Feature-guided Convolution for Pencil Rendering

Heekyung Yang[1] and Kyungha Min[2]
[1] Dept. of Computer Science, Graduate School, Sangmyung Univ.,
Hongji-dong, Jongro-gu, Seoul, Korea
[e-mail: hkyang@smu.ac.kr]
[2] Div. of Digital Media, School of Software, Sangmyung Univ.,
Hongji-dong, Jongro-gu, Seoul, Korea
[e-mail: rminkh@smu.ac.kr]
*Corresponding author: Kyungha Min

Received January 24, 2011; revised April 21, 2011; revised June 8, 2011, accepted July 8, 2011; published July 28, 2011

Abstract

We re-render a photographic image as a simulated pencil drawing using two independent line integral convolution (LIC) algorithms that express tone and feature lines. The LIC for tone is then applied in the same direction across the image, while the LIC for features is applied in pixels close to each feature line in the direction of that line. Features are extracted using the coherent line scheme. Changing the direction and range of the LICs allows a wide range of pencil drawing style to be mimicked. We tested our algorithm on diverse images and obtained encouraging results.

Keywords: Non-photorealistic rendering, pencil drawing, convolution, feature, coherent lines

This research is supported by Basic Science Research Program through the Korea Research Foundation (KRF) funded by the Ministry of Education, Science and Technology at 2011 (2011-0005458).

DOI: 10.3837/tiis.2011.07.006

1. Introduction

The simulation of artistic media such as pencils [1][2][3][4][5][6][7][8][9][10][11][12][13][14][15][16], pen and ink [17][18][19][20][21], hatching [22][23][24][25][26] or brushes [27][28][29][30][31][32] has been a durable and interesting research topic in NPR (non-photorealistic rendering). In particular, there has been a lot of work on pencil rendering that aims to mimic pencil drawings. Like other NPR techniques, approaches to pencil rendering can be classified by domain: some techniques re-render an image [1][2][3][5][6][7][8][9][10][12][13][14][15][16], while others start with a 3D mesh [4][22][23][11][15][25].

We introduce a new pencil rendering algorithm that re-renders a photograph into a monochrome pencil drawing. The most widely used scheme for producing pencil drawing effects is line integral convolution (LIC) [2], which integrates the noises scattered on the pixels of an image along an integration direction. The existing LIC-based schemes [5][19][7][8][13], however, cannot produce visually pleasing pencil drawing effects, since they have limitations in implementing real pencil drawing art techniques. In real art, an artist depicts a scene using a pencil by hatching areas and emphasizing important features with thicker strokes. Several pencil rendering techniques [6][7][9][14] used features, but their results are far from pencil drawing images. However, authors [31][32] have successfully exploited shape information such as flow or image parsing hierarchy to re-render images in a painterly style. Brush strokes are aligned along flow in an image extracted by RBF function in [31], and the type of brush strokes to apply is determined using image parsing hierarchy in [32]. Our pencil rendering algorithm uses LIC scheme to produce pencil drawing effects. A key idea is that we extract features from an image and construct a smooth flow that follows the feature. We exploit the smooth feature flow as the integration direction for LIC, because we believe that the underlying problem of the existing schemes is the way in which features extracted from an image are utilized. In our scheme, two feature-guided LIC processes are applied independently: one re-renders areas of smooth tone and the other emphasizes the salient features of images. The resulting pencil drawings are more convincing than those produced by many existing techniques.

Our scheme is outlined in **Fig. 1**. We start by extracting features from an image and vectorize them. Next, we enhance the contrast of the image and add some noise to assist the LIC processes: tonal LIC (tLIC) is used to create hatching effects and feature LIC (fLIC) creates bold strokes that replicates the important features of the image. The results of these two LIC processes are merged to produce the final pencil rendering.

Our most important contribution is that we present a pencil drawing scheme that presents visually pleasing results. Note that the quality of the result is a key issue in evaluating pencil drawing algorithms. We have attacked this problem by applying different styles of pencil strokes according to the features extracted from the image. This strategy, which mimics the artistic pencil drawing technique, produces visually pleasing pencil drawings. Another contribution is that our scheme also allows considerable latitude in controlling the style of pencil drawing.

The rest of paper is organized as follows: In Section 2, we briefly review related work. We describe the pre-processing steps in Section 3 and the rendering algorithm in Section 4. In Section 5, we present details of our implementation and experimental results. Finally, we conclude this paper and discuss future research directions in Section 6.

2. Related Work

2.1 Pencil drawing from images

Image-based schemes can be further classified into pen-and-ink illustration [17][18][19][20], and those based on physical models [1][2][3][16], LIC [5][6][7][8][13][14] and stroke textures [9][10][12].

Some approaches to pen-and-ink illustration produce results very similar to pencil drawings. Salisbury et al. [17] and Winkenbach and Salesin [18] presented pen-and-ink illustration schemes in which textures made up of various tones and patterns were used to re-render an image. Later, Salisbury et al. [19] extended their scheme for scale-dependent reproduction of illustrations. Salisbury et al. [20] also introduced an approach in which stroke textures are arranged along user-defined directions over an image to create a pen-and-ink illustration. The results of these schemes look like line illustrations rather than pencil drawings.

