Computer Vision Course Project
Pencil Drawing Video

GE TIANYANG, WANG YUE
Shanghaitech University
December 31, 2017

1 Introduction

In this project, we propose to construct a short video in pencil drawing style of our university. The main approach we use is combining sketch and tone for pencil drawing introduced in the paper [1]. Artists use strokes to vary wigglines and thickness, while they also use dense strokes, such as hatching, to emphasize shadow, shading, and dark objects. Thus, if we combine two parts together, we would get a whole pencil drawing. Then we recover the colors of images by taking the last two bins of bcrycr form of the image to make the whole image complete. Our goal is to transform the whole video into a new pencil drawing style.

2 Implement Details

The paper [1] introduces two main steps of pencil drawing generation, including line drawing with strokes and tone mapping with pencil texture. An overview of pencil drawing generation process is shown in fig 1.

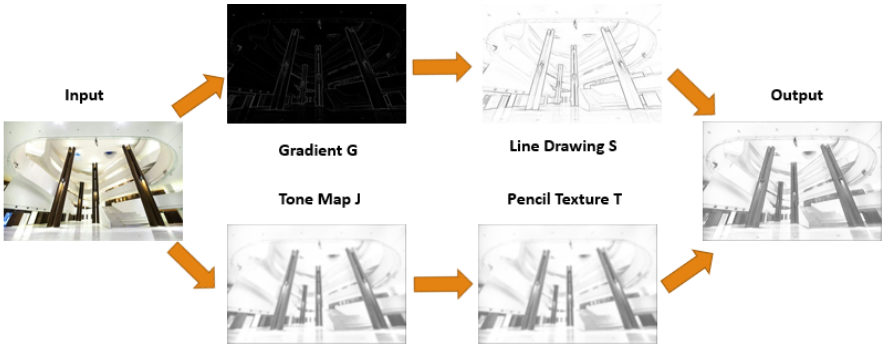

Figure 1: Outline

2.1 Generating Stroke

The most import part for sketch is drawing pencil lines, and lines are close to edges. Thus, it is easy to think of a primitive approach for sketch generation by edge detection. Similar to Canny edge detector, usual steps for drawing sketch make use of gradient map. The gradient is computed by
$G = ((\partial_x I)^2 + (\partial_y I)^2)^{1/2}$
Where I denotes the grayscale of image. Grayscale here is obtained in a different step by converting rgb space to yuv space. Y of yuv space which is related to illumination is calculated by
$lumIm = (0.299 * img[:,:,2] + 0.587 * img[:,:,1] + 0.114 * img[:,:,0])/255$
The paper [1] suggests using forward difference to compute gradient of x and y direction. In our implementation, we tried two approaches.

```
imx = abs(im[:,0:w-1] - im[:,1:w])
imy = abs(im[0:h-1,:] - im[1:h,:])
c = np.zeros((h,1))
imx = np.column_stack((imx, c)) # add a column
r = np.zeros((1,w))
imy = np.row_stack((imy, r)) # add a row
imedge = np.sqrt(imx*imx + imy*imy) # compute gradient

sobelx = cv2.Sobel(im, cv2.CV_64F, 1, 0, ksize=3)
sobely = cv2.Sobel(im, cv2.CV_64F, 0, 1, ksize=3)
imedge = np.sqrt(sobelx*sobelx + sobely*sobely)
```

In practice, sobel operator provided by library cv2 performs better for detecting edges.

Comparing to their new approach, the paper [1] points out the disadvantages of gradient based line drawing methods, which are sensitive to noise and do not contain continuous edges ready for stroke. It presents a robust ap-

proach to take estimation of line direction for each pixel by local information.

Firstly, do convolution between the gradient map and kernel for eight directions, 45 degree each, and get responses of eight directions. The kernel has 1 along the specified direction and 0 otherwise.
$G_i = Li * G, i \in \{1...8\}$
The kernel size suggested by the paper [1] is 1/30 of the image size (height or width). However, in practice, too many additional lines will occur when using a large kernel. Thus, a smaller one with size between 3 to 10 is chosen according to the size of image.

```
kerref = np.zeros((ks*2+1,ks*2+1))
kerref[ks,:] = 1
ker = np.zeros((ks*2+1,ks*2+1,8))

response = np.zeros((h,w,dirnum))
for n in range(dirnum):
    ker[:,:,n] = imutils.rotate(kerref, n*180/dirnum)
    response[:,:,n] = cv2.filter2D(im,-1,ker[:,:,n]) # same
                                        size output
```

Then, each pixel can have its classification decided by its adjacent neighbours, while using gradient map can just classify pixels by themselves. The classfication is defined by

$$C_i(p) = \begin{cases} G(p) & \text{if } argmax_i\{G_i(p)\} = i \\ 0 & \text{otherwise.} \end{cases} \quad (1)$$

