GEDContours: A High Speed Contour Detector for Grayscale Images

Cuneyt Akinlar

cakinlar@anadolu.edu.tr

Anadolu University, Computer Engineering Department, Eskisehir, Turkey

Abstract

We propose a high-speed contour detector for grayscale images that works by combining the edge segments produced by Edge Drawing (ED) algorithm at multiple scales, thus the name GEDContours. Unlike most contour detectors, which return soft binary contour maps, GEDContours returns its result as a set of edge segments, each a contiguous chain of pixels. We evaluate the performance of GEDContours in the context of the RUG dataset, which has 40 grayscale images each of size 512x512 with human annotated ground truth contours, and compare its results with some of the best contour detectors found in the literature using the F-score metric. Experimental results show that GEDContours is among the best contour detectors with an overall F-score of 0.7011 on the RUG dataset, and takes an average of just 0.42 seconds as opposed to 74 seconds for the global Probability of boundary ultrametric contour maps (gPb-ucm) with an overall F-score of 0.7051, and 125 seconds for the sparse code gradients (scg) with an overall F-score of 0.6997. Due to its good performance and blazing speed, we believe that GEDContours will be very useful for high speed image processing and computer vision applications.

Keywords:
Edge, Contour and Boundary Detection, Edge Drawing (ED), RUG dataset, F-score

1. Introduction

Edge [1]-[6] and contour detection [8]-[22] are very important preprocessing steps and the building blocks of many computer vision algorithms. Contours are boundaries that separate different objects from each other, and help in such applications as object detection [24, 25, 26], recognition [27], matching, image segmentation, scene understanding, etc.

Traditional grayscale edge detection techniques work either by finding the zero crossings of the second derivative operator [1], or by finding the maximal pixels of the first derivative operator [2, 3, 4]. These are reactive techniques and work by eliminating non-edge pixels by such techniques as non-maximal suppression, hysteresis, edge thinning, morphological operators. The authors in [5] also use the first derivative of the gradient operator, but follow a completely different proactive method for edge detection: Instead of finding the edge pixels by elimination, the algorithm works by identifying a set of stable anchor points on top of the gradient map, and joins these anchors by what is called the smart routing procedure. Thus, the result of edge detection is returned not as a binary edge map as done by the traditional edge detectors, but as a set of edge segments, each a contiguous chain of pixels. In [6], the authors extend their algorithm by incorporating edge segment validation methodology due to the Helmholtz principle to validate the returned edge segments and eliminate false detections.

Although the single-scale edge detection algorithms mentioned above give out reasonable results for many images, they produce many superfluous edges for natural images especially with textured areas, or miss out on some important boundaries. To produce better boundary detection results in general, more complex contour detection algorithms have been proposed. Grigorescu et al. [8, 9] describe one such contour detector that works by augmenting the classical Canny edge detector by a step called the surround suppression that gives low response around the textured regions of the image. They also propose a benchmark dataset, named the RUG dataset [28], consisting of 40 grayscale natural images each of size 512x512 with human annotated ground truth boundaries, and evaluate their algorithms by comparing the proposed algorithm's contours with the ground truth contours. The benchmark dataset also gives the opportunity to quantitatively compare different contour detection algorithms together. Papari et al. [10] follow up on the idea of surround inhibition to suppress texture edges by proposing a biologically motivated multi-resolution contour detection and denoising algorithm. The authors extend their idea in [11] by adding some global analysis. The proposed operator, called the Adaptive Pseudo-Dilation (APD), identifies long curvilinear structures in the edge map that are in agreement with the Gestalt low of good continuation. The grouping algorithm is implemented in a multi-threshold contour detector, which makes APD less sensitive to the values of the input parameters. APD is evaluated in the context of the RUG dataset using the F-score metric. A survey of some of the edge and line oriented contour detection algorithms can be found in [12].

The authors in [15] pose the contour detection problem as finding a small set of smooth curves to represent the contours. They model the problem in a min-cover framework and use an approximation algorithm to solve it. The proposed algorithm is evaluated in the context of the Berkeley Segmentation Dataset and Benchmark (BSDS300) [13, 14]. Dollar et al [16] propose a learning based contour detector that trains a probabilistic boosting tree classifier by combining a large number of features across different scales. Using a large aperture, they provide a significant context to make the decision for each pixel. Arbelaez [17] presents a boundary extraction and segmentation algorithm for natural images, where the geometric structure of an image is represented as a family of nested segmentations called the ultra-metric contour map. Martin et al. [18] train a classifier using human labeled ground truth contours by combining local features such as brightness, color and texture. The output of the classifier gives the Probability of boundary (Pb) of each pixel in the image. Ren [19] extends this classifier by incorporating features from multiple scales and show that the proposed multi-scale classifier improves the performance between 20% to 50%. Martin et al. [20] improves Ren's work by adding globalization techniques to the classifier, and name the new algorithm the global Probability of boundary (gPb). Arbelaez et al. [21] then post-process the output of gPb using the idea presented in [17]. This algorithm named the gPb-ucm is among the best contour detectors found in the literature. Finally, Ren and Bo [22] use Space Code Gradients (scg) to improve the performance of gPb.

