사용자 제어가 용이한 이차원 영상의 추상화된 라인 드로잉 생성

(User-Guidable Abstract Line Drawing of 2D Images)

손 민 정 [†] 이 윤 진 [††] 강 형 우 [†††] 이 승 용 [††††]
(Minjung Son) (Yunjin Lee) (Henry Kang) (Seungyong Lee)

요 약 본 논문은 이차원 영상으로부터 시각적으로 효과적인 정보를 제공할 수 있는 라인 드로잉 영상을 생성하는 방법을 제시한다. 본 방법은 기존의 단순한 에지 추출 방법과 달리 사람의 라인 드로잉 과정을 적용하여 직관적이면서 효과적인 결과를 생성하며, 크게 라인 추출, 라인 렌더링, 사용자 지시의 세 단계로 구성된다. 라인 추출 단계에서는 우도 함수를 이용하여 물체의 중요한 경계 부분을 효과적으로 예측함으로써 라인을 추출한다. 추출된 라인을 실제 렌더링하는 라인 렌더링 단계에서는, 중요한 라인과 세부 라인의 구분 및 초점의 정확도를 표현하기 위하여 라인의 특징 크기와 번짐 정도를 고려하고, 이를 다양한 스타일로 표현하기 위해 텍스쳐 형태의 라인 영상을 적용한다. 마지막으로 사용자 지시 단계에서는 자동으로 생성된 라인의 모양이나 위치를 사용자 상호작용으로 수정할 수 있다. 이 때 사용자 입력에 대한 즉각적인 반응을 위해 라인 추출의 대부분의 과정이 GPU 상에서 구현된다. 제시된 방법의 결과로 사용자는 이차원 영상으로부터 중요한 부분과 초점이 맞는 부분의 표현 및 스타일을 원하는 대로 조절한 라인 드로잉을 얻을 수 있고, 또한 이를 자유롭게 수정할 수 있다.

키워드 : 비사실적 렌더링, 스타일화, 라인 드로잉, 사용자 지시, 그래픽스하드웨어, 우도함수, 특징 크기, 번짐 정도

Abstract We present a novel scheme for 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 to generate lines effectively and intuitively. Our technique consists of three parts: line extraction, line rendering, and user guidance. In line extraction, we extract lines by estimating a likelihood function to effectively find the genuine shape boundaries. In line rendering, we consider the feature scale and the blurriness of lines with which the detail and the focus-level of lines are controlled. We also employ stroke textures to provide a variety of illustration styles. User guidance is allowed to modify the shapes and positions of lines interactively, where immediate response is provided by GPU implementation of most line extraction operations. Experimental results demonstrate that our technique generates various kinds of line drawings from 2D images enabled by the control over detail, focus, and style.

Key words : non-photorealistic rendering, stylization, line drawing, user-guidance, GPU, likelihood function, feature scale, blurriness

· This work was supported by the IT R&D program of MKE/MCST/KEIT (KI001820, Development of Computational Photography Technologies for Image and Video Contents). This work was also supported by the Engineering Research Center of Excellence Program of Korea Ministry of Education, Science and Technology(MEST) / National Research Foundation of Korea(NRF) (Grant 2010-0001721).

† 학생회원 : 포항공과대학교 컴퓨터공학과
sionson@postech.ac.kr
†† 정 회 원 : 아주대학교 미디어학부 교수
yunjin@ajou.ac.kr
††† 정 회 원 : University of Missouri - St. Louis Dept. of Mathematics and Computer Science 교수
kang@cs.umsl.edu
†††† 종신회원 : 포항공과대학교 컴퓨터공학과 교수
leesy@postech.ac.kr

논문접수 : 2009년 9월 17일
심사완료 : 2009년 12월 18일

정보과학회논문지: 시스템 및 이론 제37권 제2호(2010.4)

1. 서 론

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 content. 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 [1-8], and has proven to outperform conventional photorealistic rendering in terms of quick and accurate communication of shapes. 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 viewer. 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 a user-guidable 2D line-drawing framework that addresses these limitations and problems in 2D line drawing. The main idea is to extract lines that locally have the biggest 'likelihood' of being genuine lines that will be of interest to the viewer. 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. In addition to automatic generation of line drawings, we provide a simple user interface which enables a user to easily modify lines while the features of the input image are also preserved.

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 points of the image (see Fig. 2(c)). This is similar to the sketching process of artists where they typically apply many small strokes over the shape contours. User guidance can also be regarded as the process which adds new small line segments. 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 tones and colors.

The basic idea of the line drawing method based on likelihood function estimation was presented in our previous work [9,10]. In this paper, we made the following three improvements induced by computing likelihood functions on a GPU. First, the method runs much faster. With the previous method, line drawing took about 7 to 30 seconds, depending on the image size. Without user intervention, our new method can generate a line drawing in less than a second. Second, we take all pixels into account in the line drawing process. In the previous work, line drawing results vary with feature point sampling, where only feature points having large gradient magnitudes are used in order to accelerate the

computation. Third, we can provide effective user guidance to control the extracted lines. With fast computation on a GPU, the response for a user guidance can be provided immediately.