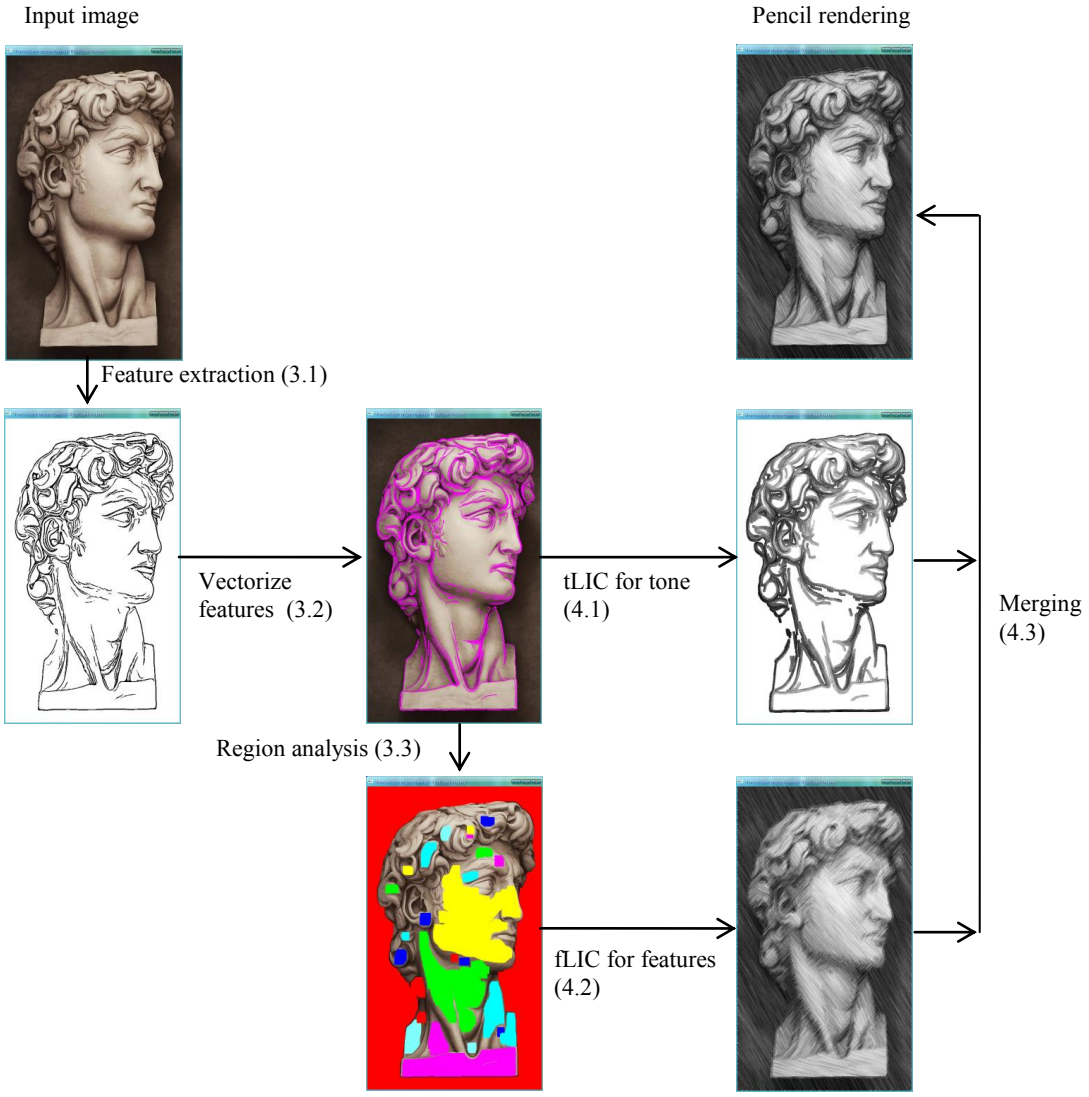


Fig. 1. Overview of the algorithm.

One approach to creating pencil drawings is to model the physical process involved. Sousa and Buchanan [28] modeled the physics of a graphite pencil drawing on paper, and used this model to render 3D scene. They went on [19] to model the action of blenders and erasers. Takagi et al. [29] contributed a volumetric model of paper. AlMeraj et al. [24] collected and analyzed actual pencil strokes to conduct a drawing scheme. These techniques produce realistic results, but they are difficult to control and require a lot of computation.

Line integral convolution (LIC) [2] can be used to visualize vector fields as textures, and thus offers a way of producing hatching. Mao et al. [5] pioneered this application of LIC. They segment an image into a number of regions, each of which is then rendered with strokes in the same direction; but this uniformity is not very realistic. Li and Huang [6] got around this problem by using gradient vectors and edges to determine stroke directions. This gives more variety, but is unable to express tonal details. Yamamoto et al. [7] added contours and paper surface effects, and also showed how to overlap textures to match the intensities of different regions of an image. They [8] also applied LIC to colored pencil drawing blending colors using the Kubelka-Monk model. Xie et al. [13] extended the scope of LIC-based rendering schemes to video, using GPU programming techniques. Chen et al. [14] used the Sobel operator to find edge information, which is added to the pencil drawing produced by LIC. But this scheme still suffers from the lack of realism, detailed shading, and contours.

In stroke-based schemes, originally brush-strokes are created along a direction field derived from an image. This approach can be extended to other artistic media. Matsui et al. [9] produced crayon drawings by overlapping stroke textures along boundary curves, and Murakami et al. [10] represented the effects produced by a range of media, including pastels, charcoal and crayons. Melikhov et al. [12] applied pencil textures to curves approximating the skeletons of objects in an image. But stroke-based schemes are not a good match with the hatching patterns seen in monochrome pencil drawings: it is difficult to control the strokes in the way necessary to create a realistic pencil drawing from an image.

2.2 Pencil drawing from 3D meshes

Most pencil drawing algorithms that start with a 3D mesh use tonal maps to build textures to represent variations in intensity. This approach can achieve accurate shading and attractive hatching, and also reproduce feature lines such as contours. Schemes based on tonal maps have been used to produce hatching effects [22][23][25] and line illustrations [15], as well as pencil drawing [4][11]. In particular, Praun et al. [22] developed the artistic tonal map for hatching a 3D mesh in real time. Webb et al. [23] extended this to achieve fine control of tone. Paiva et al. [25] presented a fluid-based hatching scheme to render triangular meshes of arbitrary topology and complicated geometry. The texture maps are applied to the mesh along the directions estimated by fluid equations.

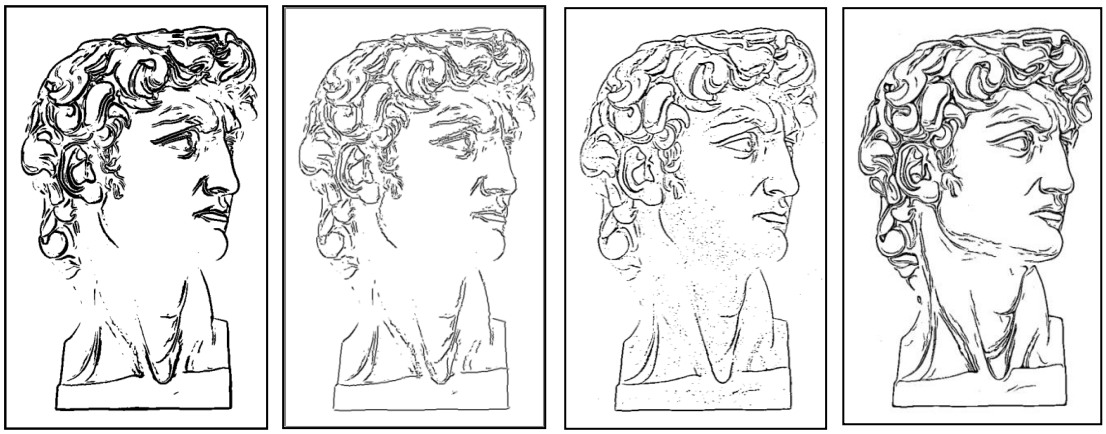
Lake et al. [4] used a tonal map to create a pencil drawing from a 3D mesh, but they do not consider appropriate tonal effects. Lee et al. [11] render triangular meshes using a tonal map by generating various pencil textures and applying them in direction of principal curvature. Their scheme is accelerated by the use of GPU. Kim et al. [15] extended this scheme to generate line art illustrations of dynamic 3D objects with highly reflective surfaces.

