Algorithmic Evaluation of Line Detection Problem

Tetsuo Asano
School of Information Science,
JAIST
(Japan Advanced Institute of Science and Technology),
Asahidai, Tatsunokuchi, Ishikawa,
923-1292 JAPAN.
TEL: 0761-51-1295 (direct)
email:t-asano@jaist.ac.jp

Abstract

The Hough transform is a well-established scheme for detecting digital line components in a binary edge image. A key to its success in practice is the notion of voting on an accumulator array in the parameter plane. This paper discusses computational limitation of such voting-based schema under the constraint that all possible line components in a given image must be reported. Various improvements are presented based on algorithmic techniques and data structures. New schema with less computation time and working space based on totally different ideas are also proposed with some experimental results.

Index Terms — computer vision, computational geometry, duality transform, Hough transform, topological walk.

浅野 哲夫
北陸先端科学技術大学院大学情報科学研究科,
923-1292 石川県能美郡辰口町旭台 1-1
TEL: 0761-51-1295 (direct)
email:t-asano@jaist.ac.jp

1 Introduction

One of the most fundamental tasks in pattern recognition is to detect lines and curves from an image. It is often done after some appropriate edge detection process that results in a binary image in which pixels are classified into edge points (pixels) and non-edge points. Up to now a great number of algorithms have been proposed under the name of Hough Transform [7, 8]. Most of them are based on the voting technique on a subdivided parameter plane [4, 5, 9, 10, 12, 13, 14].

The basic idea of the voting technique is as follows: When a line passing through an edge point is parameterized by the angle θ and the distance ρ to the line from the origin, the edge point is mapped to a sinusoidal curve in the $\rho\theta$-parameter plane. The intersection between two such curves corresponds to the line passing through the edge points. Thus, the problem is to compute busy intersections among a number of curves. Since edge points are located on integer grids, it rarely happens that many edge points lie exactly on a line. So, the goal should be to detect a set of edge points which lie roughly on a line. For this purpose the parameter plane is partitioned into small regions called buckets or cells by axis-parallel lines, and for each such bucket it is counted how often those lines pass through the bucket region. If the number of such lines exceeds a predetermined threshold, then a digital line component corresponding to the bucket region is reported.

In computational geometry the dual transform between points and lines is more common than the above-described transform. It transforms a point (a, b) into a line $y = ax + b$ and a line $y = -cx+d$ into a point (c, d). Again, an intersection between two lines for two edge points in the original plane corresponds to the line in the dual plane which passes through these two edge points. Thus, we can detect all digital line components by enumerating all intersections at which many lines intersect.

This voting technique is easily programmed and also easily tuned for practical use. This is why the Hough transform is commonly used. However, there are many problems to be resolved. For example, an optimal way of partitioning the parameter plane is somewhat strange and not good for efficient computation.

In this paper we analyze the voting technique based on the Hough transform from a standpoint of computational complexity following a mathematical definition of digital line components to be detected. We also present a new algorithm for detecting line segments with two endpoints considering their point density with some analysis of computational complexities.

2 Standard Hough Transform

The Hough transform is commonly used in computer vision for detecting lines and curves in a binary edge image after some edge-detection process. Although edge detection is very important, we do not want to discuss which algorithm for edge detection is appropriate for our purpose.

It is a mapping from the image plane to the parameter plane. More precisely, any line passing through an edge point (x_i, y_i) in the image plane is characterized by two parameters,

the distance parameter ρ and the angle descriptor θ where ρ is the perpendicular distance from the origin to the line and θ is the angle the perpendicular makes with the x axis (refer to Figure 1).

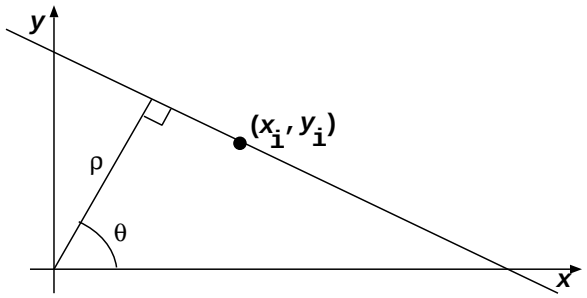

Figure 1: The principle of the Hough transform.

