Vectorization of Thick Digital Lines Using Farey Sequence and Geometric Refinement*

Sanjoy Pratihar
Computer Science and Engineering
University Institute of Technology
University of Burdwan, Burdwan, India
sanjoy.pratihar@gmail.com

Partha Bhowmick†
Computer Science and Engineering
Indian Institute of Technology, Kharagpur
India
bhowmick@gmail.com

ABSTRACT

A novel algorithm for vectorization of line-shaped objects present in a gray-scale image is proposed. The algorithm derives the straight edges of maximal length from the object boundary using the notion of *Farey sequence*, and subsequently vectorizes them by an efficient technique of *geometric refinement*. The method would be computationally attractive when vectorization of a large database of gray-scale images is in question. Experimental results on several datasets including road maps demonstrate the usefulness, efficiency, and elegance of the proposed algorithm.

Keywords

Farey sequence, straight edge, vectorization.

1. INTRODUCTION

Automated machine recognition of objects and patterns is an active area of research. Understanding of such objects as a collection of vectors or a collection of polygons/poly-chains is of great interest in recent times, which may be seen in several works on line vectorization. For example, [9] presents a method of separating the input binary image into layers of homogeneous thickness, followed by skeletonizing each layer, and then segmenting the skeleton by random sampling. The method in [23] is based on partitioning a shape into triangles and merging of skeletons extracted from each triangle. A road vectorization technique has been proposed in [3] using a thinning procedure to get the master layer. Some works are there to extract the skeleton via block decomposition [5]. An object-oriented vectorization based on progressive simplification is proposed in [21].

Polygonal approximation is also a convenient representation of the planar object boundary. It is used as intermediate steps in various applications like multiresolution modeling [13, 20], image and video retrieval [14], shape coding [16], etc. There exists several algorithms for polygonal/poly-chain approximation of digital curves [2, 4, 18, 22]. However, to apply these algorithms for approximating (objects in) a gray-scale image, at first the edge map needs to be extracted, and then thinned, for obtaining strictly one-pixel-thick curves. We have to use edge detection operators (e.g., Prewitt/Sobel operator) or edge extraction algorithm [10] to get the edge map first. The entire procedure is, therefore, not only susceptible to the pitfalls of the adopted edge extraction algorithm and subsequent thinning, but also affected by inter-stage dependence and high runtime.

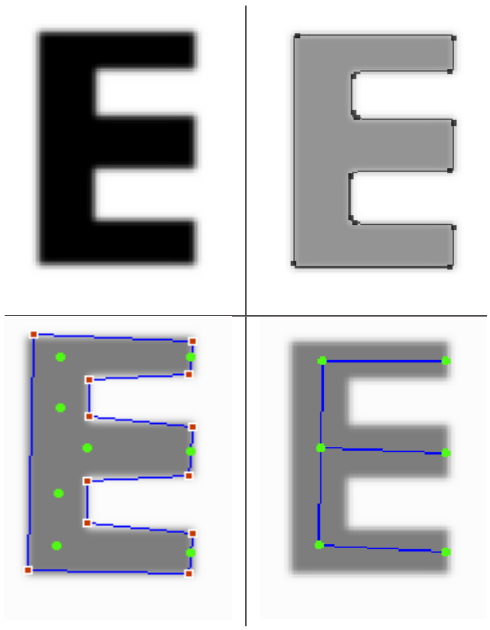


Figure 1: Vectorization by the proposed method. Top-left: Input. Top-right: Straight edges (in black). Bottom-left: Straight edges (red endpoints) after merging, and seed points (green). Bottom-right: Vectors (blue) after refinement.

In our approach, we perform polygonization/polygonal approximation of objects having "nearly uniform thick lines" in a gray-scale digital image, which is subsequently used for an efficient vectorization. The proposed algorithm is divided into the following two stages:

Stage 1: Finds the *digitally straight edges* defining the boundary of an object, and then merges the *almost collinear* edges to derive a tighter description of the object (Sec. 2).

*The work is carried out under APA project sponsored by DST (GoI), Ref. NRDMS/11/1586/2009.

†Corresponding author.

ICVGIP '10, December 12-15, 2010, Chennai, India

An edge is detected as a sequence of piecewise linear components. We extract the straight edges by inspecting the corresponding edge points one by one until there is a break in the chain because of the digital-straightness properties [12]. The ordered set of endpoints of the straight edges provides the polygonal shape of the object. The notion of *Farey sequence* and the resultant *Farey indices* [7], which capture the slope of a line segment, have been used with the *digital-geometric properties of straightness* to get the final result of polygonal approximation. Merging of extracted straight edges has been implemented using differences of indices of slope fractions in the *Augmented Farey Table ($\mathcal{F}$)*, and the collinearity of two or more straight edges is decided using addition/subtraction operations in the integer domain only.
Stage 2: From the final set of straight edges achieved in Stage 1, the *seed points* (the points using which the vectors will be drawn) are generated, which are minimized using a novel concept of *geometric refinement*, by maintaining proper association among the seed points (Sec. 3). A snapshot of our algorithm is given in Fig. 1.

2. DETECTION OF STRAIGHT EDGES

To detect the straight edges of maximal lengths, we have used an algorithm based on chain-code properties and *exponential averaging* of edge strengths (Prewitt responses) [6], as proposed in [17]. To decide the maximal straightness of an edge, certain regularity properties of *digital straightness* have been used, which can be derived from the *chord property* [11]. A curve C is digitally straight if and only if its chain codes have at most two values in $\{0, 1, 2, \ldots, 7\}$, differing by $\pm 1 (\text{mod } 8)$, and for one of these, the run-length must be 1 (Property R1). Also, if s and n be the respective singular code and non-singular code in a digital curve C, then the runs of n can have only two lengths, which are consecutive integers (Property R2).

