Interactive Multi-style Pen-and-Ink Drawings from Images

Botong Qu
Oregon State University
Corvallis, OR 97331, USA

Yue Zhang
Oregon State University
Corvallis, OR 97331, USA

Eugene Zhang
Oregon State University
Corvallis, OR 97331, USA

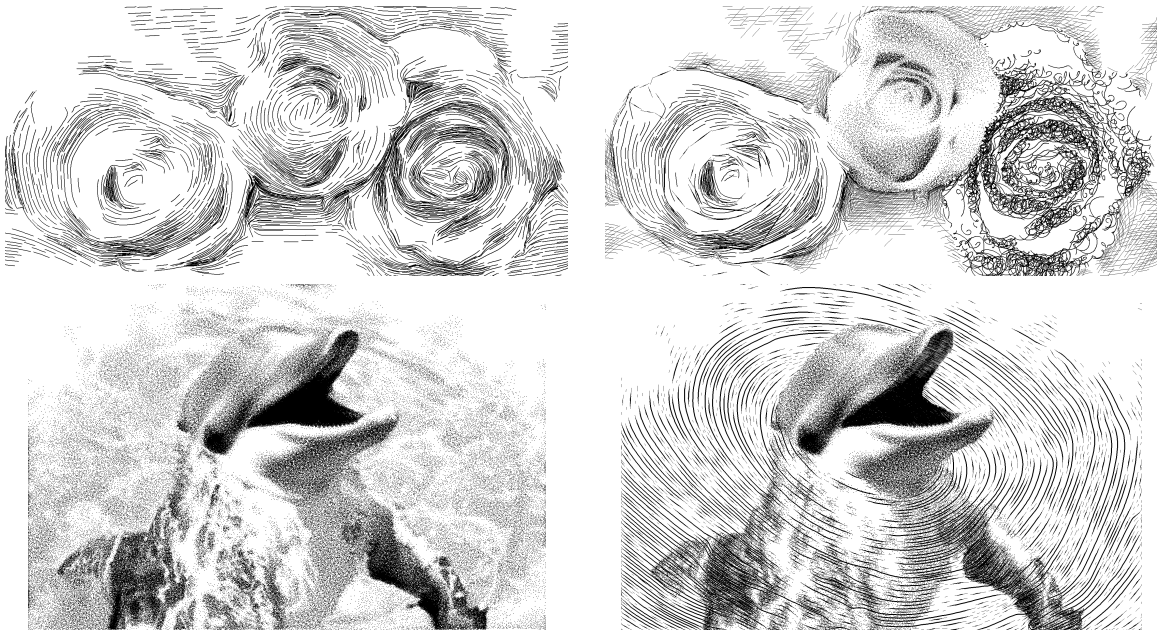


Figure 1: Multi-style pen-and-ink illustration from images can expand the repertoire from single-style illustrations. While the three roses (top left) which are depicted with hatching outline the pedals with artistic clarity, the same three roses (top right) which are depicted with multi-style highlight the individual differences. The three roses now seem to have different levels of freshness, pedal shapes and outlook. This added capability inspires novel artistic compositions and challenges the feasibility of automation. In the bottom row, we allow multiple styles to be applied to the same object, i.e., both stippling and hatching on the part of the dolphin's body under water. This leads to the ability to clearly depict the spatial relationship between water and dolphin that is difficult to achieve with a single-style illustration (left). Our system enables not only the ability to apply different styles to different objects (top), but also the capability to apply multiple styles on the same object (bottom). The original photo of the three roses is a flickr image created by user slgckgc, while the original photo of the dolphin is from shutterstock.com.

ABSTRACT

Existing research in pen-and-ink stylization of images has focused on using either stippling or hatching. In this paper, we introduce an interactive multi-style rendering system in which an image can be stylized with stippling, hatching, and scumbling. Different shading techniques can be used on different objects in the image, leading to various effects such as emphasis, de-emphasis, increase and decrease in contrast, and illusion of texture and material properties in objects. Moreover, different techniques can be applied on the same object, thus enabling illustration of mirrors and transparent materials. We also develop novel techniques that allow the user to design any scumbling pattern beyond circles (circulism). Our techniques can also generate high-quality scumbling results based on the designed pattern. Ensuring adjacent objects with different shading techniques having matching tones is essential in multi-style pen-and-ink drawing. We address this problem using an image-based technique.

