Real-Time Line Detection Using Accelerated High-Resolution Hough Transform

Radovan Jošth, Markéta Dubská, Adam Herout, and Jiří Havel

Graph@FIT
Brno University of Technology, Faculty of Information Technology
Brno, Czech Republic
{ijosth,idubska,herout,ihavel}@fit.vutbr.cz
http://www.fit.vutbr.cz/research/groups/graph/

Abstract. Hough transform is a well-known and popular algorithm for detecting lines in raster images. The standard Hough transform is rather slow to be usable in real-time, so different accelerated and approximated algorithms exist. This paper proposes a modified accumulation scheme for the Hough transform, which makes it suitable for computer systems with small but fast read-write memory – such as the today's GPUs. The proposed algorithm is evaluated both on synthetic binary images and on complex high resolution real-world photos. The results show that using today's commodity graphics chips, the Hough transform can be computed at interactive frame rates even with a high resolution of the Hough space and with the Hough transform fully computed.

Keywords: Line Detection, Hough-Transform, Real-Time, GPU, CUDA.

1 Introduction

The Hough transform is a well-known tool for detecting shapes and objects in raster images. Originally, Hough [6] defined the transformation for detecting lines; later it was extended for more complex shapes, such as circles, ellipses, etc., and even generalized for arbitrary patterns [1].

When used for detecting lines in 2D raster images, the Hough transform is defined by a *parameterization* of lines: each line is described by two parameters. The input image is preprocessed and for each pixel which is likely to belong to a line, voting accumulators corresponding to lines which could be coincident with the pixel are increased. Next, the accumulators in the parameter space are searched for local maxima above a given threshold, which correspond to likely lines in the original image. The Hough transform was formalized by Princen et al. [14] and described as an *hypothesis testing* process.

Hough [6] parameterized the lines by their *slope* and y-axis *intercept*. A very popular parameterization introduced by Duda and Hart [3] is denoted as θ-ϱ; it is important for its inherently bounded parameter space. It is based on a line equation in the normal form: $y \sin\theta + x \cos\theta = \varrho$. Parameter θ represents the

A. Heyden and F. Kahl (Eds.): SCIA 2011, LNCS 6688, pp. 784–793, 2011.

angle of inclination and ϱ is the length of the shortest chord between the line and the origin of the image coordinate system. There exist several other bounded parameterizations, mainly based on intersections of lines with image's bounding box [18][11][4]. Different properties of these intersects are used as parameters.

The majority of currently used implementations seems to be using the θ-ϱ parameterization – for example the OpenCV library implements several variants of line detectors based on the θ-ϱ parameterization and none other. It is mainly because the parameterization uses a very straightforward transformation from the image space to one bounded space of parameters and because of its uniform distribution of the discretization error across the Hough space.

Several research groups invested effort to deal with computational complexity of the Hough transform based on the θ-ϱ parameterization. Different methods focus on special data structures, non-uniform resolution of the accumulation array or special rules for picking points from the input image.

O'Rourke and Sloan developed two special data structures: *dynamically quantized spaces* (DQS) [13] and *dynamically quantized pyramid* (DQP) [17]. Both these methods use splitting and merging cells of the space represented as a binary tree, or possibly a quadtree. After processing the whole image, each cell contains approximately the same number of votes; that leads to a higher resolution of the Hough space of accumulators at locations around the peaks.

A typical method using special picking rules is the Randomized Hough Transform (RHT) [19]. This method is based on the idea, that each point in an n-dimensional Hough space of parameters can be exactly defined by an n-tuple of points from the input raster image. Instead of accumulation of a hypersurface in the Hough space for each point, n points are randomly picked and the corresponding accumulator in the parameter space is increased. Advantages of this approach are mostly in rapid speed-up and small storage. Unfortunately, when detecting lines in a noisy input image, the probability of picking two points from same line is small, decreasing the probability of finding the true line.

