Convolution Filter Based Pencil Drawing and Its Implementation on GPU

Dang-en Xie[1], Yang Zhao[2], Dan Xu[1,*], and Xiaochuan Yang[3]

[1] School of Information Science & Engineering, Yunnan University 650091, China
[2] School of Information Science, Yunnan Normal University 650092, China
[3] South China University of Technology 510641, China
Tel.: +86-871-5737873 Fax: +86-871-5737873
danxu@vip.sina.com, xde820@gmail.com, xcy1198@163.com

Abstract. Traditional pencil drawing methods have their own drawbacks, such as modeling complexity and higher time-consuming. Thus, they are difficult to be suitable for the real-time applications. In the paper, we present a new pencil texture generating method based on the pencil filter. The method can conveniently generate the pencil drawing effect by convoluting the input image with the pencil filter. Moreover, the method is implemented on GPU, and then satisfies the requirement of real-time synthesis. Optical flow technique is used to guarantee the interframe coherence in video stylization.

Keywords: pencil filter, pencil drawings, Graphics Processing Unit (GPU), optical flow, non-photorealistic rendering.

1 Introduction

In the past decade, researchers in computer graphics community began to simulate traditional artistic media and styles, such as paintings [1], watercolor [2, 4], charcoal rendering [3, 5]. This is a new technique called Non-photorealistic rendering (NPR). Its purpose is not to aspire to the photorealism but to simulate the artist's work and represent the artistry, even the drawbacks of the artwork. To some extent, NPR is the complementarity of the photorealistic rendering.

Pencil drawing rendering is an important branch of NPR, which is firstly presented in the 90s last century [7, 8, 9, 10]. A key step of pencil drawing rendering is how to simulate the pencil texture. Sousa [8] attempted to model the physical behavior of pencil, paper and eraser. Their approach attained the pencil texture vividly. Later, Mao [6] simulates pencil texture using the line integral convolution (LIC) method. Also, they gain satisfied effect. LIC is a texture based vector field visualization technique which was first presented by Cabral and Leedom in 1993[10]. Given a 2D vector field represented as a regular Cartesian grid, the LIC algorithm takes as input a white noise image of the same size as the vector field and generates an output image wherein the texture has been locally blurred in the direction of the vector field.

* Corresponding author.

M. Xu et al. (Eds.): APPT 2007, LNCS 4847, pp. 723–732, 2007.

Although the traditional pencil texture generating methods obtain good effect, they all have disadvantages of deficiency and time-consuming. Physical modeling is very complexity. LIC method needs to calculate the visualization vector field of the input image, and then convolute the pixel one by one, so it also costs much time. Generally, using the LIC method to generate a pencil drawing needs about 20 minutes (here the image size is 1024*768) [10]. Both of the methods are not suitable to the real-time applications.

In this paper, we present a new method for simulating pencil texture by pre-calculating a special convolution filter, named pencil filter. It may have different appearances according to different stroke orientations and different stroke sizes. Once the pencil filters are made ready, we convolute the pixel of the black noise image with each corresponding filter, and then the pencil drawing image is accomplished. The proposed approach saves lots of time because pencil filters are generated in advance and the convolution operation is more efficient than traditional methods.

Obviously, the method can be extended to process video. To preserve interframe coherence of a video segment, the optical flow technique is adopted, which will be described in detail in Section 4. Additional, for real-time stylization applications, the paper performs an achievement of the method on GPU (Graphics Processing Unit).

2 Image Based Pencil Drawing

All existing pencil drawing techniques can be classified primarily into two kinds: geometry-based and image based. Geometry-based techniques take 3D scene descriptions as their input. Image based techniques directly process 2D images to get pencil drawing expressions. Our method belongs to the latter. Figure 1 shows the framework of our pencil drawing algorithm. Each processing box corresponds to a step of the algorithm:

