# VECTORIZING LINE DRAWINGS WITH NEAR-CONSTANT LINE WIDTH

*Bin Bao* *Hongbo Fu*
School of Creative Media, City University of Hong Kong

**ABSTRACT**

Many line drawing images are composed of lines with near-constant width. Such line width information has seldom been used in the vectorization process. In this work, we show that by enforcing the near-constant line width constraint, we are able to produce visually more pleasing vectorization results. To this end, we develop a tracing-based approach, allowing dynamic validation of the line width constraint. The key here is to derive correct tracing directions, which are determined based on an automatically estimated orientation field, shape smoothness and the near-constant line width assumption. We have examined our algorithm on a variety of line drawing images with different shape and topology complexity. We show that our solution outperforms the state-of-the-art vectorization software systems including WinTopo and Adobe Illustrator, especially at regions where multiple lines meet and thus are difficult to locally distinguish from each other.

***Index Terms***— line drawings, vectorization

## 1. INTRODUCTION

The vectorization of line drawings serves as an important preprocessing step for many applications like animated construction of line drawings [1]. Although many solutions have been proposed, this problem has not been fully addressed. The existing solutions can be roughly categorized into two groups: skeletonization based and tracking based. The former typically starts with skeleton extraction, followed by the recognization of graphic primitives and the recovery of junctions. Although there exist various skeletonization methods [2, 3], thinning-based approaches are still the most popular due to their simplicity and efficiency [4]. However, such skeletonization approaches often suffer from serious artifacts at regions where multiple lines meet with each other. This problem deteriorates due to anti-aliasing, making individual lines even more difficult to distinguish at the pixel level. Figure 2 (middle right) shows a typical result by *WinTopo*, a state-of-the-art vectorization software based on a variant of the thinning algorithm proposed by Zhang and Suen [4].

Without first extracting the underlying skeleton of a line drawing image, the tracking-based approaches directly track along the black pixel areas and recover the junctions between tracked entities on the fly. The existing tracking-based methods are mostly designed for engineering drawings only and use the unique characteristics of such drawings. For example, the sparse pixel vectorization approach proposed by Dori and Liu [5] performs tracking solely in the horizontal and vertical directions. The line net global vectorization approach proposed by Song et al. [6, 7] improves previous methods by using a direction guided tracking, but considers simple shapes of lines and arcs only. Therefore, those methods are not suitable for vectorizing general line drawings with arbitrary and complex shapes (e.g., the input in Figure 2). The Live Trace tool in Adobe Illustrator is able to produce more promising tracking results. However, it tends to oversimplify the shape and often fails to capture fine details exhibited in the input image (see a representative result by Illustrator in Figure 2 (right)).

To address the above limitations, we propose a new vectorization algorithm for general lines drawings based on the key observation that the lines in many drawings are of near-constant width, since they are mainly drawn using pencil-like tools. We take a tracing-based approach, motivated by the recent approach for 3D shape recovery proposed by Li et al. [8], which uses a vector field to guide an iterative snake growing process. However, unlike their input in [8], where the interaction of individual snakes is rather simple, individual lines in our input are probably more ambiguous especially due to the anti-aliasing effect (e.g., see the horse's hoof in Figure 2). The key to address the ambiguous problem is to have correct tracing directions, which are determined based on an automatically estimated orientation field, shape smoothness and the near-constant line width constraint. We have tested our algorithm on a variety of line drawing images with different shape and topology complexity, and show that our solution produces visually more pleasing vectorization results than WinTopo and Illustrator, especially at those ambiguous regions (Figure 2).

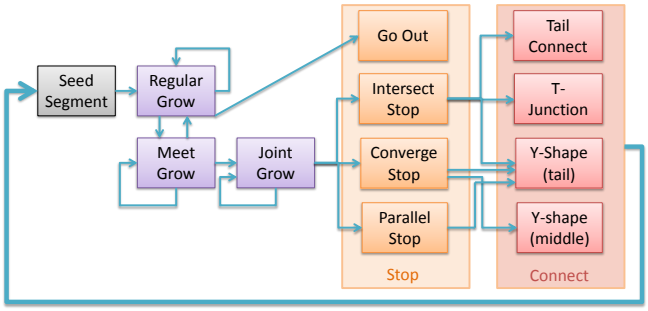


**Fig. 1**: System overview.

## 2. METHODOLOGY

Figure 1 shows an overview of our algorithm. As a preprocess, we first estimate an orientation field (Section 2.1), whose individual orientations approximate the (tangent) line directions at each pixel in an input line drawing image. Starting from a seed segment whose corresponding pixels have not been treated (Section 2.3), we grow a line basically following the orientation field but its growing direction is also influenced by the line smoothness and width constraints (Section 2.4). The latter is especially important when the current line meets the already grown lines, for which we design different strategies to faithfully recover the underlying line configurations. The current line stops growing when certain stopping conditions are satisfied (Section 2.5) and it is finally connected to the existing lines for a more complete representation (Section 2.6). The above process is repeated to extract the lines one by one until all the line pixels (whose gray values are above a certain threshold) are treated.