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:* Level of detail, level of focus, and the style of the illustration can be controlled in a way that is impossible using 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 user guidance:* A user can easily modify a line drawing with a simple user interaction. By implementing our method using GPU, changes generated by a user can be reflected in the line drawing at interactive frame rates.
- *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 [11-17], pen-and-ink illustration [18-20], pencil drawing [21,22], and engraving [23,22], 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 [24] presented an image abstraction system based on Canny edge detector [25] and mean-shift image segmentation [26]. Wang et al. [27] and Collomosse et al. [28] both applied the mean-shift segmentation to classify regions in videos. Wen et al. [29] also used mean-shift segmentation to produce a rough sketch of a scene. Fischer et al. [30] and Kang et al. [17] both employed Canny edge detector to obtain stylized augmented reality and line-based illustrations, respectively. In general edge detector is suitable for extracting lines while image segmentation is effective in pixel clustering.

Gooch et al. [31] presented a facial illustration system based on difference-of-Gaussians (DoG) filter, similar to Marr-Hildreth edge detector [32]. Winnemöller et al. [33] 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 drawings 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 [34-38]. 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 Sec. 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 line connectivity. More importantly, our line extraction process is specifically designed to enable control over various aspects of an 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 three modules: line extraction, line rendering, and user guidance (see Fig. 1). Each module is again decomposed into multiple steps which we briefly describe here.

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. 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. 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. Type of stroke texture affects the overall style or mood of the illustration. 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.

User Guidance: Our system allows for local modification of lines, such as connecting the ends of two disconnected adjacent lines, adjusting the direction of a line, and changing the connection of lines. We can achieve these by adding user inputs in the form of line segments and changing the gradients around the region where the segments are added. We can give user guidance at any time during the line drawing process.

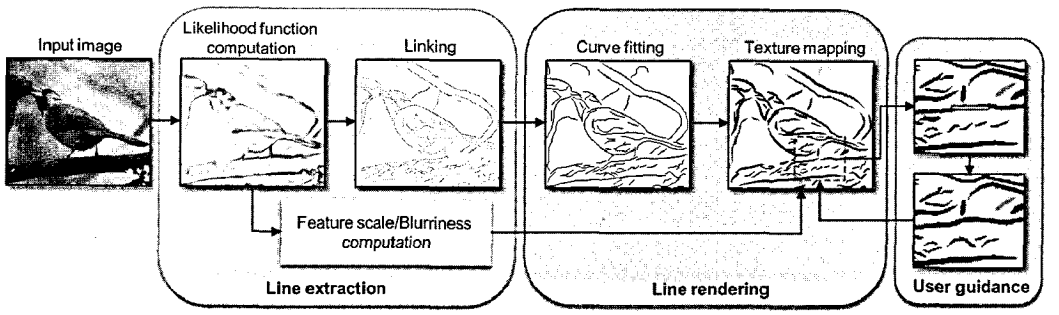


Figure 1 Overall process

Details of these three modules (line extraction, line rendering, and user guidance) will be presented in Secs. 4, 5, and 6, respectively.

4. Line Extraction

4.1 Likelihood function computation

We define a 2D point set $P = \{\mathbf{p}_1, \ldots, \mathbf{p}_N\}$ which contains the positions of all pixels on an input image $I(x,y)$. Given a 2D point set 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 P, we accumulate the local likelihood functions $L_i(\mathbf{x})$ computed at points $\mathbf{p}_i$ in 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 [39] 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(c) and 2(d), 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 [40]. 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

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

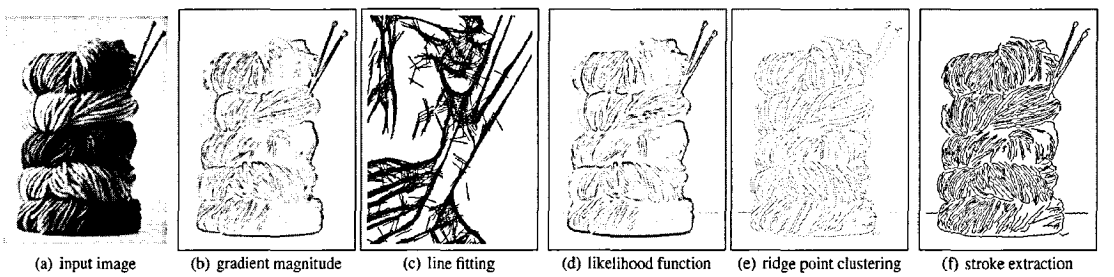
$\mathbf{g}_i$ and $\hat{g}_i$ represents a gradient vector and gradient magnitude which ranges from 0 to 1. Thus, higher weights are given to points $\mathbf{p}_j$ having strong edge features as well as similar edge directions to $\mathbf{p}_i$. This is an essential criterion for improving the quality 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.



