109 學年度二溫良期末資訊專題
玩美濾鏡

作者：二年良班 15 號 柯尹萱
指導老師：陳怡芬老師
完成日期：2020/01/10

一、專題名稱

玩美濾鏡

二、基本原理介紹

現代社會中，修圖軟體日漸增加，先進的科技讓我們能將平凡的照片變的不平凡。這次報告我運用基本的位置轉換、色彩調換及分配.....等，讓圖片呈現出不同的效果，做出了一些基本簡單的濾鏡。

三、使用工具

Codeblocks、GIMP

四、系統流程

A. 顯示開頭訊息

B. 選擇想要的濾鏡效果

C. 輸入檔案名稱並讀取圖片(有些需要再輸入想要的程度)

D. 輸出檔案名稱

E. 得到圖片

五、分項濾鏡實作

這次的程式實作我用的是網路上的照片，分別是知名鋼琴家郎朗和妻子吉娜·艾莉絲、澳洲歌手 Troye Sivan 以及中國演員張雲龍和歌手華晨宇。

這是原圖

I幾何轉換

A.複製(Copy)

a. 原理介紹：

直接讓第二張圖片的每一格都和讀取的圖片相同即可。

```
void copy(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
  image2.row=image1.row ;
  image2.col = image1.col ;
  for(int i=0 ; i<image1.row ; i++){
    for(int j=0 ; j<image1.col ; j++){
      image2.val[i][j] = image1.val[i][j] ;
    }
  }
}
```

b. 成果展現

B.放大&縮小(Enlarge & Shrink)

a. 原理介紹：

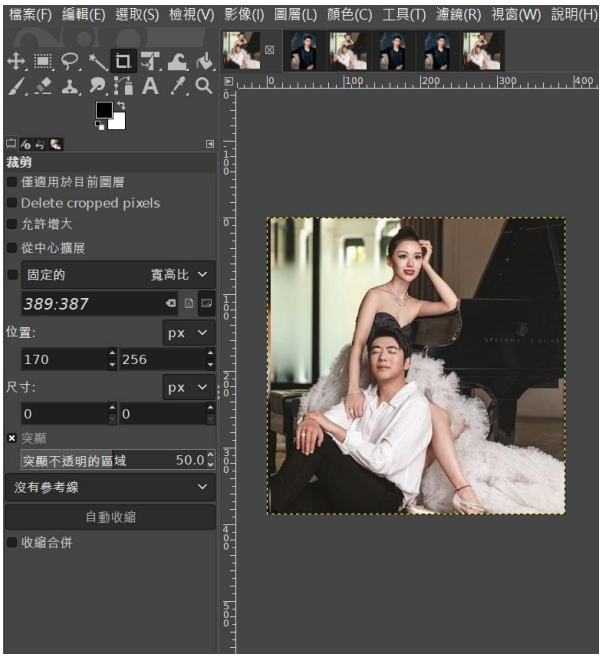
放大即是在將圖片寫入時，將寫入的內容複製三次(右、右下、下)，而縮小則是在寫入時隔行取值。

```
void enlarge(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
  image2.row=image1.row*2 ;
  image2.col = image1.col*2 ;
  for(int i1=0, i2=0 ; i1<image1.row ; i1++, i2+=2){
    for(int j1=0, j2=0 ; j1<image1.col ; j1++, j2+=2){
      image2.val[i2][j2] = image2.val[i2][j2+1]
=image2.val[i2+1][j2]
       =image2.val[i2+1][j2+1] = image1.val[i1][j1] ;
    }
  }
}

void shrink(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
```

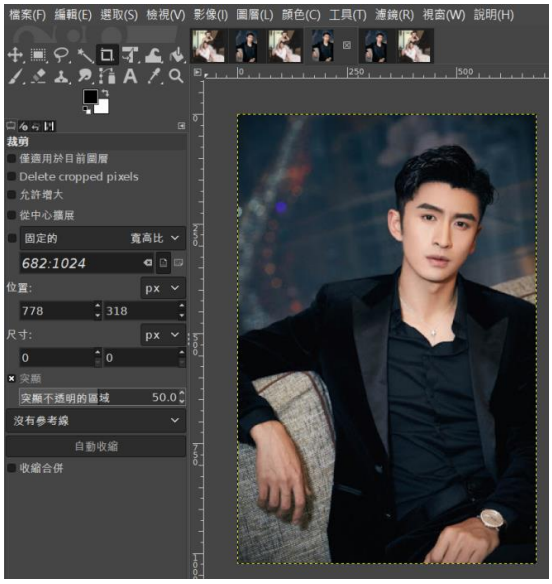
```
  image2.row=image1.row/2 ;
  image2.col = image1.col/2 ;
  for(int i1=0, i2=0 ; i1<image1.row ; i1+=2, i2++){
    for(int j1=0, j2=0 ; j1<image1.col ; j1+=2, j2++){
      image2.val[i2][j2] = image1.val[i1][j1] ;
    }
  }
}
```