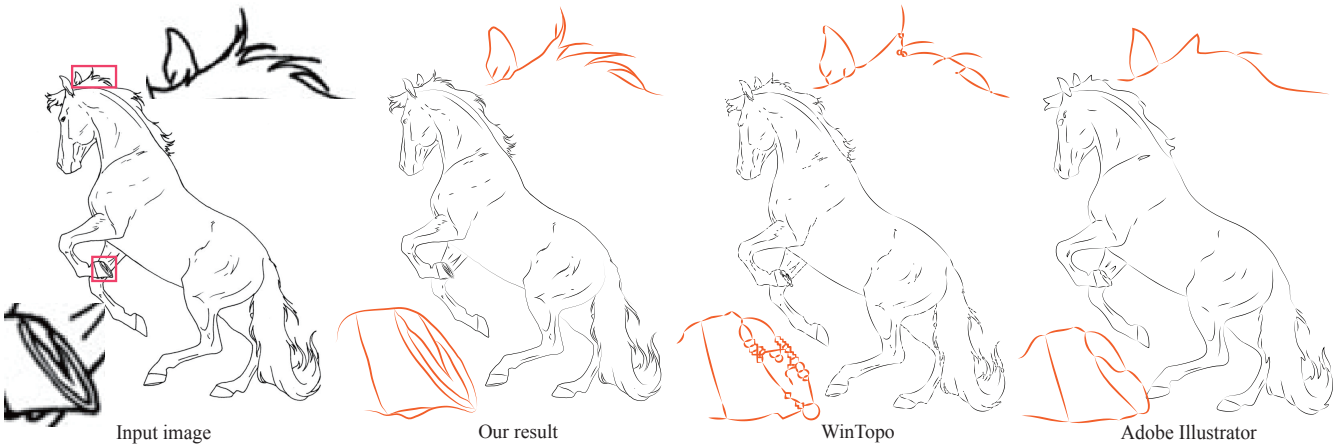


**Fig. 2**: Our algorithm produces visually more pleasing vectorization results than WinTopo and Adobe Illustrator. It is highly recommended to examine the details in a PDF reader with 400% zoom level.

### 2.1. ORIENTATION FIELD

Since we want to distinguish individual lines even in the ambiguous regions, having a robustly estimated orientation field is crucial to the success of our algorithm. We adopt the filtering approach, which is originally developed to compute a dense orientation map for captured hair images [9, 10]. We use a Canny-like edge detection filter and apply it at different angles of each pixel. The orientation of a pixel is determined by the angle at which the filter gives the highest response. Please refer to [9, 10] for more algorithm details. Let $d_{of}(p)$ denote the resulting orientation field, whose individual orientations are represented as unit vectors. Figure 3 shows an example where we overlay the orientation field on top of the input image. Note that how well the orientation field aligns with the underlying lines.

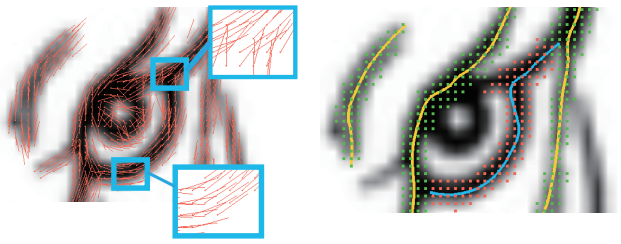

**Fig. 3**: (Left) Orientation field. (Right) Tracing lines and corresponding marked pixels. The line being traced is in blue.

### 2.2. TRACING LINES AND CROSS SECTIONS

In this subsection we introduce important geometry structures and notations that we will use in our tracing (growing) algorithm.