(a) input image (b) gradient magnitude (c) line fitting (d) likelihood function (e) ridge point clustering (f) stroke extraction

Figure 2 Line extraction steps

Thus ϕ_i peaks at $\mathbf{c}_i$, then gradually decreases as it moves toward the ellipse boundary and stays zero outside.

Finally, we obtain the global likelihood function as the accumulation of local ones in the range of 0 and 1.

$$L(\mathbf{x}) = \frac{\sum_{i=1}^{N} \hat{g}_i(1 - E(\mathbf{p}_i, h))L_i(\mathbf{x}, h)}{h^2}. \quad (4)$$

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.

4.2 Linking

4.2.1 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 use a method similar to the hysteresis thresholding of Canny edge detector [25]. 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 ridge pixels with values smaller than T_l.

Ridge pixels with similar directions in the vicinity sometimes belong to different clusters because of noise between the pixels. This problem causes a long edge in the input image to split into a set of small lines. To improve ridge point clustering, we connect ridge pixels which are not in the immediate neighborhood but are close enough and have similar directions. When we visit ridge pixels with values smaller than T_l during the tracing of ridge pixels, we search ridge pixels inside the kernel centered at the last pixel $\mathbf{p}_l$ of the tracing. For each ridge pixel $\mathbf{p}_r$, we compute the cost defined by Eq. (5) which measures how close two pixels are along a single line stroke. Details of the cost will be presented in the stroke extraction step. If $\mathbf{p}_r$ has a small cost with $\mathbf{p}_l$, it means that $\mathbf{p}_r$ can belong to a line stroke including $\mathbf{p}_l$. If we find a ridge pixel $\mathbf{p}_r$ with smaller cost than a threshold (we use the kernel size in our experiments), we continue to trace by adding $\mathbf{p}_r$ and the pixels between $\mathbf{p}_r$ and $\mathbf{p}_l$ to the cluster. Fig. 2(e) shows the result of ridge point clustering.

4.2.2 Stroke extraction

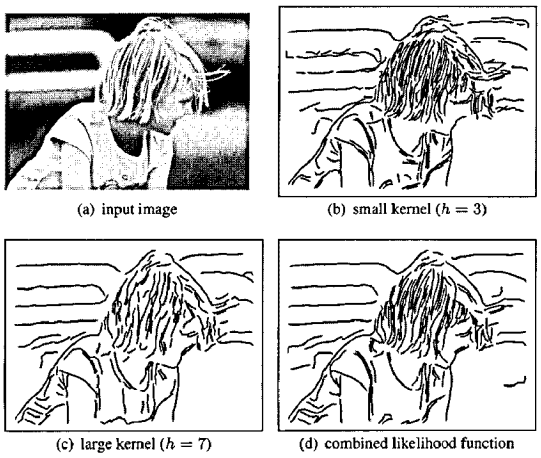
While each cluster is now a simply connected point set, it may not be smooth enough to be directly drawn as a line stroke or to be point-sampled for curve fitting. To obtain a smooth line stroke from a cluster, we first find the farthest pair of points in the cluster and then obtain the shortest path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

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

where $\delta_{i,j} = 1 - \frac{|\mathbf{n}_i \cdot \mathbf{n}_j|}{2} \cdot l_{ij}$ 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 $\mathbf{p}_j$. 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(f) shows the result of stroke extraction.

4.3 Two-level processing

In constructing the likelihood function (Sec. 4.1), kernel size plays an important role. 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



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

Figure 3 Two-level processing

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.

4.3.1 Blurriness computation

When combining the two likelihood functions, we can determine their relative weights by computing the blurriness of image regions. 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 may 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 points $\mathbf{p}_i$ in P. A blurry region has a large fitting error because strong edge points may not lie exactly 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

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

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 hardly 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. β is a scaling factor introduced to normalize the blurriness value in the range between 0 and 1. Since $E(\mathbf{p}_i,h)$ is a small value, without the scaling factor β, the blurriness would be far less than 1, which invalidates the likelihood function composition in Sec. 4.3.2. In our experiments, we use 3.5 for β, which is determined by testing various input images and kernel sizes. Blurriness b_i is clamped to have a value between 0 and 1.

There have been a variety of studies on measuring blurriness in a 2D image[41-43]. In many approaches, blurred edges are represented using Gaussian functions with a scale parameter. The optimal parameter is selected by minimizing estimation errors or finding local maxima over scale-space derivatives. In those approaches, it usually takes much time to select the optimal parameter because blurred edges should be fitted for many different scales. On the other hand, our method computes blurriness using line estimation errors. It is fast and simple because errors are measured at only two scales to define blurriness. In addition, we can obtain continuous scales of blurriness, with which we can control the properties of lines continuously during rendering.

4.3.2 Likelihood function composition

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

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

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

$$L(\mathbf{x}) = \frac{\sum_{i=1}^{N}\left((1-b_i)L_{i,d}(\mathbf{x}) + b_i L_{i,b}(\mathbf{x})\right)}{h_b^2}. \quad (9)$$

After composition, $L(\mathbf{x})$ has the value in the range of 0 and 1.