3. Preprocessing

In the preprocessing step, we extract and vectorize feature lines and then segment image using those lines.

3.1 Feature extraction

Features are a set of pixels with similar intensities, adjacent to other pixels with different intensities. Sobel operators and Canny edge detectors are frequently used to extract features from images; but more complicated schemes such as the difference of Gaussian (DoG) filter have recently been shown to achieve better results. We make use of coherent lines [33], which are obtained by applying a DoG filter to the flow of an image. We use the edge tangent flow (ETF), which is obtained by averaging the tangent vectors of the pixels. Coherent lines are sets of pixels which have DoG values, obtained from the flow, that are greater than a threshold. In **Fig. 2**, we show edges found by the Sobel, Canny, and DoG filters, together with coherent lines.



(a) Sobel **(b)** Canny **(c)** DoG **(d)** Coherent lines

Fig. 2. Comparison of feature extraction algorithms.

3.2 Vectorizing features

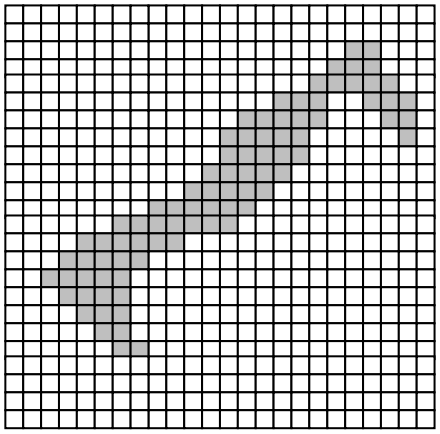
The operations that we need to perform on feature lines in pencil rendering are finding the nearest feature lines to a pixel and computing distances from a pixel to feature lines. Unfortunately, features represented as sets of pixels cannot support these operations efficiently. Therefore, we vectorize the features that we have extracted as coherent lines using following steps. An example of a coherent line is shown in **Fig. 3 (a)**.

(1) At each pixel in a feature, we construct a stream by following the ETF vectors computed as described in section 3.1 (see also **Fig. 3 (b)**). The process of following an ETF vector stops at a pixel where that vector at the pixel changes rapidly or is undefined. If there are *n* pixels in a coherent line, then at most *n* streams will be produced.

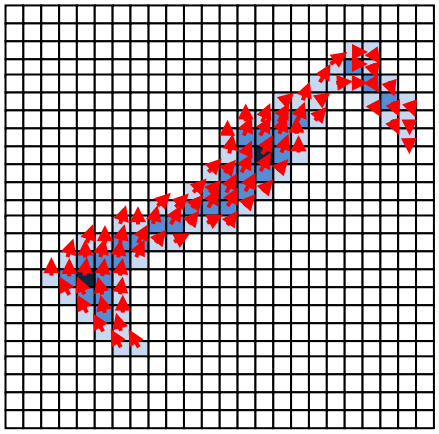
(2) We assign a *center weight* to each pixel that belongs to a feature. The weights of the pixels on the boundary of a feature are initialized to 1. Then the weights are propagated from the boundary of a feature toward its center by increasing the weight by 1 at each step (see **Fig. 3 (c)**).

(3) We then estimate a center weight for each stream by averaging the center weights of all the pixels in the stream (see the red curves in **Fig. 3 (d)**).

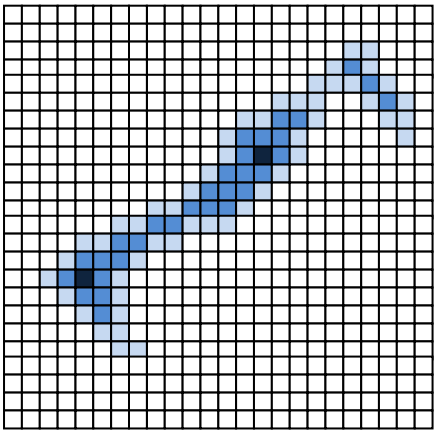
(4) We then select streams with center weights that are greater than those of their neighboring streams (see the green curves in **Fig. 3 (d)**).



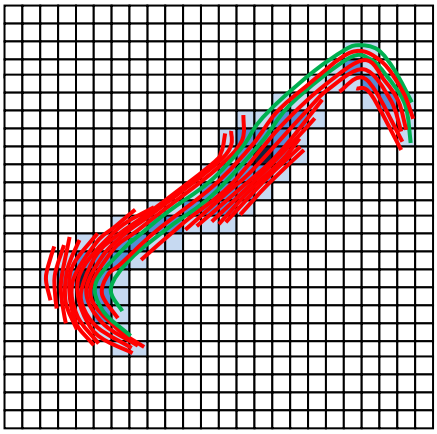
(a) A sample coherent line (grey pixels)



(b) ETF vectors at the pixels on a coherent line



(c) Estimating center weights from the boundary of a coherent line (light blue = 1, blue = 2, dark blue = 3)



(d) Estimating streams (red curves) and determining the stream with the local maximum (green curves)

Fig. 3. Vectorizing features.

The streams selected by this process can be regarded as vectorized feature lines, which segment an image into several regions. A region whose size is less than a threshold are ignored as noise. Conventionally, in image processing, the threshold is set to 10.

