DIP Homework Assignment #3

資工三 b05902115 陳建丞

PROBLEM 1: MORPHOLOGICAL PROCESSING

(a) Perform boundary extraction

- **Motivation**

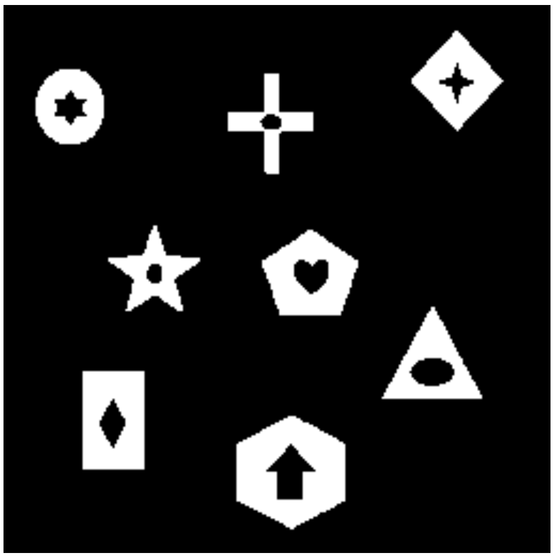
  As taught in the lecture, I first apply the erosion effect on the input image. And use the original image to subtract the erosion image. like the formula below.

$$\beta(F(j,k)) = F(j,k) - (F(j,k) \Theta H(j,k))$$

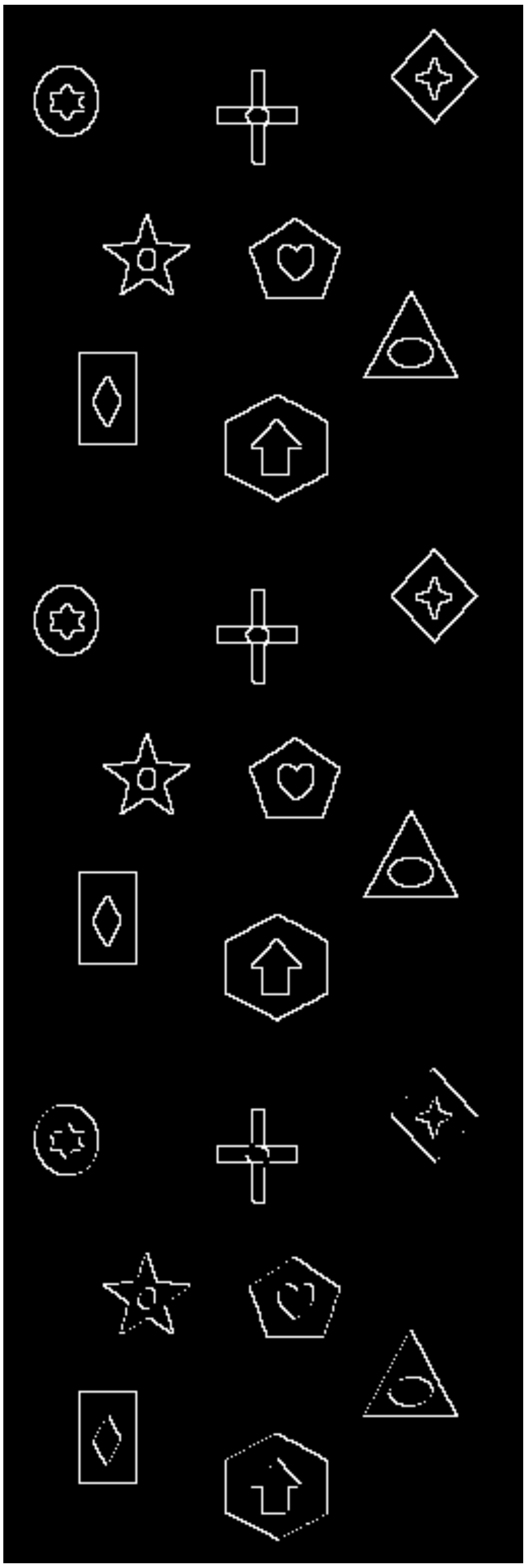
  And for the erosion process, I might try different filter mask to see if there's any difference in the output images.