Another approach based on repartitioning the Hough space is represented by the Fast Hough Transform (FHT) [8]. The algorithm assumes that each edge point in the input image defines a hyperplane in the parameter space. These hyperplanes recursively divide the space into hypercubes and perform the Hough transform only on the hypercubes with votes exceeding a selected threshold. This approach reduces both the computational load and the storage requirements.

Using principal axis analysis for line detection was discussed by Rau and Chen [15]. Using this method for line detection, the parameters are first transferred to a one-dimensional angle-count histogram. After transformation, the dominant distribution of image features is analyzed, with searching priority in peak detection set according to the principal axis. There exist many other accelerated algorithms, more or less based on the above mentioned approaches; e.g. HT based on eliminating of particle swarm [2] or some specialized tasks like iterative RHT [9] for incomplete ellipses and N-Point Hough transform for line detection [10]. For more information about different existing modifications of Hough transform, please see [7].

This paper presents an algorithm for real-time detection of lines based on the standard Hough transform using the θ-ϱ parameterization. The classical Hough transform has some advantages over the accelerated and approximated methods (it does not introduce any further detection error and it has a low number of parameters and therefore usually requires less detailed application-specific fine-tuning). That makes the real-time implementation of the Hough transform desirable. The algorithm uses a modified strategy for accumulating the votes in the array of accumulators in the Hough space. The strategy was designed to meet the nature of today's graphics chips (GPUs). The modified algorithm is presented in Section 2 of the paper. Section 3 presents the experiments comparing the commonly used variant of Hough transform with the implementation of the algorithm run on a GPU. The results show that the GPU implementation achieves such performance which allows running the Hough transform with a high-resolution accumulator space in real time. Section 4 concludes the paper and proposes directions for future work.

2 Real-Time Hough Transform Algorithm

Before discussing the new real-time Hough transform algorithm, let us review the "classical" Hough transform procedure based on the θ-ϱ parameterization in Algorithm 1 (the θ-ϱ parameterization itself is depicted by Figure 1).

Algorithm 1. HT for detecting lines based on the θ-ϱ parameterization.

Input: Input image I with dimensions I_w, I_h, Hough space dimensions H_ϱ, H_θ
Output: Detected lines $L = \{(\theta_1, \varrho_1), \ldots\}$

1: $H(\bar{\varrho}, \bar{\theta}) \leftarrow 0, \forall \bar{\varrho} \in \{1, \ldots, H_\varrho\}, \bar{\theta} \in \{1, \ldots, H_\theta\}$
2: **for all** $x \in \{1, \ldots, I_w\}, y \in \{1, \ldots, I_h\}$ **do**
3: **if** $I(x, y)$ **is edge then**
4: **increment** $H\big(\bar{\varrho}(\bar{\theta}, x, y), \bar{\theta}\big), \forall \bar{\theta} \in \{1, \ldots, H_\theta\}$
5: **end if**
6: **end for**
7: $L = \{(\theta(\bar{\theta}), \varrho(\bar{\varrho})) | \bar{\varrho} \in \{1, \ldots, H_\varrho\} \wedge \bar{\theta} \in \{1, \ldots H_\theta\} \wedge$ at $(\bar{\varrho}, \bar{\theta})$ is a high max. in $H\}$

Points in the input image I with dimensions I_w and I_h are classified with a binary decision on line 3 (e.g. by an edge detector and thresholding). Lines 2–6 accumulate curves into the Hough space. Function $\bar{\varrho}(\bar{\theta}, x, y)$ computes the corresponding $\bar{\varrho}$ for each line passing through point (x, y) at angle $\bar{\theta}$:

$$\bar{\varrho}(\bar{\theta}, x, y) = \left[\frac{H_\varrho\big((y - \frac{I_h}{2}) \sin(\frac{\pi}{H_\theta}\,\bar{\theta}) + (x - \frac{I_w}{2}) \cos(\frac{\pi}{H_\theta}\,\bar{\theta})\big)}{\sqrt{I_w^2 + I_h^2}} + \frac{H_\varrho}{2} \right]. \qquad (1)$$

