Texture-based Hatching for Color Image and Video

Heekyung Yang[1] and Kyungha Min[2]
[1] Dept. of Computer Science, Graduate School, Sangmyung Univ.,
Hongji-dong, Jongro-gu, Seoul, Korea
[e-mail: yhk775206@naver.com]
[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 December 28, 2010; revised March 17, 2011; revised April 13, 2011; accepted April 8, 2011; published April 29, 2011

Abstract

We present a texture-based hatching technique for color images and video. Whereas existing approaches produce monochrome hatching effects in considering of triangular mesh models by applying strokes of uniform size, our scheme produces color hatching effects from photographs and video using strokes with a range of sizes. We use a Delaunay triangulation to create a mesh of triangles with sizes that reflect the structure of an input image. At each vertex of this triangulation, the flow of the image is analyzed and a hatching texture is then created with the same alignment, based on real pencil strokes. This texture is given a modified version of a color sampled from the image, and then it is used to fill all the triangles adjoining the vertex. The three hatching textures that accumulate in each triangle are averaged and the result of this process across all the triangles forms the output image. We can also add a paper texture effect and enhance feature lines in the image. Our algorithm can also be applied to video. The results are visually pleasing hatching effects similar to those seen in color pencil drawings and oil paintings.

Keywords: Non-photorealistic rendering, hatching, pencil drawing, video, optical flow

A preliminary version of this paper appeared in ISVC 2010, Las Vegas, USA. This version includes paper texture effects, enhanced feature lines and video processing.
This study was supported by a research grant from Sangmyung University in 2009..

DOI: 10.3837/tiis.2011.04.008

1. Introduction

Hatching is an artistic technique that creates tonal or shading effects by drawing closely spaced parallel lines. Hatching effects are important in many graphic arts, such as line illustration, pencil drawing, and oil painting. Thus the creation of effective hatching effects from a 3D mesh or on image is an important research topic in non-photorealistic rendering (NPR). Researchers have developed a wide range of hatching technique to render 3D triangular meshes or to re-render photographs using brush strokes or line segments [1][2][3][4]. Hatching patterns can be also found in research on line illustration [5][6][7] pencil drawing [8][9][10][11][12][13][14][15] and oil painting [16][17][18][19][20][21]. Even color pencil drawings [12][13][14] and oil-paintings [20][21] can exhibit some hatching patterns.

In this paper, we present a texture-based hatching technique for re-rendering color images and videos. Our scheme is inspired by that of Lee et al. [4] who showed how to create monochrome pencil drawings from triangular meshes. We have extended their work to the re-rendering of color images. Our scheme produces a color image that contains hatching patterns similar to those found in a color pencil drawing or an oil painting. Our basic strategy is to build a color hatching texture and to apply it at each vertex of a triangular mesh in which the input image is embedded. This mesh is constructed by the Delaunay triangulation of adaptively sampled points in the image. The color hatching textures are constructed from a base hatching texture created from real strokes made by an artist. During the hatching step, we build a texture with a color selected from the input image at each vertex of the triangular mesh. We modify these sampled colors to emulate the color found in artist' drawings. The regions inside the triangles which meet at each vertex are drawn using these textures. The directions of the textures are selected with the aim of improving the understanding of the shapes in the image. Afterward, we apply paper texture effects and enhance feature lines in the hatched image. Our algorithm can also be used to create hatching effects in video. An overview of the scheme is shown in **Fig. 1**.

1.1 Related Work

We will now survey some of the more significant related work on color pencil drawing and oil painting that includes hatching patterns. Techniques for color pencil rendering can generate hatching effects using line integral convolution (LIC) [12], or strokes along contours [13], and paper textures may be also simulated [14]. Yamamoto et al. [12] segmented regions into several regions and simulated the overlapping effect of two different color pencils and created hatching patterns of uniform direction inside each region. They then used LIC to create hatching patterns, which are overlapped using the Kubelka-Munk model. However, since all the regions in the image are re-rendered using only pair of colors, the results are not particularly pleasing. Matsui et al. [13] developed a scheme that re-renders a color photograph as a color pencil drawing by extracting the boundaries of regions and then generating strokes along the boundary curves. The colors of the strokes are sampled from the image and again mixed using the Kubelka-Munk model. Murakami et al. [14] presented an algorithm that generates strokes to mimic pastels, charcoals or crayons using multiply illuminated paper textures. They captured the texture of paper and simulated strokes in various ways. They can achieve some realistic color drawings, but not the effects of color pencils with sharp tips.

Some of the research on painterly rendering [20][21] has involved hatching. Hays and Essa [20] presented a painterly rendering scheme in which strokes are drawn in directions estimated

using a radial basis function (RBF). Hatching patterns are created by capturing the stroke textures produced by artists' brushes. However, the results of this work have a limited ability to convey shape information, since the stroke-generation process does not relay any of the shape information from this original image. Zeng et al. [21] presented a painterly rendering scheme in which an image is analyzed to determine the size and the type of brushes to use. Our scheme achieves similar results for pencil drawing, although we use a 2D triangular mesh, instead of the image analysis process.