Classification map remains the maximum response among all directions. Therefore, relationship between C and G is $\sum_{i=1}^{8} C_i = G$. The paper [**?**] descirbes this classification strategy as very robustness because support of neighboring gradients ensures that noise makes no difference to gradient of the current pixel.

```
index = response.argmax(axis=2) # index of maximum response
C = np.zeros((h,w,dirnum))
for n in range(dirnum):
    C[:,:,n] = imedge*(index==n) # maximum response
```

The final step is generating lines at each pixel by convolution between C and kernel L.

$$S' = \sum_{i=1}^{8} (L_i \otimes C_i)$$

The final stroke map S is obtained by inverting the result and mapping to [0,1].

```
sp = spn.sum(axis=2) # sum 8 directions
s = 1-(sp-np.min(sp))/(np.max(sp)-np.min(sp))
# invert and map to [0,1]
```

Figure 2: Line Drawing

2.2 Generating Tone Map

The target tone distribution is: $p(v) = \frac{1}{Z}\sum_{i=1}^{3} w_i p_i(v)$. The three components $p_i(v)$ account for the three tonal layers in pencil drawing, and ÏL's are the weights coarsely corresponding to the number of pixels in each tonal layer.

$$p_1(v) = \begin{cases} \frac{1}{\sigma_b} e^{-\frac{1-v}{\sigma_b}} & \text{if } v < 1 \\ 0 & \text{otherwise} \end{cases}$$

$$p_2(v) = \begin{cases} \frac{1}{u_b - u_a} & \text{if } u_a < v < u_b \\ 0 & \text{otherwise} \end{cases}$$

$$p_3(v) = \frac{1}{\sqrt{2\pi}\sigma_d} e^{-\frac{(v-\mu_d)^2}{2\sigma_d^2}}$$

According to the paper [1], we could get the parameters which are $w_1 = 11, w_2 = 37, w_3 = 52, \sigma_b = 9, \sigma_d = 11, u_a = 105, u_b = 225, \mu_d = 90$ (The

paper did parameter learning and gave a set of ideal parameters). Our target is to transfer the images Tone map. Thus adjust the Tone map as below.

```
Iadjusted = np.zeros((Y.shape[0], Y.shape[1]))
for x in range(Y.shape[0]):
    for y in range((Y.shape[1])):
        histogram_value = ho[0, Y[x, y]]
        index = (abs(histo - histogram_value)).argmin()
        Iadjusted[x, y] = index
Iadjusted /= float(255)
Iadjusted=cv2.blur(Iadjusted,(10,10))
```

Figure 3: Tone Map

2.3 Pencil Texture Rendering

When we want to generate pencil textures for image, simulate the process using multiplication of strokes. $H(x)^{\beta(x)} \approx J(x)$

$$\beta^* = argmin_\beta ||\beta lnH - lnJ||_2^2 + \lambda || \bigtriangledown \beta||_2^2$$
$$T = H^{\beta *}$$

Python implement as below.

```
A = 0.2*(Dx.dot(Dx.T)+Dy.dot(Dy.T))+np.dot(logP.T, logP)
b=logP.T.dot(logJ)
beta = bicg(A,b,tol=1)
beta=beta[0].reshape(h,w)
T=bP**beta
```

Figure 4: Pencil Texture

2.4 Overall Framework

After the implementation of Stroke S and Pencil Texture T, we multiply two matrix element by element, which is expressed as $R = S * T$

Figure 5: Output

2.5 Color Pencil Drawing

To recover color, we use U and V space from the original image. Y space only store information of illumination, so change on Y has no influence on color which is presented by U and V.
In practice, we use YCbCr space, thus, we can transform back to RGB (actually image is stored in BGR order in opencv) using a function given by cv2. YCbCr is slightly different from YUV space and used more widely due to its faster calculation. The transform equation is given by
Python implement for transform in the beginning and transform back to color pencil drawing is shown below.

```
  Y=0.098*img[:,:,0]+0.504*img[:,:,1]+0.257*img[:,:,2]+16 # bgr
  Cb=-0.148*img[:,:,2]-0.291*img[:,:,1]+0.439*img[:,:,0]+128
  Cr=0.439*img[:,:,2]-0.368*img[:,:,1]-0.071*img[:,:,0]+128

# Recover color
    newim[:,:,0] = newY
  newim[:,:,1] = Cb
  newim[:,:,2] = Cr
  colorimg = cv2.cvtColor(newim, cv2.COLOR_YCR_CB2BGR)
```

2.6 Drawing to Video

Opencv provides useful tools for creating video. To construct a full video, we transform the origin video to pencil drawing frame by frame.

```
cap = cv2.VideoCapture('campus.mp4')
out = cv2.VideoWriter('out.avi', -1, 20.0, (568,320))
while (cap.isOpened()):
  ret, frame = cap.read()
  im = imgdemo(frame)
  out.write(im)
  cv2.imshow('frame', im)
  if cv2.waitKey(2) & 0xFF == ord('q'):
    break
cap.release()
out.release()
```

Figure 6: Color Pencil Drawing

3 Results

Comparing our results to that from paper provided on the paper's home page, ours has some missing details especially on human face.

