ALGORITHM FOR AUTOMATIC GENERATION OF INK AND CHALK ILLUSTRATIONS

Ovidiu COSMA
Technical University of Cluj-Napoca, North University Center Baia Mare, Roumania
ovidiu.cosma@yahoo.com

Keywords: computer generated illustrations, ink and chalk hatching

Abstract: *This article presents an algorithm for the automatic generation of illustrations performed with chalk and ink hatching techniques. Methods for obtaining different types of hatch patterns are presented. The results obtained with different patterns are compared.*

1. INTRODUCTION

The chalk and ink hatching technique is used very often for creating illustrations. Usually this technique is manually performed by interior designers, architects and artists. It is based on three colors: gray, which is the natural color of the support, black, which is used to simulate dark tones, and white for simulating bright tones. Although these illustrations contain only three colors, they seem to be made up of several shades, which are simulated with parallel hatching lines. Brighter or darker tones can be created by controlling the thickness or the density of the hatching lines. The two techniques can be applied independently, or they can be combined.

2. RECENT RESULTS

There have been proposed various techniques for automatic creation of illustrations. Most of them try to imitate the classic hatching techniques used by artists and designers. While some of them focus on the direction, color and shape of hatch lines, others are optimized for efficiency so they are fast enough to run in real time.

In [1] is presented a method for automatically converting a picture into a pencil drawing. The proposed method uses emphasis and elimination effects. It is based on a computational model for visual attention that predicts the focus of attention in the input image. Regions of less importance are eliminated progressively. In [2] are presented a set of algorithms and tools for generating paintings, illustrations, and animations. The algorithms produce visually pleasant and expressive images that look like hand painted or drawn. In [3] is presented a method for automatically rendering pen and ink illustrations of trees. A tree model is represented by the tree branches and the foliage, using abstract drawing primitives. A method for implementing the principles of pen and ink illustration is presented in [4]. The method achieves textures and tones with line drawing. Stroke textures allow rendering that is resolution dependent. An effective hatching technique that can run in real-time even for large resolution images is presented in [5]. The technique is based on a set of 3D models and on photo realistic textures. A set of real time methods that emulate cartoon styles are presented in [6]. The methods are based on a real time pencil sketching technique for texture mapping. In [7] is presented a method for rendering surfaces in pen and ink. The method is based on controlled density hatching for representing tones, textures, and shapes. A set of methods for line art rendering of smooth surfaces is presented in [8]. An algorithm for finding silhouettes that is based on geometric duality and an algorithm for

segmenting the curves into smooth parts with similar visibility are included. A technique of non-photorealistic rendering is presented in [9]. It is based on ribbons created by partitioning a point cloud into several unidirectional strips. Methods for hatching, cross hatching, and silhouette rendering are presented. A method for line art rendering freeform polynomial and rational surfaces is presented in [10]. The method is efficient enough to run in real time. An algorithm for rendering 3D graphics scenes is presented in [11]. It generates stylized images, suggesting the complexities of the scenes. The algorithm is customizable and produces effects for representing fur, grass and trees. In [12] is presented a method for real time non photorealistic rendering. The method is based on hatching strokes that are scaled to obtain correct size and density, depending on the resolution of the image. A framework for rendering different surfaces with complex geometry and arbitrary topology is presented in [13]. Several hatching and cross hatching patterns are used. The simplicity and effectiveness of the framework is demonstrated. Some sample renderings for a variety of models are provided.

3. THE PROPOSED METHOD

The algorithm proposed in this article simulates the gray tones in an image, with hatch patterns of parallel or crossing lines. The patterns are made up of white or black hatch lines on a gray background. The thickness, density and color of the hatch lines are correlated with the brightness of the corresponding area in the image. The illustrations created using the proposed method have an aspect situated between the illustrations created manually by artists, and the images obtained automatically by dithering techniques.

The proposed method resembles a filtering technique. These techniques are used mainly for blurring, detail enhancement and contour detection [14], [15]. Applying a filter to an image involves changing each pixel in the image based on the pixel value and a group of neighboring pixels. The filter is a square array of weights that contains an odd number of lines and columns. Filtering an image is an iterative operation in which the filter is moved over the image so that the central element of the filter passes the entire image. At each iteration, the new value of the image pixel that is covered by the central element of the filter is calculated, taking into account the pixels covered by the filter and the weights in the filter. In order to be able to filter the pixels at the edges, the image is virtually expanded by mirroring outwards.

