2019 MMILAB.DIP Seminar Week4(1/30~2/5)

Seong Su Kim

Contents

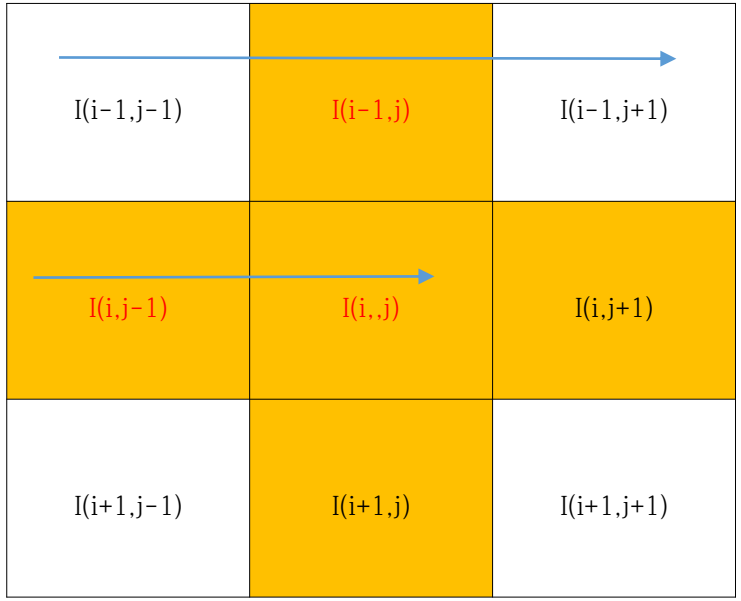
- 4,8 connected component labeling(CCL)
- Union find algorithm
- Coloring
- Effective way and Meaningful result
- Questions

Focus of this week

Comparison between algorithm

How to enhance performance??

4-CCL

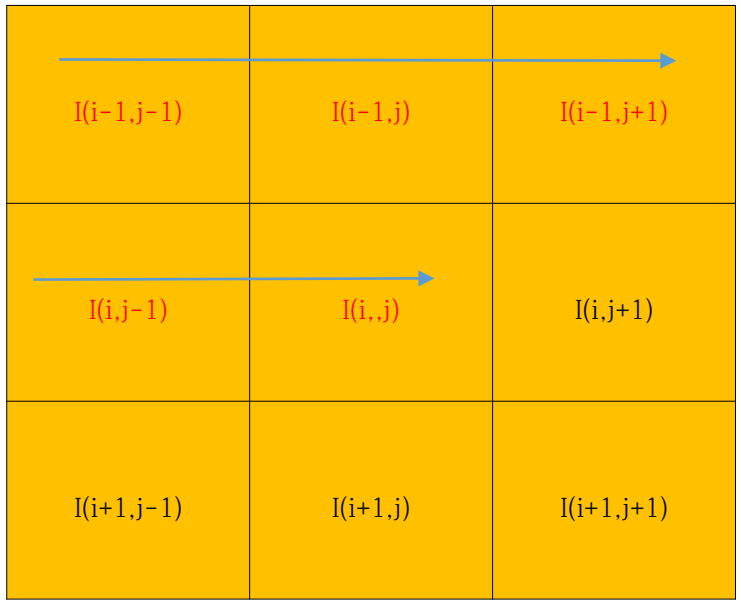


If I(i-1,j) and I(i,j-1) are both connected,

: choose one of them (min value is better)

Append I(i-1,j) and I(i,j-1) on list, to decide representative label afterward

8-CCL

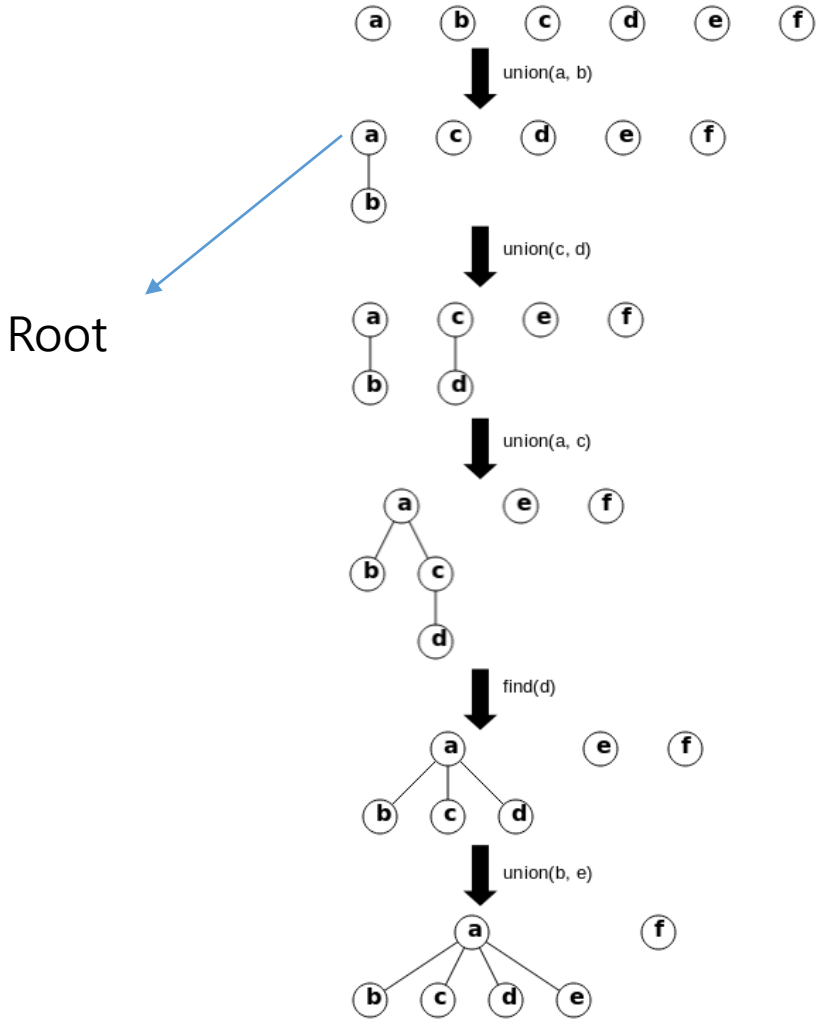


When more than two pixels are connected,

: choose one of them (min value is better)

Append connected components on list, to decide representative label afterward

Union find


a b c d e f
union(a, b)
a c d e f
b
union(c, d)
a c e f
b d
union(a, c)
a e f
b c
d
find(d)
a e f
b c d
union(b, e)
a f
b c d e
Root

