CSC320 Assignment2 Winter 2015

Soon Chee Loong

February 22, 2015

Contents

1 Introduction

This paper is where I present my results for Assignment2 for CSC320 Winter 2015. As there is no need to repeat details already described in the handout, please refer to the painterlyRendering.pdf for initial information for each parts of this assignment. The original code given is called paintrend.py. At each part, I described changes that I made to the code with respect with the previous parts. For Part 1 in particular, I described changes I made to this paintrend.py file that was given to us. Also, the images in this report has been re-scaled to fit the report. Original sizes can be viewed by opening the images themselves directly.

2 Executing Code

I have included 6 different code, one for each part of the assignment. Part 3 onwards relies on a file called canny.py. Therefore, you have to include canny.py in the same directory as the code for the parts you want to output.

You also have to ensure both input images used which are orchid.jpg and personal.jpg are in the same directory as the code for every part.

The code can be compiled with Python 2.7.5 with Anaconda's Full installation. The files provided for each parts are part1.py, part2.py, part3.py, part4.py, part5.py, and part6.py. Note: p2.py is exactly the same as part6.py which is the final iteration of the code.

3 Input Images

I am working with two different images. They are called orchid.jpg and personal.jpg. Orchid.jpg is a picture of a red orchid as shown in Figure 1. Personal.jpg is a picture of myself as shown in Figure 2. Both these

pictures are of totally different colours and sizes. Note: The pictures have been re-scaled to fit into this report.

Figure 1: Orchid.jpg, (163,197) pixels

Figure 2: Personal.jpg, (314,362) pixels

4 Part 1

The implementation for part1's code can be found in file part1.py.

4.1 Code Changes

The changes I made are:

- Removed the for loop that iterates 5oo times
- Insert while loop that iterates as long as there are unpainted pixels
- Increment iterative index k outside while loop

```
k = 0 #added to use while loop instead of for
#for k in range(500): #removed for loop that iterates 500 times
while np.any((canvas == -1).flatten('F')): #added while loop
k = k+1# Part1 changes: added increment of k
```

4.2 Parameters

The parameters I used are:

- Stroke Radius: 3
- Maximum Half Lengths: 10

4.3 Results

The results are shown in Figure 3 for the input image of Figure 1 and Figure 4 for the input image of Figure 2.

5 Part 2

The implementation for part2's code can be found in file part2.py.

5.1 Code Changes

The changes I made are:

- Calculate the location of all place where pixels are unpainted
- Calculate a random index from the unpainted pixels
- Get the x and y values from that random index
- Randomly select the stroke's center
- Assign the randomly selected centers as the center of stroke chosen

