Combining Sketch and Tone for Pencil Drawing Rendering

Jiang, Yifeng 661529830

1 Introduction

In this project, I implemented a system to generate pencil-drawing style images from photographs. The system has two main stages. Given a photo as input, it first generates a stroke layer to represent the shapes on the image, imitating painters sketching the contours. Then it produces tonal textures, imitating the hatching process when painters depict brightness and shades with pencils. Finally the two layers are combined to synthesize a nonphotorealistic pencil drawing.

This system helps people better understand how painters produce pencil drawings, which is one of the most fundamental pictorial languages to abstract human perception of natural scenes [Lu et al. 2012]. Moreover, rendering real scene in a non-realistic, artistic way in many cases is desirable in game and film industry since artists have learned that by making images look less photorealistic they enable audiences to feel more immersed in a story [McCloud 1993]. Computers thus help saving a lot of human labor as traditionally producing a non-realistic video segment would require artists drawing many pictures by hand. Finally, this kind of algorithms allow ordinary people to "become artists". Being able to transform common photos taken by themselves into unique arts would add much fun to people's life, making this system commercially valuable.

2 Related Work

This project actually tried to re-implement the algorithms from the paper Combining Sketch and Tone for Pencil Drawing Production [Lu et al. 2012]. The paper presents the two-stage system mentioned above (illustrated by Figure 1). For the stroke-drawing stage, edge detector is used as a start point for object contour drawing. Convolution with line segments is then adopted to both classify and render the edges. For the hatching stage, it uses histogram matching (equalization) to adjust the tone of the input image since pencil drawings by artists often possess a much different histogram from realistic photographs. Laplacian operator and matrix are also involved to make the texture rendering process locally smooth.
Figure 1. Related work: illustration of the two-stage system.

3 Data Collection

Most of the testing photos are obtained from the database created by the authors of [Lu et al. 2012]: http://www.cse.cuhk.edu.hk/leojia/projects/pencilsketch/pencil_sketch_images/5--12.htm. Some more photos from the Internet or taken by myself are added. The sizes of these photos vary a lot but are all resized by the program at the beginning in order not to exceed 1024*768, so that a decent computational cost is kept. All the photos are RGB images, though gray-scale input also works for the program.

There are about 19 photos in total. 5 feature mainly artificial objects, like architectures, vehicles, furniture or everyday items. 8 from nature scenes, such as flowers, trees or ocean. 6 feature persons, with or nearly without background, containing one or more than one people. 2 of the 6 are statues of humans. The test cases covers a larger portion of the topics usually seen in real pencil drawings.

4 Technical Approach

The algorithm is implemented in Matlab.

4.1 Edge detection with Sobel

As a starting point to generate stroke map, I first obtained the image’s edge magnitude map: the 2-norm of the image’s responses to x-direction and y-direction Sobel filters.

```matlab
a = filter2(x-Sobel,im)/8;
b = filter2(y-Sobel,im)/8;
magnitude = sqrt(a.^2+b.^2);
```

4.2 Edge classification with convolution

4.2.1 Observation: “Close-up’s” on pencil drawings

An important observation is that painters, as human beings, often produce close-ups when drawing strokes. The crosses at the junction of two lines, as shown in Figure 2, are caused by rapid hand movement.
A naive approach to model this, is using convolution to extend edge pixels with the same direction. The problem is that even the edge responses appear on a same straight line, some of them may have quite different orientations. (Figure 3) Directly expanding along their “natural” orientations will sometimes add noise. More importantly, discontinuous pixels with same directions will reduce the strength (darkness) of the extended lines.

4.2.2 A better approach using convolution

Thus what we want is to make all the edge pixels on the horizontal line above classified as horizontally oriented. To achieve this, we note that these “horizontal” pixels, if convolved with a horizontal line segment, will have larger response than convolved with line segments in
any other direction. So I first created dirNum (10 in my program) line segments in 10 different orientations, each differing with 18 degree.

\[
\text{ker}(1) = \text{horizontal line segment with length } L; \\
\text{ker}(i+1) = \text{imrotate( ker}(1), \ i*180 \ / \ \text{dirNum} \); \ (i=1:\text{dirNum}-1)
\]

We shall see in later sections that this approach also has its disadvantage of producing additional noise lines. So an appropriate selection of the segment length is important. I set the length L to 1/50 of the height or width of the image in my program.

