**CSE585/EE555:  Digital Image Processing II**

**Computer Project # 4:**

**Texture Segmentation**

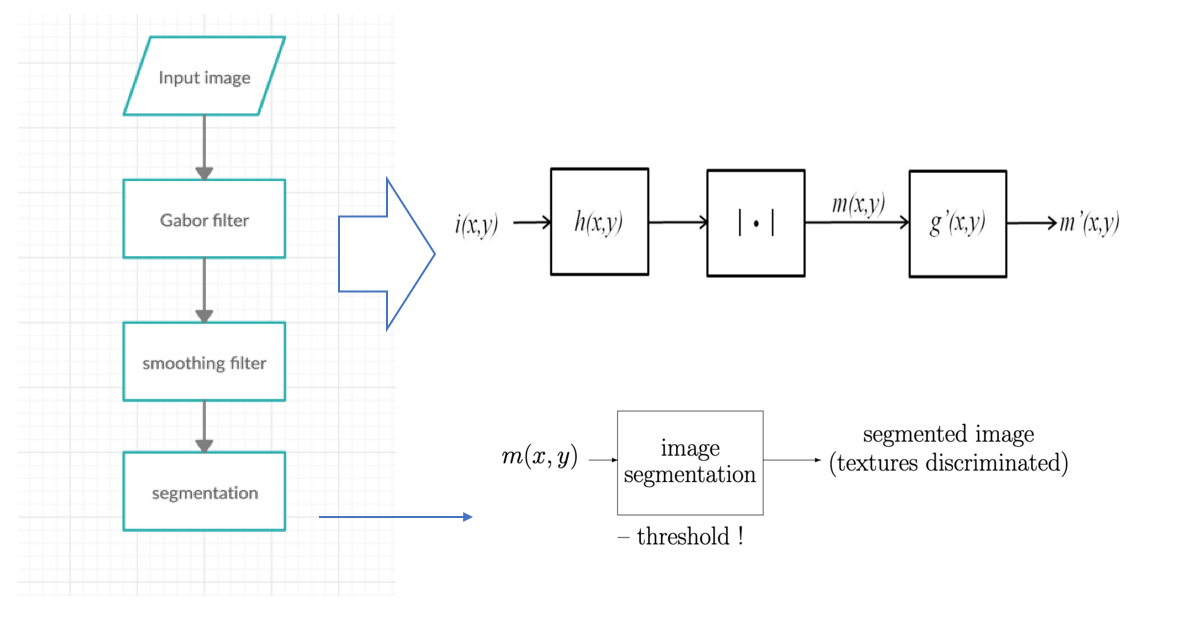
**Yanxi Yang, Jiuchao Yin, Hongjie Liu**

**Date: 04/06/2020**

1. **Objectives**

* To classify and segment bipartite texture regions in an image.
* To understand the principles of Gabor filter and how to design a single Gabor filter on textured image to accomplish optimal segmentation.
* To learn the algorithm of Gaussian part (low pass filter) and complex sinusoid part of Gabor elementary function (GEF).
* To investigate the parameters specifying the Gabor filter and smoothing filter.
* To get familiar with image 3-D plot from texture analysis from Gabor filter and smoothing filter.
* To extract boundaries between major textures regions and to explore the criteria of defining a good segmentation of texture.

1. **Methods**
2. **Flow chart**



**Figure1. flow chart of Gabor filter texture segmentation**

1. **Code structure:**

* **main.m:** the main code runs the 4 different tasks based on 4 images with various parameters settings, and output the grayscale images and 3D plot with m(x,y), m’(x,y) and final segmented results.

**support functions:**

* **g.m: the Gaussian part of GEF** This function is used to compute the circularly-symmetric Gaussian of one pixel. The inputs are the value of sigma and the x/y coordinate of the pixel. The input includes only x or only y since we compute convolution in x and y separately in main function. The output is the circularly-symmetric Gaussian value of that pixel. This function does not call any other functions.
* **hx.m: GEF of h(x)** Since we compute GEF in x and y separately in main function, is separated into and . This function is used to compute the GEF in x, i.e. . The inputs are F, theta, sigma, and x coordinate of the pixel. The output is the computation result of x coordinate. This function does not call any other functions.
* **hy.m: GEF of h(y)** This function is the same as hx.m. The only difference between them is that the inputs of this function are F, theta, sigma, and y coordinate of the pixel, not x. This function is used to compute the GEF in y, i.e. and does not call any other functions.
* **segment.m:** This function is used to do segmentation with discriminative threshold of each texture for classification. The inputs are results after Gabor filter ( or ), the original image, sigma, and threshold. The output is the visualized segmentation result. This function does not call any other functions.

1. **Algorithms**

We implemented Gabor filter following by a smoothing filter for the purpose of texture segmentation.

### 3.1 Garbor Filter:

We first apply Garbor filter to the input image:

m (x,y) = [I(x,y) \*\* h(x,y)], [1]

where I denotes the input image, h is a GEF:

h(x,y) = g(x, y) exp [j2πF(xcosθ + ysinθ)] = g(x, y) exp [j2π (Ux + Vy)]. [2]

In equation 2, θ is the orientation of sinusoid, and g is a circularly-symmetric Gaussian:

g(x,y) = exp {} [3]

The assumption of this project is that Φ = 0, so we can implement the GEF separable for x and y:

h (x,y) = h1(x) h2(y) [4]

h1(x) = exp {} exp {j2πFcosθ} [5]

h2(y) = exp {} exp {j2πF sinθ} [6]

Then the Gabor filter can be applied to the input image through three steps:

* i1(x,y) = i(x,y) \* h1(x)

= [7]

* i2 (x,y) = i1(x,y) \* h2(y)

= [8]

* m(x,y) = | i2 (x,y) | [9]

Thus the Gabor filter has a width of (4).