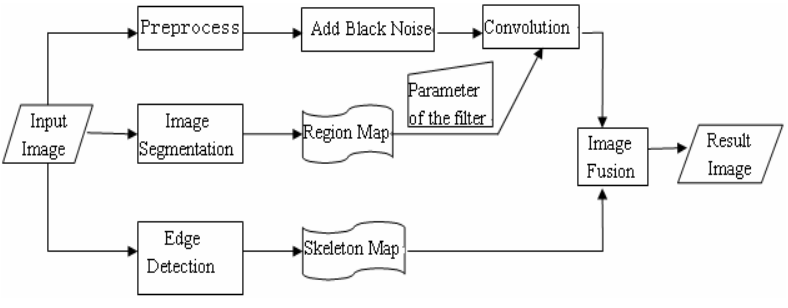
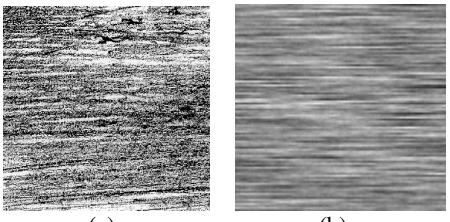


Fig. 1. The frame map of the algorithm

2.1 Generating the Pencil Filter

By observing and analyzing the real pencil texture (see Figure 2(a)), we simply suppose that: 1) graphite marks present stochastic distribution according to the coarseness of papers; 2) graphite marks stretch along the stroke tracks; 3) graphite marks on perpendicular direction of a stroke present obviously black-white staggered distribution.



(a) (b)

Fig. 2. Comparison of the real pencil texture (a) with the pencil filter generated texture (b)

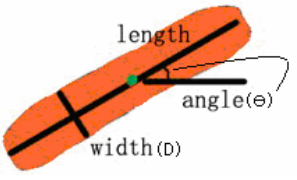


Fig. 3. Properties of the pencil stroke model

Considering the above supposes, we create a mathematic model for pencil filter. Assume that the stroke length is *len*, the stroke direction isθ and the stroke width is *2D*. As shown in Figure 3, if we know the stroke length and the stroke orientation, we can easily calculate the template size by $(\lceil len * \sin\theta \rceil \times \lceil len * \cos\theta \rceil)$.

The next problem is how to decide the value of each element in the pencil filter. As shown in Figure 4, firstly, calculate the distance *d* from each point *P* to the central axis *l* of a stroke. Then calculate the distance *r* from the point *P* to the center *O* of the stroke. The value of each element in the pencil filter lies on the relation between *d* and *D*, and also the relation between *r* and *len/2*.

Here, we take the upper right quarter of the template as an example (Figure 4(a)). Obviously, only three kinds of points are presented in the template. Points in the green area (e.g. P in Figure 4(b)) satisfy the conditions that *r* is less than *len/2* and *d* is

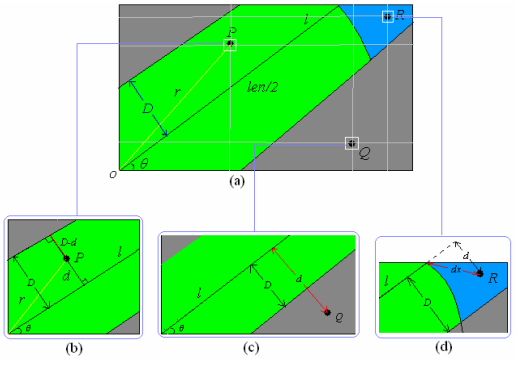


Fig. 4. Define a pencil filter

less than D, so we choose D-d as their values. Points in the gray area (e.g. Q in Figure 4(c)) satisfy the condition that d is large than D, which means the stroke can't covered this area, so their values are set to be zero. Points in the blue area (e.g. R in Figure 4(d)), satisfy the conditions that r is large than $len/2$ and d is less than D. Blue area is near to the stroke end, and the graphite marks is thin there, so the value in the template is less than D-d and none zero. We calculate the distance dx from point R to the end of line l, and then we choose D-dx as the value of this area. In this way, it decreases the value of the stroke end effectively, and the decrement is in proportion to the distance r. When D-dx is less than zero, the value should be set to zero.

