Abstract Line Drawings from 2D Images

Minjung Son
POSTECH
sionson@postech.ac.kr

Henry Kang
University of Missouri at St. Louis
kang@cs.umsl.edu

Yunjin Lee
University of Michigan
yunjin@umich.edu

Seungyong Lee
POSTECH
leesy@postech.ac.kr

Abstract

We present a novel scheme for automatically generating line drawings from 2D images, aiming to facilitate effective visual communication. In contrast to conventional edge detectors, our technique imitates the human line drawing process and consists of two parts: line extraction and line rendering. We propose a novel line extraction method based on likelihood-function estimation, which effectively finds the genuine shape boundaries. We consider the feature scale and the blurriness of lines with which the detail and the focus-level of lines are controlled in the rendering. We also employ stroke textures to provide a variety of illustration styles. Experimental results demonstrate that our technique generates various kinds of line drawings from 2D images enabled by the control over detail, focus, and style.

1. Introduction

Line drawing is a simple yet effective means of visual communication. A good piece of line art, sketch, or technical illustration typically consists of a small number of lines, describing the identifying characteristics of objects, that is, shapes. This enables quick recognition and appreciation of the subject with little distraction from relatively unimportant contents. Also, line-based object representation can provide significant gain, both in terms of time and storage space, in subsequent processing of the data.

As an effective tool for abstract shape visualization, line drawing falls squarely within the scope of non-photorealistic rendering (NPR). In recent years, 3D line drawing, that is, line drawing of 3D objects, has been a central issue of NPR [21, 14, 15, 5, 17, 35, 26, 39], and has proven to outperform conventional photorealistic rendering in terms of quick and accurate communication of shapes. The shape of a 3D object is in general *explicitly* represented by low-level geometric elements (such as points or vertices on the surface) whose coordinates are known *a priori*, and hence the problem of 3D line drawing is reduced to identification of important contours (such as creases or silhouettes) formed by these elements. In 2D line drawing, however, the target shape to convey is *implicitly* embedded in a 2D lattice (image) and often corrupted by noise, making the task of contour identification a less obvious–in fact extremely difficult–one.

Traditionally, various problems in 2D line extraction have been addressed by a low-level image analysis technique called edge detection. From the perspective of visual communication, however, edge detectors typically have limitations in the following respects. First, the resulting edge map often includes lines that may be accurate but less meaningful (or even distracting) to the viewers. Second, the 'importance' of a line typically depends on the image gradient only, hindering the possibility of a more sophisticated detail control. Third, they have no interest in the 'style' of lines and thus do not provide any mechanism for style control.

In this paper, we present an automatic 2D line-drawing framework that addresses these limitations. The main idea is to extract lines that locally have the biggest 'likelihood' of being genuine lines that will be of interest to the viewers. While extracting lines, we also compute additional line properties, such as *feature scale* and *blurriness*. The rendering module then performs line drawing by mapping on the extracted lines stroke textures with a variety of styles. For the versatility of line drawing, the attributes of the lines are automatically adjusted according to the line properties delivered from the line extraction module.

Our technique resembles the human line drawing process. The likelihood function used for line extraction is constructed by merging small line segments fitted for the neighborhoods of feature points of the image (see Fig. 2(e)). This is similar to the sketching process of artists where they

typically apply many small strokes over the shape contours. In rendering extracted lines, the thicknesses and opacities of lines are controlled by their feature scales and blurriness. This imitates a typical artistic drawing where important shape features are drawn prominently with strong colors while background is depicted with soft tone and color.

In summary, our line drawing technique provides the following merits;

- *Effective shape extraction and depiction:* Due to the resemblance to the human line drawing process, 'perceptually meaningful' lines can be captured and then rendered in 'recognition efficient' styles.
- *Effective style control:* The level of details, the level of focus, and the style of the illustration can be controlled in a way that is impossible with conventional edge detection techniques. For example, we can remove a set of strong but unimportant edges, or switch line styles for different moods of the illustration.
- *Effective visual communication:* Given the above properties, our technique results in fast and accurate visual communication in terms of conveying shapes and also identifying subjects.

2. Related Work

2.1 Stroke-based rendering

Most of the existing image-guided NPR techniques aim at creating styles that are somewhat distant from 'pure' line drawing. These styles include painting [20, 4, 12, 13, 9, 11, 18], pen-and-ink illustration [29, 28, 30], pencil drawing [34, 7], and engraving [25, 7], where clean, accurate depiction of outlines is either unnecessary or relatively unimportant. Instead, they focus on filling the interior regions with certain types of 'strokes' (thus the term of stroke-based rendering), such as lines, rectangles, or polygons, to stylistically describe the tonal variation across the surface. Some of these techniques use textured polygons as strokes to widen the possible rendering styles. While we also use textured strokes for the line rendering, the strokes in our case are placed along the detected shape boundaries to convey the shapes, not the interior tonal information.

2.2 Image abstraction

In image abstraction (also called image tooning), the interior regions are abstracted by color smoothing and pixel clustering. To clearly distinguish the clustered regions and reveal the shape boundaries, line drawing is often used as part of the rendering process. DeCarlo and Santella [6] presented an image abstraction system based on Canny edge detector [1] and mean-shift image segmentation [3]. Wang et al. [36] and Collomosse et al. [2] both applied the mean-shift segmentation to classify regions in video. Wen et al. [37] also used mean-shift segmentation to produce a rough sketch of the scene. Fischer et al. [8] and Kang et al. [18] both employed Canny edge detector to obtain stylized augmented reality and line-based illustrations, respectively. Note in general edge detector is suitable for extracting lines while image segmentation is effective in pixel clustering.

Gooch et al. [10] presented a facial illustration system based on difference-of-Gaussians (DoG) filter, similar to Marr–Hildreth edge detector [22]. Winnemöller et al. [38] recently extended this technique to general color images and video. Unlike Canny's method, their DoG edge detector produces a group of edge pixels along the boundaries in non-uniform thickness, creating an impressive look reminiscent of pen-and-ink line drawing done by real artists. On the other hand, it also makes it difficult to extract the accurate shape and direction of each contour, which may hinder the flexible control of line styles.

