Sketch Generation with Drawing Process Guided by Vector Flow and Grayscale

Zhengyan Tong,[1] **Xuanhong Chen,**[1,2] **Bingbing Ni,**[1,2*] **Xiaohang Wang** [1]
[1] Shanghai Jiao Tong University
[2] Huawei Hisilicon
{418004, chen19910528, nibingbing, xygz2014010003}@sjtu.edu.cn

Abstract

We propose a novel image-to-pencil translation method that could not only generate high-quality pencil sketches but also offer the drawing process. Existing pencil sketch algorithms are based on texture rendering rather than the direct imitation of strokes, making them unable to show the drawing process but only a final result. To address this challenge, we first establish a pencil stroke imitation mechanism. Next, we develop a framework with three branches to guide stroke drawing: the first branch guides the direction of the strokes, the second branch determines the shade of the strokes, and the third branch enhances the details further. Under this framework's guidance, we can produce a pencil sketch by drawing one stroke every time. Our method is fully interpretable. Comparison with existing pencil drawing algorithms shows that our method is superior to others in terms of texture quality, style, and user evaluation. Our code and supplementary material are now available at: https://github.com/TZYSJTU/Sketch-Generation-with-Drawing-Process-Guided-by-Vector-Flow-and-Grayscale

1 Introduction

The pencil sketch is one kind of drawing with a highly realistic style. It is not only a popular form of art creation but also the essential basic of other art forms such as oil painting. To draw a pencil sketch, artists should have accurate contour configuration ability and superb shading drawing skills, requiring long-term professional training. Therefore, there has always been a strong demand for the pencil sketch rendering algorithm.

The existing pencil drawing algorithms are mainly implemented by Non-Photorealistic Rendering (NPR) (Rosin and Collomosse 2012) and can be further divided into 3D model-based sketching and 2D image-based sketching (Lu, Xu, and Jia 2012). 3D models can provide the complete geometric information of the objects, and the lighting condition in the 3D scene is fully controllable. Thus 3D model-based sketching can accurately grasp the spatial structure and render the shading texture according to the light condition. However, in most application scenarios, we can not get 3D models but only 2D natural images, so there is a greater demand for image-based sketch rendering algorithms. For 2D natural images, the geometric information is often incomplete, and the light components are usually complicated and noisy, making it hard to do sketch rendering. Generally, mathematical rendering algorithms can maintain the structures well, control the rendering effect in a fine-grained manner, and are often interpretable. In recent years, many deep learning methods for image style transfer tasks have been developed. Results of these methods usually have a stronger style than those of procedural rendering algorithms. However, due to the complexity of pencil drawing, neural methods do not perform well at capturing pencil sketch texture (Li et al. 2019). Deep learning style translation usually suffers structure distortion and artifacts, which is a serious defect for the pencil drawing task that requires reserving structures and producing high-quality textures. Besides, neural networks' parameter control mechanisms are usually high-level and are difficult to explain.

Whether it is a procedural algorithm or a deep learning neural style transferer, existing algorithms can only get the final result without offering the drawing process. Actually, there exists very limited research on auto-drawing with the drawing process. Our algorithm implements image-to-pencil with a drawing process for the first time, which significantly enhances our algorithm's novelty. As shown in Figure 1, given an image (in the rightmost column), our algorithm can produce a pencil sketch by drawing one stroke at a time (Figure 1 only shows the drawing process in stages, the whole process can be found in the supplementary material). Our algorithm could generate high-quality details, and the final result has a strong style. Besides, we can produce pencil drawings with obviously different visual effects by adjusting the strokes' properties. Comparison with existing pencil drawing algorithms shows that our method performs favorably in terms of texture quality, style, and user evaluation.

Our work is inspired by the observation of real pencil drawings, and our algorithm models the real artists' drawing techniques. Thus the interpretability of our method is stronger than others. Since pencil drawing has many different drawing styles and artists use various drawing tools (Dodson 1990), we only simulate the most popular sketching method. That is, drawing strokes with diverse directions, shades, lengths, and widths on a canvas to gradually form a picture. For the strokes' direction, artists usu-

*Corresponding author: Bingbing Ni.