### 3.2 Smoothing Filter

Smoothing filter will be applied to m(x,y) when m(x,y) is noisy for better segmentation.

m’(x,y) = m(x,y) \* g’(x,y) [10]

where g′ is another circular-symmetric Gaussian defined in equation 3 using a different .

### 3.3 Segmentation

Based on the result of the step function m(x,y) or m’(x,y), we selected a threshold that can separate the step function into two parts. We then overlay it with the original image for segmentation visualization.

### 3.4 Parameters

Parameters used for each image are shown in table 1.

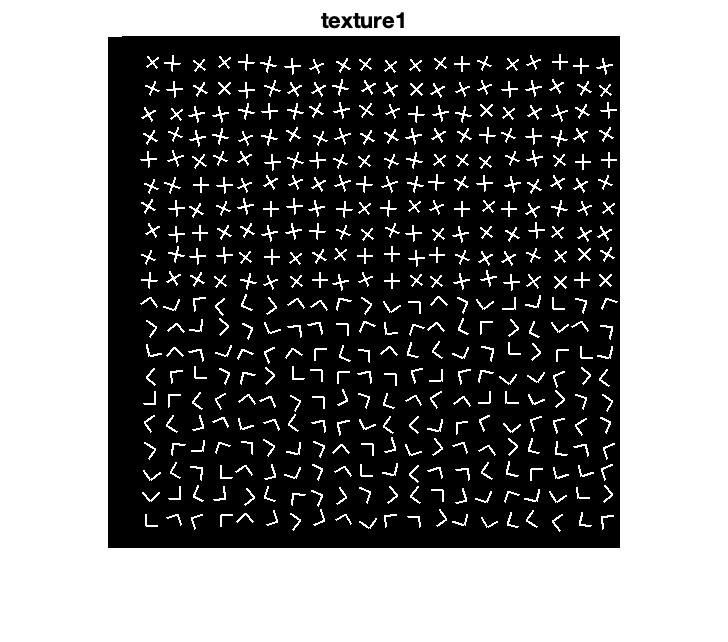
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Gabor Filter | | | Smooth Filter | Segmentation |
| F | θ |  |  | Threshold |
| Texture 1 | 0.042 | 0 | 24 | 24 | 3.6e-9 |
| Texture 2 | 0.059 | 135 | 8 | 24 | 9.1e-8 |
| d9d77 | 0.063 | 60 | 36 | / | 0.5e-3 |
| d4d29 | 0.6038 | -50.5 | 8 | 30 | 9.9e-9 |

Table 1 Parameters to segment different images used in this project.

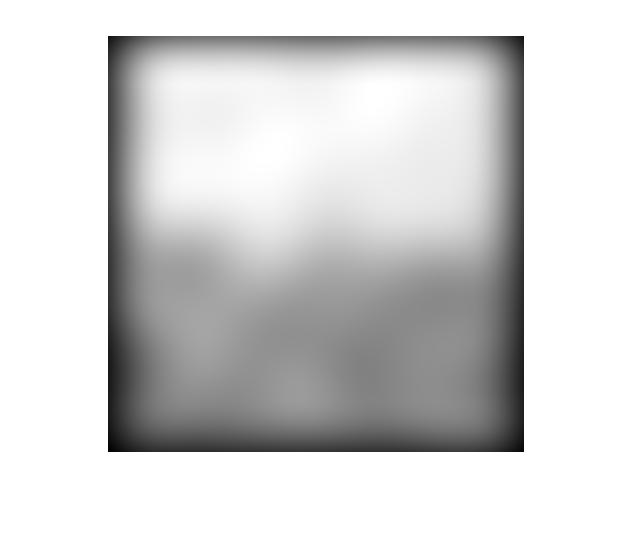
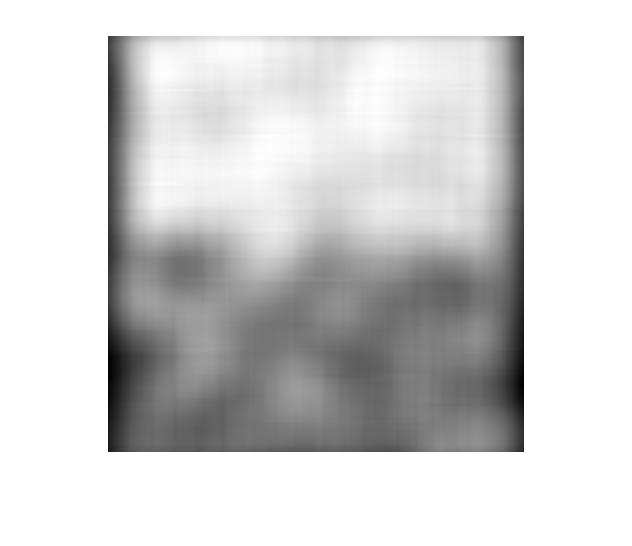
1. **Results**

Since that values near the outer perimeter of the image cannot be completely processed, thus, when displaying results, we have zeroed out unprocessed perimeter areas.