Line 7 detects above-threshold local maxima in the accumulated space and transforms the discretized Hough space coordinates $\bar{\varrho}$ and $\bar{\theta}$ to ϱ and θ by the following functions:

$$\varrho(\bar{\varrho}) = \frac{\sqrt{I_w^2 + I_h^2}}{H_\varrho} \Big(\bar{\varrho} - \frac{H_\varrho}{2}\Big), \qquad \theta(\bar{\theta}) = \frac{\pi}{H_\theta}\,\bar{\theta}. \qquad (2)$$

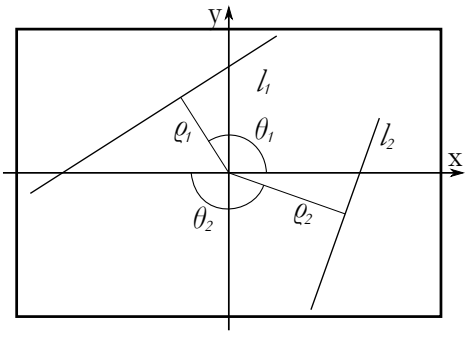

Fig. 1. The θ-ϱ parameterization of lines in a coordinate system with origin in the center of the input image.

Usually, a small neighborhood (3×3 in OpenCV, 5×5 or 7×7 in cases of high resolution of the Hough space) is used for detecting the local maxima by line 7. The accumulator value must be above a given threshold to be considered for a "high local maxima". The threshold is another input parameter of the algorithm, but since it does not influence the algorithm's structure, it is used silently by line 7 for simplicity of the algorithmic notation.

The key characteristic of this algorithm is rasterization of the sinus curve and incrementation the corresponding accumulators in the Hough space. On some systems, this might be expensive or even not available at all.

2.1 Hough Transform on a Small Read-Write Memory of Accumulators

The classical Hough transform accesses sparsely a relatively large amount of memory. This behavior can diminish the effect of caching. On CUDA and similar architectures, this effect is even more significant, as the global memory is not cached. To achieve real-time performance, the memory requirements must be limited to the *shared memory* of a multiprocessor (typically 16 kB).

Algorithm 2 shows the modified Hough transform accumulation procedure. The key difference from Algorithm 1 is the actual size of the Hough space. The new algorithm stores only $H_\varrho \times n$ accumulators, where n is the neighborhood size required for the maxima detection. Functions $\bar{\varrho}, \theta, \varrho$, and the edge and maxima detection are identical to Algorithm 1. First, the detected edges are stored in a set P (line 1). Then, first n rows of the Hough space are computed by lines 2–7. The memory necessary for containing the n lines is all the memory required by the algorithm and even for high resolutions of the Hough space, the buffer of n lines fits easily in the *shared memory* of the GPU multiprocessors.

In the main loop (lines 9–18), for every row of the Hough space, the maxima are detected (line 10), the accumulated neighborhood is shifted by one row (lines 11–13) and a new row is accumulated (lines 14–17); please refer to Figure 2 for an illustration of the algorithm. Thus, only the buffer of n lines is being

Algorithm 2. HT accumulation strategy using a small read-write memory.