4. Pencil rendering using LIC

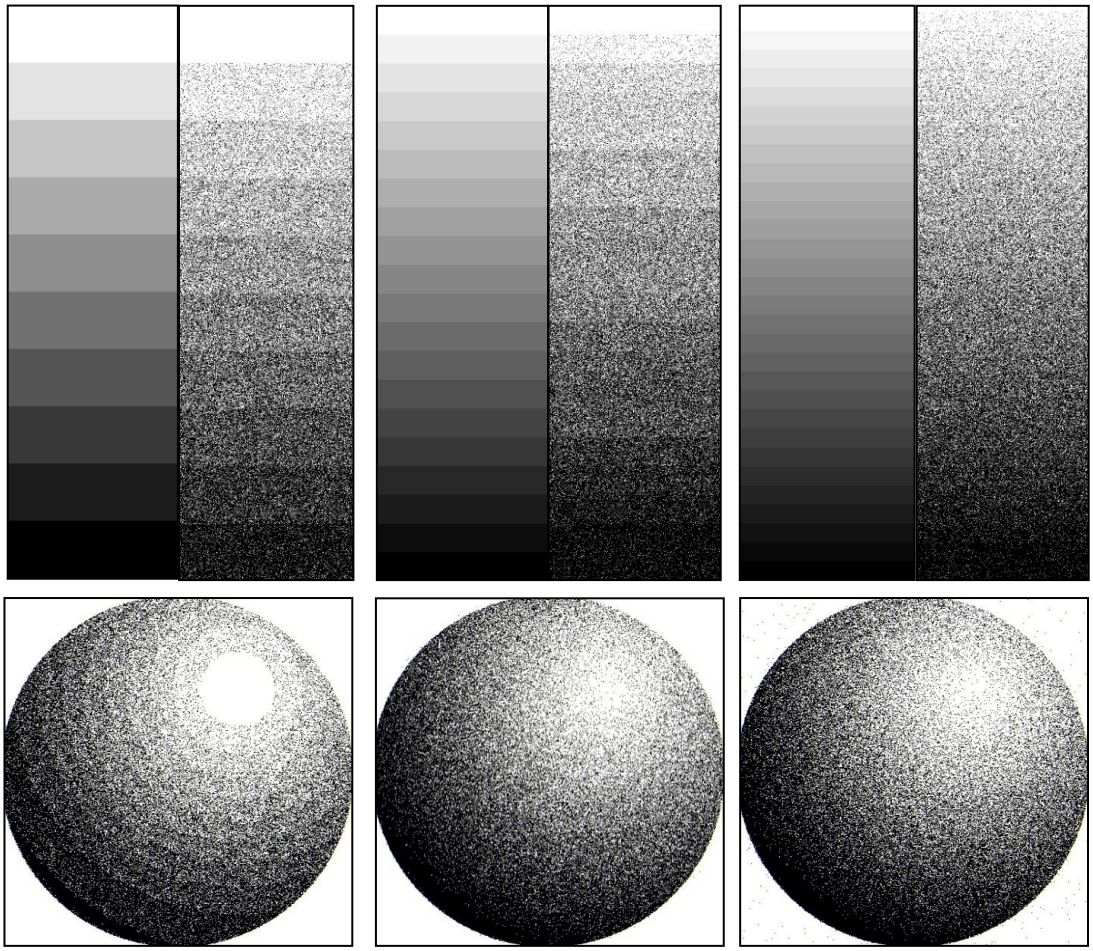
We use line integral convolution (LIC) to produce pencil effects. The computation of an LIC requires an image to contain noise, which is then integrated in the directions of the lines.

4.1 Generating noise

We generate monochrome noise with an intensity that is proportional to pixel intensity. However, in line drawings, artists tend to express gray scales in M tonal steps. Therefore, we generate noise with stepped intensity. We denote the noise at a pixel $\boldsymbol{p}$ whose intensity is i_p as $N(\boldsymbol{p})$, which has a binary value, either BLACK or WHITE. $N(\boldsymbol{p})$ is determined as follows:

```
if ( (rand( )%(M+1))/M ) > i_p )
    N(p) = BLACK;
else
    N(p) = WHITE;
```

M denotes the number of tonal steps, and `rand ( )` is a random number generation function. In the procedure, "`rand( )%(M+1)`" generates a number in (0, *M*), and "`(rand( )%(M+1))/M`" generates a number in (0, 1). A darker pixel whose i_p is small has a higher probability for generating BLACK noise, while a whiter pixel whose i_p is large has a higher probability for generating WHITE noise. Results for M = 10, 20, and 30 are shown in **Fig. 4**.



(a) 10 tones **(b)** 20 tones **(c)** 30 tones

Fig. 4. Noise generated to produce different numbers of tones.

4.2 tLIC: LIC for tone

We perform LIC on the noise generated using the procedure described in Section 4.1. A line integral convolution has the following form:

$$\int_{-L}^{L} N(\mathrm{p}+t\,\mathrm{d})dt \tag{1}$$

This integration is performed at pixel **p** in direction **d** over the range of (-*L*, *L*). N(**x**) denotes the noise on the pixel **x**. The integration procedure is illustrated in **Fig. 5**.

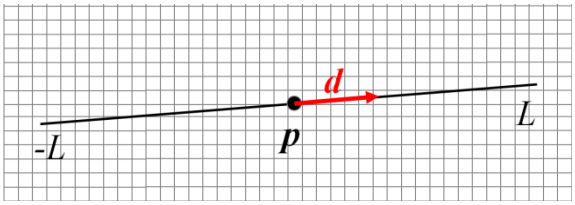


Fig. 5. Integrating noises with pixels lying on (**p** – *L***d**, **p** + *L***d**).

We can control the style of pencil rendering using the parameters *L* and **d**. Increasing *L* produces a smoother style. **Fig. 6** shows uni-directional hatching and **Fig. 7** shows cross-hatching. The direction **d** determines whether there will be uni-directional hatching or cross-hatching. To achieve cross-hatching, the direction of integration **d** at each pixel is determined as follows:

```
if ( rand( )%2) = 0 )
    d = d0;
else
    d = d0 x Z;
```

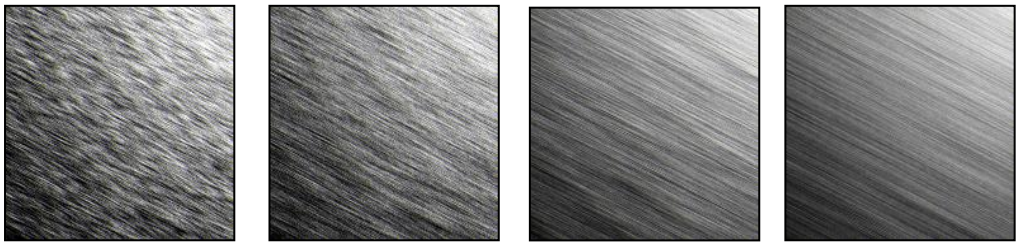
The integration direction **d** at each pixel is either along $\mathbf{d}_0$ or orthogonal to $\mathbf{d}_0$ at same probability. Note that "`rand( )%2`" generates 0 or 1 at same probability. The operator " x " is a cross-product operator, Z denotes the direction of the positive *z*-axis, which is orthogonal to the plane of the image.