In this paper we propose a high-speed multi-scale contour detection algorithm for grayscale images that works by combining the edge segments detected by the Edge Drawing (ED) [5] algorithm at different scales, thus the name GEDContours. Unlike our previous contour detector EDContours [23], which returns its result as a multi-pixel wide soft contour map, GEDContours returns its result as a set of edge segments, each contiguous chain of pixels. This is important since the produced result can then be easily used for line detection [25], arc, circle and ellipse detection [26] and similar such object detection applications.

2. GEDContours: Multi-scale Contour Detection Using Grayscale Edge Drawing

In this section we present the details of the proposed multi-scale contour detector, GEDContours. Before we move into the details of the proposed algorithm, we first describe its precursor, EDContours [23], which is our previously proposed parameter-free soft contour detector for grayscale images.

Algorithm 1 gives the details of EDContours [23]. Our goal in designing EDContours was to come up with a high-speed, parameter-free contour detector for grayscale images. The idea was to combine the edge segments detected at different scales by our parameter-free edge segment detector, EDPF [6]. The observation was that at small scales, all details of the object boundaries get detected by EDPF; however, as the scale gets bigger, only the boundaries of the prominent objects get detected with small details getting erased by Gaussian smoothing [23]. So, if we are to combine the edge segments from different scales, a balanced contour detector can be designed.

Although EDContours listed in algorithm 1 satisfies our design goals, i.e., it is very fast and parameter-free, it produces a soft contour map that is obtained by simply adding up the edge segments detected by EDPF [6] at different scales (steps II and III in the algorithm). Although the general quality of the resulting contour map is good as can be seen in Fig. 1, the produced result is nothing more than a soft contour map with multi-pixel wide edge areas due to edge localization problems at higher scales. Further processing of this multi-pixel wide soft contour map for object detection and recognition problems is a formidable job.

Algorithm 1 EDContours [23]

```
Symbols used in the algorithm:
I: Input grayscale image

EDContours(I)
I. ContourMap[x, y] = 0;
for sigma = 1.0 to 4.0 increment by 0.25 do
    II. ES = EDPF(I, sigma);
    III. ContourMap[x, y] += ES[x, y];
end for
IV. Scale ContourMap to [0-255];
V. return ContourMap;
```

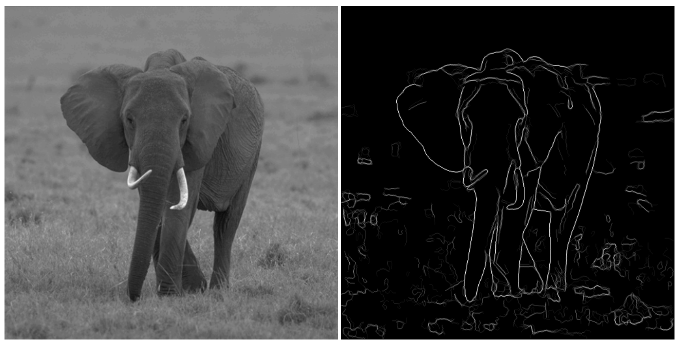
Figure 1: Image elephant_3 from the RUG dataset, and its soft contour map produced by EDContours.

To solve the problems associated with EDContours, we extend it in this paper and propose a new contour detector named GEDContours, short for Gray EDContours to distinguish it from EDContours, that also takes in a grayscale image, but instead produces a set of edge segments, each a one-pixel wide, contiguous chain of pixels. The edge segments are then readily available for higher level processing for things like line detection [25], arc, circle, and ellipse detection [26], etc.

Algorithm 2 gives the details of GEDContours, which takes two parameters: The input grayscale image and the gradient threshold. Similar to EDContours, GEDContours also works by detecting the edge segments by Edge Drawing with Validation (EDV) [5, 6] at different scales and adds them up (steps II and III) to obtain a soft contour map. But instead of returning this soft contour map with multi-pixel wide edge areas as done by EDContours, GEDContours first computes a set of one-pixel wide skeleton edge segments over the soft contour map (step IV), stacks up all non-skeleton contour pixels over the skeleton pixels (step V) and returns a thresholded version of this skeleton edge segments.