Input: Input image I with dimensions I_w, I_h, Hough space dimensions H_ϱ, H_θ, neighborhood size n

Output: Detected lines $L = \{(\theta_1, \varrho_1), \ldots\}$

1: $P \leftarrow \{(x,y) | x \in \{1,\ldots,I_w\} \wedge y \in \{1,\ldots,I_h\} \wedge I(x,y) \text{ is an edge}\}$
2: $H(\bar{\varrho}, i) \leftarrow 0, \forall \bar{\varrho} \in \{1,\ldots,H_\rho\}, \forall i \in \{1,\ldots.n\}$
3: **for all** $i \in \{1,\ldots,n\}$ **do**
4: **for all** $(x,y) \in P$ **do**
5: **increment** $H(\bar{\varrho}(i,x,y), i)$
6: **end for**
7: **end for**
8: $L \leftarrow \{\}$
9: **for** $\bar{\theta} = \lceil \frac{n}{2} \rceil$ **to** $H_\theta - \lfloor \frac{n}{2} \rfloor$ **do**
10: $L \leftarrow L \cup \{(\theta(\bar{\theta}), \varrho(\bar{\varrho})) | \bar{\varrho} \in \{1,\ldots H_\varrho\} \wedge (\bar{\varrho}, \lceil \frac{n}{2} \rceil) \text{ is a high local max. in } H\}$
11: **for** $i = 1$ **to** $n - 1$ **do**
12: $H(\bar{\varrho}, i) \leftarrow H(\bar{\varrho}, i+1), \forall \bar{\varrho} \in \{1,\ldots,H_\varrho\}$
13: **end for**
14: $H(\bar{\varrho}, n) \leftarrow 0, \forall \bar{\varrho} \in \{1,\ldots,H_\varrho\}$
15: **for all** $(x,y) \in P$ **do**
16: **increment** $H(\bar{\varrho}(\bar{\theta} + \lceil \frac{n}{2} \rceil, x, y), n)$
17: **end for**
18: **end for**

reused. The memory shift can be implemented using a circular buffer of lines to avoid data copying.

In the pseudocode, maxima are not detected at the edges of the Hough space. Eventual handling of the maxima detection at the edge of the Hough space does not change the algorithm structure, but it would unnecessarily complicate the pseudocode. Two solutions exist – either copying the border data or rasterizing necessary parts of the curves outside of the Hough space. Both approaches perform similarly and their implementation is straightforward.

On CUDA, the threads in a block can be used for processing the set of edges P (lines 15–17 and 4–6) in parallel, using an atomic increment of the shared memory to avoid read-write collisions. In order to use all multiprocessors of the GPU, the loop on line 9 is broken to a number (e.g. 90 is suitable for current NVIDIA GeForce graphics chips) of sub-loops processed by individual blocks of threads.

The algorithm as described above uses exactly $H_\varrho \times n$ memory cells, typically 16-bit integer values. In the case when the runtime system has more fast random-access read-write memory, this memory can be used fully, and instead of accumulating one line of the Hough space (lines 15–17 of the algorithm), several lines are accumulated and then scanned for maxima. This leads to further speedup by reducing the number of steps carried out by the loop over $\bar{\theta}$ (line 9).

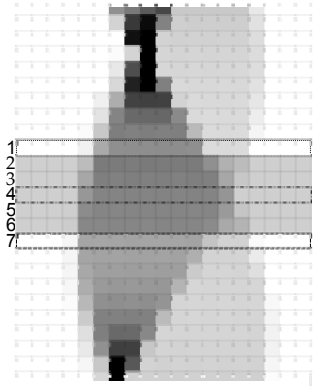

Fig. 2. Illustration of Algorithm 2. The gray rectangle represents the buffer of n lines. For row 4, the above-threshold maxima are detected in each step within the buffer. Then, the row 7 values are accumulated into the buffer, using the space of row 2, which will not be needed in future processing.

2.2 Harnessing the Edge Orientation

In 1976 O'Gorman and Clowes came with the idea not to accumulate values for each θ but just one value instead [12]. The appropriate θ for a point can be obtained from the gradient of the detected edge which contains this point [16]. One common way to calculate the local gradient direction of the image intensity is using the Sobel operator. Sobel detector uses two kernels, each approximates the derivation in horizontal (G_x), respectively vertical (G_y) direction. Using these two values, the gradient's direction can be obtained as $\theta = \arctan(\frac{G_y}{G_x})$. To avoid errors caused by noise and rasterization, accumulators within several degrees around the calculated angle are also incremented. From experimental testing, the interval's radius equal to 20° seems suitable. This approach reduces the computation time and highlights the maxima peaks. A disadvantage of this method is its dependency of the results on another user parameter – the radius. Small radius of the incremented interval of θ can lead into discarding some maxima due to inaccurate θ location. On the other hand, a too high radius can diminish the performance benefits of the method.

