Homework1

Image input/output

Bmp(Bitmap):

Bmp 檔分為四個部分：

1. Bitmap File Header
2. Bitmap Info Header
3. Color Table (Palette)
4. Bitmap Array

不同顏色代表較為重要的值，其他設定錯誤好像不會造成太大影響。

Bitmap File Header:

Identifier	File Size	Reserved	Bitmap Data Offset
2 bytes	4 bytes	4 bytes	4 bytes

Identifier: 大部分都是存著"BM"的 ASCII code，

File Size: 存著整個檔案的 bytes 數，

Reserved: 保留於是設 0，

Bitmap Data Offset: 記錄著圖片檔案從哪裡開始儲存。

Bitmap Info Header:

Info header size	Width	Height	Planes
4 bytes	4 bytes	4 bytes	2 bytes
Bits per pixel	Compression	Bitmap data size	H-Resolution
2 bytes	4 bytes	4 bytes	4 bytes
V-Resolution	Used Colors	Important Colors	
4 bytes	4 bytes	4 bytes	

Info header size: Info header 的 byte 數 。

Width: 寬度是幾個 pixels。很重要的一點是 DWORD-aligned 的問題，若是 (width*bits per pixel / 8) % 4 !=0，則需自己補 0，使之成為 4 的倍數。

Height: 高度是幾個 pixels。

Planes: 點陣圖的位元圖層數(不知道實際功能為何)。

Bits per pixel: 每個 pixel 使用幾個 bits 來存，8 個 bits 為一個 byte。(檔案採取 little-endian 的方式，若是為 4bits，則一個 byte 裡存兩個 pixel)

Compression: 此檔案是否有壓縮並且是為何種方式。

Bitmap Data Size: 圖片總共的 byte 數。

H-Resolution: 水平解析度（單位：像素/公尺）

V-Resolution: 垂直解析度（單位：像素/公尺）

Used Colors: 調色盤的顏色數。若為 0 則代表使用全部顏色，16bits 以下一定得使用調色盤。

Important Colors: 重要的顏色數。

Palette:

大小為 N*4 bytes。若是 bits 數在 16 以下則一定需要使用，其數量由 used color 設定，4 bytes 分別為 BGRA。

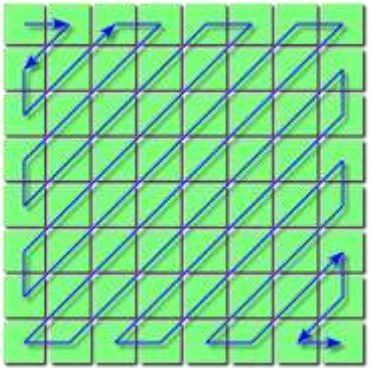
Bitmap Array:

其儲存方式由 bits per pixel 數來決定，唯一的要求就是照片的每個橫排掃描完後，必須是 4 的倍數，否則無法開啟。

參考資料：瘋小貓的華麗冒險、靖 技場

JPEG(Joint Photographic Experts Group):

此種檔案是使用 YUV 的色彩空間，Y 代表亮度；U 代表色調；V 代表飽和度。因為人眼對於亮度差異敏感度高於色彩變化，於是在這種色彩空間就能夠簡易的調整圖片的色彩了。



對於 YUV 三個區域，對每個區域切成 8*8 的矩陣後，經平移，使用 DCT(Discrete Cosine Transform)將圖片從實域空間轉換到頻域空間，再將其除上一個量化矩陣後就是我們壓縮後的檔案了。

再來使用 Z 字型儲存檔案，因為左上角為高頻區、右下角為高頻區，於是以此方法可以照頻率排列。若在某一個時刻之後都是 0，則使用 EOB 來代表結束，可節省很多儲存空間。

Name	Comments
Start Of Image	
Start Of Frame (baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
Start Of Frame (progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
Define Huffman Table(s)	Specifies one or more Huffman tables.
Define Quantization Table(s)	Specifies one or more quantization tables.
Define Restart Interval	Specifies the interval between RST*n* markers, in Minimum Coded Units (MCUs). This marker is followed by two bytes indicating the fixed size so it can be treated like any other variable size segment.
Start Of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low three bits of the marker code cycle in value from 0 to 7.
Application-specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF.
Comment	Contains a text comment.
End Of Image	

此圖片為 JPEG 的檔案儲存方式

參考資料：維基百科

Resolution

Quantization，以訊號與系統的角度變是將資料以不同的 bits 數來儲存，使 pixel 能展現的顏色範圍不同，造成圖片的精細度有不同的結果。

Approach 1:

因為 pixel 的 bits 不同，於是能展現出的範圍不同，最直觀也最正確的方法應該是改變 Bmp 檔中的 bits per pixel 值和 raw data 的儲存方式，再藉由調色盤來代表每一個數字對應的顏色等等。但這方法實在太難以 debug，因為相片編輯器只會顯示無法使用或格式不符的訊息，而每一種 bits 所對應的程式碼很難相同，除非寫的非常客製化，但那又須考量太多因素無法在短時間完成。於是在完成 bits per pixel 為 8bits 和 4bits 後無法實現 2bits 的資料轉換便放棄。

Approach 2 (Final Approach):

直接算出轉換後能夠出現什麼樣的數值，以$\text{int}\left(\frac{\text{color}}{256/\text{bits}}\right)*\frac{256}{bits}$ 公式轉換後，直接以原本的 bits 數輸出。

Result:

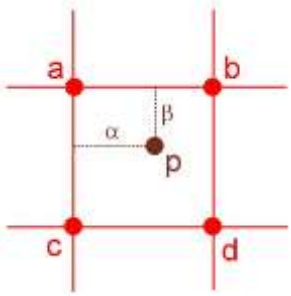
因為能展現的顏色數量不同，調色盤越少顏色會使得圖片顏色的間隔越大，也就是邊緣感越明顯，除此之外，因為每個顏色的使用範圍變廣，會使畫面產生一片一片的同色群。也就是精細度降低。

Recommendation：

希望助教對題目的敘述和要求能講清楚一點，因為此題最正確做法的應該是 approach 1 的方法，但因為其極難做，實在是花了我一半以上的時間去嘗試它。若是題目不需做得這麼精細的話，我希望能夠有點線索，多一點對 input 和 output 的敘述和要求，謝謝助教。

Scaling

Bilinear interpolation:



F(p)=(1-α)*(1-β)*f(a)+α*(1-β)*f(b)+(1-α)*β*f(c)+α*β*f(d)

而內插的每一個座標距離假設為 xstep 跟 ystep，其算式為

$$xstep = \frac{(\text{Original x}-\text{pixels})-1}{(\text{Scaling x}-\text{pixels})-1},\ ystep = \frac{(\text{Original y}-\text{pixels})-1}{(\text{Scaling y}-\text{pixels})-1}$$

以此算出變換後每一個 pixel 的位置，找出離這點最近的四個點做內插法，算出其值。唯一要注意的是在邊邊角角的地方很容易發生溢位的問題，要特別注意。

這種方法的優點是比 NNI (Nearest Neighbor) 做出來的影像來的精緻，較不會出現鋸齒狀的邊緣。

Bicubic interpolation:

$$p(x,y) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}x^i y^j$$

同樣是找出變換後的每一個 pixel 位置，找出離其最近的四個點，但做的是較為複雜的內插計算。此方程式能因加上偏微、微分後產生 16 個方程式。假設附近的四個點為 (0,0)、(1,0)、(0,1)、(1,1)。

1. $f(0,0) = p(0,0) = a_{00}$,
2. $f(1,0) = p(1,0) = a_{00} + a_{10} + a_{20} + a_{30}$,
3. $f(0,1) = p(0,1) = a_{00} + a_{01} + a_{02} + a_{03}$,
4. $f(1,1) = p(1,1) = \sum_{i=0}^{3}\sum_{j=0}^{3} a_{ij}$.

1. $f_x(0,0) = p_x(0,0) = a_{10}$,
2. $f_x(1,0) = p_x(1,0) = a_{10} + 2a_{20} + 3a_{30}$,
3. $f_x(0,1) = p_x(0,1) = a_{10} + a_{11} + a_{12} + a_{13}$,
4. $f_x(1,1) = p_x(1,1) = \sum_{i=1}^{3}\sum_{j=0}^{3} a_{ij}i$,
5. $f_y(0,0) = p_y(0,0) = a_{01}$,
6. $f_y(1,0) = p_y(1,0) = a_{01} + a_{11} + a_{21} + a_{31}$,
7. $f_y(0,1) = p_y(0,1) = a_{01} + 2a_{02} + 3a_{03}$,
8. $f_y(1,1) = p_y(1,1) = \sum_{i=0}^{3}\sum_{j=1}^{3} a_{ij}j$.

1. $f_{xy}(0,0) = p_{xy}(0,0) = a_{11}$,
2. $f_{xy}(1,0) = p_{xy}(1,0) = a_{11} + 2a_{21} + 3a_{31}$,
3. $f_{xy}(0,1) = p_{xy}(0,1) = a_{11} + 2a_{12} + 3a_{13}$,
4. $f_{xy}(1,1) = p_{xy}(1,1) = \sum_{i=1}^{3}\sum_{j=1}^{3} a_{ij} ij.$

$$\begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 1 & 0 & 3 \end{bmatrix},$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -3 & 3 & -2 & -1 \\ 2 & -2 & 1 & 1 \end{bmatrix} \begin{bmatrix} f(0,0) & f(0,1) & f_y(0,0) & f_y(0,1) \\ f(1,0) & f(1,1) & f_y(1,0) & f_y(1,1) \\ f_x(0,0) & f_x(0,1) & f_{xy}(0,0) & f_{xy}(0,1) \\ f_x(1,0) & f_x(1,1) & f_{xy}(1,0) & f_{xy}(1,1) \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 & 2 \\ 0 & 0 & 3 & -2 \\ 0 & 1 & -2 & 1 \\ 0 & 0 & -1 & 1 \end{bmatrix},$$

$$p(x,y) = \begin{bmatrix} 1 & x & x^2 & x^3 \end{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} 1 \\ y \\ y^2 \\ y^3 \end{bmatrix}.$$

經過這一系列的計算之後，就能算出 P(x,y)的值了！

參考資料：天空部落、維基百科

心得與討論：

- 為何 Header 中的點陣圖大小跟用長寬跟 bits 數算出來的結果不同？

  Header 中的點陣圖大小，有 DWORD-aligned 的問題，會將每一列的大小變成 4 的倍數，於是可能會比真正的大小還大。正確的方法必須使用長*寬*bits 數算出真正的 byte 數量。
- 16 bits 以下的資料需要給出調色盤資訊檔案才能運作。
- 請助教多多描述題目謝謝 =)