが難しい．また，画面を複数保存するには非常に手間がかかる．

Windows はビットマップデータを Win32API 対応のデバイス独立ビットマップである DIB 形式（Device Independent Bitmap）[2]を使用して表示している．ディスク上のビットマップファイルは，この形式で格納されている．画像データを読み込んで表示するプログラムは多数ある[2, 3]が，保存する処理プログラムは公開されていない．そこで，本論文では Visual C++を用いて画面を DIB 形式によるビットマップのフューマットで保存するプログラムの開発について紹介する．

２．ＤＩＢ形式の構造

ＤＩＢ形式は基本的には通常のビットマップ（Ms-Windows 標準のラスタ形式）と同じだが，カラーテーブルを持ち，特定のディスプレイハードウェアに依存しないように設計されたビットマップである．ＤＩＢの格納形式は次の４つの部分から構成される．

（１）ファイルヘッダ（BITMAPFILEHEADER 構造体：ビットマップファイルの全体情報を保持する）
- ファイルのタイプ
- ファイルの全サイズ
- ビットマップデータの先頭までのオフセット

（２）情報ヘッダ（BITMAPINFOHHEADER 構造体：画像構成に関する情報を保持する）
- 画像の幅と高さ
- 色ビット数など

（３）カラーデータ（RGBQUAD 構造体：色調合の実情報を保持する）
- カラーパレットデータ

（４）ピクセルデータ
- 画像データ(1 ドットごとの色指定)

各部分の具体的な構造は次のようになっている．

（１）BITMAPFILEHEADER 構造体

```
typedef struct tagBITMAPFILEHEADER
{
    WORD  bfType;          //ファイルタイプ
    DWORD bfSize;          //ファイルサイズ
    WORD  bfReserved1;     //０予約
    WORD  bfReserved2;     //０予約
```

```
    DWORD bfOffBits;        //この構造体からﾋﾞｯﾄﾏｯﾌﾟﾃﾞｰﾀまでのｵﾌｾｯﾄ
} BITMAPFILEHEADER;
```

（２）BITMAPINFOHEADER 構造体：

```
typedef struct tagBITMAPINFOHEADER
{
    DWORD biSize;           //この構造体のサイズ（４０bytes）
    LONG  biWidth;          //画像幅（単位：ピクセル）
    LONG  biHeight;         //画像の高さ（単位：ピクセル）
    WORD  biPlanes;         //プレーン数(常に１)
    DWORD biBitCount;       //1 ピクセルあたりのカラービット数
    DWORD biCompression;    //圧縮状態
    DWORD biSizeImage;      //イメージの全バイト数
    DWORD biXPelsPerMeter;  //水平解像度
    DWORD biYPelsPerMeter;  //垂直解像度
    DWORD biClrUsed;        //実使用カラーインデックス数
    DWORD biClrImportant;   //重要カラーインデックス数
} BITMAPINFOHEADER;
```

（３）BITMAPINFO 構造体

```
typedef struct tagBITMAPINFO
{
    BITMAPINFOHEADER bmiHeader;      //BITMAPINFOHEADER 構造体
    RGBQUAD;         bmiColors[1];   //RGB カラー情報
} BITMAPINFO, FAR *LPBITMAPINFO, *PBITMAPINFO;
```

BITMAPINFOHEADER の情報に加えて色情報を保持する.

（４）RGBQUAD 構造体

```
typedef struct tagRGBQUAD
{
    BYTE  rgbBlue;          //青の輝度
    BYTE  rgbGreen;         //緑の輝度
    BYTE  rgbRed;           //赤の輝度
    BYTE  rgbReserved;      //未使用(必ず 0)
} RGBQUAD;
```

色は通常ＲＧＢで表現されるが，ＤＩＢ形式ではイメージ表示の際にＢＧＲで，赤と青が逆である．また，色ビット数が 16 未満ならパレット情報（色調合情報である RGBQUAD 構造体）が存在するが，16 ビット以上の場合，その配慮が必要にならない．今回は色が 16 ビット以上の場合を想定したものである．