The direction of integration **d** also affects the pencil rendering style. We can perturb **d** as follows:

```
d = rotate ( d0, δ );
```

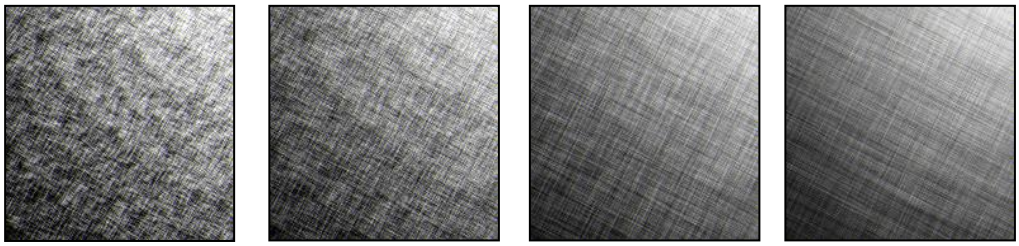
Increasing δ smudges the pencil strokes. **Figs. 8** and **9** show the effects on uni-directional hatching and cross-hatching. The following procedures shows how tLIC is performed at each pixel. **Fig. 11 (a)** shows a tLIC image rendered using *L* = 25 and δ = 7.5°.

```
for each pixel p
    determine d;
    determine L;
    tLIC(p) ← Integrate ( p, d, L );
```



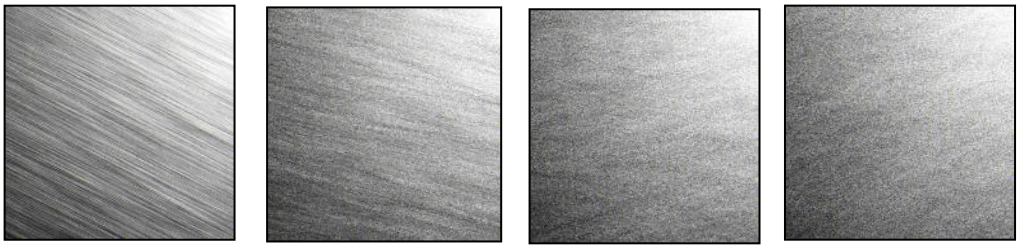
(a) $L = 5$ **(b)** $L = 10$ **(c)** $L = 25$ **(d)** $L = 50$

Fig. 6. tLIC results with various values of L for uni-directional hatching.



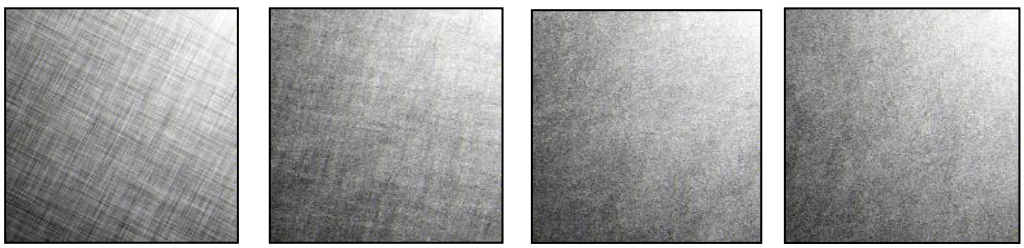
(b) $L = 5$ **(c)** $L = 10$ **(d)** $L = 25$ **(e)** $L = 50$

Fig. 7. tLIC results with various values of L for cross-hatching.



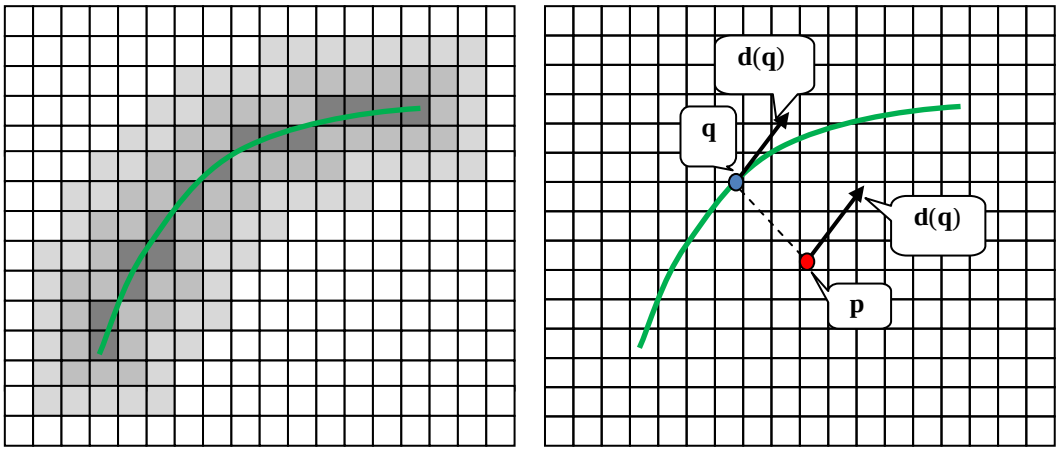
(a) $\delta = 0^\circ$ **(b)** $\delta = 35^\circ$ **(c)** $\delta = 60^\circ$ **(d)** $\delta = 90^\circ$

Fig. 8. tLIC results with various perturbations for uni-directional hatching ($L = 25$)



(a) $\delta = 0^\circ$ **(b)** $\delta = 35^\circ$ **(c)** $\delta = 60^\circ$ **(d)** $\delta = 90^\circ$

Fig. 9. tLIC results with various perturbations for cross-hatching ($L = 25$)



(a) Vectorized feature **(b)** Determining direction at a pixel **p**

Fig. 10. Determining direction for fLIC at a pixel **p**.

