EasyBMP User Manual (Version 0.57)

Paul Macklin

email: pmacklin@math.uci.edu
WWW: http://easybmp.sourceforge.net

May 12, 2005

Abstract

We define and document a simple, easy-to-use, cross-platform BMP library for the x86 architecture written in C++. (Due to endianess, this probably won't work on other architectures.) The EasyBMP library will work for input and output on 4-bit, 8-bit, 24-bit, and 32-bit Windows BMP files in Linux, Unix, and Windows.

Contents

1 What's New in this Release (Version 0.57)

In Version 0.57, we improved the handling of corrupted and/or truncated files; as of this writing, no file has been encountered that causes a program crash. (As expected, nonsensical files yield nonsensical outputs, but compiled programs won't choke when encountering gibberish.) Version 0.57 also has more consistent naming notions. (That is, we don't refer to palettes anymore.) But Version 0.57 is primarily a release intended for greater stability and robustness.

2 Introduction to the EasyBMP Library

In the course of my studies at the University of Minnesota and the University of California, I came to need a simple method to create and modify images. Because the Windows BMP file is nearly universally readable, flexible, and simple, I decided to work with this format. (No compression to worry about, potential for 8 bits per color channel or just 16 colors per pixel, etc.)

There are many excellent open- and closed-source BMP and image libraries available, and I in no way claim that anything here is even equal to those libraries. However, as I looked about I noticed that quite a few existing libraries had one or more of the following properties:

- too feature-rich (and accordingly more difficult to learn);
- required extensive installation;
- relied upon Linux or Windows libraries for simple functions;
- were too poorly documented for the novice programmer;
- required programming changes when moving code from one platform to another.

At that point, I decided to create my EasyBMP library. My goals included easy inclusion in C++ projects, ease of use, no dependence upon other libraries (totally self-contained), and cross-platform compatibility.

2.1 Sample Application: Converting a Color Image to Greyscale

Here, we give a first sample application using the EasyBMP library. Notice that inclusion of the library is simple: we include the `EasyBMP.h` file. In this application, we see a simple example of opening an existing BMP file, reading its RGB information, and manipulating and writing that information to another BMP file. The commands are pretty straightforward. This example should illustrate how easy the library is for even the novice programmer.

```
#include <iostream.h>
#include <fstream.h>
#include "EasyBMP.h"

int main( int argc, char* argv[] )
{
 if( argc != 3 )
 {
  cout << "Usage: ColorBMPtoGreyscale <input_filename> <output_filename>\n\n";
  return 1;
```

```
 }

 // declare and read the bitmap
 BMP Input;
 Input.ReadFromFile( argv[1] );

 // convert each pixel to greyscale
 for( int i=0 ; i < Input.TellWidth() ; i++)
 {
  for( int j=0 ; j < Input.TellHeight() ; j++)
  {
   double Temp = pow( Input(i,j)->Red  ,2.0) +
                 pow( Input(i,j)->Green,2.0) +
                 pow( Input(i,j)->Blue ,2.0);
   Temp = sqrt( Temp / 3.0 );
   Input(i,j)->Red   = (BYTE) Temp;
   Input(i,j)->Green = (BYTE) Temp;
   Input(i,j)->Blue  = (BYTE) Temp;
  }
 }

 // Create a greyscale color table if necessary
 if( Input.TellBitDepth() < 24 )
 { CreateGreyscaleColorTable( &Input.Colors , Input.TellBitDepth() ); }

 // write the output file
 Input.WriteToFile( argv[2] );

 return 0;
}
```

Additional code samples are available for download at

http://easybmp.sourceforge.net

3 Installing and Using the EasyBMP Library

Installing the EasyBMP library is easy. Simply copy all the `*.h` files to the directory of your project. Alternatively, copy all the header files anywhere in your compiler's path. You should have the following files:

1. `EasyBMP.h`
2. `EasyBMP_DataStructures.h`
3. `EasyBMP_StandardColorTables.h`
4. `EasyBMP_BMP.h`
5. `EasyBMP_VariousBMPutilities.h`

To use the EasyBMP library, simply include the EasyBMP.h file via

```
#include "EasyBMP.h"
```