b. 成果展現

原始圖片的大小



放大後的圖片大小



縮小後的圖片大小

C.水平&垂直鏡像(Horizontal Mirror & Vertical Mirror)

a. 原理介紹：

鏡像是透過將原圖片的上下左右進行調換，進而得到一張如同鏡中影像的圖片。水平鏡像是將原圖片的下變上，而垂直鏡像則是將左右對調。

```
void hMirror(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
  image2.row=image1.row ;
  image2.col = image1.col ;
  for (int i1=image1.row-1, i2=0 ; i1>=0 ; i1--, i2++){
    for (int j=0 ; j<image1.col ; j++){
      image2.val[i2][j] = image1.val[i1][j] ;
    }
  }
}

void vMirror(const Image& image1, Image& image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;
  for(int r=0 ; r<image1.row ; r++){
    for(int c2=0, c1=image1.col-1 ; c2<image2.col ; c2++, c1--){
      image2.val[r][c2] = image1.val[r][c1];
    }
  }
}
```

b. 成果展現

(1)水平鏡像

(2)垂直鏡像

D.旋轉(Rotate)

a. 原理介紹：

在我的程式碼中，我做的是向右轉90°，也就是將左右變成上下，上下便右左。

```
void rotate(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
  image2.row=image1.col ;
  image2.col = image1.row ;
  for(int r2=0, c1=0 ; r2<image2.row ; r2++, c1++){
    for(int c2=0, r1=image1.row-1 ; c2<image2.col ; c2++, r1--){
      image2.val[r2][c2] = image1.val[r1][c1];
    }
  }
}
```

b. 成果展現

II色彩轉換

A.灰階(Gray Scale)

a. 原理介紹：

利用網路上找到的公式，對圖片的 RGB 做計算。我找到了兩個公式，一個是常見的 $0.299 * R + 0.587 * G + 0.114 * B$，而另外一條則是 Adobe Photoshop 所使用的公式 $\sqrt[2.2]{(R^{2.2} * 0.2973) + (G^{2.2} * 0.6274) + (B^{2.2} * 0.0753)}$。兩條公式做出來的效果其實看起來差的並不多，但於我個人而言，我比較喜歡 Adobe Photoshop 那一條做出來的效果。

由於我有兩個選擇，因此我在程式碼中寫了一個 if-else 的迴圈，就由輸入不同選擇來決定變數 gray 的數值要用哪一條算式計算。

```
void grayScale(int l, const Image& image1, Image& image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;
  for(int r=0 ; r<image1.row ; r++){
    for(int c=0 ; c<image1.col ; c++){
      const Pixel& p=image1.val[r][c] ;
      int gray ;
      if(l==1) gray = 0.299*p.R + 0.587*p.G + 0.114*p.B ;
      else gray=pow((pow(p.R, 2.2)*0.2973+pow(p.G, 2.2)*0.6274+
        pow(p.B, 2.2)*0.0753), (1/2.2)) ;
      image2.val[r][c] = {gray, gray, gray} ;
    }
  }
}
```

b. 成果展現

第一種灰階

第二種灰階

B.保留單色(Red Scale, Green Scale, Blue Scale)

a. 原理介紹：

將除了要保留的顏色以外的數值全部改成 0，只留下想要的顏色。

由於保留單色總共有三種選擇，所以我在函式裡用 switch-case 來替 if-else 對進來函式的不同數值進行不同的行為。

```
void singleColorScale(int color, const Image& image1, Image&
image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;
  switch(color){
    case 0:
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          image2.val[r][c] = {image1.val[r][c].R, 0, 0} ;
        }
      }
      break ;
    case 1:
```

```
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          image2.val[r][c] = {0, image1.val[r][c].G, 0} ;
        }
      }
      break ;
    case 2:
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          image2.val[r][c] = {0, 0, image1.val[r][c].B} ;
        }
      }
      break ;
    default:
      break ;
  }
}
```

