J. Vis. Commun. Image R. 23 (2012) 862–872

Contents lists available at SciVerse ScienceDirect

J. Vis. Commun. Image R.

journal homepage: www.elsevier.com/locate/jvci

Edge Drawing: A combined real-time edge and segment detector ☆

Cihan Topal *, Cuneyt Akinlar

Department of Computer Engineering, Anadolu University, 26470 Eskisehir, Turkey

ARTICLE INFO

Article history:
Received 6 December 2011
Accepted 7 May 2012
Available online 14 May 2012

Keywords:
Edge detection
Edge segment detection
Edge quality metrics
Real-time imaging

ABSTRACT

We present a novel edge segment detection algorithm that runs real-time and produces high quality edge segments, each of which is a linear pixel chain. Unlike traditional edge detectors, which work on the thresholded gradient magnitude cluster to determine edge elements, our method first spots sparse points along rows and columns called anchors, and then joins these anchors via a smart, heuristic edge tracing procedure, hence the name Edge Drawing (ED). ED produces edge maps that always consist of clean, perfectly contiguous, well-localized, one-pixel wide edges. Edge quality metrics are inherently satisfied without a further edge linking procedure. In addition, ED is also capable of outputting the result in vector form as an array of chain-wise edge segments. Experiments on a variety of images show that ED produces high quality edge maps and runs up to 10% faster than the fastest known implementation of the Canny edge detector (OpenCV's implementation).

1. Introduction

Edge detection is a very basic and important problem, and is a popular first step in many computer vision and pattern recognition applications [2–17]. A high quality edge map, which is defined to consist of perfectly contiguous, well-localized, non-jittered and one-pixel wide edge segments [18–24], significantly improves the performance of the rest of the computer vision and pattern recognition application; whereas, a low quality edge map, which has poor connectivity, edge width that varies, erroneous branching and notches, etc., significantly impairs the success of the rest of the application.

A generic way to detect edges in traditional edge detection algorithms is to apply a sequence of filters to the image. The first step is usually to pass the image through a low pass filter in an effort to reduce noise and smooth out the image. Since edges are high frequency components of an image, a high pass filter (a derivative or gradient operator, e.g., Prewitt [1], Sobel and Feldman [2], Scharr [3], etc.) is then applied to extract out areas where edges would be located. The output of the gradient operator is called the gradient map, which is usually thresholded to suppress low frequency, non-edged areas of the image. Pixels that survive the gradient thresholding constitute what is called the *edge areas*, which consists of several pixel-wide, thick strips of the image (Fig. 3(a). The final task is then to go over the remaining pixels in the edge areas and obtain thin, contiguous, non-jittered and well-localized edges. This is the most important and difficult step in edge detection, and is achieved by such techniques as skeletonization, non-maximal suppression, edge thinning, or the application of morphological operators. But these techniques evaluate each pixel in the edge areas to be a potential edge element (edgel) in an isolated and individual manner. Canny edge detector, for instance, tests each pixel for being an edgel by comparing its gradient value with two of its neighbors, and if it (among a pre-set thresholded gradient magnitudes) is higher than its neighbors' gradient values along the same gradient direction, then the pixel is marked as an edgel. Such short-sighted local reasoning leads to low quality edge maps consisting individual, garbage-like edgel formations with multi-pixel wide, discontinuous edge segments.

To improve the quality of the edge maps produced by traditional edge detectors, many post-processing techniques [25–35] have been proposed. These techniques take a binary edge map as input and try to improve the edge map quality by employing rule-based methodologies with pixel-wise templates in an effort to clean up the edge map and link possible discontinuities. They also try to optimize the modal structure of edge segments with the help of morphological operations.

To better understand why edge linking and optimization studies constitute a crowded set in the literature, refer to Fig. 1, which shows the edge map for the famous Lena image produced by OpenCV Canny [4,36] algorithm, where some of the low quality edge patches have been highlighted. We see from the figure that the edge map has many low quality components consisting of ragged, discontinuous, unattended, multi-pixel wide edges, while some prominent details in the image, e.g., the left vertical line in patch 'a', are undetected. Threshold values can be decreased in an attempt to detect more details, but the resulting edge map would contain more garbage-like edge instances instead of mean-

☆ This work is partially supported by The Scientific and Technological Research Council of Turkey (TUBITAK) with Grant No. 111E053.

* Corresponding author.
E-mail addresses: cihant@andolu.edu.tr (C. Topal), cakinlar@andolu.edu.tr (C. Akinlar).

1047-3203/$ - see front matter © 2012 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.jvcir.2012.05.004

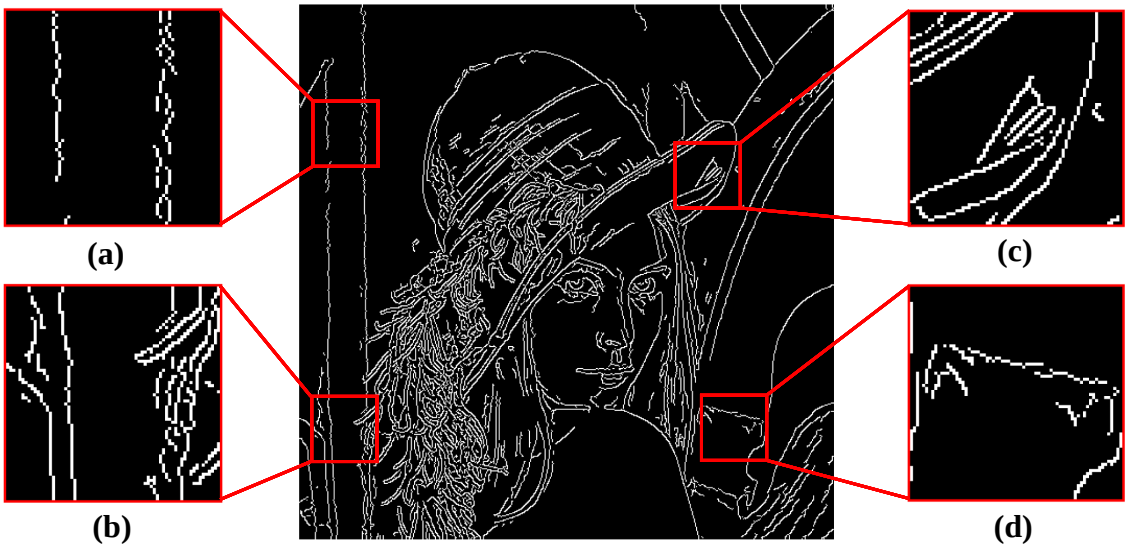

Fig. 1. In the middle: Lena's edge map result by OpenCV Canny with parameters; low threshold: 30, high threshold: 60, aperture: 3. Lena image was first smoothed by a 5×5 Gaussian kernel with a $\sigma = 1$. (a) Notch instances and discontinuities caused by noise, (b) unattended edge formations, (c) two-pixels wide edge segment on the boundaries of Lena's hat, (d) unsmooth edge segment having discontinuities.

ingful details. Although edge linking methods may improve the visual quality of the edge map by making use of various rules and templates, they still remain insufficient to produce high quality edge maps. Moreover, an edge linking method can not create a more "correct" edge map, even though it can improve the structural quality of the edge segments.

In real images sharp changes take part in a continuous manner, and the boundaries of the objects in the image lie contiguously unless they are obstructed by other objects. Yet, traditional edge detectors rarely make use of this fact, and employ a strictly local reasoning to determine edgels, which results in unattended, discontinuous, multi-pixel wide edge segment formations as shown in Fig. 1. However, we human beings employ a very different and basic strategy to determine the edges on an image. Specifically, if a person is asked to determine the boundaries of all objects in a given image like the experiments in [23], s/he would not evaluate each pixel individually, but rather, s/he would pick a point on an edge of an object and draw its entire boundary by joining consecutive pixels on the boundary. The person would then pick another point from the boundary of another object and draw its entire boundary. This process would continue until the person thinks that s/he has completed drawing the all edges in the image. Thus, connectedness, localization and smoothness of the edges are guaranteed.