To obtain the start point of a straight edge, each point p of the image is visited (in row-major order). If the Prewitt response at p exceeds the threshold value, T (= 100 in our experiments), and the response is a local maximum in the 8-neighborhood (8N) of p, then p is the start point, p_s. The next point on the edge commencing from p_s is obtained from the responses in 8N of p_s. The direction d_s from p_s is the chain code from p_s to its neighbor having the maximum response. In case of multiple maxima (which indicates multiple edges incident at p_s), we consider each of them, one by one, for finding the straight edges from p_s.

To get the (straight-)edge point next to any current point p, we need not apply the convolution at each neighbor (in 8N) of p with the Prewitt operator (in order to get their responses, and the maximum/maxima, thereof). Instead, in our algorithm, checking the Prewitt responses at three neighbors corresponding to three directions suffices: d, $(d+1) (\text{mod } 8)$, and $(d+7) (\text{mod } 8)$, where d is the chain code of p. For, from Property R1, no other neighbor can be the next point on the current edge. We have used effective method of exponential averaging that estimates the edge strength at an edge point using its own response and the weighted contribution of responses at the previous edge points. In order to compute the exponential average of the responses in and around a point p, we consider the responses—which have been already computed and stored—at the points preceding p up the straight edge.

Table 1: AFT $\mathcal{F}_4$.

		Denominator								
		-4	-3	-2	-1	0	1	2	3	4
N	4	19	18	16	14	13	12	10	8	7
u	3	20	19	17	15	13	11	9	7	6
m	2	22	21	19	16	13	10	7	5	4
e	1	24	23	22	19	13	7	4	3	2
r	0	25	25	25	25	–	1	1	1	1
a	-1	26	27	28	31	37	43	46	47	48
t	-2	28	29	31	34	37	40	43	45	46
o	-3	30	31	33	35	37	39	41	43	44
r	-4	31	32	34	36	37	38	40	42	43

2.1 Farey Sequence to Obtain Longer Edges

The Farey sequence F_i of order i is the sequence of completely reduced (i.e., simple/irreducible), proper, and positive fractions that have denominators less than or equal to i, and are arranged in increasing order of their values. There are several studies and related works related with Farey sequences and their indices, some of which may be seen in [7, 8, 15, 19]. As shown in this work, a Farey sequence can be of interesting and practical use to decide whether three (or more) points are collinear or not. It involves only addition, comparison and memory access, but no multiplication. Thus it helps in reducing the running time for the linearity-checking function compared to the existing procedures.

For example, for three given points $p_1(i_1, j_1)$, $p_2(i_2, j_2)$, and $p_3(i_3, j_3$ in succession, the metric $\Delta(p_1, p_2, p_3)/\max(|i_1 - i_3|, |j_1 - j_3|)$ is used to decide the deviation of p_2 from $\overline{p_1 p_3}$. However, computation of the triangle area given by $\Delta(p_1, p_2, p_3)$ involves multiplication, and is therefore computationally expensive. Such multiplications are avoided by us using Farey sequences. When we have multiple images to be processed one after another, we can compute a Farey table of an appropriate size and use it for computational optimization. As a result, the total time of polygonal approximation for all images in large database would be significantly reduced.

A Farey sequence starts with the fraction $\frac{0}{1}$ and ends with the fraction $\frac{1}{1}$. For example, the Farey sequence of orders 1 to 5 are as follows:

$$F_1 = \frac{0}{1}, \frac{1}{1}$$

$$F_2 = \frac{0}{1}, \frac{1}{2}, \frac{1}{1}$$

$$F_3 = \frac{0}{1}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{1}{1}$$

$$F_4 = \frac{0}{1}, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{1}{1}$$

$$F_5 = \frac{0}{1}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{1}{1}$$

Interestingly, each F_i can be computed from F_{i-1}. If $\frac{p}{q}$ has neighbors $\frac{a}{b}$ and $\frac{c}{d}$ in a Farey sequence, then $\frac{p}{q}$ is the mediant of $\frac{a}{b}$ and $\frac{c}{d}$. In other words, $\frac{p}{q} = \frac{a+c}{b+d}$. For example, since $\frac{3}{5}$ is the mediant of $\frac{1}{2}$ and $\frac{2}{3}$ in F_5, $\frac{3}{5}$ is obtained for F_5 by adding the corresponding numerators and denominators of $\frac{1}{2}$ and $\frac{2}{3}$ from F_4.

2.1.1 Augmented Farey Table

The original Farey sequence F_i of order i consists of all the simple, proper, positive fractions with denominators less than or equal to i. Compound fractions (that can be reduced to simple fractions of F_i), improper fractions (with numerators less than or equal to i), and negative fractions do not find any place in F_i. With simple operations, we obtain an *augmented Farey sequence* $\overline{F}_i$ from the Farey sequence F_i in order to include the above fractions as well. The augmented sequence $\overline{F}_i$ aids the linearity checking procedure while merging the end points of straight edges, which are almost collinear. For each member $\frac{a}{b}$ of F_i, we prepare a sub-list containing the equivalent compound fractions with denominators less than or equal to i. Corresponding to each $\frac{a}{b}$, a new fraction $\frac{a+a'}{b+b'}$ is computed, where $\frac{a'}{b'}$ is already a member of the sub-list corresponding to $\frac{a}{b}$, and b' is the highest denominator in the sub-list corresponding to $\frac{a}{b}$, such that $(b+b') \leq i$. This new fraction is kept in the sub-list of $\overline{F}_i$ linked to $\frac{a}{b}$. The first member $\frac{a'}{b'} = \frac{a+a}{b+b}$ of such a sub-list is obtained by adding the numerator a of $\frac{a}{b}$ with itself and the denominator b of it with itself, provided $b' \leq i$.

