國立雲林科技大學

106學年度第二學期

**數位影像處理**

第三次平時作業

班 級：四電子三A

學 號：B10400001

學 生：郭鎧碩

授課教師：藍呂興 老師

繳交日期：107年06月23日

[一、 Skin Detection 3](#_Toc517528346)

1. [題目說明 3](#_Toc517528347)

[2. 本程式之演算法 3](#_Toc517528348)

[3. 製作方法 4](#_Toc517528349)

[4. 函數程式碼 4](#_Toc517528350)

[5. 結果圖與原圖比較 5](#_Toc517528351)

[二、 Butterworth Lowpass Filtering and Highpass Filtering 6](#_Toc517528352)

[**1.** 題目說明 6](#_Toc517528353)

[**2.** 本程式的演算法 6](#_Toc517528354)

[3. 製作方法 7](#_Toc517528355)

[4. 函數程式碼(函數寫在mask.cpp) 7](#_Toc517528356)

[5. 低通結果圖比較 11](#_Toc517528357)

[6. 高通結果圖比較(highpass) 12](#_Toc517528358)

[三、 Morphological Processing 14](#_Toc517528359)

[**1.** 題目說明 14](#_Toc517528360)

[**2.** 本程式的演算法 14](#_Toc517528361)

[**3.** 製作方式 15](#_Toc517528362)

[**4.** 函數程式碼(函數寫在mask.c) 15](#_Toc517528363)

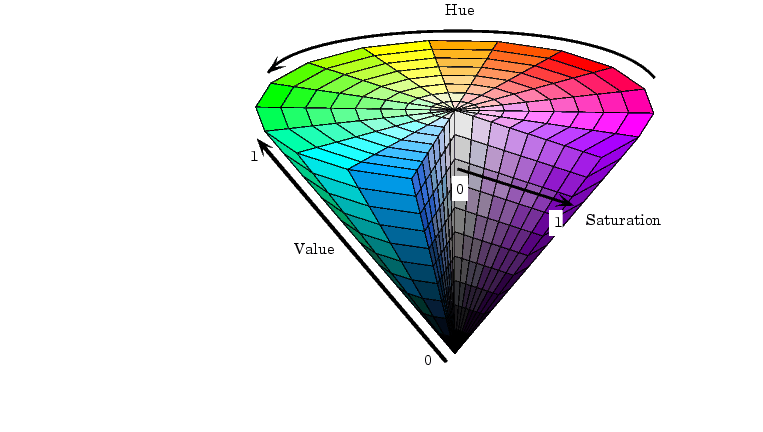
[5. 結果圖與原圖比較 16](#_Toc517528364)

[四、 心得 16](#_Toc517528365)

1. Skin Detection
2. 題目說明

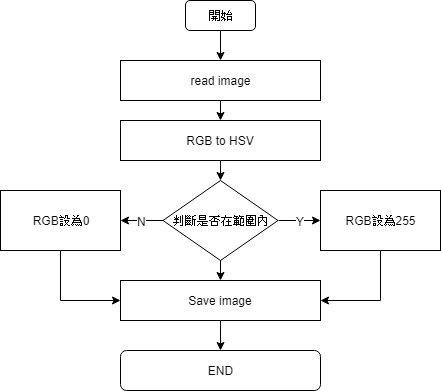
RGB(red, green, blue)顏色空間最常用的用途就是顯示器系統

HSV(Hue, Saturation, Value)是一種比較**直觀**的顏色模型，所以在許多圖像編輯工具中應用比較廣泛，如Photoshop，但這也決定了它不適合使用在光照模型中，許多光線混合運算、光強運算等都無法直接使用HSV來實現