This approach to utilizing the detected gradient can be incorporated to the new accumulation scheme presented in the previous section. When extracting the "edge points" for which the sinusoids are accumulated in the Hough space (line 1 in Algorithm 2), also the edge inclination is extracted:

$$1:\ P \leftarrow \{(\alpha, x, y) | x \in \{1, \ldots, I_w\} \wedge y \in \{1, \ldots, I_h\} \\ \wedge\ I(x, y) \text{ is an edge with orientation } \alpha\}.$$

Then, instead of accumulating all points from set P (lines 4–6), only those points which fall into the interval with radius w around currently processed θ are accumulated into the buffer of n lines:

4: **for all** $(\alpha, x, y) \in P \wedge\ i - w < \bar{\alpha} < i + w$ **do**
5: **increment** $H(\bar{\varrho}(i, x, y), i + \lfloor \frac{n}{2} \rfloor)$
6: **end for**

and similarly for lines 15–17:

16: **for all** $(\alpha, x, y) \in P \wedge\ \bar{\theta} + \lfloor \frac{n}{2} \rfloor - w < \bar{\alpha} < \bar{\theta} + \lfloor \frac{n}{2} \rfloor + w$ **do**
17: **increment** $H(\bar{\varrho}(\bar{\theta} + \lfloor \frac{n}{2} \rfloor, x, y), n)$
18: **end for**.

Please, note that the edge extraction phase (line 1) can sort the detected edges by their gradient inclination α, so that loops on lines 15–17 and 4–6 do not visit all edges, but only edges potentially accumulated, based on the current $\bar{\theta}$ (line 9 of Algorithm 2). For (partial) sorting of the edges on GPU, an efficient prefix sum can be used [5].

3 Experimental Results

This section evaluates the speed of the newly presented line-detection algorithm. Two groups of experiments were made: the first one is focused on the speedup in the case when ϱ is calculated for each θ (Section 2.1, Algorithm 2); the second test evaluates the situation when the Sobel operator is used for detection of edge orientation and only an interval of the sinusoid curves is accumulated to the Hough space (Section 2.2).

Each test compares the computation time of 4 implementations: new algorithm running on (i) ASUS nVIDIA GTX480 graphics card (1.5GB GDDR5 RAM) and (ii) ASUS nVIDIA GTX280 graphics card (1GB GDDR3 RAM), (iii) an OpenMP parallel CPU implementation of the presented algorithm (Intel Core i7-920, 6GB 3×DDR3-1066(533MHz) RAM – the same machine was used for evaluating the GPU variants), (iv) and an OpenMP parallel "standard" implementation running on the same machine. As the "standard" implementation, the code based on OpenCV functions was used and optimized by parallelization.

3.1 Performance Evaluation on Synthetic Binary Images

As the dataset for this experiment we used automatically generated black-and-white images. The generator randomly places L white lines and then inverts pixels on P different positions in the image. The evaluation is done on 36 images (resolution 1600×1200): images 1–6, 7–12, 13–18, 19–24, 25–30, 31-36 are generated with $L = 1, 30, 60, 90, 120, 150$ respectively, with increasing $P = 1, 3000, 6000, 9000, 12000, 15000$ for each L. The suitable parameters for images of these properties were $H_\varrho = 960$ and $H_\theta = 1170$ (resolution of the Hough space) and the threshold for accumulators in the Hough space was 400.