Since $\overline{F}_i$ is derived as stated above, it contains all positive fractions (simple and compound) with denominators less than or equal to i. Now we take mirror reflection of this list about $\frac{1}{1}$, such that in the reflected part each member is the reciprocal of its counterpart. The compound fraction in the sub-lists linked to the simple fractions are also treated in the same way, i.e., numerators become denominators, and vice versa. This reflected part is appended to the original list. Next, we again take a reflection of this enlarged list, with the signs of all denominators in the reflected part changed to $-$ve. Thus, finally we get all the fractions with $+$ve numerator and $+$ve/$-$ve denominator. Taking their positions in the list we build the *augmented Farey table*, namely $\mathcal{F}_i$, corresponding to $\overline{F}_i$, as shown in Table 1. For example, when compound fractions are included in F_4, it gets augmented to

$$\overline{F}_4 = \frac{0}{1}\left(\frac{0}{2}, \frac{0}{3}, \frac{0}{4}\right), \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\left(\frac{2}{4}\right), \frac{2}{3}, \frac{3}{4}, \frac{1}{1}\left(\frac{2}{2}, \frac{3}{3}, \frac{4}{4}\right).$$

On including the improper fractions, we get

$$\overline{F}_4 = \frac{0}{1}\left(\frac{0}{2}, \frac{0}{3}, \frac{0}{4}\right), \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\left(\frac{2}{4}\right), \frac{2}{3}, \frac{3}{4}, \frac{1}{1}\left(\frac{2}{2}, \frac{3}{3}, \frac{4}{4}\right), \frac{4}{3}, \frac{3}{2}, \frac{2}{1}\left(\frac{4}{2}\right), \frac{3}{1}, \frac{4}{1}, \frac{1}{0}\left(\frac{2}{0}, \frac{3}{0}, \frac{4}{0}\right).$$

With $+$ve numerator and $-$ve denominator, we get

$$\overline{F}_4 = \frac{4}{-1}, \frac{3}{-1}, \frac{2}{-1}\left(\frac{4}{-2}\right), \frac{3}{-2}, \frac{4}{-3}, \frac{1}{-1}\left(\frac{2}{-2}, \frac{3}{-3}, \frac{4}{-4}\right), \frac{3}{-4}, \frac{2}{-3}, \frac{1}{-2}\left(\frac{2}{-4}\right), \frac{1}{-3}, \frac{1}{-4}, \frac{0}{-1}\left(\frac{0}{-2}, \frac{0}{-3}, \frac{0}{-4}\right), \frac{0}{1}\left(\frac{0}{2}, \frac{0}{3}, \frac{0}{4}\right), \frac{1}{4}, \frac{1}{3}, \frac{1}{2}\left(\frac{2}{4}\right), \frac{2}{3}, \frac{3}{4}, \frac{1}{1}\left(\frac{2}{2}, \frac{3}{3}, \frac{4}{4}\right), \frac{4}{3}, \frac{3}{2}, \frac{2}{1}\left(\frac{4}{2}\right), \frac{3}{1}, \frac{4}{1}, \frac{1}{0}\left(\frac{2}{0}, \frac{3}{0}, \frac{4}{0}\right).$$

For other two types of fractions, i.e., fractions with $-$ve numerator and $+$ve denominator, and fractions with $-$ve numerator and $-$ve denominator, we have to take reflection of the above list $\overline{F}_4$ and then change the signs of numerators and denominators accordingly. Thus, $\overline{F}_4$ becomes a complete list of all possible fractions. For each fraction in $\overline{F}_4$, we obtain its index by accessing the table $\mathcal{F}$ (Table 1).

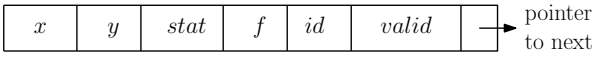


Figure 2: Data structure S for storing seed points.

2.2 Merging of Straight Edges using $\mathcal{F}$

Extraction of straight edges from a gray-scale image generates an ordered set E (endpoints of straight edges), as explained earlier. Now, in order to reduce the number of straight edges defining the boundary of the object, vertices are taken out from E, and if they are "almost collinear", then they are combined together to form a longer straight edge. If $\langle e_i, e_{i+1}, \ldots, e_j \rangle$ be a maximal (ordered) subset of straight edges that are almost collinear, then these $j-i+1$ edges are combined to a single edge. The process is repeated for all such maximal subsets in succession to obtain a reduced set of (almost) straight edges corresponding to the object boundary. There are several techniques available in the literature to replace the almost-collinear pieces by a single piece [1, 11]. We have used a novel technique using differences of indices corresponding to line slopes—which are equivalent to fractions—in the augmented Farey table, $\mathcal{F}$. Each $\mathcal{F}$-index is obtained by a single probe in $\mathcal{F}$ and the decision on linearity of three points is taken in the integer domain using addition/subtraction operation only. For a straight edge with end points $p := (x_p, y_p)$ and $q := (x_q, y_q)$, we do access the index of the fraction $\frac{y_p - y_q}{x_p - x_q}$, which is the slope of the line segment $\overline{pq}$, in $\mathcal{F}$. If two line segments L_1 and L_2 are having their respective $\mathcal{F}$-indices as f_1 and f_2, then L_1 and L_2 are merged if the difference of f_1 and f_2 is less than a threshold ϕ, which is a differential Farey index and a parameter of our algorithm.

3. VECTORIZATION

After preparing the set of straight edges, E, as discussed in Sec. 2, we get straight edges of appreciably larger length. We sort these edges in E in ascending order of their $\mathcal{F}$-indices (Farey indices in $\mathcal{F}$). So we can distinctly identify many clusters in E, where each cluster contains the edges of identical or nearly identical slopes apropos their $\mathcal{F}$-indices. These straight edges having identical or nearly identical slopes/$\mathcal{F}$-indices are used in pairs to generate some seed points, which are of interest to us for the purpose of vectorization. The process of extracting the seed points has been discussed in detail in Sec. 3.1. We store each seed point in a list S of seed points and then apply a minimization process on the list to reduce the number of seed points. Finally, depending on the association among the seed points in the reduced set, we draw the corresponding vectors. All about the data structure for storing seed points and the minimization process have been discussed in Sec. 3.1 and Sec. 3.2.

