REPORT: HW1 CAP 5400

Submitted by: Shivam Srivastava
U-09993569

Main Function:
void hw1()
Takes user input from a file and determines valid ROIs, and images for source and target. For each ROI in each image, the respective function is called: doubleBinarize(), meanSmoothFilter() and colorBinarization().

2.(a) Region of Interest (ROI):

Region of Interest refers to the specific part of the data on which action is to be performed.

Input for ROI:

- x, y Co-ordinates of the left top pixel of ROI.
- Sx, Sy Size of ROI

Implementation:
Using the inputs provided by the user, x-y coordinates of the top-left and bottom-right pixels are calculated and a user-defined structure “roi”.

```
struct roi{ x1, y1, x2, y2;};
```

A *vector<struct roi> validROIs* is used to store all the non-overlapping ROIs. The preference for ROI is decided on the sequence in which it is provided by the user.
With an initially empty vector validROIs, all the ROIs are first checked if they are a non-overlapping ROI and if yes, they are added to this vector to maintain a list of approved ROIs. Also, along with this, it is passed to perform its operation on the target image. In case ROI turns to be an overlapping ROI, they are ignored and not processed.

Functions:
static bool isRoiOverlapping(roi a, roi b):
To compare the approved ROI a and new ROI b and determine if they are overlapping or not.
Condition: a.x1 < b.x2 and a.x2 > b.x1 and a.y1 < b.y2 and a.y2 > b.y1

static bool isRoiValid(vector<roi > validROIs, struct roi current_roi):
To validate the dimension provided for ROI and then using the helper function determining if the ROI is overlapping or not.

2.(b) Image Double Thresholding

Function:

static void doubleBinarize(image &src, image &tgt, struct roi current_roi, int t1, int t2)

Source image,
Target image,
Current ROI coordinates,
Threshold values t1, t2

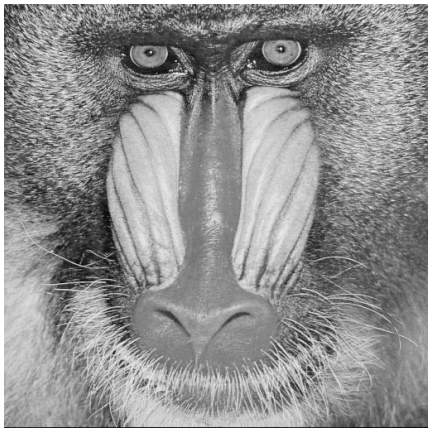
For each valid ROI, the function doubleBinarize is called with the above parameters.

Algorithm:

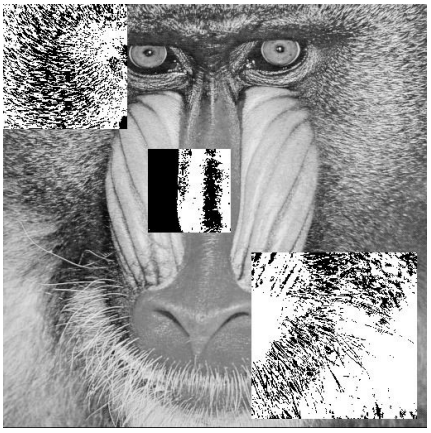
For each pixel inside the ROI, (Complexity for looping through pixels in the image: $O(n^2)$)

if the value of the pixel is in between t1 and t2, then
Set Pixel value to 255,
else,
Set Pixel value to 0.

Samples:



(a)



(b)

Parameters for Image (b):

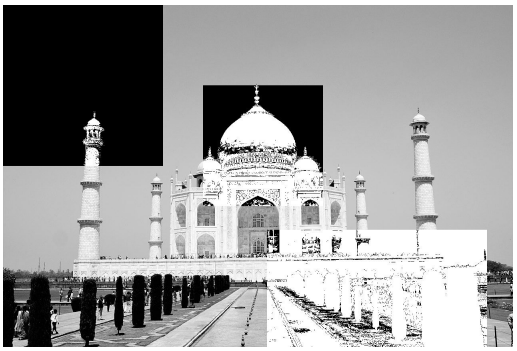
(x1,y1)(x2,y2) = (100,100) (400,400) 50 < T < 150
(x1,y1)(x2,y2) = (650,500) (200,300) 80 <T < 170
(x1,y1)(x2,y2) = (100,200) (300,300) 100 < T < 160 <- ROI Ignored
(x1,y1)(x2,y2) = (900,1100) (200,350) 50 < T < 200

(c)

(b)