Figure 3 reports the results of the four implementations. Note, please, that the CUDA version is several times faster than the commonly used OpenCV implementation and achieves real-time or nearly real-time speeds.

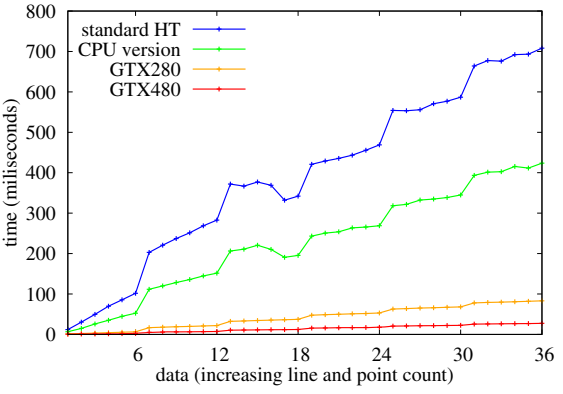

Fig. 3. Performance evaluation on synthetic binary images. Red: GTX480, Orange: GTX280, Green: Striped algorithm on the CPU, Blue: Standard HT accumulation.

3.2 Performance Evaluation on Real-Life Images

The images used in this test were real-world images (see Figure 4). For possibility of comparison with previous test, resolution of Hough space was same; i.e. $H_\varrho = 960$ and $H_\theta = 1170$; the threshold for accumulators in the Hough space was dependant on the input image resolution (one fourth of the diagonal; this corresponds to the shortest possible line detected by Hough transform); the radius of the accumulated interval (refer to Section 2.2) was 20°.

Figure 5 contains the measured results. They indicate that even for complex real-world images the proposed algorithm implemented on commodity graphics hardware can detect lines at interactive frame rates. Contrary to the version that

Fig. 4. Representative real images used in the test. The number in the top-left corner of each thumbnail image is the image ID – used on the horizontal axis in Figure 5. The bottom-left corner of each thumbnail states the pixel resolution of the tested image.

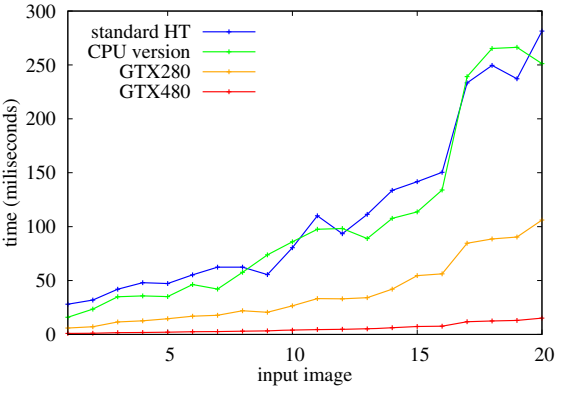

Fig. 5. Performance evaluation on real-world images (see Figure 4) using the Sobel operator and only accumulating intervals of the sinusoids. Red: GTX480, Orange: GTX280, Green: Striped algorithm on the CPU, Blue: Standard HT accumulation.

works with the whole sinusoids in the Hough space (Section 3.1), the speed of the CPU implementation of the presented algorithm is about as fast as the standard CPU version. This can be explained by better cache coherency when only fractions of the sinusoids are rasterized. However, for efficient implementation on CUDA and similar architectures, the presented algorithm is required.

4 Conclusions

This paper presents a modified algorithm for line detection using the Hough transform based on θ-ϱ parameterization. The algorithm was designed to fully using of small read-write memory; that makes it suitable for execution on recent graphics processors.

The experiments show that on commodity graphics hardware, the algorithm can operate at interactive frame rates even on high-resolution real-life images, while accumulating to a high-resolution Hough space to achieve accurate line detections. This real-time processing speed is achieved for the plain Hough transform, which – contrary to the acceleration and approximation mechanisms used in the literature – does not require many application-specific parameters. Therefore, its use in applications is more straightforward and does not require complicated human-assisted parameter tweaking. While the algorithm was designed for GPU processing, it outperforms the standard HT implementation even on the CPU, thanks to better cache usage of the new accumulation scheme.