Algorithm 2 GEDContours

```
Symbols used in the algorithm:
I: Input grayscale image
thresh: Gradient threshold
ES: Edge Segments

GEDContours(I, thresh)
I. ContourMap[x, y] = 0;
for sigma = 0.25 to MaxSigma increment by 0.25 do
    II. ES = GrayEDV(I, sigma, thresh);
    III. ContourMap[x, y] += ES[x, y];
end for
IV. SkeltonES = ComputeContourSkeleton(ContourMap);
V. BoostSkeletonES(SkeletonES, ContourMap);
VI. Scale SkeletonES to [0-252] and threshold at 252;
VII. return SkeletonES;
```

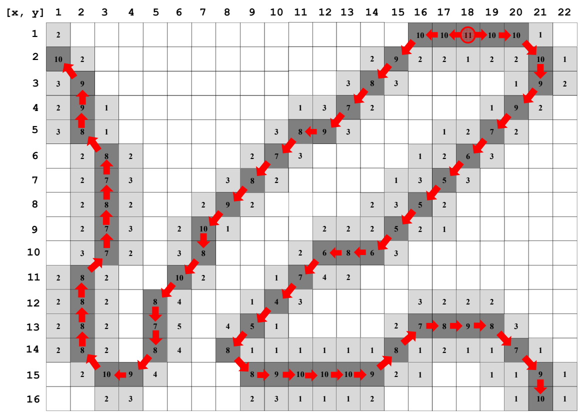
Figure 2: A section of an example cumulative soft contour map, and the generation of the skeleton edge segment starting at pixel (18, 1).

The foremost advantage of GEDContours over EDContours is the fact that it is returns its result as a set of edge segments, each a one-pixel wide chain of pixels instead of a multi-pixel wide soft contour map. This is achieved by the final two steps of the algorithm: Computation of the skeleton edge segments (step IV), and boosting of the skeleton by the non-skeleton contour pixels (step V).

Computation of the skeleton edge segments over the contour map follows an 8-directional edge linking procedure and is illustrated in Fig. 2 for a section of an example soft contour map. Each number in a gray pixel in the figure represents the number of times an edge segment has passed through that pixel during step III of the algorithm. As expected, we have a multi-pixel wide soft contour map with the actual edge pixels (dark gray pixels) having larger values. To compute the skeleton edge segment over this contour map, our idea is to start at the pixel having the largest value, which in our example is the pixel (18, 1), and walk in two directions creating two chains, which are then combined together into one single edge segment. The initial edge direction from the starting pixel is assumed to be towards one of the 8-neighbors having the largest value. In the figure, the two walk directions are Left and Right as illustrated because

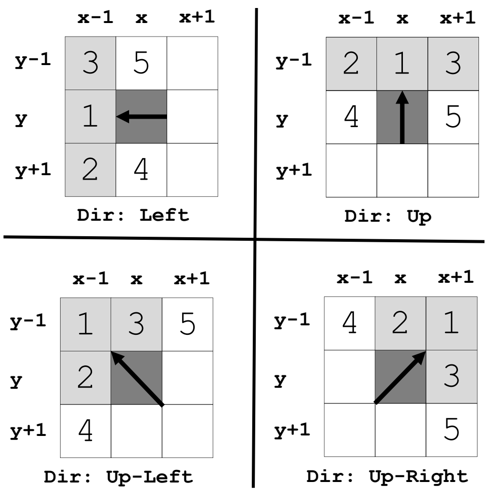

Figure 3: Four walk directions: Left, Up, Up-Left, Up-Right. The other four directions, i.e., Right, Down, Down-Right, and Down-Left, are the symmetric versions of these.

(17, 1) and (19, 1) are the two neighbors having the largest values. Once a walk starts in one direction, it continues following the rules depicted in Fig. 3. The idea during the walk is to first look at the three neighbors in the walk direction, and go to the neighbor having the largest value. For example, if we are at pixel (x, y) and walking Left as depicted in Fig. 3, we first check the three neighbors (x-1, y), (x-1, y+1), and (x-1, y-1) labeled as 1, 2, and 3 in the figure, and move to the one having the largest value. If we move to (x-1, y), the current walk direction remains Left. However, if we move to (x-1, y-1), then the current walk direction changes to Up-Left. If we move to (x-1, y+1), then the current walk direction changes to Down-Left. If, however, all three neighbors to the left are 0, then we check the two neighbors marked 4 and 5, and move to the one having the bigger value. The goal of this check is to make the currect walk turn radically and continue in case we reach the end of a contour area. If pixels marked 4 and 5 are both 0, then the walk stops. The other three directions depicted in Fig. 3, i.e., Up, Up-Left, and Up-Right, follow a similar procedure to determine the next pixel during the walk. The remaining four directions, i.e., Right, Down, Down-Right, and Down-Left, are the symmetric versions of the directions depicted in Fig. 3, and are therefore not illustrated.

