Group members:蔡宇軒 b02202020、徐宇霆 b02202050、林裕洲 b02901064

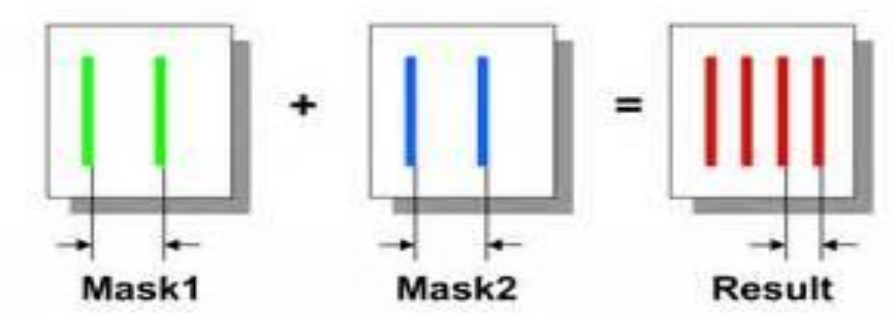
EDA 期末專題報告

Project: Color Balancing for Double Patterning

From: ICCAD 2015 Contest

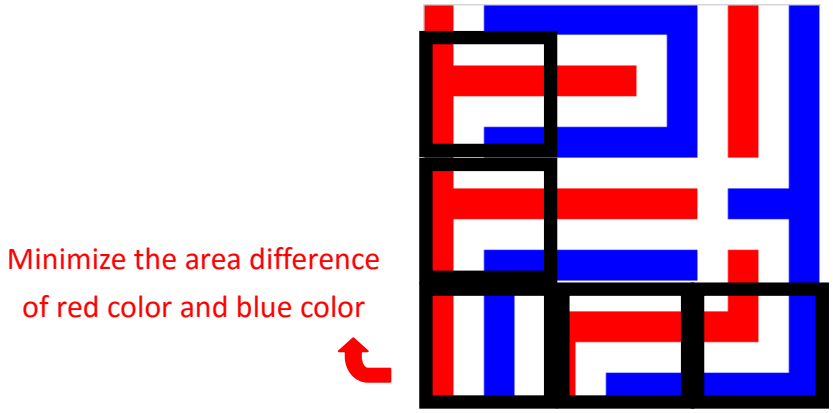
A.Color Balancing 目的:

隨著科技的發展，對於 IC 晶片面積的要求也愈來愈小，有些晶片甚至需要到奈米等級的大小，然而在利用光罩進行電路設計時，卻必須考量到光學的極限，也就是當線路寬度小於光波長的一半時就無法順利成像，為了突破這種困境有人就提出一種簡單的方法，就是將電路設計分成兩部份去製造光罩，如下圖:



這樣就能夠大幅減低因為線路設計過窄而導致的成像扭曲，甚至可以設計成好幾層的製成，但相對的成本就會上升。

我們挑選的專題就是將這類型的線路設計問題簡化成 double patterning 問題，也就是只用兩種顏色來分配，由於實際設計的線路面積可能比光罩還要大，因此需要將線路分割成好幾個區塊來實行，這時我們會希望**每個分割區域兩種顏色所佔的比例愈相近愈好**，也就是最終目標需要達成 color balancing。



B.程式實作:pseudo code

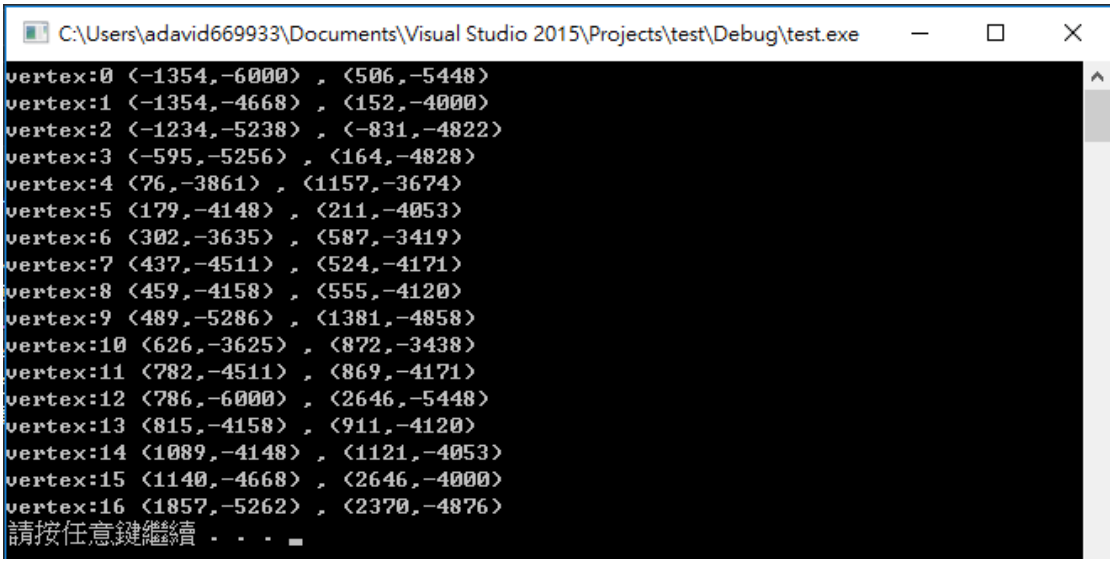
1. Parse :利用主辦單位給的測資，首先題目會給我們許多長方體的座標，我們可以將每一個長方體視作一個 vertex，不同的 vertex 用 struct vertex 來儲存，讀取測資的 x 和 y 座標分別存在 struct rect 裡，顏色則是分成紅色和綠色兩種，存在 enum color 裡，另外我還特別做了一個 struct adjacent_vertex，是為了之後做 Link List 用的。

```
struct vertex {
        int vertex_number = -1;                  //vertex 的編號,initial 設成-1
        rect position;                           //vertex 的座標
        color shape_color = initial;             //vertex 的顏色,initial 設成-1
        int graph = -1;                          //graph 的編號,initial 設成-1
        adjacent_vertex* adjacent = NULL;        //Link List 的 ptr,initial 設成 NULL
};
```