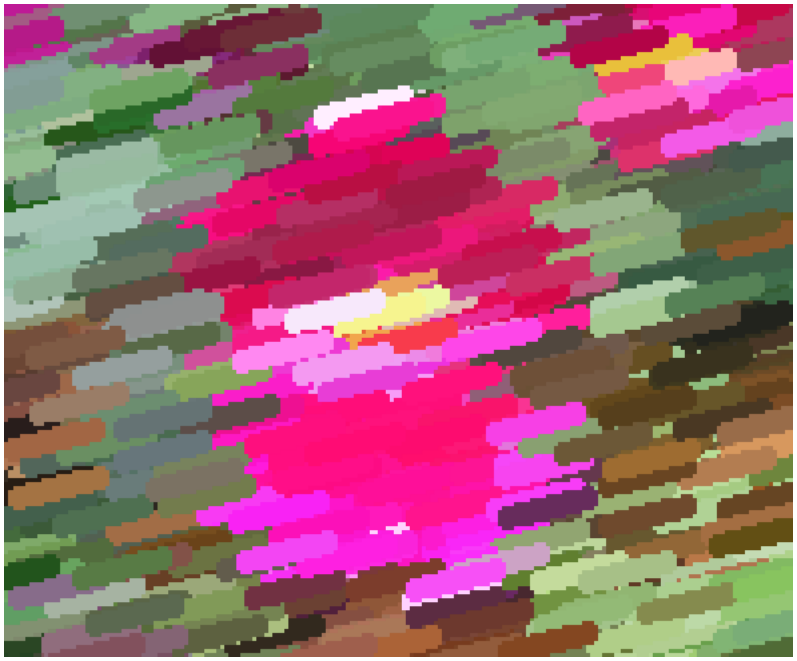

Figure 3: Part 1 Output of Orchid.jpg

```
# Calculate the location of all place where pixels are unpainted
available = np.where(canvas == -1) # store location of unpainted pixels
# Calculate a random index from the unpainted pixels
randomIndex = np.floor(np.random.rand(1,1).flatten() * np.array(len(available[0])
randomIndex = int(randomIndex) # convert to integer
# Get the x and y values from that random index
xSelected = available[0][randomIndex]
ySelected = available[1][randomIndex]
#Randomly select the stroke's center
cntr = np.floor(np.random.rand(2,1).flatten() * np.array([sizeIm[1], sizeIm[0]]))
# Assign the randomly selected centers as the center of stroke chosen
cntr[0] = ySelected + 1 #note: Need add 1 as paintStroke is 1 based
cntr[1] = xSelected + 1 #note: the paintStroke works on opposite x and y!
```

5.2 Parameters

The parameters I used are:

- Stroke Radius: 3

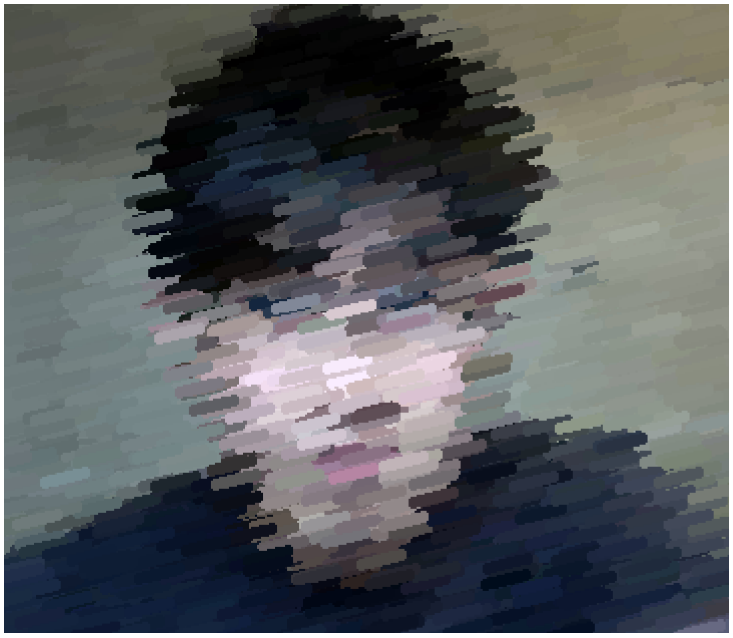

Figure 4: Part 1 Output of Personal.jpg

- Maximum Half Lengths: 10

5.3 Results

The results are shown in Figure 5 for the input image of Figure 1 and Figure 6 for the input image of Figure 2.

6 Part 3

The implementation for part3's code can be found in file part3.py.

6.1 Code Changes

The changes I made are:

- Used a grayscale intensity image with Litwinowicz's suggestion.
- Save the output of the edges into the output pictures

```
for i in range(sizeIm[0]):
```