Then, the line is described by the equation:

$$\rho = x_i \cos\theta + y_i \sin\theta. \tag{1}$$

In the Hough Transform, the equation above is considered as a transformation from a point (x_i, y_i) in the xy-plane to a curve $\rho = x_i \cos\theta + y_i \sin\theta$ in the $\rho\theta$-plane. An important property is that parameter values of ρ and θ defining a line passing through two points in the xy-plane are given by the coordinates of the intersection between the curves in the $\rho\theta$-plane for these two points. Due to this property we can reduce the problem of detecting lines in the image plane into that of finding busy intersections of curves in the parameter plane. Another important property is that any two such curves intersect exactly once in the range $[\theta = 0, \theta = \pi]$ and the distance ρ is always bounded by $\sqrt{2}N$ where N is the length (number of pixels in a row) of an image (assumed to be a square image). Thus, we are required to find intersections at which many sinusoidal curves meet. Unfortunately, due to integrality of coordinates many points rarely lie on a line. In fact, a line we can recognize in a binary image is a sequence of black points (on grids) which lie sufficiently close to a line. To take such ambiguity into accounts, the parameter plane is partitioned into small cells (rectangular regions). This corresponds to quantization of ρ and θ values to define an accumulator array. That is, for each of the quantized θ values we compute the corresponding ρ value and put one vote into the cell defined by the θ and quantized ρ values.

Hough transform is a well-established scheme, but several problems are involved. Some of them are listed below:

1: Threshold selection The voting process for all of edge points is followed by peak detection in the accumulator array. When the voting process is over, the array elements are scanned to find those which have the numbers of votes exceeding a predetermined threshold value. There is no reasonable way to determine the best threshold value, which may be dependent on image sizes and image types. Moreover, it may not be so good to use the same threshold value over the entire image.

2: Suppressing non-maximal line components For safe side the threshold value should be small enough to detect all line components, but on the other hand small value produces so many line components. One solution to this tradeoff is to output only local peaks, that is, those array elements which are largest among the adjacent elements. Such a constraint seems to be reasonable, but there is no theory behind it. Actually it is easy to create an example for which this approach fails to output maximal line components by a choice of quantization. Ideally we want to output only maximal line components, which are formally defined later. A question is whether it is possible or not. Theoretically, the answer to this question is positive. We will show how to do it.

3: Distortion of the parameter plane A naive way of partitioning the parameter plane into buckets is due to equally separated vertical and horizontal lines. Although this partition is simple and easily implemented, it is also known that it does not lead to good performance of line detection. The relationship between the size of bucket and property of lines to be detected is also vague. In fact, there are some reports concerning the distortion of the accumulator array [14].

4: Information on endpoints Outputs of the above-mentioned version of the Hough transform are just coordinates of buckets, and thus no information on the lengths or endpoints of those lines are not reported.

5: Density constraint Suppose that pixel count is the only information available in the above-mentioned algorithm. Then, it is essentially difficult to distinguish two equal-sized groups of edge points such that the one is short and densely distributed while the other is long but sparse. In practice, it is sometimes important to output only dense line segments. However, density of line segments has not been characterized well in the literature.

3 Definition and Characterization of Digital Lines

We start with the definition of our objects, digital line components. Throughout the paper we assume that lines have slopes between 0 and 1 for simplicity. It is easy to adapt the following discussions for lines of slopes greater than 1 or smaller than 0.

[Definition: Digital Line Components]

A common algorithm to draw a line $y = ax + b$ on a screen puts white dots at distance at most 0.5 from the line. Such a set of grid points is called a **digital representation** of a line $y = ax + b$ and denoted by $G(a, b)$. Then, a set P of edge points is called a **(digital) line component** if there exists a line whose digital representation includes P, that is, there exist a and b such that $P \subseteq G(a, b)$. A line component is **maximal** if addition of any edge point violates the condition above.

For those lines of slopes between 0 and 1 the vertical distance to the line is always smaller (exactly no greater) than the horizontal distance. In this paper we are concentrated on those lines: To deal with lines of slopes not in the above range it suffices to exchange the x and y coordinates. Thus, it does not lose any generality.

Now, the problem of detecting line components is described as follows:

[Problem of Detecting Line Components:]
Given a binary edge image of size $N \times N$ containing n edge points and a threshold t, report all maximal line components of size greater than t in the image.

Our goals are listed below:

Goal 1: Report all maximal line components without reporting any non-maximal ones.

Goal 2: The algorithm should be robust against numerical errors and degeneracy.

Goal 3: Achieve low space complexity.

Goal 4: Achieve low time complexity.

Let P be a set of edge points. If P is a line component, there exist constants a and b such that

$$-\frac{1}{2} \leq y_i - ax_i - b \leq \frac{1}{2} \tag{2}$$

holds for each point $p_i = (x_i, y_i)$ in P. Here again remark that our objects are lines with slope between 0 and 1 and thus it suffices to consider the vertical distance to the line.