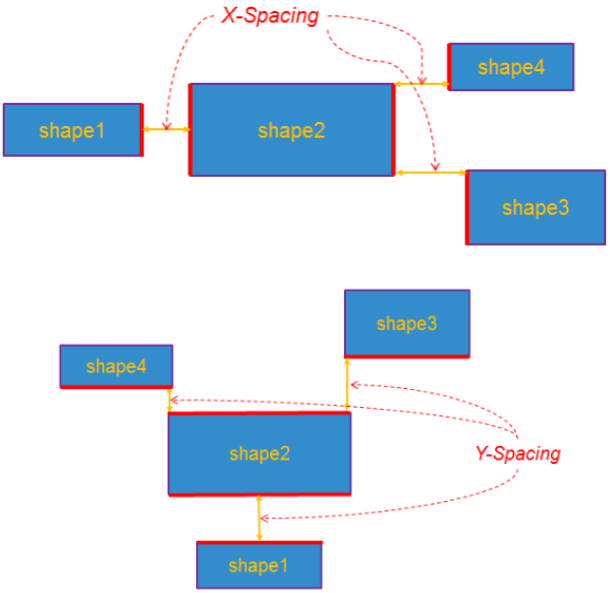
```
struct rect {
        int x1;
        int y1;
        int x2;
        int y2;
};
```

```
enum color{
        banned = -2;
        initial = -1;
        red = 0;
        green = 1;
};
```

```
struct adjacent_vertex {
    int vertex_number = -1;
     color shape_color = initial;
    int graph = -1;
    adjacent_vertex* adjacent = NULL;
};
```



2. Construct Link List :測資同時會給我們分辨不同 vertex 是不是不同長方體的最短距離，對於 x 座標和 y 座標分別給定一個最短距離 α 和 β，如果長方體和長方體邊長之間的 x 座標或是 y 座標小於給定的最短距離，則可以將 vertex 和 vertex 連起來，如下圖:



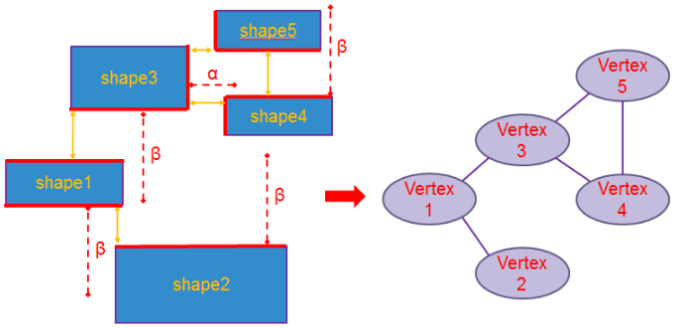
主要方法: 跑過所有的 vertex，利用 Construct_Link_List function 檢查任兩個 vertex 是否相連，如果有相連就將其放入原先 vertex 的 adjacent_vertex 的 Link List 裡面。

Pseudo code:

```
1. i , j = 0;
2. for (i < array_size ; i++) {       //所有的 vertex 都要找過一次
3.     for (j <array_size ; j++) {    //因此有相連的 vertex，都會互相連接一次
4.           number = Construct_Link_List(i,j);
5.                                    //利用 Construct_Link_List 找出與 i 相連的 vertex
6.           vertex(i).adjacent_vertex=adjacent_vertex(number);
7.                                    //將找到的 vertex 用 vertex(i)的 adjacent_vertex 連起來
8.     }
9.}
```

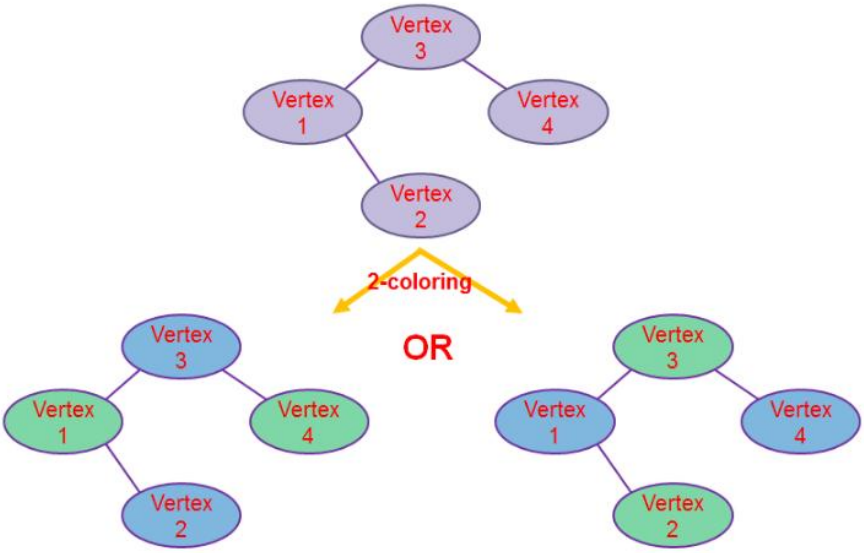
```
1. int Construct_Link_List(i,j) {
2.       test whether the x distance of vertex(i) and vertex(j)< α;
3.       test whether the y distance of vertex(i) and vertex(j)< β;
4.       return j;                   //回傳有跟 vertex(i)相連的 vertex number
5. }
```

連起來的結果就會如下圖所示:

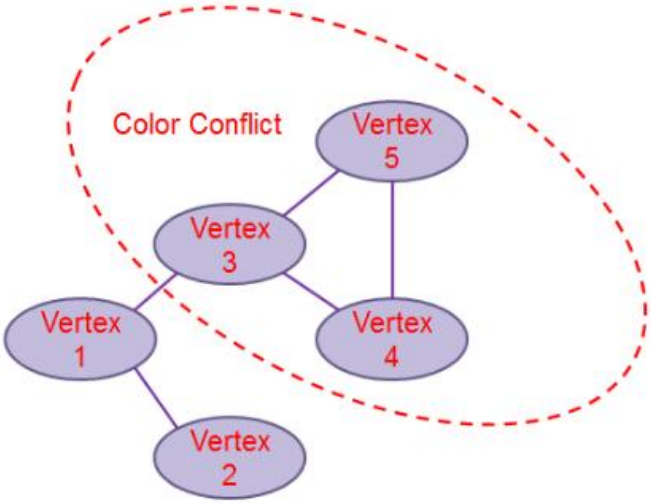


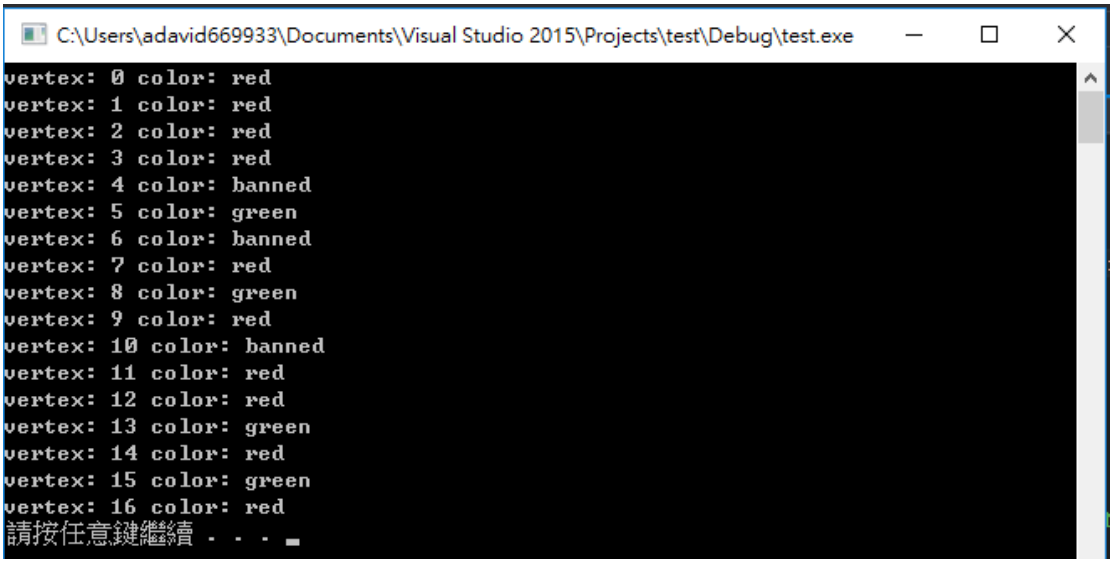

```
C:\Users\adavid669933\Documents\Visual Studio 2015\Projects\test\Debug\test.exe
vertex: 0 adjacent:
vertex: 1 adjacent: 5
vertex: 2 adjacent:
vertex: 3 adjacent:
vertex: 4 adjacent: 6 10
vertex: 5 adjacent: 1
vertex: 6 adjacent: 4 10
vertex: 7 adjacent: 8
vertex: 8 adjacent: 7
vertex: 9 adjacent:
vertex: 10 adjacent: 4 6
vertex: 11 adjacent: 13
vertex: 12 adjacent:
vertex: 13 adjacent: 11
vertex: 14 adjacent: 15
vertex: 15 adjacent: 14
vertex: 16 adjacent:
請按任意鍵繼續 . . .
```

3. Assign Color :這時候會形成大大小小的 graph，分析每一個 graph 是否有奇數個 vertex 所形成的 cycle，利用兩種顏色去 assign 每一個 vertex，相鄰的 vertex 顏色不能一樣，因此奇數個點的所形成的 cycle 無法被 assign 顏色。



主要方法: 跑過所有的 vertex，並且呼叫 SetColor function，這個 function 會稍微複雜一點，首先他會給定原先的 vertex 固定的顏色(red)，這時 vertex 連接的 adjacent_vertex Link List 就會給予相反的顏色(green)，對於所有的 adjacent_vertex 會再重新呼叫一次 SetColor function，不斷遞迴，如果遇到牴觸的顏色就呼叫 SetBanned function，這個 function 會將 vertex 以及所有連接到的 adjacent_vertex Link List 的顏色都設為 banned，直到所有 vertex 都被 assign 完就結束。





Pseudo code:

```
1.for (i < array_size ; i++) {                    //所有的 vertex 都要執行過一次
2.    if (vertex(i).shape_color == initial) {
3.          SetColor(vertex(i), initial);         //呼叫函數
4.    }
5.}
```

```
1.void SetColor(vertex(i),color) {
2.          //這個 function 吃兩個 input 分別是 vertex 以及 adjacent_vertex 的顏色
3.    if (vertex(i).shape_color == initial&&color==initial) {
4.          // 情況一. 整條 LinkList 都還沒 SetColor 的時候
5.          vertex(i).shape_color = red;
6.          while (adjacent_vertex != NULL) {       //如果 Link List 還有未連接的 vertex
7.                adjacent_vertex.shape_color = green;
8.                SetColor(adjacent_vertex,green);     //遞迴呼叫
9.          }
10.   }
11.   else if (vertex(i).shape_color == initial&&color == red) {
12.         // 情況二. 從 LinkList 的 adjacent 傳下來的顏色==red
13.         vertex(i).shape_color = red;
14.         while (adjacent_vertex != NULL) {       //如果 Link List 還有未連接的 vertex
15.               adjacent_vertex .shape_color = green;
16.               SetColor(adjacent_vertex,green);     //遞迴呼叫
17.         }
18.   }
19.   else if (vertex(i).shape_color == initial&&color == green) {
20.         // 情況三. 從 LinkList 的 adjacent 傳下來的顏色==green
21.         vertex(i).shape_color = green;
22.         while (adjacent_vertex != NULL) {       //如果 Link List 還有未連接的 vertex
23.               adjacent_vertex .shape_color = red;
24.               SetColor(adjacent_vertex , red);     //遞迴呼叫
25.         }
26.   }
27.   else if (vertex(i).shape_color == red&&color == red) {}
28.         // 情況四. ptr 和從 LinkList 的 adjacent 傳下來的顏色一樣==red
29.   else if (vertex(i).shape_color == green&&color == green) {}
30.         // 情況五. ptr 和從 LinkList 的 adjacent 傳下來的顏色一樣==green
31.   else if (vertex(i).shape_color == red&&color == green) {
32.         // 情況六. ptr 和從 LinkList 的 adjacent 傳下來的顏色不一樣
33.         SetBanned(vertex(i));}     //呼叫 SetBanned function
34.   else if (vertex(i).shape_color == green&&color == red) {
35.         // 情況七. ptr 和從 LinkList 的 adjacent 傳下來的顏色不一樣
36.         SetBanned(vertex(i));}     //呼叫 SetBanned function
37.   else {}
38.}
```

```
1.void SetBanned(vertex(i)) {
2.    if (vertex(i) .shape_color != banned) {
3.        vertex(i) .shape_color = banned;
4.        while (adjacent_vertex != NULL) {
5.            adjacent_vertex .shape_color = banned;
6.            SetBanned(adjacent_vertex );
7.        }
8.    }
9.    else {}
10.}
```