Note that if you have copied all the EasyBMP header files to your compiler path, you may not need the quotes, but rather brackets:

```
#include <EasyBMP.h>
```

Compile your source code as you normally would; you don't have to link to anything. For instance, to compile the code example above with g++, use

```
g++ -o ColorBMPtoGreyscale ColorBMPtoGreyscale.cpp
```

4 Basic Bitmap Operations

As of Version 0.55, EasyBMP has a unified interface for all bit depths. To initialize a new BMP object, simply declare it:

Example:

```
// Declare a new bitmap object
BMP AnImage;
```

When you declare a BMP image, you will have a 1×1 blank 24-bit bitmap image. Next, set the size and bit depth of the image. You can do this either by reading an existing bitmap image or setting this information manually, as below:

Example:

```
BMP AnImage;
// Set size to 640 × 480
AnImage.SetSize(640,480);
// Set its color depth to 8-bits
AnImage.SetBitDepth(8);
// Declare another BMP image
BMP AnotherImage;
// Read from a file
AnotherImage.ReadFromFile("sample.bmp");
```

To check the bit depth, width, and height of a BMP object, use:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
cout << "File info:\n";
cout << AnImage.TellWidth() << " x " << AnImage.TellHeight()
     << " at " << AnImage.TellBitDepth() << " bits\n";
```

EasyBMP also provides a simple routine to compute and display the number of colors:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
cout << "colors:  " << AnImage.TellNumberOfColors() << "\n;
```

Note that for a 32-bit file, we don't regard two colors that differ only in the alpha channel as different colors; this function will state that 32-bit and 24-bit files have the same number of colors.

The bit depth and dimensions of a bitmap can be changed at any time:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
// Change the bit-depth
AnImage.SetBitDepth(8);
AnImage.SetBitDepth(24);
// Change the size
AnImage.SetSize(1024,768);
```

Note that whenever the bit depth is changed, any existing color table is erased. Likewise, whenever the size is changed, all pixels are deleted.

To access pixels, use `RGPApixel* operator()(int,int)`:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
// show the color of pixel (14,18)
cout << "(" << (int) AnImage(14,18)->Red << ","
            << (int) AnImage(14,18)->Green << ","
            << (int) AnImage(14,18)->Blue << ","
            << (int) AnImage(14,18)->Alpha << ")\n;"
// Change this pixel to a blue-greyish color
AnImage(14,18)->Red = 50;
AnImage(14,18)->Green = 50;
AnImage(14,18)->Blue = 192;
AnImage(14,18)->Alpha = 0;
```

Lastly, to save to a file, use:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
AnImage.WriteToFile("copied.bmp");
```

5 Advanced Usage: Modifying the Color Table

In EasyBMP_StandardColorTables.h, we have included two routines for changing the color table of a BMP object. If you want to set the color table to the "Windows standard" color table, use the following:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
AnImage.SetBitDepth(8);
CreateStandardColorTable( &(AnImage.Colors) , AnImage.TellBitDepth() );
```

Notice that in the example, the first argument is a pointer to the color table, and the second is the bit depth. Similarly, we can create a greyscale color table:

Example:

```
BMP AnImage;
AnImage.ReadFromFile("sample.bmp");
AnImage.SetBitDepth(4);
CreateGreyscaleColorTable( &(AnImage.Colors) , AnImage.TellBitDepth() );
```

If you want to modify a color table for a BMP file, it is best to do so by passing the memory address of the color table as well as the bit depth or number of colors. Be careful not to address more colors (RGBApixel's) than are expected for the given bit depth. In particular, any color table operation, when applied to a 24-bit or 32-bit file, should do nothing. Consider this example:

Example:

```
void CreateRedColorTable( RGBApixel* pColorTable , int Depth) {
{
 if( Depth > 8 ){ return; }
 int NumberOfColors = (int) pow(2,Depth); int i;
 BYTE StepSize = 256/NumberOfColors;
 for( i=0 ; i < NumberOfColors ; i++)
 {
  (*pColorTable)[i].Red   = i*StepSize;
  (*pColorTable)[i].Green = 0;
  (*pColorTable)[i].Blue  = 0;
  (*pColorTable)[i].Alpha = 0;
 }
}
```

To call this new function, you would do this:

Example:

```
BMP RedImage;
RedImage.ReadFromFile("sample.bmp");
CreateRedColorTable( &(RedImage.Colors) , RedImage.TellBitDepth() );
```

6 Extra Goodies: Various Bitmap Utilities

We have provided several sample utilities to make the library more immediately useful. We shall detail some of these goodies here. :-).