We defined a line component corresponding to a parameter pair (a, b) by

$$G(a,b) = \{(x_i, y_i) \in G \mid -\frac{1}{2} \leq y_i - ax_i - b \leq \frac{1}{2}\}. \tag{3}$$

When two parameter pairs (a, b) and (a', b') give the same set, i.e., $G(a, b) = G(a', b')$ holds, we say that they are equivalent. The equivalence relation partitions the parameter plane into equivalence classes. Then, we can prove that each equivalence class contains a point of rational coordinates.

Lemma 3.1 *For any line component P, there is such a parameter pair (a, b) among those pairs characterizing P that a and b are both rational numbers of the forms $\frac{q}{p}$ and $b = y - \frac{q}{p}x \pm \frac{1}{2}$, respectively, where $0 \leq q \leq p \leq N-1$ and $0 \leq x, y \leq N-1$.*

Proof: By the definition, there exist real numbers a and b that satisfy $-\frac{1}{2} \leq y_i - ax_i - b \leq \frac{1}{2}$ for each point $p_i = (x_i, y_i)$ in P, and $P \subseteq G(a, b)$. This set P of edge points is equal to that of edge points contained in the region bounded by the two parallel lines $b = y - \frac{q}{p}x - \frac{1}{2}$ and $b = y - \frac{q}{p}x + \frac{1}{2}$. Now, we translate these two lines downward (by decreasing the b values) until one of the lines touches some edge point p_j of P. Note that the set of edge points in the bounded region remain unchanged as a set. Then, we rotate the two parallel lines counterclockwisely (by increasing the value of a) around the point p_j on the boundary until one of the lines touches any other edge point p_k of P.

After the above translation and rotation, two edge points (on integer lattice) p_j and p_k must lie on the lines and the vertical distance between these two lines is exactly 1. Therefore, both of the lines must pass through at least two integer lattice points (not necessarily edge points of P). This implies that the slope and y-intercepts of the lines are rational numbers described in the lemma. □

4 Voting Technique and Its Limit

Lemma 3.1 says that it suffices to consider $O(N^2)$ different slopes (a values), say $(a_0, a_1, \ldots, a_K)$, where $K = O(N^2)$. For y-intercepts (b values), there are $O(N^4)$ possibilities since x_i and y_i can take $O(N)$ different values. However, if a slope a_j is fixed, then there are only $O(N^2)$ different b values. This means that if we make different partitions for each slope then we can save space complexity to $O(N^2 \times N^2)$.

Now we have a collection of one-dimensional arrays one for each slope instead of a uniform 2-dimensional array. When a slope a is fixed to a_j in the range $0 \leq a_j \leq 1$, the largest and smallest possible b values are defined by

$$\begin{aligned} b^{(j)}_{\min} &= \min\{y_i - a_j x_i \mid 0 \leq x_i, y_i \leq N-1\} = -a_j(N-1), \\ b^{(j)}_{\max} &= \max\{y_i - a_j x_i \mid 0 \leq x_i, y_i \leq N-1\} = N-1. \end{aligned}$$

Such extreme b values are defined similarly for other ranges of a values. To define buckets for the slope a_j, we partition the interval $[b^{(j)}_{\min}, b^{(j)}_{\max}]$ into $O(N^2)$ small subintervals. Let $(b_0, b_1, \ldots, b_{N^2})$ be the discrete b values defined by the mid-values of those subintervals.

Now, with the slope fixed to a_j, for each edge point (x_i, y_i) we find all b_k values satisfying the inequality

$$-\frac{1}{2} \leq y_i - a_j x_i - b_k \leq \frac{1}{2} \tag{4}$$

and then put one vote to each such bucket b_k. Here notice that $O(N)$ votes are put for each edge point and thus it takes time. Of course, without any sophisticated algorithm or data structure it takes $O(N)$ time. One way for efficient implementation is to first find the smallest and largest b_k values satisfying the inequality (4) by binary search, say b_u and b_v, respectively, and then to put the interval $[u, v]$ as a vote. If we have n edge points, n such intervals are produced. Required is to enumerate all indices ks such that the number of intervals containing k exceeds a predefined threshold. Fortunately, we can rely on a data structure, such as a segment tree or interval tree [6], in which insertion of an interval is done in $O(\log N)$ time.

More precise description of such a data structure is here: Let M be the maximum number of buckets to be prepared for a slope. The value of M may be different depending on slopes. Anyway, $M = O(N^2)$. We prepare a segment tree defined for the interval $[1, M]$. The root of the tree corresponds to the whole interval $[1, M]$. It has two children, one for the interval $[1, \lfloor (M+1)/2 \rfloor]$ and the other for $[\lfloor (M+1)/2 \rfloor + 1, M]$. Generally, a node for an interval $[L, R]$ has two children for $[L, \lfloor (L+R)/2 \rfloor]$ and $[\lfloor (L+R)/2 \rfloor + 1, R]$ as far as $R > L$. Nodes of the form $[L, L]$ are leaves of the tree. Let $T(u, v)$ denote the node for the interval $[u, v]$.