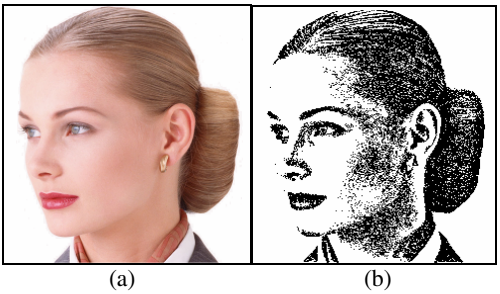
Viewing the pencil filter generating procedure, it properly simulates the real pencil texture:

1. The points near the stroke central line l have greater values. The larger the distance, the smaller the value. That is in accord with the real pencil texture property.
2. When the point beyond the stroke width, the value is set to zero. It insures the pencil stroke width;
3. The filter kindly simulates the stroke ends' physical property—graphite marks tapered with the disappear of the pressure on papers;
4. The points which have the equal distance to the central axis line have the equal value in the template. It insures the strength direction along the stroke.

2.2 Generating the Black Noise Image

To make sure the pencil texture has stochastic distribution, we generate the black noise image from the reference image. Our method for generating the black noise image is similar with the method for adding white noise in Mao [10]. We use the tone of the input image to guide the distribution of the black noise. Let I_{input} be the intensity of a pixel in the input image, P is a floating-point number generated with a pseudo-random function, and then the intensity I_{input} of the corresponding pixel in the noise image is decided in the following way:

$$I_{noise} = \begin{cases} 255, & if \quad P \le T \qquad P \in [0.0, 1.0] \\ 0, & otherwise \end{cases}, \quad T = k \cdot \left(\frac{I_{input}}{255} \right), \quad k \in (0.0, 1.0], \tag{1}$$



(a) (b)

Fig. 5. The black noise image ((a) is the original image; (b) is the corresponding black noise image)

in which k is a coefficient for controlling the density of the black noise. In this way we can promise the pencil drawings have the stochastic distribution character, and also promise the density of the black noise correspond to the intensity of the input image. If the intensity of the current pixel is lower, the value of T is smaller, and the probability of P large than T is larger, so the value of I_{noise} has more probability to be 0. On the contrary, the output value I_{noise} is 255. Figure 5(b) is the black noise image generated from the input image shown in Figure 5(a).

2.3 Extract the Contour Lines

A simplest artistic expression is extruding the outlines of the artwork. It is also an important step for generating the pencil drawing. In computer, we need to extract the contour lines for simulating the action of extruding outlines.

Gradient operators, such as Sobel, Robert, Prewitt and Kirsch operators are commonly used in digital image processing to extract edges of an image. Considering Kirsch operator [11] has the bigger weighted factors, we prefer to choose the Kirsch gradient operator to extract the contour lines in this paper, so that we can obtain the contour lines clearly.

The Kirsch operator has 8 filters. Formally, the Kirsch operator is defined by:

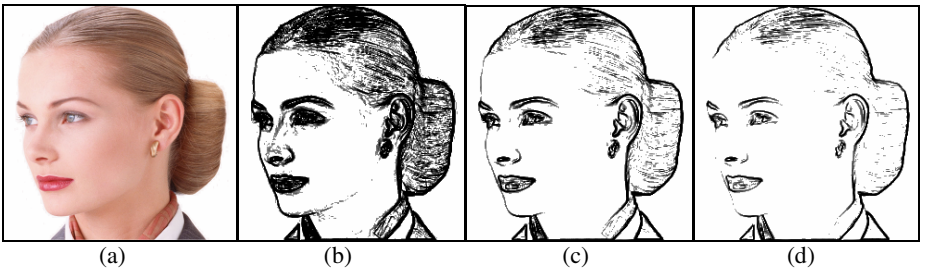
$$K(x, y) = \max\{1, \max\{5S_i - 3T_i\}\}, i = 0 \cdots 7 \tag{2}$$

in which,

$$S_i = f(A_i) + f(A_{i+1}) + f(A_{i+2}), \tag{3}$$

$$f(A_i) \text{ stands for the pixel value on position } A_i$$

$$T_i = f(A_{i+3}) + f(A_{i+4}) + f(A_{i+5}) + f(A_{i+6}) + f(A_{i+7}) \tag{4}$$



(a) (b) (c) (d)