After convolution, I got 10 response maps

\[
\text{response}(i) = \text{mag} \ast \text{ker}(i),
\]

where \( \ast \) denotes the 2D convolution sum. I then duplicated 10 identical magnitude maps. For every pixel in magnitude map \( i \), its value was kept if its corresponding response value \( i \) was the largest among the 10. Otherwise, that magnitude was set to 0.

After this classification, we see that the original magnitude map was split into 10 mutually exclusive parts, each part \( i \) contained the edge pixels approximately oriented in \( \text{ker}(i) \)'s direction. This is a more robust solution to the problem discussed in 4.2.1.

### 4.3 Line shaping with convolution

Now we are able to produce the “close-ups” by extending edges in its own direction. It is achieved by convolution again. For each magnitude map \( i \), I convolved it with \( \text{ker}(i) \), extending the edges classified in \( \text{ker}(i) \)’s direction. The final stroke map \( S \), is generated by adding the 10 responses together, inverting its value, and scaling to \([0 \ 1]\) so that the “strongest” edge pixel will have value 0, having the darkest appearance, while it shows white where is no stroke.

Note that only by principle only long straight lines will be significantly extended and that a line tends to be darker in its middle part due to convolution. They are both desirable in pencil drawing.

### 4.4 Target histogram generation and matching

Now we can begin with the second stage of generating the hatching (tone) map. We first observe that the perceptual tone (histogram) of a pencil drawing is quite different from photos. The fact that artists draw nothing on most part of the white paper and that pencil hatching cannot be totally black makes a pencil drawing perceptually much brighter than photos.

Based on that, one key finding proposed by [Lu et al. 2012], is that “Unlike the highly variable tones in natural images, a sketch tone histogram usually follows certain patterns”. If we focus on observing works from one style of pencil drawing, we may argue that their histograms can be described with a single model (Figure 4):
The model parses the drawing into three layers with different brightness. In p3, the dark layer, artists hatches “hard” to depict the shading area or area with much information. In p1, as mentioned above, artists nearly draw nothing. In the middle layer p2, various pressure is adopted to make the “transition band”, significantly enriching the perceptual information in the drawing.

We may further argue that even the weights of the three layers follow a certain pattern since for example the dark layer p3 must occupy less area than p1. Various real drawings indeed show this. (Figure 5)
Thus we make the simplification that all our generated tone maps ideally should have a same target histogram:

\[ p = p_1 \cdot w_1 + p_2 \cdot w_2 + p_3 \cdot w_3, \]

where \( p, p_1, p_2, p_3 \) are all 256 bins histogram vector and \( w_1, w_2, w_3 \) are the weights. Then we just use simple histogram matching (Matlab histeq() function) to match a photo’s histogram to our target. Note that Matlab will automatically do the normalization job for us. The matched tone map \( J \) is

\[ J = \text{histeq}(im, p). \]

For the parameters and the three weights, I directly used the result from [Lu et al. 2012], which the authors obtained by collecting more pencil drawing similar to Figure 5 and then making estimations with parameter learning:

\[
\begin{array}{cccccccc}
\omega_3 & \omega_2 & \omega_1 & \sigma_b & \sigma_u & \sigma_b & \sigma_u & \sigma_d \\
11 & 37 & 52 & 9 & 105 & 225 & 90 & 11
\end{array}
\]

(Note: An apparent problem with this is not all photo histograms can fit the target well. Consequentially, some tone maps are so dark that they affect the visibility of the strokes. There will be some discussion on this later.)

4.5 Pencil texture rendering

Adjusting the histograms of input photos is not enough for generating the tone map. A pencil drawing possess non-realistic pencil textures. To imitate this, I start from the following texture picture found on Internet. (Figure 6)
A trivial observation is if our input photo has a huge flat gray area, instead of setting every pixel to have the same gray-scale value, we should “copy and paste” the pixel values from Figure 6 to produce the “non-uniform” pencil texture. Further, if on some part of the drawing, we “draw” the texture in Figure 6 twice (mathematically, the square of the area’s gray level if scaled to [0 1]), that area will appear darker while the texture is preserved.