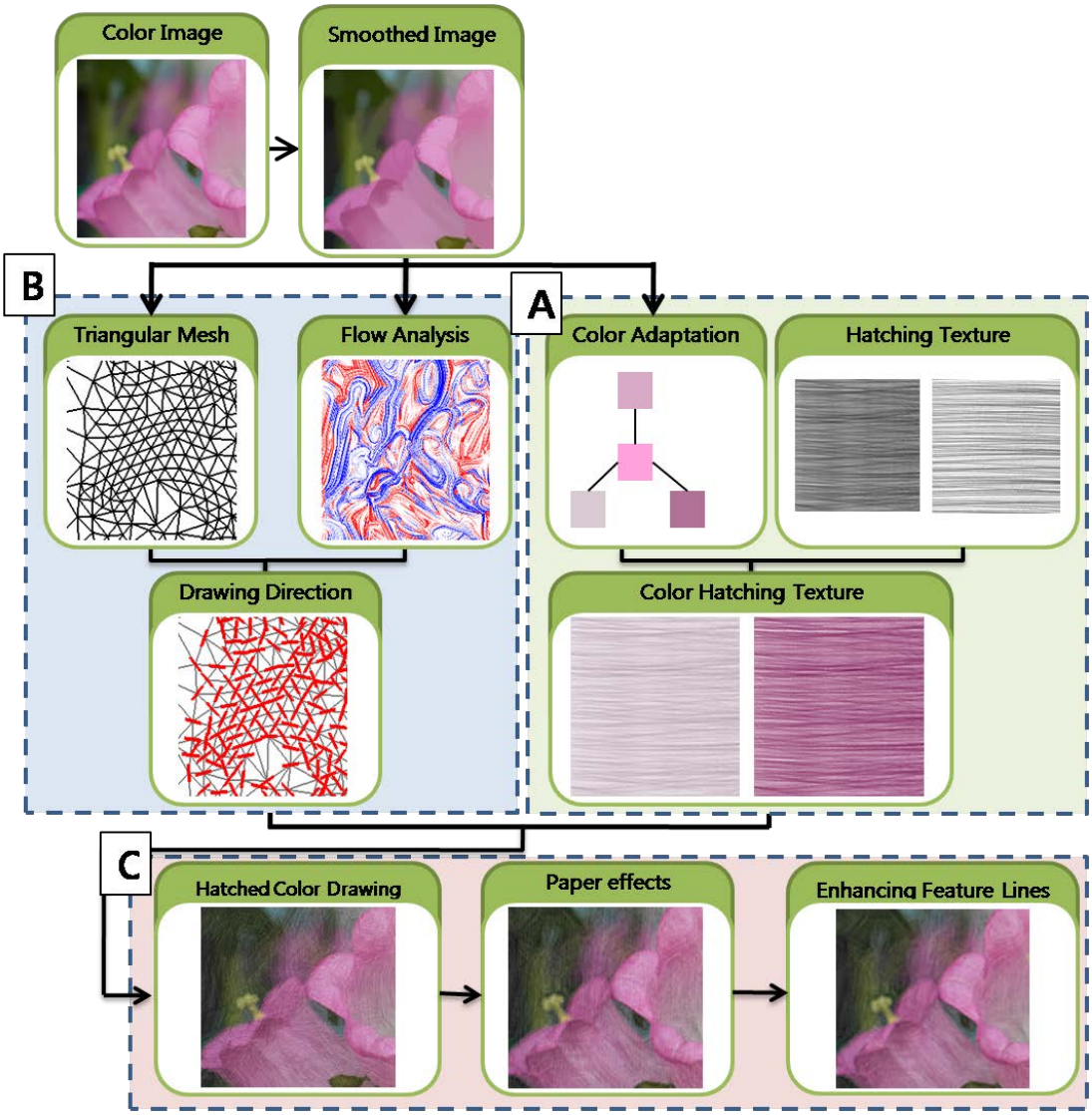

Fig. 1. An overview of our hatching algorithm. The contents of the dotted boxes A, B and C are explained in Sections 2, 3, and 4 respectively.

1.2 Contributions

Our approach offers several advantages: First, the use of different hatching patterns can produce a range of effects, suggesting media as diverse as color pencils and oil painting.

Second, embedding an input image in a triangular mesh allows us to control the variation in scale of the hatching patterns across the image to conform to the shapes of the objects that it depicts. Less important regions, such as the backgrounds are embedded in large triangles while more important regions are segmented into small triangles; then we can naturally apply coarse hatching patterns to the less important regions and fine patterns to more important regions. Third, we are able to improve the results of hatching by modifying the colors that we extract from the image. For example, reducing the saturation of the sampled colors suggests a pencil drawing, whereas increasing the saturation suggests oil painting. Finally, we have also applied our algorithm to create hatching effects on video. In this medium we can emphasize the hatching effect by drawing additional hatching lines that depict motions.

The rest of this paper is organized as follows: In Section 2 we explain how sampled colors are modified and how colored hatching textures are created with the sampled colors. In Section 3 we describe how a color hatching texture is applied to an input image embedded in a triangular mesh. We add a paper texture effect and feature line enhancement in Section 4. In Section 5 we explain how our scheme is implemented and present several results. Finally, we conclude our paper and suggest some directions for future research in Section 6.

2. Generating Color Hatching Textures

A color hatching texture is generated by rendering a base texture, which we build by capturing and overlapping real strokes with a target color [4]. We define the *tone* t^0 of a hatching texture to be the average of all the intensities in the texture, and its *range* (t^0_m , t^0_M) span the maximum and minimum intensity across the texture. **Fig. 2-(a)** illustrates a base texture with its tone and range.

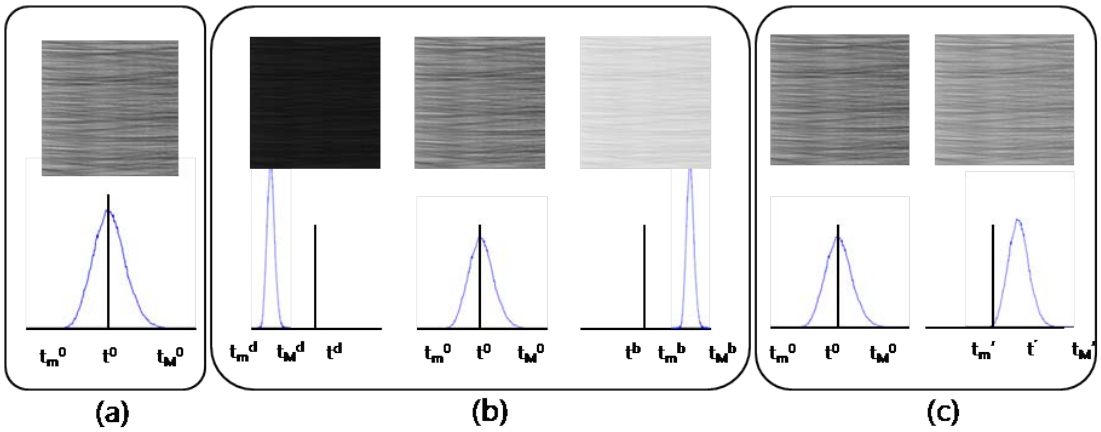

Fig. 2. Using a histogram to control the tone of a hatching texture: **(a)** base texture; **(b)** darkest and brightest textures; **(c)** texture with specified tone t' and range (t_m', t_M').

