DIP Homework Assignment #2

姓名：鍾毓安

學號：B01902040

系級：資訊四

Email: [iamyuanchung@gmail.com](mailto:iamyuanchung@gmail.com)

繳交日期：2016.4.12

**Source Code**

The following script and functions were implemented for this assignment. More details of implementations were commented in each corresponding file.

1. README.m: This is the main script that acts like the main function. All the required tasks with best-tuned parameters will be completed one by one when README.m is executed.
2. sobelEdgeDetection.m: Given an image and threshold , this function computes the 1st order gradients by convoluting with two Sobel masks and takes those pixels whose gradients are greater or equal to as edges. This function returns the row gradients map , column gradients map , and the final edge map .
3. laplacianOfGaussian.m: Given an image and filter size , this function performs the Laplacian of Gaussian (LoG) technique for detecting edges. Hinted by its name, LoG uses a Gaussian smoothing filter for reducing the noise in before Laplacian is applied. However, this results in two rounds of convolution, including the Gaussian smoothing process and the Laplacian operation, which causes heavy computation. Since the convolution operations are associative, one can convolve the Gaussian smoothing filter with the Laplacian filter first, and then convolve this “hybrid filter” with to achieve the required result. We will refer to this hybrid filter as LoG filter. By using the LoG filter, only one round of convolution operation with the image is needed. Argument is used to determine the size of the LoG filter. This function returns the resulting edge map.
4. cannyEdgeDetection.m: Canny algorithm contains five steps and are too long to describe here, please see the function file for details of implementations.
5. stitchFourFigures.m and warpToGourdShape.m: These two functions are hard-coded and only suitable for taking sample4 ~ sample7 as inputs. The details of stitching and the design of warping functions are described in Problem 2.

**Problem 1: Edge Detection**

Three images are given in this problem. For each given image, you are required to generate several edge maps using the following methods. [Please mark the edge points with intensity value 1 and background points with intensity value 0.]

1. Sobel edge detection
2. 2nd order edge detection
3. Canny edge detection

For each method, please apply different parameters and provide some discussions on how they would affect the resultant edge maps. From the observations of your results, list pros and cons of each method, respectively.

**Source code**

1. sobelEdgeDetection.m
2. laplacianOfGaussian.m
3. cannyEdgeDetection.m

**Solution**

**Problem 2: Geometrical Modification**

The goal of this problem is to register the given four images and perform proper geometrical modification on the overlapped square image to obtain a desired shape.

1. Please stich these four images into one complete image and paint the residual regions in black. Denote the result as R.
2. Crop the largest square image of image R and denote it as S. (Hint: the size of S is 512 x 512.)
3. Segment the image S into three parts with predefined size and design three warping functions to convert the image to a gourd-shaped image. Output the result as G.

**Source code**

1. stitchFourFigures.m
2. warpToGourdShape.m

**Solution**

1. stitchFourFigures.m takes sample4.raw ~ sample7.raw, denoted as as inputs and outputs the registered image and the cropped 512 x 512 image . Before coming up a way for stitching these four figures, we should guess what will the resultant image look like to give us a better starting point. From our guess as displayed in Figure ?, we get the information of the relative positions of : , , , and are placed clock-wisely and the relationships of overlapping are also obtained. For each pair , used two for loops that slid through to find the maximal number of pixels matching for locating the overlapping region between and . The resultant image, denoted as , is shown in Figure ?.



Figure ?: my guess of what will look like



Figure ?: the resultant image after registration

1. stichFourFigures.m also returns the cropped image , and is shown in Figure ?. Same as the hint, is a image matrix.



Figure ?: the cropped image

1. warpToGourdShape.m was implemented to perform this task. This task can be divided into three parts: warping to a small rectangle with height equals to 64 and a specified width, to a circle with radius equals to 112, and to an oval with major axis equals to 512 and minor axis equals to 224. The small rectangle was generated by simply sliding a window through and averaging the columns vectors within the window together. To warp B to a circle, the following steps were implemented (please refer to Figure ?):

