Homework 1 (30%)

姓名：楊哲旻
學號：413511003
學系：電機工程學系博士一年級
日期：2024.10.03

1. Image input/flip/output. Explain BMP format in most 2 pages (A4). [Code Demo Video]

BMP(Bitmap) [1] 文件主要用於 Windows 系統，它以設備無關的方式存儲數字圖像，稱為 DIB（Device-Independent Bitmap），因此它可在不同設備上保持一致的顯示效果。由於我是用 MAC 筆電，並無「#include <windows.h>」，但可自行定義建構，如圖 1 至圖 3。下列詳細介紹 BMP 格式結構與優缺點:

(1) BMP 文件的結構：由文件頭(File Header)、資訊頭(Information Header)、調色板(Color Table) 和 像素數據(Pixel Data)組成:

A. 文件頭 (BITMAPFILEHEADER): 提供關於 BMP 檔案的基本資訊，共 14 位元組，比如：文件大小和像素數據的起始位置等。

欄位	大小(位元組)	說明
bfType	2	文件類型標識 ('BM' 表示 BMP 文件)
bfSize	4	文件大小，以位元組為單位
bfReserved1, 2	2, 2	保留欄位，必須為 0
bfOffBits	4	從文件開頭到像素數據開始的偏移量

```
// 定義 BITMAPFILEHEADER 結構
#pragma pack(1)
typedef struct {
    uint16_t bfType;      // 檔案類型，必須為 'BM' (0x4D42)
    uint32_t bfSize;      // 檔案大小（位元組）
    uint16_t bfReserved1; // 保留，必須為 0
    uint16_t bfReserved2; // 保留，必須為 0
    uint32_t bfOffBits;   // 從檔案頭到實際像素數據的偏移量（位元組）
} BITMAPFILEHEADER;
```

圖 1、BITMAPFILEHEADER

B. 資訊頭 (BITMAPINFOHEADER): 描述了圖像本身的資訊，如寬度、高度、顏色深度等。

欄位	大小(位元組)	說明
biSize	4	資訊頭的大小
biWidth	4	圖像的寬度(像素數)
biHeight	4	圖像的高度(像素數)
biPlanes	2	顏色平面數，必須為 1
biBitCount	2	每像素的位元數 (1, 4, 8, 16, 24 或 32)
biCompression	4	壓縮類型(0 表示不壓縮)
biSizeImage	4	圖像數據的大小（以位元組為單位）
biXPelsPerMeter	4	水平解析度，像素/米
biYPelsPerMeter	4	垂直解析度，像素/米
biClrUsed	4	圖像使用的顏色數量
biClrImportant	4	重要顏色數量（通常設置為 0）

```
// 定義 BITMAPINFOHEADER 結構
typedef struct {
    uint32_t biSize;          // 資訊頭大小（位元組）
    int32_t biWidth;          // 圖片寬度（像素）
    int32_t biHeight;         // 圖片高度（像素）
    uint16_t biPlanes;        // 顏色平面數，必須為 1
    uint16_t biBitCount;      // 每個像素的位元數
    uint32_t biCompression;   // 壓縮類型（0 = 不壓縮）
    uint32_t biSizeImage;     // 圖片大小（位元組）
    int32_t biXPelsPerMeter;  // 水平解析度
    int32_t biYPelsPerMeter;  // 垂直解析度
    uint32_t biClrUsed;       // 使用的顏色數
    uint32_t biClrImportant;  // 重要顏色數
} BITMAPINFOHEADER;
```

圖 2、BITMAPINFOHEADER

C. 調色板 (Color Table): 可選，如果圖像的顏色深度低於或等於 8 位元（如 1 位元或 4 位元），則會有一個調色板來定義每個像素的顏色。調色板是由顏色表組成的，每個顏色表包含藍色、綠色、紅色（和透明度）分量。

欄位	大小 (位元組)	說明
rgbtBlue	1	調色板中的藍色分量
rgbtGreen	1	調色板中的綠色分量
rgbtRed	1	調色板中的紅色分量
rgbtReserved	1	保留欄位，通常設置為 0

```
// 定義 RGBTRIPLE 結構
typedef struct {
    uint8_t rgbtBlue;
    uint8_t rgbtGreen;
    uint8_t rgbtRed;
} RGBTRIPLE;
```