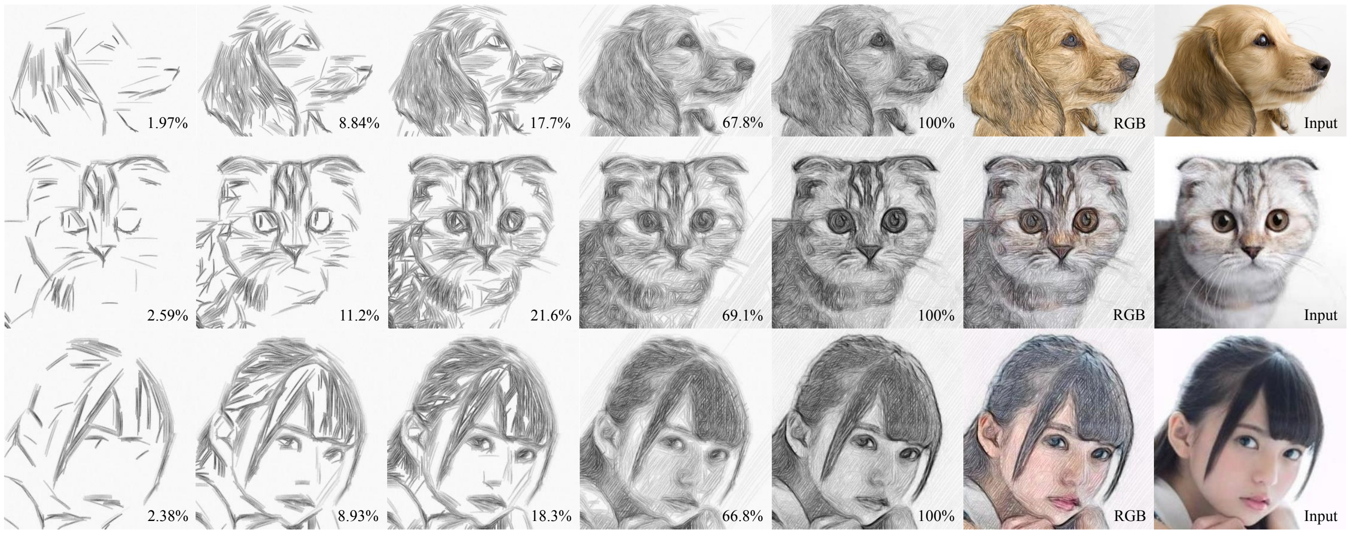


Figure 1: Given a natural image (in the rightmost column) as input, our algorithm can produce a pencil sketch with the process by drawing one stroke every time. For the dog, the cat, and the girl, we draw 10390, 11529, and 16698 strokes. The percentage in the lower right corner of each picture is the proportion of the number of strokes that have been drawn in this picture to the number of strokes in the final result.

ally use the tangent direction of objects' edges/contours to guide the strokes' direction; for the strokes' shade, artists usually adjust their pencil sketches' contrast higher than real light conditions to make their works more visually impactful (Kelen et al. 1974; Hoffman 1989). We divide the pencil sketching task into two steps. For the first step, we develop a parameter-controlled pencil stroke generation mechanism based on the pixel-scale statistical results of some real pencil drawings. For the second step, we develop a framework to guide strokes arranging on the canvas. Finally, we implement the pencil sketch auto-drawing technique.

In this work, our main contribution is that we propose a novel image-to-pencil translation method that could generate high-quality results and offer the drawing process.

2 Related Work

2.1 Non-Photorealistic Rendering

There is a rich research history on the non-photorealistic texture rendering of pencil drawing. 3D models provide all the geometric information and light conditions, thus convenient for pencil drawing rendering. (Lake et al. 2000) presented pencil sketching texture mapping technique and proposed pencil shading rendering. (Lee, Kwon, and Lee 2006) detected contours from 3D models and imitated human contour drawing. For expression of shading, they mapped oriented textures onto objects' surface. (Praun et al. 2001) achieved real-time hatching rendering over arbitrary complex surfaces using 3D models. These 3D model-based methods usually obtain satisfactory results. However, when the 3D structure and light conditions are not available, these methods cannot work.

2D image-based methods mainly include the following typical algorithms. Sousa and Buchanan presented an observational model of blenders and kneaded eraser (Sousa and Buchanan 1999b), and simulated artists and illustrators' graphite pencil rendering techniques in (Sousa and Buchanan 1999a). (Chen et al. 2004) proposed a composite sketching approach for portrait drawing. (Durand et al. 2001) presented an interactive system that allowed users to produce drawings in a variety of styles, including pencil sketching. (Mao, Nagasaka, and Imamiya 2002) detected the input image's local structure orientation and adopted linear integral convolution (LIC) to render sketch texture. (Yamamoto, Mo, and Imamiya 2004) divided the input image into several layers of successive intensity ranges, then did rendering for each layer and finally added them together. (Li and Huang 2003) analyzed the image moment and texture of each region, using the captured feature geometric attributes to implement pencil drawing rendering. Others proposed some improved LIC-based method (Chen et al. 2008; Gao et al. 2010; Kong, Sheng, and Zhang 2018; Chen et al. 2017). Pencil sketch rendering could also be implemented by image analogies (Hertzmann et al. 2001). (Lu, Xu, and Jia 2012) proposed a novel two-stage system combining both line and tone for pencil drawing production and obtained significantly better effect than the above methods.

2.2 Drawing with Process

All previous works on the image-to-pencil task can only generate a final result, without offering the drawing process. Here we review some related Stroke-Based Rendering methods which have a process.

(Fu et al. 2011) proposed an algorithm that used human pre-drawn line drawing as the input to derive a stroke order and animate the sketching automatically, but this method couldn't well recover the input line drawing. (Ha and Eck 2018) presented a RNN-based method trained on a dataset of human-drawn simple images to draw stick figures. Stro-

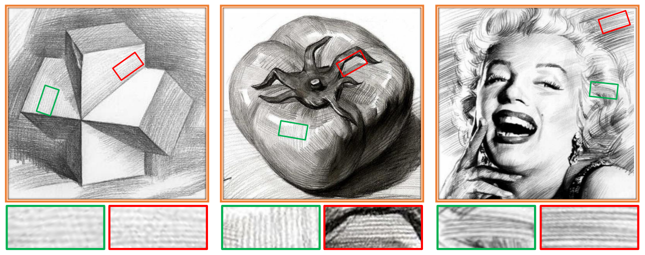


Figure 2: Three real pencil drawings. It can be seen from the zoomed-in areas that the texture of pencil drawings is actually some parallel strokes.