Three different filter dimensions were used to generate the hatch patterns: 3 x 3, 5 x 5 and 7 x 7. The number of different hatch patterns these filters can generate and the maximum thickness of the hatching lines are presented in table 1.

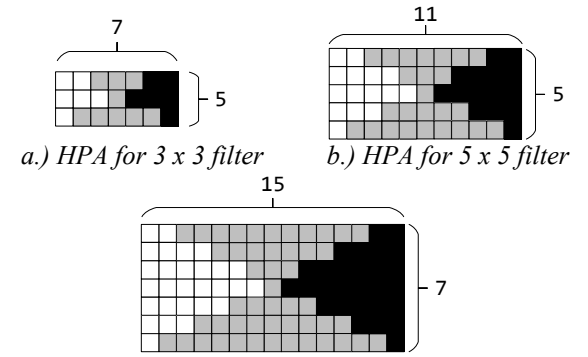
Table 1. Number of hatching patterns and maximum line thicknesses.

Filter dimensions	Parralel lines hatch patterns	Crossing lines hatch patterns	Maximum thickness
3 x 3	7	13	2
5 x 5	11	31	4
7 x 7	15	57	6

The arrays shown in figure 1 were used for generating the parallel lines hatch patterns. They will be called hatch pattern arrays (HPA). The HPAs shown in figure 2 were used for generating the 31 crossing lines hatch patterns of the 5 x 5 filter. The hatch patterns with cross-lines are generated in two passes. In the first pass, the HPA in figure 2.a is used for generating the main direction hatching lines. In the second pass, lines of hatching perpendicular to the main ones are generated, using the HPA presented in figure 2.b.

The hatch filter is different from a regular filter by the fact that it changes n pixels at each iteration, where n is the size of the filter. The central element of the filter does not go through the whole image. Its trajectory determines the directions of the hatch lines. These directions are shown in figure 3. The proposed algorithm can produce two types of hatching patterns. The first type contains only parallel lines. For generating this type of pattern, the image is traversed starting from the top-right corner, following the directions of the lines shown in figure 3.b and continues until the opposite corner is reached. The second type of hatching patterns contains crossed lines. To generate this type of pattern, after the first image traversal, a new traversal starts from the top left corner, follows the directions of the lines shown

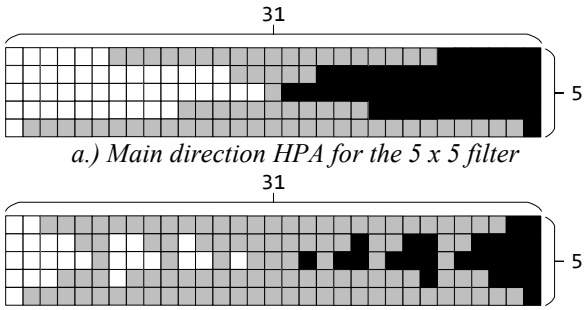
in figure 3.c and ends when the right corner is reached. The main hatching lines are generated using the HPAs of the type presented in figure 2a. The secondary hatching lines are generated using HPAs of the type shown in figure 2b.



a.) HPA for 3 x 3 filter *b.) HPA for 5 x 5 filter*

c.) HPA for 7 x 7 filter

Figure 1. Parallel lines HPAs.



a.) Main direction HPA for the 5 x 5 filter

b.) Secondary direction HPA for the 5 x 5 filter

Figure 2. Crossing lines HPAs.

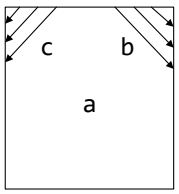


Figure 3. Hatch lines directions. a.) Original image. b.) Main hatch lines. c.) Secondary hatch lines.

4. THE HATCHING ALGORITHM

The algorithm for generating hatch patterns uses the following parameters:

- `MAX` is the maximum brightness of the image pixels. Typically `MAX` is `255`.
- `n` is the size of the filter.
- `d` is the position of the central element within the filter, and is calculated as: `d = (n-1)/2`.