Conceptually, a human's high-level cognitive boundary detection method is similar to children's dot-to-dot boundary completion puzzles, where a child is given a dotted boundary of an object, and s/he is asked to complete all the boundaries by connecting successive dots. As an example consider Fig. 2(a), which shows the famous Lena image's edge map computation problem illustrated as a dot-to-dot boundary completion puzzle. When

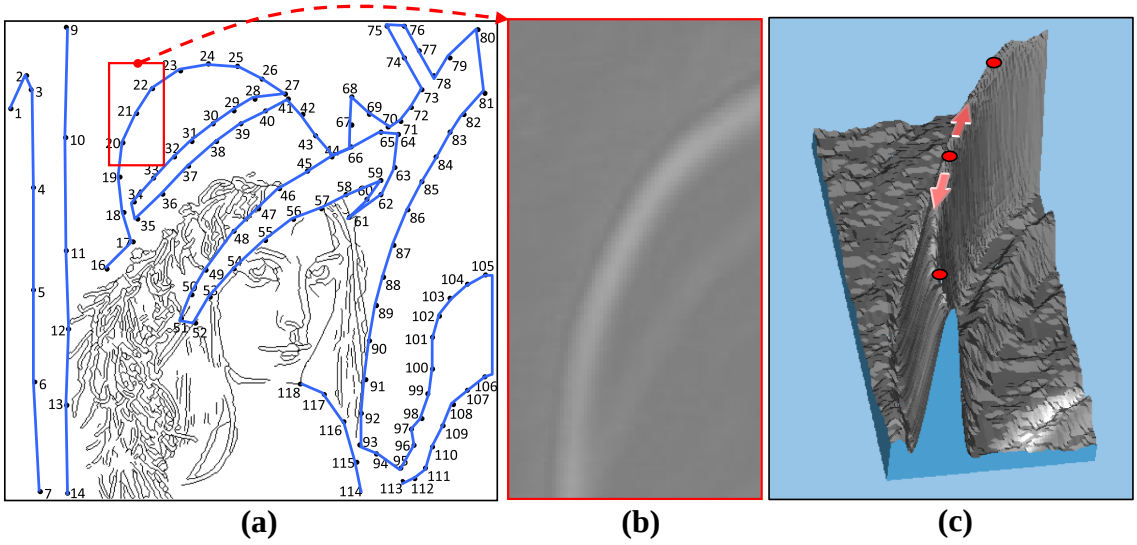

Fig. 2. (a) Lena's edge map computation problem formulated as a dot-to-dot boundary completion puzzle. When consecutive dots are connected, we obtain a high quality edge map consisting of clean, one-pixel wide, contiguous edge segments. (b) A patch from the gradient map of Lenas hat, (c) The 3D illustration of the patch in (b). Two sample anchors are marked with red (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.) circles, arrows indicate the anchor linking directions.

successive dots are connected, we obtain a high quality edge map for the Lena image.

In this paper, we propose an edge segment detection algorithm that applies the high-level cognitive reasoning employed in dot-to-dot boundary completion puzzles to edge segment detection. The proposed algorithm emulates the ideas in dot-to-dot boundary completion puzzles by computing a set of points (called the *anchors*) that represent stable landmarks on the sharp intensity changes of a given image (Section 2.3), and then uses a smart routing algorithm to connect successive anchors (Section 2.4). The proposed algorithm literally draws an edge segment between successive anchors, hence the name *Edge Drawing* (ED) [37], similar to a person's connecting successive dots in a dot-to-dot boundary completion puzzle by drawing an edge segment between them (refer to Fig. 2). This is the most important part of the contribution of this study and should not be mixed up with edge linking paradigm where an already computed edge map is taken as input and improved with respect to the edge quality metrics. Because ED detects the edges from the beginning just by computing a few number of points, high quality edges can be obtained without the need of a post processing method. Moreover, ED is able to output the result in vector form as an array of chain-wise detected edge segments (Section 3).

The rest of the paper is organized as follows: Section 2 describes step-by-step how ED processes an image to produce the final edge map. Section 3 talks about the importance of edge segment detection and why ED can produce high-quality edge segments. Section 4 analyzes the performance of ED from different perspectives, and Section 5 concludes the paper.

2. Edge Drawing algorithm (ED)

ED works on grayscale images and is comprised of four main steps:

(1) Suppression of noise by Gaussian filtering.
(2) Computation of the gradient magnitude and edge direction maps.
(3) Extraction of the anchors.
(4) Connecting the anchors by smart routing.

2.1. Gaussian filtering

the very first step of ED is similar to most image processing methods: we convolve the image with a Gaussian kernel to suppress noise and smooth out the image. For all results given in this paper, a 5×5 Gaussian kernel with $\sigma = 1$ is used.

2.2. Computation of the gradient magnitude and edge direction maps

The next step of ED is to compute the gradient magnitude and edge direction maps. Using the smoothed image, we compute the horizontal and vertical gradients, G_x and G_y, respectively. Any of the well-known gradient operators, e.g., Prewitt [1], Sobel and Feldman [2], Scharr [3], etc., can be used in this step. The gradient magnitude G at a pixel is then obtained by the formula $G = \sqrt{G_x^2 + G_y^2}$. Alternatively, gradient magnitudes can be computed by simply adding G_x and G_y, i.e., $G = |G_x| + |G_y|$ for faster computation and the threshold value can be rescaled.

Simultaneously with the gradient magnitude map, the edge direction map is also computed by comparing the horizontal and vertical gradients at each pixel. If the horizontal gradient is bigger, i.e., $|G_x| \geqslant |G_y|$, a vertical edge is assumed to pass through the pixel. Otherwise, a horizontal edge is assumed to pass through the pixel. Our experiments have shown that using two gradient directions were enough to efficiently route the connecting process of anchors with the routing mechanisms shown in Fig. 4(b) and (c). Increasing the number of gradient angles simply increases the computational time without improving the anchor connection process.

To help us suppress image locations where have smooth intensity changes we threshold the gradient map and eliminate the so-called "weak" pixels. Fig. 3(a) shows the Lena image, and the corresponding thresholded gradient map. This thresholded gradient cluster represents the edge areas because all edgels of the image must reside within the boundaries of this cluster. In other words, the final edge map will be a subset of this cluster and none of the eliminated black pixels will contain an edgel.

The first two steps of ED (Gaussian filtering and gradient map computation) are similar to most edge detection algorithms. After the computation of the edge areas, however, ED follows a very unorthodox approach: Instead of testing individual pixels within the edge areas for being edgels, ED first spots a subset of pixels (called the anchors) and then connects these anchors via a smart heuristic routing procedure.

2.3. Extraction of the anchors

Intuitively, anchors should correspond to points where the edges would put over. The points that satisfy this condition may

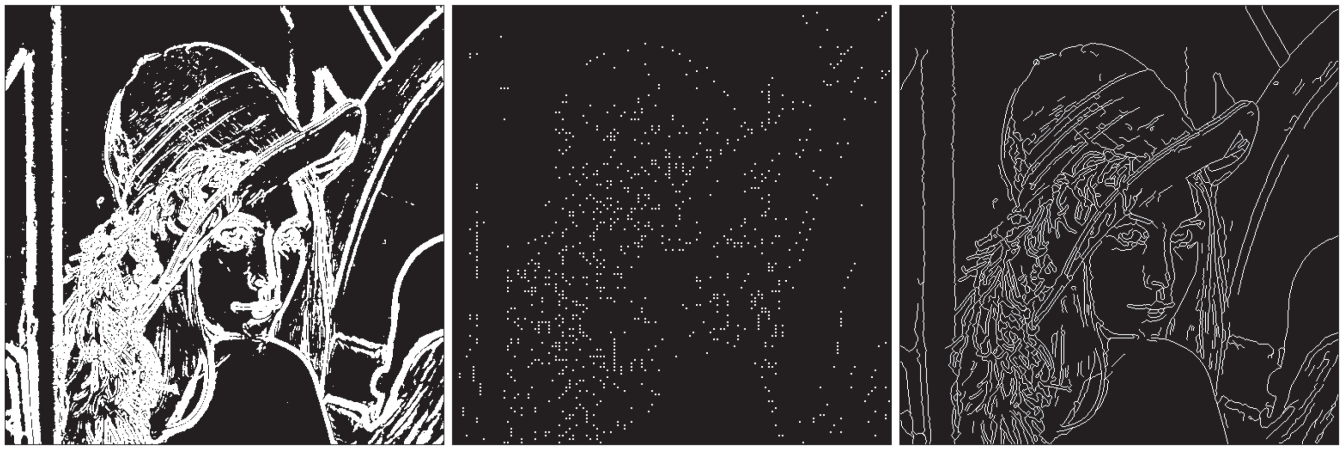

Fig. 3. Lena's (a) thresholded gradients cluster, (b) computed anchor points, (c) Lena's final edge map produced by ED using the anchors in (b).

possibly be computed in various ways. For instance, corners would be very good candidates for anchors since they are located on the image where the intensity sharply changes in both horizontal and vertical directions. This situation greatly increases the probability of an edge to lie on that location. An application that aims to detect the rigid geometric objects may be a good example to prefer corners for anchors. Alternatively, anchor extraction procedure may specifically be designed according to the application, i.e. an interest point detector or/with a keypoint descriptor can be substituted to detect specific edges on the image for the sake of performance. Thus, edges of unnecessary details can be filtered out regardless of how much sharp they are. In this paper, we simply prefer the use of local gradient extremums as anchor points for the following reasons, i.e. we already obtain the required information to compute local gradient extremums in previous steps, thus it eases the explanation of the main idea of our algorithm and becomes computationally beneficial. Note that an anchor is just a location that the *edge drawing* procedure begins, and it does not mean that every anchor will be included in the final edgemap. We will talk about this situation in the Section 2.4 in more detail.