4.3.3 Feature scale computation

In addition to the blurriness, we compute for each point $\mathbf{p}_i$ another property, called feature scale f_i, from the line fitting errors. Feature scale is usually considered to be almost similar to blurriness because there are few details in blurred regions [44]. However, we must differentiate feature scale from blurriness because there can be many kinds of feature scales in unblurred regions. Feature scale defines the size of the feature in the surrounding region of a 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 image.

Around a large feature, the line fitting error remains consistent because a line can nicely appro-

ximate 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

$$f_i = \frac{1}{2}(1.0 - f_i'), \tag{10}$$

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 $E(\mathbf{p}_i,h_b) - E(\mathbf{p}_i, h_d)$. Since f_i is a value between -1 and 1, we can get the expected feature scale in the range of 0 and 1 from Eq. (10).

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 resulting 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 sample more points in a segment where normals are changing abruptly. Sampling density is controlled with the sampling cost between two consecutive points $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$ along a line stroke, defined by

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

In Eq. (11), $\delta_{i,j} = 1 - \frac{|\mathbf{n}_i \cdot \mathbf{n}_j|}{2} \cdot l_{ij+1}$ 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 at 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 curve. Type of stroke texture determines the overall style or mood of the resulting line drawing. To further imitate 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

Feature scale of a line can be computed by averaging the feature scale values of the points belonging to the line. 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 detail and line thickness, we use two thresholds f_l and f_h for the feature scale. 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 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),

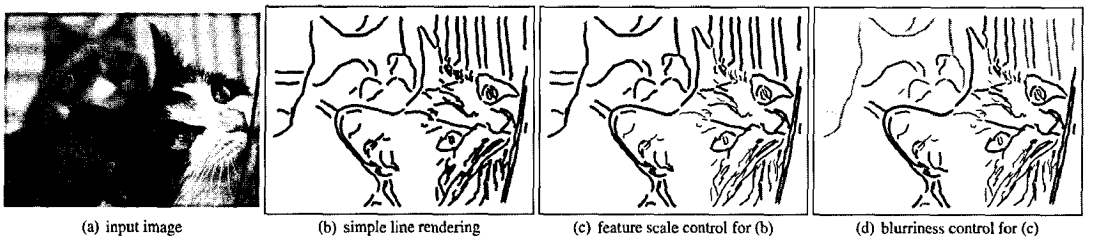


Figure 4 Line attribute control with feature scale and blurriness

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 points along the line. Blurriness of a line relates to how much the region around a line is focused or important in the image. 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 its opacity zero.

In the line rendering process, line opacity is controlled by 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. User Guidance

A user sometimes needs to partially modify line drawings to reflect their intension or improve the quality of the lines. In this section, we describe how to apply user intervention to the line drawing process.

For effective user interaction, a user interface should be intuitive and easy to use. In our method, a user simply draws a piecewise linear curve around the region where he or she wants to modify lines (Fig. 5(b), (e), and (h)). By drawing simple curves, we can connect small disconnected segments (Fig. 5(a)-(c)), adjust the direction of a line (Fig. 5(d)-(f)), or change the connectivity of lines (Fig. 5(g)-(i)). Two or more effects may be achieved together by a single input curve. In Fig. 6,

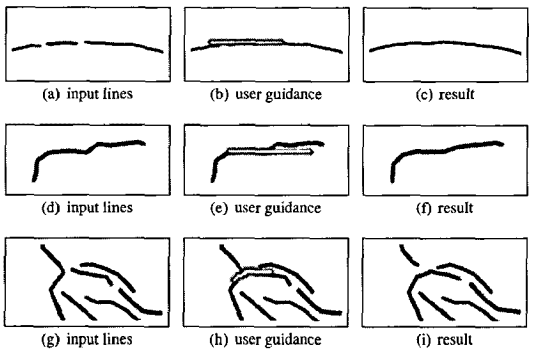


Figure 5 Various effects by user guidance. (a)-(c): connecting small adjacent line segments, (d)-(f): adjusting the direction of a line, and (g)-(i): changing the connectivity of lines

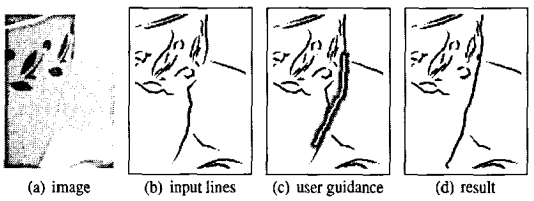


Figure 6 Example of combined effects by user guidance

we connected two silhouette lines on the right of the cup and straightened the connected line.