The algorithm consists of the following steps:

1. Initialize all pixels of the output image with gray color.
2. The main hatching lines generation operation starts from the upper-right corner of the original image. The secondary hatching lines generation operation starts from the upper-left corner.
3. If the last hatch line ended, then stop. Otherwise place the filter so that its central element covers the first pixel of the hatch line.
4. Calculate the arithmetic mean (`m`) of the group of pixels covered by the filter.
5. Calculate the column number (`c`) to be retrieved from the HPA:

```
c = (int)(m*(n+1)/(MAX+1))
```

6. Replace a group of `n` pixels from the output image with column `c` in the HPA, so that the central element of the filter is in the middle. Only the white or black pixels from the HPA will be copied in the output image. This allows the generation of cross-hatch lines in two separate passes.
7. If the last pixel of the hatch line is reached, jump to the next hatch line and proceed to step 3. (The distance between two adjacent hatch lines is `n` pixels.) Otherwise move the filter to the next pixel along the hatch line and proceed to step 4.

This algorithm generates hatch patterns with thicknesses and densities that simulate the brightness of regions covered by the filter. According to table 1, the size of the filter determines the number of gray levels that can be simulated and the maximum width of the hatch lines. Thus, a `n` x `n` filter will generate hatch lines of thicknesses between `0` and `n-1` pixels. For n-pixel lines, solid black or white are obtained. Figure 4 presents hatch patterns generated with the proposed algorithm.

The images in figure 5 were created with hatching patterns of parallel lines (images 5.a and 5.b), and hatching patterns of cross lines (image 5.c). The 3 x 3 filter was used for images 5.a and 5.c, and the 5 x 5 filter was used for image 5.b. The original image is presented in figure 7.a, and was taken from [16].

A more pleasant hatching effect can be obtained if the main black hatching lines are perpendicular to the white main hatching lines. This type of hatching pattern will be named *Enhanced Hatching Pattern* (EHP). For implementing this type of hatching, four separate image traversals are needed. At the first traversal the main black hatching lines will be drawn, following the line pattern shown in figure 3.b. Only areas whose average brightness exceeds the brightness of the gray background (`MAX/2`) will be processed. At the second crossing, the secondary black hatch lines will be drawn, following the line pattern shown in figure 3.c. At the next two traversals, the white hatching lines will be drawn. For the main and secondary ones, the trajectories shown in figures 3.c and 3.b will be followed.

The images shown in figure 6 were generated with EHP.

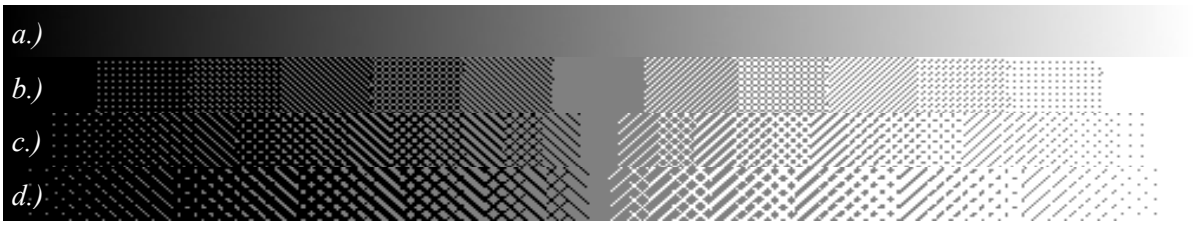
CONCLUSIONS

A few examples of illustrations based on the proposed algorithm have been presented in this paper. The more the filter size increases, the more hatching patterns are possible. But for a pleasant aspect of the generated illustration, the size of the filter must be correlated with the resolution of the original image. The original image, shown in figure 7.a is a relatively small image, having a resolution of 512 x 512 pixels. Because of this, the image shown in figure 5.b appears to be blurred in comparison with the one figure 5.a, although more hatching patterns were used. This effect occurs due to the fact that the filter retains only the averages of the blocks resulting in the loss of fine details. But for large images, large filters produce the best results.