All of this research leads to an interactive multi-style pen-and-ink system from images.

SA '17 Technical Briefs, November, 2017, Bangkok, Thailand

ACM ISBN 978-1-4503-5406-6/17/11…$15.00
https://doi.org/10.1145/3145749.3149427

CCS CONCEPTS

• **Computing methodologies** → **Computer graphics**; **Rendering**;

KEYWORDS

Pen-and-ink sketching, Non-photorealistic rendering, Multi-style pen-and-ink, scumbling

ACM Reference format:
Botong Qu, Yue Zhang, and Eugene Zhang. 2017. Interactive Multi-style Pen-and-Ink Drawings from Images. In *Proceedings of SIGGRAPH Asia 2017 Technical Briefs, Bangkok, Thailand, November, 2017 (SA '17 Technical Briefs),* 4 pages.
https://doi.org/10.1145/3145749.3149427

1 INTRODUCTION

Pen-and-ink illustration is a powerful art form that has fascinated generations. Existing research in pen-and-ink illustrations has mostly focused on stylizing an input image or rendering a 3D scene with well-known shading techniques such as stippling and hatching (including cross-hatching) [Rosin and Collomosse 2012].

However, artists often combine these shading techniques to provide more or less emphasis on an object in the scene, increase and decrease contrasts between adjacent objects, and depict material properties and textures of the objects. In this paper, we provide an interactive system in which the user can apply different styles on an input image. Not only can different objects receive different styles (Figure 1: top), one object can also be given multiple styles (Figure 1: bottom). Unlike existing research, our system is interactive, thus enabling quick feedback to the user.

Scumbling is a popular pen-and-ink style that has received relatively little attention from the NPR community when compared with stippling and hatching. While some research has been conducted on simulating scumbling [Chiu et al. 2015], it focuses on *circulism*, a form of scumbling in which the artist scumbles by moving the pen in a circular fashion around a moving center. The moving center is located on a second curve, which is referred to as the *track*. While circulism is perhaps one of the most popular scumbling patterns, artists can choose to scumble in other ways, such as elliptical or even hyperbolic shapes. Our system allows the user to design the scumbling patterns that actuate artistic goals.

One of the most important aspects of pen-and-ink illustrations is the notion of *tone*. Unlike painting and watercolor, pen-and-ink is usually *monochromatic*, i.e., applying black ink on a white paper. Consequently, tone becomes the most important factor in showing the shade in the result such as highlights and shadows. Existing research has focused on achieving tone control for either stippling or hatching. However, these techniques are inadequate for multi-style rendering, especially if there are multiple styles applied to the same object. In this paper, we address this difficulty with the ability to generate matching tones on adjacent objects even when they are given different styles.

2 PIPELINE

We first describe the interface and typical workflows of our system.

The input to our system is an image, which is used to automatically generate a segmentation, a grey image (tone map), a set of feature edges, and a directional field [Zhang et al. 2007]. The segmentation is used to enable the user to apply different pen-and-ink styles to different objects. The tone map is used to guide the placement and densities of stipples, hatches, and scumbles during the rendering stage. The feature edges, when stylized with hatches, can show the silhouette and other feature lines in the objects. The directional field is used to guide the orientations of the hatches and the tracks of scumbles.

Our system enables the editing of any of these objects. For example, two regions in the segmentation can be merged, the tone of object can be darkened, and the directions in one region can be rotated by 90 degrees. All this editing capability can provide the detailed control over the final rendering result. Figure 2 shows an example input image and the derived segmentation, tone map, and feature lines after user editing.