With this data structure we can implement the voting of an interval $[j, k]$ as follows:

procedure voting(u, v, j, k)

(1) If the interval for a node $T(u, v)$ is completely included in the voting interval $[j, k]$, add one vote to the node $T(u, v)$ and terminate.

(2) If the voting interval intersects that of the left child $T(u, v')$ of $T(u, v)$, then apply voting(u, v', j, k) recursively.

(3) If the voting interval intersects that of the right child $T(u', v)$ of $T(u, v)$, then apply voting(u', v, j, k) recursively.

It is easily seen that the number of nodes visited by the procedure above is $O(\log M) = O(\log N)$. Thus, the voting process for a slope is done in $O(n \log N)$ time. When the voting process is over, we have to enumerate all the leaves whose counts of votes exceed some threshold and also greater than those of their neighbors. For each leaf $[u, u]$ the number of intervals including u is computed by taking the sum of counts of the nodes on the path from the leaf to the root. This is a bottom-up process, and is done in $O(N^2)$, which can be improved to $O(n)$ although we do not go in detail.

Our algorithm proceed slope by slope. So, after finishing one slope, we have to proceed to the next slope. Before starting the voting process the segment tree must be initialized. It is done in $O(N^2)$ time by a naive manner. However, it is possible to design a segment tree data structure so that no initialization is required by keeping a version number. Again, we do not go in detail.

The above process is iterated for each slope. Thus, the overall running time is improved to $O(nN^2 \log N + N^4)$. Although this is a great improvement over the previous method which takes $O(nN^4)$ time, it looks pessimistic to have further improvement without a revolutionary change of idea.

5 An Algorithm Based on Arrangement of Lines

There is an algorithm based on a totally different idea based on an arrangement of lines in the dual plane. A basic idea is presented by the author [2]. In this paper we refine it.

Our basic idea is as follows: An edge point $p = (x, y)$ is contained in a line component $G(a, b)$ when the following inequality hods:

$$-\frac{1}{2} \leq y - ax - b \leq \frac{1}{2}. \tag{5}$$

Rewriting it, we have

$$-ax + y - \frac{1}{2} \leq b \leq -ax + y + \frac{1}{2}. \tag{6}$$

A set of points in the dual plane which satisfy the inequality (6) above corresponds to a tube bounded by two parallel lines. In other words, an edge point is mapped to a tube. When tubes associated with two edge points have non-empty intersection, for any point (a, b) in the intersection there is a line component for the line $y = ax + b$ which contains these two edge points. Thus, our task is again to compute intersections where many such tubes meet.

Drawing the $2n$ lines bounding the tubes in the dual plane, the dual plane is partitioned into $O(n^2)$ small regions called cells. Here note that the cells in the arrangement are exactly the equivalence classes define earlier. That is, for any two points in one cell, their corresponding line components are exactly the same. Thus, it suffices to check all the cells. A naive method to enumerate all the cells takes $O(n^3)$ time and $O(n^2)$ space, but Topological Walk by Asano, Guibas and Tokuyama [1] can visit all of them in $O(n^2)$ time and $O(n)$ space. An advantage of

the Topological Walk is that it can visit cells continuously. In other words, the next cell to be visited is one of the cells adjacent to the current cell by an edge. Therefore, if we distinguish upper and lower lines bounding tubes, it is not so hard to check the maximality of a cell. More precisely, paint the upper line of a tube red and the lower one blue. Then, a cell is not maximal if one of the lines in its upper boundary is blue since the cell just above the blue line corresponds to a larger set of edge points. We have a similar observation for red lines on the lower boundary. Thus, it is an easy observation that a maximal cell must be bounded by red lines from above and by blue lines from below. This implies that we can always check the maximality of a cell. Of course, a brute-force method for the check takes time. Fortunately, we can do it spending only some constant time per cell: Topological Walk reports cell information when it visits the rightmost vertex of a cell. So, we keep a Boolean information concerning maximality for each cell under consideration. We can update the Boolean value in $O(1)$ time whenever we cross a line during the walk. See Figure 2 for pictorial illustration.

