Digital Image Processing (2023) HW1

I. Image Input/Flip/Output

BMP 影像(Bitmap)通常以 Raster 形式將影像儲存而成，其也具備壓縮版與非壓縮，但通常未壓縮的較常見。其格式為可區分成三大區塊：Header, Corlor Table, 及 Image Data。

Header (54 bytes)		
Bitmap File Header (14 bytes)	Signature (2 bytes)	In Windows system it is usually “0x42 0x4D” in Hexadecimal, same as “BM” in ASCII.
	File Size (4 bytes)	Total .bmp file size (unit: byte)
General Info about the Image	Reserved (4 bytes)	Reserved field, typically set to 0
	Data Offset (4 bytes)	The offset from the beginning of the file to the start of the image data. (unit: byte)
Bitmap Info Header (40 bytes)	Header Size (4 bytes)	The size of the Info Header, usually set to 40 bytes.
	Image Width (4 bytes)	The width of the image in pixels
	Image Height (4 bytes)	The height of the image in pixels
	Color Planes (2 bytes)	The number of color planes (usually 1)
	Bits per Pixel (2 bytes)	The number of bits used to represent each pixel (typically 1, 4 ,16, 24, or 32)
Info specific to the image	Compression (4 bytes)	Compression method used (if no compressed, set it to 0)
	Image Size (4 bytes)	The size of the image data in bytes (if no compressed, it can be set to 0)
	Horizontal Resolution (4 bytes)	Pixels per meter
	Vertical Resolution (4 bytes)	Pixel per meter
	Colors Used (4 bytes)	Number of colors in the color palette, or 0 for maximum (2^n, where n is the Bits per Pixel)
	Important Colors (4 bytes)	Number of the important colors, or 0 for all colors

Color Table (Optional)
The color table, also known as the palette, contains color information for indexed color images (bits per pixel < 24). Each entry in the color table is 4 bytes (RGBA)
Image Data
The image data itself, stored row by row. The data is organized from bottom to top, left to right, with each row padded to a multiple of 4 bytes.

根據助教公告的題目規則，由於使用 24 與 32bit per pixel 的圖片，因此不需要考慮 Color Tabel 的存在。因此實際上讀檔案，僅需切割 Header(54 bytes)與 Image Data。

```cpp
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

#pragma pack(push, 1) // Ensure that structures are byte-aligned
struct BMPHeader {
    uint16_t type; // Magic identifier "BM"
    uint32_t size; // File size
    uint32_t reserved; // Reserved
    uint32_t offset; // Offset to image data
    uint32_t dibHeaderSize; // DIB Header size
    int32_t width; // Image width
    int32_t height; // Image height (positive for bottom-up)
    uint16_t planes; // Number of color planes (must be 1)
    uint16_t bitsPerPixel; // Bits per pixel
    uint32_t compression; // Compression method
    uint32_t imageSize; // Image data size
    int32_t xPixelsPerMeter; // Horizontal pixels per meter
    int32_t yPixelsPerMeter; // Vertical pixels per meter
    uint32_t colorsUsed; // Number of colors in the palette
    uint32_t colorsImportant; // Important colors
};
#pragma pack(pop)
```

圖一、Struct for BMP Header

```cpp
    //* Open the input bmp file
    std::ifstream inputFile(inFileName, std::ios::binary);
    if (!inputFile) {
        std::cerr << "[Error] Failed to open input BMP file." << std::endl;
        return 1;
    }

    //* Read the input bmp header
    BMPHeader header;
    inputFile.read(reinterpret_cast<char*>(&header), sizeof(BMPHeader));

    //* Check if the input bmp file is valid
    if (header.type != 0x4D42) {
        std::cerr << "[Error] Invalid BMP file format." << std::endl;
        inputFile.close();
        return 1;
    }
    int channel = header.bitsPerPixel/8;
    //* Read the input bmp file
    const int imageSize = header.width * header.height * channel;
    std::cout << "[INFO] Input File Width: " << header.width << std::endl;
    std::cout << "[INFO] Input File Height: " << header.height << std::endl;
    std::cout << "[INFO] Input File Channel: " << channel << std::endl;
    std::vector<char> imageData(imageSize);
    inputFile.read(imageData.data(), imageSize);
    inputFile.close();
```