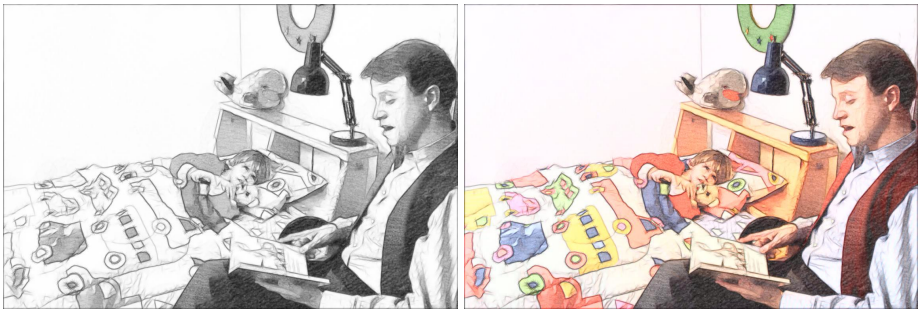

Figure 7: Color Pencil Drawing and Sketch from paper

From our point of view, the paper's authors might do some special modifications in their implement. Although we followed their outline, we failed to meet the same performance as theirs shown in their gallery. Also, we found that some steps written in the paper should be better or it could not present such results.
The video demo is constructed by large amount of images, however, bright light when taking video has bad influence on the final result, which makes the video not as optimal as static photographs.
Some images for our university is presented below.

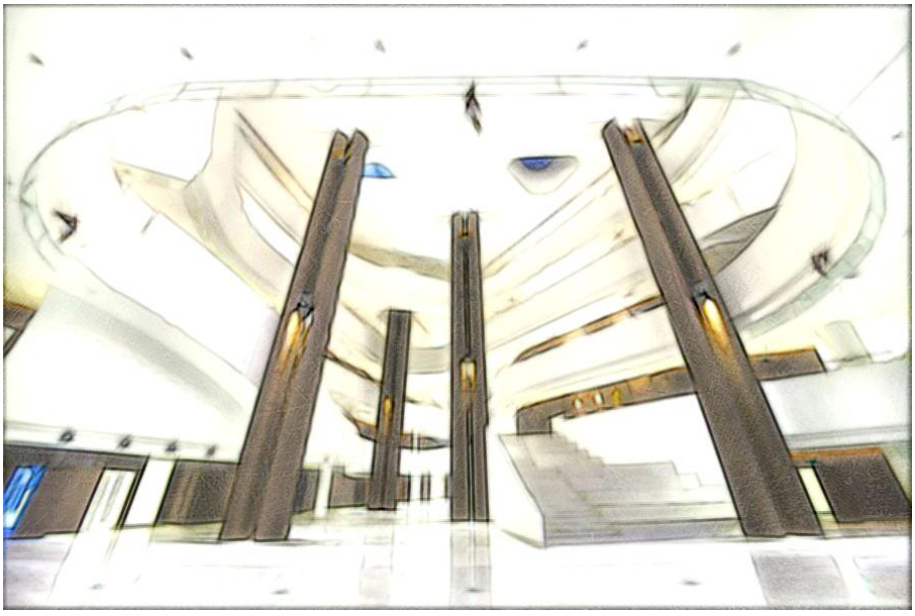

Figure 8: Library

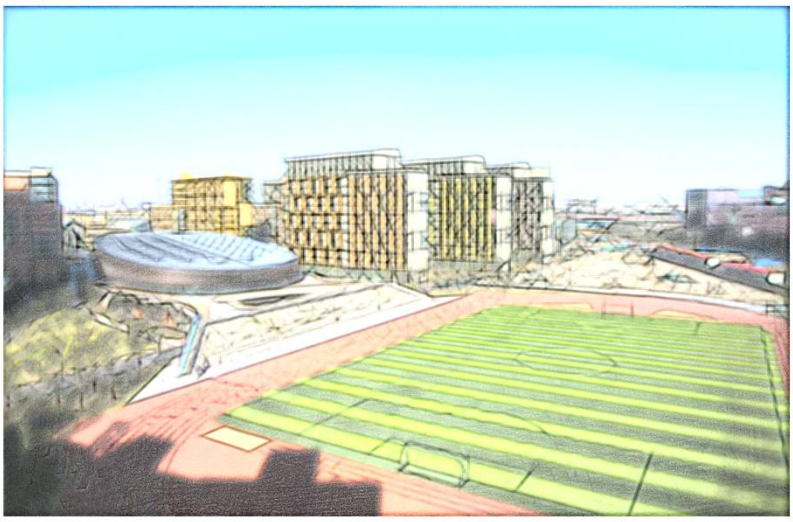

Figure 9: Playground

References

1. LU, C. W., XU, L.,AND JIA, J. Y. 2012. Combing Sketch and Tone for Pencil Drawing Production, International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2012), June, 2012

Home page
http://www.cse.cuhk.edu.hk/leojia/projects/pencilsketch/pencildrawing.htm