Reading, Writing, and Modifying BMP Files Using C++

by Robert J. Yager
NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer’s or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.
Reading, Writing, and Modifying BMP Files Using C++

Robert J. Yager
Weapons and Materials Research Directorate, ARL
BMP files can be used to store images as arrays of pixels. This report presents a set of functions, written in C++, that can be used to read, write, and modify 24-bit, uncompressed BMP files.
Contents

List of Figures v
List of Tables v
Acknowledgments vi
1. Introduction 1
2. BMP Headers 1
3. BMP Pixel Data 2
   4.1 NewImage() Code ................................................................. 2
   4.2 NewImage() Parameters ....................................................... 2
   4.3 NewImage() Return Value .................................................... 3
5. Reading BMP Files – the ReadBmpFile() Function 3
   5.1 ReadBmpFile() Code ............................................................ 3
   5.2 ReadBmpFile() Parameters .................................................. 3
   5.3 ReadBmpFile() Return Value ............................................... 3
6. Writing BMP Files – the WriteBmpFile() Function 4
   6.1 WriteBmpFile() Code .......................................................... 4
   6.2 WriteBmpFile() Parameters ................................................ 4
   6.3 WriteBmpFile() Return Value .............................................. 4
7. Accessing Individual Pixels – the GetPixel() Function 4
   7.1 GetPixel() Code ................................................................. 4
   7.2 GetPixel() Parameters ........................................................ 5
   7.3 GetPixel() Return Value ...................................................... 5
8. Image Dimensions – GetWidth(), GetHeight(), and GetSize() Functions

8.1 GetWidth() Code .................................................5
8.2 GetWidth() Parameters ........................................5
8.3 GetWidth() Return Value ..................................5
8.4 GetHeight() Code ...............................................5
8.5 GetHeight() Parameters ....................................5
8.6 GetHeight() Return Value ..................................6
8.7 GetSize() Code ..................................................6
8.8 GetSize() Parameters .......................................6
8.9 GetSize() Return Value ...................................6

9. Copying Images – The CopyImage() Function

9.1 CopyImage() Code ..............................................7
9.2 CopyImage() Parameters ...................................7
9.3 CopyImage() Return Value .................................7

10. Example – Creating an Image of the Mandelbrot Set

11. Example – Adjusting Contrast Level

12. Summary

Distribution List
List of Figures

Figure 1. Source and destination images. .................................................................6
Figure 2. The BMP file mandelbrot.bmp, created using example code. .......................8
Figure 3. Incrementally increasing levels of contrast (top-left is the original image). ..........10

List of Tables

Table 1. BMP header values. ......................................................................................1
Acknowledgments

The author would like to thank Dr. Benjamin Breech of the U.S. Army Research Laboratory’s Weapons and Materials Research Directorate. Dr. Breech provided technical and editorial recommendations that improved the quality of this report.
1. Introduction

BMP files can be used to store images as arrays of pixels. This report presents a set of functions, written in C++, that can be used to read, write, and modify 24-bit, uncompressed BMP files. A summary sheet is provided at the end of this report. It presents the yBmp namespace, which contains the eight functions that are described in this report.

2. BMP Headers

BMP files begin with header information that stores, among other things, the size of the image contained in the BMP file. BMP headers store numeric values in little-endian order. Bitwise-shift operators can be used to convert to and from little-endian format. Table 1 lists the contents of the BMP headers that are associated with the BMP files that are described in this report.

Table 1. BMP header values.

<table>
<thead>
<tr>
<th>Offset</th>
<th>Size</th>
<th>Decimal Value (0 to 255)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>- 66 77</td>
<td>&quot;BM&quot;</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>N N&gt;&gt;8 N&gt;&gt;16 N&gt;&gt;24</td>
<td>size, N, of a BMP file</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>- 0 0</td>
<td>application specific - not used by the yBmp namespace</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>- 0 0</td>
<td>application specific - not used by the yBmp namespace</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>54 0 0 0</td>
<td>number of bytes from beginning of file to beginning of pixel data</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>40 0 0 0</td>
<td>number of bytes from this point to the beginning of pixel data</td>
</tr>
<tr>
<td>18</td>
<td>4</td>
<td>w w&gt;&gt;8 0 0</td>
<td>width, w, of image (in pixels, limited to $2^{16}-1=65,535$)</td>
</tr>
<tr>
<td>22</td>
<td>4</td>
<td>h h&gt;&gt;8 0 0</td>
<td>Height, h, of image (in pixels, limited to $2^{16}-1=65,535$). Height values can be positive or negative. Negative values imply that the image is mirrored, top to bottom</td>
</tr>
<tr>
<td>26</td>
<td>2</td>
<td>- 1 0</td>
<td>number of color planes used</td>
</tr>
<tr>
<td>28</td>
<td>2</td>
<td>- 24 0</td>
<td>number of bits per pixel</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>0 0 0 0</td>
<td>no compression</td>
</tr>
<tr>
<td>34</td>
<td>4</td>
<td>n n&gt;&gt;8 n&gt;&gt;16 n&gt;&gt;24</td>
<td>size, n, of image (n=h*(3*w+w%4)=N-54)</td>
</tr>
<tr>
<td>38</td>
<td>4</td>
<td>19 11 0 0</td>
<td>horizontal resolution of image (2835 pixels/meter)</td>
</tr>
<tr>
<td>42</td>
<td>4</td>
<td>19 11 0 0</td>
<td>vertical resolution of image (2835 pixels/meter)</td>
</tr>
<tr>
<td>46</td>
<td>4</td>
<td>0 0 0 0</td>
<td>number of colors in the palette</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0 0 0 0</td>
<td>all colors are important</td>
</tr>
</tbody>
</table>
3. BMP Pixel Data