**Tracing line.** A tracing line is a polyline denoted as $L = \{n_i\}$, which is either an evolving line during its growing process or an already stopped line. Each node is associated with a triple of properties, denoted as $n_i = (p_i, d_i, w_i)$, where $p_i$, $d_i$, $w_i$ are the node position, tracing direction and line width respectively. For each tracing line, we mark the pixels that are covered by it (Figure 3 (right)).

**Cross section.** A cross section of a tracing line is a line segment at $p_i$, whose direction is perpendicular to $d_i$ and whose endpoints touch at the boundary of the tracing line (Figure 4). Such line segment might initially cross multiple underlying lines which either meet at the pixel level due to the anti-aliasing (Figure 4 (a)) or indeed intersect/overlap with each other (Figure 4 (b) and (c)). To get the desired cross section, we first construct and analyze the density estimate along the line segment (see the diagrams in Figure 4). For lines that meet due to the anti-aliasing effect, we can identify their cross sections of individual lines by locating the valleys in the density estimate (Figure 4 (a)). If there is no such valley and the width of the cross line segment is below a threshold (we use 1.2 $w_{avg}$, where $w_{avg}$ is the average width of lines in the input image), we consider that there is only a single line crossed, which is the simplest case. Otherwise, $p_i$ is in an *ambiguous region*, where multiple lines intersect or meet (Figure 4 (b) and (c)). In this case, we rely on the near-constant width assumption to identify the desired cross section of the current tracing line, which is part of the line segment (in orange) with width as $w_{avg}$ and starts from the nearer endpoint of the line segment to $p_i$. Let $p_i^{cross}$ and $w_i^{cross}$ denote the mid-point and width of the cross section at $p_i$.

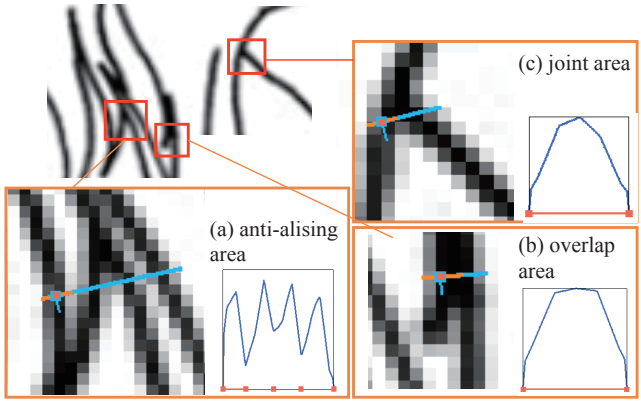


**Fig. 4**: Cross sections (in orange) at different points (in red).

### 2.3. SEED SEGMENT

To get a seed segment, we first randomly pick an unmarked pixel (i.e., not covered by any existing tracing line), denoted as $p_s$ with direction as $d_s = d_{of}(p_s)$. Then we establish a short segment starting from $p_s$. For brevity, we explain with a three-pixel segment $L_s = \{n_0, n_1, n_2\}$, with $\{n_0 = p_s - d_s, n_1 = p_s, n_2 = p_s + d_s\}$. $L_s$ is a valid seed segment only if none of $n_0$, $n_1$, $n_2$ is in an ambiguous region, *and* the ratio of the already marked pixels in $L_s$ is below a user-specified threshold, denoted as $\gamma_{seed}$, whose value will be given in Section 3.

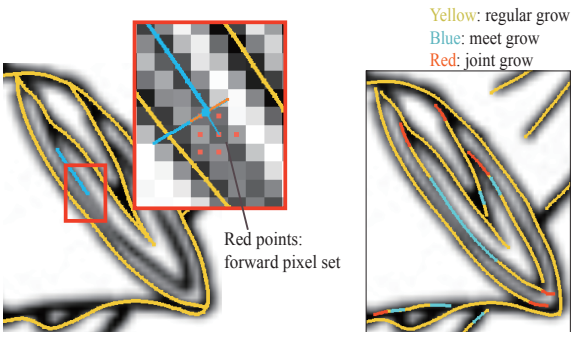


**Fig. 5**: Illustration for growing statuses.

### 2.4. TRACING BY LINE GROWING

Let $n_k = \{p_k, d_k, w_k\}$ be the currently traced node (front node) of a tracing line. Below we show how to proceed to a new node $n_{k+1}$.