If you visualize the gradient map in 3D, anchors would be located at the peaks of the gradient map mountains. To connect consecutive anchors, we would simply proceed over the cordillera peak of the gradient mountain. Fig. 2(b) and (c) shows part of the gradient map from Lena's hat, and the corresponding 3D view. Clearly, the gradient operator produces maximum values at edge crossings. That is, edgels are located on top of the gradient mountain. Anchors (marked with red circles) are the peaks of this mountain. ED would first identify these peaks and then connect them with a smart routing procedure. In Fig. 2(c), two arrows indicate the anchor proceeding directions.

Algorithm 1 shows our intuitive algorithm to test a pixel (x,y) for being an anchor. Since an anchor must be a local peak of the gradient map, we simply compare the pixel's gradient value with its neighbors. For a horizontal edge, the up and down neighbors are compared; for a vertical edge, left and right neighbors are compared. If the pixel's gradient value is bigger than both of its neighbors by a certain threshold, then the pixel is marked as an anchor. By changing the anchor threshold, the number of anchors can be adjusted. The lower the anchor threshold, the denser the anchors will be. The higher the anchor threshold, the sharper and sparser the anchors will be. Furthermore, anchor testing can be performed at different scan intervals, i.e., every row/column, every second row/column etc. Increasing the number of anchors would increase the details of the edge map, and vice versa. Different scan intervals can also be used for horizontal and vertical anchor computation depending on the application, thus undesired features can be excluded.

Anchor scan interval is independently adjustable, however, there is a vital point to consider about it. Despite the inherent robustness of the edge drawing algorithm, the overall edge accuracy somewhat depends on the selection and the sparsity of the anchor points. Although the scan interval for anchor point search is a free parameter, it must be kept below the radius of the smoothing kernel (which happens to be "5" for the experiments presented herein) to assure that no prominent edge feature is discarded. In this way, the subsampling (due to scan interval selection) operation is assured to be kept below the Nyquist rate and obtaining ambiguous results is avoided. A multi scale approach could also be adopted to improve the edge detail parameterization via scale-space approaches. However the current work is the introduction of the novel edge drawing paradigm, so the multi scale concept is kept beyond the scope of this work for an individual case study. Fig. 3(a) shows a set of anchors for the Lena image for an anchor threshold of 8, and a scan interval of 4; that is, every 4th row or column has been scanned to look for anchors.

2.4. Connecting the anchors by smart routing

The final and most crucial step of ED is connecting the anchors by drawing edges between them. Since the edge map is constructed in this step by using the information gathered in the first three steps of the algorithm, this step deserves special attention. To connect consecutive anchors, we simply go from one anchor to the next by proceeding over the cordillera peak of the gradient map mountain. This process is guided by the gradient magnitude and edge direction maps computed in step 2 of the algorithm. The smart routing process is independent from the anchor extraction method and works as follows: Starting at an anchor, we look at the direction of the edge passing through the anchor. If a horizontal edge passes through the anchor, we start the connecting process by proceeding to the left and to the right (refer to Fig. 4(b). If a vertical edge passes through the anchor, we start the connection process by proceeding up and down (refer to Fig. 4(c). During a move, only three immediate neighbors are considered, and the one having the maximum gradient value is picked. The process stops under two conditions:

(1) We move out of the edge areas, i.e., the thresholded gradient value of the current pixel is 0.
(2) We encounter a previously detected edgel.

Algorithm 1: Algorithm to test for an anchor at (x,y)

Symbols used in the algorithm:
(x,y): Pixel being processed
G: Gradient map
D: Direction map

```
IsAnchor(x, y, G, D, AnchorThresh)
G[x,y] ⇐ Not Anchor
if D[x,y] = HORIZONTAL then
  if G[x,y]−G[x,y − 1] ⩾ AnchorThresh and
    G[x,y]−G[x,y + 1] ⩾ AnchorThresh then
    G[x,y] ⇐ Anchor
  end if
else
  //Vertical Edge
  if G[x,y]−G[x − 1,y] ⩾ AnchorThresh and
    G[x,y]−G[x + 1,y] ⩾ AnchorThresh then
    G[x,y] ⇐ Anchor
  end if
end if
```

Algorithm 2 describes smart routing starting at an anchor (x,y). We assume that a horizontal edge passes through this anchor and we give the algorithm to proceed left of the anchor. Proceeding right, up or down would be very similar, and are not elaborated. Starting at an anchor (x,y), we proceed left for as long as the two conditions listed above are satisfied. At each iteration of the loop, we simply look at the three immediate neighbors to the left, i.e., $(x\text{-}1,y\text{-}1)$, $(x\text{-}1,y)$ and $(x\text{-}1,y\text{+}1)$, and pick the one having the greatest gradient value. If we were to proceed right, we would have considered three immediate neighbors to the right, i.e., $(x\text{+}1,y\text{-}1)$, $(x\text{+}1,y)$ and $(x\text{+}1,y\text{+}1)$.

A detailed illustration of the smart routing procedure is given in Fig. 4.a on a magnified region of a thresholded gradient cluster. Numbers inside the pixels are the gradient magnitudes; the black

[1] For interpretation of color in Figs. 4 and 5, the reader is referred to the web version of this article.

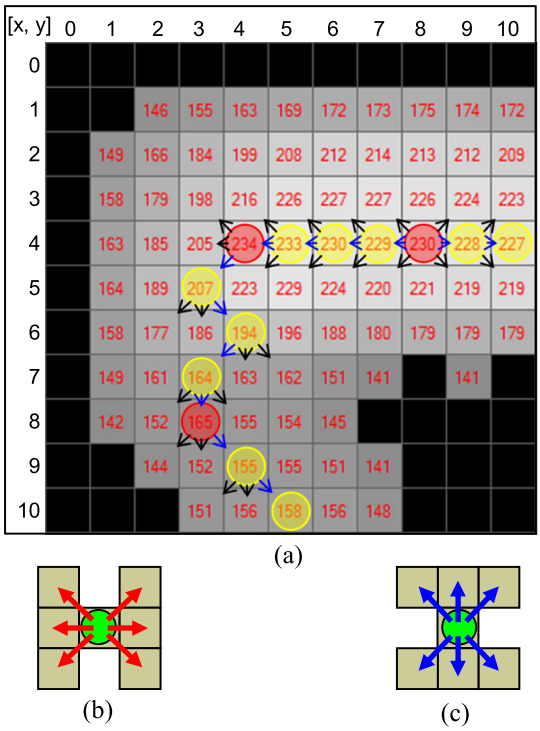

Fig. 4. (a) An illustration of the smart routing procedure, (b) Horizontal proceeding, (c) Vertical proceeding.

pixels represent the thresholded pixels. Anchors are marked with red[1] circles and the pixels collected during the connecting process are labeled with yellow circles. Assume that the pixel at (8,4) is the starting anchor. Since a horizontal edge is passing through this anchor, a horizontal walk (first to the `left` and then to the `right`) is initiated. Small arrows show the pixels being considered during the routing process, and the blue arrows indicate the selected pixels. As seen in Fig. 4(b) and (c), three immediate neighbors are checked at each iteration with respect to the edge direction, and the pixel having the greatest gradient value is selected. At (3,5) the direction of edge changes from horizontal to vertical. At this point, we stop the current horizontal `left` walk, and start two vertical walks, one going `up` and one going `down`. The vertical `up` walk terminates immediately, as it would pick the already selected edge segment pixel (4,4). The vertical `down` walk continues all the way to (5,10). Notice that there may be many different `walks` within the same edge segment. That is, during the extraction of a single edge segment, we may move horizontally and vertically many times as illustrated in Fig. 4, where we had one horizontal `left` move, and one vertical `down` move. Also notice that the resulting edge segment is always contiguous and one-pixel wide. Fig. 3(c) shows the Lena's final edge map obtained by joining the anchors shown in Fig. 3(b). Note that all anchors in Fig. 3(b) do not have to be included in the final edge map. It is possible to discard some of the edge segments if they consist of less than a specific number of anchors or edgels. This simple control enables to improve the final edge map by excluding short and small structures in an effortless manner.