2.3 Edge detection

In addition to the standard edge detectors mentioned above, there are a variety of edge detectors that are useful in many image processing applications [32, 27, 16, 33, 23]. In fact, the literature on edge detection is vast, and we make no attempt to provide a comprehensive survey. The limitations of edge detectors in general have been discussed in Sec. 1.

Our line extraction algorithm (which will be described in Section 4) can also be regarded as a novel edge detector. The main difference is that our approach is based on local line fitting, mimicking the human line drawing process. We show that this approach is effective in terms of capturing genuine lines and also preserving the line connectivity. More importantly, our line extraction process is specifically designed to enable control over various aspects of the illustration, such as line details, focus, and styles, which is essential in a line drawing application but not supported in typical edge detection algorithms.

3. Overall Process

Our overall framework consists of two modules: line extraction and line rendering (see Fig. 1). Each module is again decomposed into multiple steps which we briefly describe here.

Feature point sampling: Our line extraction method is based on kernel-based density estimation. To expedite the process, we perform this estimation from the sample pixels (rather than the entire pixels) that are highly relevant to the task of line drawing, that is, the pixels with high enough

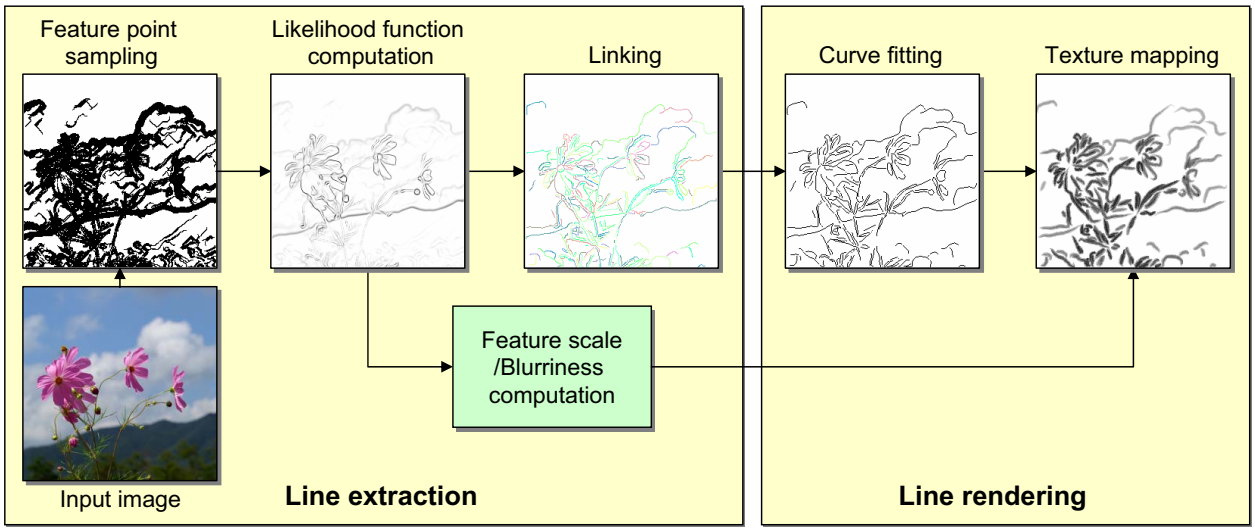


Figure 1. Overall process

edge strengths. We use gradient magnitude to measure the initial edge strength.

Likelihood function computation: For each pixel, we compute its likelihood of being part of a genuine line by performing least-square line fitting in the neighborhood. The local likelihood functions obtained in this way are then combined to construct a global likelihood function over the entire image, from which we extract lines.

Feature scale and blurriness computation: In general, the size of the neighborhood (kernel) strongly affects the resulting likelihood function. We compute the likelihood functions in two levels (with small and large kernel sizes), to extract additional information including feature scale and blurriness, based on the line fitting errors. The feature scale of a pixel refers to the size of the feature that the pixel belongs to, and it is useful for separating large dominant features from unimportant details. The blurriness measures how blurry its neighborhood is, and is used to control the level of focus in the illustration between foreground objects and the background. In addition, the two likelihood functions are combined using the blurriness so that the resulting likelihood function enables us to extract lines from blurry regions in a more robust way.

Linking: We connect the ridge points on the global likelihood function to create individual line strokes. We first create a set of connected components by naively connecting the adjacent ridge points (which we call clustering). We then extract an ideal (minimum-cost) line stroke from each cluster.

Curve fitting: Each line stroke is further smoothed by curve fitting. In particular, we adjust the number of points along the curve based on the local normal derivatives, to reduce the number of points while preserving the shape.

Texture mapping: The final illustration is created by visualizing the lines as textured strokes. The type of stroke texture affects the overall style or mood of the illustration. The line attributes such as thickness and opacity are controlled by the feature scale and the blurriness to emphasize dominant and foreground objects while deemphasizing detailed and background parts. Since a line with zero thickness or zero opacity is invisible, the line attributes can also be used to change the amount of lines in a drawing for level-of-detail (LOD) control.

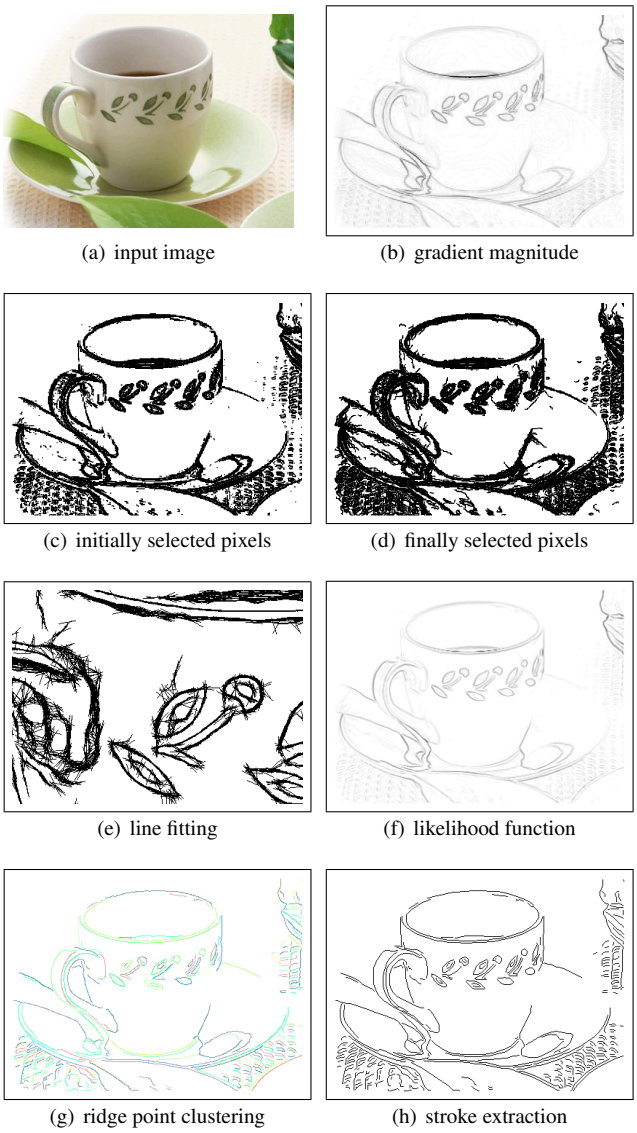
The details of these two modules (line extraction and line rendering) will be presented in Secs. 4 and 5, respectively.