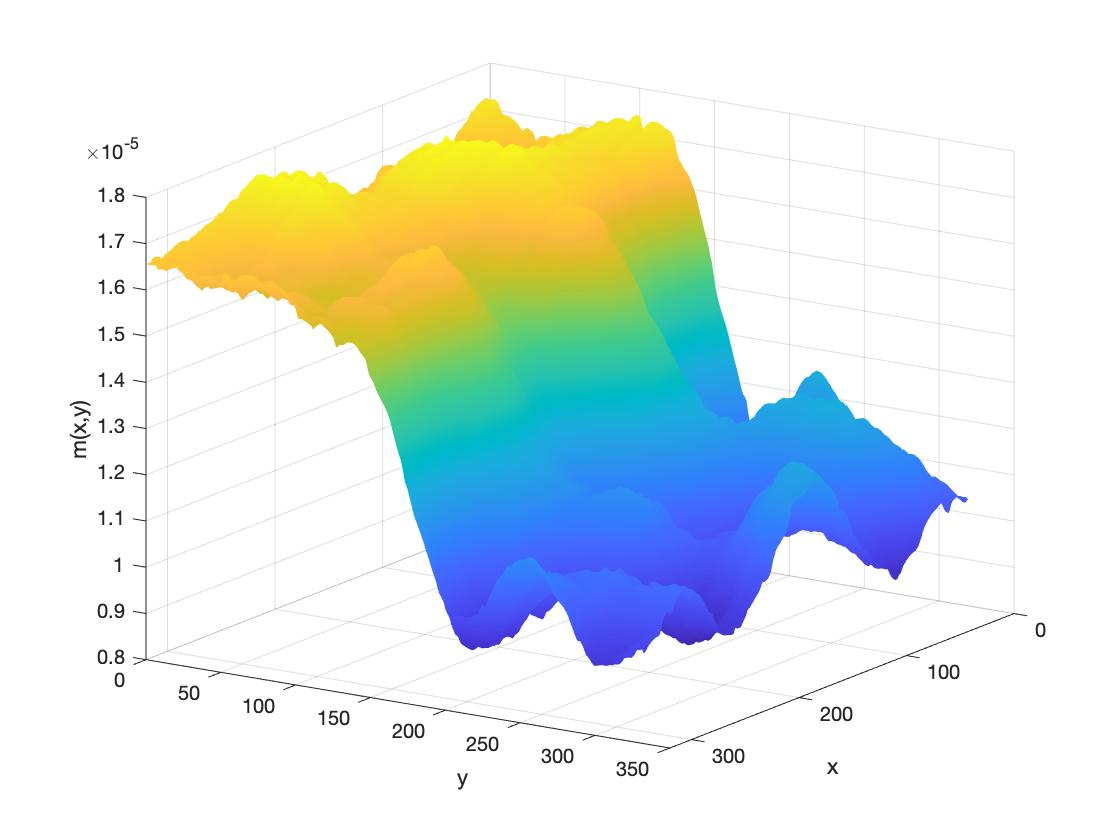
Image1:



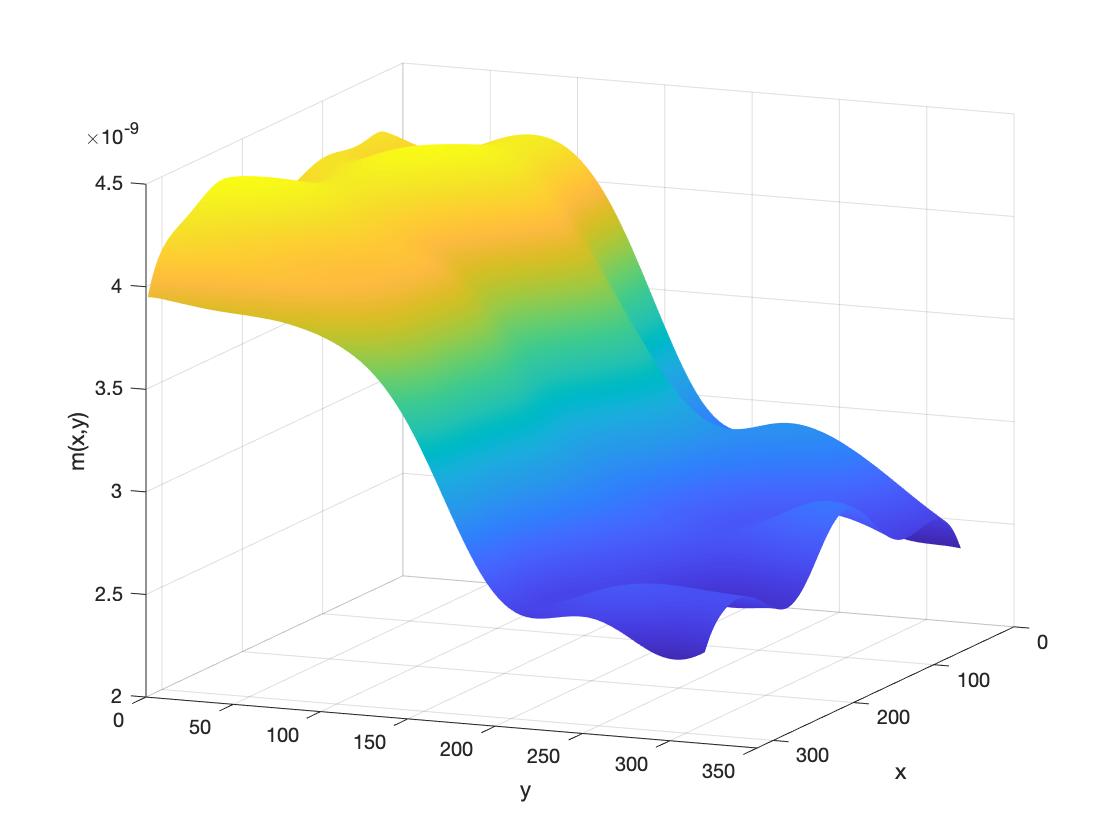
**Figure 2. binary image of ‘texture1’**