keNet (Zheng, Jiang, and Huang 2019) can generate a sequence of only a few strokes to write Chinese characters. However, the strokes as well as their sequence are very different from that of human writing. (Huang, Heng, and Zhou 2019) adopted model-based Deep Deterministic Policy Gradient (DDPG) algorithm to train a neural agent to learn to do oil painting with process. However, this method cannot be directly applied to pencil drawing because pencil strokes' characteristics and fusion mode are distinctive from the oil painting. The strokes of pencil drawing are lines while the oil painting' strokes are color blocks; the newly drawn strokes cannot cover the old ones in pencil drawing while the oil painting's strokes can. Besides, lines' sparsity makes it hard to train pencil drawing neural agents. Deep reinforcement learning (DRL) requires a massive amount of parameters when training, so the network's input size is very limited. (Huang, Heng, and Zhou 2019)'s oil agent can only handle 128×128 images, unable to generate fine-grained details, while our algorithm has no restriction on the size of the input image and could generate high-quality details.

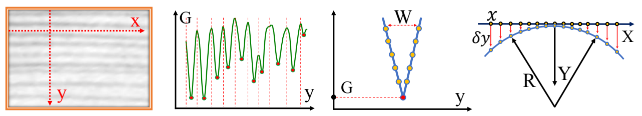
3 Stroke Simulation

Lines are the fundamental elements of pencil sketching. Since pencil drawing strokes are lines, we regard the "line" and the "stroke" as the same concept in this article.

3.1 Observation and Statistic

The analysis of real pencil drawings can be performed globally or locally. The statistics of global features are mainly on the histogram. (Lu, Xu, and Jia 2012) counted and fitted the histogram distribution of several real pencil drawings, then did histogram matching to transfer the tone of input images. The analysis of local features is mainly for texture. (Sousa and Buchanan 1999b) observed the blenders and erasers' absorptive and dispersive properties, and studied their interacting with lead material which deposited over the drawing paper. (Hertzmann and Zorin 2000) analyzed different hatching styles of pencil sketch. (Xie, Zhao, and Xu 2007) studied the graphite's distribution in pencil drawings and made three assumptions about the local distribution characteristics. Generally, local features are more important than global features because local features can better reflect the characteristics of pencil drawings, while the global histogram distribution of various artists' works is often personalized. Our observation and analysis methods are entirely based on local features.

Three real pencil drawings are shown in Figure 2. It can be seen from the zoomed-in area that the texture is composed of many parallel curves. This pattern is also prevalent and evident in the rest region of these drawings. For any group of parallel curves, lines within the group have a high degree of similarity. That is, the distance between any two adjacent lines, the shade, length, and width of each line are very close. Therefore, we can perform statistical analysis on these parallel lines. In order to facilitate statistics, we cut some patches just containing one set of parallel curves from some realistic pencil drawings, and then rotate the patches until the lines' direction is nearly horizontal, as shown in Figure 3(a). We notice that the lines are slightly curved and not strictly parallel, which brings great difficulty to the statistics in the horizontal direction, as shown by the red dotted line marked with x in Figure 3(a). However, the lines' curving does not affect the gray values' distribution in the vertical direction, as shown by the red dotted line marked with y in Figure 3(a). So we do statistics in the vertical direction. For the pixels on the red dotted line marked with y in Figure 3(a), their gray values are shown in Figure 3(b). We draw some red dotted lines at all peak points in (b), it shows that the gray value's distribution curve between any two adjacent peak points is close to the letter V, as shown in Figure 3(c). In fact, each "V" curve in Figure 3(b) corresponds to a stroke in Figure 3(a). The same statistics are performed on all the columns of pixels. Suppose there are n columns, then each stroke corresponds to n "V". For the n "V" corresponding to a particular stroke, the gray value of pixels at the same positions (the yellow points in Figure 3(c)) can be assumed to be independent and identically distributed. Thus, the gray value's mean and variance of each pixel in each "V" can be calculated.



(a) Real pencil (b) Curve (c) "V" model (d) Shifting

Figure 3: (a) is a patch cut from a real pencil drawing; (b) shows the gray value of the pixels on the red dotted line marked with y in (a); (c) is the fitting of the gray value's periodic change in (b). (d) is the illustration of stroke bending.

We use the following method for fitting. We define two variables to determine a "V" curve: width W and the mean value G of the central pixel's gray value. As shown in Figure 3(c), W represents the pixel amount of the "V" curve. The red dot indicates the central pixel of the "V" curve. G is the mean value of the central pixel's gray value. Assume a pixel on this curve is d pixels away from the central red pixel, then the gray value's mean and variance of this pixel can be calculated by the following equations:

$$mean(d) = G + (255 - G) \times 2d/W - 1 \qquad (1)$$

$$variance(d) = (255 - G) \times \cos(\pi d/W - 1) \qquad (2)$$

Now suppose we can straighten the lines in Figure 3(a) in the horizontal direction, and assuming the length of the line is L, then these lines can be represented by a gray value matrix with W rows and L columns. For the pixels in one specific row, their gray value can be considered to be independent and identically Gaussian distributed. Now we define a matrix F with the shape of $(W, 2)$ to record the gray value's distribution of a line. Element $(w, 1)$ and $(w, 2)$ in F indicates the gray value's mean and variance of all the pixels in row w in the line. As long as G and W are specified, the distribution matrix F of the line can be calculated.