4. Line Extraction

4.1. Feature point sampling

Given an input image $I(x, y)$, we construct a Sobel gradient map, denoted by $I_s(x, y)$, where each pixel is associated with its gradient vector $\mathbf{g}(x, y)$. To represent the edge strength, we use a normalized gradient magnitude, denoted by $\hat{g}(x, y) \in [0, 1]$. From I_s, we extract N sample pixels $\mathcal{P} = \{\mathbf{p}_1, ..., \mathbf{p}_N\}$ by applying a hysteresis thresholding method (similar to that of [1]) on the values of $\hat{g}(x, y)$.

We first choose pixels having $\hat{g}(x, y)$ larger than α_h. Next, among the pixels connected to the selected pixels in the normal directions to their gradient vectors $\mathbf{g}(x, y)$, we choose pixels having $\hat{g}(x, y)$ larger than α_l. We repeat this point tracing until no more points can be added. For the results in this paper, α_h was automatically determined as $2/3$ of the average gradient magnitude, except Fig. 2 where a larger α_h was used to clearly show the process. For α_l, we



(a) input image (b) gradient magnitude

(c) initially selected pixels (d) finally selected pixels

(e) line fitting (f) likelihood function

(g) ridge point clustering (h) stroke extraction

Figure 2. Line extraction steps

used $\alpha_h/2$. Figs. 2(c) and 2(d) show the selected points in these two steps.

4.2. Likelihood function computation

Given a 2D point set $\mathcal{P}$, we wish to estimate an unknown likelihood function $L(\mathbf{x})$, representing the probability that a point $\mathbf{x} \in \mathbf{R}^2$ belongs to a genuine edge. To derive $L(\mathbf{x})$ from $\mathcal{P}$, we accumulate the local likelihood functions $L_i(\mathbf{x})$ computed at points $\mathbf{p}_i$ in $\mathcal{P}$. Each $L_i(\mathbf{x})$ is computed by fitting a line to the neighborhood of $\mathbf{p}_i$. This process is motivated by a robust point set filtering technique proposed in [31] and we adapt the technique to the domain of 2D line extraction. The line fitting step and the final likelihood function $L(\mathbf{x})$ are visualized in Figs. 2(e) and 2(f), respectively.

To compute a local likelihood function $L_i(\mathbf{x})$, we define a circular kernel K_i of radius h centered at $\mathbf{p}_i$, and estimate the location of a representative line e_i in the kernel. Let e_i be represented by ${\mathbf{n}_i}^T\mathbf{p} + d_i = 0$, where $\mathbf{n}_i$ is a unit vector orthogonal to e_i. We obtain e_i using the well-known total least square method [24]. In particular, we compute e_i by minimizing $E(\mathbf{p}_i, h)$ under the constraint that ${\mathbf{n}_i}^T\mathbf{n}_i = 1$, where

$$E(\mathbf{p}_i, h) \;=\; \frac{1}{h^2} \cdot \frac{\sum_{j=1}^{n} w_i(\mathbf{p}_j)({\mathbf{n}_i}^T\mathbf{p}_j + d_i)^2}{\sum_{j=1}^{n} w_i(\mathbf{p}_j)}. \quad (1)$$

$w_i(\mathbf{p}_j)$ is the weight for a point $\mathbf{p}_j$ in the kernel K_i, defined by

$$w_i(\mathbf{p}_j) \;=\; \hat{g}_j \cdot \max\left[\frac{\mathbf{g}_i \cdot \mathbf{g}_j}{|\mathbf{g}_i||\mathbf{g}_j|}, 0\right]. \quad (2)$$

Thus, higher weights are given to points $\mathbf{p}_j$ having strong edge features as well as similar edge directions to $\mathbf{p}_i$. This is an essential criterion for improving the quality of line extraction in our application. A line in an image most likely lies on or nearby pixels with strong gradient magnitudes. Also, by considering gradient directions in Eq. (2), we can separate neighboring lines with different directions.

We then define an anisotropic (elliptical) kernel K_i' for $\mathbf{p}_i$ using the line e_i. The center $\mathbf{c}_i$ of kernel K_i' is determined by projecting onto e_i the weighted average position of points $\mathbf{p}_j$ in K_i, using the weights $w_i(\mathbf{p}_j)$. The longer radius of K_i' (in the direction of e_i) is fixed as h. To determine the shorter radius of K_i' (in the direction of $\mathbf{n}_i$), we compute the weighted average distance from e_i to the points in K_i. The shorter radius is set by multiplying a constant γ to the average distance. In our experiments, we used $\gamma = \sqrt{2}$.

Now we define the local likelihood function $L_i(\mathbf{x})$ as inversely proportional to the distance to e_i;

$$L_i(\mathbf{x}, h) \;=\; \phi_i(\mathbf{x} - \mathbf{c}_i)\left[\frac{h^2 - [(\mathbf{x} - \mathbf{c}_i) \cdot \mathbf{n}_i]^2}{h^2}\right], \quad (3)$$

where ϕ_i denotes the kernel function defined by an anisotropic, bivariate Gaussian-like function oriented to match the shape of K_i'. In our implementation, we use a uniform cubic B-spline basis function. Thus ϕ_i peaks at $\mathbf{c}_i$, then gradually decreases as moving toward the ellipse boundary and stays zero outside.