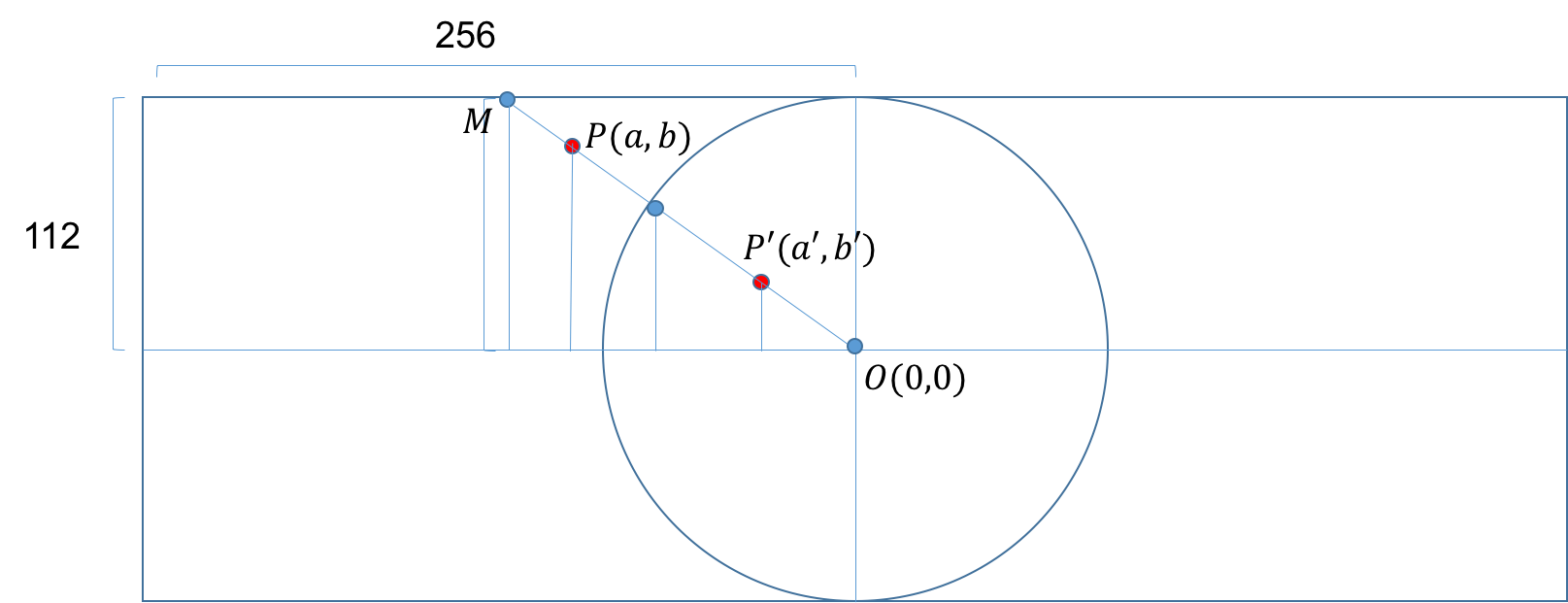


Figure ?: the design of the warping function for

* + - 1. For each pixel in , we can connect it with and form a line .
      2. Find the crossing point of line and the boundary (four sides of ).
      3. Compute the distance of and and scale to with the same proportion.

The warping function for is the same as , and the resultant gourd-shaped image is shown in Figure ?.



Figure ?: the resultant gourd-shaped image

**Problem 3: Texture Analysis**

The attached figure demonstrates a gray-level image which is compose of several animals with different texture patterns.

1. Perform Law’s method on the given image to obtain the feature vector of each pixel.
2. Use k-means to classify each pixel and label same kind of texture with same intensity. Please specify the intensity levels you adopt and output the result as L.
3. Based on image L, try to attach the correct texture to each animal as best as you can. Output the result as C.

Please provide the details of each step and discussions for each part in the report.

**Source Code**

1. lawsFeatureExtraction.m
2. classifyPixels.m
3. kMeansCluster.m
4. attachTexture.m