- **Approach**
  1. Store the pixel values in a 2d array.
  2. Create another 2d array C with size 258 x 258, and with all the pixel values are 255.
  3. Set up a 3 x 3 filter mask H.
  4. Implement erosion process on the input image (Use **Sternberg definition**)
     - $G(j,k) = \cap \cap_{(r,c)\in H} T_{r,c}\{F(j,k)\}$
     - for any $H(r,c) = 1$, shift the original image r pixel distance downward and c pixel distance to the right. Do the **and operation** on the shifted image and the array C. Store the result in the array C.
     - After we process all the $H(r,c)$, the erosion is finished and the result is in C.
  5. The boundary is the original image subtracts C.

- **Original image (sample1.raw)**



- **Result**



  - $H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
  - $H = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
  - $H = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$

- **Discussion**

In fact, different filter masks do not lead to much difference in the results. Most of the results have clear boundary, unless we use some extreme filter like $H = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ or $H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ If we use these extreme filters with very little 1s in it, the output boundary will become a little unclear. That said, we still can see the patterns of each components.

(b) Perform connected component labeling

- **Motivation**

  We can use dilation to compare with the original image to see if the connection is in the original image. As the formula mentioned in the lecture. We can keep adding connecting components until there is no new componets.

$$G_i(j,k) = (G_{i-1}(j,k) \oplus H(j,k)) \cap F(j,k) \quad \text{i} = 1,2,3,\ldots$$

  Also, it's funny to adjust the RGB values of the images to generate different output colors.

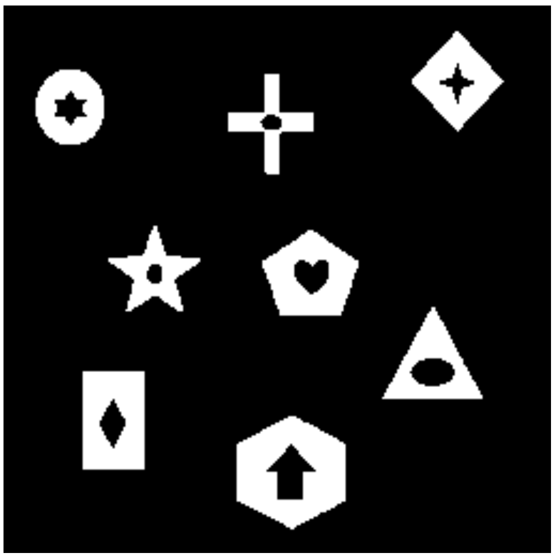
- **Approach**

  1. Store the pixel values in a 2d array.
  2. Select a pixel with value 255 (white pixel).
  3. Use the eight connectivity filter to implement **dilation** with the original image.

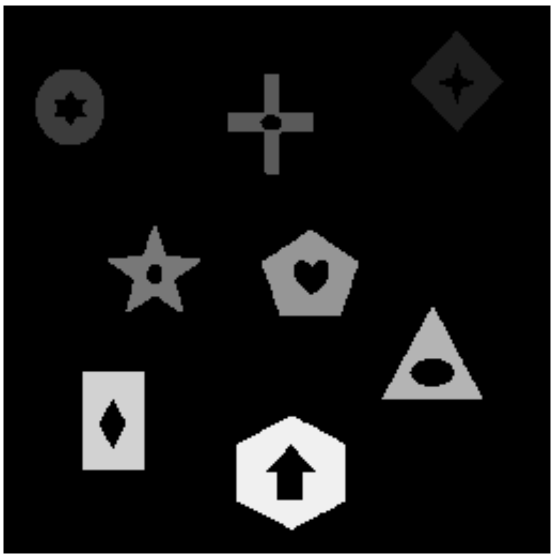
$$G(j,k) = \bigcup_{(r,c)\in H}\bigcup T_{r,c}\{F(j,k)\}$$