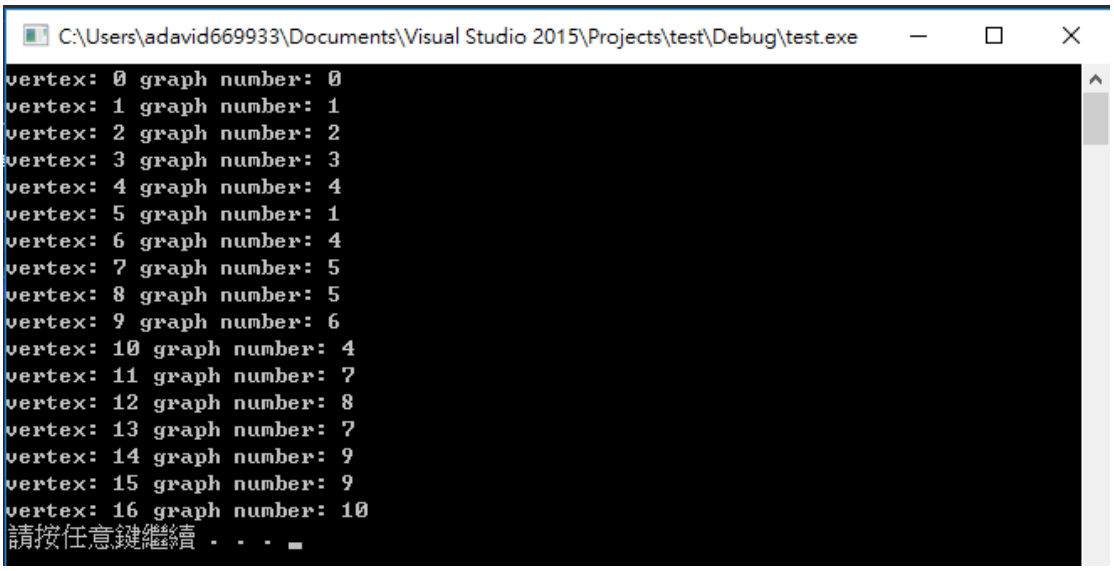
4. Assign Graph Number :

主要方法:跑過所有的 vertex，並且呼叫 SetGraph function，這個 function 會將所有沒被 assign graph number 的 vertex assign number，並且會呼叫在 vertex 底下的 adjacent_vertex 執行 function，確保相連的 vertex 有相同的 graph number。

Pseudo code:

```
1.graph , i = 0;
2.for (i<array_size;i++) {                    //所有的 vertex 都要執行過一次
3.    if (vertice(i).graph == -1) {
4.        SetGraph(vertice(i), graph); //呼叫函數
5.        graph++;
6.    }
7.}
```

```
1.void SetGraph(vertice(i), graph) {
2.    if (vertice(i) == -1) {
3.        vertice(i).graph = graph;
4.        while (adjacent_vertex != NULL) {
5.            adjacent_vertex .graph = graph;    //在 Link List 裡面的顏色都要一樣
6.            SetGraph(adjacent_vertex, graph);
7.        }
8.    }
9.    else {}
10.}
```



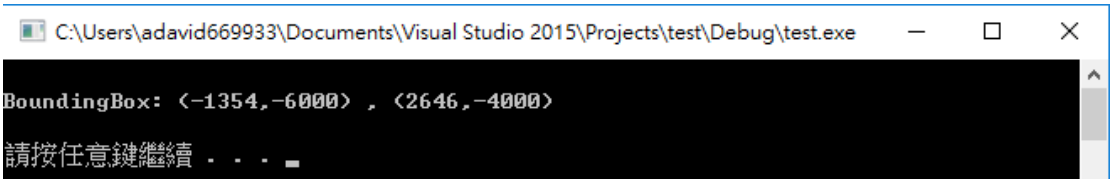
5. Find Color Bounding Box :

最後再用一個大長方體把所有有被 assign 顏色的 vertex 圈起來。

主要方法:跑過所有的 vertex，不斷比較 vertex 的邊界，並且檢查 vertex 是否有被上色，最後找出 Color Bounding Box 的 x , y 座標。

