DIP Homework Assignment #4

資工三 b05902115 陳建丞

PROBLEM 1: Hough transform for line detection

(a) Perform edge detection

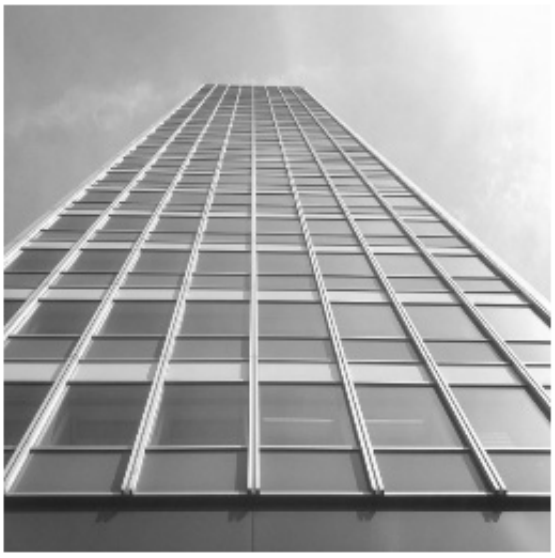
- **Motivation**

  The input image has no complex structure inside. Therefore, I just simply apply the 1st order edge detection. The outline will be clear. Set up a higher threshold for 1st order edge detection so that there will be less edge points remain, which is better for Hough transform. In previous homework, the result shows that using Prewitt Mask and Sobel Mask has very little difference. Hence, I just simply apply Sobel Mask.

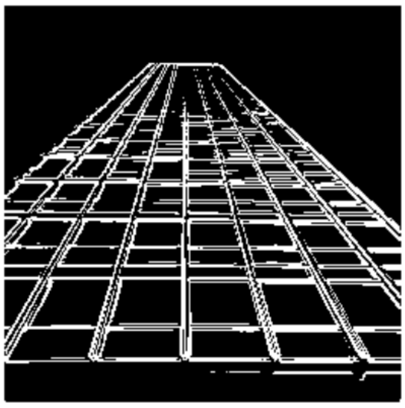
- **Approach**

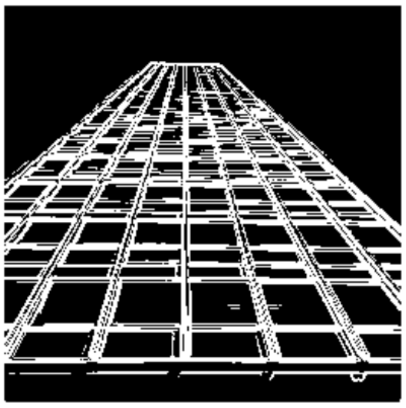
  1. Store the pixel values in a 2d array.
  2. Expand the original image to deal with the border condition.
  3. Set up mask constant $K = 2$ (Sobel Mask)
  4. Calculate the gradient (use two for loop to run through)
     - $G_r(j,k) = \frac{1}{K+2}[(A_2 + KA_3 + A_4) - (A_0 + KA_7 + A_6)]$
     - $G_c(j,k) = \frac{1}{K+2}[(A_0 + KA_1 + A_2) - (A_6 + KA_5 + A_4)]$
     - $\Rightarrow G(j,k) = \sqrt{(G_r^2(j,k) + G_c^2(j,k))}$
  5. Pick up a threshold T
     - if $G(j,k) < T \rightarrow$ Set pixel value to 0 (Non-edge point)
     - if $G(j,k) > T \rightarrow$ Set pixel value to 255 (Edge point)

- **Original image**



- **Result**


T=50


T=30

- **Discussion**

  We can see that if the threshold T is higher, the edge remaining will be less. The original image is not complex so the edge detection can detect the outline clearly.

(b) Perform Hough transform and output the accumulator array

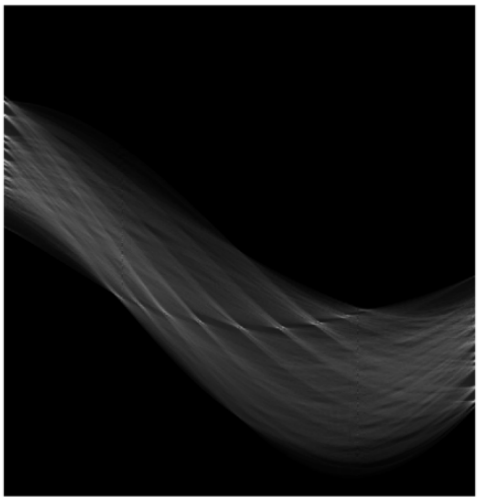
- **Motivation**

  In Hough transform, we calculate the appear times of each rho and theta value. The more the value appears, the line is more important. Since the values of rho are between $-256\sqrt{2}$ and $256\sqrt{2}$, which are approximately 730 different value, I split theta into 720 segments to match it.

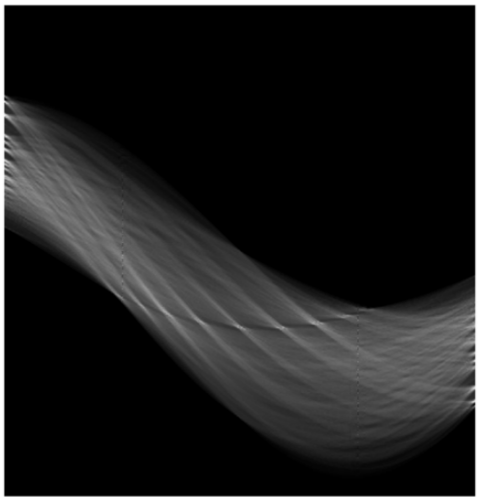
- **Approach**

  1. Create a look-up cos value and sin value table. Otherwise, it tooks lots of time to calculate these values again and again in the process afterwards.
  2. Fill the accumulator
     - $\rho = xcos\theta + ysin\theta$
     - Since the ρ value might be negative and a float number, we round the values to int. And then add 375 in case that the index of array out of range.
     - accumulator(ρ, θ) add 1.
  3. Output to image according to accumulator values.