Fig. 6. The contour lines maps with different value of μ

In practice, we are used to change the values of the Kirsch operator according to different images. Let K_i be the filter of the Kirsch operator, then we have:

$$K_i = \mu * K_i, i = 0 \cdots 7, \mu \in (0,1] \tag{5}$$

Here, μ is a coefficient for controlling the weight value in the filter. One can adjust the value of μ interactively, but we suggest that the value of μ should between 0

and 1. Figure 6 shows the different contour line maps with the different value of μ. Figure 6(b), (c) and (d) are show the results that μ is 1, 0.5 and 0.3, respectively. Generally, if an image has more details, the value of μ should be smaller. A smaller μ can prevent the contour line conglutination. On the contrary, if an image has little details, a larger μ should be set in order to make sure the consistency of the contour lines.

3 Implementation on GPU

In this paper, we transplant the pencil filter based stylization algorithm onto GPU. With the GPU's powerful parallel processing ability, we have achieved the real-time video synthesis for pencil drawing style. In this section, we will describe the GPU implementation of our algorithm in detail.

3.1 Convolution on GPU

It is difficult to generate pencil filter directly on GPU because the instructions and the registers are limited in GPU. Fortunately, we can generate the weight values of pencil filter in advance (like what shown in Figure 7). This idea makes a way to using our method on GPU. Firstly we load the weight values of pencil filter into GPU, which is generated in CPU in advance, and then convolution is executed for each pixel.

0	0.1509	0.3767
0.1509	0.6431	0.1509
0.3767	0.1509	0

0	0.0000	0.3293
0.3293	0.6827	0.3293
0.3293	0.0000	0

0	0	0	0.1001	0.0608
0	0	0.1038	0.3543	0.1001
0	0.1038	0.3543	0.1038	0
0.1001	0.3543	0.1038	0	0
0.0608	0.1001	0	0	0

Fig. 7. pencil filters using our method generated with the template parameters (left: *D*=1, *len*=3, θ=45°; middle: *D*=1, *len*=3, θ=30°; right: *D*=1, *len*=5, θ=45°)

Many effective means can achieve the convolution on GPU. Generally, we can store the weight values of the convolution template as a constant or uniforms type, and then execute the convolution operation. This method seems simple and feasible, but the fact is that if we solidify these constants into GPU's shader program, we can not easily expand the program function later on. It's not suitable to our method because we need to use the weight values to simulate the different property of the pencil stroke, such as stroke length, stoke width and stroke orientation. To solve the problem, we load the template data as a texture image, and call the image as Filter Texture. When a video fragment is synthesized in real-time, the corresponding filter texture will be chosen according to the user's need, and then convolution is executed. The convolution process can be expressed by:

$$filtered\ color = \frac{\sum k(s-s_0,t-t_0)tex(s,t)}{\sum k(s-s_0,t-t_0)} \tag{6}$$

where the current pixel position is (s0, t0), tex is a function for searching the corresponding filter texture. To save the memory, we store the total value of the template on the alpha channel.

3.2 Generating the Black Noise Image on GPU

At present, there's no function supplied for generating the floating-point pseudo-random number in GPU. Therefore, we generate a noise texture image in advance; the pixel value of the noise image is making up of floating-point random number. GPU can thereby get the floating-point random number by sampling the noise texture image. Then, we can generate the black noise image on GPU according to the approach described in section 2.2.

4 Stylization for Video Segment

The method described in Section 2, also can be used for rendering video segments. First, capture each frame from the input video. Then, use the same way to process each frame like what is used to deal with a single picture. Finally, rebuild the video with the processed frames. This procedure usually brings a negative effect to the result video because it cannot preserve the interframe coherence.