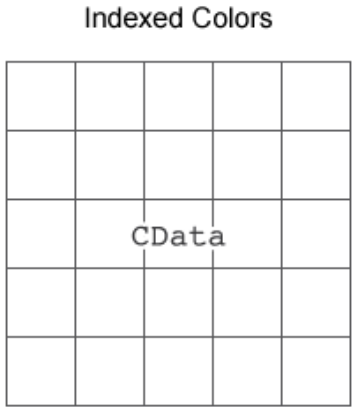
Coloring

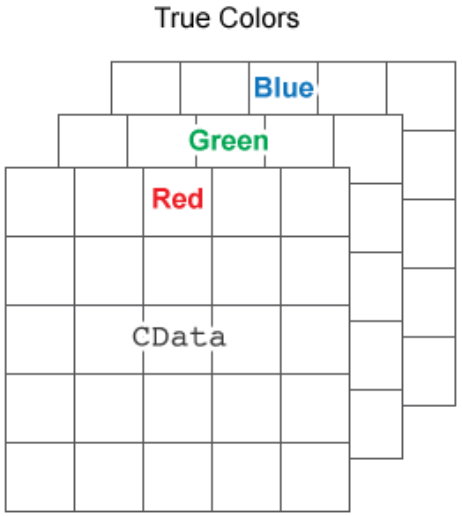
After label resolving

1. Sort label value (ex: [1,3,19,29….])

2. Divide each Label value's index by constant value and get remainder
(ex: constant =5, [0,1,2,3,4,5,0,1,2…])

3. Assign different color for each pixel's remainder

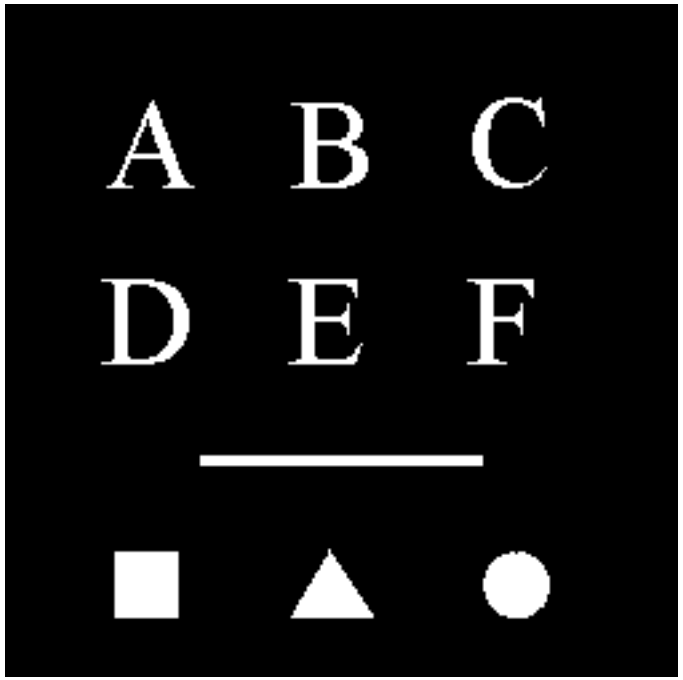


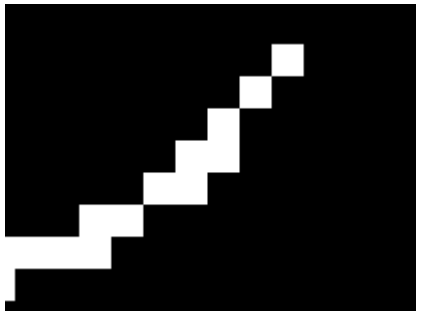


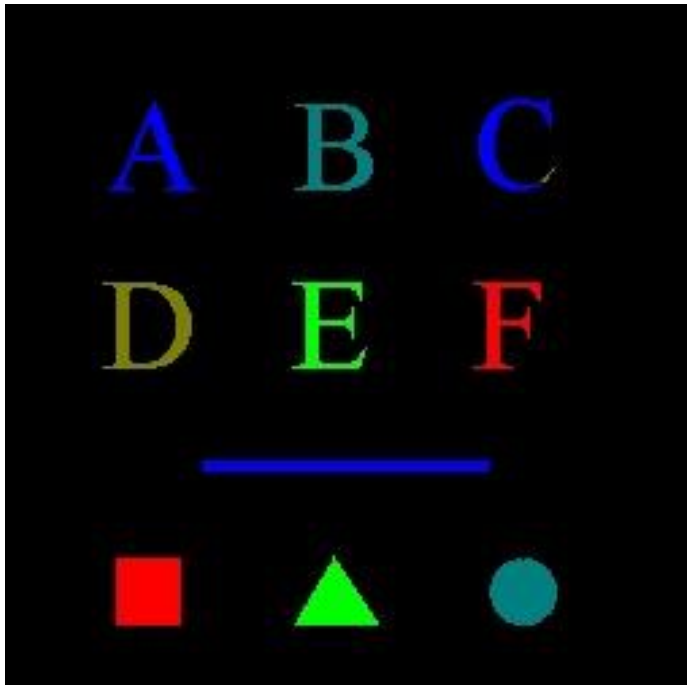
Convert gray image to color image

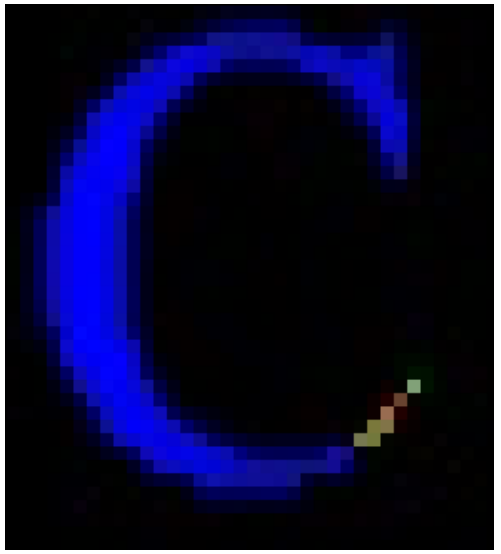
1. make three image size of zero array (for 3 color channel, initialize)
2. Stack 3 zero array
3. Assign color (R,G,B) value to each array for each case