The algorithm is very suitable for recent GPUs; however, it can be used on other architectures with limited fast read-write memory and high degree of parallelism, as well. In near future, we are intending to explore its usability on other platforms – focusing on embedded, mobile and low-power systems.

Acknowledgements. This work was partially supported by the BUT FIT grant FIT-10-S-2, by the research plan MSM0021630528, by the research program LC-06008 (Center for Computer Graphics), and by the FP7-ARTEMIS project no. 100230 SMECY.

References

1. Ballard, D.H.: Generalizing the Hough Transform to Detect Arbitrary Shapes, pp. 714–725 (1987)
2. Cheng, H.D., Guo, Y., Zhang, Y.: A novel hough transform based on eliminating particle swarm optimization and its applications. Pattern Recogn. 42, 1959–1969 (2009)
3. Duda, R.O., Hart, P.E.: Use of the Hough transformation to detect lines and curves in pictures. Commun. ACM 15(1), 11–15 (1972)
4. Eckhardt, U., Maderlechner, G.: Application of the projected Hough transform in picture processing. In: Proceedings of the 4th International Conference on Pattern Recognition, pp. 370–379. Springer, London (1988)
5. Harris, M.: GPU Gems 3. In: Parallel Prefix Sum (Scan) with CUDA, ch. 39, pp. 851–876. Addison-Wesley, Reading (2007)
6. Hough, P.V.C.: Method and means for recognizing complex patterns, u.S. Patent 3,069,654 (December 1962)
7. Illingworth, J., Kittler, J.: A survey of the Hough transform. Comput. Vision Graph. Image Process. 44(1), 87–116 (1988)
8. Li, H., Lavin, M.A., Le Master, R.J.: Fast Hough transform: A hierarchical approach. Comput. Vision Graph. Image Process. 36, 139–161 (1986)
9. Lu, W., Tan, J.: Detection of incomplete ellipse in images with strong noise by iterative randomized hough transform. Pattern Recogn. 41, 1268–1279 (2008)
10. Mochizuki, Y., Torii, A., Imiya, A.: N-point hough transform for line detection. J. Vis. Comun. Image Represent. 20, 242–253 (2009)
11. Natterer, F.: The mathematics of computerized tomography. Wiley, John & Sons, Incorporated, Chichester (1986) ISBN 9780471909590
12. O'Gorman, F., Clowes, M.B.: Finding picture edges through collinearity of feature points. IEEE Trans. Computers 25(4), 449–456 (1976)
13. O'Rourke, J.: Dynamically quantized spaces for focusing the Hough transform. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence, vol. 2, pp. 737–739. Morgan Kaufmann Publ. Inc., San Francisco (1981)
14. Princen, J., Illingowrth, J., Kittler, J.: Hypothesis testing: A framework for analyzing and optimizing Hough transform performance. IEEE Trans. Pattern Anal. Mach. Intell. 16(4), 329–341 (1994)
15. Rau, J.Y., Chen, L.C.: Fast straight lines detection using Hough transform with principal axis analysis. J. Photogrammetry and Remote Sensing 8, 15–34 (2003)
16. Shapiro, L.G., Stockman, G.C.: Computer Vision. Tom Robbins (2001)
17. Sloan, K.R.: Dynamically quantized pyramids. In: Proc. International Joint Conference on Artificial Intelligence (IJCAI), pp. 734–736. Morgan Kaufmann, San Francisco (1981)
18. Wallace, R.: A modified Hough transform for lines. In: Proceedings of CVPR 1985, pp. 665–667 (1985)
19. Xu, L., Oja, E., Kultanen, P.: A new curve detection method: Randomized Hough Transform (RHT). Pattern Recognition Letters 11, 331–338 (1990)