4.3 fLIC: LIC for features

To render features, we apply the integration formula (1) to the pixels near the vectorized feature lines. We assign an offset γ to each vectorized feature, and then estimate the LIC for every pixel that is closer to a feature line than the offset. At each pixel **p**, the direction of integration **d** is the tangent direction of **q**, which is the projection of **p** on to the feature line (see **Fig. 10**). In performing fLIC, pixels within the offset are tagged with the closest vectorized feature line and the distance. This information is used in merging tLIC and fLIC. The procedure for tLIC is as follows:

```
for each pixel p
    if p is within γ to a feature line f
       q ← projection of p onto f;
         dist(p) ← dist ( p, q );
         d(q) ← the direction of q along f;
         fLIC (p) ← Integrate ( p, d(q), L );
```

Fig. 11 (c) shows an fLIC image rendered using γ = 10.

4.4 Merging LICs

The two pencil rendering results, tLIC and fLIC, are merged by linear interpolation. Since the fLIC is only obtained from pixels within the offset γ from vectorized feature lines, pixels outside the offset γ have fLIC values of 0. The final value of a pixel **p** is obtained as follows:

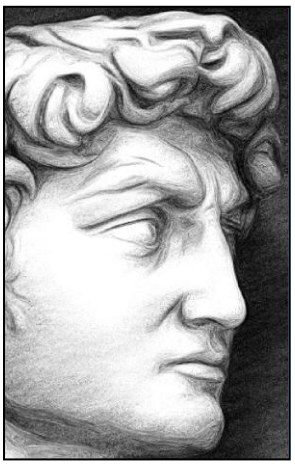
```
for each pixel p
    if dist(p) > γ
        LIC(p) = tLIC(p);
    else
        LIC(p) = (dist(p)/γ) * tLIC(p)
               + (1 - dist(p)/γ) * fLIC(p);
```

Fig. 11 (b) shows the value of (dist(**p**)/ γ), and **(d)** shows the result of merging the tLIC and fLIC results. The pseudo code for the overall process of pencil rendering is suggested in Appendix A.

(a) Result of tLIC

(b) (dist(**p**)/γ)

(c) Result of fLIC

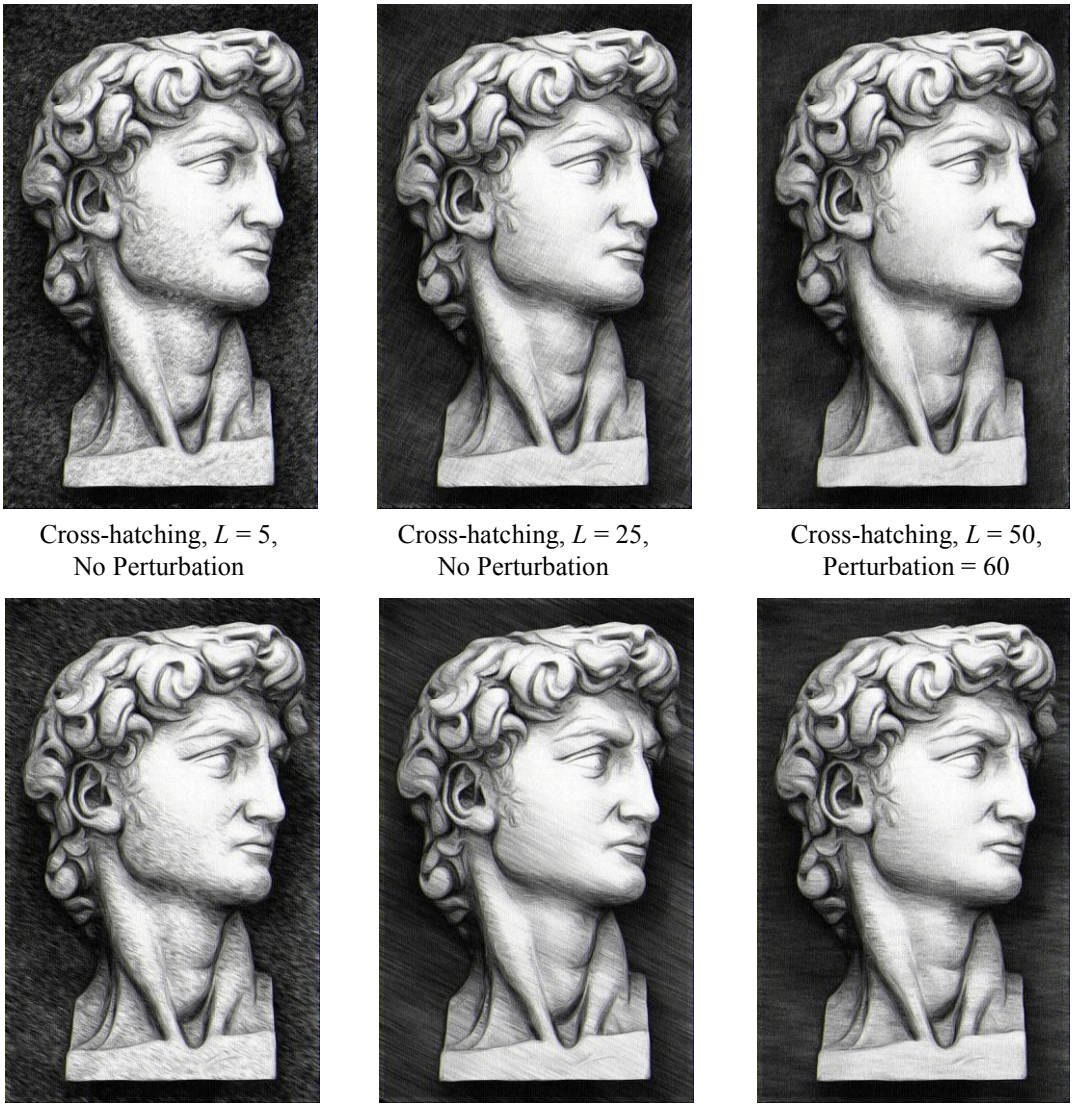


(d) Merged image of tLIC and fLIC

Fig. 11. Merging of tLIC and fLIC

5. Implementation and Results

Our algorithm was implemented on a PC with an Intel Pentium QuadCore™ Q6600 CPU and 4Gb of main memory. The programming environment was Microsoft Visual Studio™ 2008 with the OpenGL libraries. We modified the pencil rendering results using a paper texture effect [11]. We tested our pencil rendering algorithm with various combinations of parameters and results are shown in **Fig. 12**. We also re-rendered a range of photographic images, including human faces, animals, inanimate objects, landscapes, and animation characters. The resolutions, parameters and processing times for the test images are given in **Table 1**. Thumbnails of the test images are shown in **Fig. 13**, and the re-rendered are shown similarly in **Fig. 14**. Higher-resolution versions of these images can be found in the accompanying appendix.