  4. Keep doing the iteration until there is no new connected points.
  5. After finish a component, add label to the pixels in that component.
  6. Keep finding new component until all the pixels with value 255 have been labeled.
  7. Add different colors to the labeling pixels.
  8. The rgb images should be stored in a 3d array with size 3 x 255 x 255

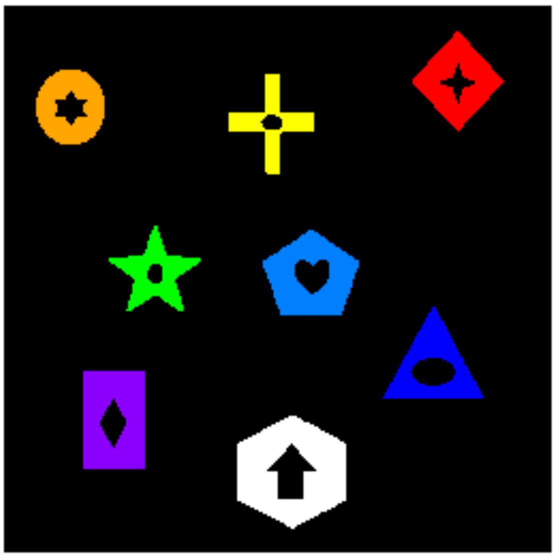
- **Original image (sample1.raw)**



- **Result**
    - Different gray scale



    - Different RGB color



- **Discussion**

  For this sample image, all the patterns are strongly separated, it's not hard to implement the connected component labeling. The result will be very good. However, for some images that has patterns close to one another, i guess that the result might have some error in the boundary area.

(c) Perform thinning and skeletoning

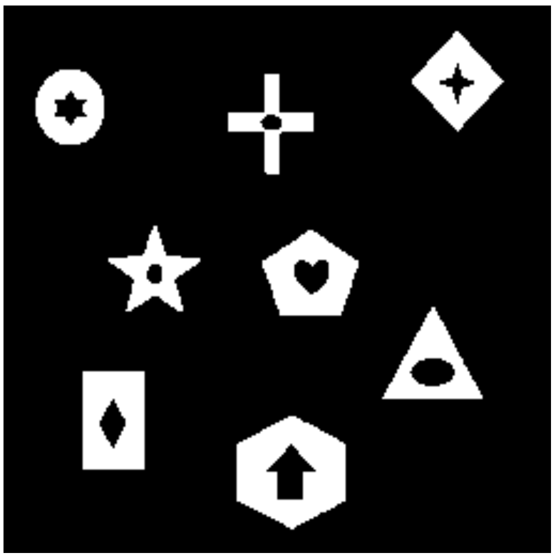
- **Motivation**

  Use lots of filters with different patterns to examine if a white pixel is connected to or covered with other white pixel. There are well-defined filters with different to make sure that we can clearly detect the relation with white pixels. Also, there is a second stage check to make sure that we won't eliminate too much pixels.