Finally, we obtain the global likelihood function as the accumulation of local ones and normalize it into the range of 0 and 1;

$$L(\mathbf{x}) \;=\; \sum_{i=1}^{N} \hat{g}_i(1 - E(\mathbf{p}_i, h))L_i(\mathbf{x}, h). \quad (4)$$

The gradient magnitude $\hat{g}_i$ is used as a weight to reflect the edge strength. $E(\mathbf{p}_i, h)$ is included so that a local likelihood function with a lower line fitting error can have more influence on the global one. Note that the value of $E(\mathbf{p}_i, h)$ is between 0 and 1.

4.3. Linking

Ridge point clustering In the global likelihood function $L(\mathbf{x})$, the ridges with high function values are most likely to form the genuine edge lines. The goal of clustering is to put all the pixels belonging to the same line into a single cluster. We again use a method similar to the hysteresis thresholding of Canny edge detector [1]. That is, we start with ridge pixels with local maximum function values larger than T_h, then trace along the ridge directions until we visit rigid pixels with values smaller than T_l. Fig. 2(g) shows the result of ridge point clustering.

Stroke extraction While each cluster is now a simply connected point set, it may not be smooth enough to be directly drawn as a line stroke or to be point-sampled for curve fitting. To obtain a smooth line stroke from a cluster, we first find the farthest pair of points in the cluster and then obtain the shortest path between them. For the path computation, the cost c_{ij} for connecting two points $\mathbf{p}_i$ and $\mathbf{p}_j$ is defined by

$$c_{ij} = l_{ij}^2 \cdot \delta_{i,j} \cdot \max\left(|\mathbf{n}_i \cdot \mathbf{s}_{ij}|, |\mathbf{n}_j \cdot \mathbf{s}_{ij}|\right), \qquad (5)$$

where $\delta_{i,j} = 1 - \frac{|\mathbf{n}_i \cdot \mathbf{n}_j|}{2}$. l_i is the distance between $\mathbf{p}_i$ and $\mathbf{p}_j$, and $\mathbf{s}_{ij}$ is the unit direction vector from $\mathbf{p}_i$ to $\mathbf{p}_j$. The cost c_{ij} becomes low when the distance is short, when the normals are similar, or when the path direction is orthogonal to the normals. Fig. 2(h) shows the result of stroke extraction.

4.4. Two-level processing

In constructing the likelihood function (Sec. 4.2), the kernel size plays an important role. The use of a small kernel is good for extracting minute details but the resulting lines can be disconnected or jagged especially when the region is blurry (see the top left part of Fig. 3(b)). With a bigger kernel, it is easier to construct long lines even in a blurred region but small-scale details may disappear (see the hair region in Fig. 3(c)). To have both effects, we compute the likelihood functions in two levels with different (small and large) kernel sizes, and obtain a new likelihood function by combining them. Fig. 3(d) shows the line strokes extracted from the combined likelihood function. Note that in Fig. 3(d), lines are extracted properly in both blurry regions and detailed regions.

(a) input image (b) small kernel ($h = 3$) (c) large kernel ($h = 7$) (d) combined likelihood function

Figure 3. Two-level processing

Blurriness computation When combining the two likelihood functions, we can determine their relative weights by computing the blurriness of image regions. The blurriness indicates the degree to which a region is blurred. In general, an important region has a clearly visible structure while an unimportant area (such as the background) is often blurry due to the photographer's intentional defocusing. Therefore, we can assume that the blurriness represents how important (or focused) a region is.

We use the line fitting errors $E(\mathbf{p}_i, h)$ from small and large kernels to compute the blurriness at feature points $\mathbf{p}_i$ in $\mathcal{P}$. A blurry region has a large fitting error because strong edge points may not exactly lie on the fitted line. On the other hand, the fitting error becomes small in a region without blurring.

We define the blurriness b_i at $\mathbf{p}_i$ by

$$b_i \quad = \quad E(\mathbf{p}_i, h_d) + w_i E(\mathbf{p}_i, h_b), \qquad (6)$$

where h_d and h_b are the two different sizes (small and large) of the kernel, respectively. w_i is the relative weight when we combine the line fitting errors. If $E(\mathbf{p}_i, h_d)$ is small, it means the region is not much blurred, and we can almost neglect $E(\mathbf{p}_i, h_b)$. On the other hand, when $E(\mathbf{p}_i, h_d)$ is big, we need to check again with the large kernel to determine the amount of blurriness. In this case, the line fitting error $E(\mathbf{p}_i, h_b)$ should be reflected on the blurriness value. For simplicity, we set w_i as $E(\mathbf{p}_i, h_d)$, which worked well

in our experiments. The blurriness b_i has a value between 0 and 1.

Likelihood function composition With the blurriness b_i computed for each feature point $\mathbf{p}_i$ in $\mathcal{P}$, we can compose the two likelihood functions by

$$L_{i,d}(\mathbf{x}) = \hat{g}_i(1 - E(\mathbf{p}_i, h_d))L_i(\mathbf{x}, h_d), \quad (7)$$

$$L_{i,b}(\mathbf{x}) = \hat{g}_i(1 - E(\mathbf{p}_i, h_b))L_i(\mathbf{x}, h_b), \quad (8)$$

$$L(\mathbf{x}) = \sum_{i=1}^{N} \big((1 - b_i)L_{i,d}(\mathbf{x}) + b_i L_{i,b}(\mathbf{x})\big). \quad (9)$$

After composition, we normalize $L(\mathbf{x})$ into the range of 0 and 1.

Feature scale computation In addition to the blurriness, we compute for each feature point $\mathbf{p}_i$ another property, called feature scale f_i, from the line fitting errors. The feature scale defines the size of the feature in the surrounding region of a feature point. If the feature scale is large, the region belongs to a large dominant structure of the image. Otherwise, the region corresponds to a small detail of the shape.