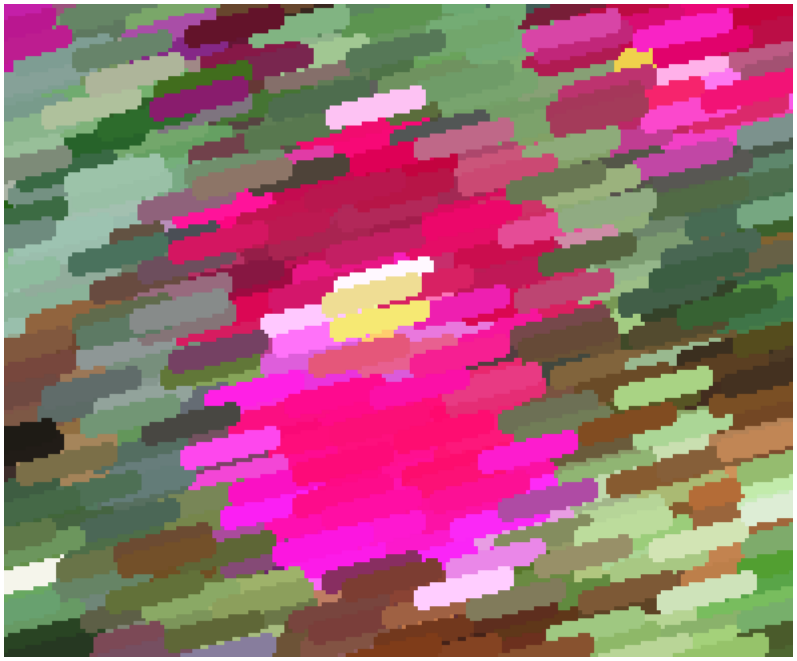

Figure 5: Part 2 Output of Orchid.jpg

```
for j in range(sizeIm[1]):
    # Used a grayscale intensity image with Litwinowicz's suggestion.
    imRGBCanny[i][j] = 0.30*imRGB[i][j][0] + 0.59*imRGB[i][j][1] + 0.11*imRGB
# Save the output of the edges
binaryCanny = canny.canny(imRGBCanny, 2.0, 35, 25)
```

6.2 Parameters

The parameters I used are:

- Stroke Radius: 3
- Maximum Half Lengths: 10
- Gaussian Standard Deviation for Edge Detection: 2.0
- Low Threshold on Minimum Size of Edge Detection: 25
- High Threshold on Minimum Size of Edge Detection: 35

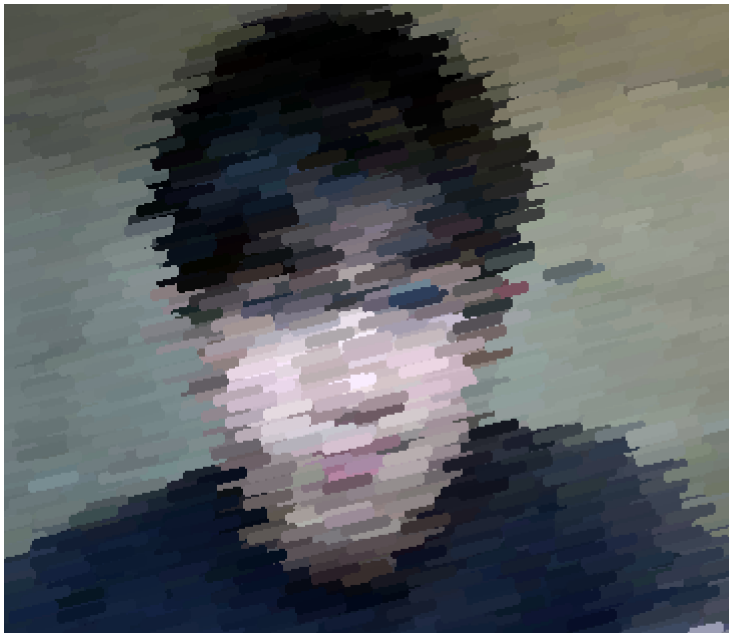

Figure 6: Part 2 Output of Personal.jpg

6.3 Results

The results are shown in Figure 7 for the input image of Figure 1 and Figure 8 for the input image of Figure 2. These figures shows the edges detected for the given parameters above.

7 Part 4

The implementation for part4's code can be found in file part4.py.

7.1 Code Changes

The changes I made are:

- Update delto so that won't walk more than 1 pixel at a time
- Set lengths to 0 if there is an edge on center of pixel
- Change stroke radius to 1 for testing
- Compute Length 1