3.2 Stroke Generation

We first simulate a straight line and then bend it in the vertical direction to get a more natural effect. To draw a straight line, we need to specify three parameters: central pixel's gray value mean G, line width W, and line length L. Firstly, use G and W to calculate the distribution matrix F of this line. Then for every row, the gray value of each pixel is randomly generated according to F. As shown in Figure 4(a), We have drawn some lines with the width $W = 7$ pixels but with different G. These straight lines look too rigid, so we adjust their shape further. By observing realistic pencil

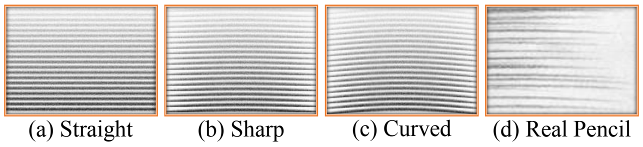


Figure 4: Stroke generation steps

drawing strokes, as shown in Figure 4(d), we found that the head/tail of the strokes are thinner and lighter than the middle part, which is because when the pencil tip just touches the paper's surface or when it is about to leave, the pressure of the pencil tip on the paper's surface is less than when drawing the middle part of the line. Lines are not entirely straight but are slightly curved is because artists draw lines by swinging their wrists, so the movement of the pencil tip on paper is essentially a circular motion with a large radius. We bend the previously generated straight lines twice to achieve these effects. For the first time of bending, as shown in Figure 3(d), the yellow dots on the X-axis indicate the pixels in a particular row of the line. These pixels will be shifted to the blue circle. Use the midpoint of the line as the origin to establish a coordinate system, pixels with different abscissas on the line will have different degrees of deviation in the Y direction. Assuming the blue circle radius is R, pixels with abscissa x will be shifted by $\delta y(x)$ pixels in the Y direction. The radius R and offset $\delta y(x)$ can be calculated as $R = \frac{L^2}{4W}$ and $\delta y(x) = \frac{x^2}{2R}$. Since $\delta y(x)$ is usually a decimal, we perform linear interpolation in the Y direction to achieve this operation. After bending, some pixels will exceed the matrix with W rows and L columns. For these pixels out of the matrix, we directly discard them. For the blank part of the matrix, we fill it with pure white pixels. Now we have the curves as shown in 4(b). The purpose of the first bending operation is to make the head and tail of the lines sharp. The

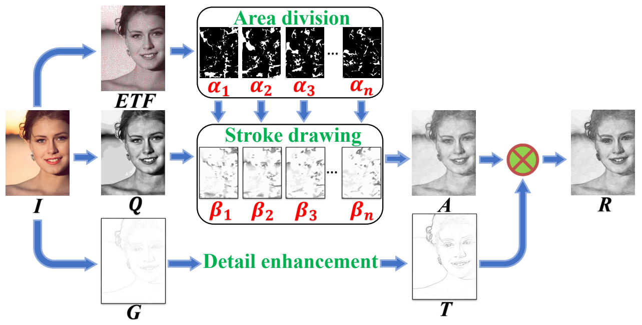

Area division
ETF
α_1 α_2 α_3 α_n
Stroke drawing
β_1 β_2 β_3 β_n
I
Q
A
R
Detail enhancement
G
T

Figure 5: Schematic illustration of our algorithm. I is the input. *ETF* is the visualization of edge tangent flow vector field (Kang, Lee, and Chui 2007). $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ are the area divisions of the input according to the direction of *ETF* vectors. Q is the quantization result of I. $\{\beta_1, \beta_2, \ldots, \beta_n\}$ are the stroke drawing results of each area. A is the aggregation of $\{\beta_1, \beta_2, \ldots, \beta_n\}$. G is the gradient map of I. T is the edge map generated by G. R is the final result obtained by multiplying A and T.

second time of bending is almost the same as the first time, but we preserve those pixels out of the matrix. The purpose of the second bending is to increase the curvature further. Now we have the curves as shown in 4(c). It is worth noting that these curves are essentially still straight lines, only looking more natural than straight lines. We model the strokes as such straight lines rather than arbitrarily shaped curves is because, on the one hand, the real pencil strokes are mostly like this, and on the other hand, the straight lines are convenient for our subsequent work.

4 Guided Stroke Drawing

Now we introduce how to do sketching by drawing one stroke every time on the canvas. To determine a stroke, we need to know the line's width W, length L, central pixel's gray value mean G, starting point's coordinates, and this line's direction. For line width W, now we specify the width of all strokes as a fixed value (we will discuss the influence of line width W in Section 4.5 User Control). We will utilize the local characteristics of the input image to determine the other parameters.

4.1 Grayscale Guidance

To reduce the difficulty of the task, in this section, we do not consider how to determine the direction of the strokes temporarily, but only show how to draw strokes in a fixed direction. Figure 6(a) is the result of drawing strokes only in the horizontal direction. In the following, we will introduce how to draw it.