3. Edge segment detection with ED

In addition to obtaining high quality edges of an image, there is a substantial demand for detecting the edges in edge segment format. An edge segment can be defined as a set of connected edgels in the edge map, and is very helpful in line and arc extraction applications [46–60]. An edge segment can also be chain-coded with relative position vectors between consecutive edgels, a.k.a., Freeman chain-code [45], and significant information about the geometric structure and orientation of the segment can be derived by analyzing the distribution and moments of the chain code [49–57]. The obtained feature vectors can then be used in detection [56,58], recognition [61], and registration [54,62] applications. Furthermore, obtained edge segments can be validated with the Helmholtz principle to eliminate invalid detections, which leads to a parameter-free edge segment detector based on the algorithm described in this paper [63].

If one wants to obtain the edge segments of an image with conventional tools, they would first obtain the binary edge map by a traditional edge detector and then apply a connected component analysis, which outputs each fully connected set of pixels as an edge segment. In [38] we compared the edge segment detection performances of ED and Canny followed by 8-pixel neighborhood connected component analysis (8N CCA). We performed the experiments with an extensive set of images and also present the computation timings for both of the algorithms. From the experiments we clearly show that the Canny's edge segments obtained by CCA have disjoint and jagged structures. Many segments, which should have been joined together, fail to get connected because Canny's edge map have many discontinuities. Notice also that Canny's edge segments are detected as a "blob of connected pixels" rather than as a "chain of pixels". This means that many "blob" segments should in fact be broken down to segments consisting of a linear chain of pixels. Furthermore, there are many small-sized noise-like edge segments due to the noisy nature of Canny's edge map. All of these drawbacks reduces the effectiveness of shape detection and recognition applications that make use of the extracted edge segments.

Algorithm 2: Algorithm to proceed left of an anchor at (x,y)

Symbols used in the algorithm:
(x,y): Pixel being processed
G: Gradient map
D: Direction map
E: Edge map

```
GoLeft(x, y, G, D, E)
while G[x,y] > 0 and E[x,y] ≠ EDGE and
      D[x,y] = HORIZONTAL do
  E[x,y] = EDGE; // Mark this pixel as an edgel
  //Look at 3 neighbors to the left &
  //pick the one with the max. gradient value
  if G[x − 1,y − 1] > G[x − 1,y] and
     G[x − 1,y − 1] > G[x − 1,y + 1] then
    x = x − 1; y = y − 1; // Up-Left
  else if G[x − 1,y + 1] > G[x − 1,y] and
          G[x − 1,y + 1] > G[x − 1,y − 1] then
    x = x − 1; y = y + 1; // Down-Left
  else
    x = x − 1; // Straight-Left
  end if
end while
```

The edge segments extracted by ED are is one-pixel wide, contiguous chain of pixels and also it is able to distinguish between adjoint edge segments and output them as separate linear pixel chains without the need for any further post-processing. This situation is more prominent especially around the complex details on the test images where Canny followed by CCA detects a whole local area as a blob of edges, but ED is able to detect this area as many distinct linear pixel chains (ref. to [38]).

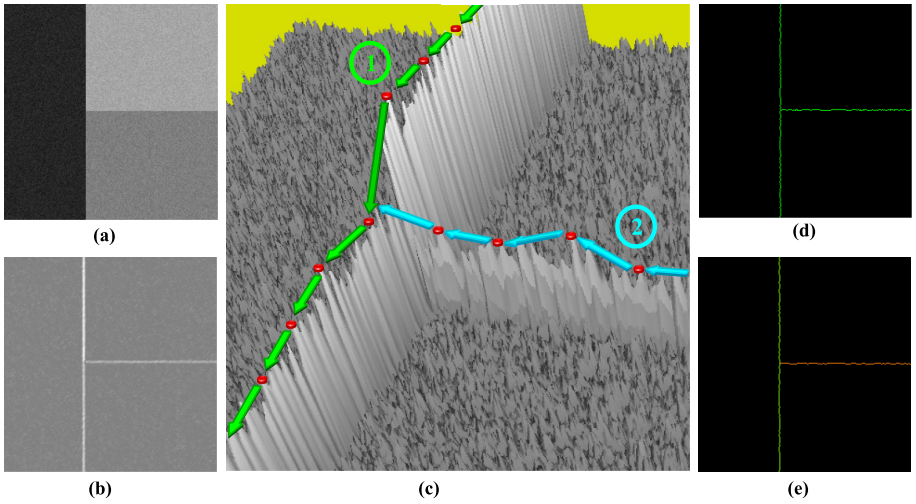

Fig. 5. Illustration of how ED is able to distinguish between different edge segments. (a) Synthetic noisy test image, (b) Gradient magnitude of image in a, (c) 3D illustration of ED algorithm, (d) Edge segment obtained with Canny+8N CCA, (e) Edge segments obtained with ED algorithm.

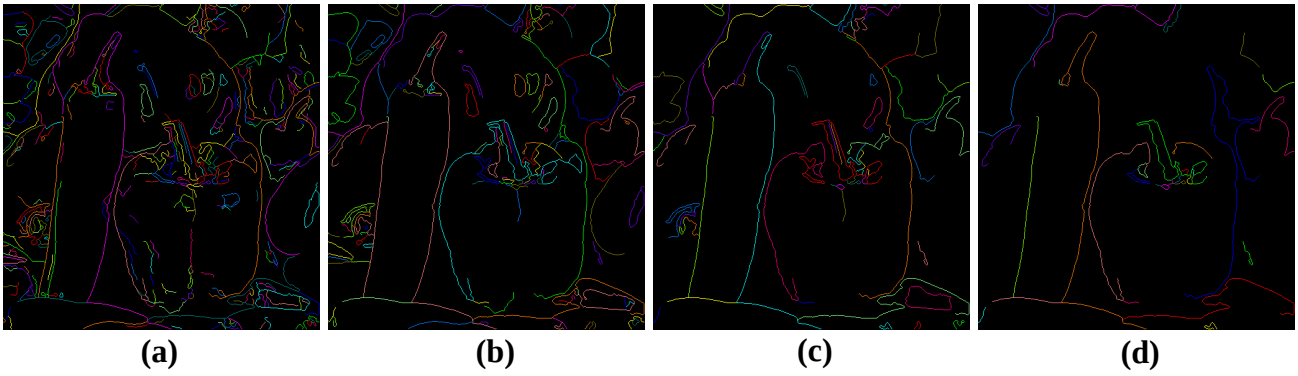

Fig. 6. ED's edge segment results on Peppers image. Parameters Gradient Threshold: 36, Scan Interval: 1, Anchor Threshold: (a) 0, (b) 20, (c) 40, (d) 60, respectively.

To better understand why ED is able to distinguish between adjoint edge segments and output them as separate linear pixel chains, consider the synthetic image given in Fig. 5(a), which contains a long vertical line and a short horizontal line intersecting the long line in the middle. A 10% uniform noise has been added on top of the image to make edge detection more difficult. Fig. 5(b) shows the image's gradient map (obtained by the Sobel operator), and Fig. 5(c) shows the 3D model of this gradient map. Red points on top of the cordillera peak of the gradient map are the anchors and arrows illustrate the anchor connecting process. Starting at the top most anchor on the long vertical line, ED proceeds down (green arrows) and goes all the way to the end of the vertical line. This linear chain of pixels is then output as the first edge segment. ED then starts at the right-most anchor on the short horizontal line and moves left (blue arrows) until it hits the long vertical line in the middle. This linear chain of pixels is then output as the second edge segment. Fig. 5(e) shows the two edge segments output by ED. If we perform the same experiment with Canny edge detection followed by CCA, we get a single "blob" edge segment containing the pixels of both lines as shown in Fig. 5(d).

To measure the effects of the anchor threshold on the final edge map, we performed another experiment and present the results in Fig. 6 for the Peppers image. To obtain the results in Fig. 6, we set all parameters of the algorithm to certain default values and only change the anchor threshold. Specifically, we increase the anchor threshold from left to right to indicate its effect on the resulting edge segments. As seen from the figure, with increasing anchor threshold, the minor details disappear and only the major edge segments remain. This is because at high anchor threshold values, anchor points are located only on the ridges where intensity changes are more rapid, i.e., strong edges. For this reason, edge segments of trivial details are filtered out and more prominent edges are detected.

In this paper, as a complementary study refer to the [38], we present an important extension of experiments about edge segment detection that clearly show and compare the performances Canny+8N CCA and ED in tough cases(Section 4.3). In these experiments, we show how to efficiently tune the parameters of ED in order to handle the situation in noisy and smooth cases on images.