Around a large feature, the line fitting error remains consistent because a line can nicely approximate the feature regardless of the kernel size. It could even decrease with a larger kernel because the fitting error is normalized by the kernel size. Around small features, however, the line fitting errors increase when the kernel size becomes large. Therefore, we define the feature scale f_i at $\mathbf{p}_i$ by

$$f_i = 1 - \frac{f'_i - \min_i\{f'_i\}}{\max_i\{f'_i\} - \min_i\{f'_i\}}, \quad (10)$$

where f'_i is $\arctan(E(\mathbf{p}_i, h_b) - E(\mathbf{p}_i, h_d))$, and h_b and h_d are the same as in Eq. (6). We use the $\arctan$ function in Eq. (10) to suppress the influence of extremely small and large values of $E(\mathbf{p}_i, h_b) - E(\mathbf{p}_i, h_d)$.

From Eq. (10), it is clear that the feature scale f_i is between 0 and 1. Note that when there is no strong feature around $\mathbf{p}_i$, the line fitting errors can be consistently large with different kernel sizes, resulting in a large feature scale. However, in this case, no line will be drawn through $\mathbf{p}_i$ in the rendering process and the wrong feature scale has no effect on the result image.

5. Line Rendering

In this section, we first describe the basic process of line rendering, including curve fitting and texture mapping. We then explain how to render individual line strokes with various styles using feature scale and blurriness.

5.1. Line rendering process

We apply curve fitting to connected stroke points to construct smooth lines similar to human-drawn line illustrations. We first reduce the number of points in each line stroke by point sampling. For effective shape preservation, we should sample more points in a segment where normals are changing abruptly. The sampling density is controlled with the sampling cost $s_{i,i+1}$ between two consecutive points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ along a line stroke, defined by

$$s_{i,i+1} = l_{i,i+1} \cdot \delta_{i,i+1} \cdot \delta_{i+1,last}. \quad (11)$$

In Eq. (11), $\delta_{i,j} = 1 - \frac{|\mathbf{n}_i \cdot \mathbf{n}_j|}{2}$. $l_{i,i+1}$ is the distance between $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. $\mathbf{p}_{last}$ is the last point sampled so far.

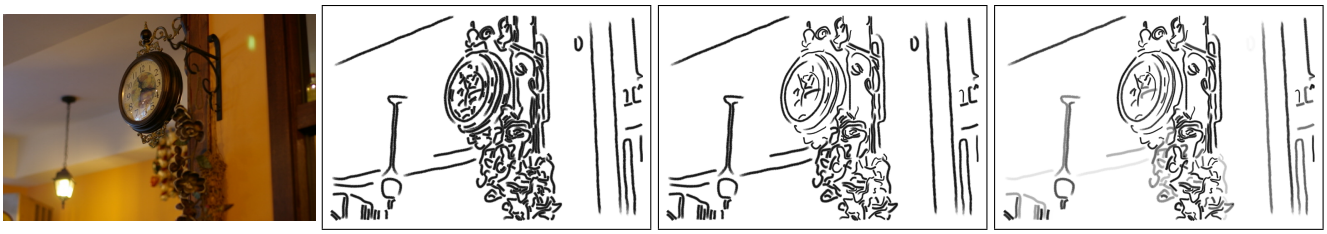
We start with sampling the first point of a line stroke and set it as $\mathbf{p}_{last}$. We visit the line stroke points in sequence and sample the next point $\mathbf{p}_{i+1}$ when the cumulative distance from $\mathbf{p}_{last}$ exceeds a pre-defined threshold. After the sampling, the cumulative distance is reset to zero and $\mathbf{p}_{last}$ is updated. In our experiments, the pre-defined threshold is a half of the logarithmic length of the line stroke. Thus, a relatively smaller number of points are sampled from a longer line, which makes it smoother than a shorter line. To avoid having too few samples along a nearly straight line, we also regularly sample points for every pre-defined number of points, which is 10 in our experiments.

A line stroke is converted to a smooth curve by generating a Catmull-Rom spline curve from the sampled points. The final line drawing is obtained by mapping a material texture along the spline curves. The type of the stroke texture determines the overall style or mood of the resulting line drawing. To further imitate the human line drawing, we allow line attributes (such as thickness and opacity) to be adjusted, using the feature scale and blurriness computed in the line extraction process. The following sections discuss this issue.

5.2. Detail control using feature scale

The feature scale of a line can be computed by averaging the feature scale values of the feature points belonging to the line. The feature scale of a line estimates the size of the feature that the line is describing. Using the feature scale values, we can control the amount of details in the line drawing by removing lines with small feature scales. We can also adjust the thickness of a line stroke according to its feature scale.

To control the amount of details and line thickness, we use two thresholds f_l and f_h for the feature scale. The lines whose feature scales are smaller than f_l are either omitted or drawn with the minimum line width, while lines with larger feature scales than f_h are drawn with the maximum



(a) input image (b) simple line rendering (c) feature scale control for (b) (d) blurriness control for (c)

Figure 4. Line attribute control with feature scale and blurriness

line width. Line widths between f_h and f_l are obtained by linear interpolation.

Fig. 4 shows an example of detail control with feature scale. In Fig. 4(c), compared to Fig. 4(b), small details have been removed while large features are preserved.

5.3. Level-of-focus control using blurriness

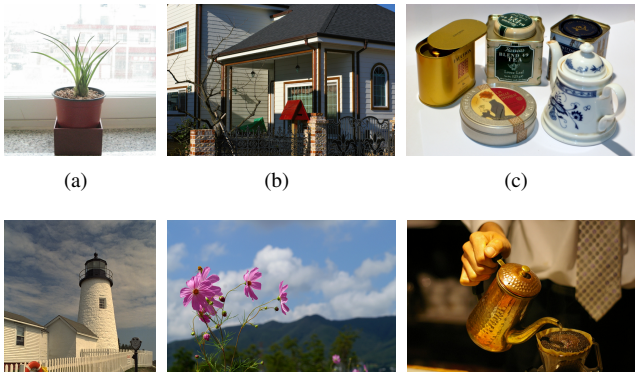
Similarly to the case of feature scale, the blurriness of a line can be computed by averaging the blurriness of feature points along the line. The blurriness of a line relates to how much the region around a line is focused or important in the image. The blurriness can be used to adjust the opacity of a line, that is, to control the level-of-focus, where large blurriness means less opacity. In addition, the blurriness can help remove an unimportant line by making the opacity zero.