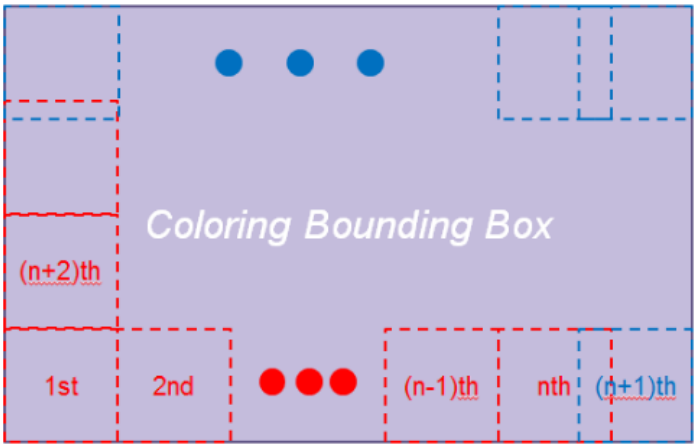
Pseudo code:

```
1.rect BoundingBox;
2.int initial = 0;
3.for (int i = 0;i < array_size;i++) {
4.    if (vertice(i).position.x1 < x1&&vertice(i).shape_color != banned) {
5.          x1 = vertice(i).position.x1;}
6.    if (vertice(i).position.y1 < y1&&vertice(i).shape_color != banned) {
7.          y1 = vertice(i).position.y1;}
8.    if (vertice(i).position.x2 > x2&&vertice(i).shape_color != banned) {
9.          x2 = vertice(i).position.x2;}
10.   if (vertice(i).position.y2 > y2&&vertice(i).shape_color != banned) {
11.         y2 = vertice(i).position.y2;}
12.}
13.   BoundingBox.x1 = x1;
14.   BoundingBox.y1 = y1;
15.   BoundingBox.x2 = x2;
16.   BoundingBox.y2 = y2;
```



6. Compute Color Density Window Position:

題目還會給定一個正方形的邊長 ω，將這個正方形鋪滿圍出來的大長方體，分析每一個正方形的位置。



Pseudo code:

rect: current rectangle

w: window

q: quantity

p: position

```
1.rect = (0, 0, OMEGA, OMEGA)
2.for (int i = 0; i<w_q; i++) {
3.    w_p[i] = rect;
4.    rect.x += omega;
5.    if (rect.out_of _bounding_box) {
6.          rect.x = 0
7.          rect.y += OMGEA;
8.    }
9.}
```

7. Calculate Area Matrix:

將每一個 graph 在每一格 Color Density Window 的紅色以及綠色面積比例轉換成矩陣的方式來運算，矩陣相乘完的結果就是所有 Color Density Window 的顏色比例差值。

Pseudo code:

area_matrix 即 c_w_g。c_w_g[i][j]表示第 j 個 graph 在第 i 個 windows 中所佔有的紅色面積減去綠色面積(此即為下一部份所說的 differential area)

c: cross area

w: window

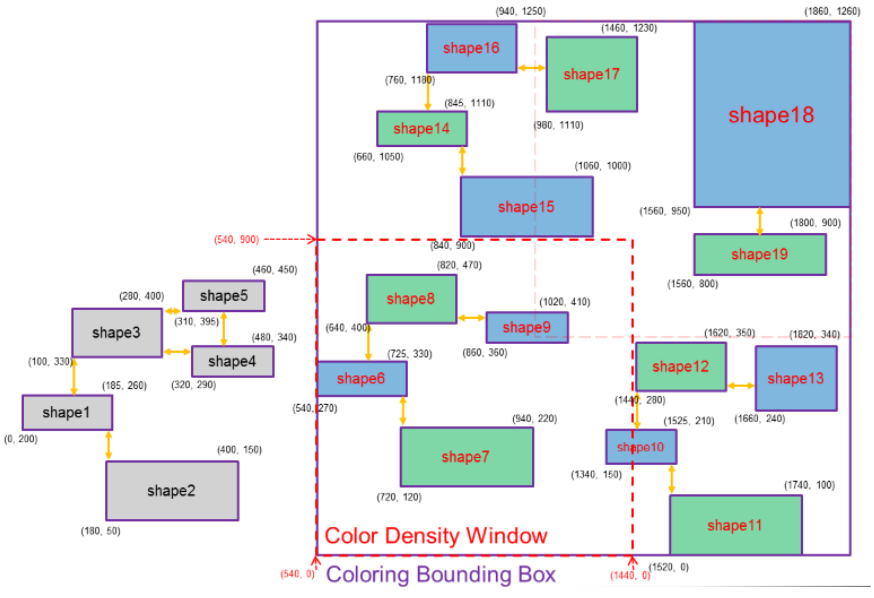
g: graph

q: quantity

v: vertex

cl: color

```
1.c_w_g[][] = 0;
2.for (int i = 0; i<w_q; i++) {
3.    for (int j = 0; j<v_q; j++) {
4.          calculate c_w_v[i][j];
5.          if (v_cl[j] == red)
6.                c_w_g [i][v_g[j]]+= c_w_v[i][j];
7.          else if (v_cl[j] == green)
8.                c_w_g[i][v_g[j]] -= c_w_v[i][j];
9.    }
10.}
```



8. Used Simulated Annealing to draw color for highest score:

利用 **Simulated Annealing** 演算法改變每一個 graph 的顏色分配，使得每一個正方形內兩種顏色所占總面積的比例相當。

Pseudo code:

Simulated Annealing 簡易實作方法:

```
for (1~500) {
    temperature = 2;
    for (each graph)
        randomly draw colors;
    for (1~30) {
        temperature *= 0.9;
        for (each graph) {
            change color;
            calculate score;
            cost = last score - score;
            possibility = exp(-cost / temperature);
            if (possibility > rand(0, 1))
                retain changed color;
            else
                recover unchanged color;
        }
        update the best solution;
    }
    update the best solution;
}
```