Cross-hatching, $L = 5$, No Perturbation | Cross-hatching, $L = 25$, No Perturbation | Cross-hatching, $L = 50$, Perturbation = 60

Uni-directional hatching, $L = 5$, No Perturbation | Uni -directional hatching, $L = 25$, No Perturbation | Uni -directional hatching, $L = 50$, Perturbation = 35

Fig. 12. The effects of changing the integration parameters.

5.1 Comparison and discussion

We compare our scheme with existing LIC-based pencil rendering schemes in two points: direction of integration and various drawing styles. The LIC-based schemes produce pencil drawing effect on an image by applying LIC using the noise on the image along an integration direction. In most existing schemes, the image is segmented into several regions, where the LIC process is performed along the same direction. **Fig. 15** shows some images from existing schemes. Comparing them with our result (**Fig. 14**) shows the excellence of our algorithm.

Table 1. Resolution, parameters and computation times.

Image	Resolution (pixels)	Integration length (L)	Perturbation (δ)	Feature offset (γ)	Computation time (sec)
(a)	600 × 800	10	18	5	7.4
(b)	816 × 984	10	18	5	26.1
(c)	800 × 600	10	40	5	373.8
(d)	600 × 800	10	90	5	16.8
(e)	580 × 857	10	18	5	18.7
(f)	1200 × 606	10	18	5	37.8
(g)	900 × 800	10	90	5	26.1
(h)	1192 × 1193	10	18	5	136.2
(i)	1600 × 1200	10	18	5	117.7
(j)	1100 × 550	10	18	5	20.3
(k)	1024 × 685	10	6	5	18.7

Li and Huang's work [6] is devised from real pencil drawing process that divides a scene into regions, and applies pencil strokes along image moment and texture direction. However, they do not apply strokes to distinguish two neighboring regions, which is one of the key techniques in real pencil drawing art. The common point between [6] and ours is that individual pencil strokes are applied to the segmented regions. The differences are (i) we apply pencil strokes along feature lines to distinguish two neighboring regions, (ii) we apply pencil strokes of randomly uniform direction to each region, and (iii) we present an LIC scheme whose parameters are controlled to produce various pencil rendering effects. We compare our results and the results from [6] that render similar objects in **Fig. 16**. We execute a user test for comparison. The test questions are as follows:

(1) Which image is more similar to real pencil art drawings?
(2) Which image gives greater visually pleasing effect?

The answers from twenty subjects of different sexualities and ages are analyzed in **Table 2**.

Table 2. User test results.

Question	Results from [6]	Our result
(1)	1	19
(2)	2	18

Most LIC-based schemes take little effort in producing various pencil drawing styles. They didn't support various drawing styles by testing the parameters of their drawing schemes. In contrast, we have produced various pencil styles by perturbing the integration parameters such as integration range and integration direction. Our scheme cannot express all the pencil drawing styles. For example, miniature style, which is often used in portrait, is not supported

by our scheme. Another shortcoming is lack of user control in performing pencil rendering. Users may require freedom in selecting features to emphasize or to ignore. They may want to assign the hatching direction directly for a region. Our scheme doesn't support those user interactions.

Fig. 13. Test images

Fig. 14. Re-rendered images

(a) Result from [17]

(b) Result from [7]

(c) Result from [8]

(d) Result from [13]

Fig. 15. Results from existing LIC-based pencil rendering schemes.

(a) Results from [6]

(b) Our results

Fig. 16. Comparison of results from [6] and ours. Both results deal with similar objects: flowers and woman.

6. Conclusions and Future Plans

We have presented a novel pencil rendering algorithm in which using LICs which are guided by features. Our tLIC renders the tone of an image and our fLIC emphasizes its salient features. The fLIC is executed along vectorized features extracted as coherent lines. Our algorithm also supports various pencil drawing styles which can be achieved by controlling the style of tone and the width of lines. We have tested our algorithm on various photographs including portraits, landscapes, animals, inanimate objects, and animation characters.

Our algorithm requires further development, for example, it cannot replicate a miniature pencil drawing style. Another task is to develop colored pencil rendering schemes that can be used for portraits and botanical illustrations. We also aim to develop a pencil rendering scheme for real-time video re-rendering.

References

[1] M. C. Sousa and J. W. Buchanan, “Computer-generated graphite pencil rendering of 3D polygonal models,” in *Proc. of Eurographics 1999*, pp. 195-207, 1999. Article (CrossRef Link)