Three 8-bit integers, each with values 0 to 255, are used to define the color of each pixel in an image. The integers describe the intensity of blue, green, and red contributions to a pixel’s color. For example, {0,0,255} is red, while {255,0,0} is blue. Pixels are ordered by column, then by row, beginning with either the bottom-left corner of the image for positive height values or the top-left corner of the image for negative height values. The end of each row of pixel data must be padded to ensure that the number of 8-bit integers stored in each row is evenly divisible by 4.

4. Creating New Images – the NewImage() Function

New images can be created using the NewImage() function.

Note that the NewImage() function uses the “new” command to allocate memory for the image that is pointed to by the return value. Thus, to avoid memory leaks, each use of the NewImage() function should be accompanied by a use of the “delete[]” operator.

4.1 NewImage() Code

```c
inline unsigned char* NewImage(unsigned int w, int h, unsigned char f=0)
{
    unsigned int n=abs(h)*(3*w+w%4),N=n+54;
    unsigned char H[54]={66,77,N,N>>8,N>>16,N>>24,0,0,0,0,54,0,0,0,40,0,0,0,w,
        w>>8,0,0,h,h>>8,h>>16,h>>24,1,0,24,0,0,0,0,n,n>>8,
        n>>16,n>>24,19,11,0,0,0,0,0,19,11,0,0,0,0,0,0,0,0,0};
    unsigned char* I=new unsigned char[N];
    memcpy(I,H,54),memset(I+54,f,n);
    return I;
}
```

4.2 NewImage() Parameters

- **w** specifies the width of an image, in pixels (1 to $2^{16}–1$).
- **h** specifies the height of an image, in pixels (1 to $2^{16}–1$). Values for **h** can be positive or negative. Positive height values imply that pixels are ordered beginning with the bottom-left corner of an image. Negative height values imply that pixels are ordered beginning with the top-left corner of an image.
- **f** specifies a numeric value (0 to 255) that will be used to fill the pixel values. Use 0 to fill the image with black pixels, use 255 to fill the image with white pixels. The default value is 0.
4.3 NewImage() Return Value

The NewImage() function returns a pointer to a newly created image.

5. Reading BMP Files – the ReadBmpFile() Function

The ReadBmpFile() function can be used to create an image from a BMP file.

Note that the ReadBmpFile() function uses the “new” command to allocate memory for the image that is pointed to by the return value. Thus, to avoid memory leaks, each use of the ReadBmpFile() function should be accompanied by a use of the “delete[]” operator.

5.1 ReadBmpFile() Code

```cpp
inline unsigned char* ReadBmpFile(const char* filename) {
    FILE* f = fopen(filename, "rb");
    if (!f) {
        printf("Can't open "%s".
", filename);
        exit(1);
    }
    unsigned char H[6];
    fread(H, 1, 6, f);
    rewind(f);
    unsigned char* I = new unsigned char[N];
    fread(I, 1, N, f);
    fclose(f);
    return I;
}
```

5.2 ReadBmpFile() Parameters

filename specifies the name of a 24-bit, uncompressed BMP file.

5.3 ReadBmpFile() Return Value

The ReadBmpFile() function returns a pointer to an image that contains the header and pixel data from the BMP file that was specified by the filename parameter.

If the file that is specified by filename cannot be opened, the ReadBmpFile() function calls the exit() function with status code of 1. Inability to open a file is typically the result of an incorrectly specified filename or path.
6. Writing BMP Files – the WriteBmpFile() Function

The WriteBmpFile() function can be used to write an image to a BMP file.