3.1 Computing and Storing Seed Points

Every pair of straight edges obtained from E, which are nearly parallel as verified from their $\mathcal{F}$-indices, is considered by us to generate seed points. Two straight edges are said to be *nearly parallel* only if their $\mathcal{F}$-indices differ by at most ϕ. We consider $\tau_{\min}$ and $\tau_{\max}$ as the respective minimum and maximum thicknesses over all thick edges in the input image.

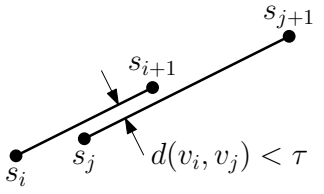
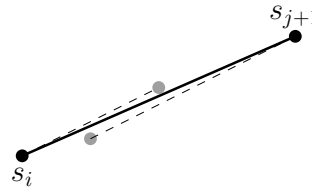
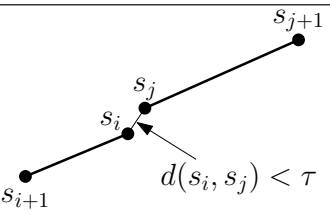
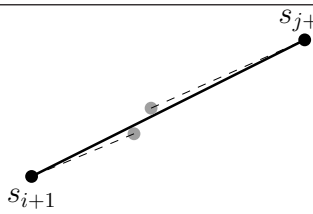
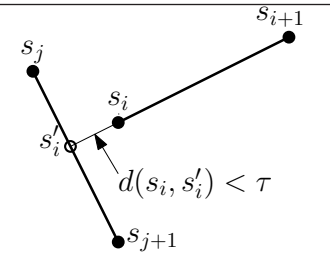
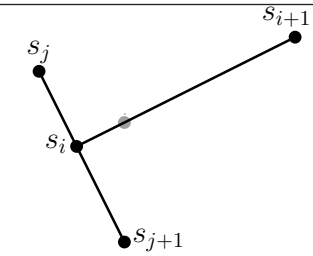
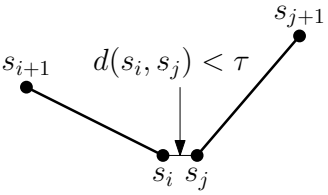
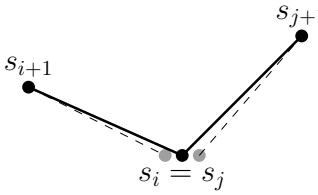
Case	Before refinement	After refinement	Changes in the data structure S
Case 1			$s_{i+1}.valid \leftarrow$ FALSE; $s_j.valid \leftarrow$ FALSE; $s_i.f \leftarrow \mathcal{F}[y_\Delta][x_\Delta]^a$; $s_{j+1}.f \leftarrow s_i.f$; $s_{j+1}.id \leftarrow s_i.id$. ——— [a] $y_\Delta = s_{j+1}.y - s_i.y$, $x_\Delta = s_{j+1}.x - s_i.x$.
Case 2			$s_i.valid \leftarrow$ FALSE; $s_j.valid \leftarrow$ FALSE; $s_{i+1}.f \leftarrow \mathcal{F}[y_\Delta][x_\Delta]^b$; $s_{j+1}.id \leftarrow s_{i+1}.id$. ——— [b] $y_\Delta = s_{j+1}.y - s_{i+1}.y$, $x_\Delta = s_{j+1}.x - s_{i+1}.x$.
Case 3			$(s_i.x, s_i.y) \leftarrow (s_i'.x, s_i'.y)$; $s_i.stat \leftarrow 1$ (closed).
Case 4			$s_i.x \leftarrow \frac{1}{2}(s_i.x + s_j.x)$; $s_i.y \leftarrow \frac{1}{2}(s_i.y + s_j.y)$; $s_i.stat \leftarrow 1$; $s_j.(x, y, stat) \leftarrow s_i.(x, y, stat)$; $s_i.f \leftarrow \mathcal{F}[y_\Delta][x_\Delta]^c$; $s_j.f \leftarrow \mathcal{F}[y'_\Delta][x'_\Delta]^c$. ——— [c] $y_\Delta = s_{i+1}.y - s_i.y$, $x_\Delta = s_{i+1}.x - s_i.x$, $y'_\Delta = s_{j+1}.y - s_j.y$, $x'_\Delta = s_{j+1}.x - s_j.x$.

Figure 3: Four cases and their geometric refinements based on Farey indices.

Hence, for two opposite and parallel straight edges, namely $e_i \in E$ and $e_j \in E$, extracted from a common thick edge of the input image, we always have $\tau_{\min} \leq d(e_i, e_j) \leq \tau_{\max}$, where $d(e_i, e_j)$ signifies the orthogonal distance between the straight edges e_i and e_j.

For two edges $e_i := (p_i, p_{i+1})$ and $e_j := (p_j, p_{j+1})$, where $p_i = (x_i, y_i)$, etc., two seed points are generated as $s(x_s, y_s)$ and $s'(x'_s, y'_s)$, where $x_s = (x_i + x_j)/2$, $y_s = (y_i + y_j)/2$, $x'_s = (x_(i+1) + x_{j+1})/2$ and $y'_s = (y_{i+1} + y_{j+1})/2$, provided the following conditions are satisfied.

(i) $\mathcal{F}$-indices of e_i and e_j differ by at most ϕ;

(ii) $\tau_{\min} \leq d(e_i, e_j) \leq \tau_{\max}$;

(iii) the projection of e_i onto e_j is non-empty.

The seed vector generated corresponding to the seeds s and s' is then given by $\overline{ss'}$.

We store the seed points one by one in a list S whose structure is shown in Fig. 2. For each seed vector, two seed points are stored. Attributes stored in S corresponding to a seed point s, which was formed from two edge points, namely p_i and p_j, are as follows: (1) x and y coordinates of s. (2) status of s; if the two mother points (i.e., p_i and p_j) corresponding to s are the end points of two straight edges e_i and e_j of E, then its status is $stat = 1$ (*closed*); otherwise, $stat = 0$ (*open*). (3) $\mathcal{F}$-index of s, given by f, which is the $\mathcal{F}$-index of the seed vector $\overline{ss'}$. (4) *id* of the seed vector $\overline{ss'}$; note that, s' has also the same id, and each seed vector gets a unique id. (5) boolean flag *valid* that says whether s is finally selected; if $valid =$ TRUE, then s is there in the final solution. Initially, for all seeds, $valid =$ TRUE; and during the refinement some of them have *valid* set as FALSE. If $valid =$ FALSE for some seed, then it is not considered further for vectorization.