In addition, for each object in the segmentation the user can specify one or more styles. When multiple styles are applied to one object, each style requires a weight. The total weight of all styles defined on one object is 1. For hatching, the user can decide to use a single family of hatches or more families of hatches (cross-hatching). For scumbling, the user can design a scumble pattern using our interface. In addition, the user can change the size of stipples and the width and maximum length of one hatch or scumble stroke.

Once all this control is finalized, our system will generate a pen-and-ink rendering based on the control.

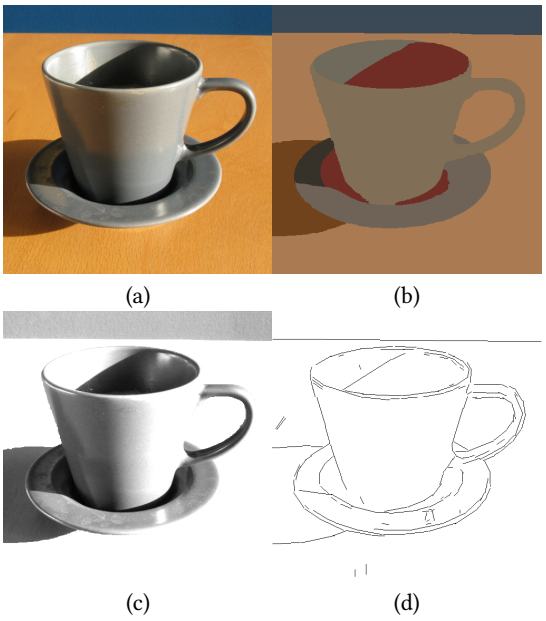


Figure 2: An example input image to our system (a) and derived segmentation (b), tone map (c), and feature lines (d). Note that the segmentation, tone map, and feature lines shown here have been edited using our system. The original photo for the cup is a flickr image created by user rgarcia-suarez74.

3 SCUMBLING PATTERN DESIGN

In this section we provide more detail on the design of scumble patterns using our system and how we generate a scumble given a scumble pattern.

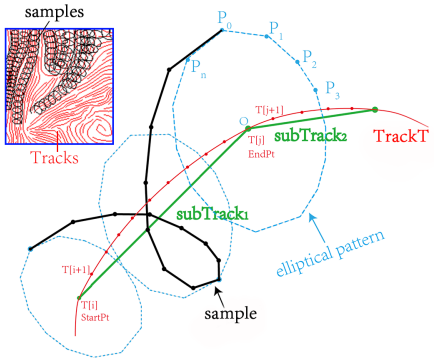


Figure 3: One scumbling sample generated along one subtrack. left top: draw scumbling strokes use the streamlines as tracks.

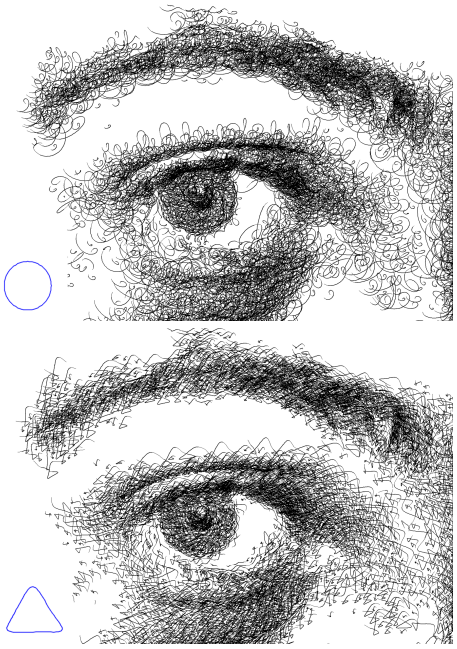


Figure 4: Scumblings with different patterns (shown in the inset). Scumbling drawings with circle pattern makes the drawing feel fuzzy, while scumbling with a triangular pattern creates a sharp and chaotic result. The input photo is from Pics4Learning.com.

Similar to circulism, which is essentially drawing circles around a moving center, our scumble placement is to continuously draw a closed curve around a reference point that moves along a track.