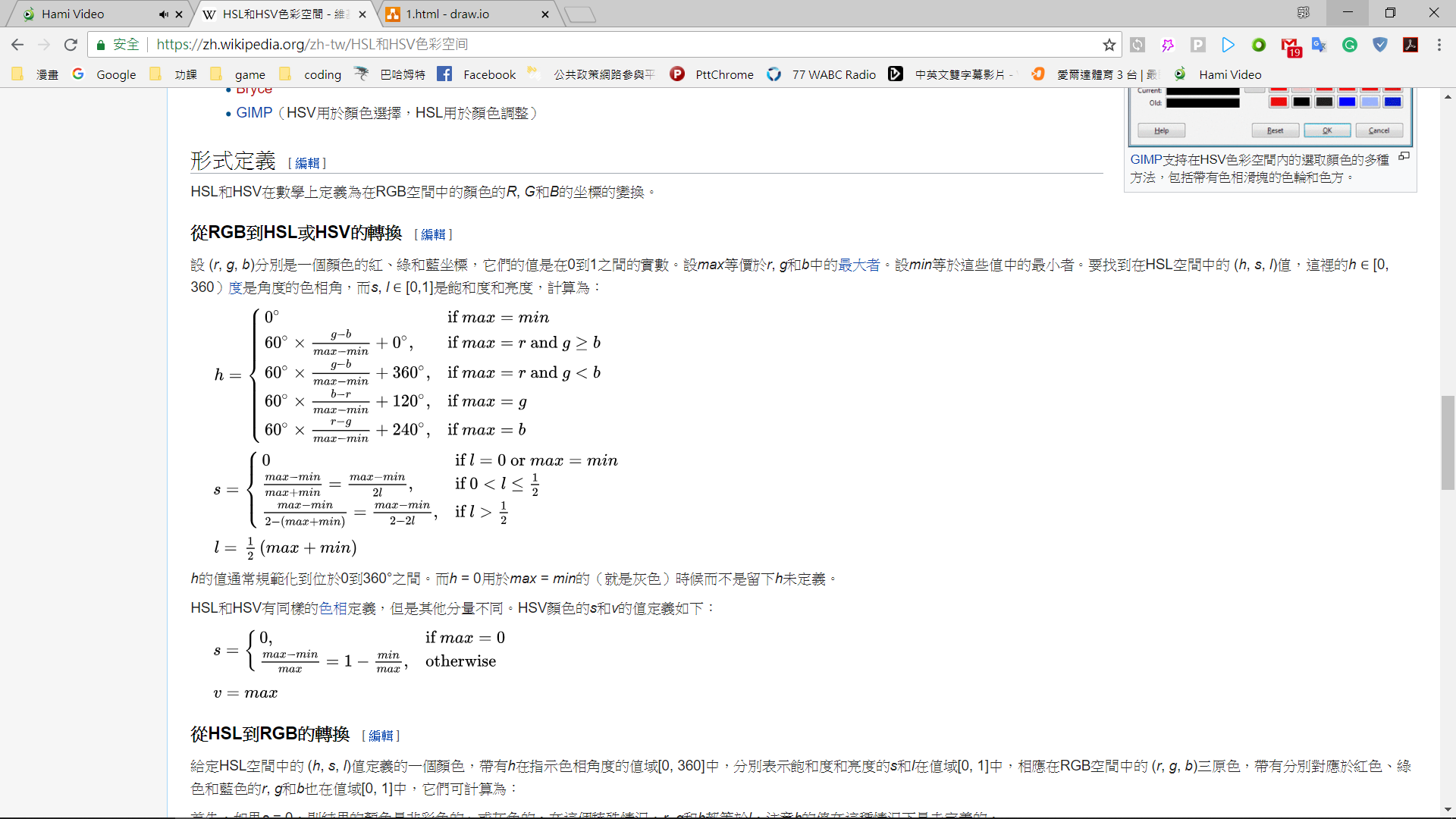
從ＲＧＢ轉到ＨＳＶ空間中得到：  
Ｈ：色相      0~360度  
Ｓ：飽和度   0~1  
Ｖ：明度      (黑)0~1(白)  
在顏色分明的情況時，色相（Ｈ）可以很容易地將顏色分離出來