We first adjust the histogram distribution of the input gray image to improve its hue. We use contrast limited adaptive histogram equalization (CLAHE) (Zuiderveld 1994) to enhance the contrast of the input. Next, we uniformly quantify the image into several gray levels. We denote their gray values as $\{G_1, G_2, ..., G_n\}$. These values will be used as the

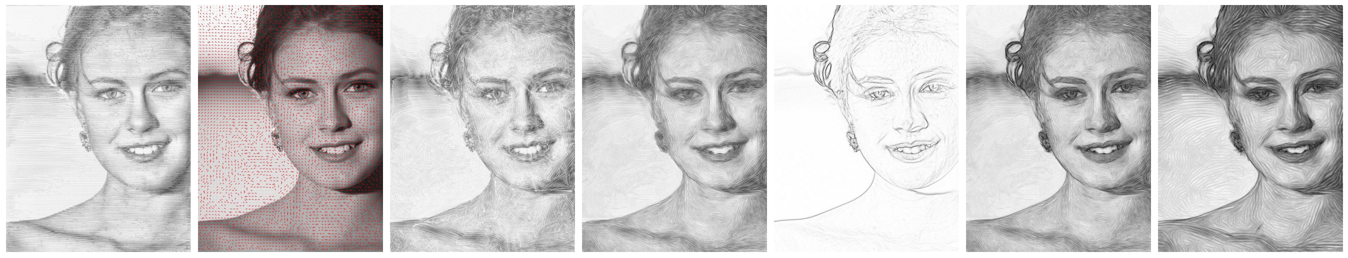


Figure 6: Some images used for algorithm introduction. (a) is the result of drawing lines only in the horizontal direction; (b) is the visualization of edge tangent flow vector field (Kang, Lee, and Chui 2007); (c) and (d) are the results of drawing lines without/with extension respectively; (e) is the edge map obtained by (Lu, Xu, and Jia 2012); (f) is the drawing result of 5 pixel wide lines while lines in (g) are 9 pixel wide.

strokes' central pixel gray value mean *G*. As shown in Figure 5, *I* is the input and *Q* is the result after quantization. Then we use *Q* to search and determine the strokes' parameters by scanning in the horizontal direction.

Take the first row of the pixels in *Q* as an example. Search out all the intervals in the first row where the pixels' gray value is less than or equal to G_1. Use the starting point and length of these intervals as the starting point and length of the strokes to be drawn; use G_1 as the strokes' central pixel gray value mean *G*. Then we can draw several strokes in the horizontal direction (width *W* is specified in advance). Here we define a new random variable $D \sim N(W, 1)$, which is used to generate the lines' pixel distance in the vertical direction. We have searched and drawn strokes in the first row. Every next time we move down *D* rows in the vertical direction and repeat the same operation as the first row until reaching the bottom of the quantization image *Q*. In this way, we could draw all the strokes with central pixel gray value mean G_1. Then we draw strokes for $\{G_2, ..., G_n\}$ in the same way. In this process, different strokes' coverage regions will overlap. The gray value of the pixels in the overlapped region is determined by the minimum (darkest). Now we can get the drawing result shown in Figure 6(a). Although we only introduce the method of drawing strokes in the horizontal direction, different directions are equivalent: we can rotate the input image by an angle clockwise before drawing strokes in the horizontal direction. After the drawing is done, we rotate it back counterclockwise. In this way, we can draw strokes in every direction.

4.2 Direction Guidance

We have introduced how to draw strokes in a fixed direction. Now suppose we can divide the picture into several areas. Only strokes in the same area have the same direction. Then we can use the method in Section 4.1 Grayscale Guidance to draw strokes for every area according to the area division.

By observing real pencil drawings, it is easy to find the direction of strokes is usually along the edges' tangent (Kelen et al. 1974). Therefore, we hope to use the edges of objects to guide the direction of the strokes nearby. However, it is difficult to predict the structure of an object from a 2D image. Actually, we do not need accurate predictions but only an estimation. We could use the input image's gradient information to do this estimation because gradient and edges are often closely related. Under the natural light condition, the change of light's intensity at objects' edges is often more apparent than in flat areas. Therefore, the gradient vector field could offer suggestions for determining the direction of the strokes. We use the edge tangent flow (*ETF*) proposed by (Kang, Lee, and Chui 2007) to estimate the strokes' direction for each pixel. The constructing of *ETF* is as follows: First, calculate the modulus and direction of the gradient vector for each pixel. Then rotate the directions of these vectors counterclockwise by 90 degrees. Adjust the vectors' direction iteratively so that the direction of the vectors with small modulus tends to the direction of the vectors with larger modulus nearby. Finally, the direction of all the vectors will be roughly parallel to the tangent of the edges. The visualization of the edge tangent flow vector field is shown in Figure 6(b). The small red arrows point to the direction of the *ETF* vectors.

Now we could divide the input image into several areas according to the *ETF*. We uniformly quantify the direction $0 \sim 2\pi$ into *n* values. Vectors with a phase difference of π are regarded as the same direction. After quantifying the direction, pixels with the same direction are divided into one area. As shown in the "Area division" box in Figure 5, $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ indicate the area divisions (pixels belong to a certain area are white, while the others are black). After drawing strokes with different directions for each area we can get *n* results, indicated by $\{\beta_1, \beta_2, \ldots, \beta_n\}$ in the "Stroke drawing" box in Figure 5.