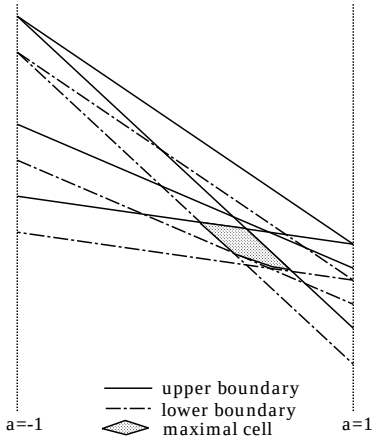

Figure 2: Partition of the parameter plane into cells by tubes.

Another problem for practical implementation is caused by degeneracy. When we transform edge points to tubes in the dual plane, many lines may happen to meet at one point. Since edge points have integer coordinates, slopes and y-intercepts are rational numbers. In the worst case $O(N)$ lines meet at one point. Fortunately, Topological Walk is robust against degeneracy and in fact it is proved in [3] that it is implemented without serious overhead on running time to deal with degenerated cases. There is only one serious problem caused by a type of degeneracy, which is caused by two edge points vertically adjacent by one lattice edge (more exactly, two points of the same x coordinates but different y coordinates just by 1). In this case the lower boundary line of one edge point exactly coincides with the upper one of the other edge point. They are different only in their colors. To deal with this degeneracy, we have to deal with a cell without an area (that is, a line segment). An alternative way is to use a different width for a digital representation of a line, more precisely, to use, say $6/11$ instead of $1/2$ in the equations for tubes so that no two boundary lines coincide with each other.

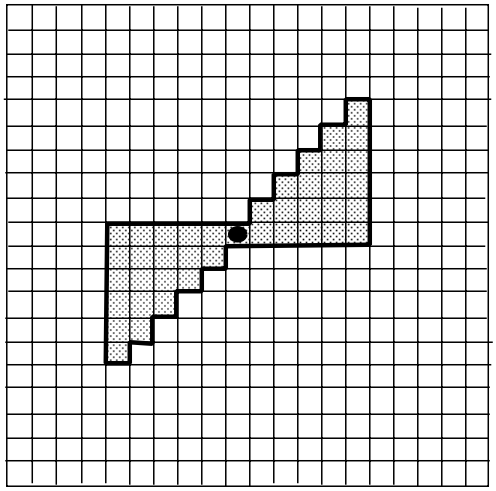

Figure 3: Noise reduction by a fanfilter.

6 Fanfilter: Noise Reduction

The computational complexities of the two algorithms described so far heavily depend on the number n of edge points. Especially the arrangement-based one runs in $O(n^2)$ time which is independent on the image size, and thus it is crucial to remove as many noisy edge points as possible. In this paper we propose a filter which is specialized to remove noisy edge points.

The idea is simple: Suppose that we are trying to detect line components of slopes between 0 and 1 which satisfy the local density condition mentioned in the previous section. More formally, we define a notion of an α-dense line segment: A set P of edge points is said to be an α-dense line segment if P contains at least $\alpha \times w$ edge points in any interval $[s, s+w-1]$ of length w such that $[s, s+w-1] \subseteq I(P)$, where $I(P)$ is the interval defined by the smallest and largest x-coordinates of edge points in P and w is some predetermined constant. To remove noisy edge points which are not contained in any such α-dense line segment, for each pixel p we define an area consisting of two triangles meeting at p as depicted in Figure 3. We call this area a fan area for p and denote it by $F_w(p)$. More formally, the fan area for p consists of two triangular areas swept by a line segment of horizontal length w rotating around p from its horizontal position to 45 degrees, where w is some constant determining the fan size. In other words, any line segment of horizontal length w with slope between 0 and 1 should be properly included in the fan area if it passes through p. With these definitions it is clear that an edge point is a noise if its associated fan area does not contain $\alpha \times w$ edge points. Here note that the reverse is not always the case. So, in this filter we count for each edge point the number of edge points in its associated fan area, and if the count is less than a threshold ($\alpha \times w$), we remove it as a noise. In our experiments we settled the α value and the area size w to be 10/13 and 12, respectively.

The fanfilter described above is the one with slopes between 0 and 1. Other fanfilters for the slope intervals $[1, \infty]$, $[-\infty, -1]$, and $[-1, 0]$ are defined similarly.