圖二、Read BMP input file

```cpp
    //* Flip the image vertically
    const int bytesPerRow = header.width * channel;
    std::vector<char> flippedImageData(imageSize);
    for (int y = 0; y < header.height; y++) {
        for (int x = 0; x < header.width; x++){
            memcpy( &flippedImageData[y * bytesPerRow+ channel*x],
                    &imageData[y* bytesPerRow+ channel*(header.width-x)],
                    channel);
        }
    }
```

圖三、Flip horizontally

```cpp
    //* Create/Open the output bmp file
    std::ofstream outputFile(outFileName, std::ios::binary);
    if (!outputFile) {
        std::cerr << "[Error] Failed to create output BMP file." << std::endl;
        return 1;
    }

    //* Output the header and image raw data
    outputFile.write(reinterpret_cast<const char*>(&header), sizeof(BMPHeader));
    outputFile.write(flippedImageData.data(), imageSize);
    outputFile.close();
```

圖四、Write BMP output file

第一份程式主要將 input bmp 做水平翻轉後，即可直接輸出。使用 fstream 函式庫處理 binary file(.bmp)讀寫，並透過 vector 函式庫承載欲讀寫的資料。首先如圖一、圖二所示，使用 struct type 將 Header 格式讀出。隨後創建一個 flippedImageData 的容器以水平翻轉的方式複製將原圖的資料進去（圖三）。最後，由於檔案大小、圖片大小、屬性皆沒有改變，因此寫入 output bmp 時，Header 可以直接沿用原先讀進來的 Header，接著把 flippedImageData 的內容寫入檔案即可。

	
原圖	水平翻轉

II. Resolution

```cpp
//* Quantize Scale Setting
std::istringstream new_bit(argv[2]);
int new_bit_x;
if (!(new_bit >> new_bit_x)) {
    std::cerr << "Invalid number: " << argv[2] << '\n';
}
else if (!new_bit.eof()) {
    std::cerr << "Trailing characters after number: " << argv[2] << '\n';
}
int QUANT_SCALE = 8 - new_bit_x;
int quant_order = QUANT_SCALE/2;
```

```cpp
//* Read the input bmp header
BMPHeader header;
inputFile.read(reinterpret_cast<char*>(&header), sizeof(BMPHeader));

//* Check if the input bmp file is valid
if (header.type != 0x4D42) {
    std::cerr << "[Error] Invalid BMP file format." << std::endl;
    inputFile.close();
    return 1;
}
int channel = header.bitsPerPixel/8;
//* Read the input bmp file
const int imageSize = header.width * header.height * channel;
std::cout << "[INFO] Input File Width: " << header.width << std::endl;
std::cout << "[INFO] Input File Height: " << header.height << std::endl;
std::cout << "[INFO] Input File Channel: " << channel << std::endl;
std::vector<char> imageData(imageSize);
inputFile.read(imageData.data(), imageSize);
inputFile.close();
```

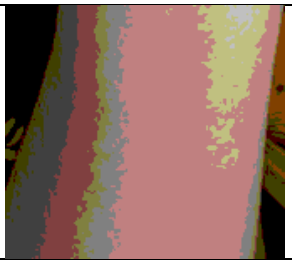
```cpp
//* Quantize the Image
std::vector<char> quant_image(imageSize);
for (int i = 0; i < imageSize; i += 1) {
    const char byte_data = imageData[i];
    quant_image[i]   = ( byte_data >> QUANT_SCALE) << QUANT_SCALE;

}
```

此份程式主要是模擬資料刪減，達到 Resolution 降低。原理是保留每個 byte 資料（一個 pixel 內的一個 channel data）靠近 MSB 區段，其他則可以空下設為 0，或是藏匿其他資訊。由於是保留 MSB 區段，若保留的越多，則與原圖的細節部分也越相近。

運算時可以透過 C 語言的 shift operator 右移捨去 data 後面的 Quantize bit，隨後在重新用 shift 左移補 0，處理的資料存在 quant_image 容器內。

在輸出檔案時，由於不是要採用 BMP 格式定義的壓縮，而是在 8bit data 內模擬影像量化到 2、4、6bit 的效果，因此原先讀入的 Header 可以直接複製使用，接著加上 quant_image 容器內的影像資料，即是輸出檔案。