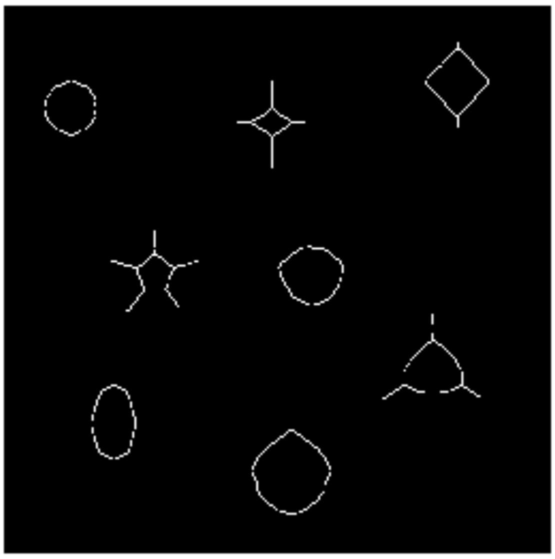
- **Approach**

  1. Store the pixel values in a 2d array.
  2. Create a binary 2d array, assigned value 1 if the original image is a white pixel.
  3. Set up all the conditional mark patterns and unconditional mark patterns (thinning and skeletion effect have different patterns)
  4. For stage 1, for any conditional mask $M(j,k)$
     - if $M(j,k)$ hits, then set (j,k) as a candidate.
  5. For stage 2, for any unconditional mask $M(j,k)$
     - if (j,k) is a candidate and $M(j,k)$ misses $\Rightarrow$ erase the center pixel.
  6. Keep iterating stage1 and stage2 until there's no more points to erase.
  7. Both thinning and skeletoning can be done with the same process, but with different patterns.

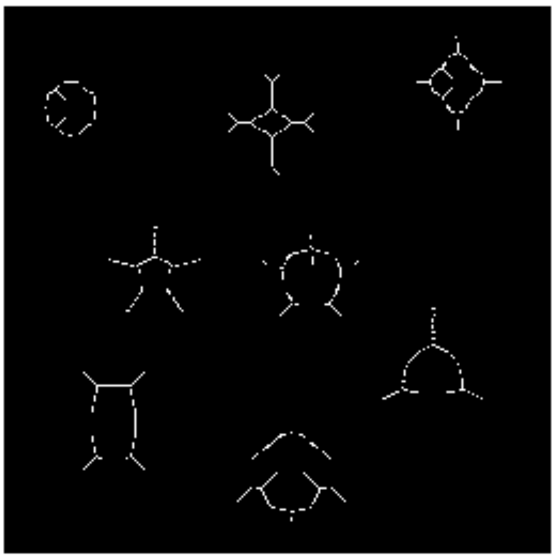
- **Original image (sample1.raw)**



- **Result**
  - thinning



- skeletoning



- **Discussion**

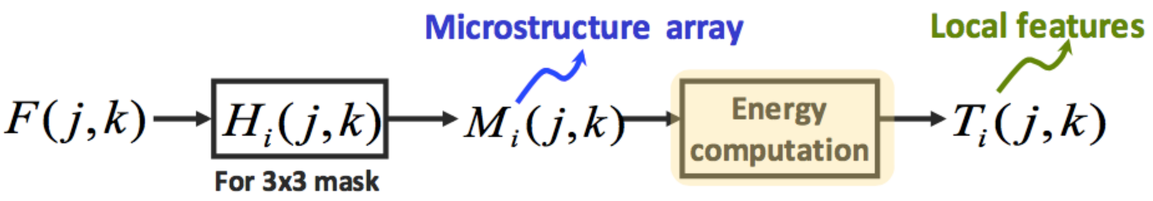
  The output images are not really clear. We can see the outline of the pattern. For the patterns that are a little more complex like the arrow pattern in the sample image, the result is bad, both thinning and skeletoning. I think that I might have erase too much points. This problem might be resolved if I use less conditional mark and decrease the number of candidates. Or I can just set up an iteration number for stage1 and stage2, which can cut down the number of erase operation.