(1) RGB

```
// 定義 RGBA 結構
typedef struct {
    uint8_t rgbtBlue;
    uint8_t rgbtGreen;
    uint8_t rgbtRed;
    uint8_t rgbtAlpha;
} RGBA;
```

(2) RGBA

圖 3、Color

D. 像素數據: 為 BMP 文件中最重要的部分，存儲每個像素的顏色值。像素數據的排列順序是從下到上、從左到右，這代表圖像的底部像素最先存儲，最頂部的像素最後存儲，但也有其他方向儲存方法，如 **備註(一)**。像素數據的存儲方式取決於圖像的顏色深度:

- **24 位元（RGB）:** 每個像素使用 3 個位元組，分別存儲 紅、綠、藍 三個通道的顏色值。
- **32 位元（RGBA）:** 每個像素使用 4 個位元組，包含 紅、綠、藍 和 透明度 (Alpha 通道)的資訊。

- 壓縮格式：在使用壓縮時，像素數據會根據壓縮算法存儲。

為了提高處理效率，BMP 文件中每一行的像素數據必須是 4 的倍數。如果一行的像素數據不足 4 的倍數，則會在行末加入填充位元組（Padding Bytes），這些位元組不包含任何圖像資訊，被稱為位元組對齊。

(2) BMP 文件的優缺點:

- 優點：BMP 格式結構簡單，容易讀取和編寫，適合低層次的圖像處理。因為通常是未壓縮格式，BMP 圖像質量無損。
- 缺點：未壓縮的 BMP 文件體積非常大，對於存儲和傳輸不夠高效。BMP 文件格式因其簡單和普及性在早期的 Windows 系統中非常常用，但隨著圖像壓縮技術的發展，它逐漸被 PNG、JPEG 等更高效的格式取代。

備註(一)：儲存方向 [2]

若 biHeight 是正數，表示圖片高度為正，像素數據從圖像的底部行開始存儲，依次向上存儲，這是 BMP 格式的傳統存儲方式，即 **Bottom-up Bitmap** 如圖 4(A)。若 biHeight 是負數，表示圖片高度為負，像素數據從圖像的頂部行開始存儲，依次向下存儲，即 **Top-down Bitmap**，如圖 4(B)。由於目前 BMP 文件以 Bottom-up 儲存之影像居多，因此我的程式碼只針對圖 4(A)處理，而圖 4(B)我僅有判斷是打印出「Top-down BMP images are not supported.」，而停止後續的水平翻轉程式碼，如圖 5；反之，確定為 Bottom-up 會繼續完成水平翻轉程式碼，分別為 RGBA 與 RGB，如圖 6。

(A) Bottom-up Bitmap　　(B) Top-down Bitmap

圖 4、BMP 儲存方向[2]

```
83      // 檢查 biHeight 是否為負數 (Top-down BMP)
84      if (infoHeader.biHeight < 0) {
85          printf("Top-down BMP images are not supported.\n");
86          fclose(fp_in);
87          fclose(fp_out);
88          return 1;
89      }
```

圖 5、避免 Top-down Bitmap 所停止後續水平翻轉之程式碼

```
123     // 假如不是RGBA，即是RGB
124     if (!isRGBA) {
125         for (int i = 0; i < H; i++) {
126             fread(rowBuffer, rowSize, 1, fp_in);  // 讀取每行像素數據，包括填充位元組
127
128             // 水平翻轉像素
129             for (int j = 0; j < W; j++) {
130                 RGBTRIPLE rgb = *((RGBTRIPLE*)&rowBuffer[j * sizeof(RGBTRIPLE)]);
131                 color[i][W - j - 1] = rgb;  // 水平翻轉
132             }
133         }
134
135         fwrite(&fileHeader, sizeof(BITMAPFILEHEADER), 1, fp_out);
136         fwrite(&infoHeader, sizeof(BITMAPINFOHEADER), 1, fp_out);
137
138         // 寫入翻轉後的每行像素數據
139         for (int i = 0; i < H; i++) {
140             for (int j = 0; j < W; j++) {
141                 RGBTRIPLE rgb = color[i][j];
142                 memcpy(&rowBuffer[j * sizeof(RGBTRIPLE)], &rgb, sizeof(RGBTRIPLE));
143             }
144             fwrite(rowBuffer, rowSize, 1, fp_out);  // 寫入每行數據，包括填充位元組
145         }
```