**Growth direction.** As the orientation field is not always reliable, we estimate the growing direction $d^{grow}$ for $n_{k+1}$ by using both the current tail direction $d^{front}$ and $d_{of}(p_k)$. $d^{front}$ is estimated as the ending tangent direction of a cubic Bezier curve fitted to a few previously traced nodes at the front. If the angular difference between $d^{front}$ and $d_{of}(p_k)$ is below a given threshold, denoted as $\alpha$ ($\alpha_{regular}$ for regular grow and $\alpha_{meet}$ for meet grow and joint grow; $\alpha_{meet} \equiv PI/4$ in our case), we set $d^{grow} = d_{of}(p_k)$. Otherwise $d^{grow} = d^{front}$.

**Insert new node.** We first compute an evolving position as $p^{extend} = p_k + 1.0 \times d^{grow}$. However, we cannot simply use $p^{extend}$ as the new position for $n_{k+1}$, as it might deviate from the underlying line in the image. Therefore, we compute a cross section at $p^{extend}$ along $d^{grow}$ to get a possibly adjusted position as $p^{cross}$ as well as the associated line width $w^{cross}$ (Section 2.2). To ensure shape smoothness of the tracing line, we then check the angular difference between $p^{cross} - p_k$ and $d^{front}$. If the difference is below a threshold $\beta$, $n_{k+1} = \{p^{cross}, d^{grow}, w^{cross}\}$. Otherwise, $n_{k+1} = \{p^{extend}, d^{grow}, w^{cross}\}$.

**Geometry optimization.** To further enhance the smoothness of the tracing line, we perform a simple geometry optimization, which locally refine the tracing line near its evolving front. The optimization is formulated as a minimization problem: $\{q_i\} = argmin_{q_i} \sum_i (q_i - p_i)^2 + (\frac{q_{i+1}+q_{i-1}}{2} - q_i)^2$.

**Growing statuses.** The above discussion is mainly for regular grow. We need special processing when the currently tracing line meets an existing traced line. To identify if such meeting happens, we maintain a set of pixels within a small neighborhood at the front node, we call forward pixel set (Figure 5). Let $r_{mark}$ denote the ratio of marked pixels in the current forward set. If $r_{mark} > \gamma_{regular}$ ($= 0.5$ in our experiments), the status is changed from regular grow to meet grow (Figure 1).

The status of meet grow is useful for guiding the tracing line to pass through the ambiguous regions (e.g., the long blue line in Figure 5). When the evolving front leaves such ambiguous regions, we change the status back to regular grow if $r_{marked} < \gamma_{regular}$. If $r_{marked}$ is above another user-specified threshold, denoted as $\gamma_{meet}$ (typically $\gamma_{meet} > \gamma_{regular}$), the status is switched to joint grow, which will gradually stop growing for the current tracing line (Figure 1).

### 2.5. STOPPING STATUSES AND JOINT TYPES

The regular grow stops when $p_{k+1}$ does not belong to any line pixels, which is the simplest stopping status (i.e., go out). We have three stopping statuses for joint grow (Figure 6). Let $L$ and $L_{meet}$ denote the current tracing line and the met line, respectively. Since the two lines are still spatially disconnected, we use joint grow to further extend $L$ as long as possible, until they are close enough to determine the joint types between them.

**Intersect stop.** We compute a line segment which extends from the current front $p_k$ to $p_k + 3.0 \times d_k$, denoted as $S_{extend}$. If $S_{extend}$ intersects with $L_{meet}$, the status of intersect stop is activated. There are three possible joint types (Figure 7). Let $p_{intersect}$ be the intersection point between $S_{extend}$ and $L_{meet}$ and $d_{intersect} = ||p_{intersect} - p_{end}||$, where $p_{end}$ is the nearer endpoint of $L_{meet}$ to $p_{intersect}$. If $d_{intersect}$ is small enough ($< 5.0$ in our case), it is detected as tail connect. If the directions of the two lines at $p_{intersect}$ are similar and $d_{intersect}$ is not very large ($< 15.0$), it is a Y-shape at tail. Otherwise, it is T-junction joint type.

**Converge stop.** If $S_{extend}$ does not intersect with $L_{meet}$, we compute the shortest distance from $p_k$ to $L_{meet}$. If the distance is below a threshold ($< 1.0$ in our case), it is detected as converge stop and leads to a Y-shape joint type. We have two types of Y-shape, depending on the distance between $p_k$ and its nearer endpoint of $L_{meet}$.

**Parallel stop.** Finally if neither of the above conditions are satisfied, $L$ continues to grow, until it reaches some non-line pixel. It is regarded as a Y-shape at tail.