Naive implementation of the fanfilter takes $O(N^2 + w^2 n)$ time (or $O(w^2 n)$ time if edge

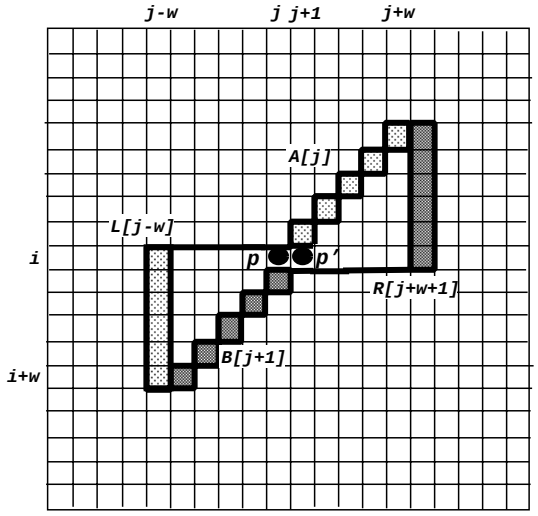

Figure 4: Efficient implementation of a fanfilter.

points are stored in an array), which is not efficient when n, the number of edge points, is large. In the following we present an efficient algorithm for implementing the fanfilter for large n. The algorithm is based on raster scan. What we are required is to compute for each pixel p the number of edge points contained in the fan area $F_w(p)$. If it is done in the raster order, after finishing the process at a pixel $p(i,j)$ we proceed to the next pixel $p'(i,j+1)$. The count for p' is obtained by adding the numbers of edge points in the two darkly painted stripe regions and subtracting those in the two lightly painted stripe regions to the count for the previous pixel p (see Figure 4). Therefore, if we maintain such counts, then the update of the count is done in constant time at each pixel. It is not so hard to maintain such counts in each row. At the beginning of each row we can update such counts in constant time per each pixel in the row, which takes $O(N)$ time in total for a row, where N is the number of pixels in a row. For example, an array element $A[j]$ keeps the number of edge points in a stripe region of angle $45°$ from the pixel at (i,j) to the one at $(i-w, j+w)$. When we move down to the next row $i+1$, the count $A[j]$ is computed based on the count $A[j-1]$ for the previous row by incrementing the count if the pixel at $(i+1,j)$ is an edge point and by decrementing it if the extreme pixel at $(i-w, j+w)$ is an edge point. This is done in constant time. The other three counts B, C and R in Figure 4 are similarly updated.

The overall running time is evaluated as follows: Four additions/subtractions are required to compute the number of edge points in a fan area $F_w(p)$ at each pixel p. At each row we need to maintain the four arrays. The initialization at each row needs eight additions/subtractions in total at each pixel. Only at the first row, $2wN$ additions are required to initialize the two arrays extending downwards. Also, at the first pixel in each row, we compute the number of edge points in the fan area in a brute-force manner. Thus, it takes w^2N additions in total. Summarizing all these operations, the number of additions and subtractions needed for an $N \times N$ image amounts to $4N^2 + 8N^2 + 2wN + w^2N \simeq 12N^2 + (w+1)^2N$. Since w is usually much smaller than N, the second term is not dominant.

7 Randomized Algorithm

A third algorithm for detecting line segments is based on the notion of randomized algorithm. Again our objective here is to detect line segments of slopes between 0 and 1. Other cases are treated similarly. After applying the fanfilter to remove noisy edge points, it first decomposes the resulting edge points into connected components. Here, we say that two edge points are connected if one of them lies in the fan area for the other. The filter size w determining the fan area may be different from that used for fanfilter. For the purpose here the size w should be sensitive to the largest allowable gap between two consecutive edge points in a line segment. This procedure can be efficiently implemented even naively or by using some data structure suitable for range search such as a quad tree [6].

In the best case each connected component gives one line segment. However, it may also happen that a number of line segments form a single connected component since two line segments of valid slopes are connected if they intersect each other. Thus, we have to check each connected component whether it includes more than one line segment. It is an easy task. Just construct a convex hull for the component and find its minimum width by computing the closest pair of parallel supports bounding the convex hull. The convex hull is computed in linear time since edge points in the component are sorted. The closest pair is also found in linear time by the rotating caliper method [11].

Recall that the algorithm based on an arrangement of lines took $O(n^2)$ time for n edge points. For practical purpose it is sometimes too severe to require reporting only maximal line components without missing any maximal one. If approximate outputs suffice, we can rely on the idea of random sampling. That is, we randomly choose r edge points out of a connected component consisting of m edge points and apply a quadratic algorithm. If the size r of random samples is $O(\sqrt{m})$, then the running time remains linear in the component size m. Of course, the result depends on the size of random samples.

Given a set of random samples, we apply some quadratic-time algorithm to detect lines of valid slopes. This time, it must be followed by a procedure to check whether detected lines really exist and satisfy conditions to form line segments (largest gap, local minimum density, etc.). It is done just by following the lines detected from left to right in the original edge image. Since it takes $O(N)$ time for each line candidate, we should rely on some other algorithm if there are a number of line candidates.