如表將 input1.bmp 處理後，可以發現這種壓縮方式是壓縮越多 bit，細節越模糊。當僅剩 2bit 時，則色彩會嚴重偏離。

	
原圖左手臂	保留 6bit 左手臂
	
保留 4bit 左手臂	保留 2bit 左手臂

不過其實也可以改成保留 LSB 區段的 bit，效果則是保留細節部分，但會難以識別出圖像內是什麼東西。

III. Scaling

使用 Biliear Interpolation 做 1.5 倍的縮放大小，其方式是先把新圖片的像數點 P`轉換坐標至原圖片坐標系位置 P，並在 P 附近採鄰近 4 點像數點 ABCD，利用 P 與 ABCD4 點的像數點相對位置做內插進似，即可求得新圖片的像數 P`內容。

舉 input1.bmp 圖片為例，原圖寬 x 高為 640x960，放大 1.5 倍時，則為 960x1440。此時，新圖片因解析度變大，需要補充很多新像數內容，填充的方式就是新圖片(x,y)點座標縮小 1.5 倍後，轉移到原圖座標上取鄰近 4 點 $\mathrm{A}\left(\left\lfloor\frac{x}{1.5}\right\rfloor,\left\lfloor\frac{y}{1.5}\right\rfloor\right)$,$\mathrm{B}\left(\left\lfloor\frac{x}{1.5}\right\rfloor+1,\left\lfloor\frac{y}{1.5}\right\rfloor\right)$,$\mathrm{C}\left(\left\lfloor\frac{x}{1.5}\right\rfloor,\left\lfloor\frac{y}{1.5}\right\rfloor+1\right)$,$\mathrm{D}\left(\left\lfloor\frac{x}{1.5}\right\rfloor+1,\left\lfloor\frac{y}{1.5}\right\rfloor+1\right)$，並由此 4 點做內插即可求出。

而縮小圖片時，新圖片為 427x640，因此採樣時則須將新座標乘上 1.5 倍，再到舊圖片上取樣。取樣點為 $\mathrm{A}(\lfloor 1.5x\rfloor,\lfloor 1.5y\rfloor)$,$\mathrm{B}\ (\lfloor 1.5x\rfloor+1,\lfloor 1.5y\rfloor)$, $\mathrm{C}\ (\lfloor 1.5x\rfloor,\lfloor 1.5y\rfloor+1)$,$\mathrm{D}\ (\lfloor 1.5x\rfloor+1,\lfloor 1.5y\rfloor+1)$，並由此 4 點內差求出。

$$P'(x,y)=P(\mathrm{A})\cdot\left(1-\left(\left(\frac{x}{scale}\right)-\left[\frac{x}{scale}\right]\right)\right)\cdot\left(1-\left(\left(\frac{y}{scale}\right)-\left[\frac{y}{scale}\right]\right)\right)$$
$$+P(\mathrm{B})\cdot\left(\left(\frac{x}{scale}\right)-\left[\frac{x}{scale}\right]\right)\cdot\left(1-\left(\left(\frac{y}{scale}\right)-\left[\frac{y}{scale}\right]\right)\right)$$
$$+P(C)\cdot\left(1-\left(\left(\frac{x}{scale}\right)-\left[\frac{x}{scale}\right]\right)\right)\cdot\left(\left(\frac{y}{scale}\right)-\left[\frac{y}{scale}\right]\right)$$
$$+P(D)\cdot\left(\left(\frac{x}{scale}\right)-\left[\frac{x}{scale}\right]\right)\cdot\left(\left(\frac{y}{scale}\right)-\left[\frac{y}{scale}\right]\right)$$

公式、一 Biliear Interpolation 算法，scale 為放大倍數。若為縮小 1.5 倍，則等同 scale=$\frac{1}{1.5}$

上方縮小至 427x640 時，會觸碰到 BMP 格式對於寬度方向的資料限制，一個 row 必須含有 4byte 倍數的資料。而此圖片 BitperPixel=24，一個 Pixel 有 3byte，(3x427)=1281 不為 4 的倍數。因此需要在每個 row 後面填充 0，直到一個 row 的 bit 數量可以達到 4byte 倍數。

IV. Reference

1. 點陣圖(Bitmap)檔案格式
https://crazycat1130.pixnet.net/blog/post/1345538#mark-6-BI_BITFIELDS
2. BMP file format
https://en.wikipedia.org/wiki/BMP_file_format
3. 雙線性差值法 https://blog.csdn.net/weixin_44638957/article/details/104501316