Returning back to the edge segment creation example in Fig 2, we start at pixel (18, 1) and walk Left. The left walk continues until (16, 1), where we turn Down-Left. The Down-Left walk continues until (12, 5), where we again turn Left. Following the walk directions depicted in Fig. 3, this chain will end up in pixel (1, 2) as traced by the red arrows and dark gray pixels. We then start a new walk to the right from our starting pixel (18, 1). This time, the right walk continues until (20, 1), where we turn Down-Right, then Down, then Down-Left. The tip of this chain ends up at pixel (21, 16) as traced by the red arrows and dark gray pixels. Pay special attention to pixel (8, 14), where the three neighbors in the Down-Left direction are 0. Therefore, we check the pixels labeled 4 and 5 in Fig. 3 and move Down-Right. Without the check for neighbor pixels

Table 1: F-score of GEDContours on the RUG dataset as *MaxSigma* is increased.

MaxSigma	F-score
2.50	.6978
2.75	.6994
3.00	**.7011**
3.25	.6988
3.50	.6968

labeled 4 and 5, this turn would not be possible.

After the computation of the skeleton edge segments, the final step of GEDContours is to stack up the neighboring non-skeleton contour pixels on top of the skeleton edge segment pixels. Here the idea is very simple: For each non-skeleton pixel (light gray pixels in Fig. 2), compute the closest skeleton pixel in Euclidean distance and add its value to the value of the skeleton pixel. For example, for non-skeleton pixel at (18, 2), the closest skeleton pixel is (18, 1); so we simply add its value 1 to 11 to make the skeleton pixel's value 12.

3. Experimental Results

In this section we evaluate the performance GEDContours. To make a fair evaluation and compare GEDContours's results with the results of some state of the art contour detectors, we make use of standard RUG dataset [8, 28], which consists of 40 grayscale images each of size 512x512 with human-annotated ground truth contours.

The quantitative evaluation is based on the F-score metric defined as follows: Let the boundaries returned by an algorithm for an image be A, and the human annotated ground truth boundary be GT. Then precision $P = \frac{(A \cap GT)}{A}$, recall $R = \frac{(A \cap GT)}{GT}$, and F-score = $\frac{(2PR)}{P+R}$.

Recall from algorithm 2 that GEDContours has an internal parameter named *MaxSigma*, which represents the maximum scale to be used during contour computation. Table 1 shows the best F-score attained by GEDContours as *MaxSigma* is increased from 2.50 to 3.50. As seen from the table, F-score is maximized when *MaxSigma* becomes 3.0, and starts decreasing for bigger values. We therefore, fix *MaxSigma* to 3.0 for all experiments in the rest of this section. Also notice that other *MaxSigma* values also produce F-score values that are close to the best result; therefore, they can also be used if desired. The other internal parameter of GEDContours is the sigma increment. Our experiments have shown that a sigma increment of 0.25 gives the best results in general. Values smaller than 0.25 do not increase the performance but increase the running time, and values bigger than 0.25 reduce the performance; therefore, we set the sigma increment to 0.25.

To measure the effects of the user supplied gradient threshold on the performance of GEDContours, we changed the gradient threshold and computed the F-score on the RUG dataset. Fig. 6 plots precision, recall and F-score as the gradient threshold is increased from 12 to 48. As expected, for small values of the gradient threshold, the recall is high but the precision is low. As the gradient threshold is increased, recall drops and precision increases. The F-score is maximized at 0.7011 for the gradient threshold value of 30. But notice that even for a variety of threshold values, F-score stays above 0.67, which is better than most contour detectors as we show below.

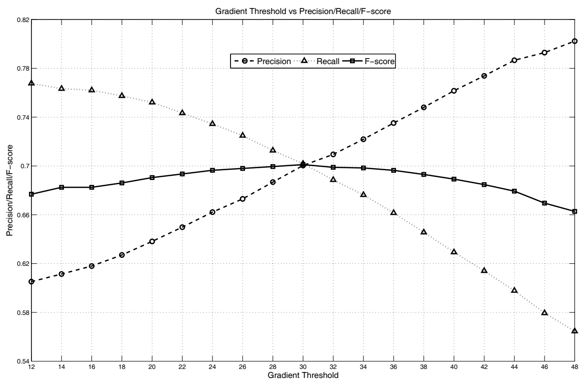

Figure 6: Precision, recall and F-score metrics for GEDContours as the gradient threshold is increased from 12 to 48. The best F-score of 0.7011 is attained at threshold 30.