3.2 Geometric Refinement

The parameter $\tau = \tau_{\min}$ is used by us to analyze the seed points during geometric refinement. This is required to minimize the number of seed points. A minimal number of seed points, in turn, leads to an efficient vectorization in terms of a minimal set of seed vectors.

The algorithm starts with seed points in S, some of which are open and some are closed. Initially no vectors are drawn. A demonstration on a typical figure has been given in Sec. 3.3.

Table 2: Results of vectorization for some images.

Image	#Corners	# Seed points (initial)	#vectors (final)	CPU time (seconds)
'A'	10	9	3	0.0504
'F'	11	7	3	0.0557
'M'	12	7	4	0.0552
'E'	12	8	4	0.0555
'curve'	33	21	4	0.1099
'bars'	24	20	3	0.1085
'road map'	145	108	23	0.2849

In the process, finally the vector $\overline{s_i s_j}$ connecting two seed points s_i and s_j is drawn only if $s_i.valid = s_j.valid = \text{TRUE}$ and $s_i.stat = s_j.stat = 1$ and $s_i.id = s_j.id$. In the process of refinement of seed points, four different cases arise, which are as follows (illustrated in Fig. 3).

Case 1 ***(Parallel and overlapping)*****:** If two seed vectors, namely $v_i := \overline{s_i s_{i+1}}$ and $v_j := \overline{s_j s_{j+1}}$, are approximately parallel—verified from their respective $\mathcal{F}$-indices—as explained earlier, and they are close enough having overlapping projections, then they are replaced by an appropriate single vector. Formally speaking, if $|s_i.f - s_j.f| \leq \phi$ *and* $d(v_i, v_j) < \tau$ *and* $\widetilde{v_i} \cap v_j \neq \emptyset$ ($\widetilde{v_i}$ denotes the projection of v_i on v_j), then we first apply a lexicographic sorting on $\{s_i, s_{i+1}, s_j, s_{j+1}\}$ using their x- and y-coordinates; only the lexicographically smallest and the lexicographically largest points are accepted, and the other two points are rejected. Accordingly, necessary modifications of the attributes of the four seed points defining the seed vectors v_i and v_j are made in their respective nodes of S, as shown in Fig. 3.

Case 2 ***(Parallel and non-overlapping)*****:** If a seed point s_i lies close to another seed point s_j, such that $s_i.stat = s_j.stat = 0$ *and* $s_i.valid = s_j.valid = \text{TRUE}$ *and* $s_i.id \neq s_j.id$ *and* $|s_i.f - s_j.f| \leq \phi$, then we set the condition $C_1 = \text{TRUE}$; otherwise, $C_1 = \text{FALSE}$. If $C_1 = \text{TRUE}$, then we read the $\mathcal{F}$-index f_{ij} of the vector $\overline{s_i s_j}$; and if $|s_i.f - f_{ij}| \leq \phi$, then we set another condition $C_2 = \text{TRUE}$; otherwise, $C_2 = \text{FALSE}$. If both C_1 and C_2 are TRUE, then we invalidate/reject s_i and s_j, and modify the attributes corresponding to the new seed vector $\overline{s_{i+1} s_{j+1}}$, as shown in Fig. 3.

Case 3 ***(Non-parallel and distant seeds)*****:** For a seed point s_i, if there is no other seed point in its τ-neighborhood, then we search for its nearest seed vector (in the set of seed vectors formed so far), namely $v_j := \overline{s_j s_{j+1}}$, and produce the vector $v_{i+1} := \overline{s_{i+1} s_i}$ to meet v_j at s_i'. If s_i' lies between s_j and s_{j+1}, then we read the $\mathcal{F}$-index f_i' of the vector $v_i' := \overline{s_i' s_{i+1}}$; if $|s_i.f - f_i'| \leq \phi$, then we translate s_i to s_i' and declare the status of s_i to be *closed*. Related attributes of s_i and s_{i+1} in S are updated accordingly (Fig. 3).

Case 4 ***(Non-parallel and close seeds)*****:** For a pair of seed points s_i and s_j, one lying within τ-neighborhood of another, if *status* of both the points are *open*, and both the points are *valid*, and if $s_i.id \neq s_j.id$, then we first determine C_1 and C_2, as discussed in Case 2. If $C_2 = \text{FALSE}$, which signifies that the corresponding seed vectors are not parallel, then we make s_i and s_j to be coincident with the point defined by the mean values of their coordinates, and hence their other attributes are also modified accordingly (Fig. 3).

3.3 Demonstration

Figure 4 demonstrates how the seed points are generated for the image 'E'. As explained in Sec. 3.1, straight edges

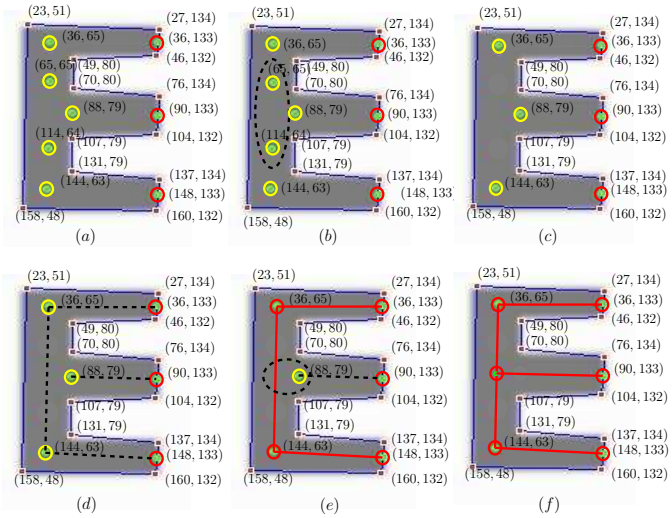


Figure 4: Demo of our algorithm on the image 'E'.