Since it is difficult for an unskilled user to draw lines precisely, input curves can be slightly different from what a user intends to draw. For example, in Fig. 6(c), the curve drawn by the user does not follow the boundary of the cup and its bottom part does not match the existing line. To utilize a user input as a guidance rather than a direct control on line drawing, we adjust gradient magnitudes and directions around the region where the user input is added. As a user draws a new

line on the image, the system automatically assigns weights around the line using a Gaussian-like function similar to the one used for local likelihood computation in Sec. 4.1. Weight values decrease gradually moving outward from the line and the kernel size can be changed by the user. In Fig. 5(b), (e), and (h) and Fig. 6(c), yellow lines indicate lines drawn by a user and cyan regions show the regions where weights are given. Using these weights, gradient magnitudes and directions around user-drawn lines are modified by the following equations.

$$\mathbf{g}_i = normalize(\mathbf{g}_i \cdot (1 - w_i) + d_i \cdot w_i), \quad (12)$$

$$\hat{g}_i = \min(\hat{g}_i \cdot (1 + w_i), \hat{g}_{max}). \quad (13)$$

Here, $\hat{g}_{max}$ is a maximum gradient magnitude value of the input image, w_i and d_i and are the weight and perpendicular line direction for a point $\mathbf{p}_i$, respectively. d_i is computed by fitting a line to the pixels inside the kernel centered at $\mathbf{p}_i$. We can increase the probability that a genuine line follows the user guidance by increasing gradient magnitudes for a point $\mathbf{p}_i$. Also, lines tend to be connected following the user guided direction by adjusting the gradient directions. Nevertheless, since we modify gradient magnitudes and directions obtained from the input image, we are able to preserve image features while reflecting the user's intention.

In general, an edge detection system has difficulty finding weak-contrasted edges, even the ones that a human visual system can perceive (as in Fig. 8(a)). Our interactive approach makes it easy to reveal such weak (but potentially meaningful) edges as shown in Fig. 8(c), by increasing gradient magnitudes along weak edges with user guidance.

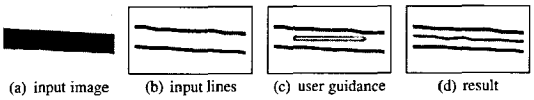


Figure 8 Addition of a new line guided by user input

Note that we could have directly adjusted the global likelihood function on the target region in order to incorporate a user-guided line modification. However, line drawing is sensitive to the global likelihood function, which means that an inaccurate user curve may end up generating a new line (if the relative influence w_i is high) or that it may be difficult to change a line with just a single user stroke (if w_i is low). In contrast, gradient magnitudes and directions affect lines indirectly. Even when a user draws a less-than-perfect line, it is fixed in the process of local likelihood function computation because we consider all gradient magnitudes and directions inside the kernel centered at each pixel.

Fig. 7 shows an example of user guidance, where several user interactions are applied. In these examples, we could make some strokes longer by connecting small strokes such as the line around the top of the lighthouse, and the lines drawn on the roof of the house. Some lines have become more apparent like the lines around the windows of the house or have been aligned along a specific direction like the lines on the upper part of the lighthouse. As shown in this figure, we can improve the quality of line drawings easily with simple user guidance.

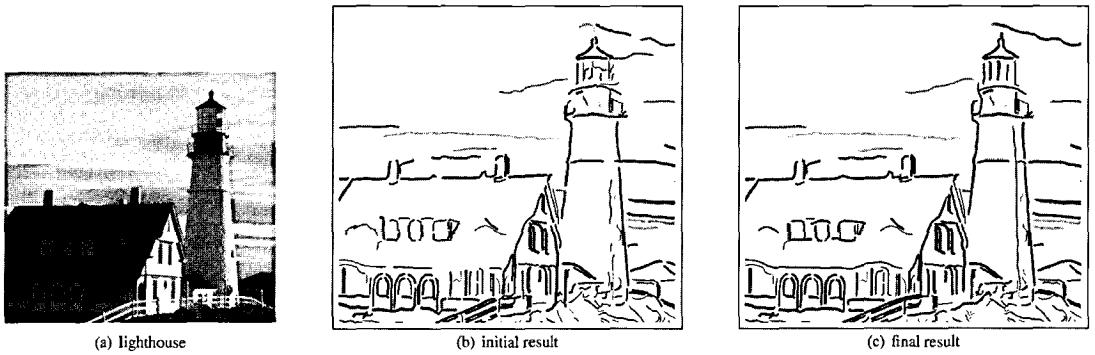


Figure 7 User guidance examples

7. Implementation on GPU

From the viewpoint of implementation, the overall process of our method can be divided into two parts, computations on a GPU and computations on a CPU. Most of the heavy computations are performed on a GPU to accelerate the process. We implemented the GPU part using shader model 3 on OpenGL platform.

The steps before ridge point clustering are suitable for pixel shader implementation since the same operations are carried out for each pixel in an input image. We use pixel shaders to implement the steps of likelihood function estimation and feature scale/blurriness computation. Ridge pixel detection is also handled using a pixel shader for finding local maximums of the likelihood function. Input and output of each shader are stored in a 32 bit texture format for more precise computation. Since the values of a likelihood function, blurriness, and feature scale are always between 0 and 1, it is not necessary to normalize the values explicitly before we convert them to textures.