2.1 Color Modification

Different artistic media tend to be associated with pictures containing colors that a particular relationship with those in the scene being rendered. To emulate this relationship, we modify the color extracted from the input image before we build a color hatching texture. Let C_p be the color sampled at a pixel p of the input image in RGB format with components in the range (0, 1). We convert C_p to HSV format (H_p, S_p, V_p). We then modify C_p by changing S_p and V_p. We can set the modified color (H_p', S_p', V_p') to nine different variations on C_p, selected from the

combinations expressed by { $S_p + \delta$, S_p, $S_p - \delta$ } x { $V_p + \delta$, V_p, $V_p - \delta$ }, as shown in **Fig. 3-(a)**. Out of these, we use (H_p, $S_p - \delta$, V_p) to achieve pencil effects, and (H_p, $S_p + \delta$, $V_p + \delta$) to simulate oils. The extent δ of any change may be in the range (0.1, 0.3). Empirically, we have chosen to set δ to 0.3. Note that the modified color values are clamped to keep them in the range (0, 1). The RGB format of each modified color C_p' is reconstructed from (H_p', S_p', V_p').

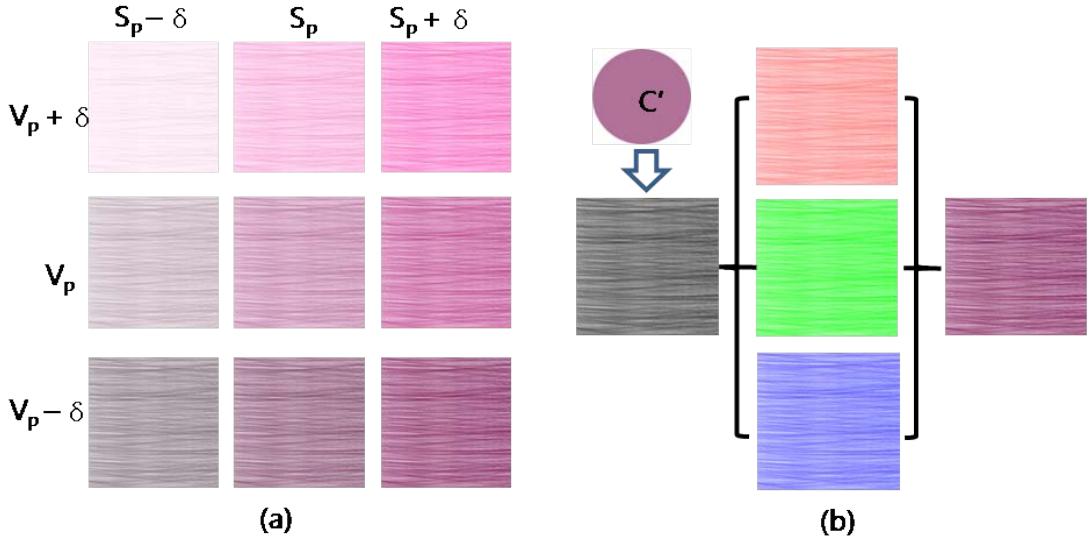

Fig. 3. (a) Eight modified colors: the center color is the original. **(b)** A color C' applied to a monochrome hatching texture: the corresponding color hatching texture is composed from three color components with different textures.

2.2 Color Hatching Textures

We build three individual monochrome hatching textures for R_p' , G_p' and B_p', and then merge them into a color hatching texture as shown in **Fig. 3-(b)**. If we assume R_p' = t', then our target texture has a tone of t ' and a range of (t_m', t_M'), where the subscripts m and M respectively denote the minimum and the maximum values in the histogram. Whereas previous techniques [2][4][22][23] build a series of textures by overlapping strokes or textures, we use a histogram-based approach to create textures of different tones. The darkest and brightest hatching textures respectively have tones of t^d and t^b, and ranges ($t_m{}^d$, $t_M{}^d$) and ($t_m{}^b$, $t_M{}^b$), as shown in **Fig. 2-(b)**. A texture of the required tone and range can be constructed by manipulating its histogram, although the range of a texture is reduced if it becomes very dark or very bright. The new range (t_m', t_M') is estimated from the tone and the original range. Then an intensity $t_i{}^0 \in (t_m{}^0, t_M{}^0)$, sampled from the base texture, is converted to $t_i' \in (t_m', t_M')$ to match the intensity of the target texture. If $t' > t_0$, then t_m' and t_M' are determined as follows:

$$t_M' = t_M^0 + (t_M^b - t_M^0)\frac{t'-t^0}{t^b-t^0}, \qquad t_m' = t_m^0 + (t_m^b - t_m^0)\frac{t'-t^d}{t^0-t^d}. \tag{1}$$

But if $t' < t_0$, then t_m' and t_M' are determined as follows:

$$t_M' = t_M^d + (t_M^0 - t_M^d)\frac{t'-t^d}{t^0-t^d}, \qquad t_m' = t_m^d + (t_m^0 - t_m^d)\frac{t'-t^d}{t^0-t^d}. \tag{2}$$

Having obtained t_m' and t_M', other intensities $t_i^0 > t^0$ or $t_j^0 < t^0$ of a base texture can be converted to t_i' and t_j' in the target hatching texture using the following formulas:

$$t_i' = t' + (t_i - t_0)\frac{t_M'-t'}{t_M^0-t^0}, \qquad t_j' = t^0 + (t^0 - t_j)\frac{t'-t_m'}{t^0-t_m^0}. \tag{3}$$

We illustrate the relationships between tones and histograms in **Fig. 3-(c)**.

3. Drawing Color Hatching Textures

Before re-rendering the input image using color hatching textures, we smooth the input image using the mean-shift scheme [24].

3.1 Adaptive Delaunay Triangulation

The first step in drawing the color textures is to embed the input image into a 2D triangular mesh, which requires the following steps:

Step 1. We sample n_0 points on the image using a Poisson disk distribution, and then use Delaunay triangulation to build an initial triangular mesh. We choose n_0 to suit the size of an image. If n_0 is too large the roughness of the hatching effect is lost, whereas a small value that is too small causes excessive triangulation. Since there is no guidance for determining the value of n_0 in the literature, we tried various values and found out that n_0 = 100 produces good results for an image containing about 500K pixels. Thus, we increase n_0 by 1 for every 5K additional pixels in an image. The formula for determining n_0 from the image size is:

$$n_0 = \begin{cases} 10, & \text{if } s(I) < 50K \\ s(I)/5K, & \text{othrewise} \end{cases}, \tag{4}$$

where, s(I) denotes the size of an image in Kbytes. For images with fewer than 50K pixels, we set n_0 to a minimum value of 10; otherwise the triangulation process does not operate properly.

Step 2. Then we generate a new point at the center of each triangle that contains at least one important point. We use the difference of Gaussian (DoG) filter [25] to find 'important' pixels, which are then with DoG values greater than a threshold.

Step 3. We apply Delaunay triangulation to the modified set of points.