6.1 WriteBmpFile() Code

```c
inline int WriteBmpFile(const char* filename, const unsigned char* I) {
    FILE* f = fopen(filename, "wb");
    if (!f) {
        printf("Can't open "%s".
", filename);
        exit(1);
    }
    unsigned int N = *(I + 2) + (*(I + 3) << 8) + (*(I + 4) << 16) + (*(I + 5) << 24);
    unsigned int n = fwrite(I, 1, N, f);
    fclose(f);
    return n;
}
```

6.2 WriteBmpFile() Parameters

- **filename**: specifies the name of a BMP file.
- **I**: specifies a pointer to an image.

6.3 WriteBmpFile() Return Value

The WriteBmpFile() function returns the number of characters that were written to the output file.

If the file that is specified by **filename** cannot be opened, the WriteBmpFile() function calls the `exit()` function with status code of 1. Inability to open a file is often the result of an incorrectly specified filename or path. However, it can also be the result of the file being marked as read only, as may be the case if the file is open in another program.

7. Accessing Individual Pixels – the GetPixel() Function

The GetPixel() function returns a pointer to a pixel in an image. The pointer can be used to read or modify the pixel that it points to.

7.1 GetPixel() Code

```c
inline unsigned char* GetPixel(unsigned char* I, unsigned int i, unsigned int j) {
    return I + 54 + j * ((*(I + 18) + (*(I + 19) << 8)) * 3 + *(I + 18) % 4) + 3 * i;
}
```
7.2 GetPixel() Parameters

- **I** specifies a pointer to an image.
- **i** specifies a column in an image. Column numbers increase moving left to right. Allowable values for i are 0≤i<w, where w is the width of an image.
- **j** specifies a row in an image. Row numbers increase moving bottom to top for images with positive heights, and top to bottom for images with negative heights. Allowable values for j are 0≤j<|h|, where h is the height of an image.

7.3 GetPixel() Return Value

The GetPixel() function returns a pointer to a pixel.

8. Image Dimensions – GetWidth(), GetHeight(), and GetSize() Functions

The GetWidth() function can be used to determine the width (in pixels) of an image, the GetHeight() function can be used to determine the height (in pixels) of an image, and the GetSize() function can be used to determine the size (in bytes) of an image.

8.1 GetWidth() Code

```c
inline unsigned int GetWidth(const unsigned char* I) //<---------------RETRIEVES THE WIDTH OF AN IMAGE
    { //<--------A POINTER TO THE BEGINNING OF A BMP IMAGE
    return *(I+18)+(*(I+19)<<8); //~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~14AUG2013~~~~~~
}
```

8.2 GetWidth() Parameters

- **I** specifies a pointer to an image.

8.3 GetWidth() Return Value

The GetWidth() function returns the width of the input-specified image, in pixels.

8.4 GetHeight() Code

```c
inline int GetHeight(const unsigned char* I) //<-------------------RETRIEVES THE HEIGHT OF AN IMAGE
    { //<--------A POINTER TO THE BEGINNING OF A BMP IMAGE
    return *(I+22)+(*(I+23)<<8)-(*(I+25)==255?65536:0); //~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~14AUG2013~~~~~~
}
```

8.5 GetHeight() Parameters

- **I** specifies a pointer to an image.
8.6 GetHeight() Return Value

The GetHeight() function returns the height of the input-specified image, in pixels. Values can be positive or negative. Positive height values imply that pixels are ordered beginning with the bottom-left corner of an image. Negative height values imply that pixels are ordered beginning with the top-left corner of an image.

8.7 GetSize() Code

```c
inline unsigned int GetSize(const unsigned char* I){//<====RETRIEVES THE SIZE OF AN IMAGE'S PIXEL DATA
    unsigned int w=GetWidth(I);
    return abs(GetHeight(I))*(3*w+w%4);
}~YAGENAUT@GMAIL.COM~LAST~UPDATED~14AUG2013~~~
```

8.8 GetSize() Parameters

I specifies a pointer to an image.

8.9 GetSize() Return Value

The GetSize() function returns the size of the pixel portion of an image, in bytes.

---

9. Copying Images – The CopyImage() Function

The CopyImage() function can be used to copy pixel data from one image to another. The two images need not be the same size and can be offset from each other, as shown in figure 1.

---

Figure 1. Source and destination images.
9.1 CopyImage() Code

```c
inline unsigned char* CopyImage(//<======COPY IMAGE FROM SOURCE TO DESTINATION
    unsigned char* S,//<------------------------------------------SOURCE IMAGE
    unsigned char* D,//<-------------------------------------DESTINATION IMAGE
    int w, int h){//<---------------------------------WIDTH AND HEIGHT OFFSETS
    int wd=GetWidth(D),W=GetWidth(S)+w,hd=abs(GetHeight(D)),
        H=abs(GetHeight(S))+(w>0?w:0),I=wd<W?wd:W;
    if(i<I)for(unsigned int j=h>0?h:0,J=hd<H?hd:H;j<J;++j)
        memcpy(GetPixel(D,i,j),GetPixel(S,i-w,j-h),3*(I-i));
    return D;
}//~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~14AUG2013~~~~~~
```

9.2 CopyImage() Parameters

- **S**: specifies a pointer to a source image.
- **D**: specifies a pointer to a destination image.
- **w**: specifies a width offset (see figure 1).
- **h**: specifies a height offset (see figure 1).