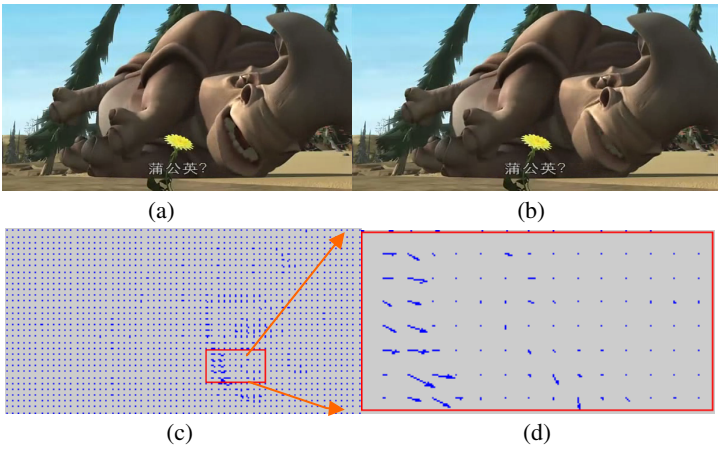


Fig. 8. Calculate the optical flow field((a) (b) are two adjacent frames. (c) is the optical flow vector field.(d) is the enlarged image of the red region of (c).)

To solve the problem, we use Horn's method [12] to estimate the optical flow field of each two adjacent frames. Figure 9 shows the optical flow vector field. As synthesis a frame, we compare the magnitude of the current pixel with an appointed threshold. If the former is small, then we believe this pixel is almost still. So we need only to copy the previous frame's corresponding pixel to the current pixel. Otherwise, we recalculate the value of the current pixel following with the method described in Section 2. Generally, the scene's change is very small in a pair of adjacent frames. So the

magnitudes of optical flow vectors are often to be zero or very small on most pixel positions, especially in the background. In this way, we can basically keep the interframe coherence of the video.

5 Experiment Result

A pencil drawing generating system on Windows environment has been built with the Matlab tool. Basically when the input image is specified, the system can generate the pencil drawing picture automatically. Users are allowed to specify some parameters interactively. These parameters control the stroke orientation, the stroke length, the density of the black points and the coefficients of the Kirsch operator. Figure 9 are some results generated by our method for static pictures. Figure 10 shows some video frames of real-time rendering results on GPU. In existing system, we do not allow the users to change the parameters during process a video stream. Instead, we set the default values for parameters in advance. Thus, the result quality is not good as static pictures because the default values are not usually suitable for all the frames.