Ridge point clustering is implemented on a CPU because the operations of the step are not independently performed on pixels. Clustering starts from ridge pixels and processes the neighborhood of the pixels in sequence. The stroke extraction step is also implemented on a CPU, where the operations are performed inside clusters. The line rendering step draws lines by applying texture mapping to extracted strokes. These CPU parts runs fast enough to achieve interactive frame rates because the processing in those steps is confined to ridge pixels and clusters, whose size is much smaller than all the pixels in an image.

User guidance which modifies gradient magnitudes and directions can easily be implemented on a GPU. We send weights generated by user inputs (Fig. 5(b), (e), and (h) and Eq. (13)) to a pixel shader by storing them in a 2D texture and then the pixel shader updates the corresponding gradient magnitudes and directions. Since our system generates a line drawing within a second, we can immediately see the modified result from a user guidance.

8. Experimental Results

Line drawing results in Fig. 10 are obtained from the input images in Fig. 9. These results demonstrate that the amount of detail and the level of focus are effectively controlled by our technique using the feature scale and blurriness. For Figs. 10(a), 10(b), and 10(c), a pastel texture was used. 10(d), and 10(e) were drawn with a black-ink texture.

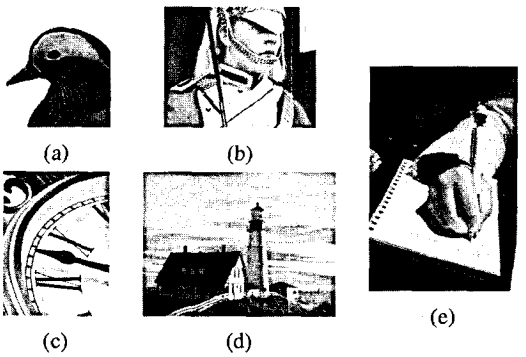


Figure 9 Input images

By default, the radii of the small and large kernels, h_d and h_b, are 3 and 7 pixels, respectively. We usually set the threshold T_h for selecting ridge pixels as 0.1. Threshold T_l is set to 0 for most cases including all the results in Fig. 10. We also provide a threshold T_c for removing very short lines. T_c is usually set to 12 pixels. For feature scale control, f_l is usually between 0.4 to 0.5, and f_h is set to 0.7 most of the time. For control using blurriness, b_l ranges between 0.1 to 0.3, and b_h between 0.5 to 0.7 in our experiments. Parameter values for the results in Fig. 10 are given in the table below. Here, for easy control of parameters, we normalize values of likelihood function, feature scale, and blurriness into the range of 0 and 1 using actual minimum and maximum values. Normalization costs are small since it is performed together with transferring data from textures to local buffers.

Each result in Fig. 10 has been improved by applying user interactions. Figs. 10(a) and (c) got

Figure 10 Line drawing results

fig	h_d	h_b	T_h	T_c	f_l	f_h	b_l	b_h
10(a)	3	8	0.15	12	0.45	0.7	0.3	0.7
10(b)	3	6	0.10	10	0.4	0.7	0.3	0.6
10(c)	3	7	0.12	15	0.4	0.7	0.1	0.6
10(d)	4	8	0.12	12	0.45	0.7	0.2	0.5
10(e)	3	7	0.10	15	0.5	0.7	0.3	0.7

only two or three interaction curves, while Figs. 10(b), (d), and (e) were modified by a user for several minutes. It is easy to improve results using our interaction method because the method is intuitive and can be executed at interactive frame rates.

The proposed line drawing system was implemented with Visual C++, OpenGL, and GLSL on a Windows PC. For an image with the size of 512× 512, our GPU implementation produces 7 frame per second for the pixel level line drawing result, and it takes less than a second to generate a final texture mapped line drawing result on a Pentium 4 PC with an nVIDIA GeForce 8800GT graphics card.

Different stroke textures can be applied to change the style or mood of the illustration. The attributes of the lines can also be adjusted. Figs. 11(a) and 11(b) use the same input images as Figs. 10(a) and 10(b), respectively. In Fig. 11(a), we attempted to remove small details and express a wide range of

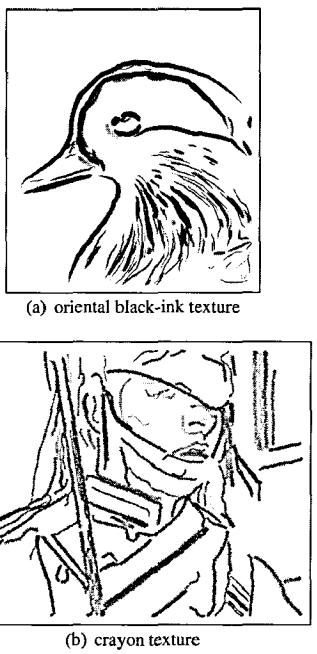
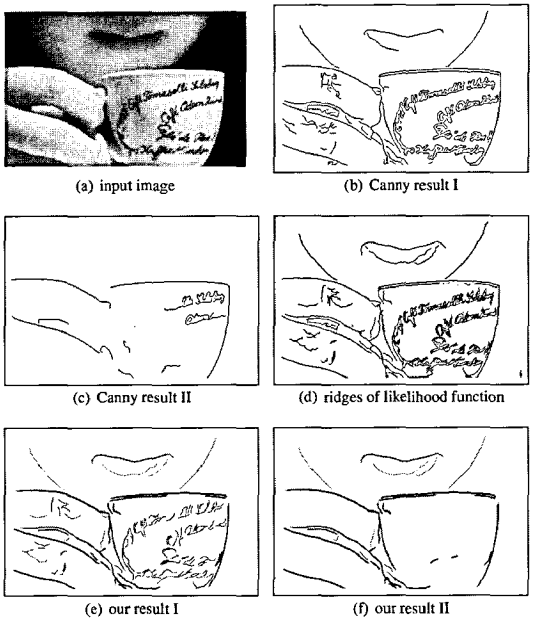