Figure 7: Part 3 Output Canny Edge Image of Orchid.jpg

Figure 8: Part 3 Output Canny Edge Image of Personal.jpg

- Compute Length 2

```
# update delta so that wont' walk more than 1 pixel at a time
if abs(delta[0]) >= abs(delta[1]):
    delta = delta/abs(delta[0])
else:
    delta = delta/abs(delta[1])
# Set lengths to 0 if there is an edge on center of pixel
if binaryCanny[cntr[1]-1][cntr[0]-1]:
   length1, length2 = (0, 0) # set lengths to 0
# Change stroke radius to 1 for testing
rad = 1


# Compute length 1
k1 = 1 # initialize k1 = 1
while hypot(round(k1*delta[0]) , round(k1*delta[1])) <= length1:
    cntrTemp = cntr.copy()
    cntrTemp[0] = cntr[0] + round(k1*delta[0])
    cntrTemp[1] = cntr[1] + round(k1*delta[1])
    # if reach end of image of max and min(0)
    if cntrTemp[1] > ny or cntrTemp[0] > nx or cntrTemp[1] < 0 or cntrTemp[0] < 0
        length1 = k1
        break
    if binaryCanny[cntrTemp[1]-1][cntrTemp[0]-1]:
        length1 = k1
        break
    k1 = k1 + 1

 # Compute length 2
 k2 = 1 # initialize k2 = 1
 while hypot(round(k2*delta[0]),round(k2*delta[1])) <= length2:
     cntrTemp = cntr.copy()
     cntrTemp[0] = cntr[0] - round(k2*delta[0])
     cntrTemp[1] = cntr[1] - round(k2*delta[1])
     # if reach end of image
     if cntrTemp[1] > ny or cntrTemp[0] > nx or cntrTemp[1] < 0 or cntrTemp[0] <
         length2 = k2
         break
     if binaryCanny[cntrTemp[1]-1][cntrTemp[0]-1]:
```

```
            length2 = k2
            break
    k2 = k2 + 1
```

7.2 Parameters

The parameters I used are:

- Stroke Radius: 1
- Maximum Half Lengths: 10
- Gaussian Standard Deviation for Edge Detection: 2.0
- Low Threshold on Minimum Size of Edge Detection: 25
- High Threshold on Minimum Size of Edge Detection: 35

7.3 Results

The results are shown in Figure 9 for the input image of Figure 1 and Figure 10 for the input image of Figure 2.

8 Part 5

The implementation for part5's code can be found in file part5.py.

8.1 Code Changes

The changes I made are:

- Set halfLen = 5
- Reset radius to 3
- Calculate gradient direction similar to Canny's code
- Changed delta calculation to account for variable theta

```
# Set halfLen = 5
halfLen = 5
# Reset stroke radius to 3
rad = 3 # reset radius = 3 for part 5
```

Figure 9: Part 4 Output of Orchid.jpg

```
# Calculate gradient direction similar to Canny's code
    im = imRGBCannyPart5
    sigma = 4.0
    thresHigh = 20
    thresLow = 10
    imin = im.copy() * 255.0

    # Create the gauss kernel for blurring the input image
    # It will be convolved with the image
    # wsize should be an odd number
    wsize = 5
    gausskernel = canny.gaussFilter(sigma, window = wsize)
    # fx is the filter for vertical gradient
    # fy is the filter for horizontal gradient
    # Please not the vertical direction is positive X

    fx = canny.createFilter([0,  1, 0,
                             0,  0, 0,
```

Figure 10: Part 4 Output of Personal.jpg

```
                        0, -1, 0])
fy = canny.createFilter([ 0, 0, 0,
                        -1, 0, 1,
                         0, 0, 0])

imout = conv(imin, gausskernel, 'valid')
gradxx = conv(imout, fx, 'valid')
gradyy = conv(imout, fy, 'valid')
gradx = np.zeros(im.shape)
grady = np.zeros(im.shape)
padx = (imin.shape[0] - gradxx.shape[0]) / 2.0
pady = (imin.shape[1] - gradxx.shape[1]) / 2.0
gradx[padx:-padx, pady:-pady] = gradxx
grady[padx:-padx, pady:-pady] = gradyy
# Net gradient is the square root of sum of square of the horizontal
# and vertical gradients
grad = hypot(gradx, grady)
theta = arctan2(grady, gradx) #returns signed angle in radian
# note: Don't have to add pi/2 since using canny's fy which is already the negati
```

```
    # Only significant magnitudes are considered. All others are removed
    defaultTheta = 90 # set default theta to be 90
    # removed the minimum gradient completely gives the best results
    #xx, yy = where(grad < 10)
    #theta[xx, yy] = defaultTheta # set to a value that is known
    #grad[xx, yy] = 0

# Changed delta calculation to account for variable theta
    delta = np.array([cos(theta[cntr[1]-1, cntr[0]-1]), sin(theta[cntr[1]-1, cntr[0]-
```