３．画面保存プログラム

画面（画像）をＤＩＢ形式で保存するためには，ファイルヘッダ，情報ヘッダ，カラーデータ，ピク

セルデータを作成すればよい．そのためにこれらの構造と各構成の要素及びサイズ，各ピクセルの持つ青，緑，赤の色データを確定して保存するプログラムを次のように作成した．

```
SaveBMP()     //プログラムの名前
{
  CString filename;  //保存ファイル名
  CFile file;        //保存ファイル構造
  WORD  bfType=19778, bfReserved1=0, bfReserved2=0; //変数の宣言と値の設定
  DWORD bfSize, bfoffBits=54;
  DWORD biSize=40, biCompression=0, biSizeImage, biClrUsed=0, biClrImportant=0;
  LONG  biWidth, biHeight, biXPelsPerMeter=0, biYPelsPerMeter=0;
  WORD  biPlanes=1, biBitCount=24;
  RECT rc;           //画面の枠
  byte *rgb;         //画面全てのドットの色の R, G, B の値（B:青成分, G:緑成分, R:赤成分）
  int  i, j;
  COLORREF  color;   //各ドットの BGR の値
  GetClientRect(&rc);          //画面の左上および右下のコーナーの座標を得る
  biWidth=rc.right-rc.left;   //画面の幅を求める
  biHeight=rc.bottom-rc.top;   //画面の高さを求める
  biSizeImage=biWidth*biHeight*3; //全てのドットの R, G, B のバイト数
  bfSize=bfoffBits+biSizeImage;
  rgb=new byte[biSizeImage];       //全てのドットの R, G, B の保存メモリを確保
  for(j=0; j<biHeight; j++)
        for(i=0; i<biWidth; i++)
        {
         color=myDC.GetPixel(i, (biHeight-1)-j);     //各ドットの BGR の値を得る
          rgb[(j*biWidth+i)*3+2]=color&0xff;          //各ドットの R の値を得る
          rgb[(j*biWidth+i)*3+1]=(color>>8)&0xff;     //各ドットの G の値を得る
          rgb[(j*biWidth+i)*3+0]=(color>>16)&0xff;    //各ドットの B の値を得る
        }
  CFileDialog  myDLG(FALSE,"bmp",NULL,OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
        "BMP file(*.bmp)|*.bmp||");  //保存ファイル名を指定するﾀﾞｲｱﾛｸﾞﾎﾞｯｸｽ
  if(myDLG.DoModal()!=IDOK) return;
  filename=myDLG.GetPathName();       //保存ディレクトリを指定
  SetWindowText(filename);
  //データを開いたファイルに書き込む
  if(!file.Open(filename, CFile::modeWrite|CFile::modeCreate)) return;
  file.Write( &bfType, sizeof(WORD));
  file.Write( &bfSize, sizeof(DWORD));
  file.Write( &bfReserved1, sizeof(WORD));
```

```
    file.Write( &bfReserved2, sizeof(WORD));
    file.Write( &bfoffBits, sizeof(DWORD));
    file.Write( &biSize, sizeof(DWORD));
    file.Write( &biWidth, sizeof(LONG));
    file.Write( &biHeight, sizeof(LONG));
    file.Write( &biPlanes, sizeof(WORD));
    file.Write( &biBitCount, sizeof(WORD));
    file.Write( &biCompression, sizeof(DWORD));
    file.Write( &biSizeImage, sizeof(DWORD));
    file.Write( &biXPelsPerMeter, sizeof(LONG));
    file.Write( &biYPelsPerMeter, sizeof(LONG));
    file.Write( &biClrUsed, sizeof(DWORD));
    file.Write( &biClrImportant, sizeof(DWORD));
    file.Write( rgb, biSizeImage*sizeof(byte));
    file.Close();          //書き込みの終了
    delete []rgb;         // メモリの開放
  }
```

図１に示すように，ＤＩＢ形式の画像を表示するだけだと画像の上下が反転してしまう．これは，通常の画像形式ではイメージの左上を画像の始点として横方向にピクセルの並びを捉え，右下のピクセルがイメージの終点となっているのに対し，ＤＩＢ形式のＢＭＰでは，左下を始点として，右上を終点として扱っているためである．そのため，本論文のプログラムでは画像のｉ行 ｊ 列のピクセルをｉ行(biHeight-1)-j 列とすることで画像の反転を防いでいる．