The closed curve is referred to as a *scumbling pattern*. Our system allows the user to design any closed curve using Bezier curve design. In addition, the user can also select from a set of predefined curves (circles, ellipses, squares, regular polygons, etc) and then apply rotation, reflection, and scaling to the curve. Once the scumbling pattern (the closed curve) has been specified, it is saved in our database of scumble patterns and becomes available for future use.

The tracks for the scumbles are generated by placing streamlines following the direction field. This process is essentially the same as placing hatches.

Finally, to place a stroke of a scumbling pattern along a track, we simply follow the process as shown in Figure 3. The user can control the length of the scumbling stroke.

Figure 4 compares the results with two different scumbling patterns: circle (top) and triangle (bottom).

4 TONE MATCHING

In this section we describe how we place shading primitives (stipples, hatches, and scumbles) so that the resulting image matches the input tone map T, a grey-scale image. Note that in pen-and-ink, a pixel is either black or white. Therefore, in the strictest sense, the resulting pen-and-ink will never match the input tone map per pixel. Fortunately, human eyes do not judge tones at the pixel level. Instead, a group of pixels around the point of interest will collectively provide the tone for the center of interest, i.e., a weighted average of intensity of these pixels (kernel). Based on this observation, when comparing the tone difference between corresponding pixels in two images (the input tone map and the current canvas), we use a kernel of the pixel to compute the average squared difference in tone values in the two images [Turk and Banks 1996]. The kernel is a $k \times k$ square. In our setting, we use $k = 16$ as the default, and the user can change it. The larger the k, the faster the rendering process at the cost of less control of tones.

We start with an all-white canvas (current tone). We then iteratively add one shading primitive at a time.

When a stipple is to be added, we identify all the pixels whose kernels cover the stipple and update the current tone values for these pixels. If adding this stipple does not result in any pixel whose tone exceeds an error threshold, this stipple will be added. Otherwise, the stipple is not added.

Hatches are added in a similar fashion except that they are polylines, i.e., a collection of line segments. Consequently, each time a line segment is to be added, all the pixels whose kernel intersects with at last one point on the line segment will be tested to see whether by adding the line segment, the error in the current tone and the input tone map T exceeds the same threshold. If no pixel will be impacted by adding the segment, it will be accepted. Otherwise, the line segment will be rejected. Starting with an initial spacing g_0 between the neighboring hatches, we add evenly-spaced hatches based on g_0 until no hatches can be added. If the resulting canvas still has relatively large errors compared to T, we will halve g_0, which allows more hatches to be added and existing hatches to be lengthened. Our system first tries to extend existing hatches, and if necessary, will add new hatches. Figure 5 illustrates this interleaving process for one iteration. The gap is iteratively halved until no more hatches can be added or lengthened.

Scumbles are also polylines. However, we have observed that if we only add one line segment at a time, the results may contain many scumbles that are too short for the viewers to recognize the underlying scumble pattern. To address this, we make a complete turn of the scumble pattern a unit, which we either add in its entirety or reject completely.

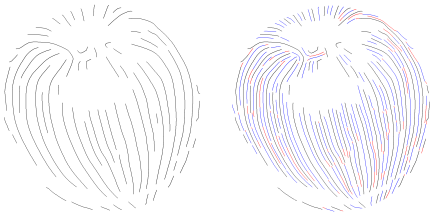


Figure 5: Interleaving strokes while rendering. The units for updating tones are primitives for hatching, stippling and scumbling (line segments, dots, and samples). For hatching, existing streamlines are extended (red) first, then more streamlines (blue) are generated to fill the blank space by decreasing 50% minimum distance between streamlines until target tone is achieved. The input photo is a flickr image created by user Dan Foy.

Note the above process does not differentiate between points or line segments when deciding to whether accept or reject a new primitive, this leads to matching of tones on adjacent objects that have different styles.

If there are at least two styles defined for the same object, we will generate for each style a desired tone map based on the weights assigned to each style. Then we can proceed to add stipples or hatches and scumbles based on their respective desired tone maps. Consequently, in the resulting image adjacent objects have matching tones even when at least one object has at least two styles.