8 Experiments

For experiments we used artificially generated images, which were used for the programming contest organized by the special interest group on computer vision of Information Processing Society of Japan in 1996. One of the test images is shown in Figure 5, which is of size 512×512. It contains 28,461 edge points. Then the fanfilter for the slope range $[0, 1]$ is applied twice. The first fanfilter is the one for large n and the second for small n. This results in 3,143 edge points after the first fanfilter and 1,795 edge points after the second one. The resulting image is shown in Figure 6. Then, the randomized algorithm was implemented to find line components

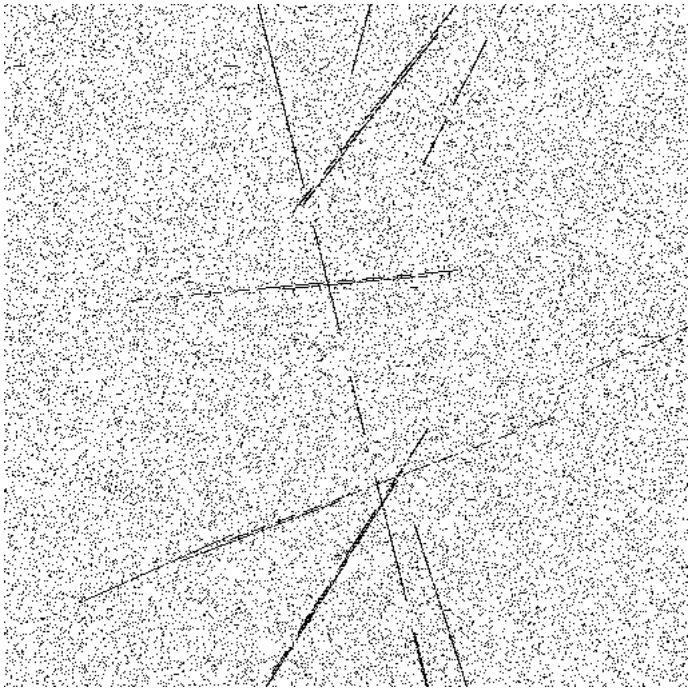

Figure 5: Input binary edge image with $28,461$ edge points which was artificially generated.

of slopes between 0 and 1. The final result is shown in Figure 7. Again note that steeper line components can be detected by exchanging x and y coordinates of edge points. The CPU time was about 0.03 seconds including the time for the fanfilter. The computer used for the experiment is SUN Workstation (Fujitsu version) S-7/400Ui with Solaris 2.6.

Acknowledgments

The author expresses his sincere gratitude to Yasuyuki Kawamura, Koji Obokata, and Takeshi Tokuyama for their helpful comments in the theory and implementation. This work was partially supported by Grant in Aid for Scientific Research of the Ministry of Education, Science and Cultures of Japan.

9 Conclusions

Hough Transform is a well-established method for detecting lines and curves in a binary edge image. In this paper we considered the problem of detecting lines from a point of computational complexity and presented three different directions. For theoretical point of view or in the sense of asymptotical behavior, the second one based on an arrangement of lines is most reliable. It achieves all of the goals listed in Section 3. Especially, it can be robust against numerical errors and degeneracy if we use Topological Walk that is proved to be robust if it is implemented using rational data types instead of floating-point data types. For practical use the conventional method based on voting may be good enough. The third approach using random sampling may

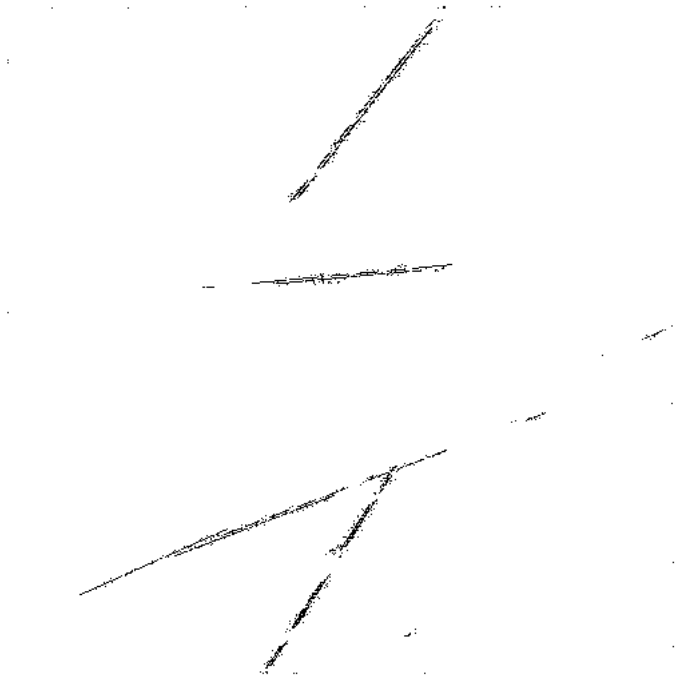

Figure 6: The effect of the fanfilter: The image resulting after applying the fanfilter for the slope interval $[0, 1]$ twice, consisting of 1,795 edge points.