Difference between 4-CCL 8-CCL

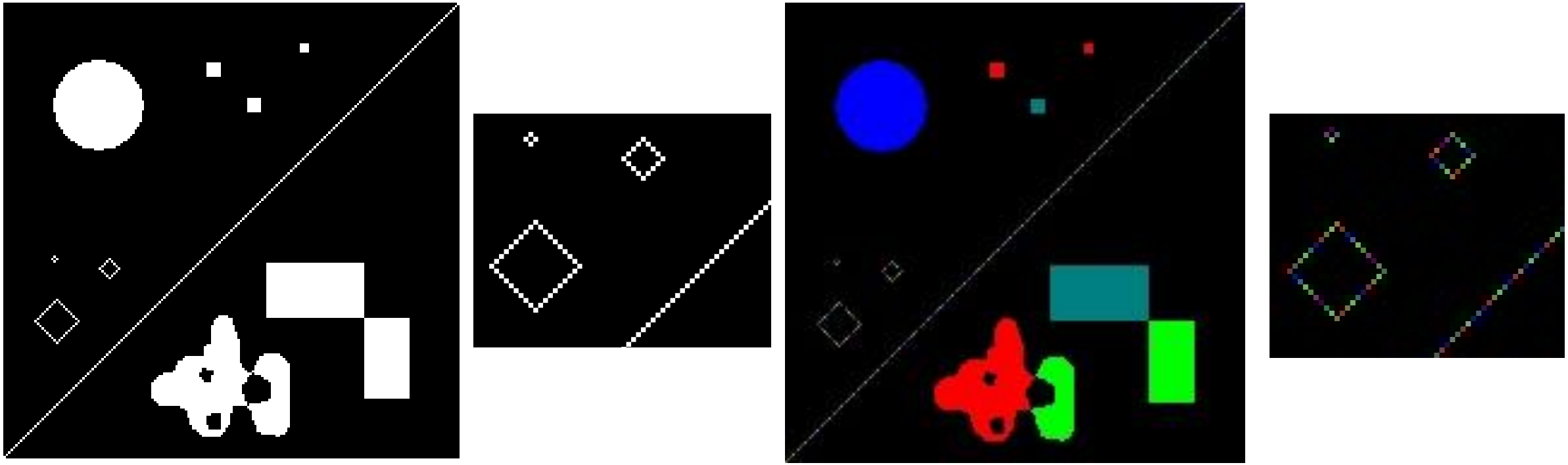




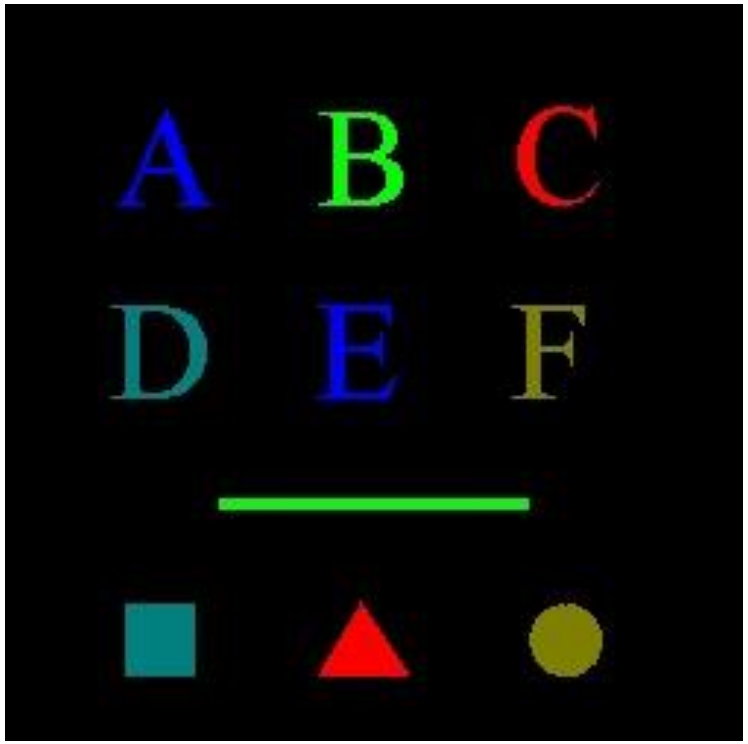




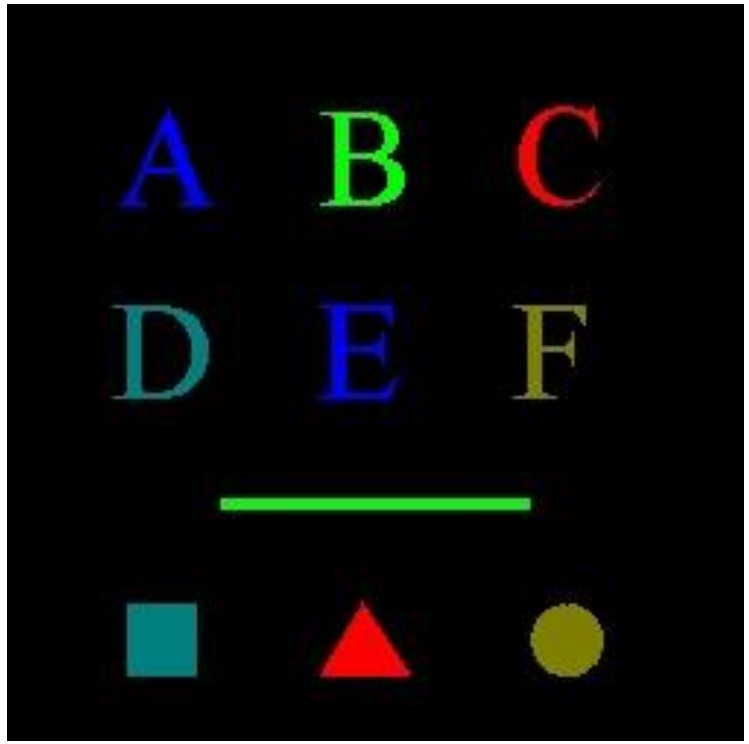
Difference between 4-CCL 8-CCL



Difference between 4-CCL 8-CCL



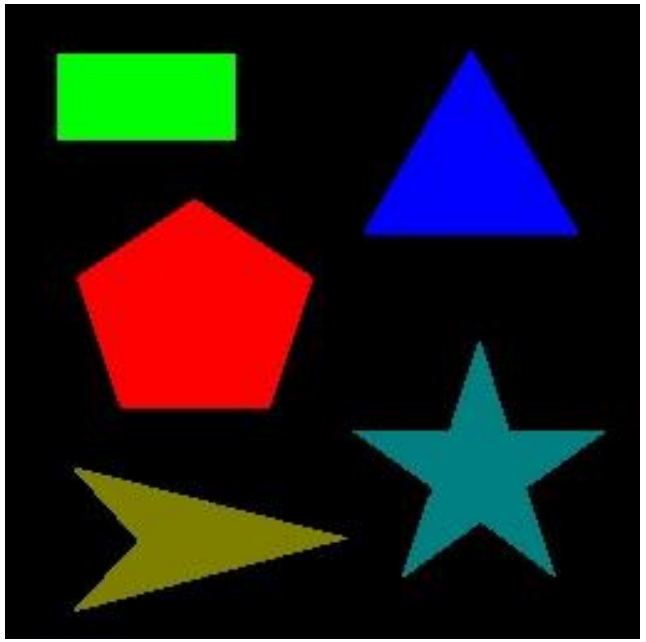
Connected component pixel counting



Pixel counting

A : Label =3, 260 pixels
B : Label =5, 400 pixels
C : Label =8, 251 pixels
D : Label =15, 396 pixels
E : Label = 20, 301 pixels
F : Label = 19, 260 pixels
— : Label = 27, 424 pixels
□ : Label = 29, 408 pixels
△ : Label = 28, 600 pixels
○ : Label = 31, 490 pixels

Connected component pixel counting



Pixel counting

□ : Label = 2, 2520 pixels
△ : Label = 1, 3298 pixels
pentagon : Label = 3, 5636 pixels
▷: Label = 21, 2532 pixels
☆: Label = 19, 3240 pixels

Effective way (for 8CCL)

1. Reducing operation (resolve, zero pixel)

2. Reducing conditional(=if) , loop iteration(=for)

- How to efficiently access to image pixel?

3. Memory access

4. Union find

Reducing operation(resolve)

I(i-1,j-1)	I(i-1,j)	I(i-1,j+1)
I(i,j-1)	I(i,j)	

1. No min() operation

2. Reducing resolve operation

Reducing operation(resolve)

Fast connected-component labeling
L He, Y chao, K Suzuki, K wu – Pattern recognition,2009- Elsevier

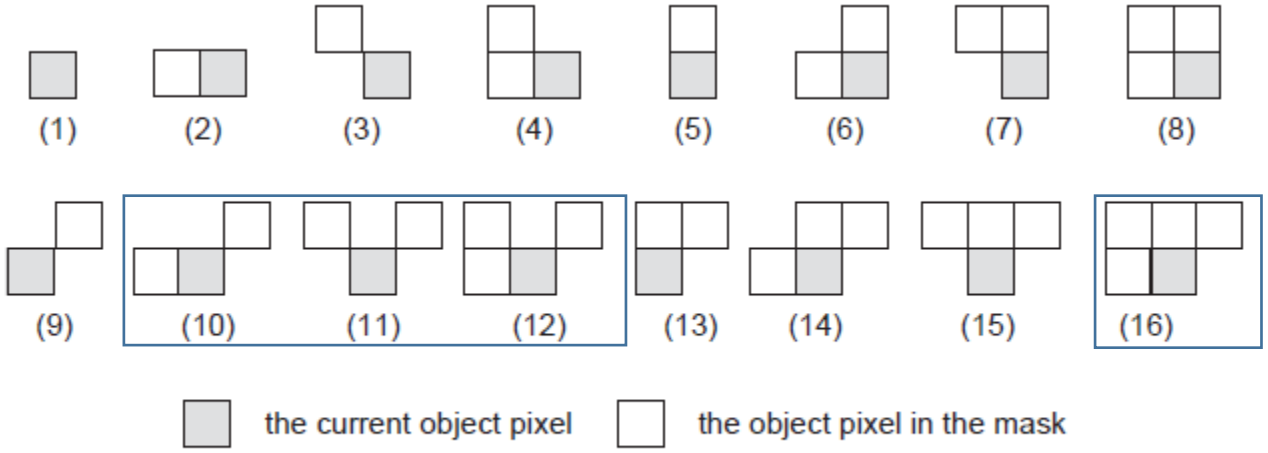
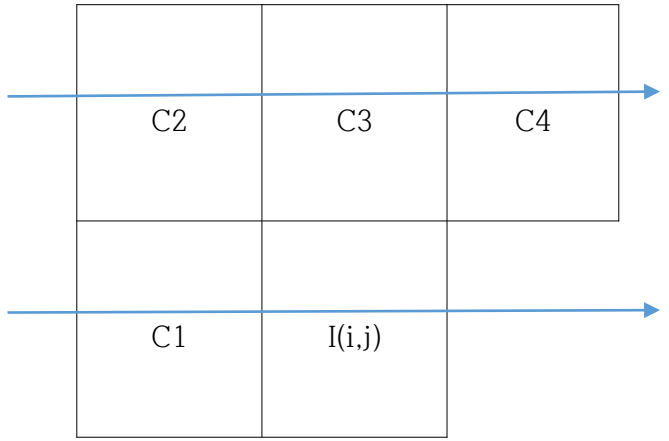


Fig. 3. Sixteen possible cases for the current object pixel in the mask for eight-connected connectivity.



The condition under which resolve does not take place

: C3 +/C4 +/C1*/C2

Reducing operation(resolve)

Fast connected-component labeling
L He, Y chao, K Suzuki, K wu – Pattern recognition,2009- Elsevier

Table 1
Operations in the sixteen cases

Case	c_4	c_3	c_2	c_1	$b(x,y)$	Operations
(1)	0	0	0	0	m	$m=m+1$
(2)	0	0	0	1	c_1	No operation
(3)	0	0	1	0	c_2	No operation
(4)	0	0	1	1	c_1 or c_2	No operation
(5)	0	1	0	0	c_3	No operation
(6)	0	1	0	1	c_1 or c_3	No operation
(7)	0	1	1	0	c_2 or c_3	No operation
(8)	0	1	1	1	c_1, c_2, or c_3	No operation
(9)	1	0	0	0	c_4	No operation
(10)	1	0	0	1	c_1 or c_4	*resolve*(c_1,c_4)
(11)	1	0	1	0	c_2 or c_4	*resolve*(c_2,c_4)
(12)	1	0	1	1	c_1, c_2, or c_4	*resolve*(c_1,c_4) or *resolve*(c_2,c_4)
(13)	1	1	0	0	c_3 or c_4	No operation
(14)	1	1	0	1	c_1, c_3, or c_4	No operation
(15)	1	1	1	0	c_2, c_3, or c_4	No operation
(16)	1	1	1	1	c_1, c_2,c_3, or c_4	No operation

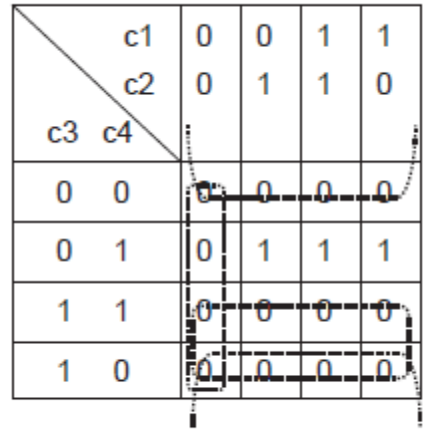


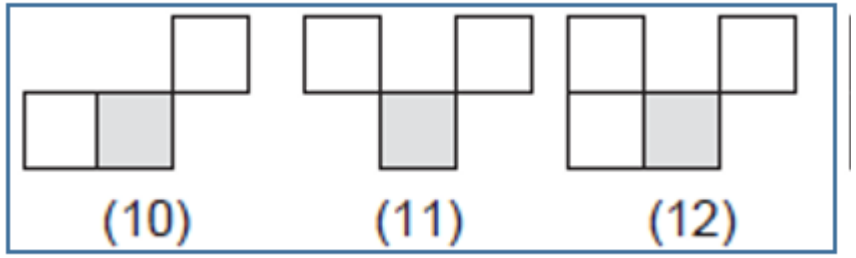
Fig. 4. The Karnaugh map for the operation *resolve*.