實際實作方法:

init: initial
t: temperature
r: temperature_reduction_ratio
s: score
a: just a variable
d: differential area of two color
p: possibility

```
1.for (int i = 0; i<500; i++) {
2.    for (int j = 0; j<g_q; j++)
3.        g_cl[j] = rand(1, -1);
4.    t = 2;
5.    r = 0.9;
6.    s = 0;
7.    for (int j = 0; j<w_q; j++)
8.        w_d[j] = 0;
9.    for (int k = 0; k<g_q; k++)
10.       w_d[j] += c_w_g[j][k] * g_cl[k];
11.   s -= abs(w_d[j]);
12.   for (int j = 0; j<30; j++) {
13.       t *= r;
14.       for (int k = 0; k<g_q; k++) {
15.           last_w_d = w_d;
16.           last_s = s;
17.           s = 0;
18.           for (int l = 0; l<w_q; l++)
19.               w_d[l] += 2 * g_cl[k]c_w_g[l][k];
20.           s -= abs(w_d[j]);
21.           p = exp((s - last_s) / t);
22.           if (p>rand(0~1)) {
23.               g_cl[j] *= -1;
24.           }
25.           else {
26.               s = last_s;
27.               w_d = last_w_d;
28.           }
29.           if ((i == 0 && j == 0) || best_s<s) {
30.               best_s = s;
31.               best_g_cl = g_cl;
32.           }
33.       }
34.   }
35.}
```

C.實作結果:

Simulated annealing for hundreds of times with random initial condition

首先，我們定義 cost function 為分數變化乘以-K，其中 K 為一正數。由於 simulated annealing 中機率是 cost/temperature 的函數，因此任意選定 K 值並不喪失一般性，而 temperature 的決定法如下。我們希望在 cost function>0 的情形中，機率的數量級大約在 10%~50%之間，不宜過大或過小，由此選定一個較好的初始溫度以及每次溫度下降的比率以符合預期。

由於 simulated annealing 本身由機率決定是否達到目標，假設在一次 simulated annealing 演算法後達到目標的機率期望值為 p，而在多次測試中，只要一次達到目標就算成功的話，那麼測試 n 次無法達到目標的機率為(1-p)^n。若我們增加 n 次 simulated annealing，那麼無法達到目標的機率是以等比級數下降，因此只要用線性的時間複雜度，就能將失敗機率降至很低，這也是 simulated annealing 厲害之處。因此我們經過多次實驗，選定 iteration 將最困難的題目失敗機率降至約 50%左右即可接受。

Only calculate the area change of the colored graph

我們使用三種層面的加速機制。

一， 如同 KL algorithm 一般，我們不重新計算全部的面積差異，而是將所有面積差異加上改變量就好。如果 graph 的數量有 n 個的話，那麼將可以降低面積差異計算時間約 n 倍。

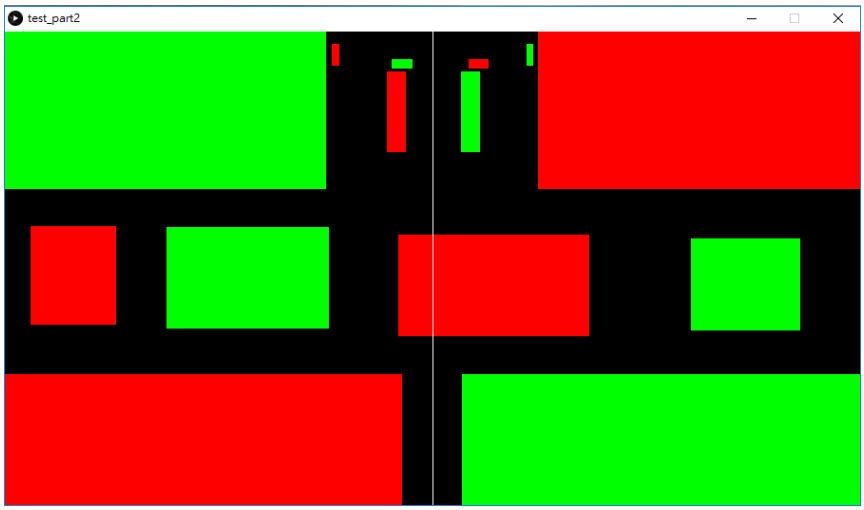
二， 在讀取資料時，由於 shape 數量不固定，因此需要用 vector 資料結構來儲存 shape 的位置，但在需要大量運算的 simulated annealing 之前，我們將最耗時的運算資料由 vector 轉換成 array，實驗後發現產生大幅的加速效果。

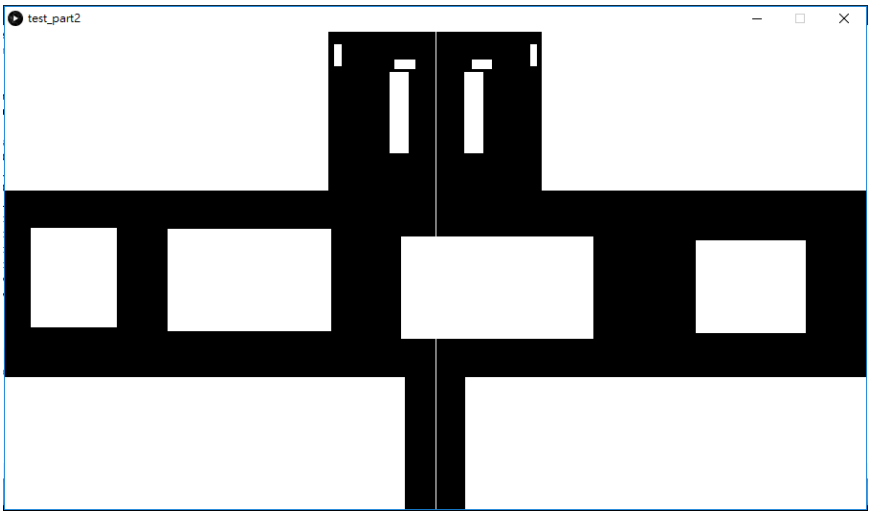
三， 某些現代的電腦配備獨立顯卡，尤其以 NVIDIA 公司的產品為大宗。我們下載其 GPGPU(general-purpose GPU)的套件 CUDA，由於他與普通 g++編譯器不同，會使用 GPU 記憶體，因此我們也能從中得到不少幅度的加速效益。