Step 4. We repeat steps **2** and **3** until the image is triangulated appropriately, so that the triangles are not so small that they compromise the hatching effects, and not so large that they ignore necessary shape information. In practice, the triangulation only has to be performed three or four times.

The final result of this procedure is a triangular mesh which reflects the structure of the input image. **Fig. 4-(a)** shows three stages in the triangulation of a test image, and **Fig. 4-(b)** shows how the result is affected by the triangulation.

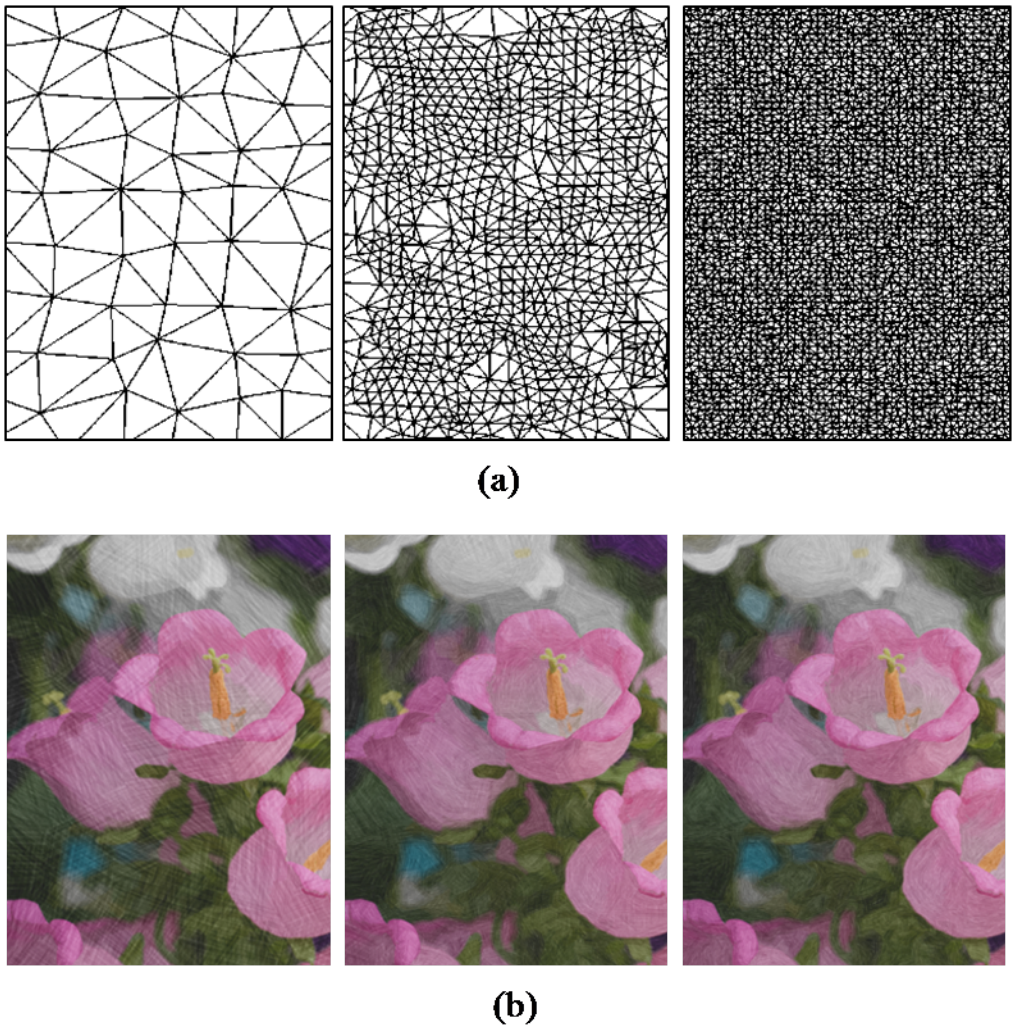

Fig. 4. **(a)** Triangulation of images and **(b)** the corresponding hatching results. From left to right is a coarse uniform mesh, a mesh modified to reflect the structure of the image, and a fine uniform mesh.

3.2 Estimating Drawing Directions

The directions in which the hatching textures are drawn are determined at the vertices of the embedding triangular domain by performing an edge tangent flow (ETF) algorithm [25] at the pixel in which each vertex is located. This produces similar drawing directions to those seen in real pencil drawings, which usually have one of three characteristics: (i) they follow contours or feature lines; (ii) they are drawn in locally uniform directions; or (iii) they have random directions. The directions at important vertices have characteristic (i), since the ETF's computed at the matching pixels follow contours or feature lines and the vertices are close

together. Since the unimportant vertices are much further apart, their ETF's correspond to a locally uniform flow, producing characteristic (ii).

3.3 Drawing Textures on a 2D Triangular Mesh

Here we follow Lee at al. [4]. At each vertex v, we find the pixel p that contains v. The color at p, which is C_p, is modified to C_p' using the scheme described in Section 2. C_p' is then used in generating a color hatching texture. The pixels inside triangles which have v as one of their vertices are filled with this hatching texture. Thus, every pixel inside a triangle receives three colors from the three different textures generated from each of its vertices. These colors from the three textures are averaged at each pixel. **Fig. 5** shows this process.