In the line rendering process, line opacity is controlled with the blurriness using two thresholds b_l and b_h in a similar way to the feature scale. Fig. 4(d) shows an example where the opacities of the lines in Fig. 4(c) have been changed according to the blurriness.

6. Experimental Results

The line drawing results in Fig. 6 are obtained from the test images in Fig. 5. These results demonstrate that the amount of details and the level of focus are effectively controlled by our technique using the feature scale and blurriness. For Figs. 6(b), 6(d), and 6(f), a pastel texture was used. Figs. 6(a), 6(c), and 6(e) were drawn with a black-ink texture.

By default, the radii of the small and large kernels, h_d and h_b, are 3 and 6 pixels, respectively. We usually set the threshold T_h for selecting ridge pixels as 0.08. For a black-ink style illustration, we use a larger value for T_h to remove more details. The threshold T_l is set to 0 for all the results in this paper. We also provide a threshold T_c for removing very short lines. T_c is usually set to 7 pixels. For feature scale control, f_l is usually between 0.4 to 0.5, and f_h between 0.6 to 0.7. For control with blurriness, b_l ranges between 0.1 to 0.3, and b_h between 0.5 to 0.8 in our experiments. The parameter values for the results in Fig. 6 are given in Table 1.



(a) (b) (c) (d) (e) (f)

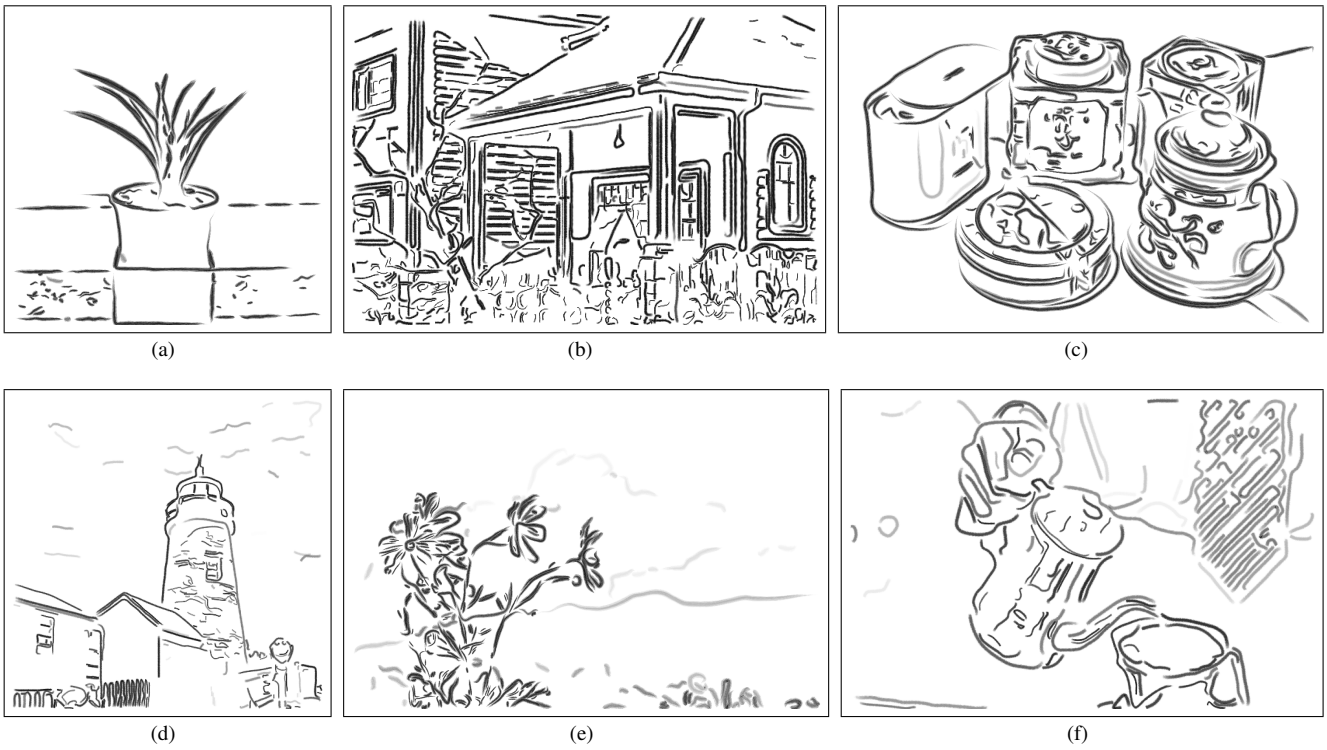
Figure 5. Input images

fig.	h_d	h_b	T_h	T_c	f_l	f_h	b_l	b_h
6(a)	3	6	0.22	7	0.5	0.7	0.3	0.6
6(b)	3	6	0.08	10	0.4	0.6	0.3	0.6
6(c)	4	6	0.08	15	0.5	0.7	0.3	0.55
6(d)	3	6	0.08	7	0.45	0.7	0.1	0.5
6(e)	3	6	0.08	7	0.4	0.7	0.3	0.8
6(f)	3	6	0.08	19	0.5	0.6	0.1	0.7

Table 1. Parameter values for results in Fig. 6

The proposed line drawing system was implemented with Visual C++ and OpenGL on a Windows PC. The computation time mostly depends on the image size. For an image with the size of 512×512, it takes about 20 seconds to generate a line drawing result on a Windows PC with a 3.0 GHz Pentium processor and 2 GB memory. For each of the results in this paper, the computation time took about 7 to 30 seconds.

Different stroke textures can be applied to change the style or mood of the illustration. The attributes of the lines



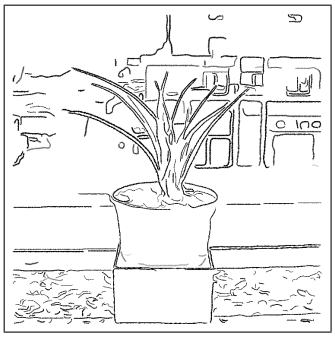
(a) (b) (c)

(d) (e) (f)

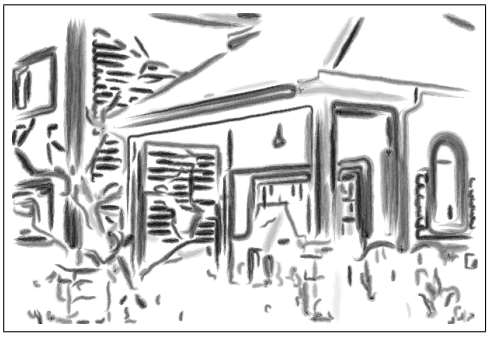
Figure 6. Line drawing results

can also be adjusted. Figs. 7(a) and 7(b) use the same input images as Figs. 6(a) and 6(b), respectively. Fig. 7(a) uses a crayon texture and keeps the background details. In Fig. 7(b), we attempted to express a wide range of darkness to imitate the oriental black ink style.