Table 2: The best F-score values by different algorithms on the RUG dataset.

Algorithm	Recall	Precision	F-score
Edison [7]	0.8633	0.3055	0.4514
Surround [9]	0.7837	0.3594	0.4928
Canny [2]	0.6063	0.6059	0.6061
ED [5]	0.6171	0.6160	0.6165
EDContours [23]	0.6598	0.6152	0.6367
APD [11]	0.6337	0.6839	0.6578
scg [22]	0.6779	0.7229	0.6997
GEDContours	0.7018	0.7003	**0.7011**
gPb-ucm [21]	0.6966	0.7139	0.7051

Having fixed *MaxSigma* at 3.00, and the gradient threshold at 30, we now show some qualitative and quantitative experiments to compare GEDContours with the state of the art contour detectors. Fig. 4 and Fig. 5 show 12 images from the RUG dataset together with human-annotated ground truth contours, and the corresponding thresholded best contour detection results by four state of the art contour detection algorithms: Advanced Pseudo Dilation (APD) [11], global Probability of boundary-ultrametric contour maps (gPb-ucm) [21], sparse code gradients (scg) [22], and GEDContours. For all of these images, GEDContours produces the best results as measured by the F-score metric. The visual analysis of GEDContours's results compared to the ground truth contours also confirm the high quality nature of GEDContours's outputs. Notice that for some images, e.g., bear_2 and bear_6, GEDContours produces way better results than the other contour detectors. For example, in bear_6, only GEDContours is able to detect the ridge that traces the back of the bear while the other detectors are not able to detect any part of it in a meaningful manner.

Table 2 lists the best F-scores attained by most of the state of the art edge and contour detection algorithms on the RUG dataset. We see from the table that single-space edge detectors