The condition under which resolve does not take place

: C3 +/C4 +/C1*/C2

Reducing operation(resolve)

Fast connected-component labeling
L He, Y chao, K Suzuki, K wu – Pattern recognition,2009- Elsevier



(10)	1	0	0	1	c_1 or c_4	*resolve*(c_1, c_4)
(11)	1	0	1	0	c_2 or c_4	*resolve*(c_2, c_4)
(12)	1	0	1	1	c_1, c_2, or c_4	*resolve*(c_1, c_4) or *resolve*(c_2, c_4)

Addtionally : if C1==C4, C2 ==C4 , no operation

Reducing operation(resolve)

Fast connected-component labeling
L He, Y chao, K Suzuki, K wu – Pattern recognition,2009- Elsevier

$$
\begin{aligned}
&\text{if } (c3 \neq V_B) \\
&\quad b(x,y) = c3; \\
&\text{else if } (c1 \neq V_B) \\
&\quad b(x,y) = c1; \\
&\quad \text{if } (c4 \neq V_B) \\
&\qquad resolve(c4, c1); \\
&\text{else if } (c2 \neq V_B) \\
&\quad b(x,y) = c2; \\
&\quad \text{if } (c4 \neq V_B) \\
&\qquad resolve(c2, c4); \\
&\text{else if } (c4) \\
&\quad b(x,y) = c4; \\
&\text{else} \\
&\quad b(x,y) = m,\ m = m + 1.
\end{aligned}
$$

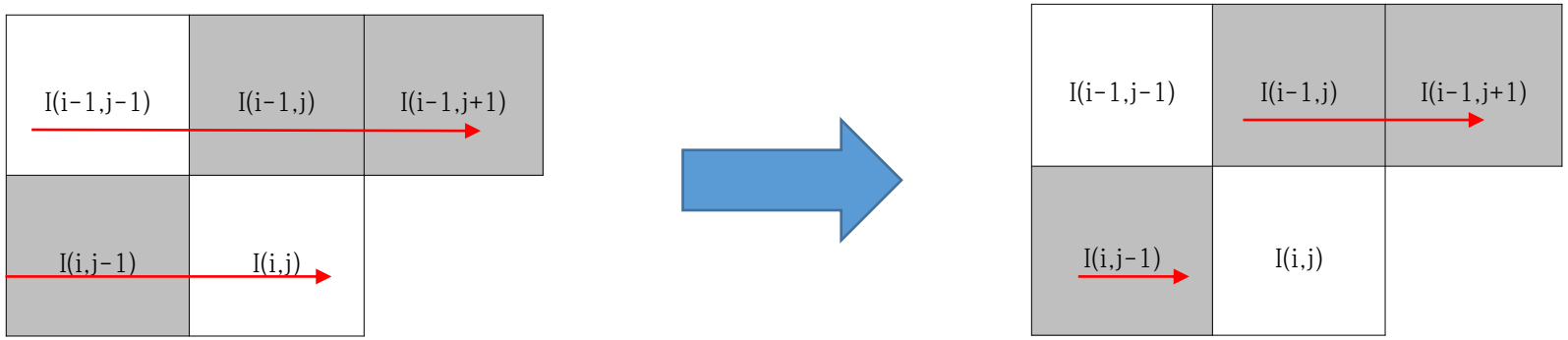
```
for i in range(y):
    for j in range(x):
            if img[i][j] != 0:
                if Labeled_img[i-1][j] !=0 :
                    Labeled_img[i][j] = Labeled_img[i-1][j]
                elif Labeled_img[i][j-1] !=0 :
                    Labeled_img[i][j] = Labeled_img[i][j-1]
                    if Labeled_img[i-1][j+1] != 0 :
                        pair_list = np.append(pair_list,[Labeled_img[i-1][j+1], Labeled_img[i][j-1]])
                elif Labeled_img[i-1][j-1] != 0 :
                    Labeled_img[i][j] = Labeled_img[i-1][j-1]
                    if Labeled_img[i - 1][j + 1] != 0:
                        pair_list = np.append(pair_list,[Labeled_img[i - 1][j + 1], Labeled_img[i - 1][j - 1]])
                #elif Labeled_img[i-1][j+1] !=0 and (Labeled_img[i-1][j-1], Labeled_img[i-1][j], Labeled_img[i][j-1])==(0,0,0) :
                elif Labeled_img[i - 1][j + 1] != 0 and Labeled_img[i - 1][j - 1] ==0 and Labeled_img[i - 1][j] == 0 and Labeled_img[i][j - 1] == 0:
                    Labeled_img[i][j] = Labeled_img[i-1][j+1]
                else :
                    Labeled_img[i][j] = cnt
                    cnt += 1
```

Reducing operation(zero pixel)

Don’t need to operate for zero pixel

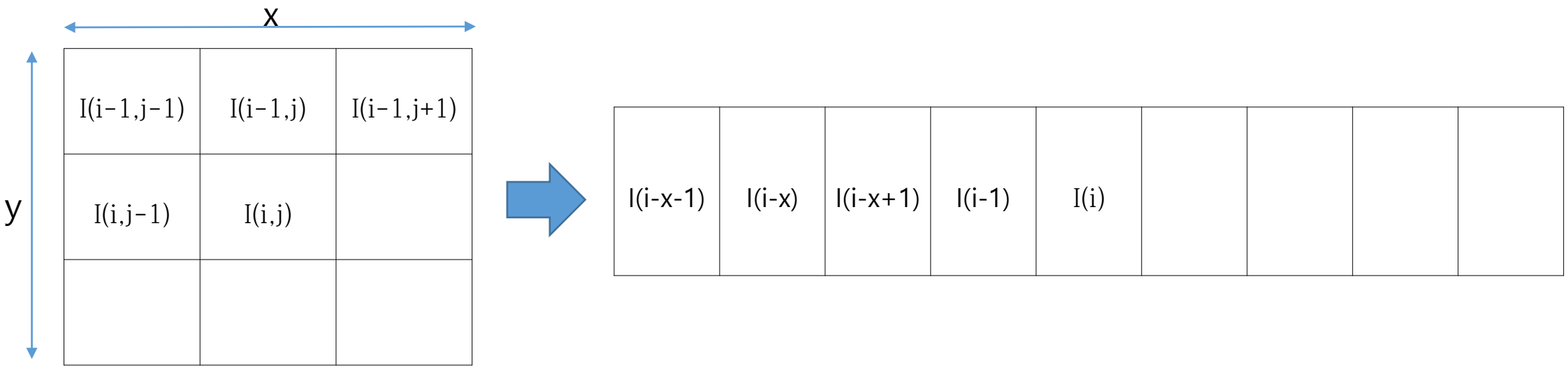
-After initializing, no access for zero pixel

```
img_true = np.where(img > 0)
img_true = np.asarray(img_true)
########################################
########################################
start = timeit.default_timer()
for i in range(img_true.shape[1]):
        y = img_true[0][i]
```



Reducing operation(2d array -> 1d array)

Nested for loop -> one for loop
2d array access -> 1d array access



flattening -> reshaping

: 2 times faster

Reducing operation(2d array -> 1d array)

2d array access -> 1d array access

Resolve pair

Pair_list
:[(3,4),(4,5),(5,6),....]

Resolve pair

Pair_list
:[3,4,4,5,5,6,....]

```
for p in range(len(pair_list)):
    disjoint.union(pair_list[p][0], pair_list[p][1])
```

```
for p in range(0,pair_list.shape[0],2):
    disjoint.union(pair_list[p], pair_list[p+1])
```

Reducing operation(Index calculation)

```
if Labeled_img[i-x] !=0 :
    Labeled_img[i] = Labeled_img[i-x]
elif Labeled_img[i-1] !=0 :
    Labeled_img[i] = Labeled_img[i-1]
    if Labeled_img[i-x+1] != 0 and Labeled_img[i-x+1] != Labeled_img[i-1]:
        pair_list = np.append(pair_list,[Labeled_img[i-x+1], Labeled_img[i-1]])
elif Labeled_img[i-x-1] != 0 :
    Labeled_img[i] = Labeled_img[i-x-1]
    if Labeled_img[i-x+1] != 0 and Labeled_img[i-x+1] != Labeled_img[i-x-1]:
```

```
v = img_true[0][i]
v1 = v-x
if Labeled_img[v1] != 0:
    Labeled_img[v] = Labeled_img[v1]
elif Labeled_img[v - 1] != 0:
    Labeled_img[v] = Labeled_img[v - 1]
    if Labeled_img[v1+1] != 0 and Labeled_img[v1+1] != Labeled_img[v - 1]:
        pair_list = np.append(pair_list, [Labeled_img[v1+1], Labeled_img[v - 1]])
```

Index pre-calculation

-> variable

1.More efficient memory access

2.Reduce summation operation

Reducing operation(Union find)

```python
class Disjointset_pc:

    def __init__(self,n):
        self.data = np.arange(n)
        self.size = n

    def upward(self,change_list,index):
        value = self.data[index]

        if value == index:
                return index

        change_list.append(index)
        return self.upward(change_list,value)

    def find(self,index):
        change_list =[]
        result = self.upward(change_list,index)

        for i in change_list:
            self.data[i] = result

        return result
```