Hence the strategy is to set the texture template’s gray-level map to be matrix $H(x, y)$. Since we already have a tone map $J(x, y)$, what we want is to obtain a map $\beta(x, y)$ representing the times of “repeatedly hatching” on every point, such that $H(x, y)^{\beta(x,y)} \approx J(x, y)$, or in linear form, $\ln H(x, y) * \beta(x, y) \approx \ln J(x, y)$. Meanwhile, we want the texture to be locally smooth, so the “repeating times” matrix $\beta$ should be smooth. Balancing both requirements, it turns out we need to solve the following equation ([Lu et al. 2012]):

$$\beta^* = \arg\min_\beta \| \beta \ln H - \ln J \|^2_2 + \lambda \| \nabla \beta \|^2_2$$

The argmin values are reached when the partial derivatives of the right hand side with respect to every pixel element of $\beta(x, y)$ are equal to 0. Thus the equation transforms to a large linear system with all elements from $\beta^*$ as unknowns.

Note that the coefficient matrix of this system $A$, is actually a laplacian-like matrix with all elements on main diagonal modified. Thus Matlab’s built-in sparse matrix representation (with functions spdiags() and ““”) enables this system to be solved in decent time.

For the parameter $\lambda$, since large pictures tend to display in a smaller scale on devices, it should better have a larger $\lambda$ for users to perceive the pencil texture. I set $\lambda$ to $1/700$ of the image height or width in my program.

The final tone map $T$, is calculated from $T = H^{\beta^*}$, distinguishing it from $J$.

4.6 Combining stroke and tone in final drawing

In principle, if $S$ and $T$ are all scaled to [0 1] with 0 representing the darkest pixel, the final result $R$ should be $S^*T$. However in my experiment, as mentioned above $J$ has often been too dark due to practical histogram matching. So in my program I shrink and shift the $T$’s histogram range to make it brighter since the loss of gray levels in pencil drawings is quite perceptually acceptable. The actual formula I used is $R = S .* (T + 0.5)/1.5$. 
4.7 (Option) Color pencil drawing

To produce color pencil drawing, we just use rgb2ntsc() instead of rgb2gray() when reading the input image to keep the I and Q information in the YIQ color space. Note that the Y layer here is identical to the gray-scale image we are using for previous algorithms. After we have a result map R, we just let it to be Y’ and use Y’IQ and Matlab ntsc2rgb() to get the color pencil drawing. That is, the I and Q layer is kept unchanged during the whole process.

5 Intermediate Results

This section will be a brief review of the previous section with illustrations from real image examples. Emphasis will be on the potential issues of the algorithm in some cases.

5.1 Stage one: stroke map

We use the following image (Figure 7) as raw input, for which it turns out that the algorithm does not work very well.

![Figure 7. Input image for stroke map generation.](image)

As the first step, the edge map from Sobel is as follows (Figure 8), for this part we will focus on the lower-left area of the image where the texture is quite complex:
Figure 8. Edge detection result from Sobel.

Next we can show what pixels are classified by the algorithm as horizontal edges. (Figure 9)
We can see that for the relatively long horizontal edges, for example the stones in the water, the program indeed selects them out. But for all the “horizontal leaves”, the program totally misses them. Generally, if the texture is much smaller than the length of convolution kernel, the program just exhibits random behavior.

A direct consequence of this, it that since we are to extend the edges in every direction in the next step of line shaping, the “leaves area” will have a lot of non-sense noise lines. Figure 10 shows the final stroke map $S$ from this image, with the lower-left area in detail.
However, the algorithm does have its advantage in inhibiting weak responses from edge detectors which are not really edges for a drawing. As an example, for the coconut leaves in the upper-left region in Figure 10, if we instead use the naïve method of classifying edges by directly quantizing their directions, the result will be Figure 11. (Note that we keep the line shaping step unchanged.)

This advantage is important as we are to multiply this stroke map with tone map. For one thing, we don’t want too much gray shade inside the leaves to reduce the region’s luminance. For another, we want the real edges of the leaves to be as distinguishable as possible since multiplying the tone map generally make the stroke map less visible (as to be discussed in the following subsection).

Another fact about this algorithm, is that it will link edges that are not originally connected. Use the following part of an input image as an example (Figure 12):
Figure 12. Part of an input image and its edge map from Sobel. Note the small windows on the building behind.

The stroke map generated from this actually links all the windows to form a long straight line, as shown in Figure 13:

![Stroke map generated for the buildings.](image1)

For now this seems quite visually distracting because we even cannot tell the true contour line of the building behind. However, it turns out that things generally get better when we add the tone map.