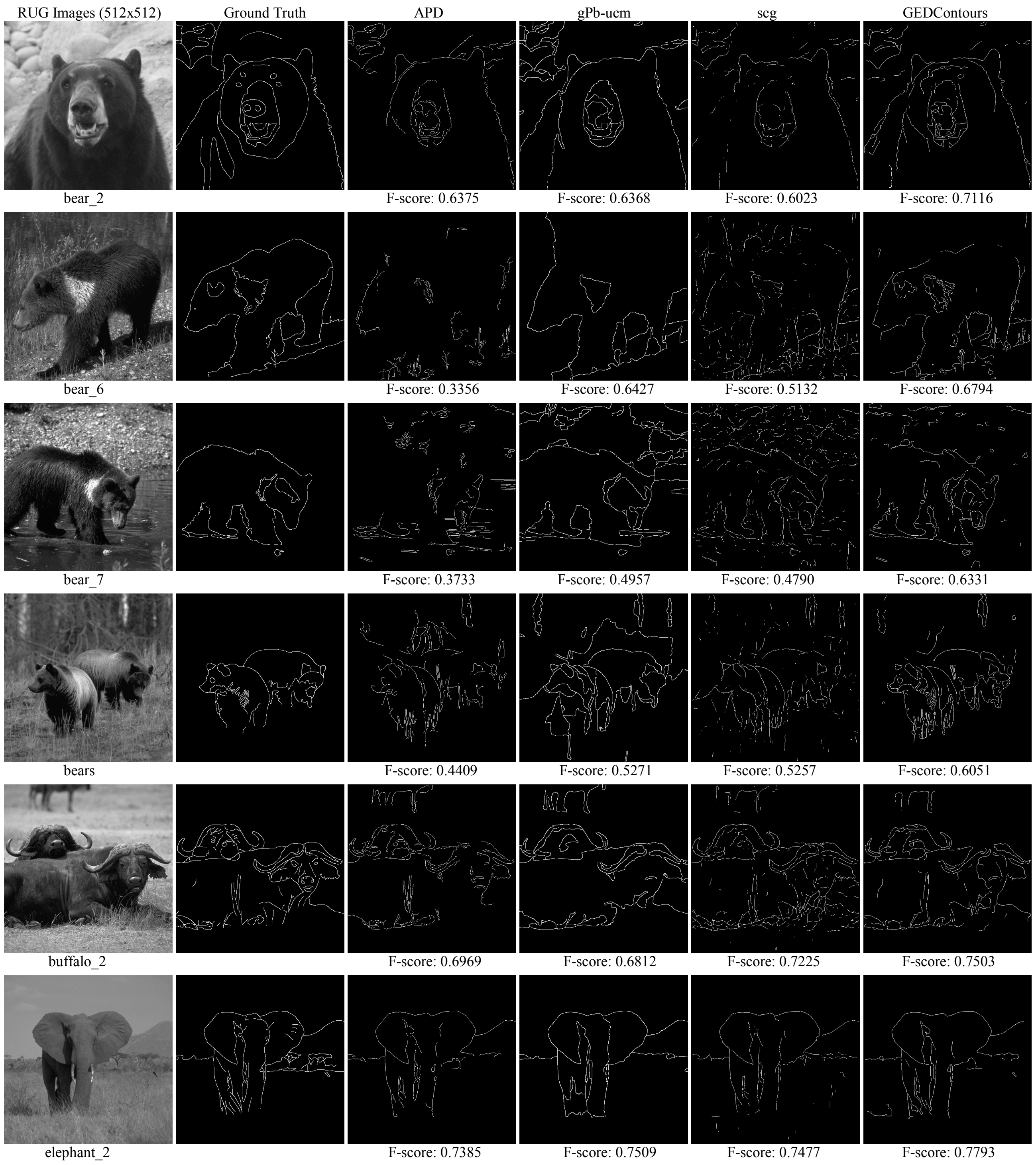

Figure 4: Six images from the RUG dataset [8, 28] together with human-annotated ground truth contours, and the corresponding thresholded best contour detection results by four algorithms, Advanced Pseudo Dilation (APD) [11], global Probability of boundary-ultrametric contour maps (gPb-ucm) [21], sparse code gradients (scg) [22], and GEDContours. For all of these images, GEDContours produces the best results as measured by the F-score metric.

Edison [7], Canny [2], and ED [5] can achieve best results up to about 0.61. More complex and state of the art contour detectors Advanced Pseudo Dilation (APD) [11] results in 0.6578, Sparse Code Gradients (scg) [22] has an F-score of 0.6997, and global Probability of boundary ultrametric contour maps (gPb-ucm) [21] 0.7051. GEDContours performs the second best among all contour detectors producing a F-score value of 0.7011, which much better than APD and very close to gPb-ucm.

Table 3 lists the average running times of state of the art edge and contour detectors for images of size 512x512 in the RUG dataset. The running times were obtained on a machine hav-