PROBLEM 2: TEXTURE ANALYSIS

(a) Perform Law's method to obtain the feature vector of each pixel

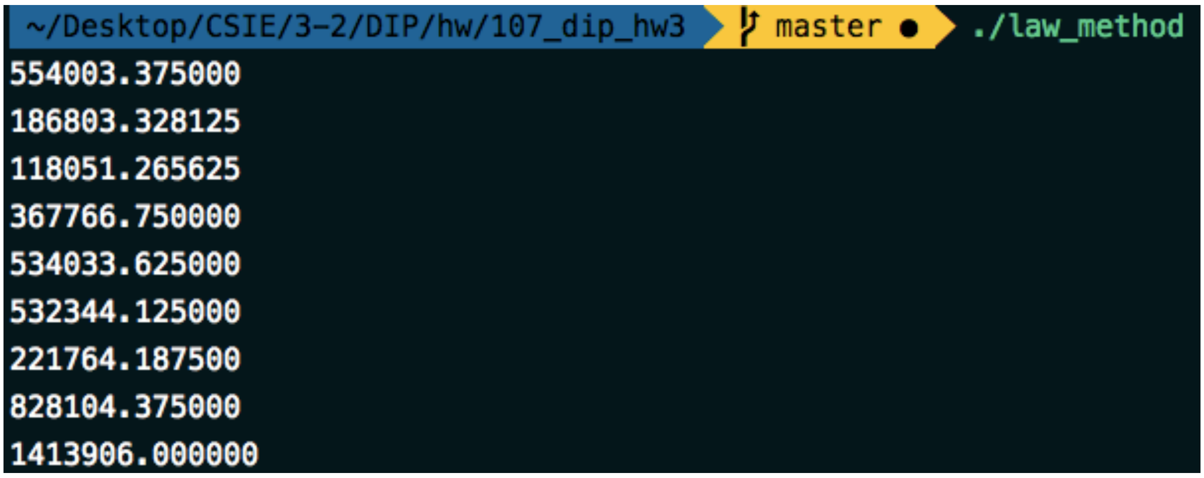
- **Motivation**

  Follow the steps of Law's method and avoid the problems in the boundary.



- **Approach**
  1. Store the pixel values in a 2d array F.
  2. Set up all the 9 micro-structure impulse response arrays.
     - Ex : Laws 1 $\Rightarrow \frac{1}{36}\begin{bmatrix}1 & 2 & 1\\ 2 & 4 & 2\\ 1 & 2 & 1\end{bmatrix}$
  3. Convolutionally apply tensor product on F and the 9 filters H_i. For each pixel, there will be 9 features, and I store the product of feature i in the array M_i
  4. Set up window with size 13 x 13.
  5. Since the window is big, we have to expand the M_i for next process.
     - Copy the leftmost 6 columns and the rightmost 6 columns of M_i and expand horizontally with them.
     - After expanding the horizontal direction, copy the topmost 6 rows and the downmost 6 rows and expand vertically with them.
     - We get a new M_i with size 524 x 524
  6. Applying energy computation and store in the array T
     - $T_i(j,k) = \Sigma\Sigma|M_i(j+m,k+n)|^2$
  7. $T_i(j,k)$ is the feature vector of pixel (j,k) in of feature i.

- **Result**
  - Ex : 9 features energy of pixel $(0,0)$



```
~/Desktop/CSIE/3-2/DIP/hw/107_dip_hw3  master ●  ./law_method
554003.375000
186803.328125
118051.265625
367766.750000
534033.625000
532344.125000
221764.187500
828104.375000
1413906.000000
```

- **Discussion**

  I encountered many problems in this question. First, there are many steps in the Laws' method. It's easy to make mistake in the calculation process. Next, Since the window size is big, it's complicated to expand the boundary of the arrays. And last, the output is lots of big nubmers, it's pretty hard to check if there is any mistake.