5.2 Stage two: tone map

As one of the observations from previous subsection, though the algorithm for line
drawing often inhibits non-contour edge pixels and strengthens the real contours, it sometimes produces distracting noise lines especially at regions where textures are intensive. But one underlying idea of this two-stage algorithm is that the tone map, in many cases, would “cover” (inhibit) the internal noise lines while preserving the close-ups. The following pictures illustrate this: (Figure 14, 15)

Figure 14. The stroke map. Some noise lines can be seen on the women’s and man’s face, as well as on the trees on the right.

Figure 15. The final result after multiplying the tone map. Much of noise has been reduced while the close-ups (like on the woman’s shoulder) are generally preserved.

As mentioned earlier, the tone after histogram matching is sometimes still not as bright as expected, while we do need the tone to be bright enough for the stroke lines to stand out. A typical example is Figure 16.
Figure 16. The histogram to be matched (left) and the actual histogram of the output image (right). It is typical that the middle layer, instead of being depressed, often has many more pixels than expected.

Since I found no quick solution to this, I would argue that shrink and shift the tone map’s histogram somewhat makes the output more in “pencil-style” and less photo-realistic. (See Figure 17, 18.)

Figure 17. The final result by directly multiplying S and T map.
Figure 18. The result by multiplying S with \((T+0.5)/1.5\).

One final note is that the pencil texture rendering process is quite satisfying, as can be seen from the final results below.

6 Final results

As expected, the algorithm works better for artificial objects than humans (or statues). This is because the system is more capable of depicting straight, long lines. A minor reason is that compared with objects, humans are much more sensitive and picky to portraits, especially to human faces.

For nature scenes, generally the algorithm works fine when the input photo is not too much detailed. Meanwhile, most pencil drawings won’t include too many irregular details. Even if an artist needs to draw a tree, he/she won’t bother drawing every leaves on it. Instead, often abstraction of textures will be adopted, which is too hard for algorithms. So my personal feeling is that if one think a certain image should look good after being transformed into a pencil drawing, meaning it may have decent brightness and not too many irregular details, the actual result is often not bad.

Finally, this algorithm may not be suitable for being used in an App as people do care how they look in the pictures produced. In fact, the commercial Apps often directly adopt edge detectors for contour drawing.
6.1 Results for artificial objects

Figure 19. Input: Cars and slogans.

Figure 20. Output: Cars and slogans.
Figure 21. Input: Furniture.

Figure 22. Output: Furniture. Lines are somewhat washed out by the tone map.
Figure 23. Input: Buildings and boat.

Figure 24. Output: Buildings and boat.
Figure 25. Input: Everyday items.

Figure 26. Output: Everyday items. Obama’s face is slightly distorted. The tissue box looks good though.
Figure 27. Input: Building.

Figure 28. Output: Building.
6.2 Results for natural scenes

Figure 29. Input: Sunflowers.

Figure 30. Output: Sunflowers.
Figure 31. Input: Ocean rocks and plants.

Figure 32. Output: Ocean rocks and plants. The color drawing result is slightly better, though both of them have much distracting noise, as discussed earlier.
Figure 33. Input: Trees and flags.

Figure 34. Output: Trees and flags. The trees are distracting due to noise lines, while the flags look better.
6.3 Results for humans

Figure 35. Input: Boy.

Figure 36. Output (Color): Boy.
Figure 37. Output (grayscale): Boy. There are many more irregular noise lines here.

Figure 38. Input: Girl.
Figure 39. Output: Girl. The problem here is the stroke lines are nearly invisible because the girl is in shade.

Figure 40. Input: Man and Woman.
Figure 41. Output: Man and women.

Figure 42: Input: G20 leaders.
7 Discussion and Further Work

The authors of [Lu et al. 2012] get much better results for humans than me. I am not sure whether there are mistakes with my implementation.

I am thinking if what the authors really mean by histogram matching is to first threshold the image into three layers, match each of them to the target layer, then re-compose the three together. This might help because it is making more constrains. But I have no idea how to histogram match a thresholded image (with a big 0 value bin to be neglected).

Finally, things similar to Gaussian pyramid may be useful since for more detailed areas shorter kernel length might be better. We may even directly adopt edge maps in these delicate areas.

8 References