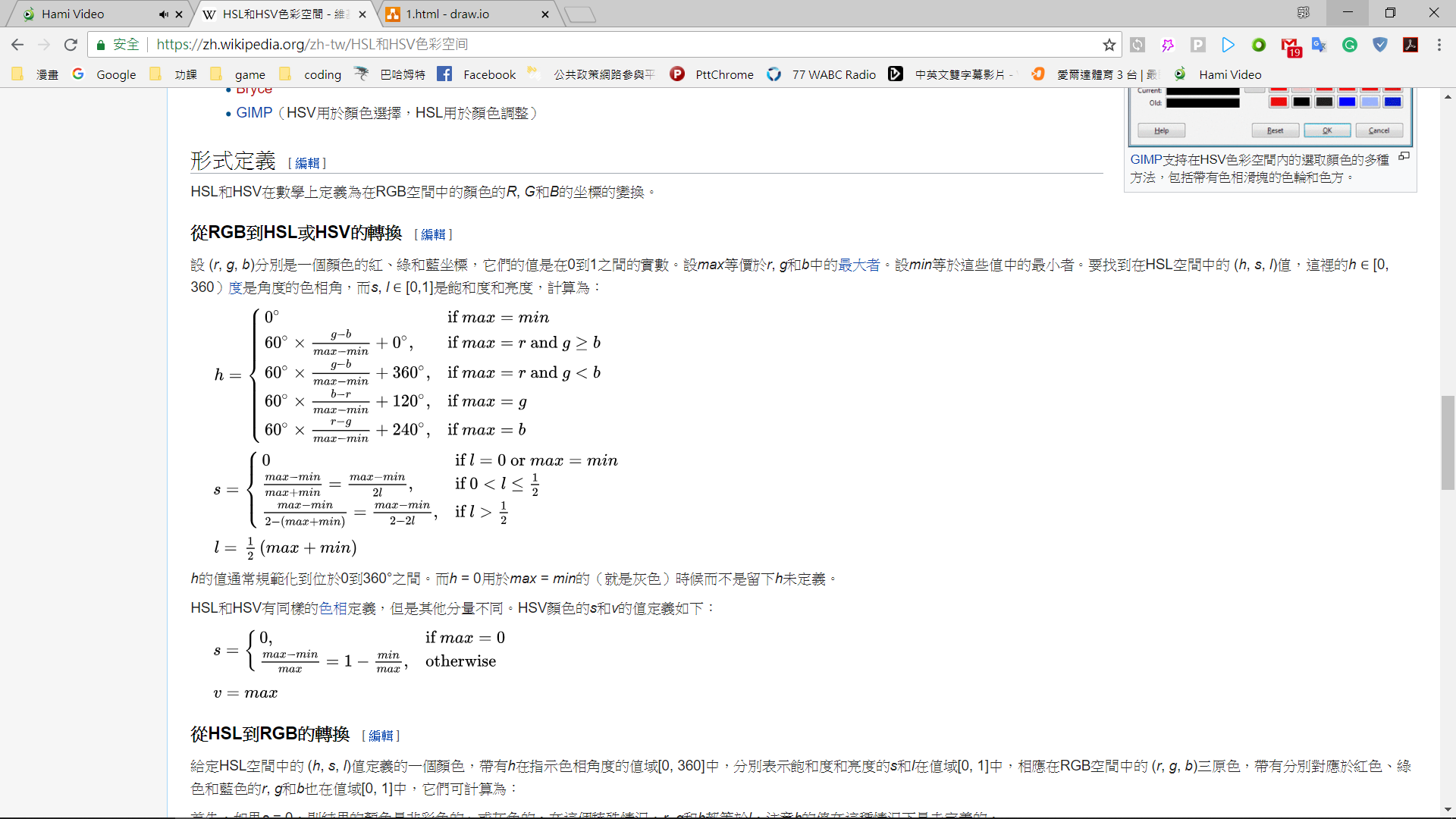
1. 本程式之演算法



1. 製作方法

根據wiki的形式定義，只要依據下方的公式即可得到hsv



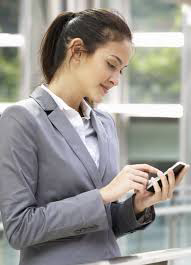


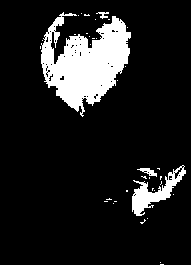
1. 函數程式碼

Rgb2hsv(H，S，V，R，G，B，width，height)

|  |
| --- |
| void rgb2hsv(int H[1024][1024], float S[1024][1024], float V[1024][1024],  int R[1024][1024], int G[1024][1024], int B[1024][1024],  int& width, int& height) {  float temp, delta;  float max, min;  float r, g, b;  for (int x = 0; x < width; x++)  for (int y = 0; y < height; y++) {  //normalize  r = (float)R[x][y]/255;  g = (float)G[x][y]/255;  b = (float)B[x][y]/255;  temp = (r >= g) ? r : g;  max = (temp >= b) ? temp : b; //max  temp = (r <= g) ? r : g;  min = (temp <= b) ? temp : b; //min    V[x][y] = max; //get V  delta = max - min;  if (max != 0)  S[x][y] = delta / max; //get S  else  S[x][y] = 0;  if (max == min)  H[x][y] = 0; //get H  else if (r == max && g >= b)  H[x][y] = round((g - b) / delta \* 60);  else if (r == max && g <= b)  H[x][y] = round((g - b) / delta \* 60) + 360;  else if (g == max)  H[x][y] = round((b - r) / delta \* 60) + 120;  else if (b == max)  H[x][y] = round((r - g) / delta \* 60) + 240;  }  } |