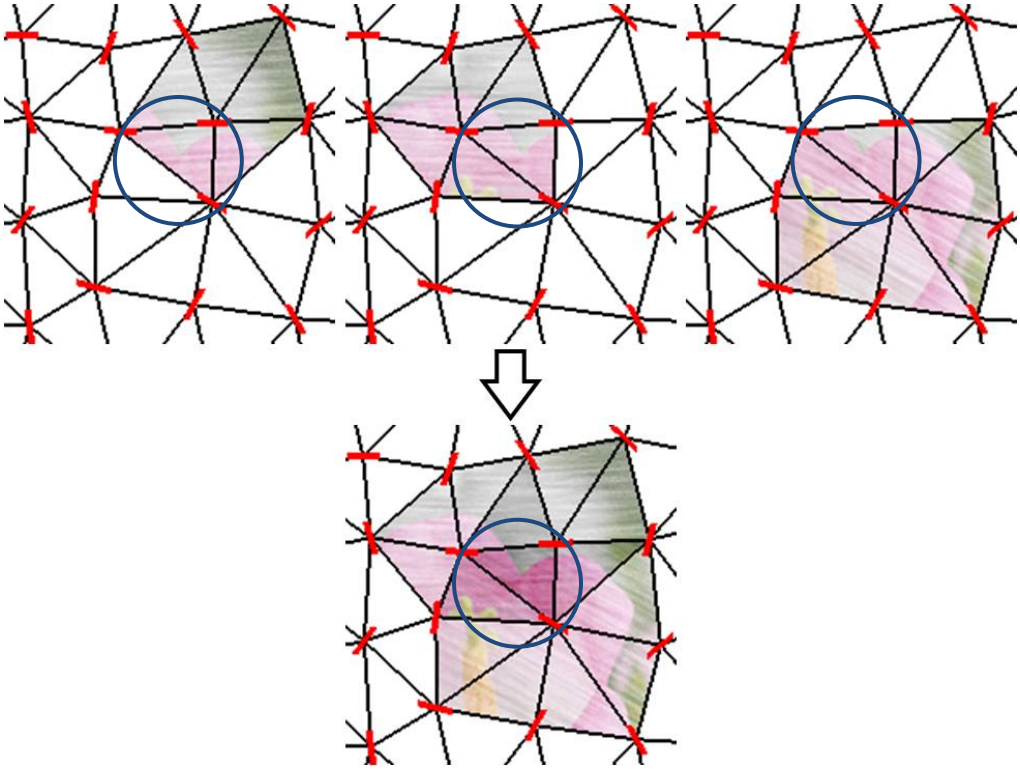

Fig. 5. Merging the three textures inside a triangle (inside the blue circle).

4. Postprocessing

We apply two postprocesses to the hatching results: a paper texture effect and feature line enhancement.

4.1 Paper texture effect

Many researchers have represented paper texture as a height map $(x, y, h(x, y))$, where $h(x, y)$ denotes the height at position (x, y). Lee et al. [4] modeled paper texture as a normal map $(x, y, n(x, y))$, where $n(x, y)$ is the normal at position (x, y). The intensity of a stroke is modified by taking the dot product between its direction and the normal from the map, and this creates a paper texture effect. We implement the paper texture effect by generating pseudo paper texture using a random function. The technique of Lee et al. [4] was based on 3D meshes, whereas our

drawing direction is always orthogonal to the *z*-axis, since we start with a 2D image. Therefore, the dot products between the paper normals and the drawing direction can be computed. The color $c(x, y)$ of each pixel is changed to $c_p(x, y)$ using the following formula:

$$c_p(x, y) = c(x, y) + \mu_p p(x, y), \tag{5}$$

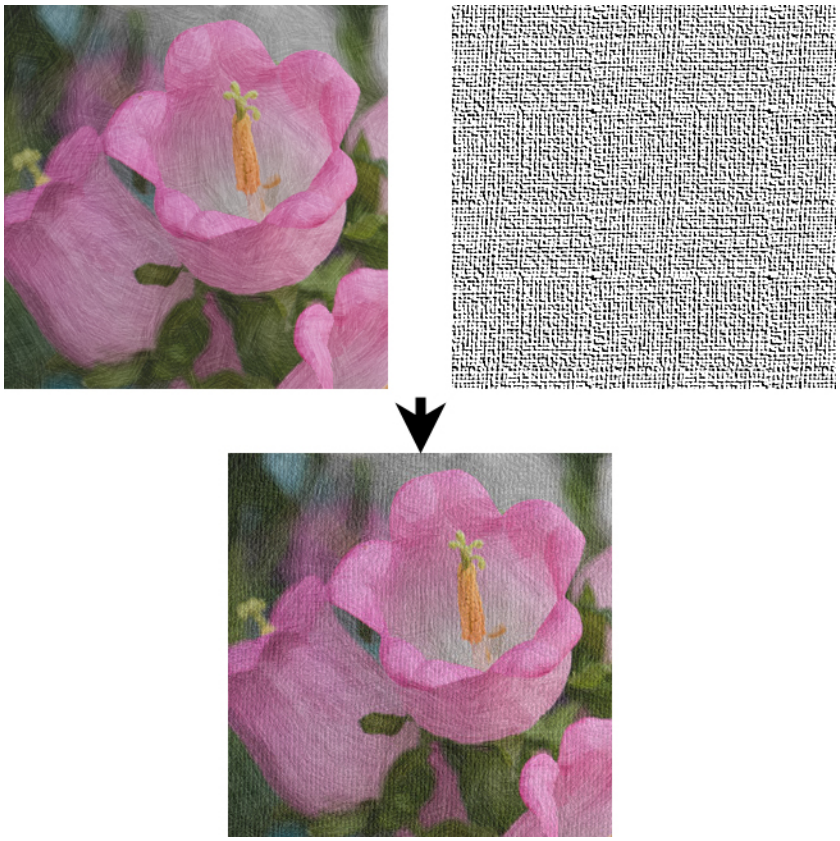

Fig. 6. Applying a paper texture effect to a hatched color rendering.

where μ_p is a weighting factor in the range [0, 0.1] and $p(x, y)$ is a random value in the range [-1, 1] that provides the paper texture. The result of applying paper texture effect is shown in **Fig. 6**.

4.2 Feature line enhancement

Some artists draw feature lines to convey important shape information using thicker pencil strokes. We mimic this technique by extracting feature lines [25] and then enhancing them. We obtain the ETF by distributing tangent vectors evenly across an image. Then, we generate coherent lines by applying DoG filter to the ETF. The pixels on these lines are emphasized by modifying their colors as follows:

$$c_f(x, y) = c_p(x, y)(1 - \mu_p i(x, y)), \tag{6}$$

where $i(x, y)$ is 1 if (x, y) is on the feature line, and 0 otherwise; and μ_p is a weighting factor in the range [0.6, 0.8]. The result of line enhancement is shown in **Fig. 7**.

