Edge Drawing: A Heuristic Approach to Robust Real-Time Edge Detection

Cihan Topal

Edge Drawing: A Heuristic Approach to Robust Real-Time Edge Detection

Cihan Topal[1] Cüneyt Akınlar[1] Yakup Genç[2]

Department of Computer Engineering[1]
Anadolu University, 2 Eylül Kampüsü, Eskişehir, 26470, TURKEY
{cihant, cakinlar}@anadolu.edu.tr

Siemens Corporate Research[2]
College Road East 755, Princeton, NJ, 05840, USA
yakup.genc@siemens.com

Abstract

We propose a new edge detection algorithm that works by computing a set of anchor edge points in an image and then linking these anchor points by drawing edges between them. The resulting edgemap consists of perfect contiguous, one-pixel wide edges. The performance tests show that our algorithm is up to 16% faster than the fastest known edge detection algorithm, i.e., OpenCV implementation of the Canny edge detector. We believe that our edge detector is a novel step in edge detection and would be very suitable for the next generation real-time image processing and computer vision applications.

1. Introduction

Edge detection is one of the fundamental problems in image processing and computer vision [1-9]. For image understanding, object detection, segmentation, feature extraction and tracking applications, the detection of object boundaries must be done in an efficient manner [10-16].

Traditional edge detection algorithms work blindly by applying different sets of filters and thresholding, which result in edgemaps that consist of individual edge pixels with no real relationship and connectivity. That is, the edge pixels in the resulting edgemaps are not analyzed for thinness and connectivity.

In this paper, we propose a different method for edge detection. Unlike conventional methods, our algorithm computes a set of anchor points in the image and *draws edges* between them. The resulting edgemap consists of contiguous, 1-pixel wide edges with real connectivity.

2. Edge Detection by Edge Drawing

Our algorithm runs in 4 consecutive steps:

1. Image Smoothing
2. Determination of Edge Areas and Edge Directions
3. Computation of Edge Anchor Points
4. Linking of Edge Anchor Points by Edge Drawing

Below we cover each step in detail. To illustrate the output of each step of the algorithm, we use a 512x512 grayscale "Lena" image.

2.1. Image Smoothing

The goal of this step is to reduce the effect of noise in the image by blurring each pixel with its neighboring pixels. We achieve this by a standard 5x5 Gaussian kernel with $\sigma = 1$.

2.2. Determination of Edge Areas and Edge Directions

The goal of this step is to determine potential edge areas, i.e., regions of interest, where the actual edges reside. These areas are found by applying a derivative operator or a high-pass filter to each pixel of the smoothed image. Our method works well with several operators such as the Sobel operator [2].

Figure 1. High-pass filtering residue of Lena image.

We apply the Sobel operator to each pixel in the smoothed image and obtain two gradient values, G_x and G_y. G_x and G_y are then used to determine both the edge areas and the edge directions: If $G_x + G_y$ is bigger than a given threshold, then the pixel is marked as an "edge area" pixel, otherwise the pixel is marked as a "non-edge area" pixel. We also compute the direction of the edge as follows: If $G_x >= G_y$, then a

1051-4651/10 $26.00 © 2010 IEEE
DOI 10.1109/ICPR.2010.593

"**vertical edge**" is assumed to pass through this pixel; otherwise a "**horizontal edge**" is assumed to pass through.

Figure 1 shows the edge areas corresponding to our test image for a threshold value of 36. White edge areas correspond to pixels for which $G_x + G_y >= 36$. Black pixels are suppressed as non-edges.

2.3. Computation of Edge Anchor Points

Having computed the edge areas and directions, we continue with the computation of a set of edge point pixels inside the edge areas that would serve as "**anchor points**" for the final "**edge drawing**" process. To achieve this goal, we simply do a raster scan of the edge area image in row-major order and mark maximas as our anchor points. In this step, the user is given the option of selecting the amount of detail in the final edgemap. We define what we call a "**detail ratio**" that determines how many edge anchor points are selected. The more anchor points you have to start the linking process, the more detail you will have in your final edgemap. Thus, if you are only interested in obtaining the major (long) edges in the image, you can specify a big detail ratio, e.g., a value bigger than 10. But if you want more details in your edge map, you can specify a small detail ratio such as 1, 2, 3, 4, etc.