Parameters for Image (d):
(x1,y1)(x2,y2) = (0,0) (150,150) 50 < T < 150
(x1,y1)(x2,y2) = (350,350) (160,160) 100 < T < 200
(x1,y1)(x2,y2) = (100,200) (300,300) 100 < T < 160 <- ROI Ignored
(x1,y1)(x2,y2) = (175,175) (100,150) 80 < T < 150

(e)



(f)

Parameters for Image (f):
(x1,y1)(x2,y2) = (0,1000 (400,400) 50 < T < 150
(x1,y1)(x2,y2) = (560,660) (300,550) 120 <T < 170
(x1,y1)(x2,y2) = (200,500) (300,150) 150 < T < 200

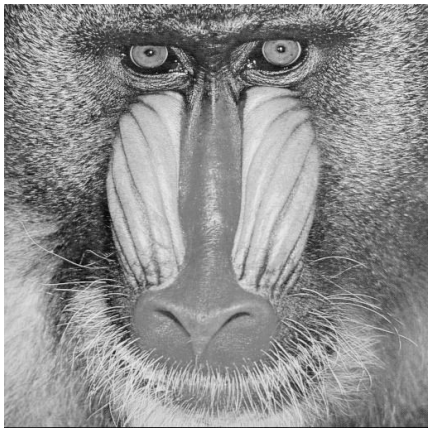
2.(c) Uniform Adaptive Smoothening

Function:
static void meanSmoothFilter(image &src, image &tgt, struct roi current_roi, int windowSize)
Source image,
Target image,
Current ROI coordinates,
Window Size

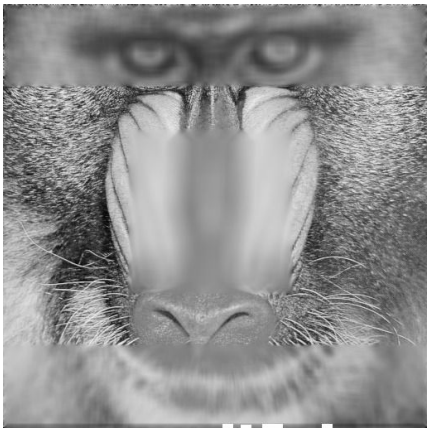
For each valid ROI, the function meanSmoothFilter is called with the above parameters.
Algorithm:
For each pixel at (i,j) inside the ROI, (Complexity for looping through pixels in the image: $O(n^2)$)
{
    Calculate current window dimensions with boundary conditions for adaptive window size:
        If complete or part of window lies outside the ROI, then
            Decrease the window size to the next smaller odd size >=3.
    For each pixel in window: Complexity $O(w^2)$
        Calculate sum
        Mean = sum / windowSize * windowSize
        Set pixel value at (i,j) to Mean.
}

Findings:
For large images and large window size, the function was slow.
The blur amount is directly proportional to window size.

Samples:



(a)



(b)

Parameters for Image (b):
(x1,y1)(x2,y2) = (0,0) (100,512) WS = 15150
(x1,y1)(x2,y2) = (150,150) (200,200) WS = 33
(x1,y1)(x2,y2) = (410,0) (200,512) WS = 9
(x1,y1)(x2,y2) = (300,100) (200,200) WS = 13 <- ROI Ignored

(c)

(d)

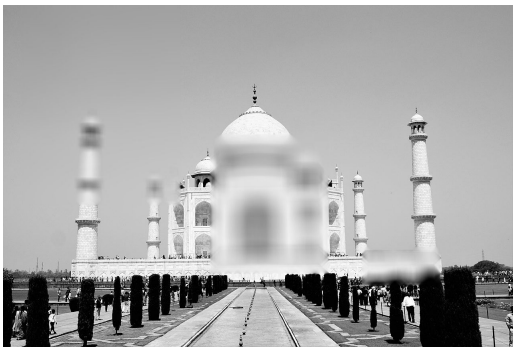
Parameters for Image (d):

(x1,y1)(x2,y2) = (0,50) (500,400) WS = 23

(x1,y1)(x2,y2) = (700,1050) (300,300) WS = 35

(x1,y1)(x2,y2) = (500,700) (300,225) WS = 9

(e)



(f)

Parameters for Image (f):

(x1,y1)(x2,y2) = (0,50) (400,400) WS = 23

(x1,y1)(x2,y2) = (600,900) (100,200) WS = 35

(x1,y1)(x2,y2) = (320,520) (350,300) WS = 55

(x1,y1)(x2,y2) = (500,500) (300,300) WS = 17 <- ROI Ignored

3.(a) Color Thresholding

This function takes as input color and a scalar value for the range. The output is an image with white at the pixels with value that lies in the given distance in R-G-B space, and black otherwise.