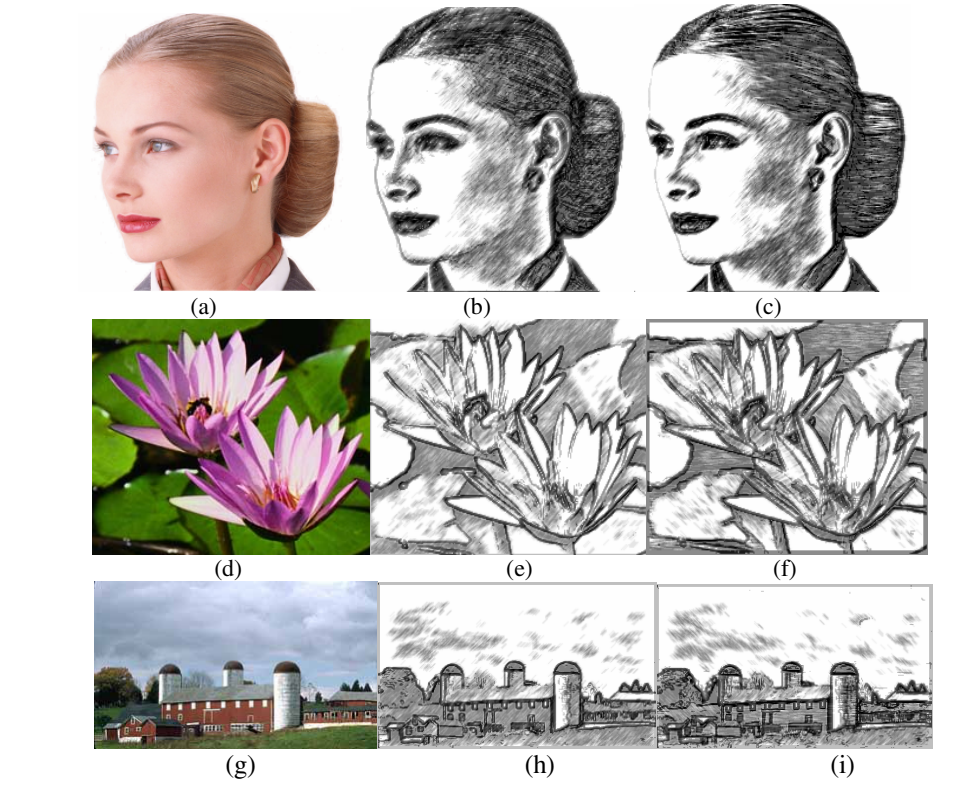


Fig. 9. Experiment results on CPU(figure(a),(d),(g) are original images; figure (b),(e),(h)are the pencil drawing images with single stroke orientation 45^0;figure (c),(f),(i) are the pencil drawing images with different stroke orientations.

Compared with the LIC method, our method saves lots of time. The primary reason is our method needn't to calculate the visualization vector field of the input image, and also needn't to do the hundreds of iterations. Our method only needs to do the convolution once for each pixel, and the kernel operator elements (Figure 7) usually have many zeros.
The system is developed using Matlab7.04. All experiments run on a 1.73GHz Pentium PC with 512M RAM. Cg(C for graphics) language is used for GPU programming.

Fig. 10. Real-time rendering result on GPU for some frames of the movie *Ice Age*

6 Conclusion

This paper presents a simple and efficient method for simulating pencil texture. Using the method, we accomplished an image based pencil drawing algorithm. Also, the method is successfully implemented on GPU to support real-time video stylization. The proposed method has the following advantages: (1) Efficient. The time cost for rendering a 1024*768 image on Matlab is 43.26 seconds; it is far less than LIC method, which needs about 20 minutes. (2) Convenient. Only single convolution is needed for pencil drawing image synthesis. (3) Bring a new way for real-time synthesis. It brings a significant reference for the other computer hardware, such as FPGA.

Acknowledgement

This work is supported by NSFC (No. 60663010) and NSF (No. 2006F0017M) of Yunnan province. All images are downloaded from the Internet. The video fragments are captured from the movie of "Ice Age".

References

1. Hertzmann, A.: Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In: SIGGRAPH 1998 conference proceedings, pp. 453–460 (1998)
2. Laerhoven, T.V., Reeth, F.V.: Real-time simulation of watery paint. Computer Animation and Virtual Worlds 16, 3–4, 429–439 (2005)

3. Lee, H., Kwon, S., Lee, S.: Real-Time Pencil Rendering. In: Proc. of the 4th Intl' Symposium on Non-Photorealistic Animation and Rendering, pp. 37–45 (2006)
4. Luft, T., Deussen, O.: Interactive watercolor animations. In: Proc. Pacific Graphics 2005, pp. 7–9 (2005)
5. Majumder, A., Gopi, M.: Hardware accelerated real time charcoal rendering. In: Proc. NPAR 2002, pp. 59–66 (2002)
6. Mao, X., Nagasaka, Y., Imamiya, A.: Automatic Generation of Pencil Drawing from 2D Images Using Line Integral Convolution. In: Proceedings of the Senventh International Conference on Computer Aided Design and Computer Graphics CAD/GRAPHICS 2001, pp. 240–248 (2001)
7. Takagi, S., Fujishiro, I., Nakajima, M.: Volumetric modeling of colored pencil drawing. In: Pacific Graphics 1999 conference proceedings, pp. 250–258 (1999)
8. Sousa, M.C., Buchanan, J.W.: Observational Model of Blenders and Erasers in Computer-Generated Pencil Rendering. In: Graphics Interface 1999 conference proceedings, pp. 157–166 (1999)
9. Sousa, M.C., Buchanan, J.W.: Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models. In: EUROGRAPHICS 1999 conference proceedings, pp. 195–207 (1999)
10. Cabral, B., Leedom, C.: Imaging Vector Field Using Line Integral Convolution. In: SIGGRAPH 1993 conference Proceeding, pp. 263–270 (1993)
11. Castleman, K.R.: Digital Image Processing, pp. 390–391. Publishing House of Electronics Industry, Beijing (2002)
12. Horn, B.K.P., Schunck, B.G.: Determining optical flow. Artificial Intelligence 17, 185–203 (1981)