Fig. 8 compares our method with Canny edge detector [1], which is a standard edge detection technique. Fig. 8(b) is a result from Canny edge detector, where the detailed edges have been extracted. Since Canny's method strongly depends on the gradient magnitudes, it is often impossible to remove a set of strong but unimportant edges without losing other important features. See for an example the letters on the cup in Fig. 8(a). If we adjust the Canny's parameters to remove them, important features (such as the boundaries of the face and the cup) are also removed (see Fig. 8(c)). On the other hand, our method provides feature scale to effectively handle such a case. Fig. 8(d) shows the ridge points extracted from the likelihood function, and Figs. 8(e) and 8(f) are the line drawing results with different LOD control. We can choose to draw the letters with fine lines as in Fig. 8(e) or remove them as in Fig. 8(f). In addition, it is possible to emphasize the main structures by adjusting the opacity in the background (such as the face in Figs. 8(e) and 8(f)). For the abstracted result of Fig. 8(f), we set f_l as 0.65 and f_h as 1.0, while b_l and b_h are set to 0.3 and 1.0, respectively.

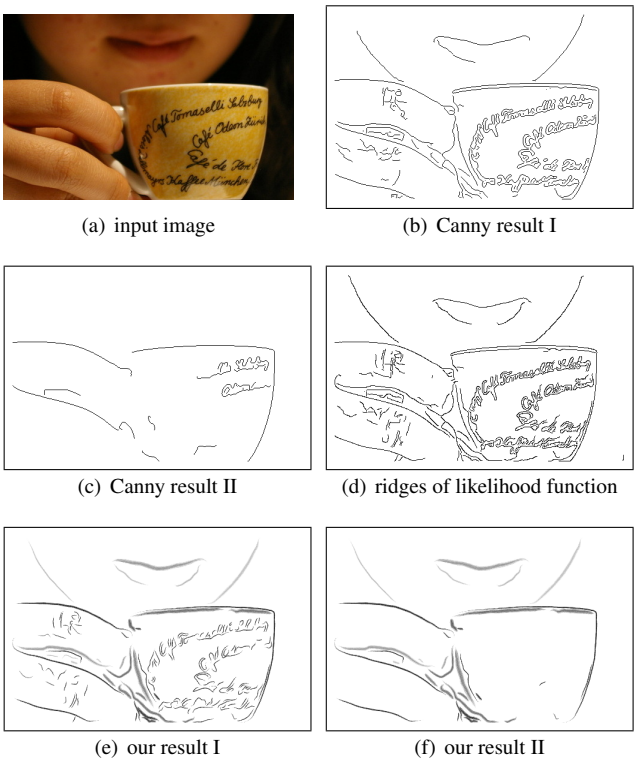


(a) crayon texture



(b) oriental black-ink texture

Figure 7. Control with different textures



(a) input image (b) Canny result I (c) Canny result II (d) ridges of likelihood function (e) our result I (f) our result II

Figure 8. Comparison with Canny edges

7. Discussion and Future work

We have presented a novel framework for automatic image-guided line drawing. Inspired by human line drawing process, our method extracts lines that are most likely to represent the genuine and meaningful shape boundaries. In addition, based on the information obtained from the line extraction process, our rendering module visualizes individual lines with different thicknesses and opacities, in order to maximize the effectiveness of visual communication. Finally, the overall style and mood of the illustration may also be controlled by proper selection of stroke texture.

In a separately developed line drawing scheme by Kang et al. [19], the DoG edge detector [10, 38] is further extended to exploit the edge flow extracted from the image, resulting in a new line construction filter called flow-based DoG (FDoG). The FDoG filter delivers improved line-drawing performance in terms of coherence enhancement and noise suppression. Compared to the FDoG filtering approach, our line construction scheme involves more sophisticated algorithms, whereas it exclusively provides control over feature selection, level-of-focus, and line style, each of which could lead to more effective visual communication.

An existing image abstraction system [6] enables interactive LOD control using an eye-tracking device for better visual communication. Our framework also provides some amount of LOD control, based on parameter adjustment with feature scale and blurriness analysis. It requires further study to support such functionality to a greater degree without the use of specialized hardware.

As discussed in Section 2, image-guided line drawing is often coupled with abstract region coloring to obtain stylized image abstraction. Our line drawing result may similarly benefit from adding colors to image regions in providing more effective visual communication.

While our line drawing framework takes into account multiple factors, such as gradient, feature scale, and blurriness, the image gradient still plays an important role in determining the level of pixel salience. As a result, it may not be easy to entirely discard, say, some strongly textured but unimportant background. We believe it would be beneficial to incorporate a texture analysis procedure to address this problem. We are currently exploring the possibility of providing control over 'stroke merging' to further enhance the stroke connectivity and reduce the number of strokes in the illustration. Also, content-based style selection could be another interesting future research topic, that is, the development of an automatic mechanism for selecting stroke style based on the image content.

Acknowledgements

We would like to thank the owners of the photographs included in this paper for kindly allowing us to use them for experiments. This research was supported in part by the ITRC support program (Game Animation Center) and Daegu Digital Industry Promotion Agency.

References

[1] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(6):679–698, November 1986.

[2] J. P. Collomosse, D. Rowntree, and P. M. Hall. Stroke surfaces: Temporally coherent non-photorealistic animations from video. *IEEE Trans. Visualization and Computer Graphics*, 11(5):540–549, 2005.

[3] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[4] C. Curtis, S. Anderson, J. Seims, K. Fleischer, and D. Salesin. Computer-generated watercolor. *ACM Computer Graphics (Proc. SIGGRAPH '97)*, pages 421–430, 1997.