Figure 6 shows a stylized apple using hatching, stippling, and scumbling, respectively, with the same desired tone map. The average pixelwise difference between the tone of stippling in Fig. 6 and the input image is 1.25%. The average pixelwise tone difference between the hatching and the input and between the scumbling and the input are 2.0% and 3.9% respectively.

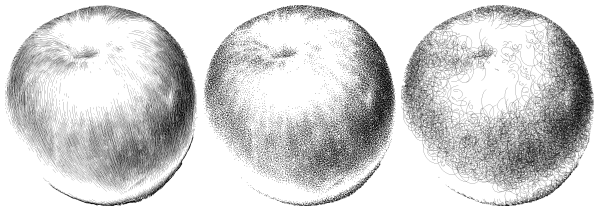


Figure 6: From left to right, hatching, stippling and scumbling preserves similar tones in rendering results despite their different primitives.

5 PERFORMANCE

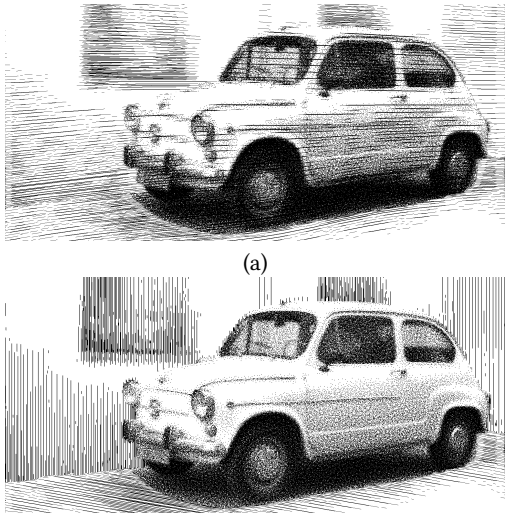
All the results used in this paper are created by our system on a laptop with Intel i7 CPU (2.7 GHz). They are created in a few minutes; in particular, it took less than one minute to generate the rendered results while the rest is attributed to the processing of the input image as well as the design process.

In general, the total design time varies for each user on each artwork. Some users may render one region with different parameters a number of times. Our system is efficient in completing stylization, and users can repeatedly render to reach satisfaction.

Our system allows interactive control over the steps in our pipeline after importing one input image. Several computer science students took a short training to use our system and were able to create satisfactory results.

6 RESULTS

Figure 1 shows some benefits of multi-style pen-and-ink, such as differentiating objects that are similar (top row) and more clearly demonstrating the spatial relationships between overlapping objects (bottom row). Figure 7 shows two multi-style rendering from the same input image that shows a moving car (top) and a parked car (bottom).



(a)

(b)

Figure 7: Two multi-style renderings from the same input image: (a) a moving car, and (b) a parked car. The input photo is a flickr image created by user stvcr.

ACKNOWLEDGMENTS

We wish to thank Soo-Min Yoo for her artistic suggestions. This research is supported by US National Science Foundation awards 1566236 and 1619383.

REFERENCES

Chun-Chia Chiu, Yi-Hsiang Lo, Ruen-Rone Lee, and Hung-Kuo Chu. 2015. Tone- and Feature-Aware Circular Scribble Art. *Comput. Graph. Forum* 34, 7 (Oct. 2015), 225–234. DOI: http://dx.doi.org/10.1111/cgf.12761

Paul Rosin and John Collomosse. 2012. *Image and Video-based Artistic Stylisation.* Springer Publishing Company, Incorporated.

Greg Turk and David Banks. 1996. Image-guided Streamline Placement. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '96).* ACM, New York, NY, USA, 453–460. DOI: http://dx.doi.org/10.1145/237170.237285

Eugene Zhang, James Hays, and Greg Turk. 2007. Interactive Tensor Field Design and Visualization on Surfaces. *IEEE Transactions on Visualization and Computer Graphics* 13, 1 (Jan. 2007), 94–107. DOI: http://dx.doi.org/10.1109/TVCG.2007.16