國立陽明交通大學電子研究所 311510187 呂紹愷

DIP HW1

- **Image input/flip/output**

I. Input

本次圖像的檔案格式為 bmp (bitmap)，整個檔案由四個部分組成，Bitmap File Header, Bitmap Info Header, Color Table, Bitmap Array。

■ Bitmap File Header

Bitmap File Header BITMAPFILEREADER		
Signature	2 Bytes	12 Bytes
File Size	4 Bytes	
Reserved	2 Bytes	
Data Offset	4 Bytes	

主要是有 4 區，第 1 區 signature(ID)，用於指定 BMP 文件類型的 ASCII 碼，此值需為 BM。

第 2 區 file size，整個 bitmap file 的大小。

第 3 區 Reserved，用於圖像處理軟體添加有意義的訊息，初始值為 0。

第 4 區 Data Offset，bitmap data 開始的位置。

■ Bitmap Info Header

Bitmap Info Header BITMAPINFOREADER		
Header Size	4 Bytes	40 Bytes
Width	4 Bytes	
Height	4 Bytes	
Color Planes	2 Bytes	
Bit per Pixel	2 Bytes	
Compression	4 Bytes	
Size Image	4 Bytes	
X Pels Per Meter	4 Bytes	
Y Pels Per Meter	4 Bytes	
Clr Used	4 Bytes	
Clr Important	4 Bytes	

此 header 用來表示一些特性，像是尺寸、bit count，如此才知道影像的尺寸以及 RGB or RGBA。

- Palette
  每個索引值表示一個顏色：0x00RRGGBB，最高位保留 0
- Bitmap Array
  用來存放 data，而特別要注意的地方是每個列需為 4bytes 的倍數。

了解了 bmp 的一些格式後，確認完 size，創好 memory，檔名設定好，就可以開始讀檔，將圖像 input 進來。

```
for(i=0; i<H; i++ ) {
    for( j=0; j<W; j++  ) {
        fread(&rgb, sizeof(RGBTRIPLE), 1, fp_in);
        color[i][j].rgbtBlue=rgb.rgbtBlue;
        color[i][j].rgbtGreen=rgb.rgbtGreen;
        color[i][j].rgbtRed=rgb.rgbtRed;
    }
}
fclose(fp_in);      //檔案fp讀取完成，關閉
```

II. Flip

Flip 的作法很簡單，為水平翻轉，也就是整個影像的左、右交換，因此以每個 row 為單位，將此 row 的左右交換。

```
for(int i=0; i<H; i++ )
{
    for(int j=0; j < W; j++ )
    {
        rgb.rgbtBlue = color[i][W - j - 1].rgbtBlue;
        rgb.rgbtGreen = color[i][W - j - 1].rgbtGreen;
        rgb.rgbtRed = color[i][W - j - 1].rgbtRed;

        fwrite(&rgb, sizeof(RGBTRIPLE), 1, fp_out);        // 將 color matrix 寫入輸出圖檔中
    }
}
fclose(fp_out);
```

III. Output

當我們處理完影像後，即可開始輸出，而 bmp 有可能有 3 層(RGB)或者是 4 層(RGBA)，因此在處理這兩種形式的方式也不同，不過基本上是大同小異，僅是 RGBA 要多處理一層而已。輸出時要特別注意 size 是否有變，像是在後面的縮放，一開始沒注意到要更改尺寸，圖像就跑不出來了。

- **Resolution**

  Resolution 為解析度，又稱為分辨率，解析度越好則代表影像的品質越好，可以顯示出更多細節。

  在 RGB 中每個 pixel 為 3*8 bits，也就是數值是 0~255，可以使用 256 個數字來做訊號強烈的表達，而現在要使用 6bits (0~63), 4bits(0~15), 2bits(0~3) 來做表達，也就是說原本能使用 256 個數字，現在只能使用 64 個數字, 16 個數字, 4 個數字來表達。

  以 2bits 為例，我們只能使用 4 個數字來表達，將 256 個數字分成 4 塊，分別是 0~63, 64~127, 127~191, 192~255，然後將其映射至各區間的中間，0~63 => 31, 64~127=> 95, 127~191 => 159, 192~255 =>223，只使用這 4 個數字來表達，其他 bits 以此類推。也因如此，整個檔案 size 沒有改變，因為我們仍用了 3*8bits 來做表達。

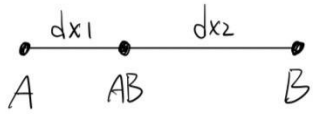
```
int quant_resolution(int a,int b){ //a = input , b = bits
    int o;
    if(b == 6) // bit = 6
        o = ((a>>2)<<2) + 1 ;
    else if(b == 4) // bit = 4
        o= ((a>>4)<<4) + 7 ;
    else // bit = 2
        o= ((a>>6)<<6) + 31;
    return o;
}
```

- Scaling

  我們將原圖進行 1.5 倍的放大及縮小，由於是要產生另一張不同尺寸的圖像，因此會出現原本沒有的和位置，這時候我們就需要使用內插，去補這些原本沒有的點。

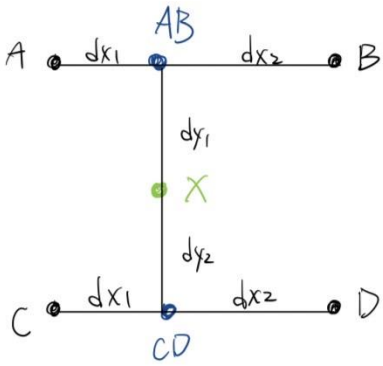
  在這題中使用的是 Bilinear Interpolation，要理解這個演算法可以從 1D - linear 開始，依照 AB 與 A 和 B 的距離去分配 AB 的位置，而 Bilinear 則是做三次，先求出 AB 以及 CD，就可以算出 X，再將此過程用 code 來實現。

  1D

  $AB = A dx_2 + B dx_1$

  2D

  $AB = A dx_2 + B dx_1$

  $CD = C dx_2 + D dx_1$

  $X = AB dy_2 + CD dy_1$

  需要特別注意的是在最邊界的點，要注意不會跑出圖片，也就是為和在 code 中需要-1，防止超出邊界。