[5] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive contours for conveying shape. *ACM Computer Graphics (Proc. SIGGRAPH 2003)*, pages 848–855, July 2003.

[6] D. DeCarlo and A. Santella. Stylization and abstraction of photographs. *ACM Computer Graphics (Proc. SIGGRAPH 2002)*, pages 769–776, 2002.

[7] F. Durand, V. Ostromoukhov, M. Miller, F. Duranleau, and J. Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proc. 12th Eurographics Workshop on Rendering*, pages 71–82, London, June 2001.

[8] J. Fischer, D. Bartz, and W. Strasser. Stylized augmented reality for improved immersion. In *Proc. IEEE VR*, pages 195–202, 2005.

[9] B. Gooch, G. Coombe, and P. Shirley. Artistic vision: Painterly rendering using computer vision techniques. In *Proc. Non-Photorealistic Animation and Rendering*, pages 83–90, 2002.

[10] B. Gooch, E. Reinhard, and A. Gooch. Human facial illustrations. *ACM Trans. Graphics*, 23(1):27–44, 2004.

[11] J. Hays and I. Essa. Image and video-based painterly animation. In *Proc. Non-Photorealistic Animation and Rendering*, pages 113–120, 2004.

[12] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. *ACM Computer Graphics (Proc. SIGGRAPH '98)*, pages 453–460, 1998.

[13] A. Hertzmann. Paint by relaxation. In *Proc. Computer Graphics International*, pages 47–54, 2001.

[14] A. Hertzmann and D. Zorin. Illustrating smooth surfaces. *ACM Computer Graphics (Proc. SIGGRAPH 2000)*, pages 517–526, July 2000.

[15] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte. A developer's guide to silhouette algorithms for polygonal models. *IEEE Computer Graphics and Applications*, 23(4):28–37, 2003.

[16] L. Iverson and S. Zucker. Logical/linear operators for image curves. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(10):982–996, 1995.

[17] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. Coherent stylized silhouettes. *ACM Computer Graphics (Proc. SIGGRAPH 2003)*, pages 856–861, July 2003.

[18] H. Kang, C. Chui, and U. Chakraborty. A unified scheme for adaptive stroke-based rendering. *The Visual Computer*, 22(9):814–824, 2006.

[19] H. Kang, S. Lee, and C. Chui. Coherent line drawing. In *Proc. Non-Photorealistic Animation and Rendering*, 2007.

[20] P. Litwinowicz. Processing images and video for an impressionist effect. *ACM Computer Graphics (Proc. SIGGRAPH '97)*, pages 151–158, 1997.

[21] L. Markosian, M. A. Kowalski, S. J. Trychin, L. D. Bourdev, D. Goldstein, and J. F. Hughes. Real-time nonphotorealistic rendering. *ACM Computer Graphics (Proc. SIGGRAPH '97)*, pages 415–420, 1997.

[22] D. Marr and E. C. Hildreth. Theory of edge detection. In *Proc. Royal Soc. London*, pages 187–217, 1980.

[23] P. Meer and B. Georgescu. Edge detection with embedded confidence. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23(12):1351–1365, 2001.

[24] N. J. Mitra and A. Nguyen. Estimating surface normals in noisy point cloud data. In *Proc. SCG '03: Nineteenth Annual Symposium on Computational Geometry*, pages 322–328, New York, NY, USA, 2003. ACM Press.

[25] V. Ostromoukhov. Digital facial engraving. *ACM Computer Graphics (Proc. SIGGRAPH '99)*, pages 417–424, 1999.

[26] M. Pauly, R. Keiser, and M. Gross. Multi-scale feature extraction on point-sampled surfaces. *Computer Graphics Forum (Proc. Eurographics 2003)*, 22(3):281–289, 2003.

[27] C. Rothwell, J. Mundy, W. Hoffman, and V. Nguyen. Driving vision by topology. In *Proc. International Symposium on Computer Vision*, pages 395–400, 1995.

[28] M. Salisbury, C. Anderson, D. Lischinske, and D. Salesin. Scale-dependent reproduction of pen-and-ink illustrations. *ACM Computer Graphics (Proc. SIGGRAPH '96)*, pages 461–468, 1996.

[29] M. Salisbury, S. Anderson, R. Barzel, and D. Salesin. Interactive pen-and-ink illustration. *ACM Computer Graphics (Proc. SIGGRAPH '94)*, pages 101–108, 1994.

[30] M. Salisbury, M. Wong, J. Hughes, and D. Salesin. Orientable textures for image-based pen-and-ink illustration. *ACM Computer Graphics (Proc. SIGGRAPH '97)*, pages 401–406, 1997.

[31] O. Schall, A. Belyaev, and H.-P. Seidel. Robust filtering of noisy scattered point data. In *Proc. IEEE/Eurographics Symposium on Point-Based Graphics*, pages 71–77, 2005.

[32] J. Shen and S. Castan. An optimal linear operator for step edge detection. *Graphical Models and Image Processing*, 54(2):112–133, 1992.

[33] S. Smith and J. Brady. Susan – a new approach to low-level image processing. *International Journal of Computer Vision*, 23(1):45–78, 1997.

[34] M. Sousa and J. Buchanan. Observational models of graphite pencil materials. *Computer Graphics Forum*, 19(1):27–49, 2000.

[35] M. Sousa and P. Prusinkiewicz. A few good lines: Suggestive drawing of 3D models. *Computer Graphics Forum (Proc. Eurographics 2003)*, 22(3), 2003.

[36] J. Wang, Y. Xu, H.-Y. Shum, and M. Cohen. Video tooning. *ACM Computer Graphics (Proc. SIGGRAPH 2004)*, pages 574–583, 2004.

[37] F. Wen, Q. Luan, L. Liang, Y.-Q. Xu, and H.-Y. Shum. Color sketch generation. In *Proc. Non-Photorealistic Animation and Rendering*, pages 47–54, 2006.

[38] H. Winnemöller, S. C. Olsen, and B. Gooch. Real-time video abstraction. *ACM Computer Graphics (Proc. SIGGRAPH 2006)*, pages 1221–1226, 2006.

[39] H. Xu, N. Gossett, and B. Chen. Pointworks: Abstraction and rendering of sparsely scanned outdoor environments. In *Proc. Eurographics Symposium on Rendering*, pages 45–52, 2004.