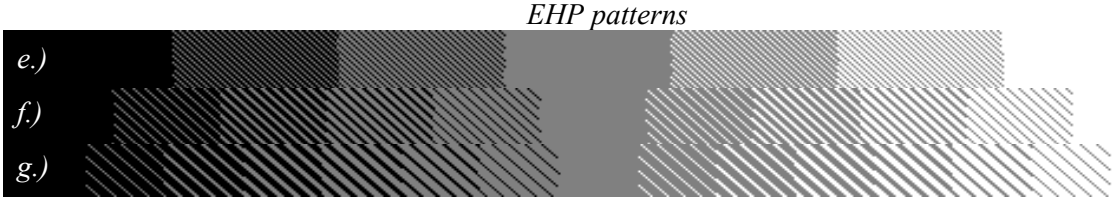
The number of hatch patterns can also be increased by using crossed hatching lines. It can be noticed that the image in figure 5.c has a better look than the one in figure 5.a, because it uses this type of hatch patterns.

The best results were obtained by using EHP. The images in figure 6 were generated with this type of patterns. The images in figures 6.b and 6.c have a pleasant look, even if relatively large filters were used for the low resolution of the original image.

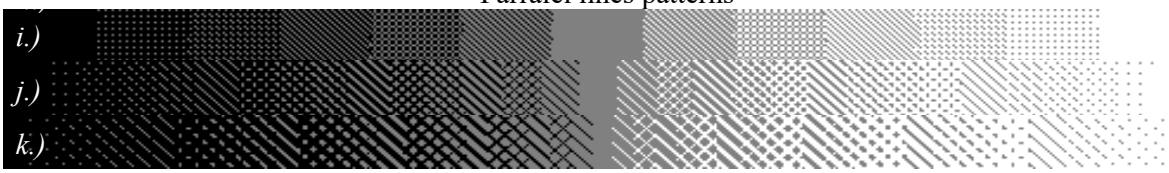
Usually In handmade illustrations, objects are outlined. The illustration shown in figure 7.b contains, in addition to the previous ones, the contours detected using a Laplacian of Gaussian (LoG) filter [14], [15]. The contours were added at the end of the hatching algorithm.