1.Path compression

->Reduce iteration in finding root

2.Disjoint set size = Maximum label value +1

(for optimization)

```python
disjoint = Disjointset_pc(np.max(pair_list) + 1)
```

Pair_list
:[3,4,4,5,5,6,….] -> maximum value

Reducing operation(Second pass loop)

Need two kind of loop -> Make Index array and iterate only for unequal component

Need one loop

Disjoint data (After union find)

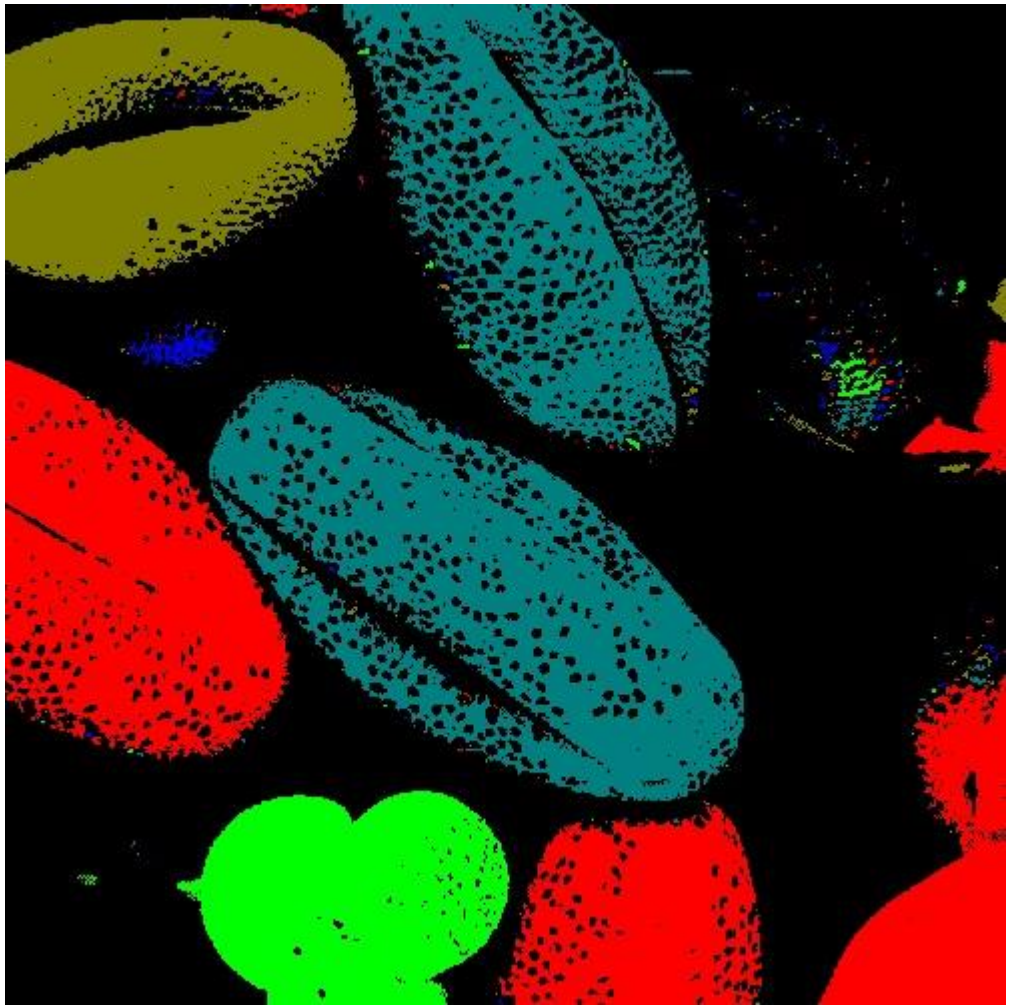
: [0 1 2 3 3 3 3 3 3 3 3 3 3 3 3 3 4 5 5 5 7 7]

Index array

: [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20]

```
disjoint_linear = np.arange(disjoint.data.shape[0])
a = np.where(disjoint.data != disjoint_linear)
```

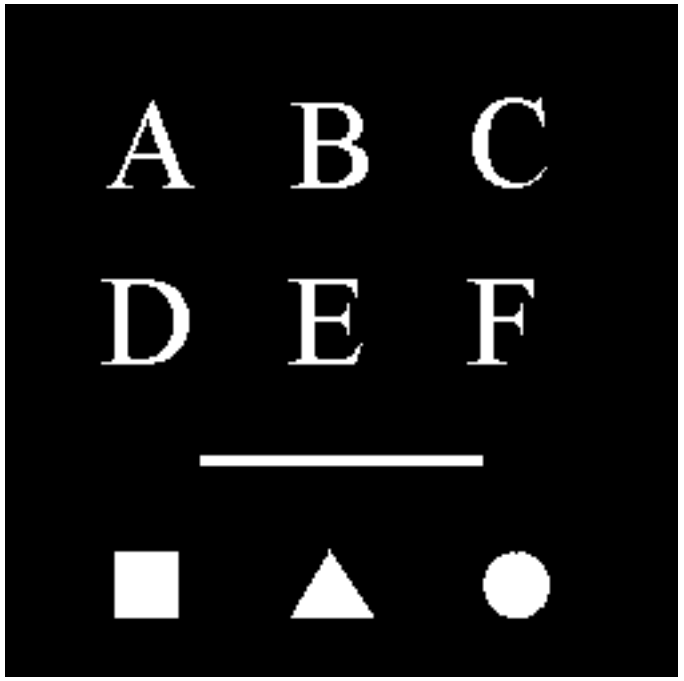
Conclusion(pollen.bmp)

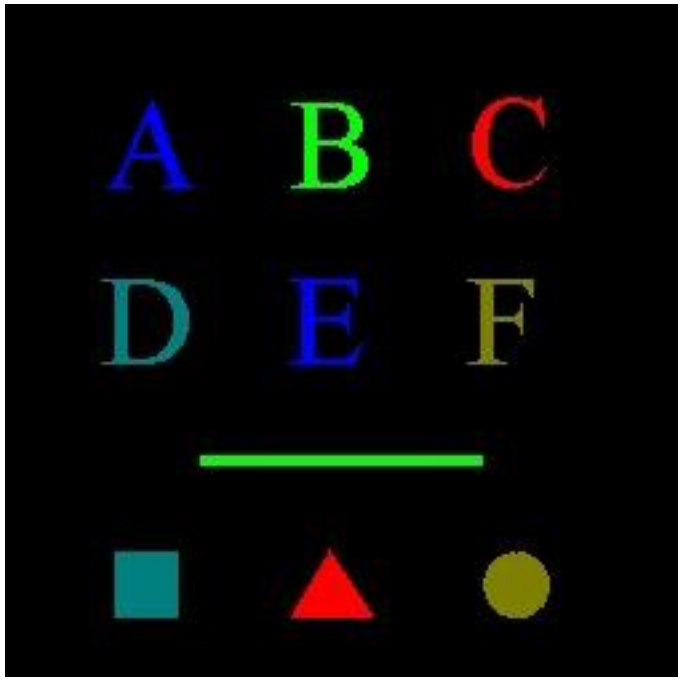


Conclusion(pollen.bmp)

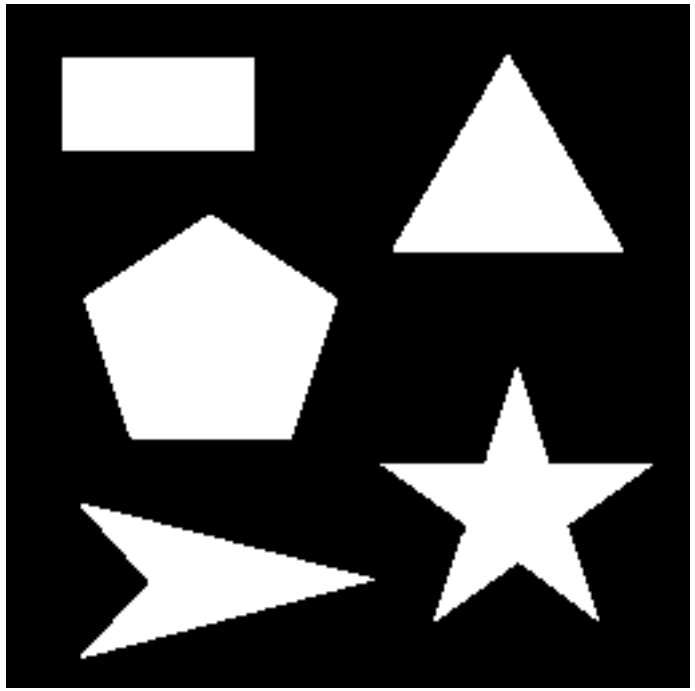
unit : s	No optimization	Resolve reduction	zero-pixel reduction	Array Dimension reduction	pass Compression (union find)	Second-pass reduction	Index pre-calculation
First - pass	35.4 →	3.25 →	1.63 →	0.80	0.80	0.80 →	0.65
Union-find	0.2 →	0.03	0.03	0.025	0.0126	0.0126	0.0126
Second-pass	1.2	1.2	1.2 →	0.54	0.54 →	0.28	0.28
Coloring	1.23	1.23 →	0.789 →	0.158	0.158	0.158	0.158
Total	38.03	5.71	3.649	1.523	1.5106	1.2506	1.1006
Total (except coloring)	36.8	4.48	2.86	1.365	1.3526	1.0926	0.9426

