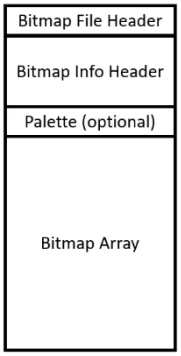
Digital Image Processing (2024)　Hw1 Report

電子碩二 312510198 黃育豪



1. Image input/flip/output

- **BMP 格式**

  如圖所示 BMP 的格式主要有 4 個區塊來組成,分別為

1. Bmp File Header

這個區塊的大小總合為 14 Bytes 依序代表的資訊如下表格

名稱	大小(Bytes)	功能
bfType	2	顯示檔案類型,0x4D42 在 ASCII 對應到 BM
bfSize	4	顯示此 BMP 檔的大小
bfReserved1	2	保留位給圖片渲染使用,初始為 0
bfReserved2	2	保留位給圖片渲染使用,初始為 0
bfOffBits	4	用來表示從開頭到 Bmp Array 的偏移,會在 Palette 時做驗證

2. Bmp Info Header

這個區塊的大小總合為 40 Bytes 依序代表的資訊如下表格

名稱	大小(Bytes)	功能
biSize	4	表示 InfoHeader 的大小
biWidth	4	圖片的寬度(單位:像素)
biHeight	4	圖片的高度(單位:像素)
biPlanes	2	向目標設備說明顏色平面數,設為 1
biBitCount	2	說明每個像素用幾個 bits 表示色彩,這次作業是 24 或 32bits
biCompression	4	圖像的壓縮類型,常用 0(BI_RGB)表示不壓縮
biSizeImages	4	BitArray 的大小,使用 BI_RGB 時可以設為 0
biXPelsPerMeter	4	水平分辨率
biYPelsPerMeter	4	垂直分辨率
biClrUsed	4	使用 Palette 中的顏色索引數,0 為使用所有
biClrImportant	4	對成像有重要影響的顏色索引數,為 0 說明都重要

3. Palette

當 biBitCount<=8 時,0~255 無法表示所有顏色,因此需要使用調色盤和索引來

對照色彩,如 biBitcount=8 就有 256 個顏色,此時調色盤大小就會是 256*4(RGB+Alpha)=1024 Bytes,所以 14+40+1024=1078 Bytes 與 bfOffBits 相符

4. Bmp Array

存放圖像中每個像素的顏色值,每列須為 4Bytes 的倍數

- **Flip 做法**

先從標頭檔中讀取出圖片的寬高和位元深度

```
fread(&fileHeader, sizeof(BITMAPFILEHEADER), 1, fp_in);
fread(&infoHeader, sizeof(BITMAPINFOHEADER), 1, fp_in);

unsigned int Width = infoHeader.biWidth;
unsigned int Height = infoHeader.biHeight;
unsigned int BitDepth = infoHeader.biBitCount;
```

使用 Raster Scan 的方式每次讀取 3Bytes(RGB)放入建立好的原始像素 Memory

```
RGBTRIPLE rgb; // 要放從fp_in讀出來的rgb
RGBTRIPLE(*rgb_Vector)
[Width] = calloc(Height, Width * sizeof(RGBTRIPLE)); // 創造一個放原始像素的記憶體

for (i = 0; i < Height; i++)
{

    for (j = 0; j < Width; j++)
    {

        // 把讀到的rgb放入記憶體
        fread(&rgb, sizeof(RGBTRIPLE), 1, fp_in);

        rgb_Vector[i][j].rgbtBlue = rgb.rgbtBlue;
        rgb_Vector[i][j].rgbtGreen = rgb.rgbtGreen;
        rgb_Vector[i][j].rgbtRed = rgb.rgbtRed;
    }
}
```

先對新 bmp 檔的標頭檔進行寫入,並且把剛剛存好的原始像素一樣從上到下但左右順序顛倒的把像素依序從右到左寫入輸出圖檔以達到左右翻轉的效果

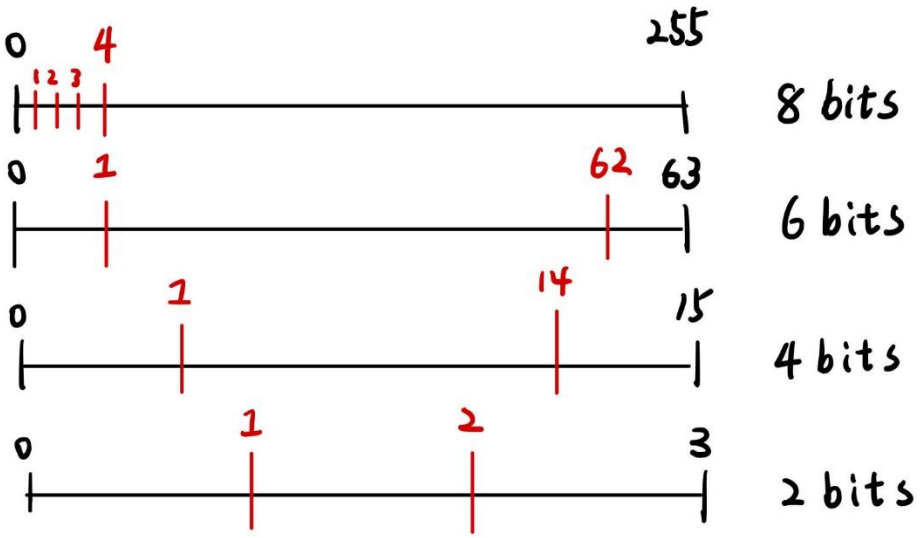
```
// 要先寫Header不然圖片會開不起來
fwrite(&fileHeader, sizeof(BITMAPFILEHEADER), 1, fp_out);
fwrite(&infoHeader, sizeof(BITMAPINFOHEADER), 1, fp_out);

for (i = 0; i < Height; i++)
{

    for (j = Width-1; j >= 0; j--)
    {
        // 把剛剛存的像素從記憶體拿出來,並寫入fp_out
        rgb.rgbtBlue = rgb_Vector[i][j].rgbtBlue;
        rgb.rgbtGreen = rgb_Vector[i][j].rgbtGreen;
        rgb.rgbtRed = rgb_Vector[i][j].rgbtRed;
        fwrite(&rgb, sizeof(RGBTRIPLE), 1, fp_out);
    }
}
```