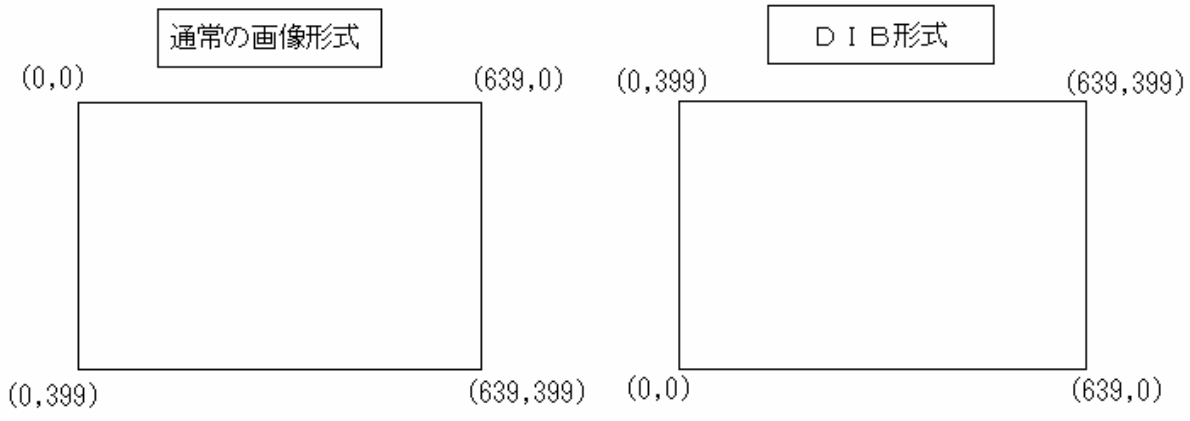

図１　通常画像とＤＩＢ形式の原点の比較

色データを保存するため，まず，ｉ行目{(biHeight-1)-j}列目のピクセルの色データ color を次の関数で取り出す．

```
color=myDC.GetPixel(i, (biHeight-1)-j);
```

ここで，color は COLORREF 型で，４バイトの 16 進数 0x00bbggrr の形式で表現されている．

次に，0x00bbggrr と 0xff のビットアンド(bit and)演算により，一番後の 8 ビットの赤成分 rr を次

のように抽出し，配列 rgb[(j*biWidth+i)*3+2]に記憶する．

```
rgb[(j*biWidth+i)*3+2]=color&0xff;
```

そして，0x00bbggrr を 8 ビット右シフトした 0x0000bbgg と 0xff のビット and 演算により緑成分 gg を次のように抽出し，配列 rgb[(j*biWidth+i)*3+1]に記憶する．

```
rgb[(j*biWidth+i)*3+1]=(color>>8)&0xff;
```

さらに，0x00bbggrr を 16 ビット右シフトした 0x000000bb と 0xff のビット and 演算により青成分 bb を次のように抽出し，配列 rgb[(j*biWidth+i)*3+0]に記憶する．

```
rgb[(j*biWidth+i)*3+0]=(color>>16)&0xff;
```

このように i を 0 から(biWidth-1)まで，j を 0 から(biHeight-1)まで繰り返すと，biWidth×biHeight の範囲の画面が保存される．また，マウスで指定した範囲を保存する場合，ウィンドウの枠 rc を指定の四角形の範囲に書き換えれば良い．

４．結　論

ＤＩＢ形式による画像保存のプログラムの作成に成功した．例として図２のように画像を保存することができた．この結果は通常の BMP 画像として，他の汎用描画ソフトでも表示することができる．開発したプログラムは Visual C++環境下においてＢＭＰの保存を可能とするものであり，他のプログラムにも応用できる．今後の課題として，ＢＭＰ以外の形式の画像ファイルを保存するプログラムの開発が期待される．

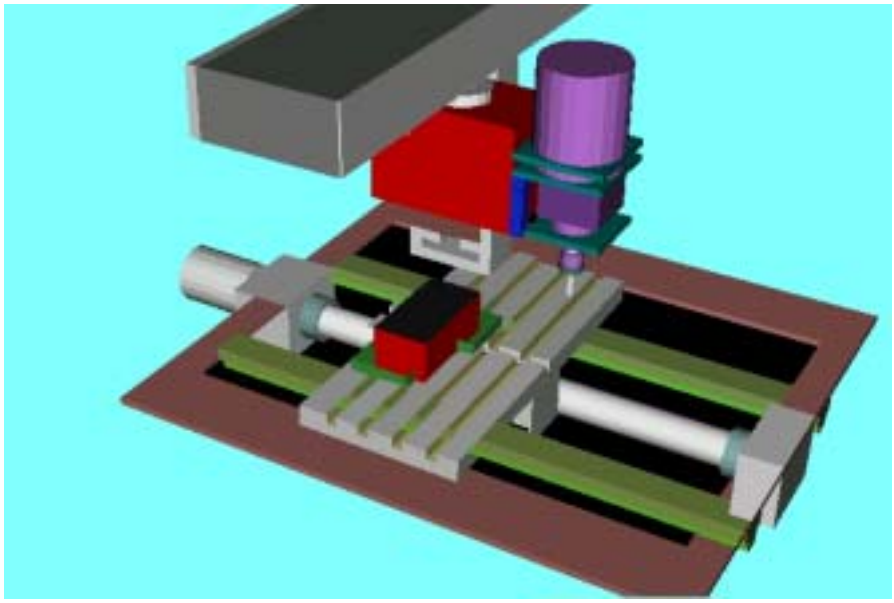

図２　画面保存の例

参考文献

[1] Ｃａｎｏｎ株式会社，http://bj.canon.co.jp/creative/

[2] 林晴比古，新 Visual C++6.0 入門シニア編，ソフトバンクパブリッシング株式会社，1999

[3] 田中ひろゆき，Visual C++6.0 の応用５０例，ソフトバンクパブリッシング株式会社，1998