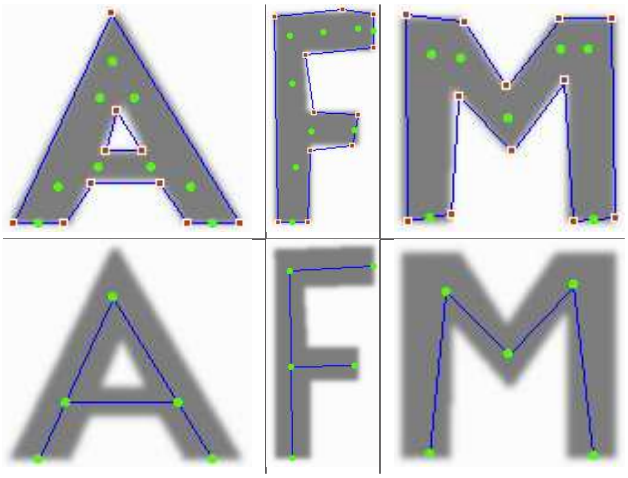


Figure 5: Experimental results for images 'A', 'F', and 'M'. Top row: Corners and seed points (ϕ = 3000). Bottom row: Vectors.

taken in pairs from E to generate these seed points. For example, $(36, 133)$ is generated by taking the respective means of x- and y-coordinates of the seeds $(27, 134)$ and $(46, 132)$, which are the endpoints of two straight edges in E. Another two endpoints of the same pair of edges generate another seed point, which is $(36, 65)$. Other seed seeds are also generated in a similar way.

When a seed is generated, its attributes $(f, stat, valid, id)$, are stored in S (Sec. 3.1). A seed encircled by a yellow circle in Fig. 4 is *open*(= 0) and that by a red circle is *closed*(= 1). Note that, both $(36, 65)$ and $(144, 63)$ will be generated twice, with different f and id values. For example, $\overline{(65, 65)(144, 63)}$ is one vector and $\overline{(36, 65)(114, 64)}$ is another. Because of Case 1, the seeds $(65, 65)$ and $(114, 64)$ get $valid = \text{FALSE}$. But the seeds $(36, 65)$ and $(144, 63)$ remain *open* with $vaild = \text{TRUE}$. In the next phase, $s_i = (36, 65)$ and $s_j = (36, 65)$ are from two different vectors, $\overline{(36, 65)(144, 63)}$ and $\overline{(36, 65)(36, 133)}$. By Case 4, both the seeds s_i and s_j merge to the same seed $(36, 65)$, which be-

comes *closed*. For similar reasons, $(144, 63)$ becomes *closed* as it is incident to two seed vectors. Now we have some vectors with both of its defining seeds as *closed*. On considering these vectors, we get the final solution of vectorization.

4. EXPERIMENTS AND RESULTS

We have implemented the algorithm in C in Linux Fedora Release 7, Kernel version 2.6.21.1.3194.fc7, Dual Intel Xeon Processor 2.8 GHz, 800 MHz FSB. To extract the straight edges using Prewitt responses, we have taken the threshold $T = 100$. Some of our experimental results have been given in Figs. 6 and 7, and the summary in Table 2, which demonstrate the strength and robustness of our algorithm.

To reduce the number of edges by merging the "almost collinear" edges, we have tested for various ϕ (Sec. 2). Table 1 shows the AFT, $\mathcal{F}_4$, of Order 4 where 48 possible slopes exist. As the order i of $\mathcal{F}_i$ goes high, the number of slopes increases appreciably. For example, for $i = 100$, total number of fractions in $\mathcal{F}_{100}$ is 24352. These 24352 slopes/fractions divide the space of 360^o into 24352 divisions. Theoretically, all the divisions will not be of equal degree. But as the number of divisions is quite large, we have assumed that the angular difference of two consecutive slopes remains almost same if the difference of $\mathcal{F}$-indices corresponding to their slopes (i.e., $|f_1 - f_2|$) remains unchanged. For $i = 200$, there exist 95876 slope vectors, which makes each division $= 0.003679^o$. Hence, $\phi = 1000$, 2000, and 4000, for $i = 200$, yield tolerances of 3.679^o, 7.356^o, and 14.716^o, respectively. Evidently, for a change in i, the significance of ϕ changes. In our implementation we have taken $i = 200$ and $\phi = 3000$.

5. CONCLUSION AND FUTURE WORK

Vectorization gives a topological description of an image and hence used in many application areas of computer vision (Sec. 1). In our work, we have first detected the straight edges defining the boundary of the object, and then we have applied a merging procedure based on a novel idea of *augmented Farey sequence*, so that we get a description of the underlying objects based on their corners and incident edges. Endpoints of the polygonal boundary, thus obtained, contribute information about positions of the seed points. As the distance between the seed-generating points plays an important role, the proposed *geometric refinement* yields the desired vectorization.

As a future work, we would like to extend this to obtain the topological description of a complex structural data (e.g., road maps) using some efficient geometric data structure, such as the *doubly connected edge list*. Since it abstracts a planar graph, along with their topological interrelations, the resultant output of vectorization would be a promising area of research in related applications.

6. REFERENCES

[1] P. Bhowmick and B. B. Bhattacharya. Fast polygonal approximation of digital curves using relaxed straightness properties. *IEEE Trans. PAMI*, 29(9):1590–1602, 2007.

[2] T. C. Chen and K. L. Chung. A new randomized algorithm for detecting lines. *Real Time Imaging*, 7:473–481, 2001.

[3] Y. Chiang and C.A. Knoblock. Automatic extraction of road intersection position, connectivity, and orientations from raster maps. In *Proc. ACM SIGSPATIAL GIS*, pages 1–10, 2008.

[4] S. Climer and S. K. Bhatia. Local lines: A linear time line detector. *PRL*, 24:2291–2300, 2003.