4.3 Area Merging and Detail Enhancement

Now we aggregate $\{\beta_1, \beta_2, \ldots, \beta_n\}$ into one picture, as shown in Figure 6(c). There are obvious defects at the boundaries of different areas. Besides, the area division will cause a large number of very short strokes, which look like noise points. These two problems are solved by extending the head and tail of all strokes by *2W* pixels (*W* is the strokes' width), as shown in Figure 6(d). *A* in Figure 5 indicates the aggregation result of strokes in different directions, which could be obtained as this equation:

$$A = minimum(\beta_1, \beta_2, \ldots, \beta_n) \qquad (3)$$

Now the strokes in different directions merge well and appear more continuous. However, extending the strokes also causes loss of detail clarity. For example, the lady's teeth and eyes in Figure 6(d) are very unclear. Therefore, we need to enhance the details of the sketch. As shown in Figure 5, *G* is the gradient map of the input image. *T* is the edge map obtained by *G*. There are countless algorithms for generating the edge map from the gradient map. Here we adopt the linear convolution method of (Lu, Xu, and Jia 2012) because the edge map obtained by his method looks more like the result of a pencil drawing than other algorithms, as shown in Figure 6(e).

Finally, we multiply the edge map *T* and the drawing result *A* to get the final output *R*, expressed as $R = A \cdot T$. The final output *R* is shown in Figure 6(f).

4.4 Process Reconstruction

The strokes' search method has been introduced: we first draw strokes in different directions for each area and then integrate them into the final output, which seems odd. However, due to all the strokes' parameters being determined and recordable when searching for, we can rearrange the strokes' drawing sequence in a more realistic and meaningful order. We call this "Process Reconstruction". When artists draw sketches, they usually draw the outlines first (which are often long or dark lines), and then the details (these lines are often short and not very dark). To imitate this, we use the index *S* to measure whether each line is more likely to be an outline or a detail:

$$S = (255 - G) \times \sum\nolimits_{i \in D} T_i \tag{4}$$

where G is the stroke's central pixel gray value mean, D is the set of pixels where the stroke covers, T_i indicates the absolute value of gradient at pixel i. The larger the S, the greater the possibility that the stroke to be an outline. So we reconstruct the strokes' drawing process according to S in descending order. Three examples are shown in Figure 1. It can be seen that when we only draw about 20% of the total strokes, we can almost present the drawn objects. The whole drawing process (video demo) and more interesting results can be found in the supplementary material.

4.5 User Control

Fineness Our pencil drawing's fineness is controlled by the strokes' width and the number of quantization order. Our method of searching strokes is essentially doing sampling, and the width *W* of the strokes is the sampling interval. The wider the stroke, the lower the sampling frequency. Therefore, the wider the strokes, the rougher the pencil drawing will be. Figure 6(g) is the result of $W = 9$ while $W = 5$ in Figure 6(f). The quantization order's influence is the same as the general case: the larger the quantization order is, the less obvious the block effect is. Usually, we fix *W* to be 5 pixels to get finer details. We fix the quantization order of direction to be 10 because too few directions will make the texture fluency worse, but too many directions will not improve the visual experience. The quantization order of gray level could be chosen from 8-16 according to the input. Inputs with more low-frequency components require more quantization orders. More detailed explanation of the hyperparameters' influence on the drawing result could be found in the supplementary material.

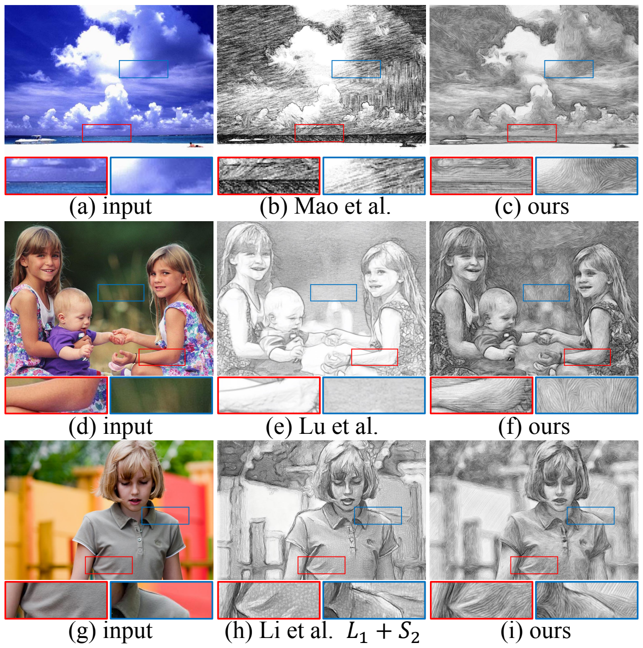


Figure 7: Comparison with several existing algorithms

Color space conversion All of the above work is performed on the gray image, and only need to do color space conversion to achieve coloring. We convert the original image to the YUV color space, replace the Y channel with the gray output, and then transfer back to RGB color space to obtain a colored pencil drawing. This coloring method is the same as (Lu, Xu, and Jia 2012). The second column from the right in Figure 1 shows our RGB result.