b. 成果展現

(1)只留紅色

(2)只留綠色

(3)只留藍色

C.去除單色(Remove Red, Remove Green, Remove Blue)

a. 原理介紹：

和保留單色的方法很像，但不是將另外兩色數值改 0，這次只把一個顏色（想去除的顏色)改成 0。

和保留單色一樣，我在函式裡也用了 switch-case 進行不同不同顏色的去除。

```
void removeSingleColor(int color, const Image& image1, Image&
image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;
  switch(color){
    case 0:
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          const Pixel& p = image1.val[r][c] ;
          image2.val[r][c] = {0, p.G, p.B} ;
        }
      }
      break ;
    case 1:
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          const Pixel& p = image1.val[r][c] ;
          image2.val[r][c] = {p.R, 0, p.B} ;
        }
      }
      break ;
    case 2:
      for(int r=0 ; r<image1.row ; r++){
        for(int c=0 ; c<image1.col ; c++){
          const Pixel& p = image1.val[r][c] ;
          image2.val[r][c] = {p.R, p.G, 0} ;
        }
      }
      break ;
  }
}
```

b. 成果展現

(1)Remove red

(2)Remove green

(3)Remove blue

D.像素馬賽克(Pixel Mosaic)

a. 原理介紹：

在進行馬賽克前，先定義想要的核心數值和周圍權重。我用的數值是從網路上找的，範圍是 11x11。我總共定義的三個不同數值的核心，不同數值做出來的結果也不一樣。

為了避免程式碼太複雜，我定義了一個函式專門做單一像素的計算，一樣用了 switch-case 來區別不同程度的模糊化。

不同核心的數值

```
constexpr int KERNEL_WEIGHT1=965 ;

int kernel1[11][11]={
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 13, 22, 13, 0, 0, 0, 0},
  {0, 0, 0, 13, 59, 97, 59, 13, 0, 0, 0},
  {0, 0, 0, 22, 97, 149, 97, 22, 0, 0, 0},
  {0, 0, 0, 13, 59, 97, 59, 13, 0, 0, 0},
  {0, 0, 0, 0, 13, 22, 13, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
} ;

constexpr double KERNEL_WEIGHT2=992 ;

int kernel2[11][11]={
  {0, 2, 3, 4, 5, 5, 5, 4, 3, 2, 0},
  {2, 3, 5, 7, 8, 8, 8, 7, 5, 3, 2},
  {3, 5, 7, 10, 12, 12, 12, 10, 7, 5, 3},
  {4, 7, 10, 13, 15, 16, 15, 13, 10, 7, 4},
  {5, 8, 12, 15, 18, 19, 18, 15, 12, 8, 5},
  {5, 8, 12, 16, 19, 20, 19, 16, 12, 8, 5},
  {5, 8, 12, 15, 18, 19, 18, 15, 12, 8, 5},
  {4, 7, 10, 13, 15, 16, 15, 13, 10, 7, 4},
  {3, 5, 7, 10, 12, 12, 12, 10, 7, 5, 3},
  {2, 3, 5, 7, 8, 8, 8, 7, 5, 3, 2},
  {0, 2, 3, 4, 5, 5, 5, 4, 3, 2, 0}
} ;
```

```
constexpr int KERNEL_WEIGHT3=980 ;

int kernel3[11][11]={
  {0, 5, 6, 7, 7, 7, 7, 7, 6, 5, 0},
  {5, 6, 7, 8, 9, 9, 9, 8, 7, 6, 5},
  {6, 7, 8, 9, 10, 10, 10, 9, 8, 7, 6},
  {7, 8, 9, 10, 11, 11, 11, 10, 9, 8, 7},
  {7, 9, 10, 11, 11, 12, 11, 11, 10, 9, 7},
  {7, 9, 10, 11, 12, 12, 12, 11, 10, 9, 7},
  {7, 9, 10, 11, 11, 12, 11, 11, 10, 9, 7},
  {7, 8, 9, 10, 11, 11, 11, 10, 9, 8, 7},
  {6, 7, 8, 9, 10, 10, 10, 9, 8, 7, 6},
  {5, 6, 7, 8, 9, 9, 9, 8, 7, 6, 5},
  {0, 5, 6, 7, 7, 7, 7, 7, 6, 5, 0}
} ;
```