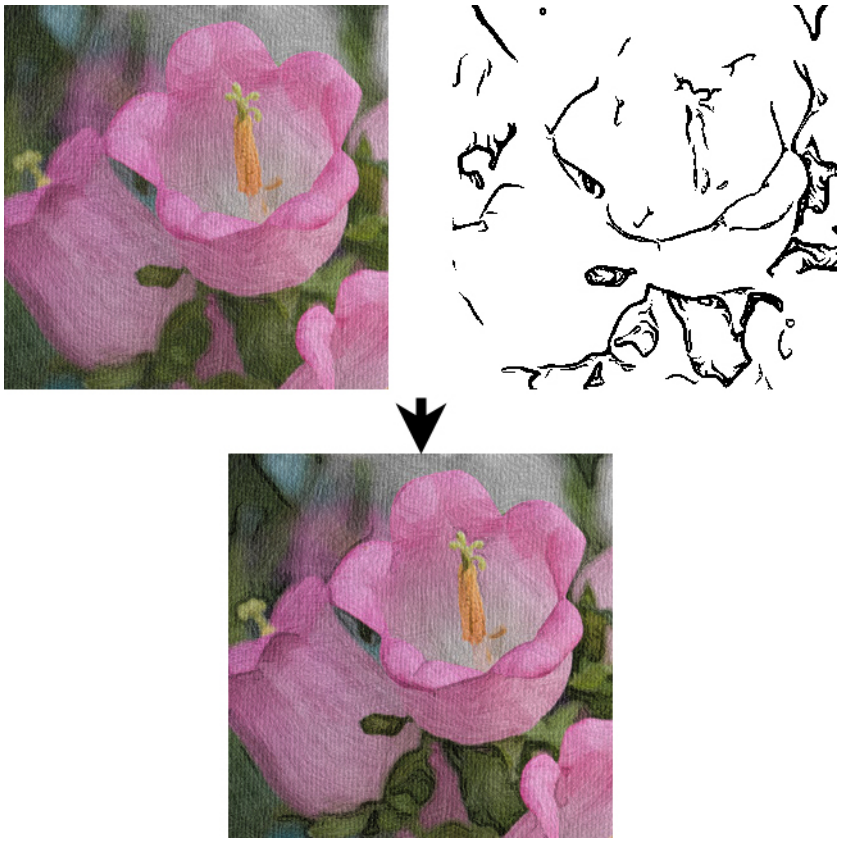

Fig. 7. Applying feature line enhancement to hatched color rendering.

5. Implementation and Results

We implemented our algorithm on a PC with an Intel Pentium QuadCore™ Q6600 CPU and 4G bytes of main memory. The programming environment was Visual Studio 2008 with the OpenGL libraries. We selected five photographs: a child, flowers, an animal and a landscape. Each of these images was re-rendered using our scheme, with the level of triangulation set to 3. The original photos are shown in **Fig. 8** and the hatched images in **Fig. 9** and **10**. The resolutions of the images and the associated computation times are given in Table 1. An attribution of the computation to the components of system is provided in Table 2. We conclude that the bottlenecks in our process are the analysis of flow and the color hatching process.

We have tested our algorithm on video, using clips from a movie and a cel animation. The accompanying clips shown both the original and the hatching versions of these videos. In addition, **Fig. 14** and **15** shows some frames from the hatched video.

6. Conclusions and Future Plans

We have presented a texture-based hatching technique for re-rendering a color image in order to give the impression of a color pencil drawing. The input image is embedded into an adaptive 2D triangular mesh. Color hatching textures, created by applying modified colors from the image to a monochrome hatching texture, are then drawn into the triangles of the mesh.

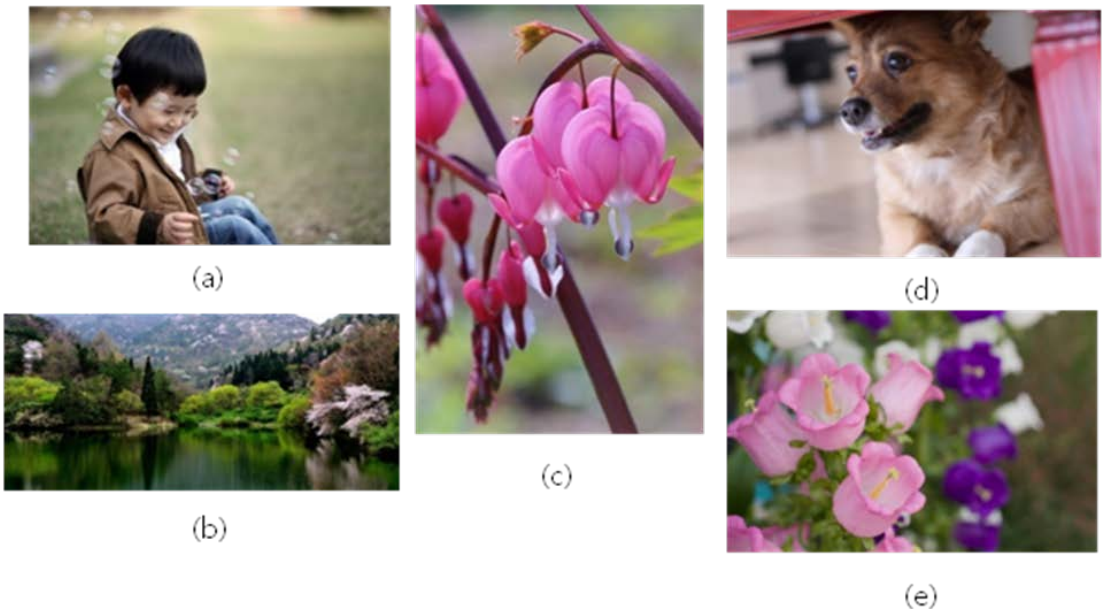

Fig. 8. Original images.

Table 1. Resolution and computation times.

Image	Resolution	Computation time	Result image
(a)	800x530	16.8 sec	**Fig. 9**
(b)	1000x456	15.1 sec	**Fig. 10**
(c)	624x988	23.9 sec	**Fig. 11**
(d)	800x530	13.8 sec	**Fig. 12**
(e)	800x530	15.6 sec	**Fig. 13**

Table 2. Attribution of computation times to individual processes.

Image	Generating triangular mesh	Analyzing flow	Performing color hatching	Post-processing	Total
(a)	4.2	5.9	5.0	1.7	16.8 sec
(b)	3.8	5.3	4.5	1.5	15.1 sec
(c)	6.0	8.4	7.2	2.4	23.9 sec
(d)	3.5	4.8	4.1	1.4	13.8 sec
(e)	3.9	5.5	4.7	1.6	15.6 sec

Our results have the flavor of an artistic rendering, but it is not possible to produce clearly identifiable pencil drawing or oil painting effects by controlling the parameters currently available. We plan to solve this problem by extending the range of the strokes used to approximate those made by artists' brushes and watercolor pencils more clearly.

We have applied our technique to video, but not yet considered temporal coherence. We plan to include this using the optical flow algorithm [26].