**Solution**

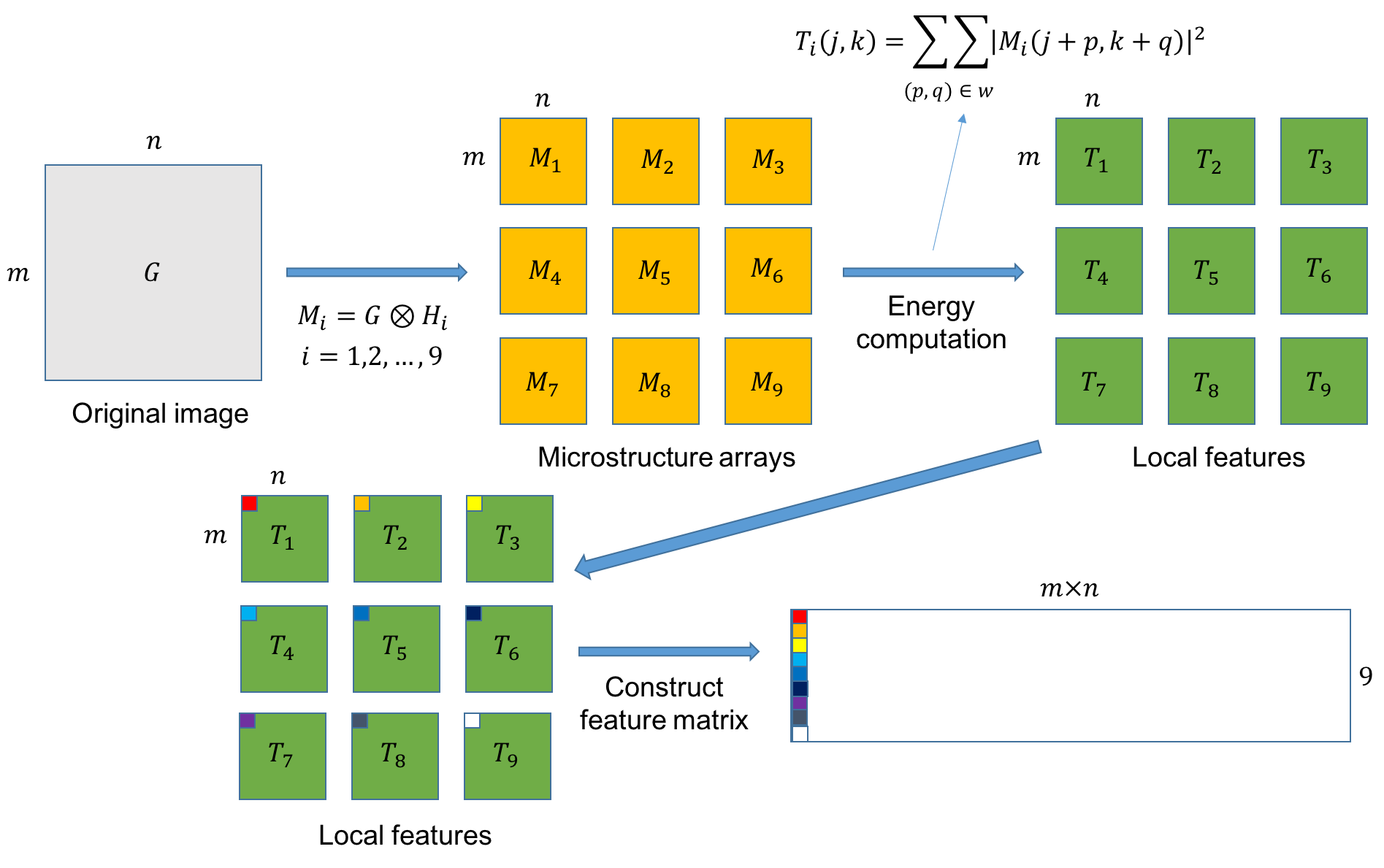
1. lawsFeatureExtraction.m implements Law’s method by taking an image and a specified window size as inputs and outputting a by feature matrix , where is the size of and each column is the feature vector of the -th pixel, indexing from top to bottom and from left to right of the original image . Basically, my implementation follows the steps stated in the course slides.

Figure 5: flow of Law’s method

Law’s method introduces 9 masks of size 3 by 3. Convolving with each of these 9 masks emphasizes the microstructures of the texture in and yields 9 microstructures arrays , where is the result of convolving with and has the the same size with . Next, Law’s method extracts the local texture features by computing the region energy of , yielding 9 feature sets , respectively, and also has the same size as . The input argument determines the window size for computing the region energy and should contains a few cycles of the repetitive texture, I chose in my implementation. The formula for computing energy is the square sum of intensity values within the window size. From the 9 feature sets , we can construct a feature matrix where each column is the feature vector of the pixel whose position is in . For each pixel whose position is in , its corresponding feature vector can be constructed by taking , , …, and and concatenating them to form a vector of length 9. Figure 5 describes the flow of Law’s method.