[5] K. Fan *et al.* A new vectorization-based approach to the skeletonization of binary images. In *Proc. ICDAR 1995*, pages 627–630.

[6] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Addison-Wesley, California, 1993.

[7] R. Graham *et al. Concrete Mathematics*. Addison-Wesley, London, UK, 1994.

[8] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 1968.

[9] X. Hilaire and K. Tombre. Robust and accurate vectorization of line drawings. *IEEE Trans. PAMI*, 28(6):890–904, 2006.

[10] J.Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 8:679–714, 1986.

[11] R. Klette and A. Rosenfeld. *Digital Geometry: Geometric Methods for Digital Picture Analysis*. Morgan Kaufmann, San Francisco, 2004.

[12] R. Klette and A. Rosenfeld. Digital straightness: A review. *Discrete Applied Mathematics*, 139(1-3):197–230, 2004.

[13] J. M. DeHaemer and M. J. Zyda. Simplification of objects rendered by polygonal approximations. *Computers and Graphics*, 15(2):175–184, 1991.

[14] F. Mokhtarian and F. Mohanna. Content-based video database retrieval through robust corner tracking. In *Proc. IEEE Workshop Multimedia Sig. Proc.*, pages 224–228, 2002.

[15] E. H. Neville. *The Farey Series of Order 1025*. Cambridge University Press, 1950.

[16] K. J. O'Connell. Object-adaptive vertex-based shape coding method. *IEEE Trans. CSVT*, 7:251–255, 1997.

[17] S. Pratihar and P. Bhowmick. A thinning-free algorithm for straight edge detection in a gray-scale image. In *Proc. ICAPR*, pages 341–344, 2009.

[18] P. L. Rosin. Techniques for assessing polygonal approximation of curves. *IEEE Trans. PAMI*, 19(6):659–666, 1997.

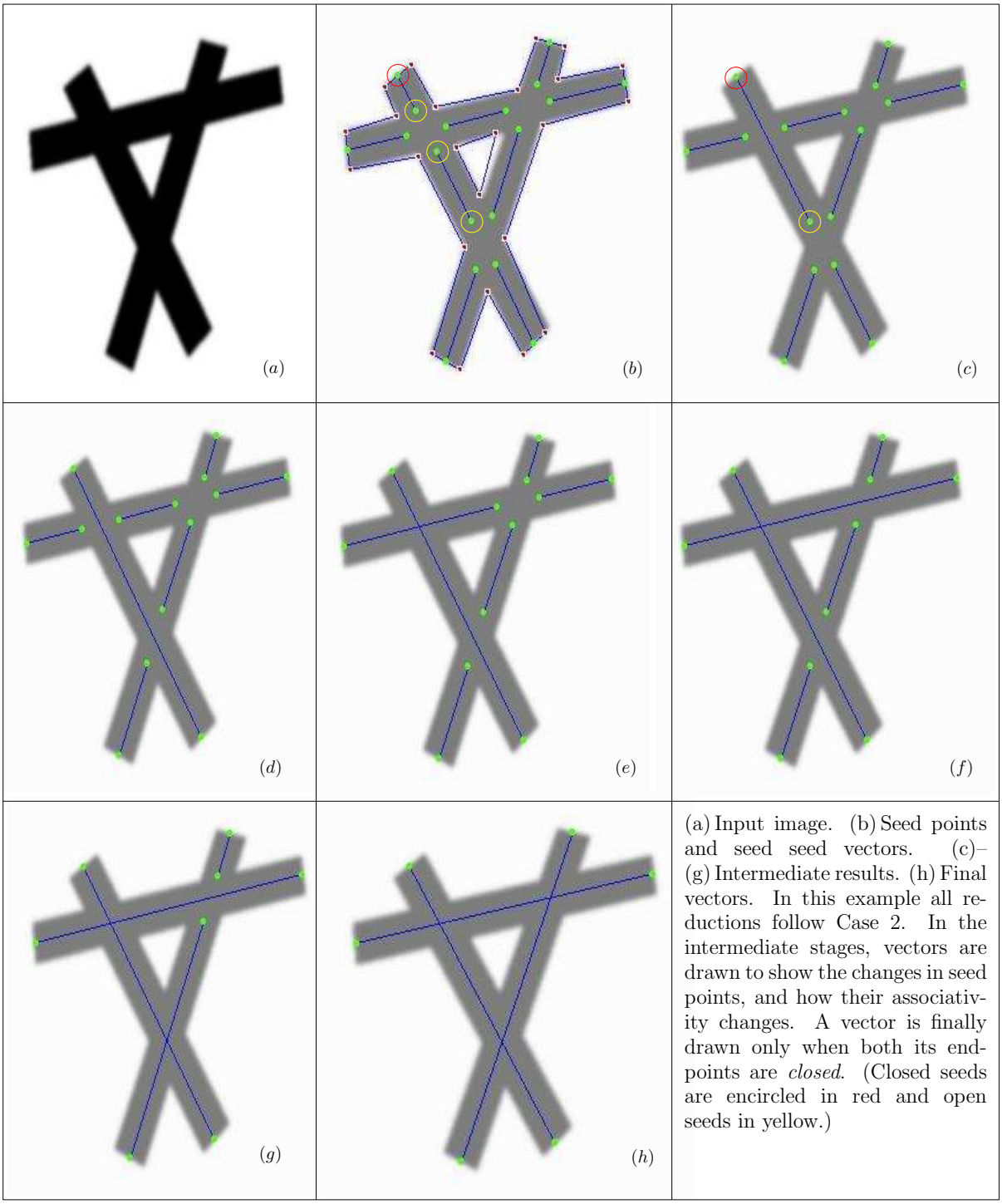
[19] M. Schroeder. *Fractions: Continued, Egyptian and Farey (Chapter 5). In Number Theory in Science and Communication.*, vol. 7, Springer, 2006.