Processing is written to check the result

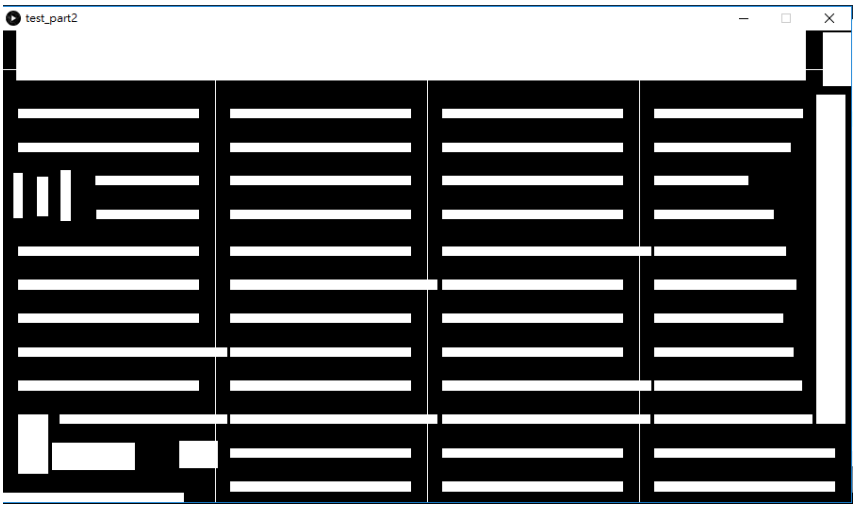
為了確保我們的結果正確，我們使用 java based 的繪圖程式 processing 來呈現我們的運算結果。