單一像素的模糊

```
Pixel singleMosaicPixel(int level, Pixel& p, const Image& image,
int r, int c){
  int R, G, B ;
  R=G=B=0 ;

  switch(level){
    case 1:
      for(int i=0 ; i<11 ; i++){
        for(int j=0 ; j<11 ; j++){
          const Pixel& tmp = image.val[r+i-1][c+j-1] ;
          //if(r+i-1<0 || r+i-1>=r || c+j-1<0 || c+j-1>=c)
continue ;
          R+=kernel1[i][j]*tmp.R ;
          G+=kernel1[i][j]*tmp.G ;
          B+=kernel1[i][j]*tmp.B ;
        }
      }
      R=round(R/KERNEL_WEIGHT1) ;
      G=round(G/KERNEL_WEIGHT1) ;
      B=round(B/KERNEL_WEIGHT1) ;
      break ;
    case 2:
```

```
      for(int i=0 ; i<11 ; i++){
        for(int j=0 ; j<11 ; j++){
          const Pixel& tmp = image.val[r+i-1][c+j-1] ;
          //if(r+i-1<0 || r+i-1>=r || c+j-1<0 || c+j-1>=c)
continue ;
          R+=kernel2[i][j]*tmp.R ;
          G+=kernel2[i][j]*tmp.G ;
          B+=kernel2[i][j]*tmp.B ;
        }
      }
      R=round(R/KERNEL_WEIGHT2) ;
      G=round(G/KERNEL_WEIGHT2) ;
      B=round(B/KERNEL_WEIGHT2) ;
      break ;
    case 3:
      for(int i=0 ; i<11 ; i++){
        for(int j=0 ; j<11 ; j++){
          const Pixel& tmp = image.val[r+i-1][c+j-1] ;
          //if(r+i-1<0 || r+i-1>=r || c+j-1<0 || c+j-1>=c)
continue ;
          R+=kernel3[i][j]*tmp.R ;
          G+=kernel3[i][j]*tmp.G ;
          B+=kernel3[i][j]*tmp.B ;
        }
      }
      R=round(R/KERNEL_WEIGHT3) ;
      G=round(G/KERNEL_WEIGHT3) ;
      B=round(B/KERNEL_WEIGHT3) ;
      break ;
    default:
      break ;
  }
  return p={R, G, B} ;
}
```

馬賽克的函式

```
void pixelMosaic(int level, const Image& image1, Image& image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
```

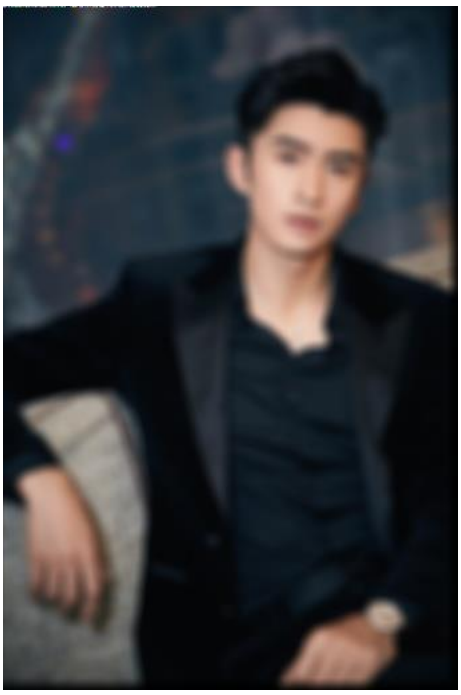
```
  image2.col = image1.col ;

  for(int r=0 ; r<image1.row ; r++){
    for(int c=0 ; c<image1.col ; c++){
      image2.val[r][c] = image1.val[r][c] ;
    }
  }

  for (int r=0 ; r<image1.row ; r++){
    for (int c=0 ; c<image1.col ; c++){
      image2.val[r][c] = singleMosaicPixel(level,
image2.val[r][c], image1, r, c) ;
    }
  }
}
```

b. 成果展現

(1) kernel1

(2) kernel2

(3) kernel3



E. 負片 (Negative)

a. 原理介紹：

說到負片我們一般會和攝影有關的事物聯想在一起，負片就是讓整張照片的色調和亮度皆和原來相反。平常我們所用的光的三原色 RGB 的最大值為 255，負片就是用 255 減去原來圖片的 RGB 值，進而達到明暗、色彩和原來照片相反的效果。

```
void negative(const Image& image1, Image& image2){
  image2.mode=image1.mode ;
  image2.type = image1.type ;
  image2.row=image1.row ;
  image2.col = image1.col ;
  for(int r=0 ; r<image1.row ; r++){
    for(int c=0 ; c<image1.col ; c++){
      const Pixel& p1=image1.val[r][c] ;
      image2.val[r][c] = {255-p1.R, 255-p1.G, 255-p1.B} ;
    }
  }
}
```