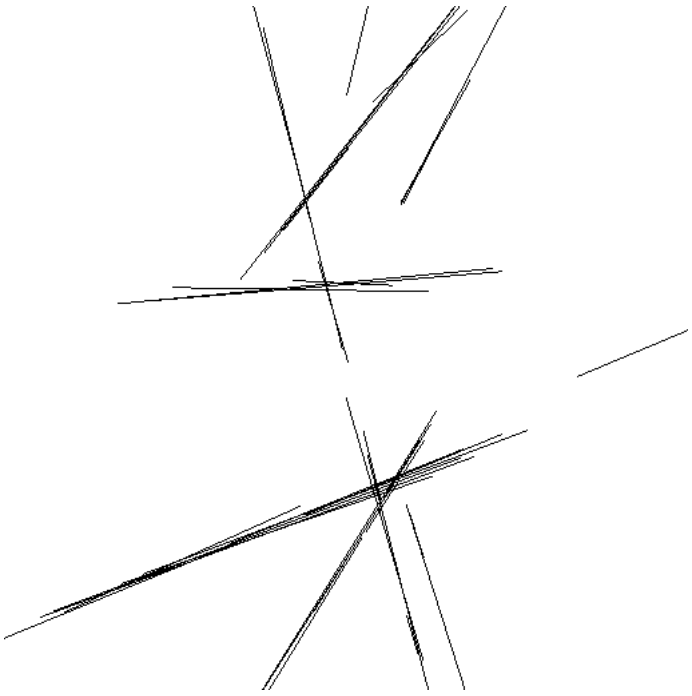

Figure 7: Line segments detected.

be located intermediate between the other two.

References

[1] T. Asano, L. Guibas, and T. Tokuyama, "Walking in an Arrangement Topologically," Int. J. of Comput. Geom. and Appl., 4, pp.123-151, 1994.

[2] T. Asano and N. Katoh: "Variants for the Hough Transform for Line Detection," Computational Geometry: Theory and Applications, 6, pp.231-252, 1996.

[3] T. Asano and T. Tokuyama: "Topological Walk Revisited," IEICE Trans. on Fundamentals, vol. E81-A, 5, pp.751-756, 1998.

[4] M. Atiquzzaman: "Multiresolution Hough Transform — An Efficient Method of Detecting Patterns in Images," IEEE Trans. Pattern Anal. Mach. Intell., PAMI-14, 11, pp.1090-1095, 1992.

[5] C.M. Brown: "Inherent Bias and Noise in the Hough Transform," IEEE Trans. Pattern Anal. Mach. Intell., PAMI-5, 5, pp.493-505, 1983.

[6] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf: "Computational Geometry: Algorithms and Applications," Springer, 1997.

[7] R. O. Duda and P.E. Hart: "Use of the Hough Transformation to Detect Lines and Curves in Pictures", Comm. of the ACM, 15, January 1972, pp.11-15.

[8] P. V. C. Hough: "Method and Means for Recognizing Complex Patterns", U.S. Patent 3069654, December 18, 1962.

[9] J. Illingworth and J. Kittler: "The Adaptive Hough Transofrm," IEEE Trans. Pattern Anal. Mach. Intell., PAMI-9, 5, pp.690-698, 1987.

[10] H. Li, M.A. Lavin, and R.J. LeMaster: "Fast Hough Transofrm," Comput. Vision Graphics Image Processing, 36, pp.139-161, 1986.

[11] F. P. Preparata and M. I. Shamos: "Computational Geometry: An Introduction," Springer-Verlag, 1985.

[12] M. Seki, T. Wada and T. Matsuyama: "High Precision $\gamma - \omega$ Hough Transformation Algorithm to Detect Arbitrary Digital Lines," Proc. SIGCV Workshop of IPSJ, CV-84-2 (Jul. 1993).

[13] I.D. Svalbe: "Natural Representation for Straight Lines and the Hough Transform on Discrete Arrays," IEEE Trans. Pattern Anal. Mach. Intell., PAMI-11, 9, pp.941-950, 1989.

[14] T. Wada, M. Seki, and T. Matsuyama: "High Precision $\gamma - \omega$ Hough Transformation Algorithm to Detect Arbitrary Digital Lines," Trans.D-II, IEICE of Japan, J77-D-II, 3, pp.529-539, 1994 (in Japanese).