1. 結果圖與原圖比較





原圖

老師給的參考值

38 >= H >= 6

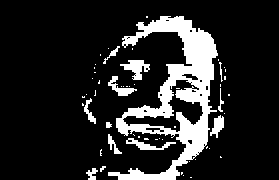
0.68 >= S >= 0.23

50 >= H >= 6

0.8 >= S >= 0.11

V >= 0.65





50 >= H >= 6

0.8 >= S >= 0.1

V >= 0.65

老師給的參考值

38 >= H >= 6

0.68 >= S >= 0.23

原圖

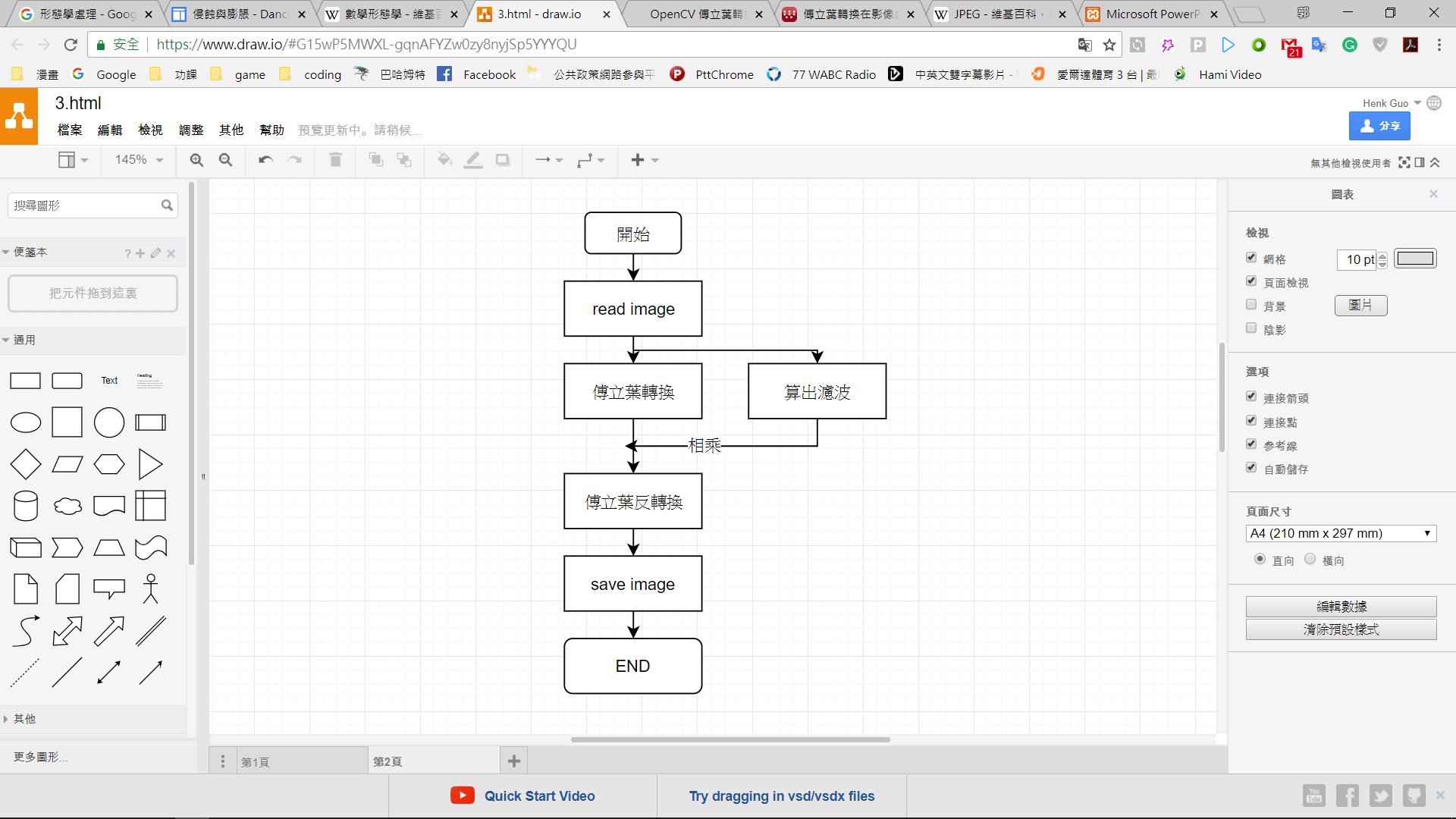
1. Butterworth Lowpass Filtering and Highpass Filtering
   1. 題目說明

二維傅立葉變換的作用是將影像由空間域變換到頻域，並對於影像不同頻段的成分進行分析與處理，所以二維傅立葉變換在影像處理領域有著舉足輕重的地位，其主要的應用為

* 影像分析
* 影像濾波
* 影像重建
* 影像壓縮

舉例JPEG相片格式中，做編碼時，就有用到離散餘弦變換（Discrete cosine transform），壓縮率這麼好的原因是：在頻率域其實是儲存某頻率波的係數，低頻部份通常很重要 (例如最低頻的常數其實就是整個色塊的平均值)，因此編碼時只要留下低頻部份 (a0, a1, …, ak)，高頻部份 (ak+1, … , an) 通常可以去除或減少取樣位元數，如此就達到了壓縮的目的。由此可見在影像壓縮時，就會用到傅立葉轉換。