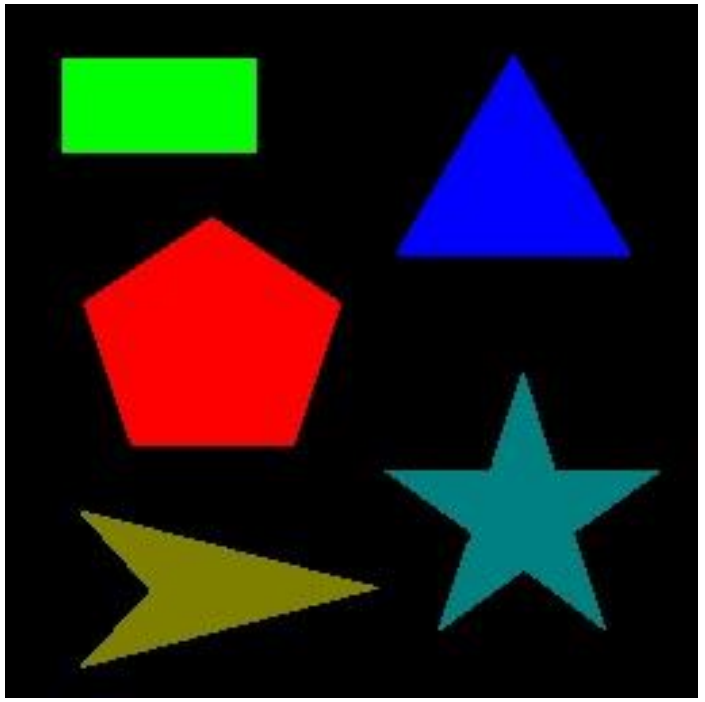
Other image (Symbol)



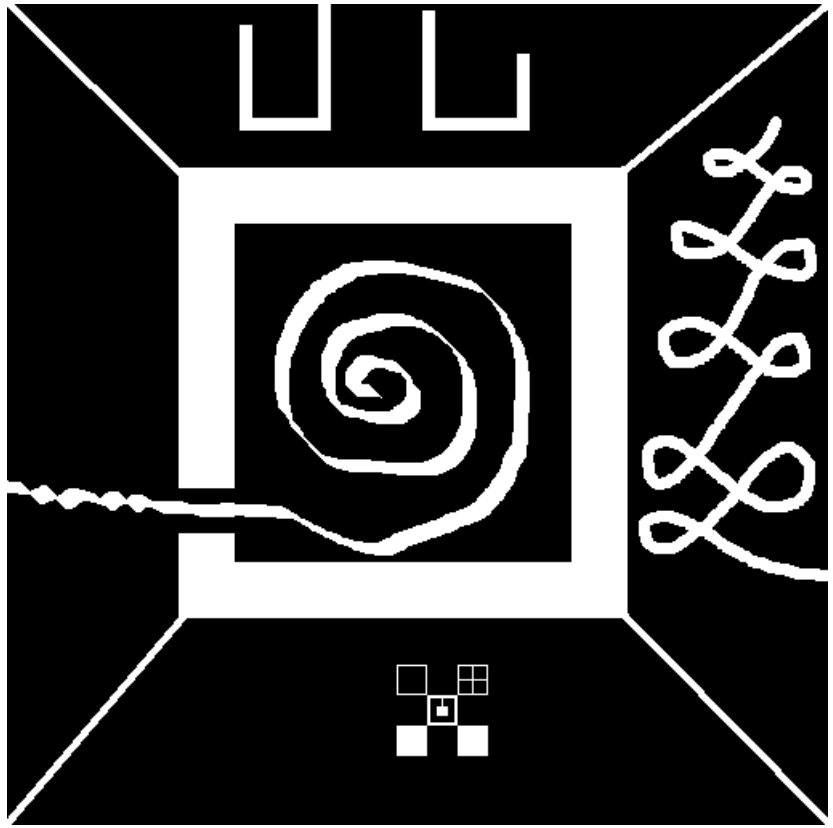


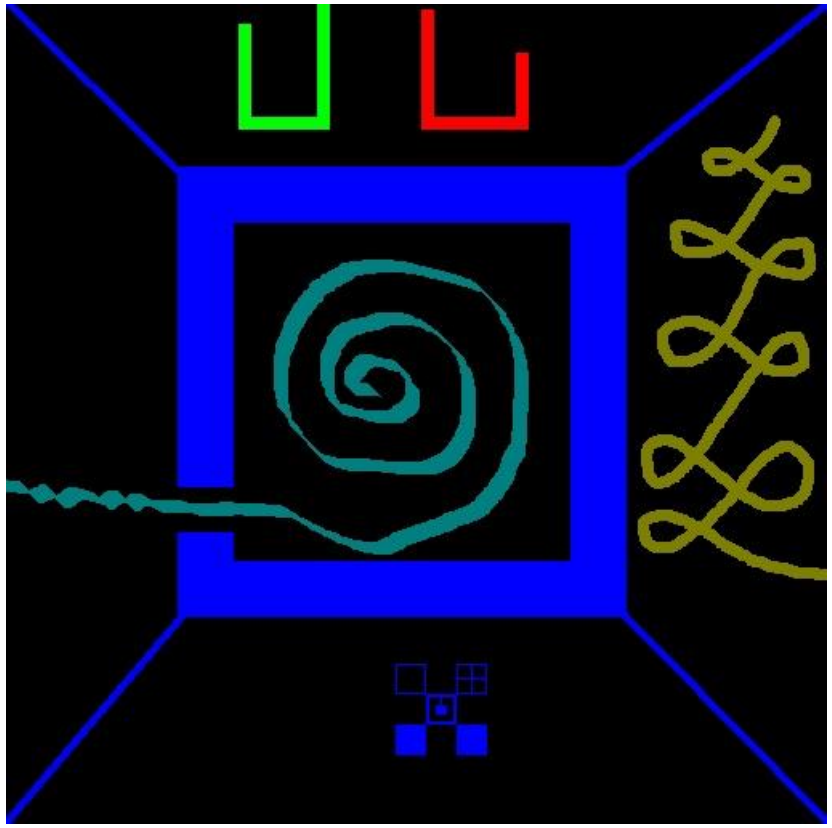
Other image (polygon)