(b) K-means classification

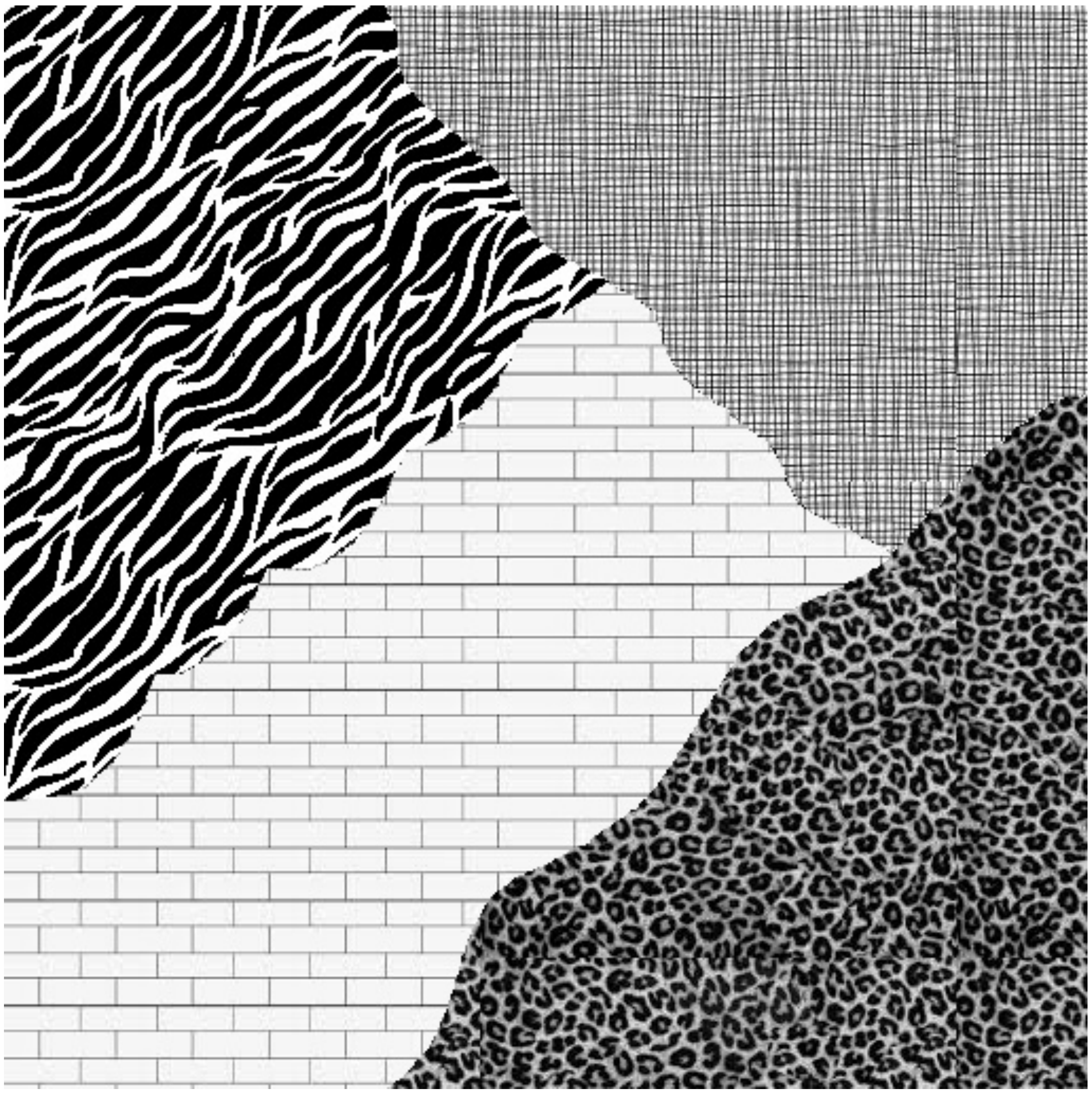
- **Motivation**

  From the sample image, we can clearly observe that there are four different textures. Therefore, we can just set up **K = 4** to classify the pixels into clusters. And since the textures coverage is big, it's easy to pick the 4 initial pixels to start the K-means classification.