Function:

static void colorBinarization(image &src, image &tgt, struct roi current_roi, r, g, b, tc)

Source image,

Target image,

Red value for color point,

Green value for color point,

Blue value for color point,

For each valid ROI, the function colorBinarization is called with the above parameters.
Algorithm:
For each pixel at (i,j) inside the ROI, (Complexity for looping through pixels in the image: $O(n^2)$)
{
this(R1,G1,B1) = get channel values at (i,j)
If Eucledian distance from this(R1,G1,B1) to input(R,G,B) < threshold
Set Pixel value to 255 for R, G and B,
else,
Set Pixel value to 0 for R, G and B.
}
Eucledian Distance:
Dist (A, B) = SQRT((A.x-B.x)*(A.x-B.x) + (A.y-B.y)*(A.y-B.y))

Samples:

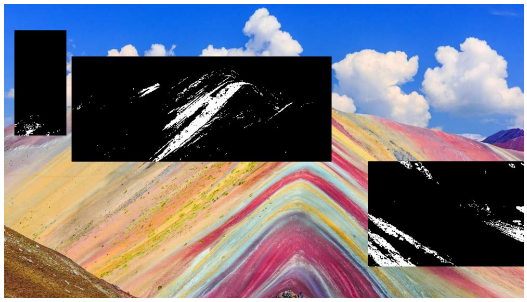
(a)

(b)

Parameters for Image (b):

(x1,y1)(x2,y2) = (100,100) (900,800)	(R,G,B) = (100,70, 120)	TC=120
(x1,y1)(x2,y2) = (2500,2500) (600,600)	(R,G,B) = (200,0,0)	TC=80
(x1,y1)(x2,y2) = (1500,2000) (1000,200)	(R,G,B) = (200,10,10)	TC=100

(c)



(d)

Parameters for Image (d):

(x1,y1)(x2,y2) = (100,130) (200,500)	(R,G,B) = (70 100 70)	TC=80
(x1,y1)(x2,y2) = (100,100) (400,400)	(R,G,B) = (100 120 150)	TC=80
(x1,y1)(x2,y2) = (100,100) (400,400)	(R,G,B) = (150 80 80)	TC=60 <- ROI Ignored
(x1,y1)(x2,y2) = (100,100) (400,400)	(R,G,B) = (150 80 80)	TC=60

Execution

Edit the following files in project/bin/.

1. **parameter.txt :** each line executes a function with the given input file, output file and parameters in the corresponding 'param_hw1_' files.

   <input_image> <output_image> function-name

input_image	path to input image from project/bin/
output_image	path to save output image from project/bin/
Function name	Function name key from: [doubleBinarize, ---------, ----------]

E.g.

parameter.txt :

baboon.pgm baboon_grey_doublebin.pgm doubleBinarization
baboon.pgm baboon_grey_smooth.pgm smoothFilter
landscape.ppm landscape_color_bin.ppm colorThreshold

2. **Parameter files**
   a. **param_hw1_q1.txt:** each line having new ROI and thresholds for the input image for doubleBinarization().
   <roi_x> <roi_y> <size_x> <size_y> <threshold_1> <threshold_2>

roi_x and roi_y	co-ordinates of the top-left pixel,
size_x and size_y	width and length of ROI,
threshold_1	lower limit of pixel value,
threshold_2	upper limit of pixel value

   E.g.
   0 0 150 150 50 150
   300 300 200 200 100 200
   100 200 300 300 100 160
   175 175 100 100 80 150

   b. **param_hw1_q2.txt:** each line having new ROI and thresholds for the input image for smoothFilter().
   <roi_x> <roi_y> <size_x> <size_y> <window_size>

roi_x and roi_y	co-ordinates of the top-left pixel,
size_x and size_y	width and length of ROI,
Window Size	Size of window - validity: Positive odd integer > 3

Eg:
0 0 100 100 15
120 70 100 100 33
200 300 200 200 9

c. **param_hw1_q3.txt:** each line having new ROI and thresholds for the input image for colorThreshold().
<roi_x> <roi_y> <size_x> <size_y> <R> <G> <B> <threshold>

roi_x and roi_y	co-ordinates of the top-left pixel,
size_x and size_y	width and length of ROI,
R	red value for the color,
G	green value for the color,
B	blue value for the color,
threshold	Distance from color point (R,G,B) in RGB-Space

Eg:
5 5 1000 1000 70 100 70 100
1100 1100 700 700 90 90 90 100
510 510 300 300 180 70 70 50