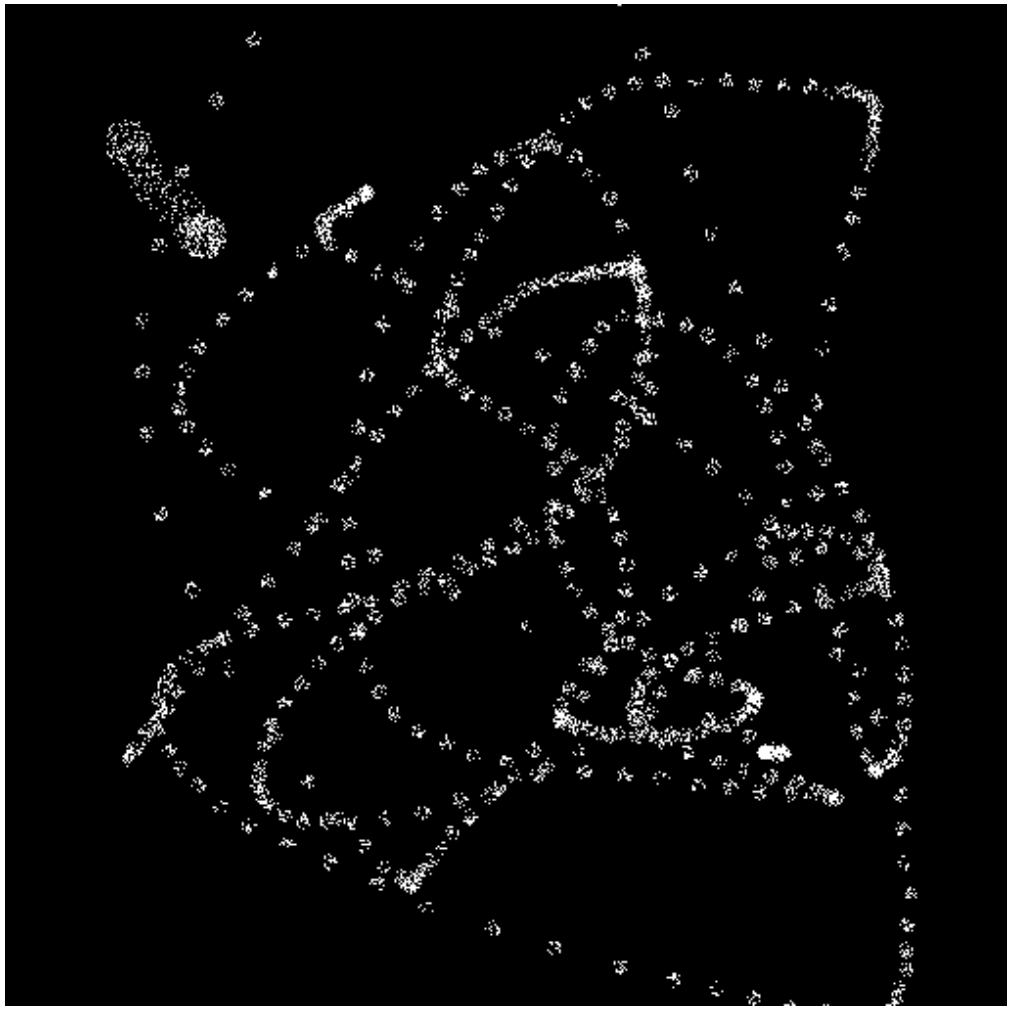


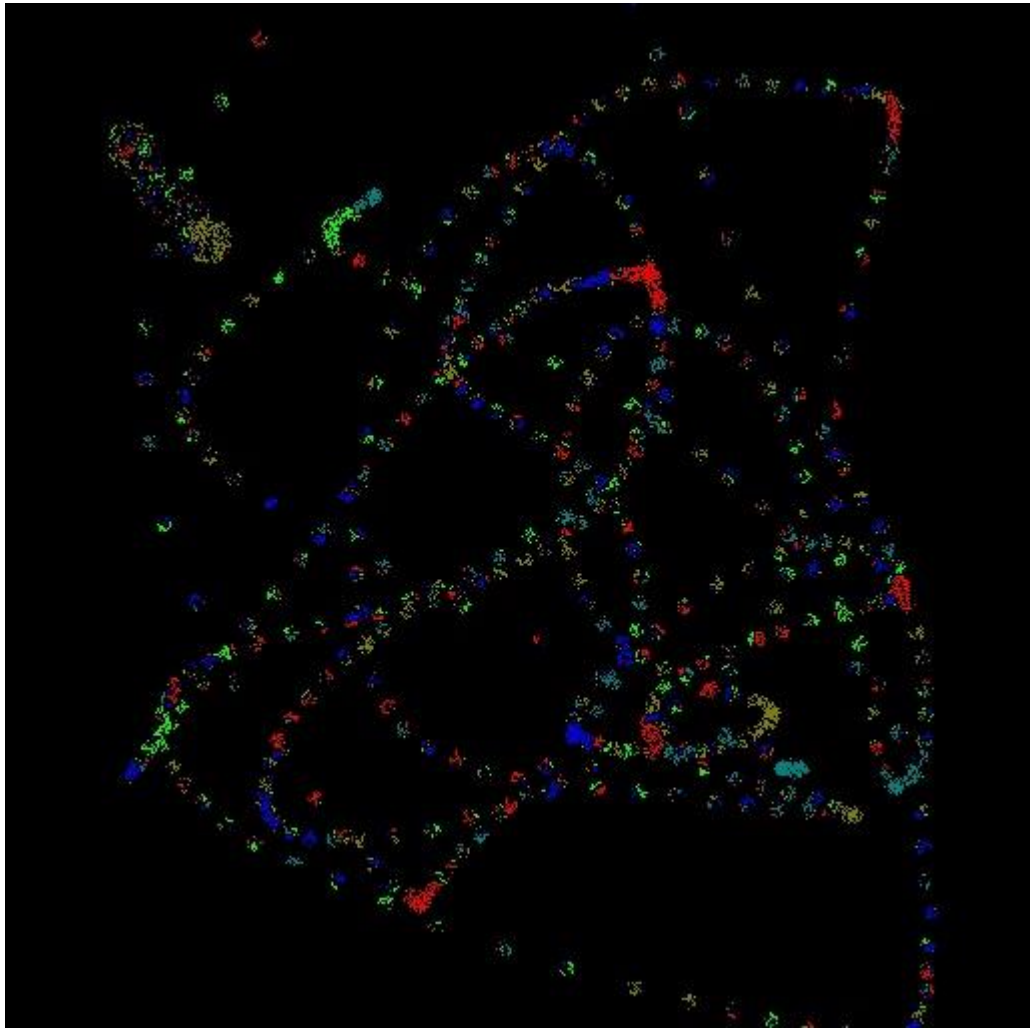
Other image



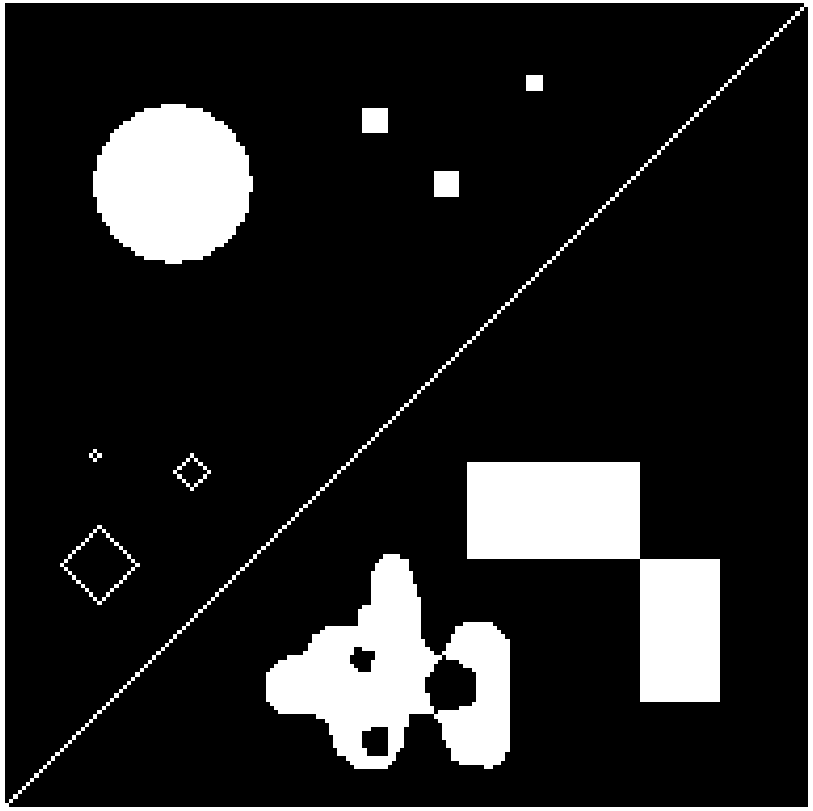


Other image(Foot print)