b. 成果展現：

F.色彩馬賽克(Color Mosaic)

a. 原理介紹：

將色彩馬賽克化其實就是調亮度的概念，把 RGB 的值同除一個數值後在乘上一個共同的數值進而達到等比例調降的概念。我在程式碼中是同除 20 在同乘 5。

```
void colorMosaic(const Image& image1, Image& image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;
  for (int r=1 ; r<image1.row-1 ; r++){
    for (int c=1 ; c<image1.col-1 ; c++){
      const Pixel& p=image1.val[r][c] ;
      int R=round(p.R/20*5) ;
      int G=round(p.G/20*5) ;
      int B=round(p.B/20*5) ;
      image2.val[r][c]={R, G, B} ;
    }
  }
}
```

b. 成果展現：

G.交換顏色(Exchange Color)

a. 原理介紹：

一般來說，照片儲存顏色的順序是紅色(Red)、綠色(Green)、藍(Blue)，也就是依照 RGB 的順序來存。交換顏色就是我讓 RGB 進行重新排列，來改變照片呈現出來的顏色。

在交換顏色這邊我總共有 5 種，分別是 RBG、BRG、BGR、GRB、GBR。

```
void exchangeColor(int choice, const Image& image1, Image&
image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;

  switch(choice){
    case 1:
      for (int r=0 ; r<image1.row ; r++){
        for (int c=0 ; c<image1.col ; c++){
          const Pixel& p=image1.val[r][c] ;
          image2.val[r][c]={p.R, p.B, p.G} ;
        }
      }
      break ;
    case 2:
      for (int r=0 ; r<image1.row ; r++){
        for (int c=0 ; c<image1.col ; c++){
          const Pixel& p=image1.val[r][c] ;
          image2.val[r][c]={p.B, p.R, p.G} ;
        }
      }
      break ;
    case 3:
      for (int r=0 ; r<image1.row ; r++){
        for (int c=0 ; c<image1.col ; c++){
          const Pixel& p=image1.val[r][c] ;
          image2.val[r][c]={p.B, p.G, p.R} ;
        }
      }
      break ;
    case 4:
      for (int r=0 ; r<image1.row ; r++){
```

```
        for (int c=0 ; c<image1.col ; c++){
          const Pixel& p=image1.val[r][c] ;
          image2.val[r][c]={p.G, p.R, p.B} ;
        }
      }
      break ;
    case 5:
      for (int r=0 ; r<image1.row ; r++){
        for (int c=0 ; c<image1.col ; c++){
          const Pixel& p=image1.val[r][c] ;
          image2.val[r][c]={p.G, p.B, p.R} ;
        }
      }
      break ;
    default:
      break ;
  }
}
```

b. 成果展現：

(1) RBG

(2) BRG

(3) BGR

(4) GRB

(5) GBR

H.銳化(Sharp)

a. 原理介紹：

其實一開始我本來是想做銳化濾鏡，但不知道為什麼一直做不出來。後來發現是因為數值超出範圍才失敗的，加了判斷式後就成功了，但是做出來的效果有點像是銳化加上曝光。我的方法是和像素馬賽克一樣先用單一像素的改變函式將數值存下來（負數)，並和前面馬賽克化的第一種數值相加。

我的數值核心

```
constexpr int S_KERNEL_WEIGHT=997 ;

int s_kernel[11][11]={
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, -1, -2, -1, 0, 0, 0, 0},
  {0, 0, 0, -3, -13, -22, -13, -3, 0, 0, 0},
  {0, 0, -1, -13, -59, -97, -59, -13, -1, 0, 0},
  {0, 0, -2, -22, -97, 1841, -97, -22, -2, 0, 0},
  {0, 0, -1, -13, -59, -97, -59, -13, -1, 0, 0},
  {0, 0, 0, -3, -13, -22, -13, -3, 0, 0, 0},
  {0, 0, 0, 0, -1, -2, -1, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
  {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
} ;
```