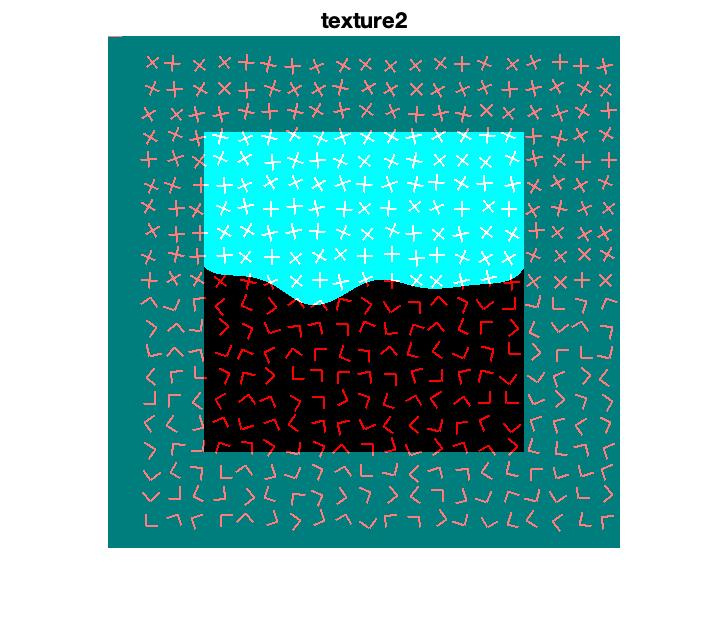
**Figure 3. grayscale image of Gabor filter (left) and smoothing filter (right)**



**Figure 4. 3D plot of ‘texture 1’ after Gabor filter**

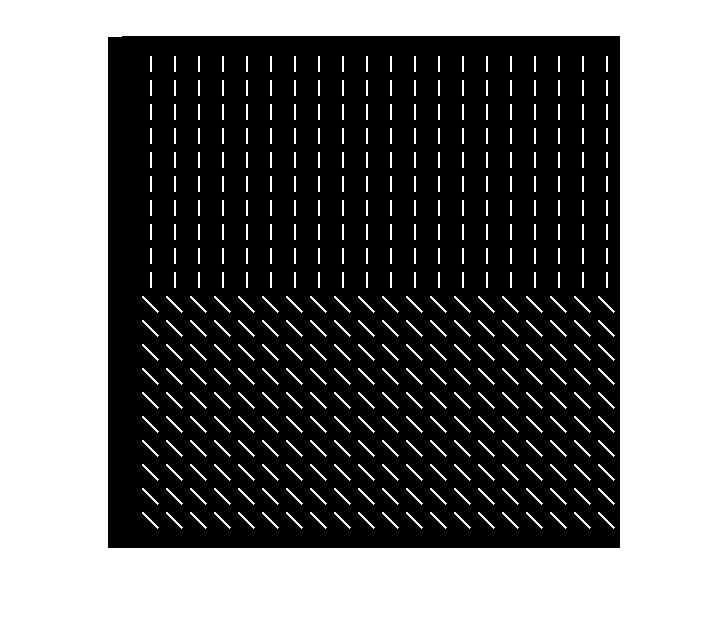


**Figure 5. 3D plot of ‘texture 1’ after Gabor filter and smoothing filter**

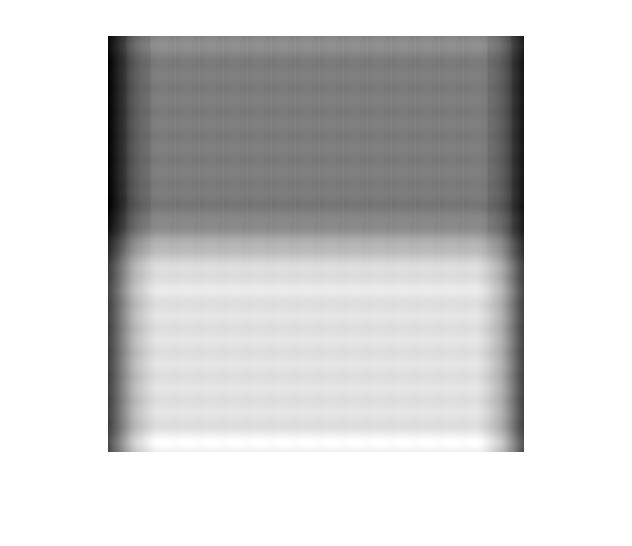
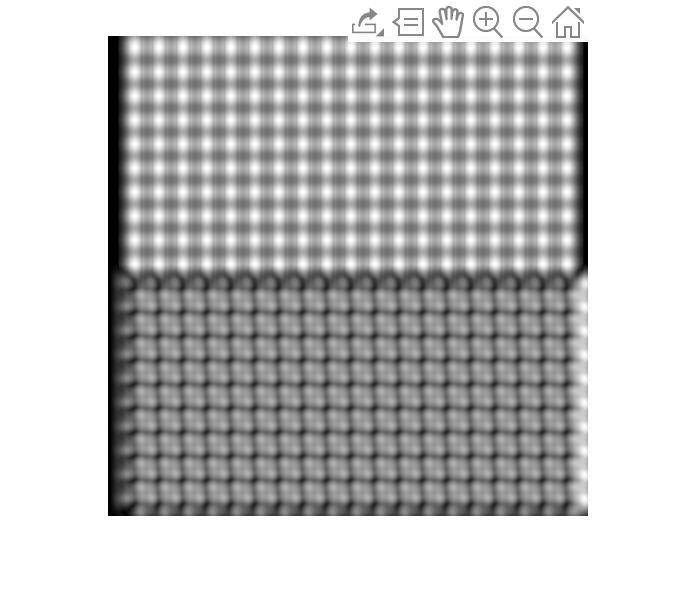