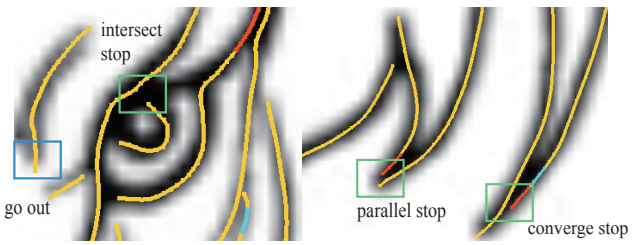


**Fig. 6**: Illustration for stop statuses.

### 2.6. CONNECTING JOINTS

After the joint types are identified, we connect $L$ and $L_{meet}$ correspondingly (Figure 7). For the case of T-junction we directly connect $L$ with $p_{intersect}$. For the Y-shape at tail, we first remove the common line segment and then use cubic Hermit splines to generate two new lines connecting to the tail. For the other type of Y-shape, we remove the joint grow segment of $L$ and generate a He1rmit curve instead. For the case of tail connect, we connect $L$ with $p_{intersect}$ and remove the redundant segment near to the endpoint of $L_{meet}$.

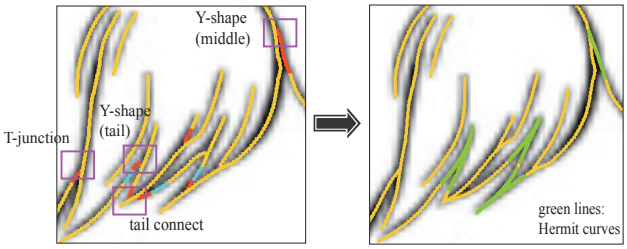


**Fig. 7**: Connecting joints of four types.

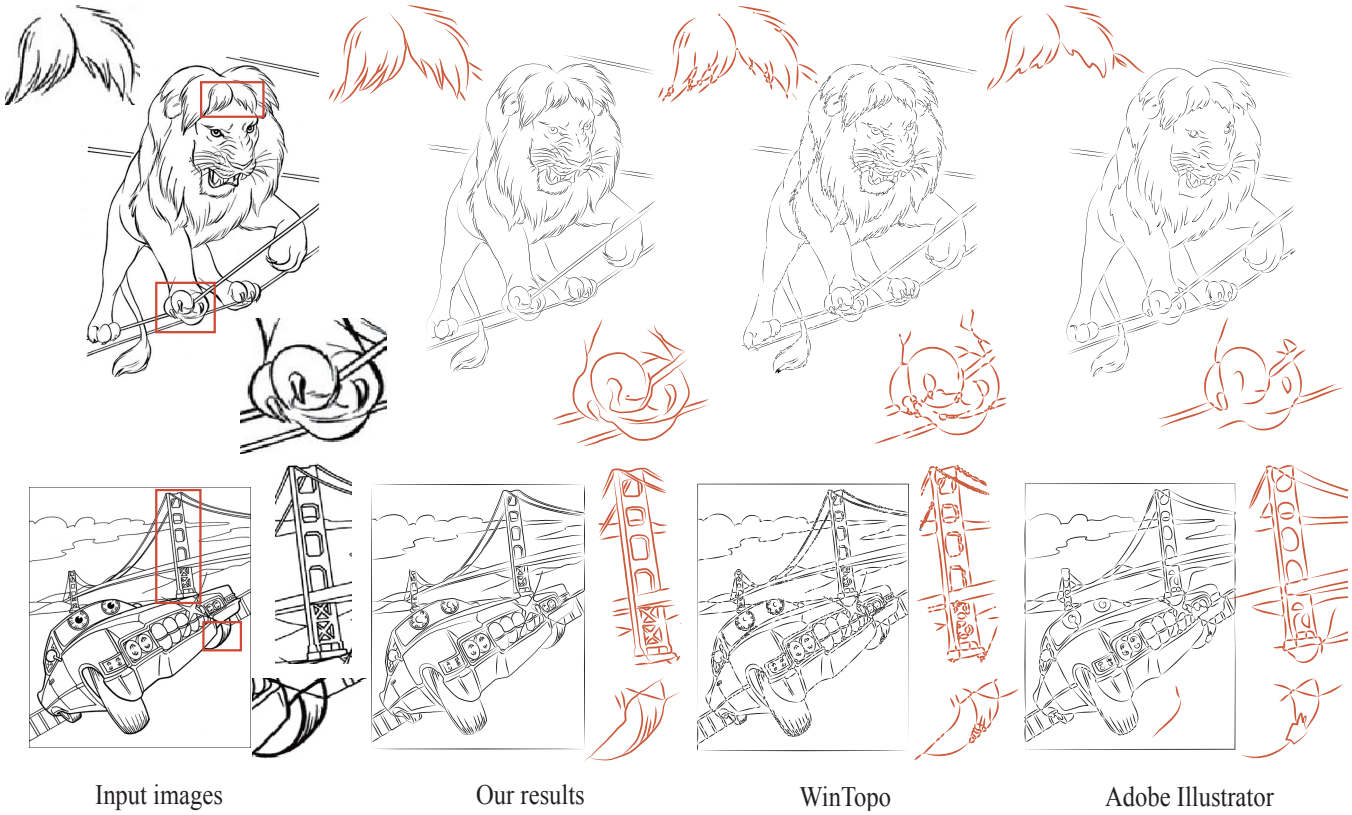


**Fig. 8**: Vectorization results. It is highly recommended to examine the details in a PDF reader with 400% zoom level.

| Examples | $\gamma_{seed}$ | $\gamma_{meet}$ | $\alpha_{regular}$ | $\beta$ |
|---|---|---|---|---|
| Horse | 0.5 | 0.8 | $PI/3$ | $PI/4$ |
| Lion | 0.4 | 0.8 | $PI/3$ | $PI/3$ |
| Car | 0.4 | 0.9 | $PI/2.7$ | $PI/2.5$ |

**Table 1**: Parameters used in different examples.

## 3. RESULTS AND DISCUSSION

We have tested our algorithm on various images of different content, style, and complexity. Figures 2 and 8 show three representative results. It is obvious to see that our method outperforms both WinTopo and Illustrator and produces visually more pleasing results. We are interested in a more objective evaluation by conducting user studies and/or computing numerical metrics (e.g., based on Haussdorf distance).

Our algorithm involves a few parameters but most of them remain fixed. Table 1 shows the parameters we adjusted for the examples in the paper by trial and error. Our current unoptimized implementation takes a few seconds for the whole vectorization (e.g., 10.482s for the car example with 412 tracing lines produced).