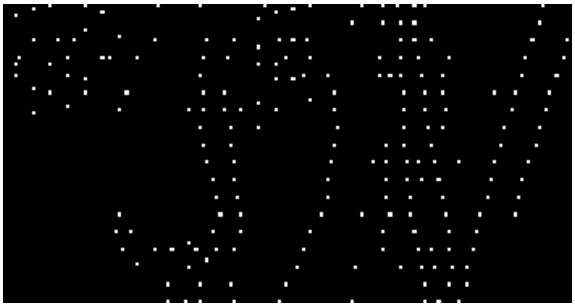

Figure 2. Lena's face area anchor points for a detail ratio 5.

The "detail ratio" parameter is used as follows: For a "detail ratio" value of "k", we scan every k^{th} row or column and mark anchor points only if they fall in these rows or columns. Thus, two consecutive anchor points along the same edge will be at least "k" pixels apart from each other. This hole in the actual edge will then be filled during the "edge drawing" process.

Figure 2 shows the anchor points for a section of the Lena image covering the eyes, nose and left hair for a detail ratio of 5. As can be seen, the distance between anchor points along the same edge is at least 5 pixels.

2.4. Edge Drawing by Anchor Point Linking

In this step, the goal is to draw (compute) actual edges by starting at an anchor point and tracing a path to the next anchor point along the same edge area. To guide our path, we make use of the gradient map and the edge directions computed in step 2.

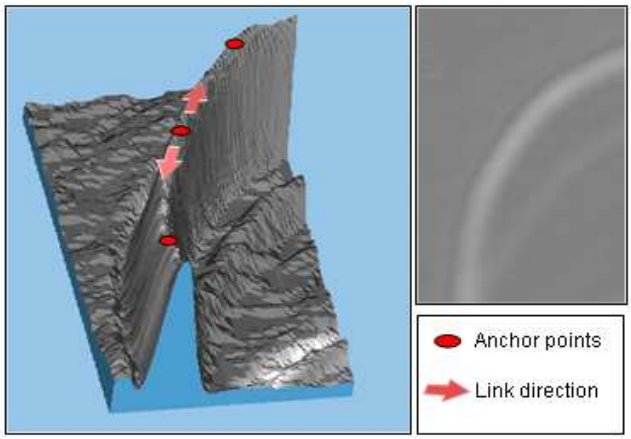

Figure 3. A pattern from Lena's hat (right) and its corresponding 3D structure (left).

Figure 3 illustrates how the edge drawing (or linking) process works. Red points in the figure illustrate anchor points, and arrows indicate the path direction. Assume we start at the anchor point in the middle. Since this is a vertical edge, we first link pixels to the north by going over the pixels having the maximum gradient values. This path should end up in the next anchor point to the north. We then start linking pixels to the south by going over the pixels having the maximum gradient values. This path should end up in the next anchor point to the south.

Figure 4 gives the details of the linking process. The numbers inside the red boxes indicate the gradient value at that pixel, i.e., $G_x + G_y$. The 3 anchor points are marked with red circles. Assume that we start at the anchor point in the middle. As we move up, we simply look at the 3 neighboring pixels to the north and go to the one having the maximum value. Specifically, if you are at pixel (i, j) moving up and the edge passing through (i, j) is a vertical edge, then we look at pixels (i-1, j-1), (i-1, j), (i-1, j+1) and simply go to the one having the maximum value. This linking process makes us go over the real edge (marked with yellow circles), until we hit the next anchor point. Similarly, as we go down from an anchor point, we look at the 3 neighboring pixels to the south, i.e., (i+1, j-1), (i+1, j) and (i+1, j+1), and go to the one having the maximum value. This leads us to the next anchor point to the south. The actual path drawn during the linking process is shown with yellow circles. Notice that the linking process draws a perfect contiguous 1-pixel wide edge.