[20] P. Shirley and A. A. Tuchman. A polygonal approximation to direct scalar volume rendering. *ACM SIGGRAPH*, 24(5):63–70, 1990.

[21] J. Song *et al.* An object-oriented progressive-simplification-based vectorization system for engineering drawings: Model, algorithm, and performance. *IEEE Trans. PAMI*, 24(8):1048–1060, 2002.

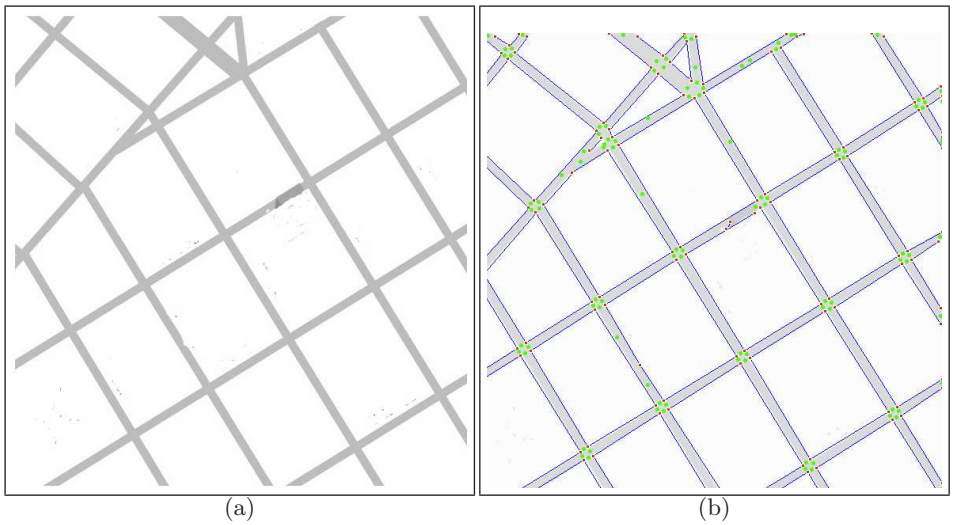
[22] C.-H. Teh and R. T. Chin. On the detection of dominant points on digital curves. *IEEE Trans. PAMI*, 2(8):859–872, 1989.

[23] J. Zou and H. Yan. Line image vectorization based on shape partitioning and merging. In *Proc. ICPR 2000*, volume 3, pages 994–997, 2000.

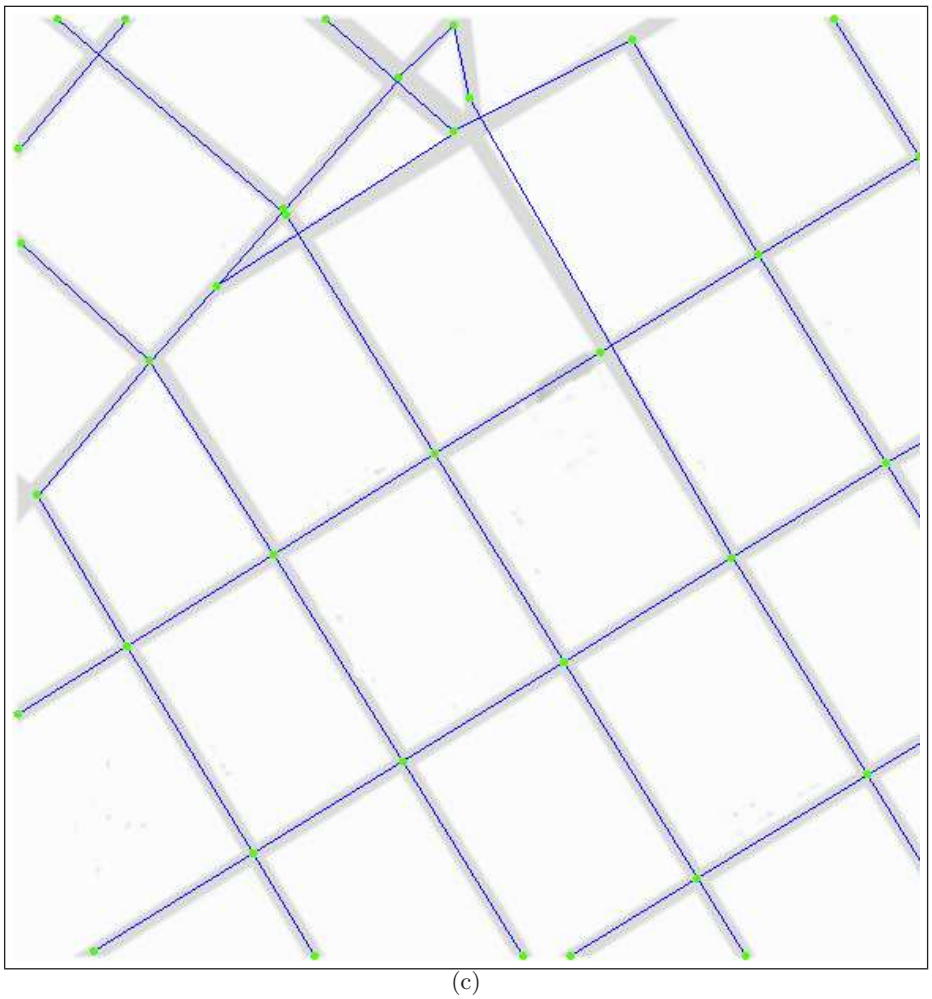


(a) Input image. (b) Seed points and seed seed vectors. (c)–(g) Intermediate results. (h) Final vectors. In this example all reductions follow Case 2. In the intermediate stages, vectors are drawn to show the changes in seed points, and how their associativity changes. A vector is finally drawn only when both its endpoints are *closed*. (Closed seeds are encircled in red and open seeds in yellow.)

Figure 6: Step-by-step results for the test image 'bars'.



(a) (b)



(c)

Figure 7: Road map vectorization: (a) Input gray-scale image. (b) Corners (red), straight edges (blue), and initial set of seed points (green). (c) Final seed points (green) and vectors (blue).