EHP patterns

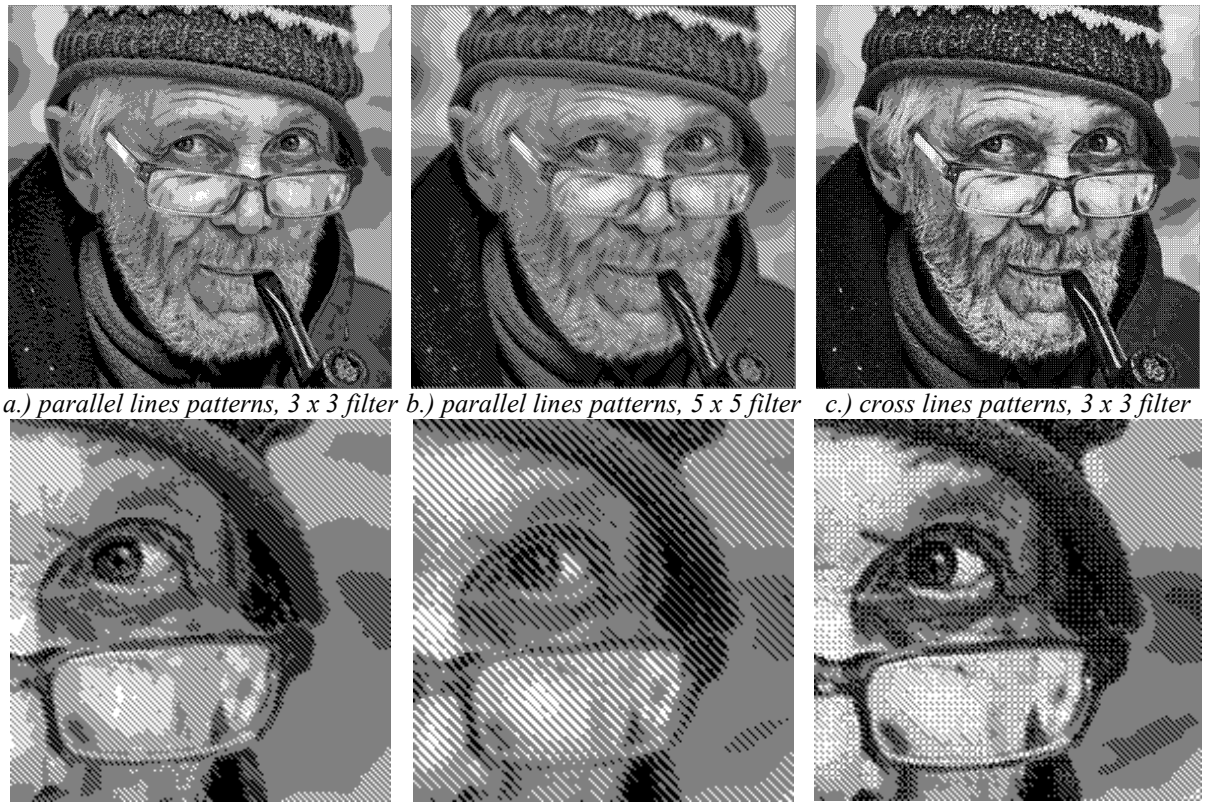


Parralel lines patterns



Coross lines patterns

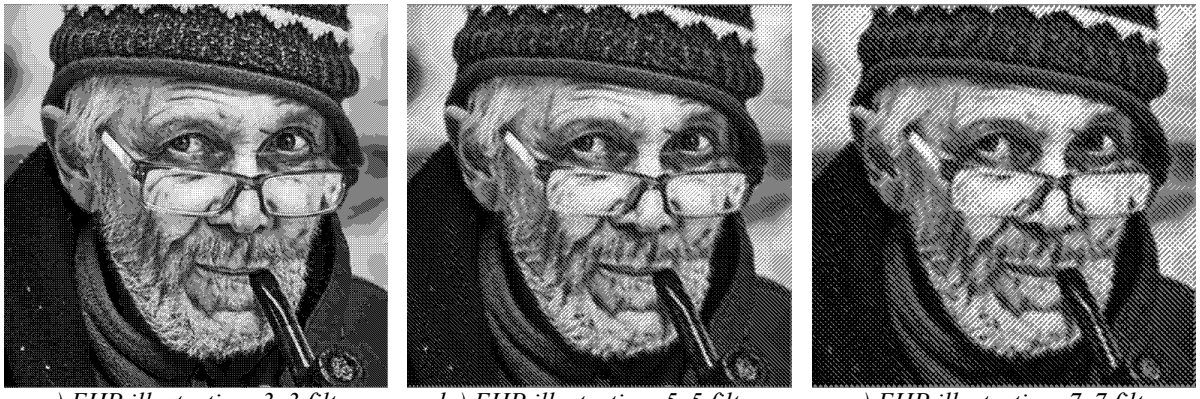
Figure 4. Hatching patterns. a.) Original image. b,c,d.) 3x3, 5x5 and 7x7 EHP. e,f,g.) 3x3, 5x5 and 7x7 parallel lines patterns. i,j,k.) 3x3, 5x5 and 7x7 cross lines patterns.



a.) parallel lines patterns, 3 x 3 filter *b.) parallel lines patterns, 5 x 5 filter* *c.) cross lines patterns, 3 x 3 filter*

d.) detail of a. *e.) detail of b.* *f.) detail of c.*

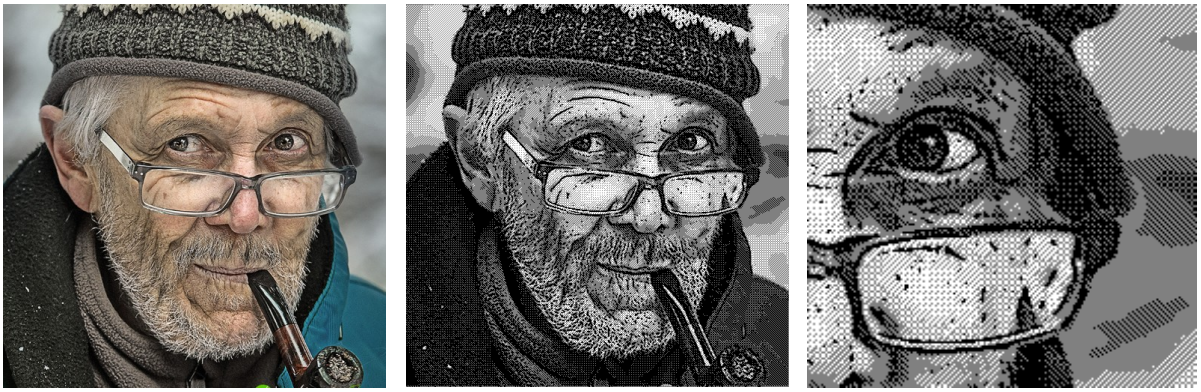
Figure 5. Parallel and cross lines patterns illustrations