Although our solution captures much more details than Illustrator, it is not guaranteed that all the details are extracted, especially at small ambiguous regions. In addition, although the resulting lines are visually pleasing viewed as a whole, individual lines might still be very different from the original strokes by artist, which is unsatisfactory for applications like animation construction of line drawings.

**Acknowledgements.** We would like to thank the anonymous reviewers for their valuable comments. This work was substantially supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. CityU 113610 and CityU 123912).

## 4. REFERENCES

[1] H. Fu, S. Zhou, L. Liu, and N.J. Mitra, "Animated construction of line drawings," *ACM Trans. Graph.*, vol. 30, no. 6, 2011.

[2] J.J. Zou and H. Yan, "Cartoon image vectorization based on shape subdivision," in *CGI*, 2001, pp. 225–231.

[3] X. Hilaire and K. Tombre, "Robust and accurate vectorization of line drawings," *IEEE TPAMI*, vol. 28, pp. 890–904, 2006.

[4] TY Zhang and C.Y. Suen, "A fast parallel algorithm for thinning digital patterns," *Communications of the ACM*, vol. 27, no. 3, pp. 236–239, 1984.

[5] D. Dori and W. Liu, "Sparse pixel vectorization: An algorithm and its performance evaluation," *IEEE TPAMI*, vol. 21, no. 3, pp. 202–215, 2002.

[6] J. Song, F. Su, J. Chen, C. Tai, and S. Cai, "Line net global vectorization: an algorithm and its performance evaluation," in *CVPR*, 2000, vol. 1, pp. 383–388.

[7] J. Song, F. Su, C.L. Tai, and S. Cai, "An object-oriented progressive-simplification-based vectorization system for engineering drawings: model, algorithm, and performance," *IEEE TPAMI*, vol. 24, no. 8, pp. 1048–1060, 2002.

[8] G. Li, L. Liu, H. Zheng, and N.J. Mitra, "Analysis, reconstruction and manipulation using arterial snakes," *ACM Trans. Graph.*, vol. 29, no. 5, 2010.

[9] S. Paris, H.M. Briceño, and F.X. Sillion, "Capture of hair geometry from multiple images," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 712–719, 2004.

[10] Yichen Wei, Eyal Ofek, Long Quan, and Heung-Yeung Shum, "Modeling hair from multiple views," *ACM Trans. Graph.*, vol. 24, no. 3, pp. 816–820, 2005.