**Figure 6. superimposition of texture segmentation of ‘texture1’**

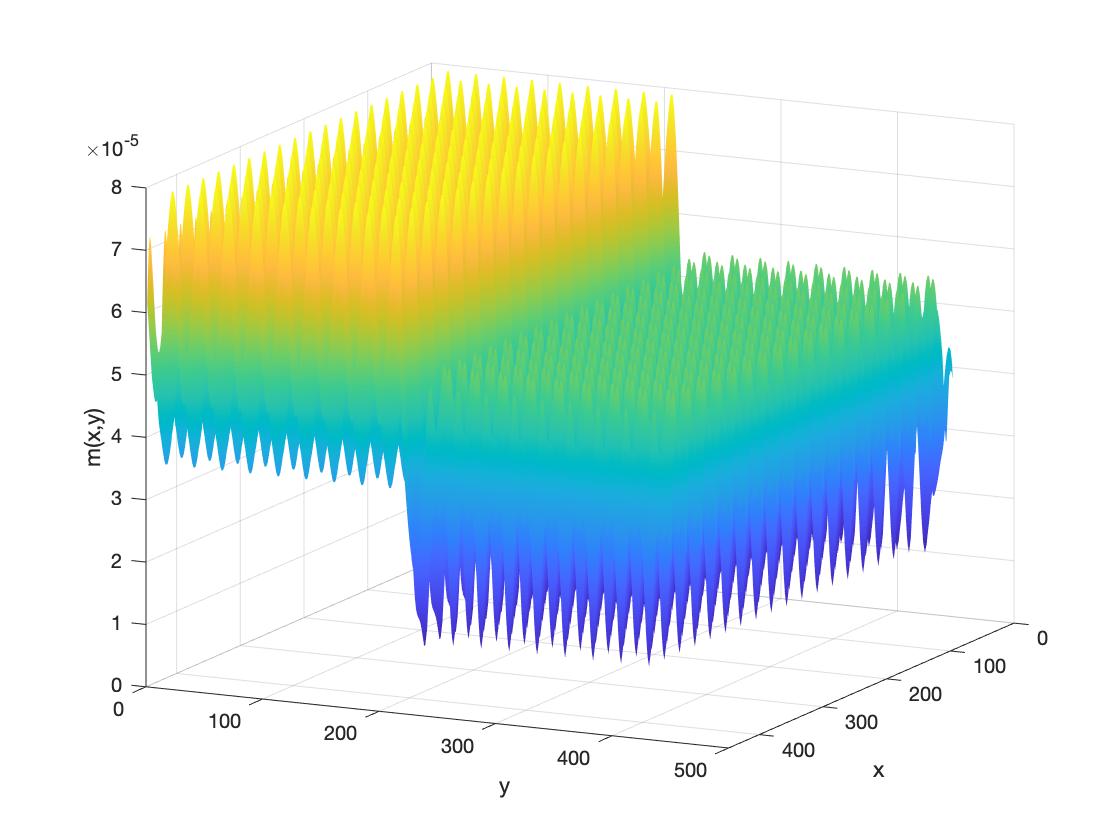
image2:



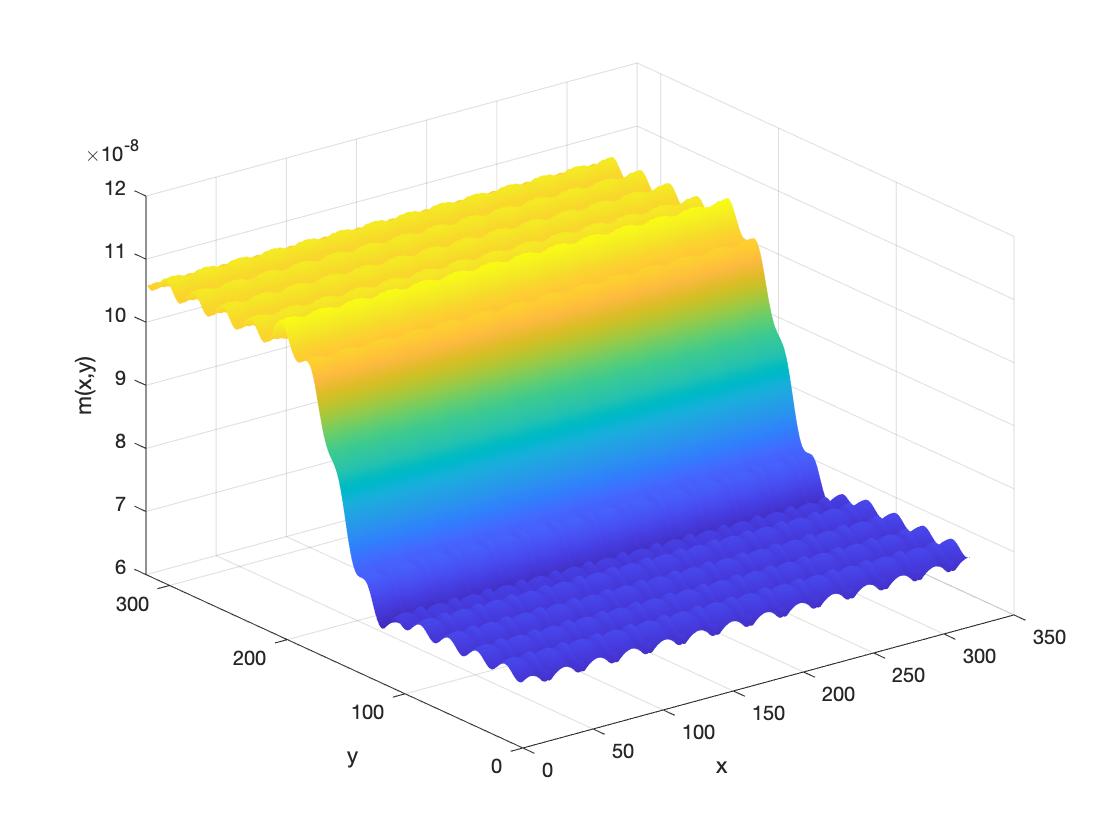
**Figure 7. binary image of ‘texture 2’**