1. classifyPixels.m takes the feature matrix obtained from lawsFeatureExtraction.m as input, calls kMeansCluster.m to perform k-means clustering for labeling each pixel to one of the four categories of textures, including zebra, leopard, giraffe, and the background. Finally, based on the label obtained from kMeansCluster.m, classifyPixels.m generates the required image by assigning intensity value 0 to those pixels that belong to category 1, 80 to those that belong to category 2, 160 to those that belong to category 3, and 240 to those that belong to category 4. My implementation of k-means clustering algorithm is stated as follows:
   * + 1. Takes feature matrix as input where each column is the feature vector of the -th pixel.
       2. Randomly set 4 data points in as initial centers.
       3. While the assignments of all data points remain unchanged:

For each data point, compute the Euclidean distances to the 4 centers and assign the data point to the category whose center is closest to the data point.

Update the centers by averaging the data points under the same category.

* + - 1. Return a vector of length where records the category the -th pixel belongs to.

Note that due to the random initialization of centers in step 2, the labels may not always refer to as the same categories! That is, if we execute kMeansCluster.m twice, label 1 may indicate the texture of zebra for the first time, but indicate the texture of giraffe for the second time. Figure 6 displays the resultant image after classifyPixels.m. We can observe that the shapes of the leopard and the giraffe are nicely depicted, while the shape of the zebra looks just passable: the main structure of the zebra is captured but the contour is somewhat uneven and unsmooth. I think the reason is that the area of the dark gray structure of the texture within the zebra is too large, which makes it hard to distinguish it apart from the background during the feature extraction of the microstructure, causing some of those dark gray parts are mislabeled as the background.