The first several utilities deal with getting file information from existing files.

- `void GetBitmapInfo( char* szFileNameIn )`: This routine gets the bitmap information from an existing bitmap file and outputs it to `cout`. All information is given. (width and height of image, bit depth, etc.)
- `BMFH GetBMFH( char* szFileNameIn )`: This returns a `BMFH` based on the file. See Section A for more information on the data structure.
- `BMIH GetBMIH( char* szFileNameIn )`: This returns a `BMIH` based on the file. See Section A for more information on the data structure.
- `int GetBitmapColorDepth( char* szFileNameIn )`: This routine returns the bit depth of the file.

The other provided functions are "cut 'n' paste" functions: they copy pixels from one `BMP` object to another, with or without transparency.

- ```
  void PixelToPixelCopy( BMP& From, int FromX, int FromY,
                         BMP& To, int ToX, int ToY)
  ```
  This function copies the `(FromX,FromY)` pixel of the `BMP` object `From` to pixel `(ToX,ToY)` of the `BMP` object `To`.
- ```
  void PixelToPixelCopyTransparent( BMP& From, int FromX, int FromY,
                                    BMP& To, int ToX, int ToY,
                                    RGBApixel& Transparent )
  ```
  This function copies the `(FromX,FromY)` pixel of the `BMP` object `From` to pixel `(ToX,ToY)` of the `BMP` object `To`, and it treats the input pixel as transparent if its color is `Transparent`. Here's an example:

  Example:

  ```
  BMP Image1;
  BMP Image2;
  Image1.ReadFromFile("Blah.bmp");
  Image2.SetSize(10,10);
  RGBApixel TransparentColor;
  TransparentColor.Red = 255;
  TransparentColor.Green = 255;
  TransparentColor.Blue = 255;
  PixelToPixelCopyTransparent(Image1,3,5,Image2,0,0,TransparentColor);
  ```

  Note that the alpha channel is ignored when considering transparency.
- ```
  void RangedPixelToPixelCopy( BMP& From, int FromL , int FromR, int FromB, int FromT,
                               BMP& To, int ToX, int ToY )
  ```
  This function copies a range of pixels from one image to another. It copies the rectangle `[FromL , FromR]` $\times$ `[FromB , FromT]` in image `From` to the rectangle whose top left corner
  ```

is `(ToX , ToY)` in image `To`. When using this function, don't forget that the top left corner of the image is `(0,0)` in the coordinate system! Also, `FromB` denotes the bottom edge of the rectangle, so `FromB` > `FromT`. However, if the algorithm detects that you accidentally reversed these numbers, it will automatically swap them for you. Lastly, if the rectangle you chose to copy from image `From` overlaps the boundary of image `To`, it will truncate the the copy selection, rather than give a nasty segmentation fault. :-)