4. Experiments

It is generally accepted that different edge detectors would be better suitable for different applications; however, Canny's edge detection algorithm has been shown to produce the best results in many situations [23,24], and is the most popular edge detection algorithm for many real-time computer vision and pattern recognition applications. Therefore, in this paper we will be comparing ED with Canny. Among various implementations of Canny's edge detector, we selected OpenCV implementation [36] for two

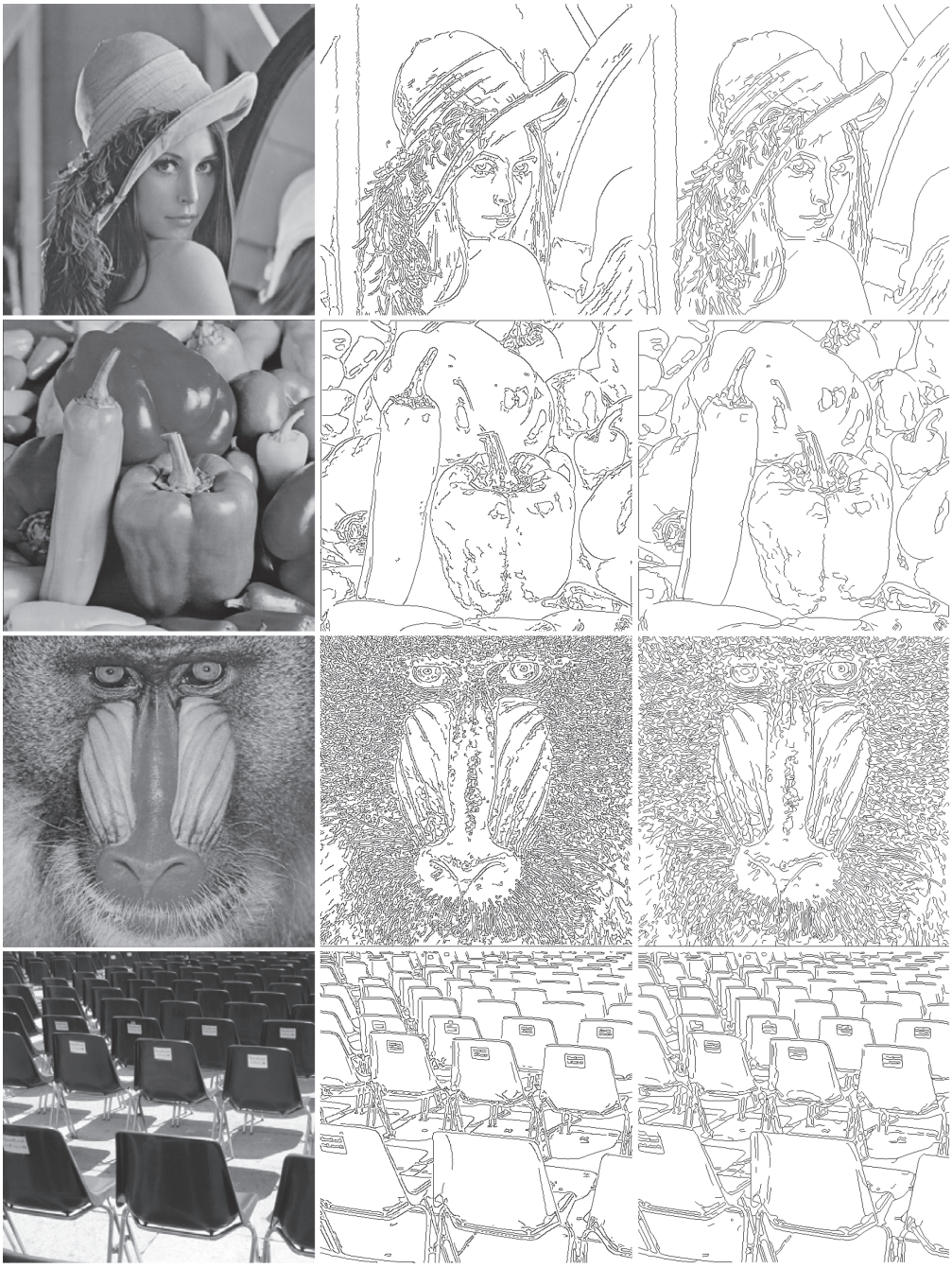

Fig. 7. (a) Original image (512 × 512), (b) OpenCV Canny edge map for low threshold = 30 and high threshold = 60, (c) ED edge map with Sobel operator, gradient threshold = 36 and anchor threshold = 8.

reasons: First, a standard implementation frees the reader from any suspicion about potential implementation discrepancies as anyone can download OpenCV freely. Second, and more importantly, OpenCV implementation of Canny is the only edge detector that can match ED in terms of speed and real-time applicability.

We analyze ED's performance in three steps. First, we compare the quality of ED's edge maps to that of Canny, and also compare their running time performance. We then look at the effects of different gradient operators on ED's final edge map. Next, the performance of ED on some tough cases are illustrated.

4.1. ED's edgemap quality and running time performance

In this section, we compare the performance of ED to OpenCV Canny (cvCanny). We first present edge map results on some test images, and then compare the running time performance of ED and OpenCV Canny on these images.

Fig. 7[2] shows the edge maps produced by OpenCV Canny (cvCanny) and ED for 4 challenging images. To obtain these results, we used a Canny low threshold of 30, and a high threshold of 60. For ED, we

[2] The reader may examine the all experimental images in fine detail by zooming since they are embedded in full resolution.

Table 1
Dissection of the running times of ED and OpenCV Canny on four images given in Fig. 7. The processing times were obtained in a PC with an Intel 2.2 GHz E4500 processor and 2 GB RAM

Image		Edge drawing (ms)					OpenCV Canny (ms)
		Gauss filtering	Gradient computation	Anchor extraction	Anchor linking	Total (ms)	
Lena	512 × 512	1.30	2.80	1.50	1.70	7.30	7.70
Peppers	512 × 512	1.30	2.74	1.35	1.47	6.86	7.53
Mandril	512 × 512	1.30	3.10	2.56	5.13	12.09	10.40
Chairs	512 × 512	1.30	2.67	1.41	1.83	7.21	7.53

used the Sobel operator with a gradient threshold of 36 and an anchor threshold of 8. In both cases, the input image was smoothed using a 5 × 5 Gaussian kernel with a $\sigma = 1$.

Looking at the resulting edge maps, we see that Canny's edge maps contain a lot of disjoint, unattended noisy edgels. On the other hand, ED's edge maps are very clean consisting of coherent, contiguous, well-localized and one-pixel wide edgels. The reader may perform additional tests online at ED's web site [39].

The asymptotic complexity of ED is simply $O(k^2n)$ where n is the total number of pixels in the image and the k is the aperture of the filter kernels. Since the algorithm uses predefined kernels for smoothing and derivation with constant apertures; the final runtime complexity of the ED reveals $O(n)$ as a linear algorithm. Table 1 shows the dissection of the processing times of ED and OpenCV Canny on four images given in Fig. 7. The processing times were obtained in a PC with an Intel 2.2 Ghz E4500 CPU and 2 GB RAM. Clearly, ED runs real-time and is faster on most of the test images up to 10% than the OpenCV's Canny. If you consider that the OpenCV algorithms are improved by the performance optimization experts, the timing results with our naïve ED implementation are quite appreciable. Also recall that ED not only generates better binary edge maps, but its output is a set of edge segments, each consisting of a pixel chain in vector form. To generate edge segments as pixel chains from OpenCV Canny's binary edge map