Figure 11 Control with different textures

darkness to imitate the oriental black ink style. Fig. 11(b) uses a crayon texture and removes small details.

Fig. 12 compares our method with Canny edge detector [25], which is a standard edge detection technique. Fig. 12(b) is the result of a 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. 12(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. 12(c)). In contrast, our method provides feature scale to effectively handle such a case. Fig. 12(d) shows the ridge points extracted from the likelihood function, and Figs. 12(e) and 12(f) are the line drawing results with different LOD control. We can choose to draw the letters with fine lines as in Fig. 12(e) or remove them as in Fig. 12(f). In addition, it is possible to emphasize the main structures by adjusting the opacity in the background (such as the face in Figs. 12(e) and 12(f)). For the abstracted result of Fig. 12(f), we set f_l as 0.6 and f_h as 0.7, while b_l and b_h are set to 0.2 and 0.8, respectively.



(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 12 Comparison with Canny edges

9. Discussion and Future work

We have presented a novel framework for image-guided line drawing with user interaction. Inspired by the 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. Overall style and mood of the illustration may also be controlled by proper selection of stroke texture. Finally, we can easily modify line drawings with simple user inputs.

In a separately developed line drawing scheme by Kang et al. [45], the DoG edge detector [31,33] 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 our method 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 [24] 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. This requires further study to support such functionality to a greater degree without the use of specialized hardware.

In contrast to interactive drawing systems, such as Windows painter program and Adobe illustrator, our method focuses on modifying existing lines effectively, instead of drawing lines from scratch. By adjusting gradient magnitudes and directions around the region to be modified, we can preserve image features and prevent user's unskilled lines from directly changing a line drawing. On the other hand, as a result, it is hard to add new lines on the pixels having small gradients and to generate the same line as the user's input.

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.

References

[1] 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)*, pp.415-420, 1997.

[2] A. Hertzmann and D. Zorin, "Illustrating smooth surfaces," *ACM Computer Graphics (Proc. SIGGRAPH 2000)*, pp.517-526, July 2000.

[3] 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*, vol.23, no.4, pp.28-37, 2003.

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

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

[6] M. Sousa and P. Prusinkiewicz, "A few good lines: Suggestive drawing of 3D models," *Computer Graphics Forum (Proc. Eurographics 2003)*, vol.22, no.3, 2003.

[7] M. Pauly, R. Keiser, and M. Gross, "Multi-scale feature extraction on point-sampled surfaces," *Computer Graphics Forum (Proc. Eurographics 2003)*, vol.22, no.3, pp.281-289, 2003.

[8] H. Xu, N. Gossett, and B. Chen, "Pointworks: Abstraction and rendering of sparsely scanned outdoor environments," in *Proc. Eurographics Symposium on Rendering*, pp.45-52, 2004.

[9] M. Son and S. Lee, "Line drawings from 2D images," *Journal of KIISE : Computer Systems and Theory*, vol.34, no.12, pp.656-673, 2007, in Korean.

[10] M. Son, H. Kang, Y. Lee, and S. Lee, "Abstract line drawings from 2D images," in *Proc. Pacific Graphics 2007*, pp.333-342, 2007.

[11] P. Litwinowicz, "Processing images and video for an impressionist effect," *ACM Computer Graphics (Proc. SIGGRAPH '97)*, pp.151-158, 1997.

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

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

[14] B. Gooch, G. Coombe, and P. Shirley, "Artistic vision: Painterly rendering using computer vision techniques," in *Proc. Non-Photorealistic Animation and Rendering*, pp.83-90, 2002.

[15] A. Hertzmann, "Paint by relaxation," in *Proc. Computer Graphics International*, pp.47-54, 2001.

[16] J. Hays and I. Essa, "Image and video-based painterly animation," in *Proc. Non-Photorealistic Animation and Rendering*, pp.113-120, 2004.

[17] H. Kang, C. Chui, and U. Chakraborty, "A unified scheme for adaptive stroke-based rendering," *The Visual Computer*, vol.22, no.9, pp.814-824, 2006.

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

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

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

[21] M. Sousa and J. Buchanan, "Observational models of graphite pencil materials," *Computer Graphics Forum*, vol.19, no.1, pp.27-49, 2000.