- **Result**



T=50



T=30

- **Discussion**

  The shape of the results of two different input edge map image has no difference. The output of $T = 30$ is brighter because lower threshold allows more edges remain. Therefore, the accumulator has more lines which makes it more brighter is reasonable.

(c) Perform contrast adjustment

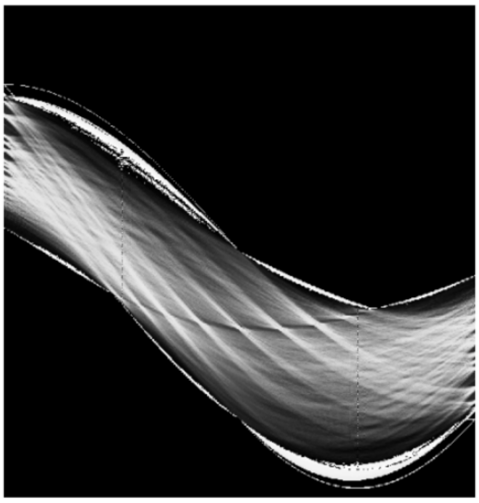
- **Motivation**

  I apply global histogram equalization to enhance contrast. Through this approach, the intensities be better distributed on the histogram. Since the original edge image has lots of black pixels, the black pixels are not counted in the histogram equalization process.

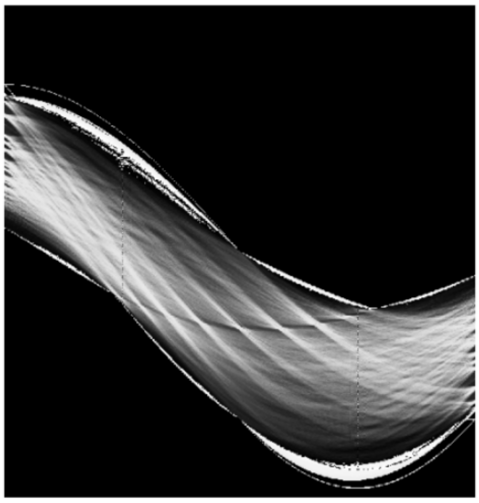
- **Approach**

  1. Use a loop to calculate the frequency of each gray scale level in an array.
  2. Use the frequency array to calculate CDF.
  3. New gray scale level = original gray scale level x CDF / Total pixels

- **Result**



T=50



T=30

- **Discussion**

  I think that my original image is not bad, only a bit darker. After the histogram equalization, the lines become brighter. Most parts of the result image have great contrast, but some dark lines in the original image become white lines. Maybe I can set up a threshold to let some dark line not to turn into a bright line to create a better result.

(d) Draw significant lines with different colors

- **Motivation**

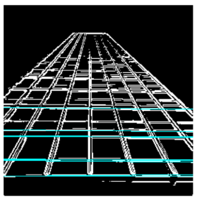
  Since the edge map has many lines close to one another, the accumulator values might concentrate on some specific rho and theta values. In such condition, the output significant lines will concentrate on specific line area, too. To handle such problem, I use a local maximum method, which will be demonstrated below.

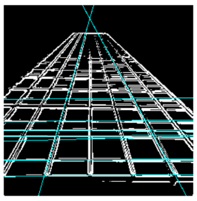
- **Approach**

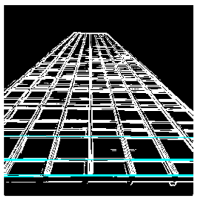
  1. Use the accumulator from previous problem.
  2. Find the top 10 and top 20 accumulator values and record the corresponding rho and theta values.
     - In this process, if we find one local maximum accumulator values, we set the other accumulator values around this pixel to 0. Therefore, we will not pick up similar accumulator again.
  3. Go through the original image , if $\rho = xcos\theta + ysin\theta$ match the top 10 / top 20 accumulatorm, adjust the image colors.

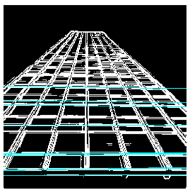
- **Result**

  - D1 (Top 10 lines)

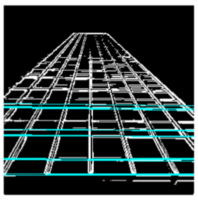

T=50

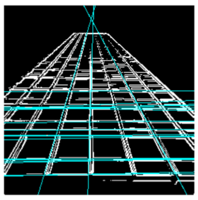

T=50
Local Max

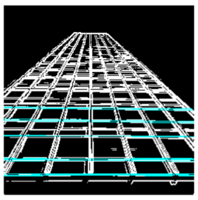

T=30

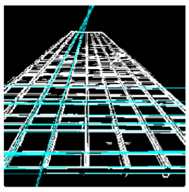

T=30
Local Max

  - D2 (Top 20 lines)


T=50


T=50
Local Max


T=30


T=30
Local Max

- **Discussion**

From the result, we can observe that $T = 50$ edge detection is better for Hough transform. Less edge lines and local maximum method will make the Hough transform not concentrate on specific lines. Therefore, it can detect more lines instead of focusing on several places.

- **Pros and cons of Hough transform**

  Hough transform can handle the line detection well, not much affected by noise or excess edge point. The line it detects in quite accurate. However, Hough transform takes lots of calculation and iterations, which takes lots of time. It's a time-consuming process. Also, Hough transform can detect lines, but it cannot detect the exact length of the line. It can only show that the lines go in such directions.

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**.