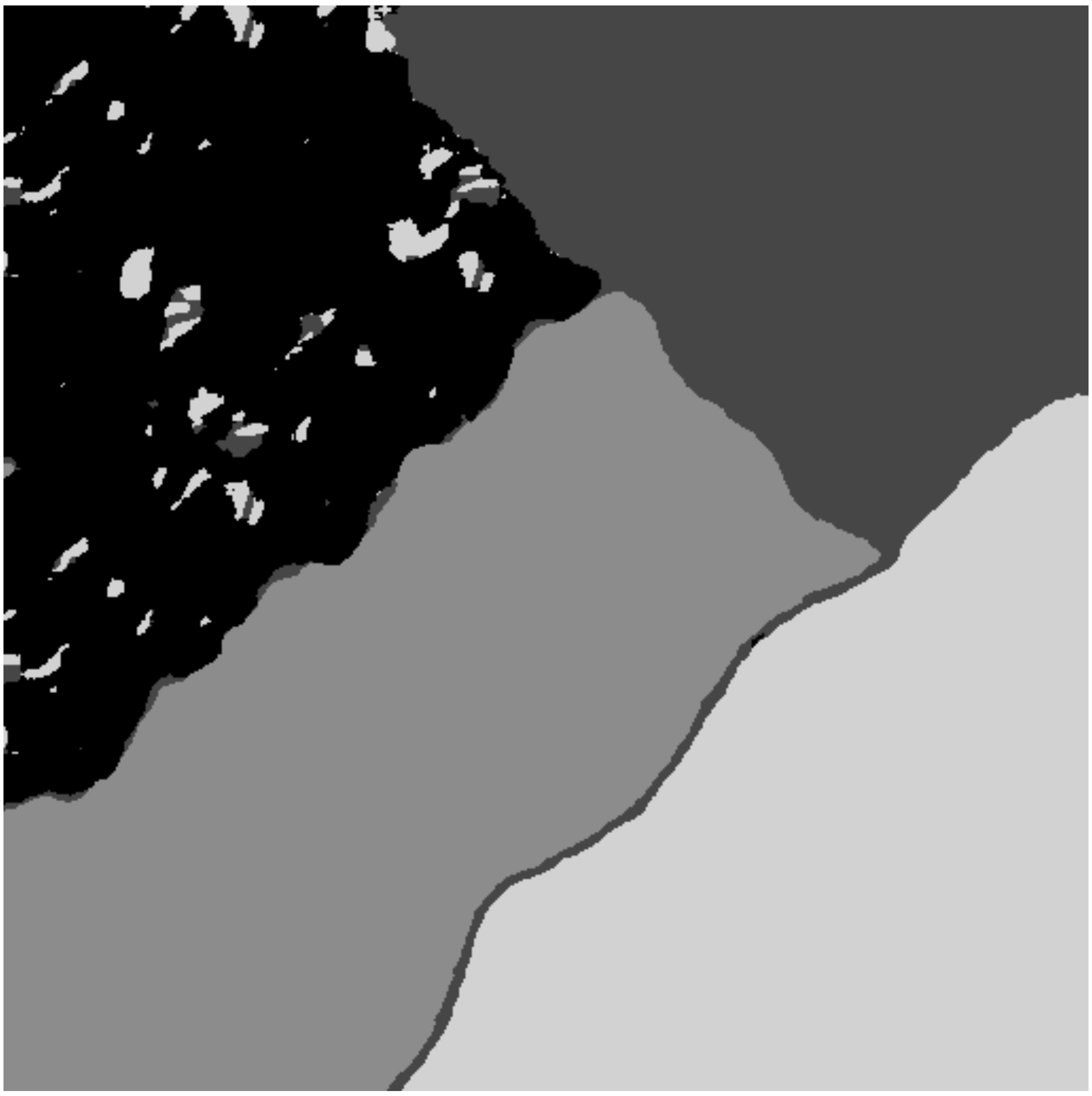
- **Approach**

  1. Get the feature vector from question (a).
  2. Observe the image, and pick 4 initial point from 4 different texture.
  3. Use the points as the center, and calculate the average feature energy of the window of size 50 x 50.
  4. For each pixel (j, k), calculate the sum of the 9 feature distances, classify pixels (j, k) to the cluster with the mininum distance sum.
  5. After all the pixels are classified into 4 different clusters, assign different gray scale values to each cluster.

- **Original image**



- **Result**



- **Discussion**

  We can clearly observe that the output image has four block. Most of the texture detection is successful. However, there is some misclassification in the left-top area. From the original input image, the left-top texture and the right-bottom texture has a little in common if we see partially. Therefore, it's reasonable that the K-means classification will have some error. To resolve the problem, maybe we can adjust the process in Laws' method to let different features have different weight. Or we can set up different size of window in the initial selection steps.