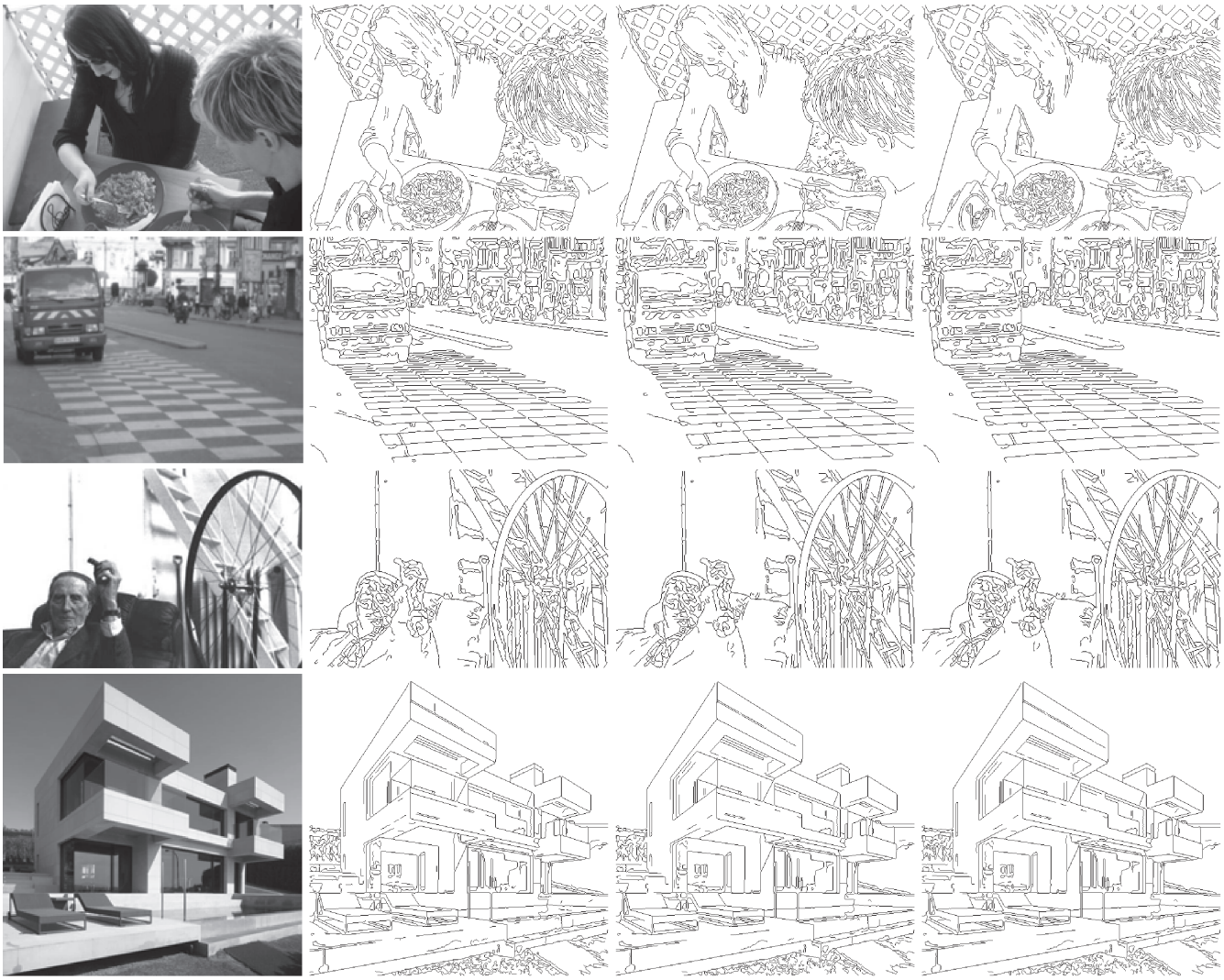

Fig. 8. (a) Original image. (b–d) ED's edge maps using Prewitt, Sobel, and Scharr gradient operators respectively. The following (Gradient threshold, Anchor threshold) pairs have been used with each operator: Prewitt (24,5), Sobel (36,8), and Scharr (144,32).

would require an additional linking (to improve the modal structure of edges) and a connected component analysis step (to detect the segments); which not only requires extra processing time, but it would also produce inferior edge segments due to the noisy and disjoint structure of the original edge map. ED, however, produces these segments in pixel chains form by default without the need for extra processing.

It is important to note that many recent studies have made use of the popular nVidia CUDA architecture [40] to speed up many image processing algorithms including edge detection [41–43]. To leverage the benefits of the CUDA architecture, we have recently implemented ED on CUDA [44], and observed that ED takes less than 0.5 ms for 640 × 480 images in a high-end CUDA card. This clearly shows that CUDA-based implementation of ED can run up to 2000 Hz.

4.2. The effects of the gradient operator on ED's edge segments

Recall from Section 2 that ED computes a gradient map, and then makes use of this gradient map during anchor computation and linking. We stated that ED is independent of the used gradient operator, and can work with a gradient map computed by any of the well-known gradient operators. In this section, our goal is to look at the effects of the gradient operator on ED's final edge map. Below we run ED with three well-known gradient operators, i.e. Prewitt, Sobel and Scharr, and compare the resulting edge maps.

Fig. 8 shows the edge maps produced by ED using Prewitt, Sobel and Scharr gradient operators for four different images. The following (Gradient threshold, Anchor threshold) pairs have been used with each operator: Prewitt (24,5), Sobel (36,8) and Scharr (144,32). Analysis of the edge maps in Fig. 8 (and many others we tested) indicates that there are very small differences between the edge maps produced by different gradient operators. Since the algorithm constructs the edge map with the pixels which give the maximum response to the gradient operator, choice of the operator does not make much of a difference on ED's final edge map. The reader may perform additional tests with different gradient operators online at ED's Web site [39].

4.3. ED on tough cases

In this section, we talk about how to set ED's parameters to get the best performance in some tough cases. Images containing a lot of noise or very smooth boundary regions can be defined as tough cases for an edge detector. In such cases, a typical edge detector either does not detect any edges at all or detects very low quality edges, which would not be very useful for further processing in the rest of the application. Fig. 9 shows a patch from the Lena image. Since this patch has a very smooth intensity change, detecting the left vertical line is very difficult. Fig. 9(b) and (d) show the thresholded gradient magnitude images for thresholds of 60 and 30 respectively, and Fig. 9(c) and (e) show the edge maps produced by OpenCV Canny. With a gradient threshold of 60, which corresponds to Canny low threshold, the resulting edge map quality is reasonable with a few number of discontinuities. But since the transition on the left vertical line is very smooth, it does not give a very high response to the gradient operator and gets eliminated by a slightly high threshold. Therefore, this part is not detected at all in Fig. 9(c). When the gradient threshold is lowered to 30 so that this area does not get eliminated during gradient thresholding (refer to Fig. 9(d), the left vertical line is detected (refer to Fig. 9(e). But observe that lowering the gradient threshold has also lowered the edge map quality in other parts of the image, making final edge map very noisy.

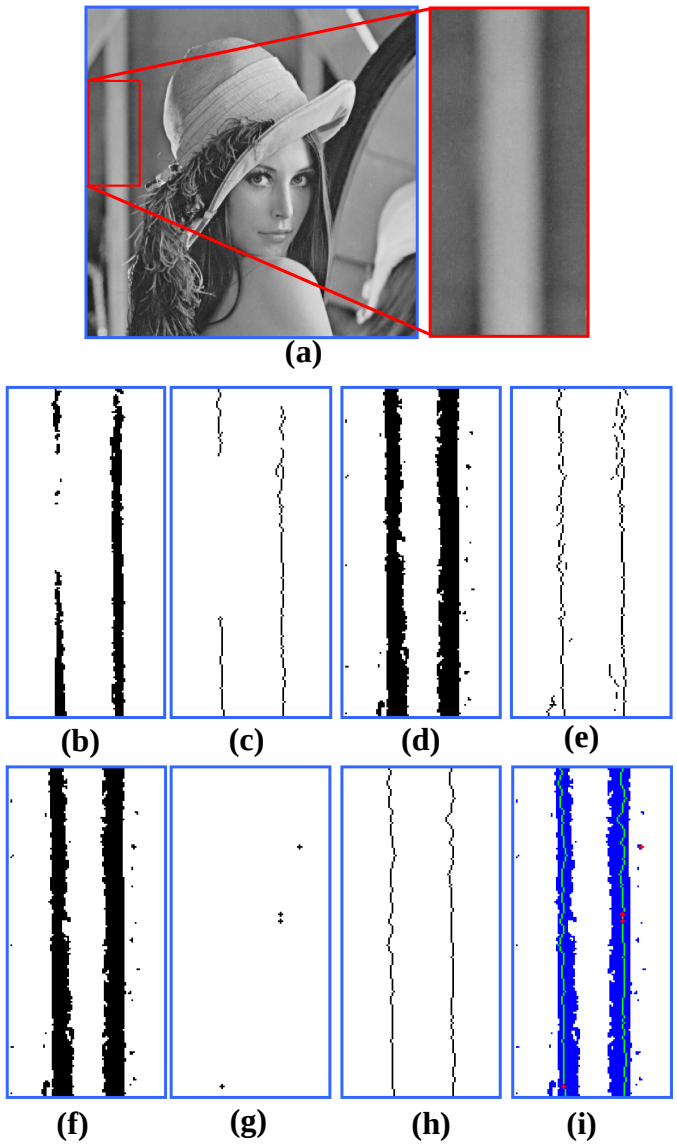

Fig. 9. (a) A patch from Lena, (b–c) Gradient cluster for a threshold of 60, and the resulting edge map by OpenCV Canny, (d–e) Gradient cluster for a threshold of 30, and the resulting edge map by OpenCV Canny. (f) Thresholded gradient magnitude image, (g) anchors, (h) EDs result with an anchor threshold of 10, (i) Edge segments and anchors overlayed on top of the thresholded gradient magnitude image.

ED can produce good results in such difficult cases when its parameters are set properly. Similar to Canny, the detected edges would be a subset of the thresholded gradient cluster, i.e., edge areas. But unlike Canny, which evaluate pixels in an individual and isolated manner, ED uses a high-level cognitive reasoning and works by joining anchors. Therefore, given a single stable anchor in an edge area, ED would be able to proceed within the edge area using the direction map and detect the entire boundary as a one-pixel wide edge segment. This means that as long as stable anchors are chosen within an edge area, lowering the gradient magnitude threshold would not result in a noisy edge map as it would in Canny. But the anchors must be chosen carefully, because a bad anchor may generate a totally meaningless edge segment.