5 Experimental Results

In this section, we compare our results with several representative methods to prove the effectiveness of our algorithm. The pictures being compared are all from the original paper. Due to the page limit, we only compared with three methods in this article. More comparisons and comparisons with more methods (especially neural methods) can be found in the supplementary material.

Firstly, we compare our proposed method with a classic LIC-based method. In the first row of Figure 7, (a) is the input, (b) is the result of (Mao, Nagasaka, and Imamiya 2002), (c) is our result. Since LIC image is obtained by low-pass filtering a white noise input image, (b) introduces too much noise on the original image and looks dirty. The texture direction of (b) is very dull, with only three directions (30°, 90°, 135°), and the change of texture direction has nothing to do with semantic information. In the zoomed-in area (red border), the sea's texture direction in (b) is 135°while (c)'s is 0°, which is more in line with the texture of sea ripples.

Group	Evaluation	Score (Average)			
		[1]	[2]	[3]	[4]
1st	Stroke/Texture	66.4	52.8	73.1	88.6
	Tone/Contrast	78.1	72.6	84.5	87.7
	Stereoscopy	54.3	68.9	80.4	84.2
	Authenticity	60.7	65.3	83.1	95.3
	Overall	68.5	74.1	83.2	92.7
2nd	Overall	72.0	80.4	84.2	87.3

Table 1: People in the 1st group were asked to rate the result in terms of Stroke/Texture, Tone/Contrast, Stereoscopy, Authenticity, and Overall Perception, while the 2nd group only need to rate the last term. [1], [2], [3], [4] represent the methods of (Mao, Nagasaka, and Imamiya 2002), (Lu, Xu, and Jia 2012), (Li et al. 2019), and ours respectively. It can be seen that our method has the highest score in the opinions of both group of people.

In the zoomed-in area (blue border), the texture direction of (b) is rigidly fixed at 30°while (c)'s texture looks very fluent. (c) well expresses the feeling of clouds floating in the wind, and the overall visual effect is much cleaner and clearer than (b). The comparison shows that our method's result performs better in terms of aesthetic perception and is closer to the artists' technique.

Since we use (Lu, Xu, and Jia 2012)'s method to extract edges and enhance details, we make a comparison with (Lu, Xu, and Jia 2012) in the second row of Figure 7. (Lu, Xu, and Jia 2012)'s method can well depict the input's contours and edges, but their texture looks too smooth, more like adding a little noise to the gray image. Besides, (Lu, Xu, and Jia 2012)'s method uses histogram matching to fix the pencil drawing's tone, making many areas appear too pale, such as the girls' arms and the baby's head in (e), with only contours but no texture. In our result (f), the girls' arm and facial skin's texture direction is highly consistent with the actual structure of the human muscles, which is very aesthetic and vivid. The background in our result (f) has a strong style, while (e) lacks texture, very close to a gray image.

Now we compare with (Li et al. 2019), which is state of the art among neural methods. As shown in the third row of Figure 7, (g) is the input images, (h) is the result of (Li et al. 2019). Li et al.'s method can produce different kinds of style combinations. For example, L_1+S_2 means using the first type of outline and the second type of shading. So we just chose one of their style combinations that archives relatively good effects for comparison. Observe (h), it is not difficult to find apparent artifacts along the edges and borders, such as the girl's shoulder in the zoomed-in area (blue border). These artifacts deteriorate the fineness of their pencil drawing. (i) is our result, our method can preserve objects' contours clearly and portray tiny details. In terms of texture, it can be seen from the zoomed-in area (red border) that (h)'s texture cannot establish a direct connection with semantics (folds of clothes), and the texture of (h) can only reflect the shade of the input but not the input's structure. Our method displays the shade and fold of the cloth well at the same time.

Group	See Drawing Process	User Preference			
		[1]	[2]	[3]	[4]
1st	Before	3	14	25	58
	After	2	9	12	77
2nd	Before	2	19	27	52
	After	0	8	11	81

Table 2: User Preference indicates how many people voted for each algorithm. [1], [2], [3], [4] represent the methods of (Mao, Nagasaka, and Imamiya 2002), (Lu, Xu, and Jia 2012), (Li et al. 2019), and ours respectively.

6 User Study

We investigated the preferences of two groups of people (100 people in each group) to different pencil drawing algorithms. People in the first group (including ten recognized artists in our city and ninety professors/lecturers/graduates in painting-related majors from several universities) have received professional training in painting, while those in the second group do not. We gave ten sets of pictures to every participant. Every set of pictures included an input and the corresponding result of four pencil drawing algorithms. Participants didn't know the drawing was from which method and the ordering of four methods were shuffled in every set. For the first group of participants, we provided them with a series of subjective evaluation indicators and asked them to rate these indicators for each result; for the second group, we only asked them to rate every result based on their overall perception. Score range was limited in $0 \sim 100$ and participants were asked to give distinguished scores. The feedback results are shown in Table 1.

After completing the above survey, we used A, B, C, and D to anonymously represent these four algorithms and let each participant choose their favorite algorithm. We counted the number of people who voted for each algorithm. Then we showed our results' drawing process and informed participants that the other three methods couldn't generate process. Now we let the participants choose their favorite algorithm again. The survey results are shown in Table 2: most participants chose our method as their favorite; after watching our drawing process, more people voted for our method.