這裡的做法適用於 Input1 這種 24Bits 的圖像,像 input2 是 32Bits 的話因為多了 Alpha 項,所以需要建立一個新的 Struct 如下圖

```
typedef struct
{
    unsigned char rgb_Blue;
    unsigned char rgb_Green;
    unsigned char rgb_Red;
    unsigned char rgb_Alpha;
} RGB_32; // 多了Alpha透明度,所以變成4Bytes
```

2. Resolution

解析度原本是 24bits 代表三原色 RGB,也就是每個顏色量化成用 8bits 來表示,如此一來每個顏色會有 256 種差別,而若是將 8bits 分別量化成 6bits 4bits 2bits 會如何?將以下圖的 6bits 為例,原本用 8bits 表示的顏色當使用 6bits 表示時就代表一個顏色變成 0~63 總共 64 種差別而已,原本的 0,1,2,3 都是不同的顏色,但在 6bits 下都變成 0 這個區間的顏色,也因此解析度會變差,同樣 2bits 的話就代表將顏色分成四種變化 0,1,2,3,原本的 0~63 總共 64 種顏色在這個表示下都變成了同一種顏色也就是 0,也因此 2bits 的解析度會最差

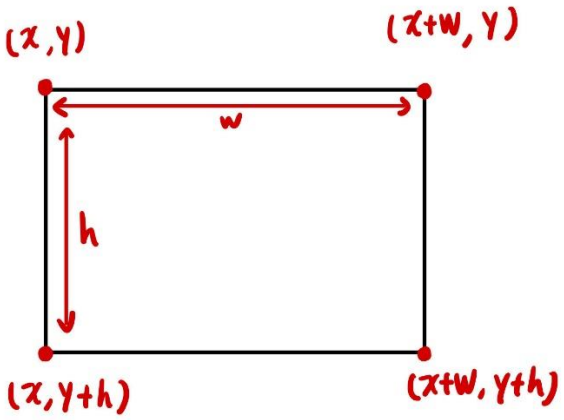


這裡使用 C 的整數除法進行 code 實現,把原本表示 RGB 的數值除以 4(6bits) 16(8bits),64(2bits)後因為浮點數被無條件捨去了,因此再還原回 8bits 時就會落在量化區監裡並且失去了低位的資訊,若是希望減少量化的誤差可以設置代表位階,以 2bits 為例,在 8bits 時 0~63 轉換完都會是 0,但如果原始值是 63 的話誤差會很大,因此可以改成如果落在 0~63 區間的話就使用中間值當代表位階 31 表示

```
for (i = 0; i < Height; i++)
{
    for (j = 0; j < Width; j++)
    {
        INT_Blue = rgb_Vector[i][j].rgbtBlue / 4;
        INT_Green = rgb_Vector[i][j].rgbtGreen / 4;
        INT_Red = rgb_Vector[i][j].rgbtRed / 4;
        // 把剛剛存的像素從記憶體拿出來,並寫入fp_out
        rgb.rgbtBlue =INT_Blue*4;
        rgb.rgbtGreen = INT_Green*4;
        rgb.rgbtRed = INT_Red*4;
        fwrite(&rgb, sizeof(RGBTRIPLE), 1, fp_out1);
    }
}
```

3. Cropping

這題為從使用者定義的座標開始進行指定寬度和高度的裁切，我的做法是一樣透過 Raster Scan 的方式將輸入檔案 input1.bmp 的 RGB 資訊讀進來，而當讀到我們要的像素(x,y)後就開始把 data 存到記憶體裡面，一直存直到超出我們需要的範圍，記得要先修改 infoHeader 裡面的寬度和高度，之後再把記憶體中的 data 依序寫入輸出圖檔 output1_crop.bmp




```
infoHeader.biWidth = w;
infoHeader.biHeight = h;

// 要先寫Header不然圖片會開不起來
fwrite(&fileHeader, sizeof(BITMAPFILEHEADER), 1, fp_out);
fwrite(&infoHeader, sizeof(BITMAPINFOHEADER), 1, fp_out);
```

```
for (i = 0; i < Height; i++)
{
    for (j = 0; j < Width; j++)
    {
        // 把讀到的rgb放入記憶體
        fread(&rgb, sizeof(RGBTRIPLE), 1, fp_in);
        if (i >= y && i < y + h && j >= x && j < x + w)
        {
            rgb_Vector[i - y][j - x].rgbtBlue = rgb.rgbtBlue;
            rgb_Vector[i - y][j - x].rgbtGreen = rgb.rgbtGreen;
            rgb_Vector[i - y][j - x].rgbtRed = rgb.rgbtRed;
        }
    }
}
```

參考資料

https://blog.csdn.net/imxlw00/article/details/124926918

https://blog.csdn.net/qq_39400113/article/details/104750460

https://blog.csdn.net/BigDream123/article/details/102536571

https://blog.csdn.net/m0_50847352/article/details/124972859

https://blog.csdn.net/u012877472/article/details/50272771

https://geocld.github.io/2021/03/02/bmp/