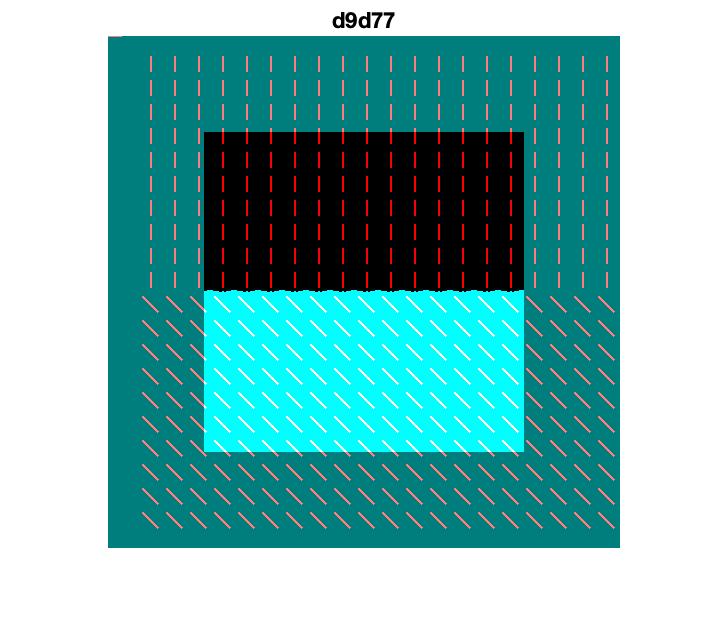
**Figure 8. grayscale image of Gabor filter (left) and smoothing filter (right)**



**Figure 4. 3D plot of ‘texture 1’ after Gabor filter**



**Figure 5. 3D plot of ‘texture 1’ after Gabor filter and smoothing filter**



**Figure 6. superimposition of texture segmentation of ‘d9d77’**

Observation:

image3 :

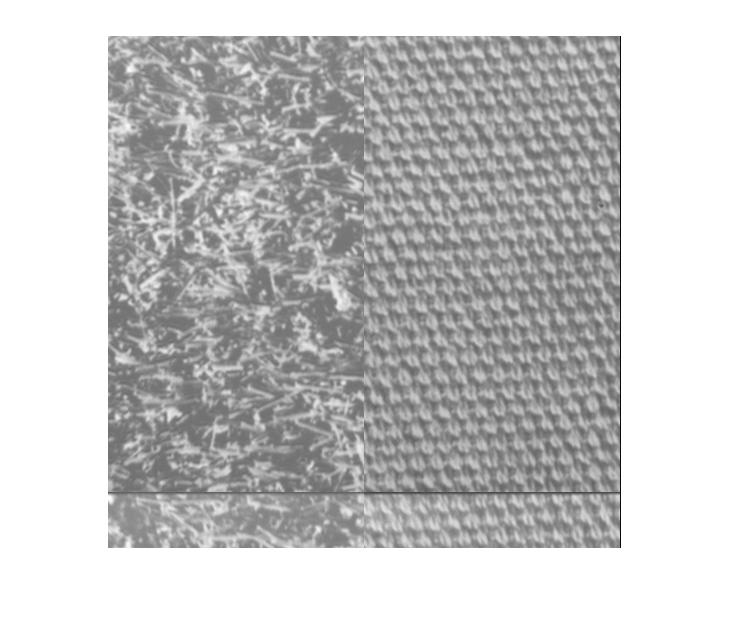
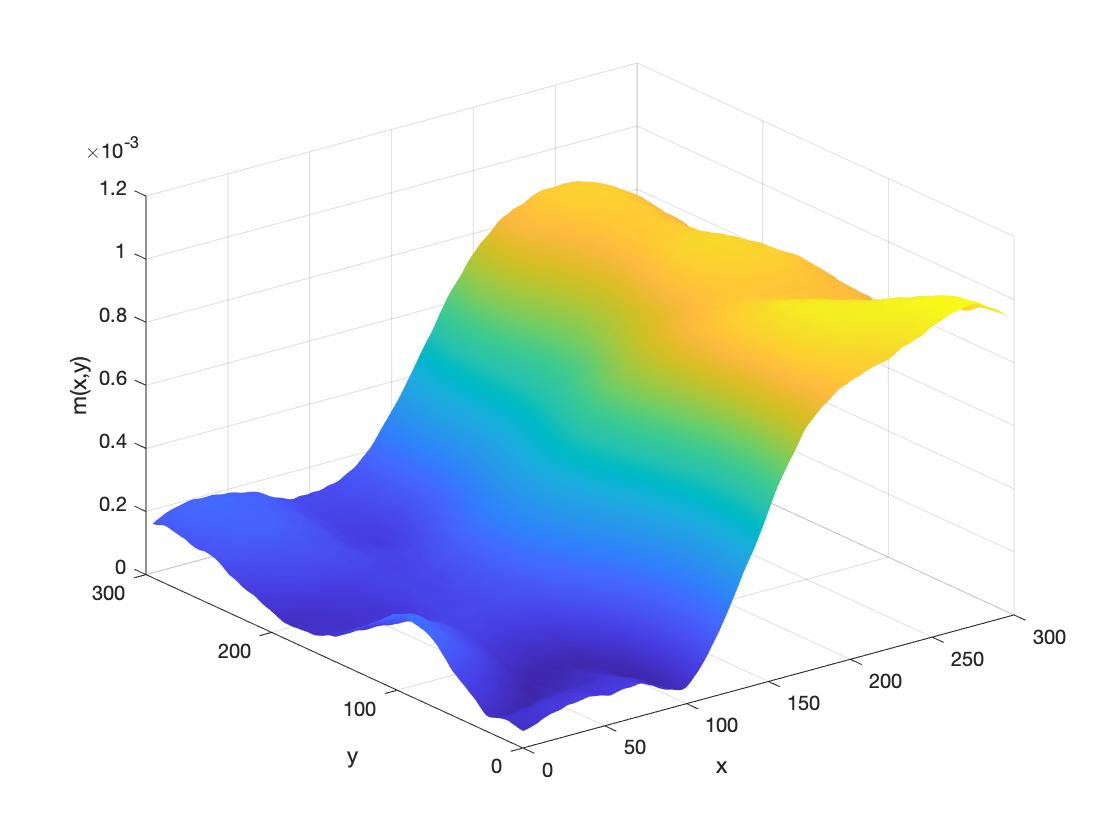
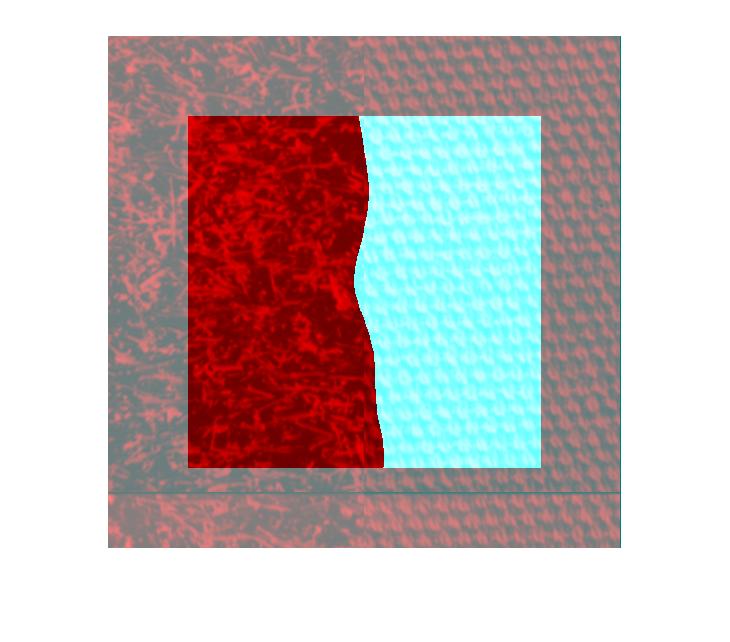
 