Fig. 9(h) shows ED's result on the same vertical Lena patch. Gradient threshold was lowered enough to include both vertical strips as potential edge areas (refer to Fig. 9(f). With an anchor threshold of 10, several stable anchors are located within the edge areas (refer to Fig. 9(g). Recall from Section 2.3 that the anchor threshold

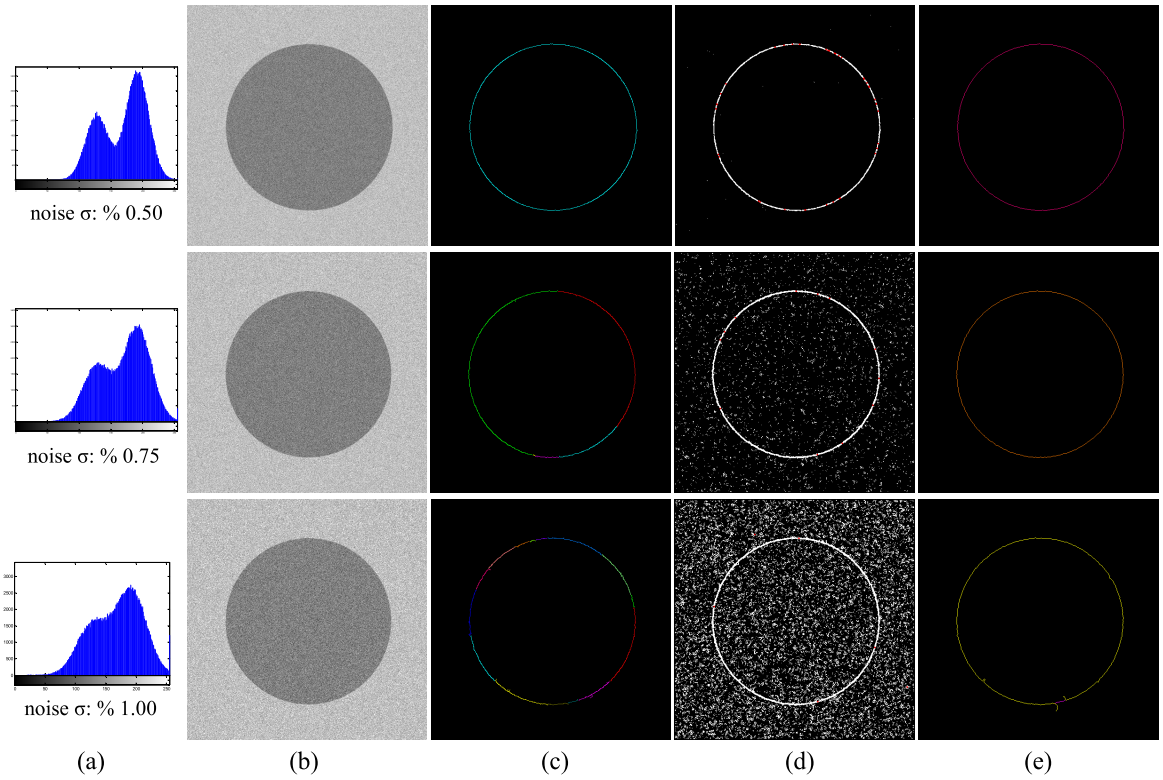

Fig. 10. (a) Histograms of the noisy synthetic images, (b) Synthetic images with different levels of additive Gaussian Noise. (c) Detected edge segments by Canny+8N CCA, (d) Thresholded gradient magnitudes with anchor points (red dots), (e) Detected edge segments by ED algorithm. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.).

determines how sharp local maximas of the gradient map will be picked as anchors. The higher the anchor threshold, the sharper or steeper the maxima must be in order to be picked as an anchor. Since we want more stable anchors in such smooth and noisy areas, a relatively high anchor threshold of 10 was chosen. This is why only a couple of anchors are detected. Fig. 9(i) shows the thresholded cluster, anchors and edge segments with an overlayed manner. Clearly, a single anchor on the left vertical strip (the one at the bottom of the strip) was enough to draw out the entire left vertical line. Similarly, two stable anchors (the ones in the middle) were enough to draw out the entire right vertical line. Note that the right-upmost anchor does not included in the final edge map since its short length in accordance with the explanation in Section 2.4.

To test performance of ED's segments with noisy images, we created the synthetic image shown in Fig. 10(b) and added different levels of Gaussian noise. Fig. 10(c) shows the edge segments produced by OpenCV Canny. Fig. 10(d) shows ED's anchors (red points), and Fig. 10(e) shows ED's edge maps. To get these results, different thresholds have been tried and the best resulting edge maps for both algorithms have been presented. In the first row of Fig. 10 both algorithms can be seen to detect the synthetic circle's periphery as one blob of connected edgels despite the added noise. As we increase the amount of noise in the second row, Canny starts detecting the circle's periphery as five separate blobs of connected edgels, i.e., edge segments (note that the each edge segment is indicated in a different color). ED, however, is still able to detect the circle's periphery as one edge segment. Eventually, in the last row with a lot of noise, Canny detects the circle's periphery as 15 separate edge segments since the edgels become more disconnected due to noise. As we decrease Canny's thresholds to make edgels more connected, the resulting edge map contains more notches and meaningless furcations. ED, however, continues producing a good result, and detects the circle's periphery in two separate edge segments. The reader should carefully examine the images in Fig. 10(d) to understand where the success of ED comes from. As seen, we can decrease the gradient threshold to obtain a contiguous circle periphery without compromising the edge quality because a few carefully chosen anchors (4 in the given example indicated as red dots) are able to draw out the entire circle boundary.

5. Conclusions and future work

We present a novel, unorthodox and real-time edge segment detector that is based on a high-level cognitive reasoning paradigm, and works similar to dot-to-dot boundary completion puzzles. The proposed algorithm computes a set of anchors, which are expected to be stable landmarks on the edges of a given image, and literally draws edges between these anchors by a heuristic smart routing algorithm, hence the name, Edge Drawing (ED). Since ED works by chaining adjoint edgels, the produced edge map is guaranteed to consist of clean, contiguous, well-localized, one-pixel wide edge segments, each of which is a linear pixel chain. By the high level reasoning that the algorithm based on, ED imitates humans' high-level cognitive reasoning and does not let any single or group of unattended edgels to appear on the final edge map. Experiments on many different images (and many more that the reader may perform online [39]) show that ED's edge maps are of very high quality. The algorithm is very generic, and can be run with different gradient operators and thresholds to adjust the level of detail in the final edge map. Even in tough noisy images or those containing smooth boundary transitions, ED is shown to produce good results with appropriate parameters. With its high quality edge segments and blazing speed, we believe that ED would be

very suitable for the next generation real-time computer vision and pattern recognition applications.

It is possible to list the following topics as future work candidates based on this study; examining the performance of ED with different anchor extraction methods and analyzing their stability, a multi-scale approach on the anchor selection step, object detection and recognition with the ED's edge segments and testing the efficiency of the ED algorithm with a robust visual assessment method on test images with ground truth edge results.

References

[1] J.M.S. Prewitt, Object enhancement and extraction, picture processing and psychopictorics, Academic Press, 1970. B. Lipkin, A. Lipkin, A.A. Rosenfeld (Eds.).

[2] I. Sobel, G. Feldman, A 3 × 3 Isotropic Gradient Operator for Image Processing, Pattern Classification and Scene Analysis, John Wiley and Sons, 1973. pp. 271–272.

[3] H. Scharr, Optimal Operators in Digital Image Processing, Ph.D Thesis, Heidelberg Univ., 2000.

[4] J. Canny, A computational approach to edge detection, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 8 (6) (1986) 679–698.

[5] F. Bergholm, Edge focusing, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 9 (6) (1987) 726.

[6] M. Gokmen, C.C. Li, Edge detection and surface reconstruction using refined regularization, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 15 (5) (1993) 492–498.

[7] P.H. Gregson, Using angular dispersion of gradient direction for detecting edge ribbons, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 15 (1993) 682–696.

[8] D. Ziou, S. Tabbone, A multiscale edge detector, Pattern Recognit. 26 (9) (1993) 1305–1314.

[9] K.R. Rao, J. Ben-Arie, Optimal edge detection using expansion matching and restoration, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 16 (12) (1994) 1169–1182.

[10] W.E. Higgins, C. Hsu, Edge detection using 2D local structure information, Pattern Recognit. 27 (2) (1994) 277–294.

[11] V. Srinivasan, Edge detection using neural networks, Pattern Recognit. 27 (12) (1994) 1653–1662.

[12] L.A. Iverson, S.W. Zucker, Logical/Linear operators for image curves, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 17 (10) (1995) 982–996.