Testcase1

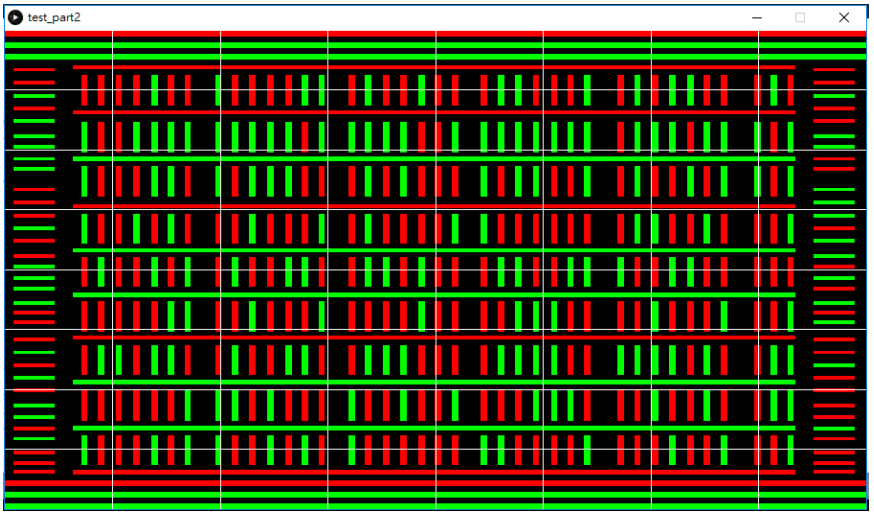


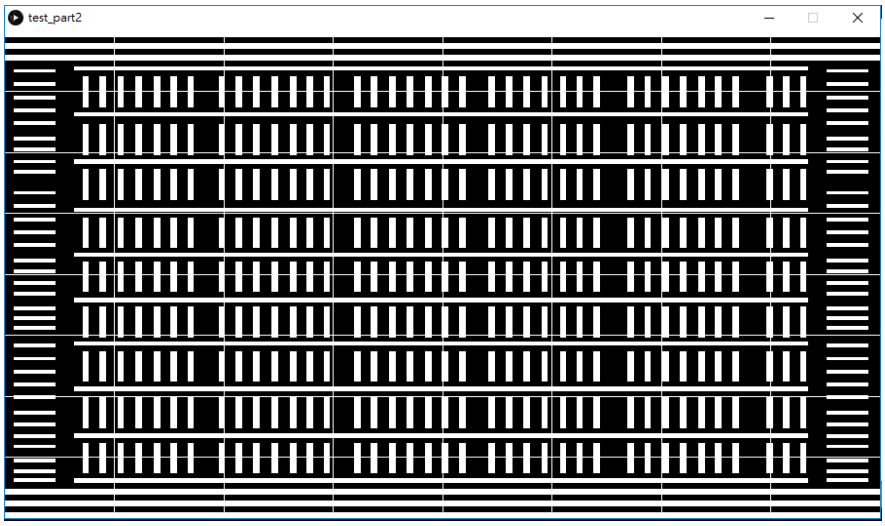


Testcase2

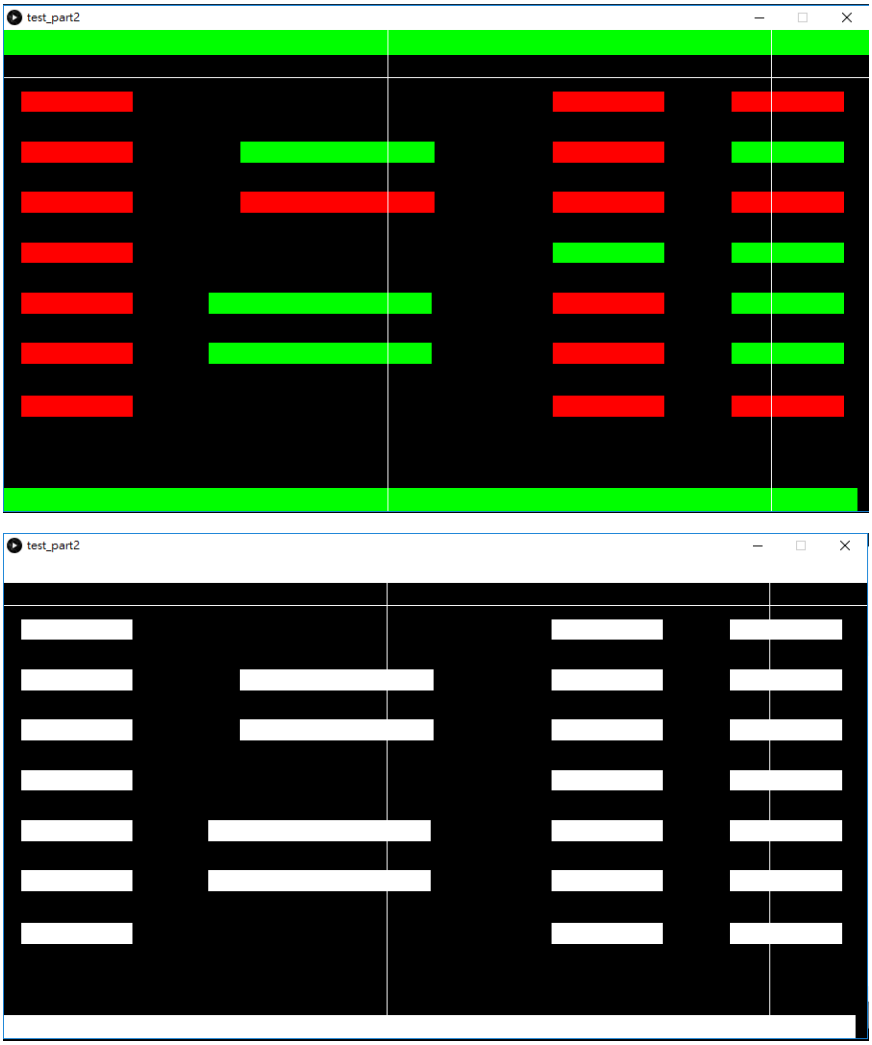


Testcase3

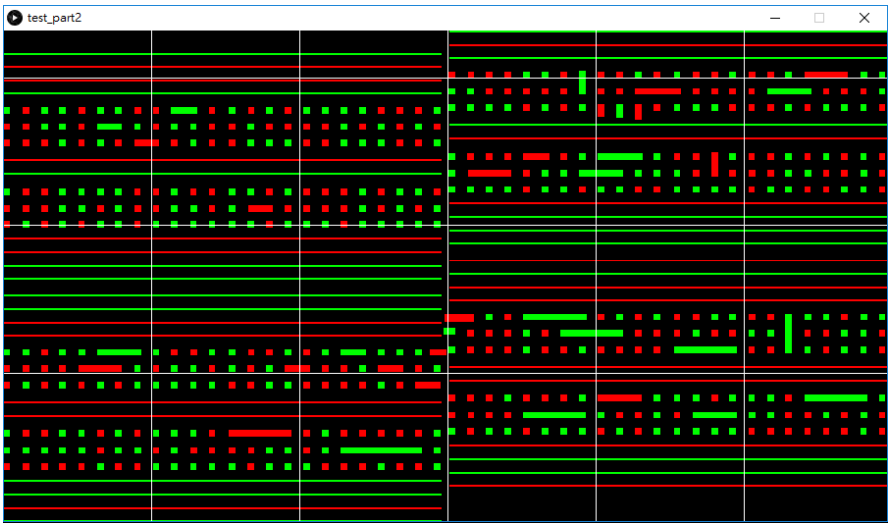


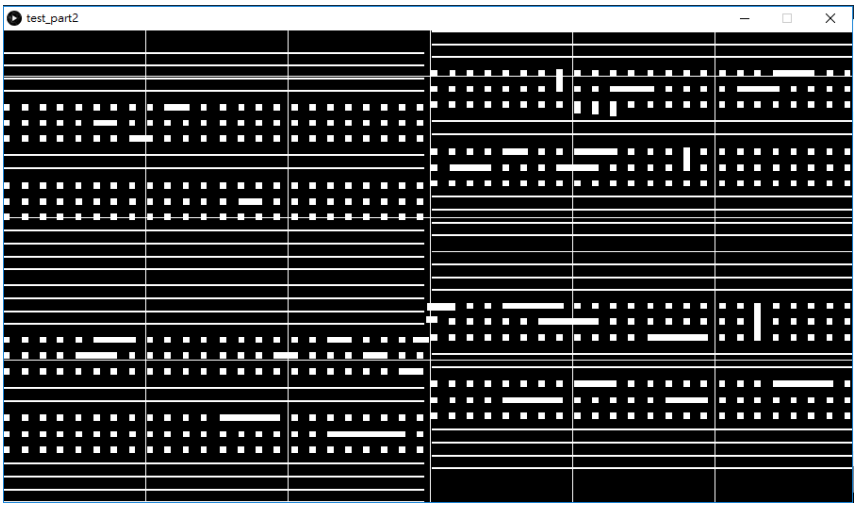


Testcase4



Testcase5





最後我們拿主辦單位的 beta_test 來進行測試與評分，並且和競賽前幾名來比較，直接進行隨機一次性的測試我們的分數是第四名。

1	2	3	4	5		avg	new order	original
99.46	95.07	91.28	99.78	99.77		97.072	1	1
99.46	95.07	91.34	99.78	99.67		97.064	2	5
99.46	95.07	91.12	99.78	99.69		97.024	3	2
99.46	95.07	90.4	99.78	99.61	ya	96.864	4	
99.46	94.7	90.45	99.78	99.7		96.818	5	4
99.46	95.07	89.04	99.78	99.69		96.608	6	3
99.23	95.07	89.9	99.27	99.57		96.608	7	10
99.46	94.98	89.18	99.63	99.59		96.568	8	8
99.46	95.07	88.67	99.78	99.66		96.528	9	6
99.46	95.07	87.86	99.78	99.65		96.364	10	7
99.46	95.07	87.05	99.78	99.52		96.176	11	9

D.Reference:

1. ICCAD 題目來源:http://cad-contest.el.cycu.edu.tw/problem_E/default.htm

2. K.M. Monahan, “Enabling Double Patterning at the 32nm Node”, Semiconductor Manufacturing 2006 IEEE International Symposium, pp.126-129, ISBN 978-4-9904138-0-4.

3. http://www.cs.cornell.edu/Courses/cs3110/2009sp/recitations/rec22.html

4. Alfred K. Wong. Resolution Enhancement Techniques in Optical Lithography. SPIE Publications, 2001.