a.) EHP illustration, 3x3 filter *b.) EHP illustration, 5x5 filter* *c.) EHP illustration, 7x7 filter*

d.) Detail from a.) *e.) Detail from b.)* *f.) Detail from c.)*

Figure 6. Illustrations generated with EHP.



a.) *b.)* *c.)*

Figure 7. a.) Original image. b.) EHP illustration, 5 x 5 hatching filter, LoG filter contours. c.) Details from b.

REFERENCES

[1]. Hata M., Toyoura M., Mao X., *"Automatic generation of accentuated pencil drawing with saliency map and LIC"*, The Visual Computer vol.28, pp.657–668, 2012.

[2]. Hertzmann A., *"Algorithms for Rendering in Artistic Styles"*, Ph.D, New York Univ. 2001, https://cs.nyu.edu/media/publications/hertzmann_aaron.pdf, accessed 27.06.2017.

[3]. Deussen O., Strothotte T. *"Computer-Generated Pen-and-Ink Illustration of Trees"*, SIGGRAPH conference proceedings, https://www.cs.princeton.edu/courses/archive/fall00/cs597b/papers/plantsketch.pdf, accessed 27.06.2017.

[4]. Winkenbach G., Salesin D.H., *"Computer-Generated Pen-and-Ink Illustration"*, SIGGRAPH proceedings, pp. 91-100, DOI: 10.1145/192161.192184s, 1994.

[5]. Suarez J., Belhadj F., Boyer V., *"Real-time 3D rendering with hatching"*, The Visual Computer, pp. 1-16, DOI: 10.1007/s00371-016-1222-3, April 2016.

[6]. Lake A., Marshall C., Harris M., Blackstein M., *"Stylized Rendering Techniques For Scalable Real-Time 3D Animation"*, Proceedings of the 1st international symposium on Non-photorealistic animation and rendering, pp. 13-20. DOI: 10.1145/340916.340918s, June 2000.

[7]. Winkenbach G., Salesin D. H., *"Rendering Parametric Surfaces in Pen and Ink"*, SIGGRAPH Proceedings, pp. 469-476, DOI: 10.1145/237170.237287, 1996.

[8]. Hertzmann A., Zorin D., *"Illustrating smooth surfaces"*, SIGGRAPH Proceedings, pp. 517-526, DOI: 10.1145/344779.345074, 2000.

[9]. Runions A., Samavati F., Prusinkiewicz P., *"Ribbons - A representation for point clouds"*, The Visual Computer, vol. 23, issue 9, pp. 945-954, September 2007.

[10]. Elber G., *"Interactive Line Art Rendering of Freeform Surfaces"*, Computer Graphics Forum, vol.18, issue 3, pp. 1-12, DOI: 10.1111/1467-8659.00322, 1999.

[11]. Kowalski M. A., Markosian L., Northrup, Bourdev L., Barzel R., Holden L. S., Hughes J. F., *"Art-Based Rendering of Fur, Grass, and Trees"*, SIGGRAPH proceedings, 1999. http://web.cs.ucdavis.edu/~ma/SIGGRAPH02/course23/, accessed 27.06.2017

[12]. Praun E., Hoppe H., Webb M., Finkelstein A., *"Real-Time Hatching"*, SIGGRAPH proceed., DOI: 10.1145/383259.383328G, 2001
[13]. Paiva A., Vital E., Petronetto F., Sousa M. C., *"Fluid-based hatching for tone mapping in line illustrations"*, The Visual Computer, vol. 25, issue 5, pp.519–527, March 2009.
[14]. Burger W. J., Burge M. J., *"Principles of Digital Image Processing"*, Springer, 2013.
[15]. Gonzalez R. C., Woods R. E., *"Digital Image Processing"*, Prentice-Hall, 2002.
[16]. https://fotodependent.wordpress.com/tag/batran/ , accessed 27.06.2017.