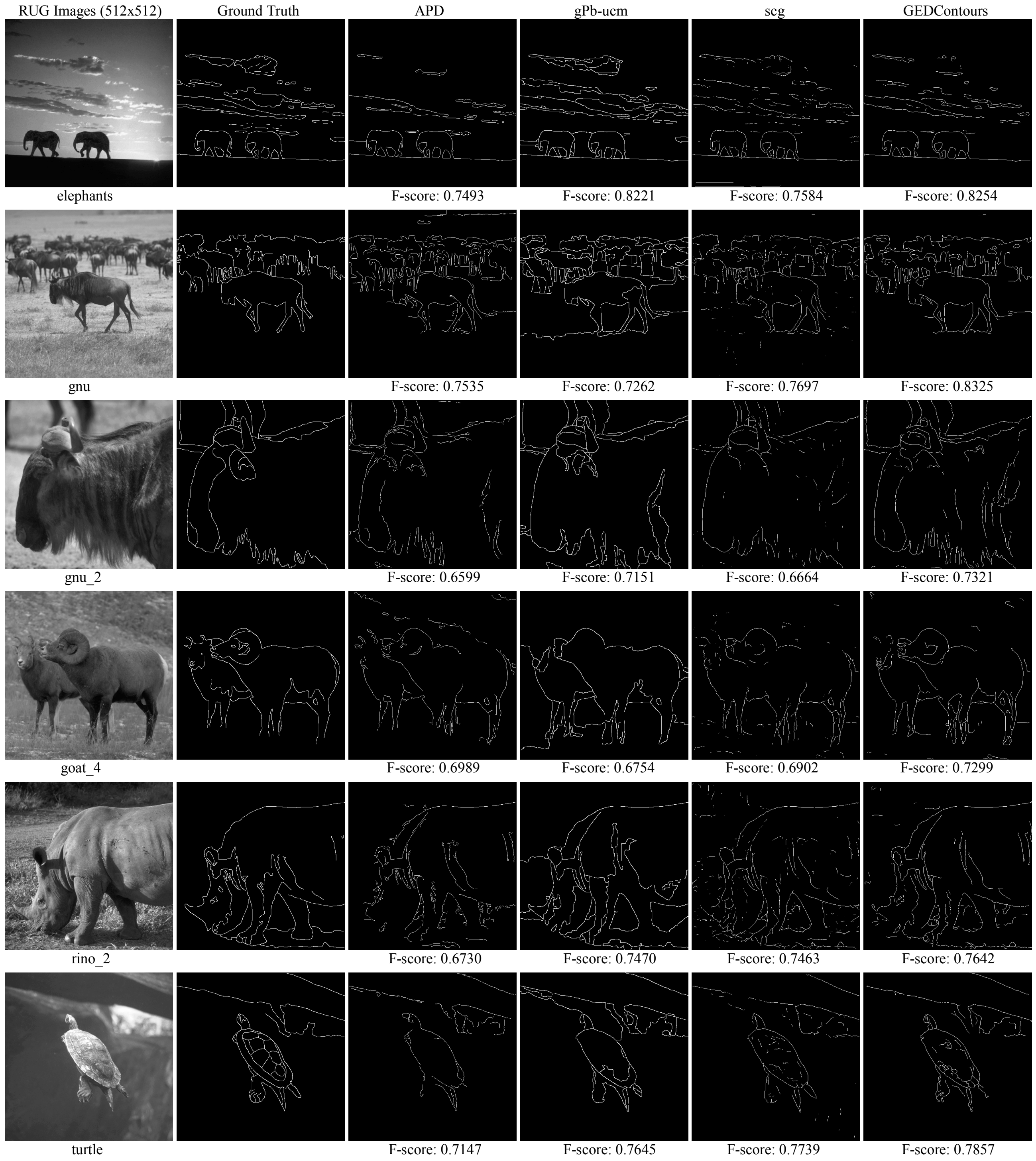

Figure 5: Six images from the RUG dataset [8, 28] together with human-annotated ground truth contours, and the corresponding thresholded best contour detection results by four algorithms, Advanced Pseudo Dilation (APD) [11], global Probability of boundary-ultrametric contour maps (gPb-ucm) [21], sparse code gradients (scg) [22], and GEDContours. For all of these images, GEDContours produces the best results as measured by the F-score metric.

ing an Intel Xeon E5-1630 processor running at 3.70 GHz and 32 GB of memory. As seen from the table, single-space edge detectors, e.g., Canny and ED, are very fast, taking just 5 milliseconds to execute. However, recall from Table 2 that Canny and ED do not produce good overall results. APD has a reasonable running time taking about 1.10 seconds per image [1], but its performance, although better than Canny and ED, is also not very good. Best producing contour detectors, i.e., gPb-ucm

[1]The running time for APD was obtained by projecting its running time reported in [11] to the CPU used in our experiments.

Table 3: Average running time of some state of the art edge and contour detection algorithms for images of size 512x512 in the RUG dataset. The running times were obtained on a machine having an Intel Xeon E5-1630 processor running at 3.70 GHz and 32 GB of memory.

Algorithm	Running time (sec)
Canny [2]	0.005
ED [5]	0.005
GEDContours	0.42
APD [11]	1.10
gPb-ucm [21]	74
scg [22]	125

and scg, however, take extremely long times; about 74 and 125 seconds respectively. GEDContours, on the other hand, takes just 420 milliseconds per image on average and still produces the second best result on the RUG dataset. GEDContours is thus about 175 times faster than gPb-ucm, and about 300 times faster than scg, making it very suitable for high-speed computer vision applications.

4. Conclusions

We present a high-speed contour detector for grayscale images, named GEDContours, that first computes a multi-pixel wide cumulative soft contour map by combining the edge segments detected by Gray Edge Drawing with Validation (GrayEDV) at different scales, which is then thinned down, thresholded, and returned to the user as a set of edge segments, each a contiguous chain of pixels. The thinning involves the computation of a set of skeleton edge segments over the contour map and stacking up all non-skeleton contour pixels on top of the skeleton edge segment pixels. We compare GEDContours's performance with different state of the art contour detection algorithms on the 40 image RUG dataset and benchmark, and conclude that GEDContours produces one of the best results in the literature while running more than 175 times faster than the alternative algorithms. The interested reader can download GEDContours's code and results on the RUG dataset from its Web-site at [30]. Our future work is to look into the problem of determining the optimal gradient threshold for an input image, and to incorporate texture gradient analysis to EDContours to improve its performance.

References

[1] D. Marr, E. Hildreth, Theory of Edge Detection, Proceedings of the Royal Society of London, Biological Sciences, 207(1167) (1980), pp. 187-217.

[2] J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6) (1986), pp. 679-698.

[3] V.S. Nalwa, T.O. Binford, On detecting edges, IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6) (1986), pp. 699-714.

[4] E. Deriche, Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector, International Journal of Computer Vision, 1(2) (1987), pp. 167-187.

[5] C. Topal, C. Akinlar, Edge Drawing: A Combined Real-Time Edge and Segment Detector, Journal of Visual Communication and Image Representation, 23(6) (2012), pp. 862-872.