* 1. 本程式的演算法



* 1. 製作方法

讀取圖片後，先使用老師給予的fft函數得到頻域值後，再與濾波相乘，最後再做傅立葉反轉換即可。

* 1. 函數程式碼(函數寫在mask.cpp)

Butterworth\_LF(輸出矩陣, 輸入矩陣, D0, n, width, height);

|  |
| --- |
| void Butterworth\_LF(int r[1024][1024], int R[1024][1024],float D0,int n, int width, int height) {  int i, j, switch\_temp;  int real[1024][1024] = { 0 }; //實數  int imaginary[1024][1024] = { 0 }; //虛數  float D;  float Data[256 \* 256 \* 2] = { 0 };  float Mask[1024][1024] = { 0 };  //Create Mask  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  D = hypot(abs(i - 128), abs(j - 128));  Mask[i][j] = 1 / (1 + powf((D / D0), 2\*n));  }  //2D to 1D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  Data[j \* 256 \* 2 + i \* 2] = (float)R[i][j];  Data[j \* 256 \* 2 + i \* 2 + 1] = 0;  }  fft2(Data, 256, 1);  //1-D to 2-D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  real[i][j] = (int)Data[j \* 256 \* 2 + i \* 2];  imaginary[i][j] = (int)Data[j \* 256 \* 2 + i \* 2 + 1];  }  //switch  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  if (i < width / 2 && j < height / 2) {  switch\_temp = real[i][j];  real[i][j] = real[i + width / 2][j + height / 2];  real[i + width / 2][j + height / 2] = switch\_temp;  }  else if ((i >= width / 2 && j < height / 2)) {  switch\_temp = real[i][j];  real[i][j] = real[i - width / 2][j + height / 2];  real[i - width / 2][j + height / 2] = switch\_temp;  }  }  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  if (i < width / 2 && j < height / 2) {  switch\_temp = imaginary[i][j];  imaginary[i][j] = imaginary[i + width / 2][j + height / 2];  imaginary[i + width / 2][j + height / 2] = switch\_temp;  }  else if ((i >= width / 2 && j < height / 2)) {  switch\_temp = imaginary[i][j];  imaginary[i][j] = imaginary[i - width / 2][j + height / 2];  imaginary[i - width / 2][j + height / 2] = switch\_temp;  }  }  //multiply  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  real[i][j] = (float)real[i][j] \* Mask[i][j];  imaginary[i][j] = (float)imaginary[i][j] \* Mask[i][j];  }  //2D to 1D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  Data[j \* 256 \* 2 + i \* 2] = (float)real[i][j];  Data[j \* 256 \* 2 + i \* 2 + 1] = (float)imaginary[i][j];  }  //fft inverse  fft2(Data, 256, -1);  for (i = 0; i < width; i++)  for (j = 0; j < height; j++)  r[i][j] = abs((int)Data[j \* 256 \* 2 + i \* 2]);  } |

Butterworth\_HF(輸出矩陣, 輸入矩陣, D0, n, width, height);

|  |
| --- |
| void Butterworth\_HF(int r[1024][1024], int R[1024][1024], float D0, int n, int width, int height) {  int i, j, switch\_temp;  int real[1024][1024] = { 0 };  int imaginary[1024][1024] = { 0 };  float D;  float Data[256 \* 256 \* 2] = { 0 };  float Mask[1024][1024] = { 0 };  //Create Mask  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  D = hypot(abs(i - 128), abs(j - 128));  Mask[i][j] = 1 / (1 + powf((D0 / D), 2 \* n));  }  //2D to 1D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  Data[j \* 256 \* 2 + i \* 2] = (float)R[i][j];  Data[j \* 256 \* 2 + i \* 2 + 1] = 0;  }  fft2(Data, 256, 1);  //1-D to 2-D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  real[i][j] = (int)Data[j \* 256 \* 2 + i \* 2];  imaginary[i][j] = (int)Data[j \* 256 \* 2 + i \* 2 + 1];  }  //switch  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  if (i < width / 2 && j < height / 2) {  switch\_temp = real[i][j];  real[i][j] = real[i + width / 2][j + height / 2];  real[i + width / 2][j + height / 2] = switch\_temp;  }  else if ((i >= width / 2 && j < height / 2)) {  switch\_temp = real[i][j];  real[i][j] = real[i - width / 2][j + height / 2];  real[i - width / 2][j + height / 2] = switch\_temp;  }  }  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  if (i < width / 2 && j < height / 2) {  switch\_temp = imaginary[i][j];  imaginary[i][j] = imaginary[i + width / 2][j + height / 2];  imaginary[i + width / 2][j + height / 2] = switch\_temp;  }  else if ((i >= width / 2 && j < height / 2)) {  switch\_temp = imaginary[i][j];  imaginary[i][j] = imaginary[i - width / 2][j + height / 2];  imaginary[i - width / 2][j + height / 2] = switch\_temp;  }  }  //multiply  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  real[i][j] = (float)real[i][j] \* Mask[i][j];  imaginary[i][j] = (float)imaginary[i][j] \* Mask[i][j];  }  //2D to 1D  for (i = 0; i < width; i++)  for (j = 0; j < height; j++) {  Data[j \* 256 \* 2 + i \* 2] = (float)real[i][j];  Data[j \* 256 \* 2 + i \* 2 + 1] = (float)imaginary[i][j];  }  //fft inverse  fft2(Data, 256, -1);  for (i = 0; i < width; i++)  for (j = 0; j < height; j++)  r[i][j] = abs((int)Data[j \* 256 \* 2 + i \* 2]);  } |