單一像素的改變。

```
Pixel singlePixelSharp(Pixel& p, const Image& image, int r, int
c){
  int R, G, B ;
  R=G=B=0 ;

    for(int i=0 ; i<11 ; i++){
      for(int j=0 ; j<11 ; j++){
        const Pixel& tmp = image.val[r+i-1][c+j-1] ;
        //if(r+i-1<0 || r+i-1>=r || c+j-1<0 || c+j-1>=c)
continue ;
        R+=s_kernel[i][j]*tmp.R ;
        G+=s_kernel[i][j]*tmp.G ;
        B+=s_kernel[i][j]*tmp.B ;
      }
```

```
    }
    R=round(R/S_KERNEL_WEIGHT) ;
    G=round(G/S_KERNEL_WEIGHT) ;
    B=round(B/S_KERNEL_WEIGHT) ;
  return p={R, G, B} ;
}
```

銳化濾鏡的函式

```
void sharpFilter(const Image& image1, Image& image2){
  image2.mode = image1.mode ;
  image2.type = image1.type ;
  image2.row = image1.row ;
  image2.col = image1.col ;

  for(int r=0 ; r<image1.row ; r++){
    for(int c=0 ; c<image1.col ; c++){
      image2.val[r][c] = image1.val[r][c] ;
    }
  }

  for (int r=0 ; r<image1.row ; r++){
    for (int c=0 ; c<image1.col ; c++){
      Pixel p1=singleMosaicPixel(1, image2.val[r][c], image1, r,
c) ;
      Pixel p2=singlePixelSharp(image2.val[r][c], image1, r, c) ;
      int R=p1.R+p2.R, G=p1.G+p2.G, B=p1.B+p2.B ;
      if(R<0) R=0 ;
      if(G<0) G=0 ;
      if(B<0) B=0 ;
      if(R>255) R=255 ;
      if(G>255) G=255 ;
      if(B>255) B=255 ;
      image2.val[r][c]={R, G, B} ;
    }
  }
}
```

b. 成果展現：

可以看到雖然銳化的效果有出來，但是更加明顯的似乎是曝光的效果。

I.疊圖(Overlay)

a. 原理介紹：

藉由將兩張圖片的 RGB 值分別乘上不同的權重，並加在一起，我讓第一張圖為主要圖片我試了兩種，第一種的權重為 0.8 和 0.2，第二種的權重為 0.7 和 0.3。

```
void overlay(const Image& image1, const Image& image2, Image&
image3){
  image3.mode=image1.mode ;
  image3.type = image1.type ;
  image3.row=image1.row ;
  image3.col = image1.col ;
  for(int r=0 ; r<image1.row ; r++){
    for(int c=0 ; c<image1.col ; c++){
      const Pixel& p1=image1.val[r][c] ;
      const Pixel& p2=image2.val[r][c] ;
      int R=p1.R*0.8+p2.R*0.2 ;
      int G=p1.G*0.8+p2.G*0.2 ;
      int B=p1.B*0.8+p2.B*0.2 ;
      image3.val[r][c] = {R, G, B} ;
    }
  }
}
```

b. 成果展現：

這是我疊上去的圖

(1) 權重為 0.8、0.2

(2) 權重為 0.7、0.3

六、失敗案例

當初在做銳化濾鏡的時候，因為沒有考慮到算出來的值超出 0~255 的範圍，所以造成了上面這種奇怪的顏色，在程式碼中加入 if-else 的判斷式後便成功解決了。

七、心得

這次的資訊專題我覺得非常有趣，平常我們常常使用的修圖技巧，原來只要透過一些基本的程式碼和數學運算就可以寫出來。一開始做的時候覺得有點困難，但抓到訣竅後就很容易上手了。我最喜歡的是疊圖的部分，我覺得所有濾鏡裡面效果最好的就是疊圖。濾鏡 H 雖然不是我原本想要的效果，但做出它的時候我其實覺得蠻酷的，因為經過它處理的濾鏡有點像溫度計感應出來的顏色，但說實話有點噁心。

這次的專題讓我學會了很多影像處理的技巧，是一個非常有趣的專題。但同時它也有點可怕，因為做濾鏡會做出興趣來，我在做的時候差點停不下來，已經完全入迷了。希望之後還有類似的專題可以做。

八、參考資料

照片來源

1. Troye Sivan 介紹：無關性向，關乎愛。以自己最真實的模樣，慶祝生命的紋路｜用音樂說故事
2. 新浪娛樂
3. 華晨宇 MV 拍攝到完成耗時 6 個月 新專輯歌迷整整等了 3 年
4. 郎朗牽超正辣妻合體現身！

參考資料

1. 【影像處理】Nearest Neighbor and Bilinear Interpolation
2. Some Image Processing and Computational Photography: Convolution, Filtering and Edge Detection with Python
3. Sharpening Filters