[2] M. C. Sousa and J. Buchanan, "Observational model of blenders and erasers in computer-generated pencil rendering," in *Proc. of Graphics Interface 1999*, pp. 157-166, 1999. Article (CrossRef Link)
[3] S. Takagi, I. Fujishiro, and M. Nakajima, "Volumetric modeling of colored pencil drawing," in *Proc. of Pacific Graphics 1999*, pp. 250-258 , 1999. Article (CrossRef Link)
[4] A. Lake, C. Marshall, M. Harris, and M. Blackstein, "Stylized rendering techniques for scalable real-time 3D animation," in *Proc. of NPAR 00*, pp.13-20, 2000. Article (CrossRef Link)
[5] X. Mao, Y. Nagasaka, and A. Imamiya, "Automatic generation of pencil drawing using LIC," in *ACM Siggraph 02 Abstractions and Applications*, pp. 149, 2002. Article (CrossRef Link)
[6] N. Li, and Z. Huang, "A feature-based pencil drawing method," in *Proc. of 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia 03*, pp. 135-140, 2003. Article (CrossRef Link)
[7] S. Yamamoto, X. Mao, and A. Imamiya, "Enhanced LIC pencil filter," in *Proc. of the International Conference on Computer Graphics, Imaging and Visualization 04*, pp. 251-256, 2004. Article (CrossRef Link)
[8] S. Yamamoto, X. Mao, and A. Imamiya, "Colored pencil filter with custom colors," in *Proc. of Pacific Graphics 04*, pp. 329-338, 2004. Article (CrossRef Link)
[9] H. Matsui, J. Johan, and T. Nishita, "Creating colored pencil images by drawing strokes based on boundaries of regions," in *Proc. of Computer Graphics International 05*, pp. 148-155, 2005. Article (CrossRef Link)
[10] K. Murakami, R. Tsuruno, and E. Genda, "Multiple illuminated paper textures for drawing strokes," in *Proc. of Computer Graphics International 05*, pp. 156-161, 2005. Article (CrossRef Link)
[11] H. Lee, S. Kwon, and S. Lee, "Real-time pencil rendering," in *Proc. of NPAR 06*, pp. 37-45, 2006. Article (CrossRef Link)
[12] K. Melikhov, F. Tian, X. Xie, and H. S. Seah, "DBSC-based pencil style simulation for line drawings," in *Proc. of 2006 International Conference on Game Research and Development*, pp. 17-24, 2006. Article (CrossRef Link)
[13] D. Xie, Y. Zhao, D. Xu, and X. Yang, "Convolution filter based pencil drawing and its implementation on GPU," *Lecture Notes in Computer Science*, vol. 4847, pp. 723-732, 2007. Article (CrossRef Link)
[14] Z. Chen, J. Zhou, X. Gao, L. Li and J. Liu, "A novel method for pencil drawing generation in non-photo-realistic rendering," *Lecture Notes in Computer Science*, vol. 5353, pp. 931-934, 2008. Article (CrossRef Link)
[15] Y. Kim, J. Yu, H. Yu, and S. Lee, "Line-art illustration of dynamic and specular surfaces," *ACM Trans. on Graphics*, vol. 27, no. 5, 2008. Article (CrossRef Link)
[16] Z. AlMeraj, B. Wyvill, T. Isenberg, A. Gooch, and G. Richard, "Automatically mimicking unique hand-drawn pencil lines," *Computers & Graphics*, vol. 33, no. 4, pp. 496-508, 2009. Article (CrossRef Link)
[17] M. Salisbury, S. Anderson, R. Barzel, and D. Salesin, "Interactive pen-and-ink illustration," in *Proc. of Siggraph 94*, pp. 101-108, 1994. Article (CrossRef Link)
[18] G. Winkenbach and D. Salesin, "Computer generated pen-and-ink illustration," in *Proc. of Siggraph 94*, pp. 91-100, 1994. Article (CrossRef Link)
[19] M. Salisbury, C. Anderson, D. Lischinski, and D. Salesin, "Scale-dependent reproduction of pen-and-ink illustrations," in *Proc. of Siggraph 96*, pp. 461-468, 1996. Article (CrossRef Link)
[20] M. Salisbury, M. Wong, J. Hughes, and D. Salesin, "Orientable textures for image-based pen-and-ink illustration," in *Proc. of Siggraph 97*, pp. 401-406, 1997. Article (CrossRef Link)
[21] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proc. of Siggraph 00*, pp. 517-526, 2000. Article (CrossRef Link)
[22] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-time hatching," in *Proc. of Siggraph 01*, pp. 579-584, 2001. Article (CrossRef Link)
[23] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe, "Fine tone control in hardware hatching," in *Proc. of NPAR 02*, pp. 53-58, 2002. Article (CrossRef Link)

[24] J. Bu, W. Yan, C. Chen, and M. Song, "Image-based real-time hatching of scene traveling," in *Proc. of WSCG*, vol. 14, no. 1-3, pp. 241-248, 2006. Article (CrossRef Link)
[25] A. Paiva, E. Brazil, F. Petronetto, and M. Sousa, "Fluid-based hatching for tone mapping in line illustrations," *The Visual Computer*, vol. 25, no. 5-7, pp. 519-527, 2009. Article (CrossRef Link)
[26] H. Yang and K. Min, "Texture-based Hatching for Color Images and Video," *KSII Transactions on Internet and Information Systems*, vol. 5, no. 4, pp. 763-781, 2011. Article (CrossRef Link)
[27] P. Haeberli, "Paint by numbers: abstract image representations," in *Proc. of Siggraph 90*, pp. 207-214, 1990. Article (CrossRef Link)
[28] B. Meier, "Painterly rendering for animation," in *Proc. of Siggraph 96*, pp. 477-484, 1996. Article (CrossRef Link)
[29] P. Litwinowicz, "Processing images and video for an impressionist effect," in *Proc. of Siggraph 97*, pp. 406-414, 1997. Article (CrossRef Link)
[30] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proc. of Siggraph 98*, pp. 453-460, 1998. Article (CrossRef Link)
[31] J. Hays and I. Essa, "Image and video based painterly animation," in *Proc. of NPAR 04*, pp. 113-120, 2004. Article (CrossRef Link)
[32] K. Zeng, M. Zhao, C. Xiong, and S. C. Zhu, "From image parsing to painterly rendering," ACM *Trans. on Graphics*, vol. 29, no. 1, 2009. Article (CrossRef Link)
[33] H. Kang, S. Lee, and C. Chui, "Flow-based image abstraction," *IEEE Trans. on Visualization and Computer Graphics*, vol. 15, no. 1, pp. 62-76, 2009. Article (CrossRef Link)

Appendix A

The overall process of pencil rendering

```
Input: a photograph
Output: pencil rendered image
//  step 1. Noise generation
    noise ← generate noise by pseudocode (A);
//  step 2. Vectorizing feature lines
    feature lines ← vectorize feature lines by the algorithm in 3.2;
//  step 3. fLIC
    fLIC ← feature LIC using noise and feature lines by pseudocode (B);
//  step 4. tLIC
    tLIC ← tone LIC using noise and feature lines by pseudocode (C);
//  step 5. Merging
    LIC ← merging fLIC & tLIC by pseudocode (D);
    return LIC as the result of pencil rendered image;
```

Heekyung Yang received her B.S. degree in Digital Media from Sangmyung University, Seoul, Korea, in 2010. She is currently a M.S. student in the same college. Her major is computer graphics, especially NPR(non-photorealistic rendering). Also she is interested in image processing, 3D-mesh processing, volume rendering and medical rendering.

Kyungha Min received his BS in Computer Science from KAIST in 1992. He received his MS and Ph.D in Computer Science and Engineering from POSTECH in 1994 and 2000, respectively. His main research interests are computer graphics and image processing.