* 1. 低通結果圖比較



N=1，d0=15

N=5，d0=15

N=10，d0=15

若d0固定，n越大看起來有擴散的效果，上方攝影師周圍有一圈的痕跡。



N=2，d0=5

N=2，d0=10

N=2，d0=50

若N固定，d0越大看起來越模糊。

1. 高通結果圖比較(highpass)







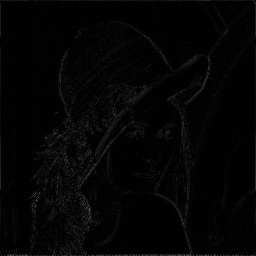


N=1，d0=15

N=5，d0=15

N=10，d0=15

D0固定時，隨著n增加，雖然感覺沒甚麼改變，但仔細看邊緣，細節的部分會增加。



N=2，d0=5

N=2，d0=10

N=2，d0=50

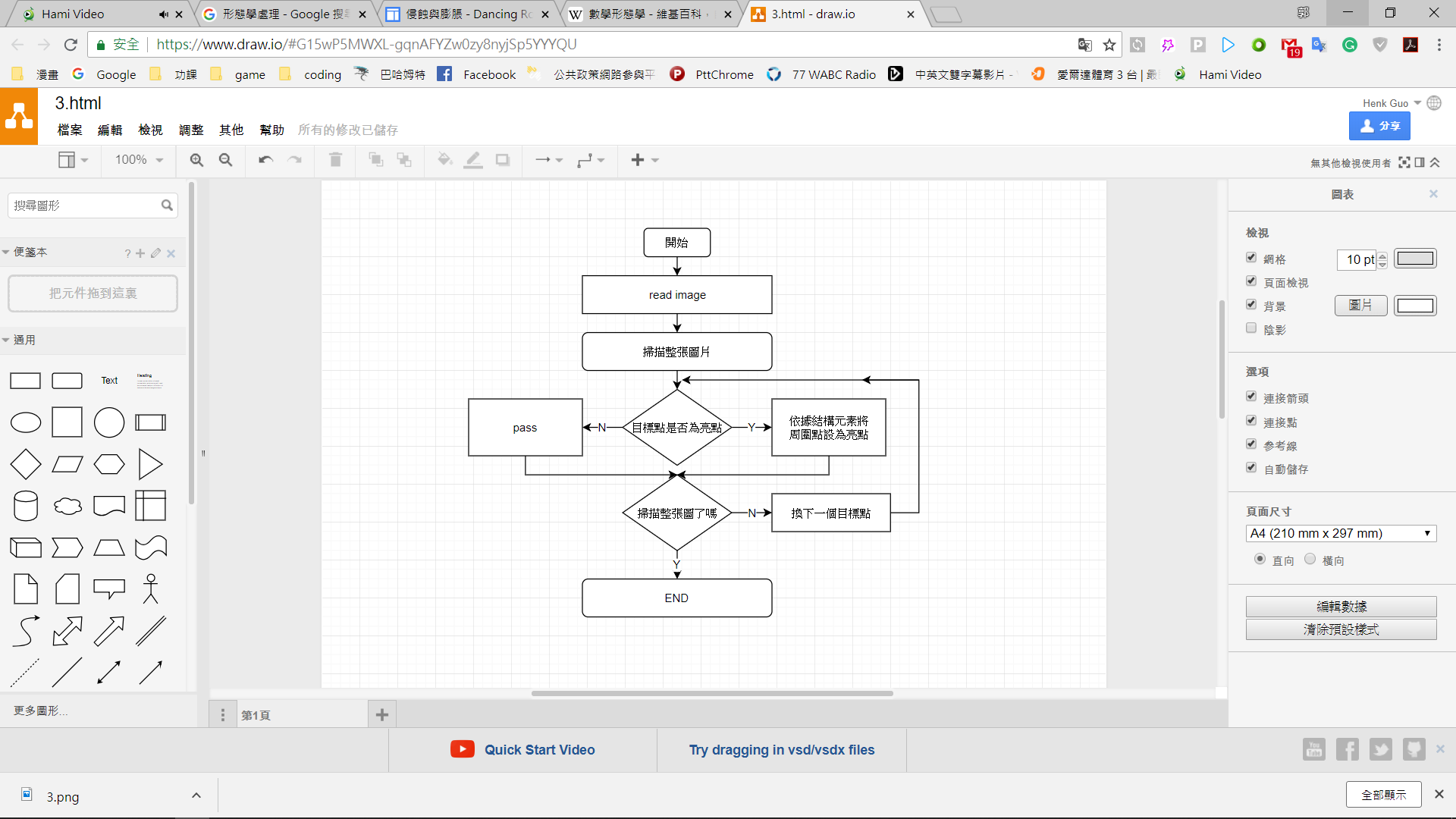
N固定時，d0增加時很明顯的看出邊緣的亮度會變暗。

1. Morphological Processing
   1. 題目說明

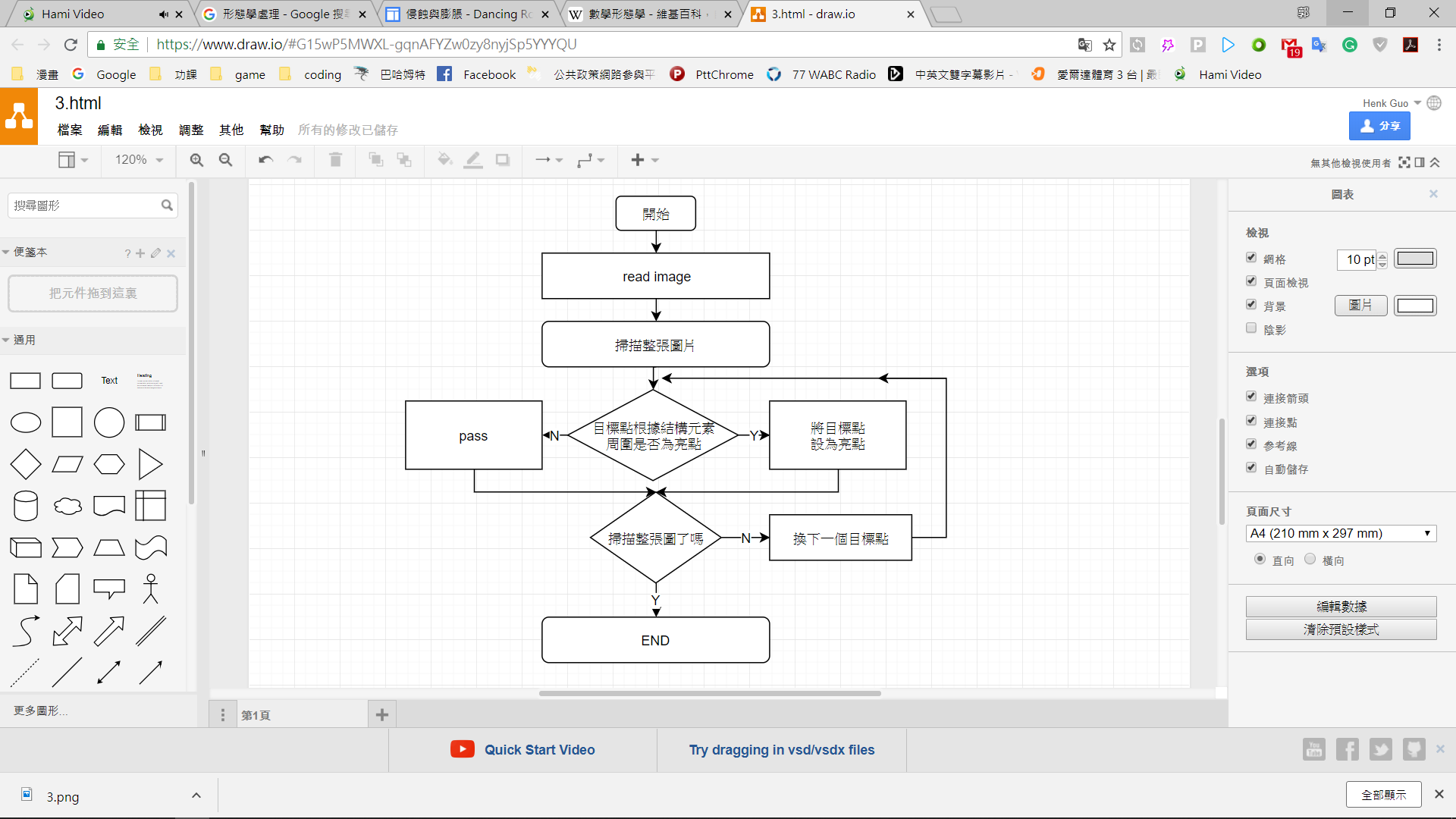
形態學主要用於二值化後的影像，根據使用者的目的，用來凸顯影像的形狀特徵，像邊界和連通區域等，同時像細化、像素化、修剪毛刺等技術也常用於圖像的預處理和後處理，形態學操作的結果除了影像本身，也和結構元素的形狀有關，結構元素和空間域操作的濾波概念類似在實際的使用上，是以奇數的矩形如3×3、5×5、7×7較常見。