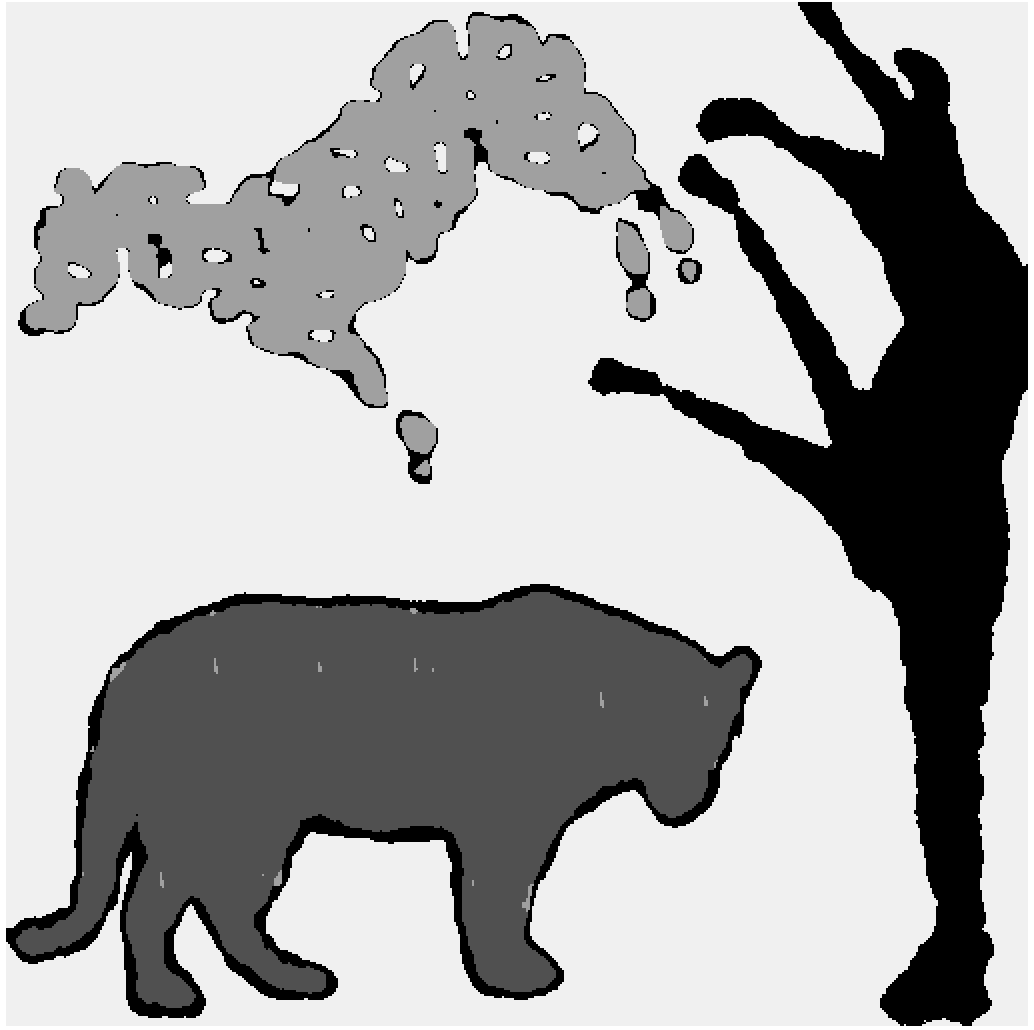


Figure 6: the resultant after classifyPixels.m

1. attachTexture.m was implemented to perform this task. In sample8.raw, the zebra has the texture of the giraffe, the leopard has the texture of the zebra, and the giraffe has the texture of the leopard. attachTexture.m first tries to grab the structure of texture from the three animals from sample8.raw and attach them to the correct animals in by repeating the corresponding structures. Figure 7 display the structures of texture grabbed from sample8.raw where the grabbed areas are bounded by squares, and Figure 8 is the resultant after attaching the correct textures to the corresponding animals. From Figure 8, we can observe that the boundary of the leopard is filled with the texture of the giraffe, such phenomenon can be explained when we look back on the boundary of the leopard in Figure 7: the leopard’s boundary was labeled as the same category as the giraffe. Fortunately, the texture on the body of the leopard and the giraffe look nice and comfortable.