7 Conclusions

In our work, we statistically analyze and explain the texture of real pencil drawings and propose a controllable pencil stroke generation mechanism. On this basis, we implement an image-to-pencil automatic drawing algorithm: we use the edge tangent flow vector field to guide the direction of the strokes; use the gray image to determine the location, length, and shade of the strokes; use the edge map for detail enhancement. Our method is a mathematical procedural algorithm with good interpretability. Comparison with other pencil drawing algorithms shows our method outperforms in terms of texture quality, style, and user evaluation. Our most prominent advantage is that we have the drawing process.

Acknowledgments

This work was supported by National Science Foundation of China (U20B2072, 61976137, U1611461).

References

Chen, D.; Yuan, L.; Liao, J.; Yu, N.; and Hua, G. 2017. StyleBank: An Explicit Representation for Neural Image Style Transfer. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2770–2779.

Chen, H.; Liu, Z.; Rose, C.; Xu, Y.; Shum, H.-Y.; and Salesin, D. 2004. Example-based composite sketching of human portraits. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, 95–153.

Chen, Z.; Zhou, J.; Gao, X.; Li, L.; and Liu, J. 2008. A novel method for pencil drawing generation in non-photo-realistic rendering. In *Pacific-rim conference on multimedia*, 931–934. Springer.

Dodson, B. 1990. *Keys to drawing*. Penguin.

Durand, F.; Ostromoukhov, V.; Miller, M.; Duranleau, F.; and Dorsey, J. 2001. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Rendering Techniques 2001*, 71–82. Springer.

Fu, H.; Zhou, S.; Liu, L.; and Mitra, N. J. 2011. Animated construction of line drawings. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, 1–10.

Gao, X.; Zhou, J.; Chen, Z.; and Chen, Y. 2010. Automatic Generation of Pencil Sketch for 2D Images. In *ICASSP*, 1018–1021.

Ha, D.; and Eck, D. 2018. A Neural Representation of Sketch Drawings. In *International Conference on Learning Representations*.

Hertzmann, A.; Jacobs, C. E.; Oliver, N.; Curless, B.; and Salesin, D. H. 2001. Image analogies. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 327–340.

Hertzmann, A.; and Zorin, D. 2000. Illustrating smooth surfaces. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 517–526.

Hoffman, H. S. 1989. *Vision and the art of drawing*. Prentice-Hall, Inc.

Huang, Z.; Heng, W.; and Zhou, S. 2019. Learning to paint with model-based deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, 8709–8718.

Kang, H.; Lee, S.; and Chui, C. K. 2007. Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, 43–50.

Kelen, E.; et al. 1974. *Leonardo da Vinci's advice to artists*. Running Press.

Kong, Q.; Sheng, Y.; and Zhang, G. 2018. Hybrid noise for LIC-based pencil hatching simulation. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, 1–6. IEEE.

Lake, A.; Marshall, C.; Harris, M.; and Blackstein, M. 2000. Stylized rendering techniques for scalable real-time 3d animation. In *Proceedings of the 1st international symposium on Non-photorealistic animation and rendering*, 13–20.

Lee, H.; Kwon, S.; and Lee, S. 2006. Real-time pencil rendering. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, 37–45.

Li, N.; and Huang, Z. 2003. A feature-based pencil drawing method. In *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, 135–ff.

Li, Y.; Fang, C.; Hertzmann, A.; Shechtman, E.; and Yang, M.-H. 2019. Im2pencil: Controllable pencil illustration from photographs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1525–1534.

Lu, C.; Xu, L.; and Jia, J. 2012. Combining sketch and tone for pencil drawing production. In *Proceedings of the symposium on non-photorealistic animation and rendering*, 65–73. Citeseer.

Mao, X.; Nagasaka, Y.; and Imamiya, A. 2002. Automatic generation of pencil drawing using LIC. In *ACM SIGGRAPH 2002 conference abstracts and applications on*, 149–149.

Praun, E.; Hoppe, H.; Webb, M.; and Finkelstein, A. 2001. Real-time hatching. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 581.

Rosin, P.; and Collomosse, J. 2012. *Image and video-based artistic stylisation*, volume 42. Springer Science & Business Media.

Sousa, M. C.; and Buchanan, J. W. 1999a. Computer-generated graphite pencil rendering of 3D polygonal models. In *Computer Graphics Forum*, volume 18, 195–208. Wiley Online Library.

Sousa, M. C.; and Buchanan, J. W. 1999b. Observational model of blenders and erasers in computer-generated pencil rendering. In *Graphics Interface*, volume 99, 157–166.

Xie, D.-e.; Zhao, Y.; and Xu, D. 2007. An efficient approach for generating pencil filter and its implementation on GPU. In *2007 10th IEEE International Conference on Computer-Aided Design and Computer Graphics*, 185–190. IEEE.

Yamamoto, S.; Mo, X.; and Imamiya, A. 2004. Enhanced LIC pencil filter. In *Proceedings. International Conference on Computer Graphics, Imaging and Visualization, 2004. CGIV 2004.*, 251–256. IEEE.

Zheng, N.; Jiang, Y.; and Huang, D. 2019. StrokeNet: A Neural Painting Environment. In *International Conference on Learning Representations*.

Zuiderveld, K. 1994. Contrast limited adaptive histogram equalization. *Graphics gems* 474–485.