If the edge was horizontal, then we would have traced a path to the right (east) and to the left (west) of the anchor point. Specifically, going left from (i, j), we would look at pixels (i-1, j-1), (i, j-1) and (i+1, j-1) and go to the one having the maximum value. Similarly, going right from (i, j), we would look at pixels (i-1, j+1), (i, j+1) and (i+1, j+1) and go to the one having the maximum value.

Figure 4. Illustration of the edge drawing process.

3. Experimental Results

In order to compare the results of our algorithm to existing edge detection algorithms, we picked the famous Canny edge detector [5]. Canny is considered to be the de-facto edge detection algorithm due to its running time performance and by its ability to detect only the "real" edge points. Although there are slower implementations of Canny, they are not suitable for real-time vision applications. OpenCV [17] implementation is the fastest known Canny implementation that we know of. Therefore, we compare the performance of our algorithm to OpenCV Canny implementation both in terms of running time, and the edge detection performance.

Figure 5 shows the edges detected by OpenCV Canny using thresholds (60, 100). Not only is the Canny edgemap very noisy, but it is also missing certain edges. To reduce the amount of noise in the edge map, you can increase the threshold values; but that would miss some more edges than it already does. To detect all edges, you can reduce the threshold values; but that would increase both the amount of noise in the edgemap and the running time of the algorithm. Therefore we have settled on the mentioned thresholds, and do not show other Canny results due to lack of space.

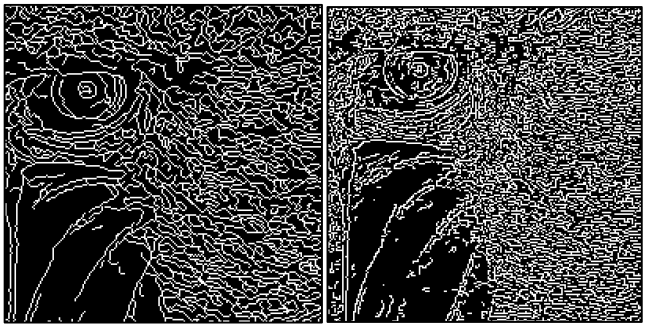

Figure 7. A close view from Mandril's edge maps Edge drawing's result (left), OpenCV Canny's result (right).

Figure 5. OpenCV Canny's edgemap of Lena.

Figure 6. Edgemap of Lena obtained with edge drawing.

Figure 6 shows the edgemap generated by our edge detection algorithm for a detail ratio of 4. We observe that a detail ratio of 4 gives a good compromise between edge details and noise. It is clear from the output that not only is our edgemap connected and very clean, but it also contains edges that Canny is not able to detect at all (see the circled areas in Figures 5 & 6).

Figure 7 shows close views of a particular region of the Mandrill image's resulted edgemaps. Because of its natural noisy structure, Mandrill is a very tough test image for all edge detection algorithms. It is easy to notice from the results that while edge drawing method draws continuous and distinguishable edges around the Mandrill's eye, Canny's output is more like garbage consisting of random pixels with no real connectivity.

Figure 8. Effect of the detail ratio on Peppers image: The edgemap with detail ratio of 5 (left), and 100 (right).

Figue 8 shows how tuning the detail ratio affects the resulting edgemap. Clearly, increasing the detail ratio causes the elimination of fine grained edges on the image, but detects the long and main edges.

Image	Edge Drawing (msec)				OpenCV Canny (msec)
	Image Smoothing	Anchor Point Detection	Edge Drawing	Total (msec)	
lena	2.60	3.80	1.14	7.54	7.79
boats	2.60	3.80	1.24	7.64	8.28
mandrill	2.60	5.19	2.04	9.84	11.41
peppers	2.60	3.67	1.12	7.40	7.55

Table 1. Running times of algorithms versus images.

Table 1 compares the running time of our algorithm to OpenCV Canny detector (cvCanny) for 4 different 512x512 grayscale images. All experiments were performed on a 2.2GHz Pentium machine running Windows XP. We set the detail ratio of our algorithm to 4 and Sobel threshold to 36, and Canny thresholds to (60, 100). We dissect the running time of our algorithm and show the amount of time that each step takes to execute. As seen from the results, our algorithm performs up to 16% faster than OpenCV Canny edge detector. You can find more results of edge drawing method at [18].