References

[1] 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)

[2] E. Praun, H. Hoppe, M. Webb and A. Finkelstein, "Real-time hatching," in *Proc. of Siggraph 01*, pp. 579-584, 2001. Article (CrossRef Link)

[3] 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)

[4] H. Lee, S. Kwon and S. Lee, "Real-time pencil rendering," in *Proc. of NPAR 06*, pp. 37-45, 2006. Article (CrossRef Link)

[5] 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)

[6] 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)

[7] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," in *Proc. of Siggraph 00*, pp. 517-526, 2000. Article (CrossRef Link)

[8] B. Cabral and C. Leedom, "Imaging vector field using line integral convolution," in *Proc. of Siggraph 93*, pp. 263-270, 1993. Article (CrossRef Link)

[9] 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)

[10] N. Li and Z. Huang, "A feature-based pencil drawing method," in *1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia 03*, pp. 135-140, 2003. Article (CrossRef Link)

[11] 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)

[12] 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)

[13] H. Matsui, J. Johan and T. Nishita, "Creating colored pencil style images by drawing strokes based on boundaries of regions," in *Proc. of Computer Graphics International 05*, pp. 148-155, 2005. Article (CrossRef Link)

[14] 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)

[15] 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)

[16] P. Haeberli, "Paint by numbers: Abstract image representations," in *Proc. of Siggraph 90*, pp. 207-214, 1990. Article (CrossRef Link)

[17] B. Meier, "Painterly rendering for animation," in *Proc. of Siggraph 96*, pp. 477-484, 1996. Article (CrossRef Link)

[18] P. Litwinowicz, "Processing images and video for an impressionist effect," in *Proc. of Siggraph 97*, pp. 406-414, 1997. Article (CrossRef Link)

[19] A. Hertzmann, "Painterly rendering with curved brush strokes of multiple sizes," in *Proc. of Siggraph 98*, pp. 453-460. Article (CrossRef Link)

[20] J. Hays and I. Essa, "Image and video based painterly animation," in *Proc. of NPAR 04*, pp. 113-120, 2004. Article (CrossRef Link)
[21] K. Zeng, M. Zhao, C. Xiong and S. C. Zhu, "From image parsing to painterly rendering," *ACM Trans. on Graphics*, vol. 29, no. 2, 2009. Article (CrossRef Link)
[22] G. Winkenbach and D. Salesin, "Computer generated pen-and-ink illustration," in *Proc. of Siggraph 94*, pp. 91-100, 1994. Article (CrossRef Link)
[23] 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)
[24] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603-619, 2002. Article (CrossRef Link)
[25] 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)
[26] M. J. Black and P. Anandan, "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields," *Computer Vision and Image Understanding*, vol. 63, no. 1, pp. 75-104, 1996. Article (CrossRef Link)

Fig. 9. Result of hatching of **Fig. 8-(a)**.

Fig. 10. Result of hatching of **Fig. 8-(b)**.

Fig. 11. Result of hatching of **Fig. 8-(c)**.

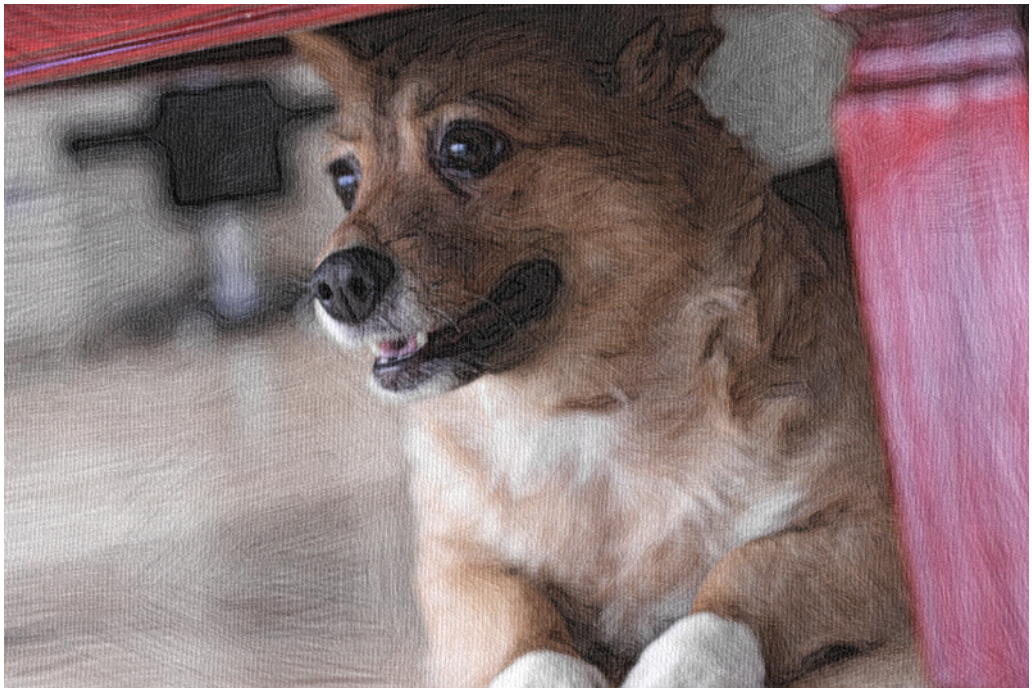

Fig. 12. Result of hatching of **Fig. 8-(d)**.

Fig. 13. Result of hatching of **Fig. 8-(e)**.