- ```
  void RangedPixelToPixelCopyTransparent(
       BMP& From, int FromL , int FromR, int FromB, int FromT,
       BMP& To, int ToX, int ToY ,
       RGBApixel& Transparent )
  ```

  This function does the same thing as the previous function, but with support for transparency. As in the example for the pixel-to-pixel copy above, you specify a transparent color of type `RGBApixel`.

7 Known Bugs and Quirks

There may be some rare instances where EasyBMP crashes when reading corrupted, damaged, or incomplete bitmap files. However, as of Version 0.57, none have been encountered. EasyBMP will probaly yield strangely-sized or oddly-colored image when dealing with corrupted files, but that's the nature of corrupted data. :-)

If there is one quirk in the library, it is that the alpha channel is largely unused. Almost all operations completely ignore the alpha channel. However, it is there if you should choose to use it. Future releases of EasyBMP may take advantage of it for blending pixels, etc.

8 Future Changes

The most important thing that we hope to work on for the next several releases is making EasyBMP more resilient to corruption in existing BMP files when reading them. In the future, we hope to continue to improve support for modifying color tables. Future work should also take better advantage of the alpha channel.

Another future tool would be the auto-generation of an *optimal* color table for 4 and 8-bit bitmap files. As the code currently stands, a "standard" color table that matches what Windows would create is used when no table is otherwise specified. The generation of an optimal color table would likely be slow, but it would be a good option. Some experimentation is already under way, but it may very well be quite some time before it ever appears in the library.

Another extension of the library would be good interfaces to OpenGL for image mapping. I am currently in the process of doing this.

It would be nice (for completeness) to create support for 1-bit files. However, in practice, these images are rarely seen. 4-bit files are already quite small for what they offer.

Lastly, future releases may make a greater distinction between private and public stuff in the classes.
  ```

A Classes and BMP Data Types

Here, we detail the various classes and data types and how to interface with them.

A.1 Miscellany

Some of the data types that are used in the construction of more complex data types are:

Type:	Info:
`BYTE`	an unsigned character of 8 bits
`WORD`	an unsigned short of 16 bits
`DWORD`	an unsigned long of 32 bits
`BMFH`	a specific header format for a BMP file
`BMIH`	provides additional information on the BMP file

For additional information on the `BMFH` and `BMIH` classes, I highly recommend that you visit

`http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html`.

A.2 RGBApixel

This data structure is exactly as their its suggests: a single pixel of (red,green,blue,alpha) data. This data structure is used both for individual pixels within an image and the color table in the palette. Here are the details on the data structure:

Member:	Function:
`Blue`	blue pixel info of type `BYTE`
`Green`	green pixel info of type `BYTE`
`Red`	red pixel info of type `BYTE`
`Alpha`	alpha pixel info of type `BYTE`

A.3 BMP

The `BMP` class consists of all the necessary pixel information for a Windows bitmap file, along with file I/O routines.

- `int BitDepth`: This gives the number of bits per pixel, i.e., the color depth. This data member is private and can only be accessed through `TellBitDepth` and `SetBitDepth`.
- `int Width`: This gives the width of the bitmap in pixels. This data member is private and can only be accessed through `TellWidth` and `SetSize`.
- `int Height`: This gives the height of the bitmap in pixels. This data member is private and can only be accessed through `TellHeight` and `SetSize`.
- `RGBApixel** Pixels`: This is the actual `Width` $\times$ `Height` array of `RGBApixel`'s.
- `RGBApixel* Colors`: This is the table of colors, stored as `RGBApixel`'s. If the `BMP` object is 24-bits or 32-bits, then `Colors = NULL`.
- `int TellBitDepth( void )`: This function outputs the bit depth of the `BMP` object.
- `int TellWidth( void )`: This function outputs the width of the `BMP` object.
- `int TellHeight( void )`: This function outputs the height of the `BMP` object.

- `int TellNumberOfColors( void )`: This function outputs the number of colors of the `BMP` object.
- `BMP()`: This constructor creates a 1×1, 24-bit `BMP` object.
- `~BMP()`: This is the destructor. You should never call this; it is automatically called when a `BMP` object goes out of scope.
- `RGBApixel* operator()(int i, int j)`: This returns a pointer to the `(i,j)` pixel.

  Example:

  ```
  BMP Sample;
  Sample.SetSize(10,10);
  Sample(3,4)->Red = 255;
  Sample(3,4)->Alpha = 0;
  Sample(3,4)->Blue = Sample(3,4)->Red;
  ```

- `void SetSize( int NewWidth, int NewHeight )`: Use this to change the size of the object to `NewWidth` $\times$ `NewHeight`. See the example above.
- `void SetBitDepth( int NewDepth )`: This function changes the bit depth to `NewDepth` bits per pixel. It also automatically creates and/or resizes the color table, if necessary.
- `void WriteToFile( char* FileName )`: This function writes the current `BMP` object to the file `FileName`.
- `void ReadFromFile( char* FileName )`: This function reads the file `FileName` into the current `BMP` object.