(A) RGB

```
146     // 假如是RGBA
147     } else {
148         for (int i = 0; i < H; i++) {
149             fread(rowBuffer, rowSize, 1, fp_in);  // 讀取每行像素數據，包括填充位元組
150
151             // 水平翻轉像素
152             for (int j = 0; j < W; j++) {
153                 RGBA rgba = *((RGBA*)&rowBuffer[j * sizeof(RGBA)]);
154                 color_4[i][W - j - 1] = rgba;  // 水平翻轉
155             }
156         }
157
158         fwrite(&fileHeader, sizeof(BITMAPFILEHEADER), 1, fp_out);
159         fwrite(&infoHeader, sizeof(BITMAPINFOHEADER), 1, fp_out);
160
161         // 寫入翻轉後的每行像素數據
162         for (int i = 0; i < H; i++) {
163             for (int j = 0; j < W; j++) {
164                 RGBA rgba = color_4[i][j];
165                 memcpy(&rowBuffer[j * sizeof(RGBA)], &rgba, sizeof(RGBA));
166             }
167             fwrite(rowBuffer, rowSize, 1, fp_out);  // 寫入每行數據，包括填充位元組
168         }
169     }
```

(B) RGBA

圖 6、水平翻轉

參考文獻

[1] Wikipedia BMP file format https://en.wikipedia.org/wiki/BMP_file_format

[2] Top-Down vs. Bottom-Up DIBs https://learn.microsoft.com/en-us/windows/win32/directshow/top-down-vs--bottom-up-dibs

[3] Homework 1-1 Code Demo Video https://youtu.be/IPj_cEYeIMs

2. Resolution. Do some discussion and explain how you do it in most 1 page (A4). [Code Demo Video]

(1) 量化[4] 的數學簡單舉例:

如果有一個 8 位元的像素值範圍為 0 到 255。我們希望將其量化為 4 位元 (16 級別)，可以按照以下步驟進行量化:

A. **量化級別計算:** 量化級別數 $= 2^n$ ，其中 n 是目標位元數。例如 4 位元量化: $2^4 = 16$ 級別。

B. 計算步長(Step Size): 每個量化級別的範圍 $= 256/16 = 16$。

C. 量化步驟: 將原值除以量化級別範圍，取整後再乘以步長。

範例 1，原像素值為 65，所以 $65 \div 16 = 4.06$，取整後為 4。而 $4 \times 16 = 64$，這就是量化後的像素值。

範例 2: 原像素值為 130，所以 $130 \div 16 = 8.125$，取整後為 8。而 $8 \times 16 = 128$ 為量化後的值。

■ 彩色 RGB 量化範例: 原始 RGB 值為 (120, 200, 75)，則量化為 4 位元:

$120 \div 16$ 取整為 7，$7 \times 16 = 112$，量化後的紅色值為 112;

$200 \div 16$ 取整為 12，$12 \times 16 = 192$，量化後的綠色值為 192;

$75 \div 16$ 取整為 4，$4 \times 16 = 64$，量化後的藍色值為 64。

因此，量化後的 RGB 值變為 (112, 192, 64)。

圖 7 為原始值與量化後的值之映射圖，可見如果量化位元數越低，則會與原先數值可能差異越大，但可以減少儲存的位元數，如圖 8。

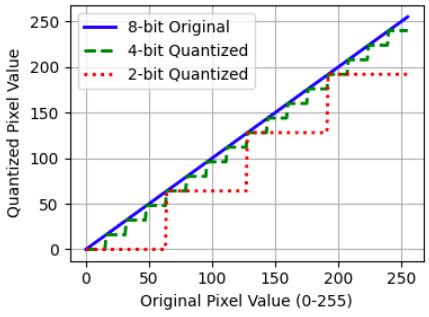

圖 7、量化前後的像素數值

圖 8、量化結果(由左而右為 2, 4, 6 位元)

(2) 量化的程式碼講解:

從上述數學可以得到量化公式(1) 為:

$$Quantized\ Value = \left(\left\lfloor \frac{Original\ Value}{Step\ Size} \right\rfloor\right) \times StepSize \qquad (1)$$

其中 $\lfloor \cdot \rfloor$ 為向下取整數。

因此，我的程式碼建立一函數為 quantizeColorRGB (與 quantizeColorRGBA) 根據指定的量化位元數來計算步長 step ，再使用公式將 RGB 的每個通道之像素進行量化計算，如圖 8。

```
45  // 量化 RGB 顏色
46  void quantizeColorRGB(RGBTRIPLE *pixel, int bits) {
47      int levels = 1 << bits;  // 計算級別數，例：6 位元有 64 級
48      int step = 256 / levels; // 每個級別的步長
49
50      // 量化每個通道
51      pixel->rgbtBlue = (pixel->rgbtBlue / step) * step;
52      pixel->rgbtGreen = (pixel->rgbtGreen / step) * step;
53      pixel->rgbtRed = (pixel->rgbtRed / step) * step;
54  }
```

圖 8、量化函數（以 RGB 為例）

為了得到作業的一次生成 6, 4 與 2 位元數的量化結果，使用迴圈 for (int bits = 6; bits >= 2; bits -= 2)進行不同級別的三次量化並儲存。

參考文獻

[4] Wikipedia Quantization (signal processing) https://en.wikipedia.org/wiki/Quantization_(signal_processing)

[5] Homework 1-2 Code Demo Video https://youtu.be/VD-jGSpTuwY

3. Cropping. Explain how you do it in most 1 page (A4). [Code Demo Video]

BMP 通常為 Bottom-up 以存儲像素數據，順序是從左下角到右上角。我的程式碼提供使用者輸入裁減位置(x, y, w, h) 對影像進行裁減，如圖 9。

```
118     // 輸入裁剪區域
119     printf("Enter cropping region (startX startY cropWidth cropHeight): ");
120     scanf("%d %d %d %d", &startX, &startY, &cropWidth, &cropHeight);
```

圖 9、使用者輸入裁減位置

裁減位置如果在影像中是超出範圍的，則不會進行裁減，並打印 Invalid cropping region. ，如圖 10。

```
63  // 檢查裁剪區域是否有效
64  if (startX < 0 || startY < 0 || startX + cropWidth > originalWidth || startY + cropHeight > originalHeight) {
65      printf("Invalid cropping region.\n");
66      fclose(fp_in);
67      return;
68  }
```

圖 10、避免裁減範圍超出原圖

循環讀取裁減區域的每一行像素：程式會從 startY 行開始，逐行讀取 cropWidth 寬度的像素數據，並將其寫入到新文件中。

對於 RGB 圖片，程式使用 RGBTRIPLE 結構來處理每行像素；對於 RGBA 圖片，程式使用 RGBA 結構來處理每行像素，如圖 11。

```
94  // 逐行讀取裁剪區域的像素並寫入輸出文件
95  for (int i = 0; i < cropHeight; i++) {
96      if (isRGBA) {
97          RGBA* row = (RGBA*)malloc(cropWidth * sizeof(RGBA));
98          fread(row, sizeof(RGBA), cropWidth, fp_in);
99          fwrite(row, sizeof(RGBA), cropWidth, fp_out);
100         free(row);  // 釋放記憶體
101     } else {
102         RGBTRIPLE* row = (RGBTRIPLE*)malloc(cropWidth * sizeof(RGBTRIPLE));
103         fread(row, sizeof(RGBTRIPLE), cropWidth, fp_in);
104         fwrite(row, sizeof(RGBTRIPLE), cropWidth, fp_out);
105         free(row);  // 釋放記憶體
106     }
107     fseek(fp_in, rowSize - cropWidth * (infoHeader.biBitCount / 8), SEEK_CUR); // 跳過行尾的填充位元組
108 }
```

圖 11、

使用者輸入 400 100 200 200，則會裁減成為圖 11；而輸入 100 100 200 200，則會裁減成為圖 12。

圖 11、輸入裁減為 400 100 200 200

圖 12、輸入裁減為 100 100 200 200

參考文獻

[6] Homework 1-3 Code Demo Video https://youtu.be/NXxEbTmbpl0