[22] 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*, London, June 2001, pp.71-82.

[23] V. Ostromoukhov, "Digital facial engraving," *ACM*

Computer Graphics (Proc. SIGGRAPH '99), pp. 417-424, 1999.

[24] D. DeCarlo and A. Santella, "Stylization and abstraction of photographs," *ACM Computer Graphics (Proc. SIGGRAPH 2002)*, pp.769-776, 2002.

[25] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.8, no.6, pp.679-698, November 1986. [Online]. Available: http://portal.acm.org/citation.cfm? id=11275

[26] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.24, no.5, pp.603-619, 2002.

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

[28] J. P. Collomosse, D. Rowntree, and P. M. Hall, "Stroke surfaces: Temporally coherent non-photo-realistic animations from video," *IEEE Trans. Visualization and Computer Graphics*, vol.11, no.5, pp.540-549, 2005.

[29] F. Wen, Q. Luan, L. Liang, Y.-Q. Xu, and H.-Y. Shum, "Color sketch generation," in *Proc. Non-Photorealistic Animation and Rendering*, pp.47-54, 2006.

[30] J. Fischer, D. Bartz, and W. Strasser, "Stylized augmented reality for improved immersion," in *Proc. IEEE VR*, pp.195-202, 2005.

[31] B. Gooch, E. Reinhard, and A. Gooch, "Human facial illustrations," *ACM Trans. Graphics*, vol.23, no.1, pp.27-44, 2004.

[32] D. Marr and E. C. Hildreth, "Theory of edge detection," in *Proc. Royal Soc. London*, pp.187-217, 1980.

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

[34] J. Shen and S. Castan, "An optimal linear operator for step edge detection," *Graphical Models and Image Processing*, vol.54, no.2, pp.112-133, 1992.

[35] C. Rothwell, J. Mundy, W. Hoffman, and V. 1 Nguyen, "Driving vision by topology," in *Proc. International Symposium on Computer Vision*, pp.395-400, 1995.

[36] L. Iverson and S. Zucker, "Logical/linear operators for image curves," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.17, no.10, pp.982-996, 1995.

[37] S. Smith and J. Brady, "Susan - a new approach to low-level image processing," *International Journal of Computer Vision*, vol.23, no.1, pp.45-78, 1997.

[38] P. Meer and B. Georgescu, "Edge detection with embedded confidence," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.23, no.12, pp. 1351-1365, 2001.

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

[40] N. J. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proc. SCG '03: Nineteenth Annual Symposium on Computational Geometry*.1em plus 0.5em minus 0.4emNew York, NY, USA: ACM Press, pp.322-328, 2003.

[41] F. Bergholm, "Edge focusing," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.9, no.6, pp.726-741, 1987.

[42] J. H. Elder and S. W. Zucker, "Local scale control for edge detection and blur estimation," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.20, no.7, pp.699-716, 1998.

[43] T. Lindeberg, "Edge detection and ridge detection with automatic scale selection," *Intl. J. of Computer Vision*, vol.30, pp.117-154, 1998.

[44] A. Orzan, A. Bousseau, P. Barla, and J. Thollot, "Structure-preserving manipulation of photographs," in *Non-Photorealistic Animation and Rendering (Proc. NPAR 2007)*, 2007, pp.103-110. [Online]. Available: http://artis.imag.fr/Publications/2007/OBBT07

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

손 민 정

2005년 2월 포항공과대학교 컴퓨터공학과(학사). 2007년 2월 포항공과대학교 컴퓨터공학과(석사). 2007년 3월~현재 포항공과대학교 컴퓨터공학과 박사과정 재학 중

이 윤 진

1999년 2월 포항공과대학교 컴퓨터공학과(학사). 2005년 8월 포항공과대학교 컴퓨터공학과(박사). 2005년 9월~2006년 5월 포항공과대학교 박사후 연구원. 2006년 5월~2007년 6월미시간 대학교 박사후 연구원. 2007년 6월~2007년 10월 포항공과대학교 박사후 연구원. 2007년 10월~2008년 2월 서울대학교 BK21 연구교수. 2008년 3월~현재 아주대학교 미디어학부 교수

강 형 우

1994년 2월 연세대학교 컴퓨터과학과(학사). 1996년 2월 한국과학기술원 전산학과(석사). 2002년 2월 한국과학기술원 전산학과(박사). 2002년 3월~2003년 7월 KAIST IERC 박사후 연구원. 2003년 8월~현재 미주리 대학 전산학과 교수

이 승 용

1988년 2월 서울대학교 계산통계학과(학사). 1990년 2월 한국과학기술원 전산학과(석사). 1995년 2월 한국과학기술원 전산학과(박사). 1995년 3월~1996년 9월 City College of New York / CUNY 연구원. 2003년 8월~2004년 7월 Max-Planck-Institut fur Informatik 방문선임연구원. 1996년 10월~현재 포항공과대학교 컴퓨터공학과 교수