[6] C. Akinlar, C. Topal, EDPF: A Real-time Parameter-free Edge Segment Detector with a False Detection Control, International Journal of Pattern Recognition and Artificial Intelligence, 26(1) (2012).

[7] P. Meer, B. Georgescu, Edge detection with embedded confidence, IEEE Transactions on Pattern Analysis and Machine Intelligence 23(12) (2001), pp. 1351-1365.

[8] C. Grigorescu, N. Petkov, M.A. Westenberg, Contour Detection Based on non-Classical Receptive Field Inhibition, IEEE Transactions on Image Processing, 12(7) (2003), pp. 729-739.

[9] C. Grigorescu, N. Petkov, M.A. Westenberg, Contour and boundary detection improved by surround suppression of texture edges, Journal of Image and Vision Computing, 22(8) (2004), pp. 609-622.

[10] G. Papari, P. Campisi, N. Petkov, A biologically motivated multiresolution approach to contour detection, EURASIP Journal on Advances in Signal Processing, Special issue on Human Perception, (2007).

[11] G. Papari, N. Petkov, Adaptive Pseudo Dilation for Gestalt Edge Grouping and Contour Detection, IEEE Transactions on Image Processing, 17(10) (2008), pp. 1950-1962.

[12] G. Papari, N. Petkov, Edge and line oriented contour detection: State of the art, Image and Vision Computing, 29(2-3) (2011), pp. 79-103.

[13] D. Martin, C. Fowlkes, D. Tal, J. Malik, A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics, IEEE International Conference on Computer Vision (ICCV), (2001), pp. 416-423.

[14] The Berkeley Segmentation Dataset and Benchmark Web site, `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds`, Accessed: February, 2016.

[15] P. Felzenszwalb, D. McAllester, A Min-Cover Approach for Finding Salient Curves, IEEE Computer Vision and Pattern Recognition Workshop (CVPRW), (2006).

[16] P. Dollar, Z. Tu, S. Belongie, Supervised Learning of Edges and Object Boundaries, IEEE Computer Vision and Pattern Recognition (CVPR), (2006), pp. 1964-1971.

[17] P. Arbelaez, Boundary Extraction in Natural Images Using Ultrametric Contour Maps, IEEE Computer Vision and Pattern Recognition Workshop (CVPRW), (2006).

[18] D. Martin, C. Fowlkes, J. Malik, Learning to detect natural image boundaries using local brightness, color and texture cues, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 26(5) (2004), pp. 530-549.

[19] X. Ren, Multi-Scale Improves Boundary Detection in Natural Images, European Conference on Computer Vision (ECCV) - Lecture Notes in Computer Science, 5304 (2008), pp. 533-545.

[20] M. Maire, P. Arbelaez, C. Fowlkes, J. Malik, Using Contours to Detect and Localize Junctions in Natural Images, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (2008), pp. 1-8.

[21] P. Arbelaez, M. Maire, C. Fowlkes, J. Malik, Contour Detection and Hierarchical Image Segmentation, IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 33(5) (2011), pp. 898-916.

[22] X. Ren, L. Bo, Discriminatively Trained Sparse Code Gradients for Contour Detection, Advances in Neural Information Processing Systems (NIPS), (2012).

[23] C. Akinlar, C. Topal, EDContours: High-Speed Parameter Free Contour Detector Using EDPF, IEEE International Symposium on Multimedia (ISM), (2011), pp. 153-156.

[24] V. Ferrari, T. Tuytelaars, L.V. Gool, Object detection by contour segment networks, European Conference on Computer Vision, (2006), pp. 1428.

[25] C. Akinlar, C. Topal, EDLines: A real-time line segment detector with a false detection control, Pattern Recognition Letters, 32(13) (2011), pp. 1633-1642.

[26] C. Akinlar, C. Topal, EDCircles: A real-time circle detector with a false detection control, Pattern Recognition, 46(3) (2013), pp. 725-740.

[27] C. Gu, J. Lim, P. Arbelaez, J. Malik, Recognition using regions, Computer Vision and Pattern Recognition (CVPR), (2009), pp. 1030-1037.

[28] The RUG Dataset Web site, `http://www.cs.rug.nl/~imaging/APD/rug/rug.html`, Accessed: February, 2016.

[29] Sparse Code Gradients (scg) Web site, `http://homes.cs.washington.edu/~xren/sparse_contour_gradients_v1.1.zip`, Accessed: February, 2016.

[30] Gray EDContours (GEDContours) Web site, `http://ceng.anadolu.edu.tr/cv/GEDContours`, Accessed: February, 2016.