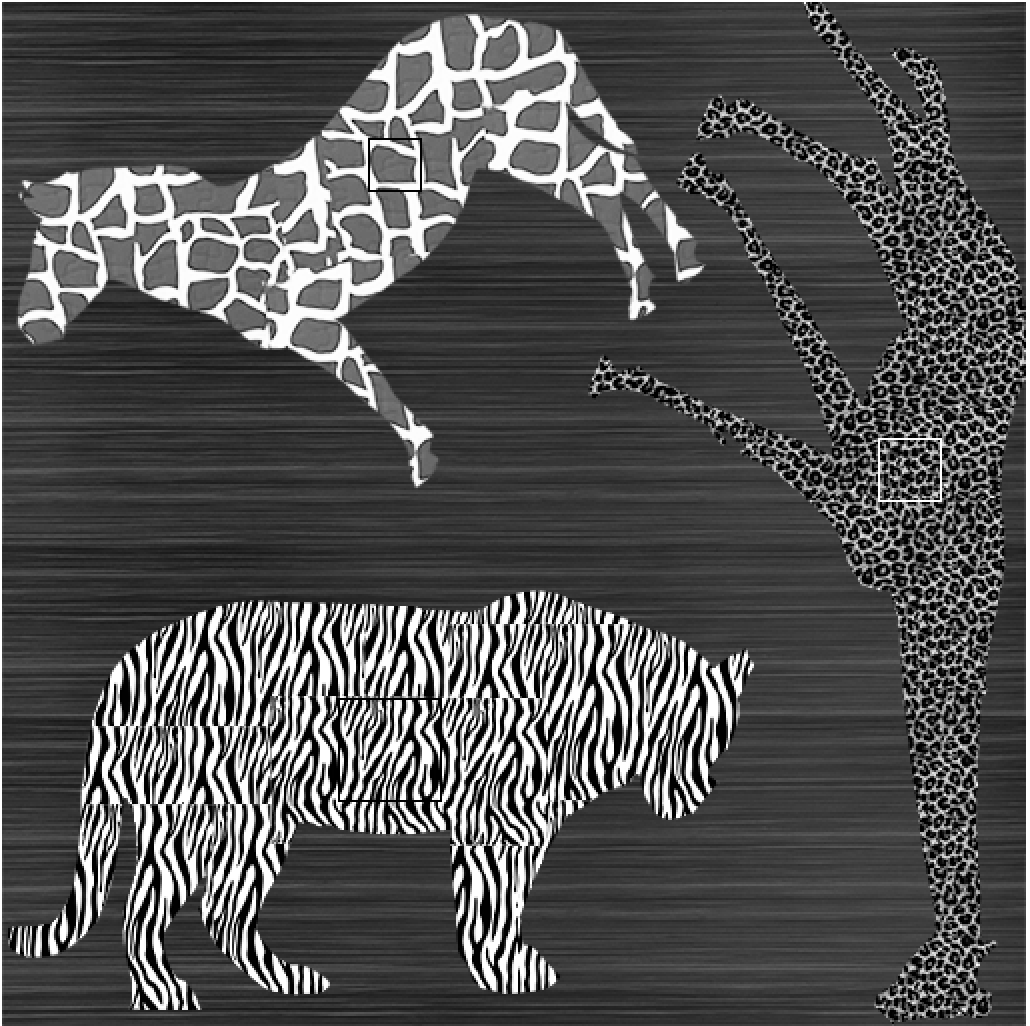


Figure 7: the structures of textures that will be repeating during attaching

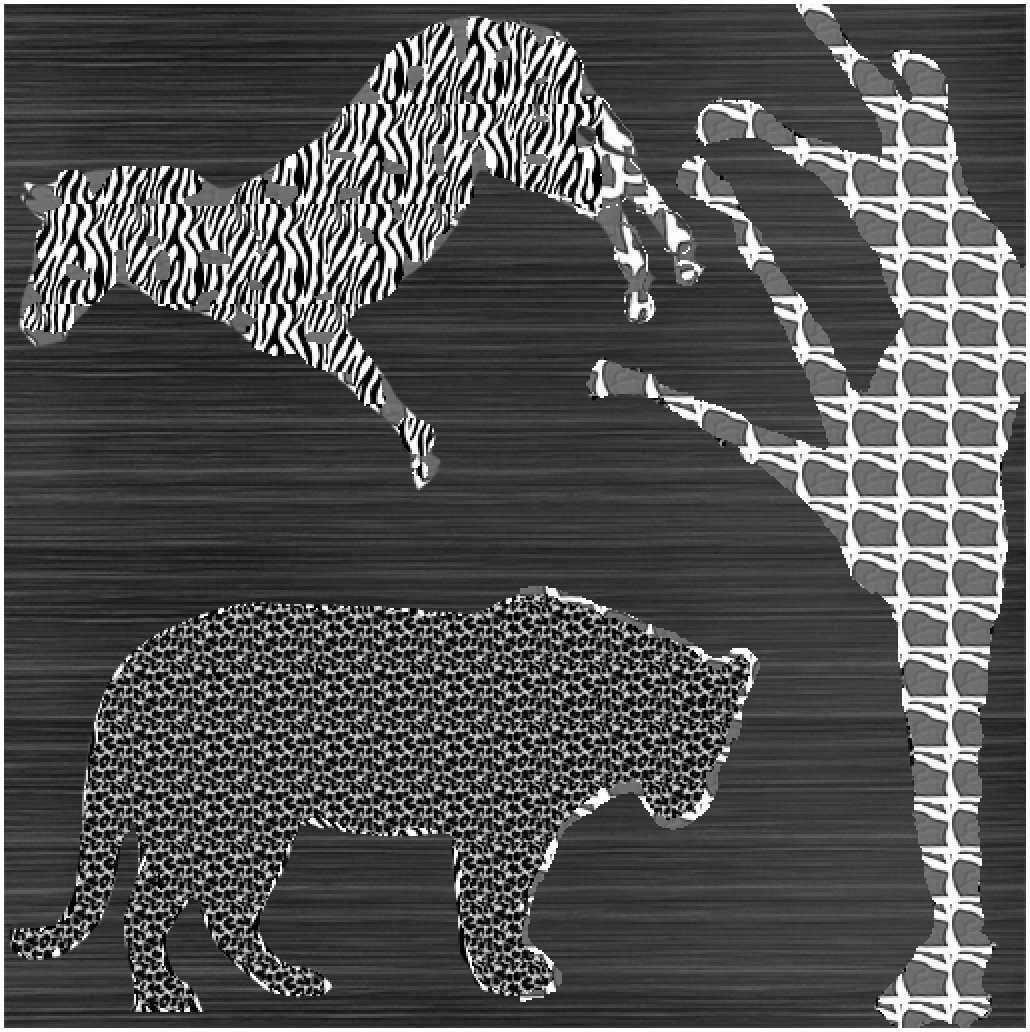


Figure 8: the resultant after attaching the correct textures to the corresponding animals

**Execution**

To reproduce the result, simply execute README.m under Matlab environment.

// 記得檢查所有圖、方程式、表格的標籤！