Figure 3. grayscale image of Gabor filter (left) and smoothing filter (right)

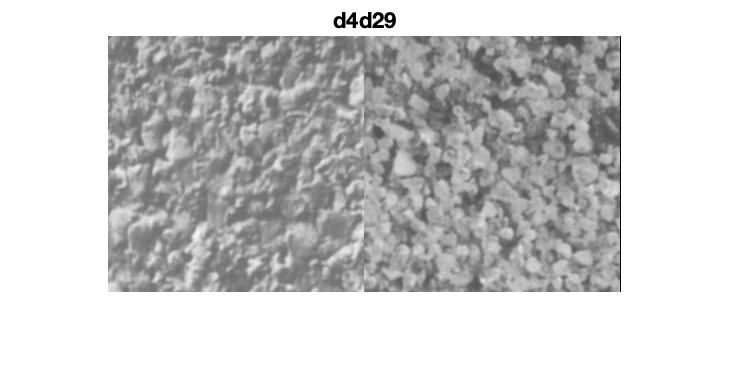


**Figure 4. 3D plot of ‘texture 1’ after Gabor filter**



Observation:

image4:



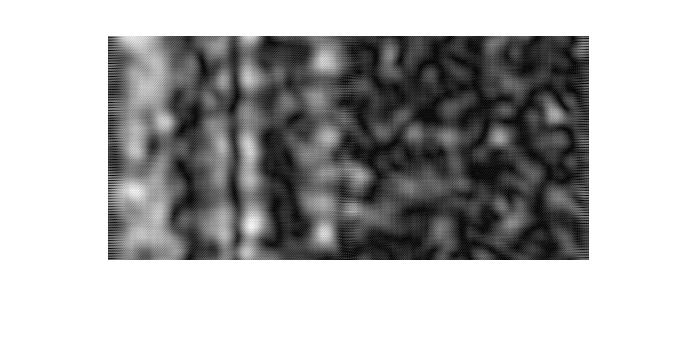
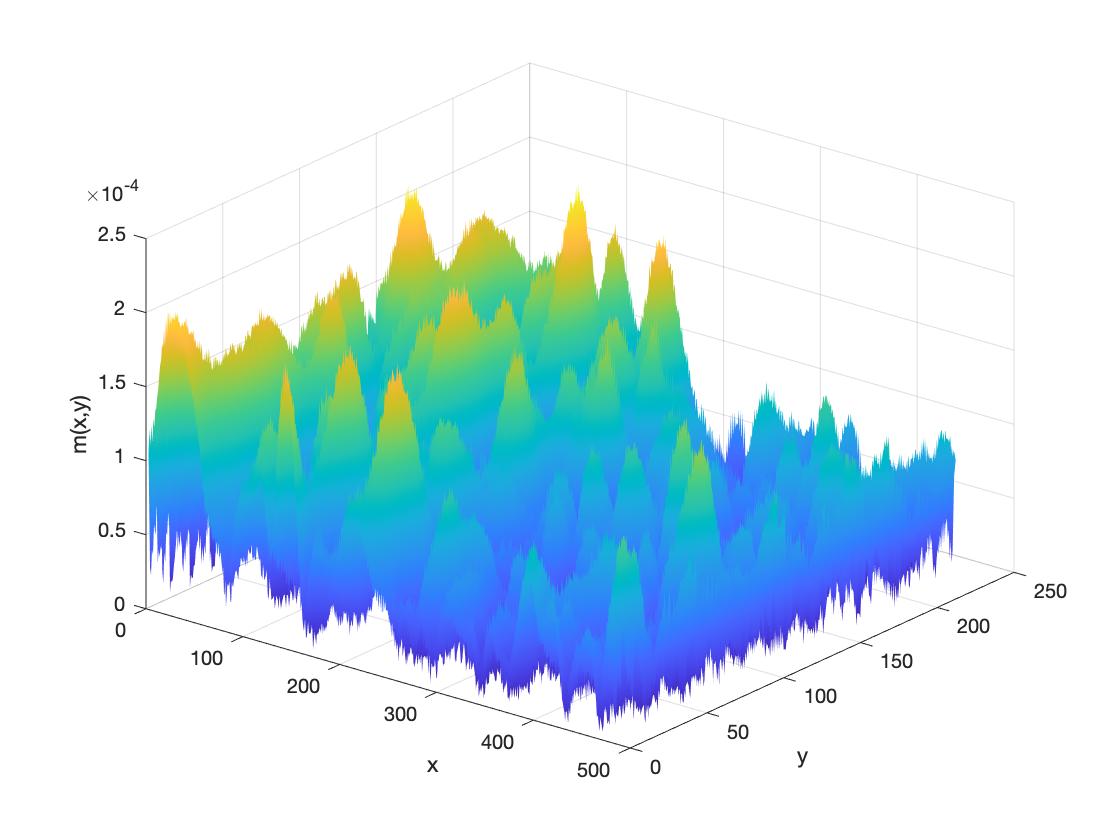
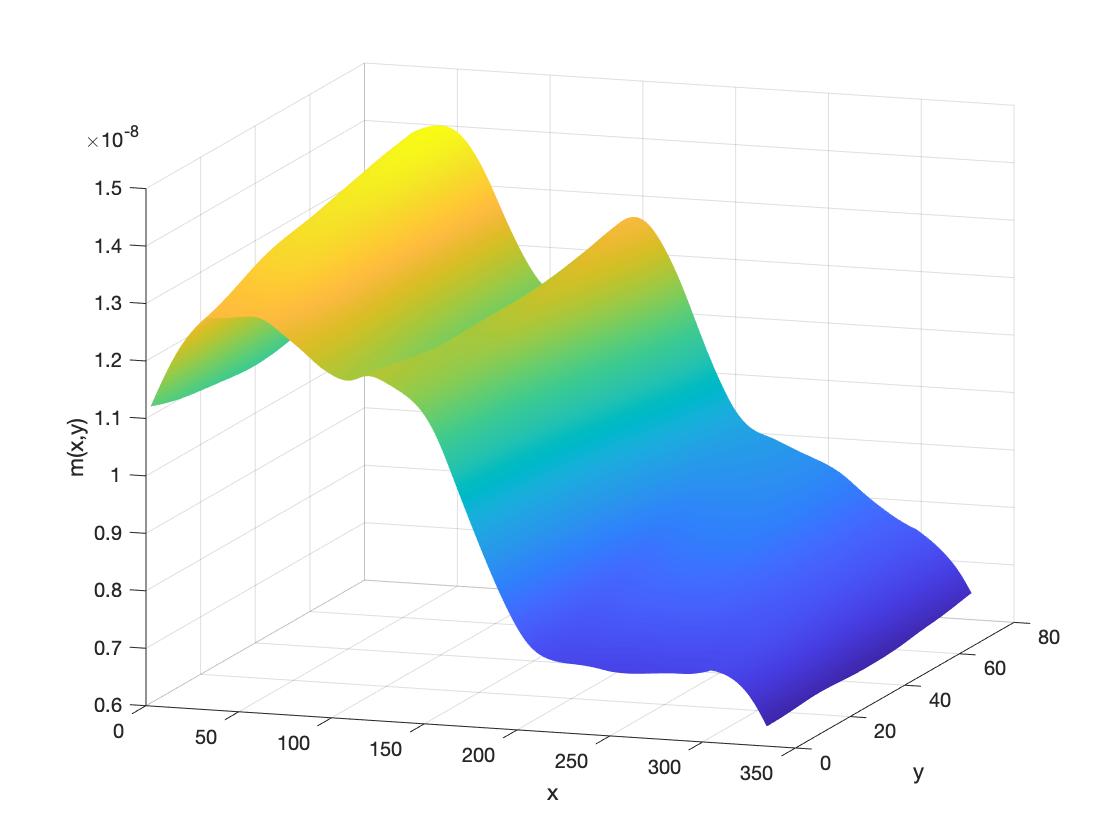


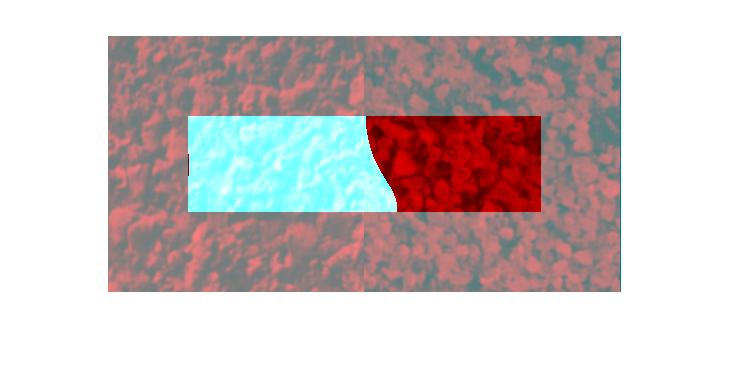
Figure 3. grayscale image of Gabor filter (left) and smoothing filter (right)



**Figure 4. 3D plot of ‘texture 1’ after Gabor filter**



**Figure 5. 3D plot of ‘texture 1’ after Gabor filter and smoothing filter**



Observation:

## Discussion

In this project, the second image whose texture includes ‘-‘ and ‘\’ was best separated, since they are well aligned. The edge separating the two textures ‘+’ and ‘L’ in the first image was slightly curved due to their random directions. This also happened in image ‘d9d77’ because the two textures in ‘d9d77’ are closer to each other. Separating textures in the last image ‘d4d29’ was the hardest because the two textures are very similar.

Since the edge (2 \* ) could not be properly processed at all, values in these regions were set to 0. As a result, pixels located up to 4 \* to the edge could all be affected as half of the values in equation 7 and 8 were zeros. Such region could be large when values are relatively big. The x and y limits delineating the segmentation for each image were given in table 2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | x1 | x2 | y1 | y2 |
| Texture 1 | 97 | 416 | 97 | 416 |
| Texture 2 | 97 | 416 | 97 | 416 |
| d9d77 | 81 | 432 | 81 | 433 |
| d4d29 | 81 | 176 | 41 | 433 |

Table 2 x and y limits delineating the segmentation for each image.

1. **Conclusion**

In conclusion, the project