4. Concluding Remarks

We have proposed a robust, efficient edge detection algorithm. Our experiments show that not only is the proposed algorithm faster than the fastest known edge detector, namely, the OpenCV implementation of the Canny edge detector, but it also produces a very clean edgemap consisting of contiguous, one-pixel wide edges. The algorithm also has a novel *detail ratio* parameter, which allows the user adjust the amount of detail in the final edgemap. Thus an application can tailor-use our algorithm with different set of parameters and obtain different levels of detail in the edgemap. This ability is not available in many edge detection algorithms.

5. References

[1] L. G. Roberts, "Machine Perception of Three-Dimensional Solids," *Optical and Electro-Optical Information Processing*, J. T. Tippett, D. A. Berkowitz, L. C. Clapp, C. J. Koester, and A. Vanderburgh, Jr., eds., pp. 159-197. Cambridge, Mass.: MIT Press, 1965.

[2] I. E. Sobel, *Camera Models and Machine Perception*, PhD thesis, Stanford Univ., 1970.

[3] K. L. Boyer and S. Sarkar, "Assessing the State of the Art in Edge Detection: 1992," *SPIE,* vol. 1,708, *Applications of Artificial Intelligence X: Machine Vision and Robotics*, pp. 353-362, 1992.

[4] V. S. Nalwa and T. O. Binford, "On Detecting Edges," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 699-714, Nov. 1986.

[5] J. Canny, "A Computational Approach to Edge Detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 8, no. 6, pp. 679- 698, Nov. 1986.

[6] S. Sarkar and K. L. Boyer, "Optimal Infinite Impulse Response Zero Crossing Based Edge Detectors," *Computer Vision, Graphics, and Image Processing: Image Understanding*, vol. 54, pp. 224-243, Sept. 1991.

[7] L. A. Iverson and S.W. Zucker, "Logical/Linear Operators for Image Curves," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 17, no. 10, pp. 982-996, Oct. 1995.

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

[9] C. A. Rothwell, J. L. Mundy, W. Hoffman, and V.-D. Nguyen, "Driving Vision by Topology," *Int'l Symp. Computer Vision*, pp. 395- 400, Coral Gables, Fla., Nov. 1995.

[10] I. E. Abdou and W. K. Pratt, "Quantitative Design and Evaluation of Enhancement/Thresholding Edge Detectors," *Proc. IEEE*, vol. 67, no. 5, pp. 753-763, May 1979.

[11] V. Ramesh and R. M. Haralick, "Performance Characterization of Edge Detectors," *SPIE,* vol. 1,708, *Applications of Artificial Intelligence X: Machine Vision and Robotics*, pp. 252-266, 1992.

[12] J. R. Fram and E. S. Deutsch, "On the Quantitative Evaluation of Edge Detection Schemes and Their Comparison With Human Performance," *IEEE Trans. Computers*, vol. 24, no. 6, pp. 616-628, June 1975.

[13] D. J. Bryant and D. W. Bouldin, "Evaluation of Edge Operators Using Relative and Absolute Grading," *Proc. IEEE Computer Society Conf. Pattern Recognition and Image Processing*, pp. 138-145, Chicago, 1979.

[14] R. N. Strickland and D. K. Cheng, "Adaptable Edge Quality Metric," *Optical Eng.*, vol. 32, no. 5, pp. 944-951, May 1993.

[15] X. Y. Jiang, A. Hoover, G. Jean-Baptiste, D. Goldgof, K. Bowyer, and H. Bunke, "A Methodology for Evaluating Edge Detection Techniques for Range Images," *Proc. Asian Conf. Computer Vision*, pp. 415-419, 1995.

[16] T. Kanungo, M. Y. Jaisimha, J. Palmer, and R. M. Haralick, "A Methodology for Quantitative Performance Evaluation of Detection Algorithms," *IEEE Trans. Image Processing*, vol. 4, no. 12, pp. 1,667- 1,674, Dec. 1995.

[17] OpenCV - http://opencv.willowgarage.com/

[18] Samples of the edge drawing method is available at http://www.mm.anadolu.edu.tr/edgedrawing