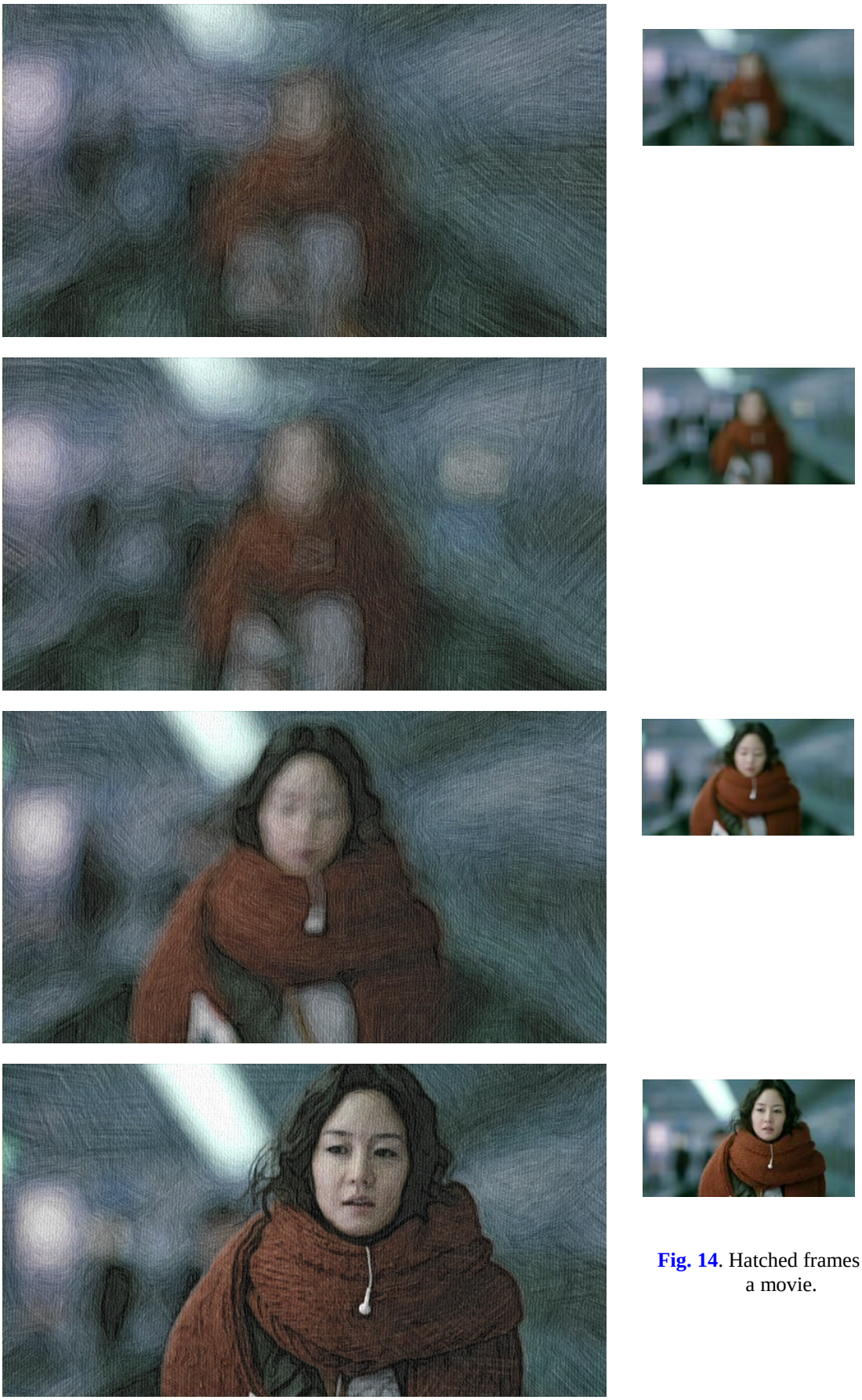

Fig. 14. Hatched frames from a movie.

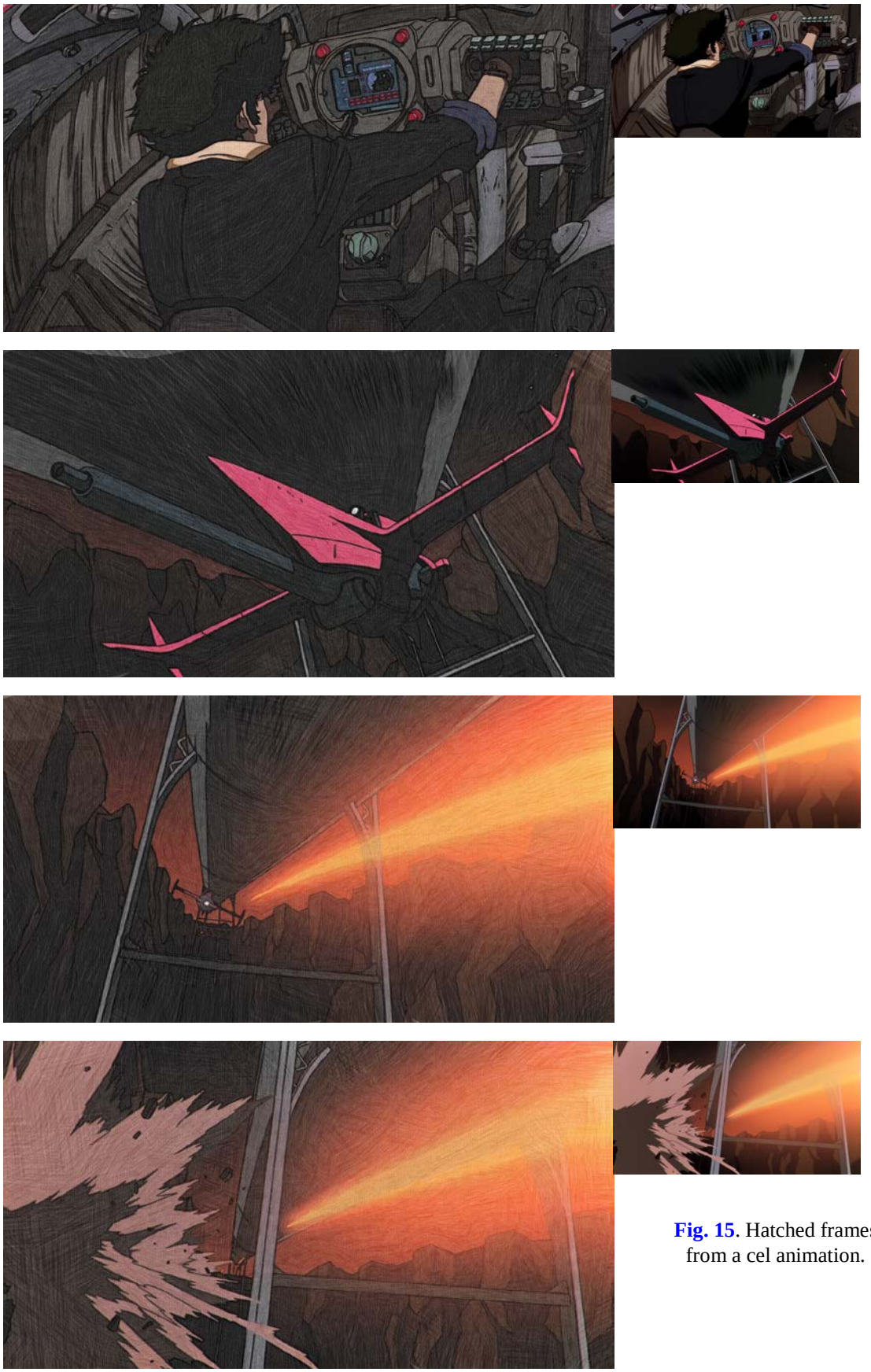

Fig. 15. Hatched frames from a cel animation.

Heekyung Yang received her B.S. degree 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 MS in Computer Science from KAIST in 1992. He received his BS 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.