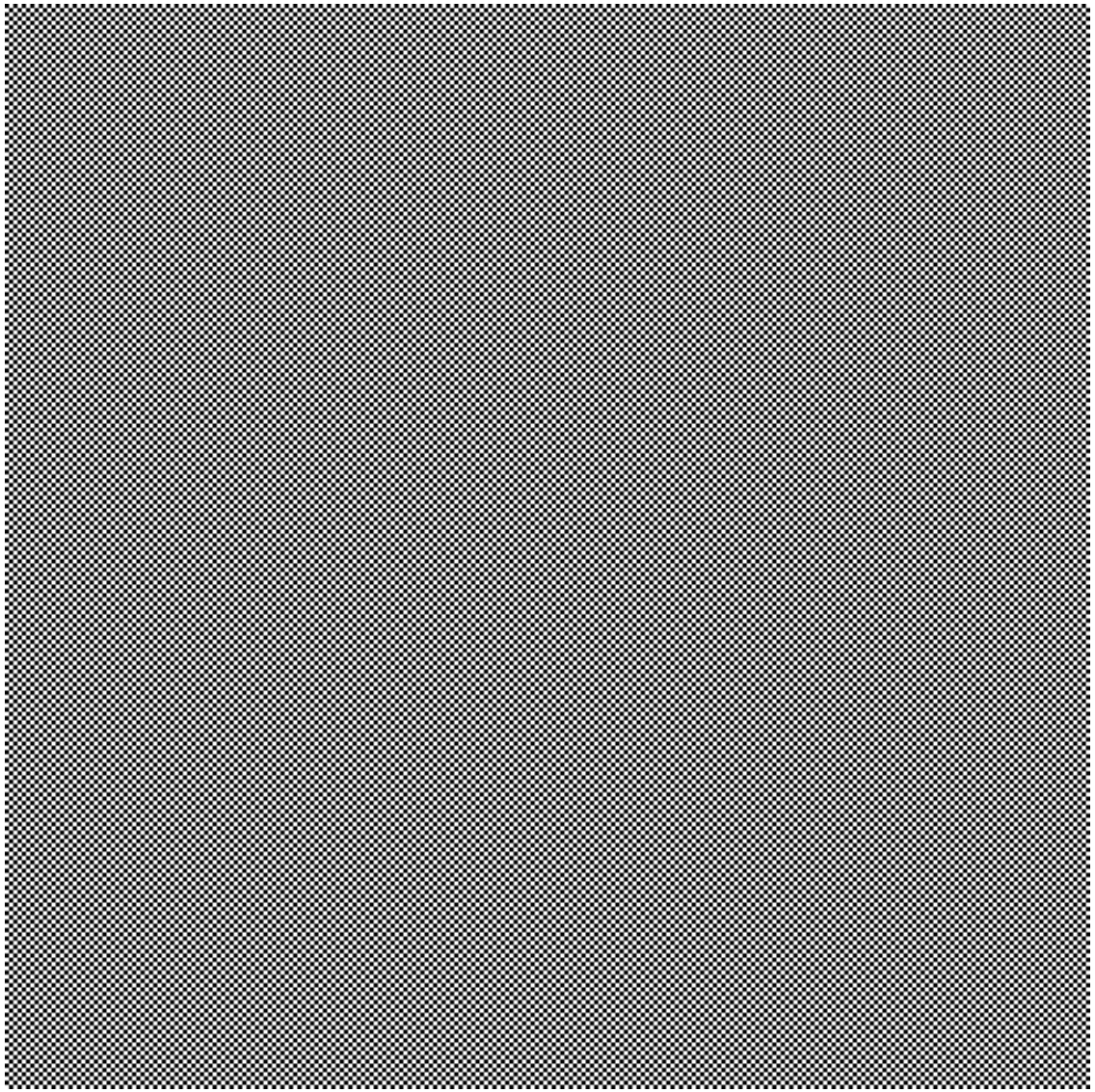


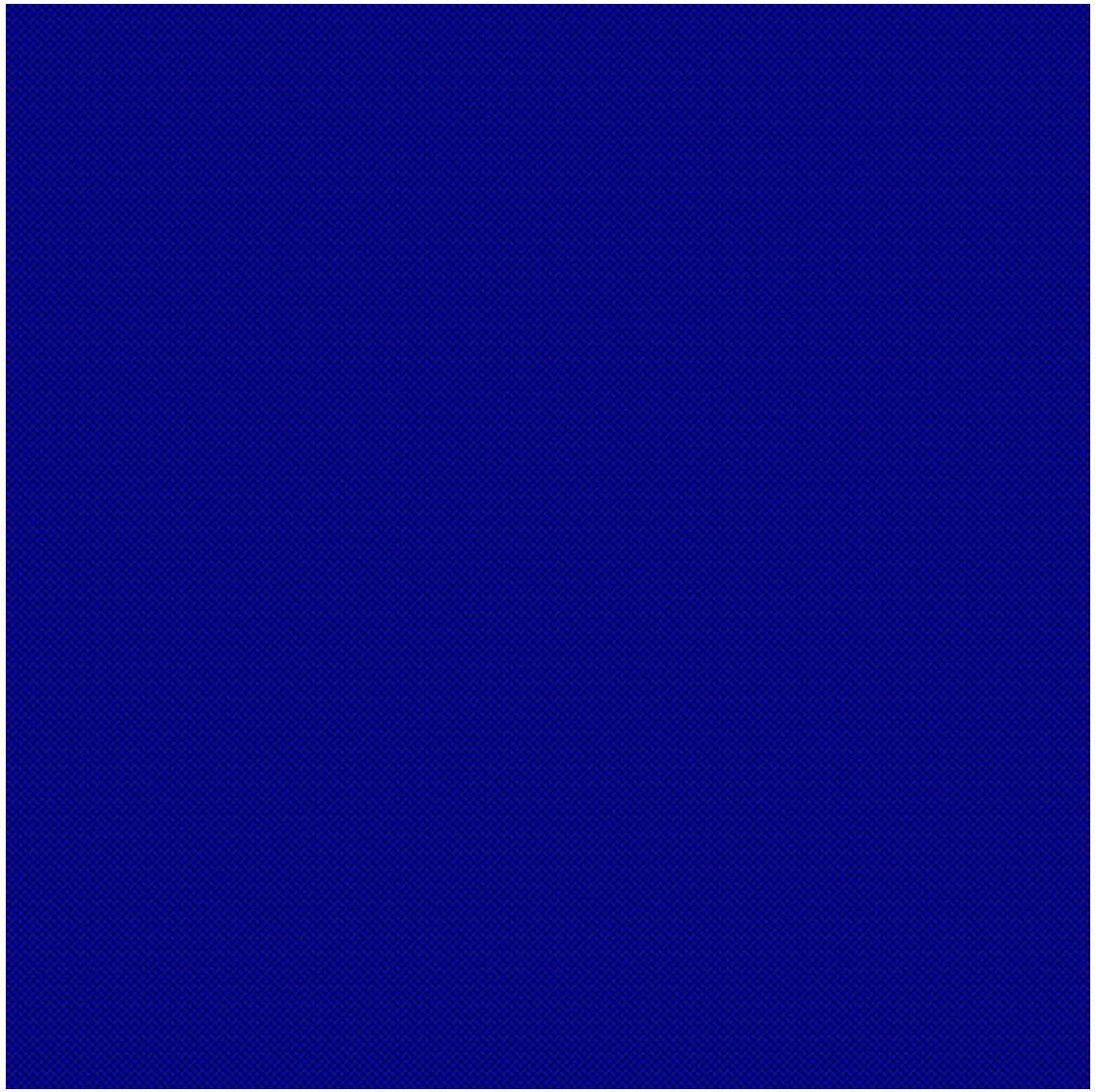


Other image(Moon light)

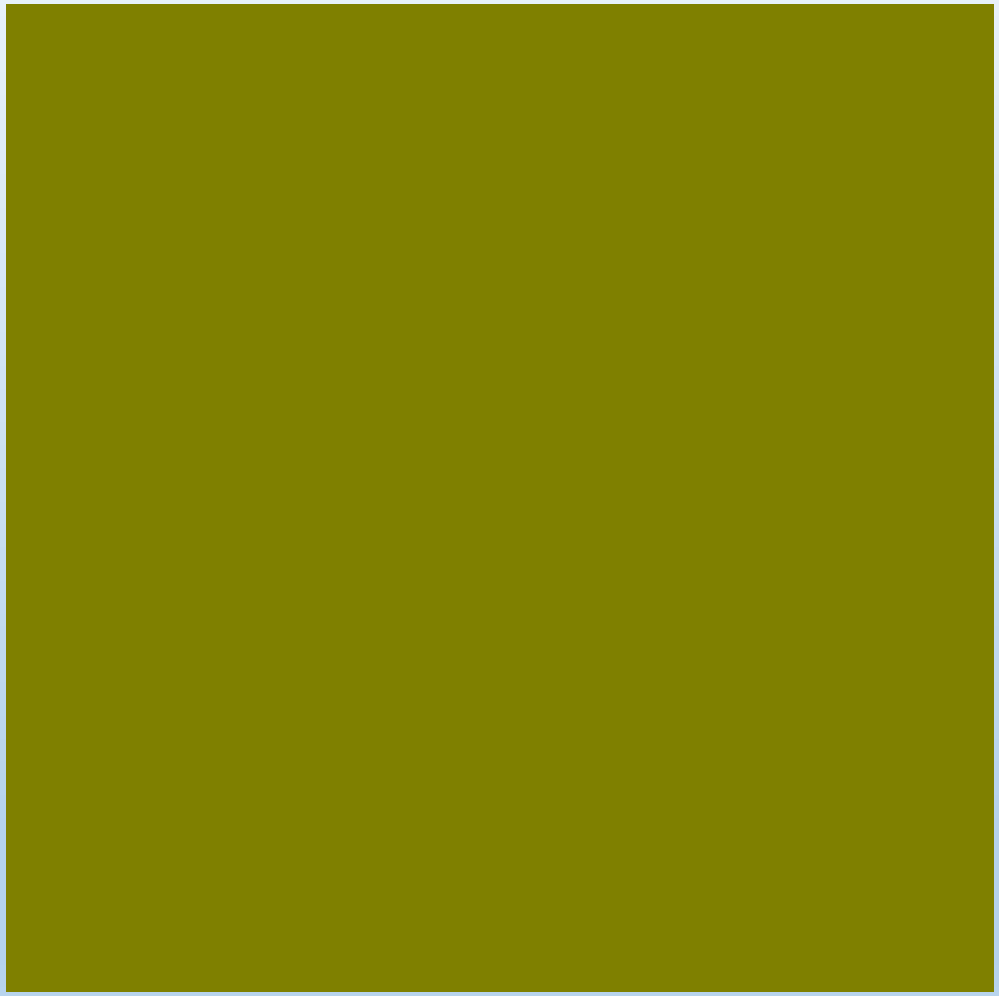


Other image(check board)





Other image (white image)



Other image

unit : s	Symbol	Polygon	Haze	Foot print	Moonlight	Check-board	White image
First - pass	0.0403	0.1208	0.3419	0.1209	0.0255	2.4197	1.6127
Union-find	0.0008	0.0006	0.0014	0.0081	0.0003	0.1816	0.0002
Second-pass	0.0041	0.0026	0.0555	0.1776	0.0008	0.0495	4.478 e-05
Coloring	0.0055	0.0071	0.0184	0.6893	0.0031	0.0175	0.020
Total	0.0507	0.1311	0.4172	0.9959	0.0297	2.6683	1.6329
Total (except coloring)	0.0452	0.124	0.3988	0.3066	0.0266	2.6508	1.6129

THANK YOU
for your attention