8.2 Parameters

The parameters I used are:

- Stroke Radius: 3
- Maximum Half Lengths: 5
- Gaussian Standard Deviation for Edge Detection: 2.0
- Low Threshold on Minimum Size of Edge Detection: 25
- High Threshold on Minimum Size of Edge Detection: 35
- Gaussian Standard Deviation for Gradient Orientation Estimation: 4.0
- Threshold on Minimum Size of Gradient Orientation Estimation: 10

8.3 Results

The results are shown in Figure 11 for the input image of Figure 1 and Figure 12 for the input image of Figure 2. Visualization of the theta array for input image of Figure 1 and Figure 2 is shown in Figure 13 and Figure 14 respectively.

9 Part 6

The implementation for part6's code can be found in file part6.py.

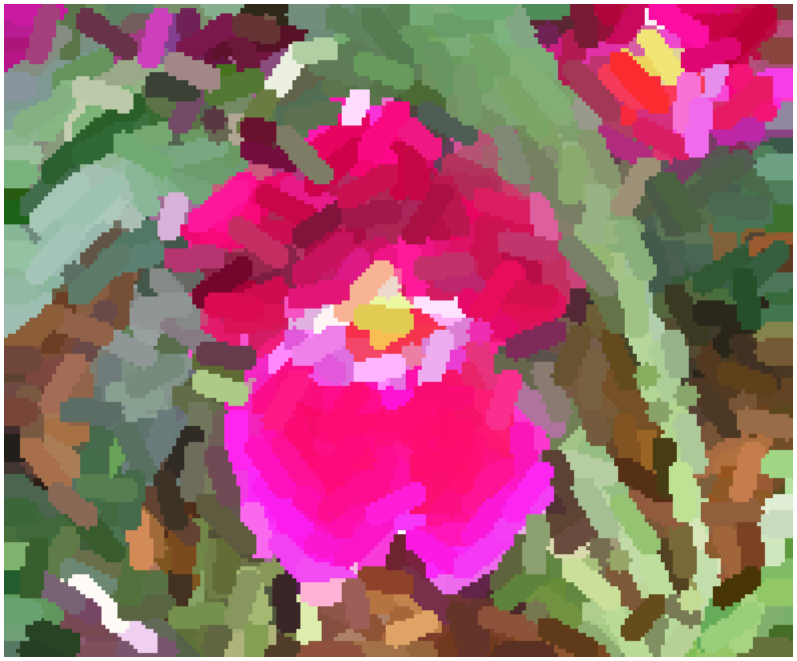

Figure 11: Part 5 Output of Orchid.jpg

9.1 Code Changes

The changes I made are:

- Perturb the stroke orientation by [-15degree, 15 degree]
- Generate random colours changes for each channel with range [-15/255, 15/255]
- Add the generated random colours to each channel
- Scale all component of colours by same random constant
- Clamp the colours

```
# Perturb the stroke orientation by [-15degree, 15 degree]
# randomDegree degree between -15 to 15
randomDegree = (np.random.rand(1,1)[0][0] * 30) - 15
randomDegree = radians(randomDegree)
delta = np.array([cos(theta[cntr[1]-1, cntr[0]-1] + randomDegree), sin(theta[cntr
```

Figure 12: Part 5 Output of Personal.jpg

```
# Add each component of the color separately
colour = np.reshape(imRGB[cntr[1]-1, cntr[0]-1, :],(3,1))
# Generate random color changes for each channel with range [-15/255, 15/255]
# Generate random color changes for Red Channel
randomColour = ((np.random.rand(1,1)[0][0] * 30) - 15)/255  #divide the color by

#  Add the generated random colors to each channel
colour[0] = colour[0] + randomColour
randomColour = ((np.random.rand(1,1)[0][0] * 30) - 15)/255  #divide the color by
colour[1] = colour[1] + randomColour
randomColour = ((np.random.rand(1,1)[0][0] * 30) - 15)/255  #divide the color by
colour[2] = colour[2] + randomColour

#Scale all component of colours by same random constant
# Multiple all 3 color channels by a constant
randomColourScale = (np.random.rand(1,1)[0][0] * 30) - 15   # get between -15 to
randomColourScale = 1  + (0.01 * randomColourScale) # get between 0.85 to 1.15
colour[0] = colour[0] * randomColourScale
colour[1] = colour[1] * randomColourScale
```