(c) Exchanging texture

- **Motivation**

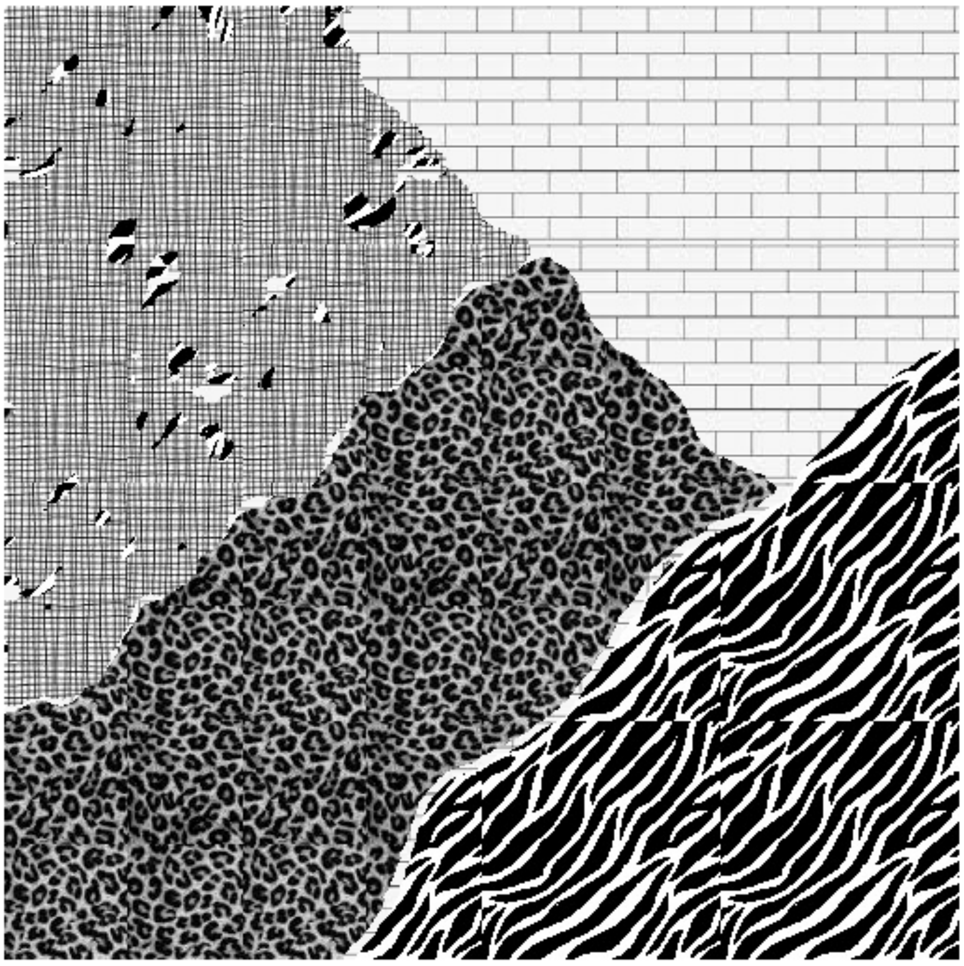
  Since we have classified each pixels to different clusters. We can easily apply textures to pixels according to their cluster. To make sure the texture's continuity, we can copy a block from the texture and assign the texture block-wise.

- **Approach**

  1. Copy 4 textures with block size 128 x 128. We can just take the four corners of the original images.
  2. Start pasting the copied block from the 1st texture to the 4th texture.
  3. Use the clusters information of each pixel from problem (b) to assign different texture

to each pixel.

- **Result**



- **Discussion**

  Since we use block to paste the texture, the output image has great continuity. We can still see that the image has lots of block outline inside. To eliminte the outline of each blocks, we can set up a larger block size to increase continuity. However, since the original image is not that regular, it's hard to define a perfect block size.

Electronic version README

- **The input image should be in the directory raw**
- call **make –f README**
- The programs will be compiled and executed automatically after the command.
- Output images will be stored in the folder **output**.