* 1. 本程式的演算法

膨脹



侵蝕



* 1. 製作方式

侵蝕顧名思義就是消融物體的邊界，侵蝕的結果是讓物體瘦一圈，而這一圈的寬度是由結構元素大小決定的，如果物體小於結構元素，則侵蝕後物體會消失，如果物體之間有小於結構元素的細小連通，侵蝕後會分裂成兩個物體。



膨脹為擴大物體的邊界，而擴大的寬度是由結構元素大小決定的，如果物體間有小於結構元素的細小間隙，膨脹能讓原本分開的物體連接起來。



* 1. 函數程式碼(函數寫在mask.c)

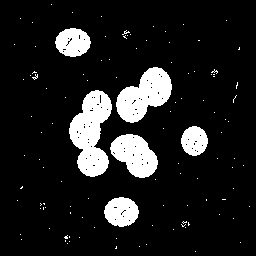
dilation (輸出入矩陣，圖長，圖高)

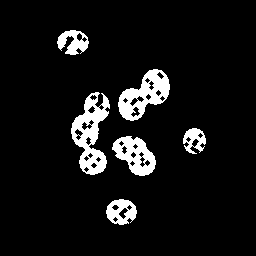
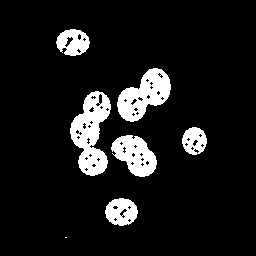
|  |
| --- |
| void dilation(int R[1024][1024], int width, int height) {  int temp[1024][1024] = { 0 };  for (int x = 1;x < width - 1;x++)  for (int y = 1; y < height - 1;y++) {  if (R[x][y] == 255){  temp[x - 1][y] = 255;  temp[x][y] = 255;  temp[x + 1][y] = 255;  temp[x][y - 1] = 255;  temp[x][y + 1] = 255;  }  }  for (int x = 0;x < width;x++)  for (int y = 0; y < height;y++)  R[x][y] = temp[x][y];  } |

dilation (輸出入矩陣，圖長，圖高)

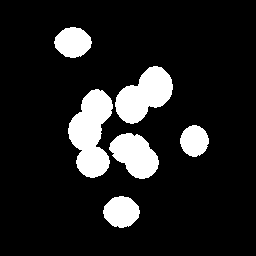
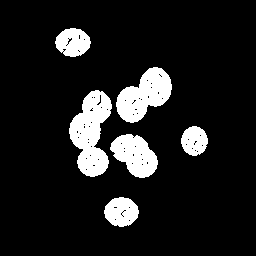
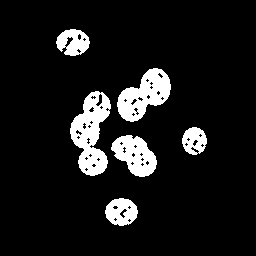
|  |
| --- |
| void erosion(int R[1024][1024], int width, int height) {  int temp[1024][1024] = { 0 };  for (int x = 1;x < width - 1;x++)  for (int y = 1; y < height - 1;y++)  if (R[x - 1][y] == 255 && R[x][y] == 255 && R[x + 1][y] == 255 &&  R[x][y - 1] == 255 && R[x][y + 1] == 255)  temp[x][y] = 255;  for (int x = 0;x < width;x++)  for (int y = 0; y < height;y++)  R[x][y] = temp[x][y];  } |

* 1. 結果圖與原圖比較

****



原圖erosion erosion



Dilation dilation dilation

1. 心得

 第一題只要參考rgb轉換成hsv的公式即可，比較困難的是找出膚色的範圍，一開始一直搞不懂為何頭髮會被納入範圍內，後來試過好幾次才知道把v限制範圍即可去除頭髮。

第二題只要使用老師提供的快速傅立葉轉換即可做出來，但因為一開始不知道虛數也要處理，所以一直跑出右邊的圖，幸好跟同學討論後才發現自己的錯誤，成功做出來。

第三題結構元素是3x3的十字，只要掌握好膨脹跟腐蝕的定義就能做出來，並不是太困難。

修完影像處理後，對於這方面的專有名詞都有一些概念了，過去玩Matlab跟Opencv時，對於函數名稱皆是有看沒有懂，但之後若有需要時，只要打這些關鍵字即可在網路上得到資訊。