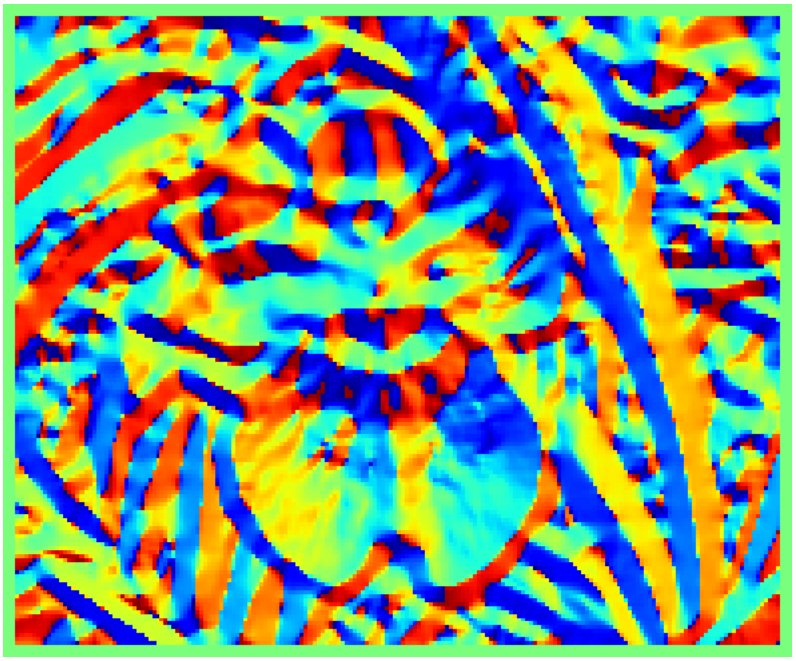

Figure 13: Part 5 Theta Array Visualization of Orchid.jpg

```
colour[2] = colour[2] * randomColourScale

# Clamp the colours
if colour[0] < 0:
    colour[0] = 0
elif colour[0] > 1:
    colour[0] = 1

if colour[1] < 0:
    colour[1] = 0
elif colour[1] > 1:
    colour[1] = 1

if colour[2] < 0:
    colour[2] = 0
elif colour[2] > 1:
    colour[2] = 1
```

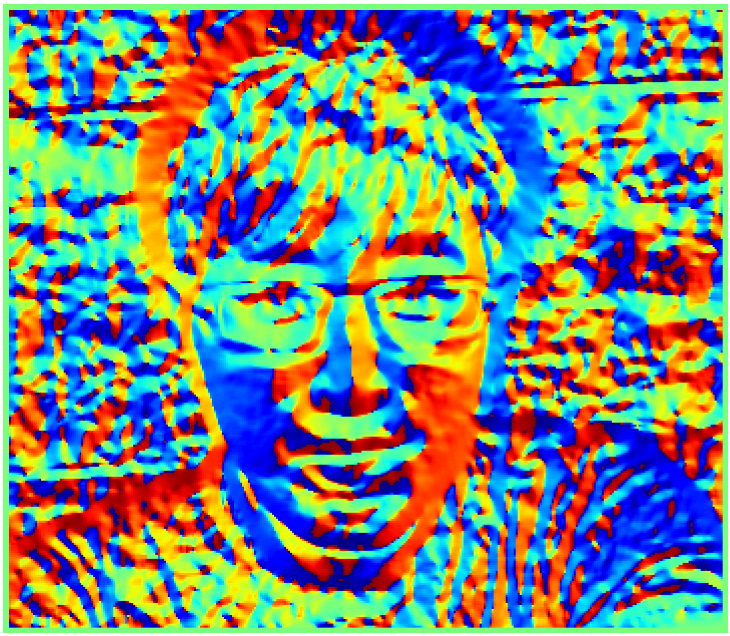

Figure 14: Part 5 Theta Array Visualization of Personal.jpg

9.2 Parameters

The parameters I used are:

- Stroke Radius: 3
- Maximum Half Lengths: 5
- Gaussian Standard Deviation for Edge Detection: 2.0
- Low Threshold on Minimum Size of Edge Detection: 25
- High Threshold on Minimum Size of Edge Detection: 35
- Gaussian Standard Deviation for Gradient Orientation Estimation: 4.0
- Threshold on Minimum Size of Gradient Orientation Estimation: 10

9.3 Results

The results are shown in Figure 15 for the input image of Figure 1 and Figure 16 for the input image of Figure 2.

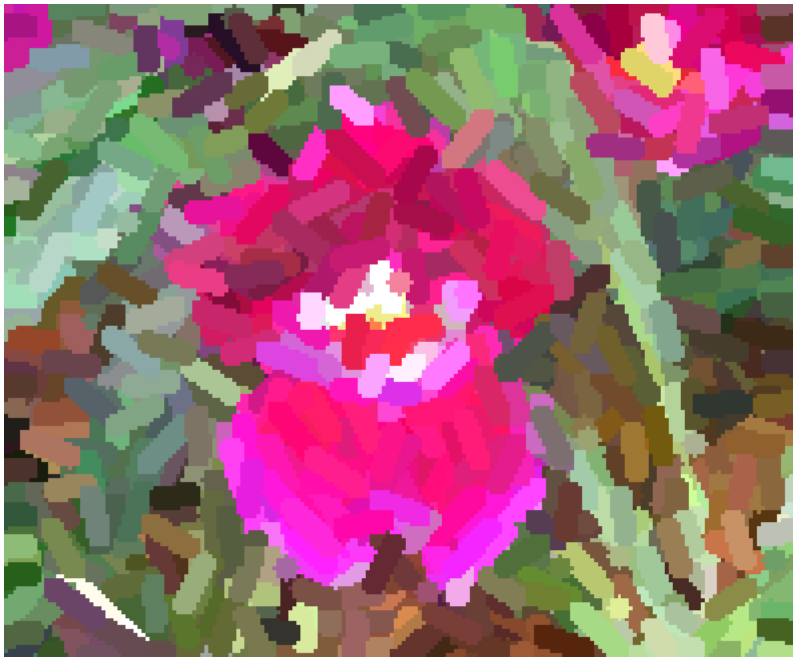

Figure 15: Part 6 Output of Orchid.jpg

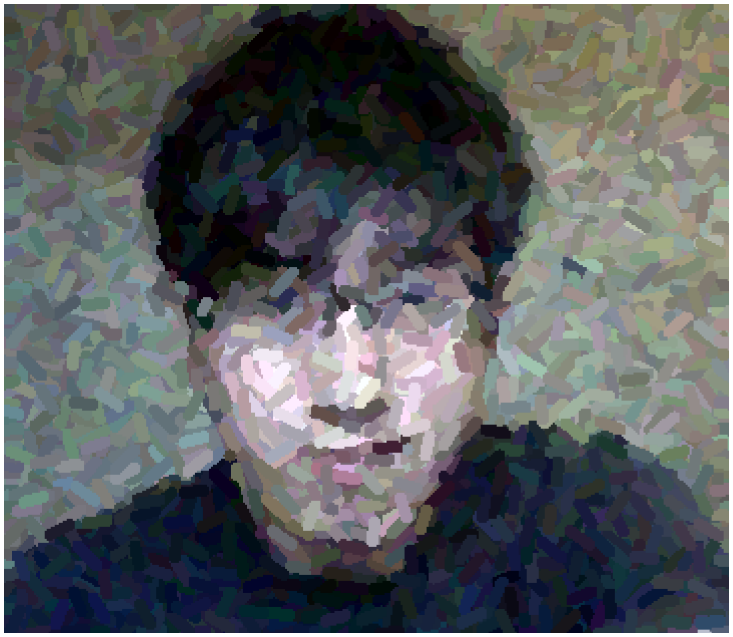

Figure 16: Part 6 Output of Personal.jpg