9.3 CopyImage() Return Value

The CopyImage() function returns a pointer to the destination image (D).

---

10. Example – Creating an Image of the Mandelbrot Set

The following example uses functions from the yBmp namespace to create an image of the Mandelbrot set. The example begins by creating an image that is 1400 pixels wide by 800 pixels high. The example then uses the escape-time algorithm\(^1\) to generate an integer value (k) for each pixel location. Next, values of k are mapped to intensities of blue and the colors of individual pixels are set using the GetPixel() function. Finally, the WriteBmpFile() function is used to create a BMP file. The result is displayed in figure 2.

---

#include <cmath>
#include "y_bmp.h"

int main(){
    int w=1400,h=800;
    unsigned char* I=yBmp::NewImage(w,h);
    for(int i=0,j,k,N=1000;i<w;++i)
        for(j=0;j<h;++j){
            double x=-2.5+i*3.5/(w-1) , y=-1+j*2./(h-1) , re=0 , im=0 , t;
            for(k=0;k<N&hypot(re,im)<4;k+=50) t=re*re-im*im+x , im=2*re*im+y , re=t;
            *yBmp::GetPixel(I,i,j)=k<N?k*255/N:0;}
    yBmp::WriteBmpFile("mandelbrot.bmp",I);
    delete I;
} //~~~~~~~~~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~14AUG2013~~~~~~

Figure 2. The BMP file mandelbrot.bmp, created using example code.

11. Example – Adjusting Contrast Level

The following example uses functions from the yBmp namespace, as well as the included AdjustContrast() function, to change the contrast of a photograph. The example begins by using the ReadBmpFile() function to convert a BMP file to an image. Next the NewImage() function is used to create an image that is twice as wide and twice as high as the original image. The original image is then copied into the top-left quadrant of the new image using the CopyImage() function. The AdjustContrast() function is then used to modify the original image. Modified images are then copied to the remaining three quadrants of the new image, proceeding clockwise. Finally,
the WriteBmpFile() function is used to convert the new image to a BMP file. Figure 3 displays the contents of the newly created file.

```cpp
#include <algorithm>//.....................................................max()
#include "y_bmp.h"

void AdjustContrast(//<===================ADJUSTS THE CONTRAST RATIO OF AN IMAGE
    unsigned char* I,//<--------------------------------A POINTER TO A BMP IMAGE
double contrast){//<-----------------------CONTRAST SETTING (0 => No Change)
    for(unsigned int i=0,n=GetSize(I);i<n;++i)
        I[i+54]=unsigned char(std::max((1+contrast)*I[i+54]-contrast*255,0.));
}~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~13AUG2013~~~~~~

int main(){
    unsigned char* I=yBmp::ReadBmpFile("image.bmp");
    int w=yBmp::GetWidth(I),h=yBmp::GetHeight(I);
    unsigned char* I2=yBmp::NewImage(2*w,2*h);
    yBmp::CopyImage(I,I2,0,h);~top-left image
    AdjustContrast(I,.25),yBmp::CopyImage(I,I2,w,h);~top-right image
    AdjustContrast(I,.25),yBmp::CopyImage(I,I2,w,0);~bottom-right image
    AdjustContrast(I,.25);yBmp::CopyImage(I,I2,0,0);~bottom-left image
    yBmp::WriteBmpFile("composite_contrast.bmp",I2);
    delete I, delete I2;
}~YAGENAUT@GMAIL.COM~~~~~~~~~~~~~~~~~~~~~~~~~LAST~UPDATED~14AUG2013~~~~~~
```
12. Summary

A summary sheet is provided at the end of this report. It presents the yBmp namespace, which contains the eight functions that are described in this report.
Example 1 Output
(rotated 90° clockwise)

Example 2 Output
(rotated 90° clockwise)

CopyImage() Parameters
<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DEFENSE TECHNICAL INFORMATION CTR DTIC OCA</td>
</tr>
<tr>
<td>2</td>
<td>DIRECTOR US ARMY RESEARCH LAB RDRL CIO LL IMAL HRA MAIL &amp; RECORDS MGMT</td>
</tr>
<tr>
<td>1</td>
<td>GOVT PRINTG OFC A MALHOTRA</td>
</tr>
</tbody>
</table>

ABERDEEN PROVING GROUND

<table>
<thead>
<tr>
<th>NO. OF COPIES</th>
<th>ORGANIZATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DIR USARL RDRL WML A R YAGER</td>
</tr>
</tbody>
</table>