[13] F. vander Heijden, Edge and line feature extraction based on covariance models, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 17 (1) (1995) 16–33.

[14] P.J. Tadrous, A simple and sensitive method for directed edge detection, Pattern Recognit. 28 (10) (1995) 1575–1586.

[15] C.A. Rothwell, J.L. Mundy, W. Hoffman, V.D. Nguyen, Driving Vision by Topology, in: Int. Symp. Comput. Vis. November 1995, Coral Gables, pp. 395–400.

[16] A. Desolneux, B. Georgescu, Edge detection with embedded confidence, J. Math. Imaging Vis. 14 (3) (2001).

[17] P. Meer, L. Moisan, J.M. Morel, Edge detection by Helmholtz principle, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 23 (12) (2001) 1351–1365.

[18] J.R. Fram, E.S. Deutsch, On the quantitative evaluation of edge detection schemes and their comparison with human performance, IEEE Trans. Comput. 24 (6) (1975) 616–628.

[19] D.J. Bryant, D.W. Bouldin, Evaluation of edge operators using relative and absolute grading, in: Int. Conf. Pattern Recognit. Image Process, Chicago, 1979, pp. 138–145.

[20] V. Ramesh, R.M. Haralick, Performance Characterization of Edge Detectors, Applications of Artificial Intelligence X: Machine Vision and Robotics, 1708, SPIE, 1992. pp. 252–266.

[21] R.N. Strickland, D.K. Cheng, Adaptable Edge Quality Metric, Optical Eng. 32 (5) (1993) 944–951.

[22] T. Kanungo, M.Y. Jaisimha, J. Palmer, R.M. Haralick, A methodology for quantitative performance evaluation of detection algorithms, IEEE Trans. Image Process. 4 (12) (1995) 1667–1674.

[23] M.D. Heath, S. Sarkar, T. Sanocki, K.W. Bowyer, A Robust visual method for assessing the relative performance of edge-detection algorithms, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 19 (12) (1997) 1338–1359.

[24] K. Bowyer, C. Kranenburg, S. Dougherty, Edge detector evaluation using empirical ROC curves, Comput. Vis. Image Understand. 84 (1) (2001).

[25] Shing-Min Liu, Wei-Chung Lin, Cheng-Chung Liang, An iterative edge linking algorithm with noise removal capability, in: Int. Conf. Pattern Recognit. (ICPR), 1988, pp. 1120–1122.

[26] M. Xie, Edge linking by using causal neighborhood window, Pattern Recognit. Lett. 13 (9) (1992) 647–656.

[27] F.R. Miller, J. Maeda, H. Kubo, Template based method of edge linking using a weighted decision, in: IEEE/RSJ Conf. on Intelligent Robots and Systems (IROS), 1993, pp. 1808–1815.

[28] J. Basak, B. Chanda, D.D. Majumder, On edge and line linking with connectionist models, IEEE Trans. Syst. Man Cybernet. 24 (3) (1994) 413–428.

[29] A. Hajjar, T. Chen, A VLSI architecture for real-time edge linking, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 21 (1) (1999) 89–94.

[30] Z. Wang, H. Zhang, Edge linking using geodesic distance and neighborhood information, in: IEEE/ASME Int. Conf. Advanced Intelligent Mechatronics (AIM), 2008, pp. 151–155.

[31] F.Y. Shih, S. Cheng, Adaptive mathematical morphology for edge linking, Inform. Sci. 167 (1–4) (2004) 9–21.

[32] Ya-Ping Wong, V. Chien-Ming Soh, Kar-Weng Ban, Yoon-Teck Bau, Improved canny edges using ant colony optimization, in: Int. Conf. Computer Graphics, Imaging and Visualisation (CGIV), 2008.

[33] A. Jevtic, I. Melgar, D. Andina, Ant based edge linking algorithm, in: IEEE Int. Conf. Industrial Electronics, 2009.

[34] J. Rahebi, Z. Elmi, A. Farzamnia, K. Shayan, Digital image edge detection using an ant colony optimization based on genetic algorithm, in: IEEE Int. Conf. Cybernetics and Intelligent Systems (CIS), 2010.

[35] Q. Lin, Y. Han, H. Hahn, Real-time lane departure detection based on extended edge-linking algorithm, in: Int. Conf. Comput. Res. Develop., 2010.

[36] OpenCV 2.1, 2011. <http://opencv.willowgarage.com>.

[37] C. Topal, C. Akinlar, Y. Genc, Edge drawing: a heuristic approach to robust real-time edge detection, in: Proc. Pattern Recognit. (ICPR) 2010, pp. 2424–2427.

[38] C. Topal, O. Ozsen, C. Akinlar, Real-time edge segment detection with edge drawing algorithm, in: Proc. of Image and Signal Processing and Analysis (ISPA), 2011, pp. 313–318.

[39] Edge Drawing Web Site, 2011. <http://ceng.anadolu.edu.tr/CV/EdgeDrawing>.

[40] NVIDIA, CUDA Programming Guide Version 1.0, 2007, NVIDIA Corporation, Santa Clara, California.

[41] Y.Luo, R. Duraiswami, Canny edge detection on NVIDIA CUDA, in: Computer Vision and Pattern Recognition Workshops, (CVPRW), 2008.

[42] A.L. Jackson, A Parallel Algorithm for Fast Edge Detection on the Graphics Processing Unit, An Honour Thesis, Washington and Lee University, 2009. Available at <http://www.cs.unc.edu/jacksona/doc/thesis.pdf>, Accessed on February 29, 2012.

[43] K. Ogawa, Y. Ito, K. Nakano, Efficient canny edge detection using a GPU, in: Int. Conf. on Networking and Computing (ICNC), 2010.

[44] O. Ozsen, C. Topal, C. Akinlar, Parallelizing edge drawing algorithm on CUDA, in: IEEE Int. Conf. on Emerging Signal Processing Applications (ESPA), 2012.

[45] H. Freeman, Computer processing of line drawing images, Comput. Surveys 6 (1974) 57–98.

[46] P. Hough, Method and means for recognizing complex patterns, US Patent 306954, 1962.

[47] L. Gupta, M.D. Srinath, Contour sequence moments for the classification of closed planar shapes, Pattern Recognit. 20 (3) (1987) 267–272.

[48] J. Illinworth, K. Kittler, A survey of the Hough transform, Comput. Vis. Graphics Image Process. (1988) 87–107.

[49] S. Marshall, Review of shape coding techniques, Image Vis. Comput. 7 (4) (1989) 281–294.

[50] L. Xu, E. Oja, P. Kultanen, A new curve detection method: Randomized Hough Transform (RHT), Pattern Recognit. Lett. 11 (5) (1990) 331–338.

[51] N. Kiryati, Y. Eldar, A.M. Bruchstein, Probabilistic Hough transform, Pattern Recognit. 24 (1991) 303–316.

[52] A. Etemadi, Robust segmentation of edge data, in: Int. Conf. on Image Processing and its Applications, 1992, pp. 311–314.

[53] J. Iivarinen, M. Peura, J. Sarela, A. Visa, Comparison of combined shape descriptors for irregular objects, in: Proc. of the Eight British Machine Vision Conference, UK, 1997.

[54] X. Dai, S. Khorram, A feature-based image registration algorithm using improved chain-code representation combined with invariant moments, IEEE Trans. Geosci. Remote Sensing 37 (5) (1999).

[55] W.J. Kang, X. Ding, J. Cui, Fast straight-line extraction algorithm based on improved Hough transform, Opto-Electron. Eng. (2007) 105–108.

[56] J.S. Stahl, S. Wang, Edge grouping combining boundary and region information, IEEE Trans. Image Process. 16 (10) (2007) 2590.

[57] S. Kiranyaz, M. Ferreira, M. Gabbouj, A generic shape/texture descriptor over multiscale edge field: 2-D walking ant histogram, IEEE Trans. Image Process. 17 (3) (2008) 377.

[58] V. Ferrari, L. Fevrier, F. Jurie, C. Schmid, Groups of adjacent contour segments for object detection, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 30 (1) (2008).

[59] C. Akinlar, C. Topal, EDLines: a real-time line segment detector with a false detection control, IEEE Pattern Recognit. Lett. 32 (13) (2011) 1633–1642.

[60] C. Akinlar, C. Topal, EDCircles: realtime circle detection by edge drawing (ED), in: Proc. Acoustics, Speech, and Signal Processing (ICASSP), 2012, pp. 1309-1312.

[61] Y. Gao, M.K.H. Leung, Face recognition using line edge map, IEEE Trans. Pattern Anal. Machine Intell. (PAMI) 24 (6) (2002) 764.

[62] M. Xia, B. Liu, Image registration by super-curves, IEEE Trans. Image Process. 13 (5) (2004) 720.

[63] 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).