Fast Detection of Curved Edges at Low SNR

Nati Ofir Meirav Galun Boaz Nadler Ronen Basri
Weizmann Institute of Science
Rehovot, 76100 Israel

Abstract

Detecting edges is a fundamental problem in computer vision with many applications, some involving very noisy images. While most edge detection methods are fast, they perform well only on relatively clean images. Unfortunately, sophisticated methods that are robust to high levels of noise are quite slow. In this paper, we develop a novel multiscale method to detect curved edges in noisy images. Even though our algorithm searches for edges over an exponentially large set of candidate curves, its runtime is nearly linear in the total number of image pixels. As we demonstrate experimentally, our algorithm is orders of magnitude faster than previous methods designed to deal with high noise levels. At the same time, it obtains comparable and often superior results to existing methods on a variety of challenging noisy images.

1. Introduction

This paper considers the problem of detecting faint edges in noisy images. Our key contribution is the introduction of a new, computationally efficient algorithm to detect faint curved edges of arbitrary shapes and lengths, under low signal-to-noise ratios (SNRs). A key feature of our algorithm is that the longer an edge is, the fainter it can be and still be detected. This is consistent with the performance of the human visual system, as illustrated in Fig. 1.

Edges with low signal-to-noise-ratio are common in a variety of imaging domains, specifically when images are captured under poor visibility conditions. Examples include biomedical, satellite, and high shutter speed imaging. As an illustrative example, Fig. 2 shows an image of Plankton acquired with an underwater imaging system [5]. Automated algorithms that detect the boundaries of different Plankton species, segment, and classify them are fundamental in the study of ocean ecosystems. However, segmenting Plankton in such images is challenging due to their low contrast and background noise.

Noise poses an obstacle to edge detection since it can reduce the contrast of actual edges, and introduce spurious

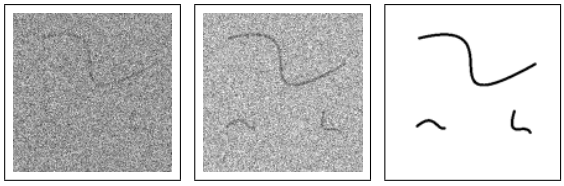


Figure 1. A noisy binary pattern with three curved fibers at high noise level (left, SNR=1) or low noise level (middle, SNR=2). The clean pattern is shown on the right. While the long fiber is noticeable already at the low SNR, the two shorter fibers can be spotted only at the higher SNR.

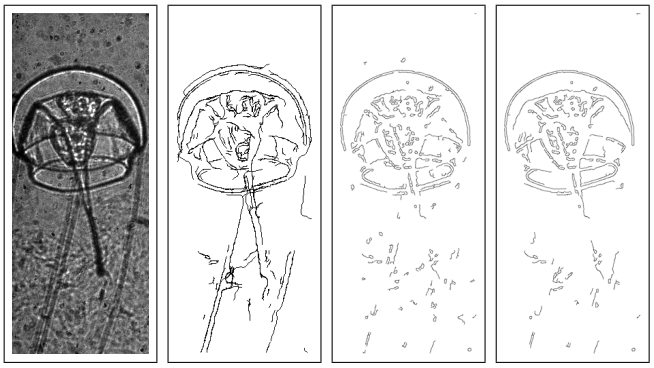


Figure 2. From left to right: Noisy image of a Plankton acquired by an underwater imaging system [5] containing several faint edges; our result; Canny result, and Canny after BM3D denoising.

high contrasts at background pixels. Thus, detecting edges using only local image gradients is highly sensitive to noise. To be more robust to noise, Canny [7], for example, first denoises the image with a Gaussian filter, before computing its gradients. While this smoothing operation indeed attenuates the noise, it also blurs existing edges, thus decreasing their contrast. Applying more sophisticated image denoising algorithms before edge detection does not, in general, solve the problem. We illustrate this point in Fig. 2. The edges found by Canny [7] on either the noisy image or the image denoised by the state-of-the-art BM3D algorithm [9] are shown in the two rightmost panels. Both methods fail to accurately detect Plankton's boundaries. In contrast, as shown in the middle-left panel, our proposed method is far

more robust to noise and yields a significantly more accurate result. We further illustrate this point in Sec. 6, using additional images from different application domains.

Our approach to detect faint curved edges is based on applying a large collection of *matched filters* to the image, and retaining only those which are statistically significant. By definition, each of these matched filters computes the average contrast of the image pixel intensities along its corresponding curve. When this curve traces an actual edge, the filter smoothes the noise on either side of the edge while maintaining its contrast. Hence, this operation does not suffer from the degradation of the general-purpose denoising approaches described above. The longer the edge and its corresponding matched filter, the more aggressively the noise is attenuated, allowing the detection of fainter edges.

The computational challenge in applying this approach is that the locations and shapes of the edges, and hence the corresponding matched filters, are a-priori unknown. Moreover, there are in general an exponential number of candidate curves and corresponding matched filters. Hence, a naive direct approach to compute all of them would be prohibitively slow. The main contribution of this paper is the development of a highly efficient algorithm to detect curved edges. Our detection scheme is multiscale and utilizes a hierarchical binary partition of the image, constructing matched filters for the detected edges in a bottom-up strategy. With this construction, even though our method searches for edges over a huge set of candidate curves, its runtime is *nearly linear* in the total number of image pixels. As we analyze in Sec. 4, our algorithm is orders of magnitude faster than other edge detection methods that can deal with high noise levels (e.g., [1]). Yet, as we demonstrate in Sec. 6, it obtains comparable, if not better, edge detection quality on challenging noisy images from a variety of applications.

2. Previous and Related Work

Edge detection is a well-studied problem with a rich history. Traditional methods considered step edges and relied on local gradients to detect them [19, 16, 7]. In contrast, recent methods addressed the problem of boundary detection in natural images [17, 3, 2, 4, 10, 15]. These methods rely on supervised learning of complex boundary features that account for intensity, color and texture. Despite the high accuracy achieved by these methods on natural images, as shown experimentally in Sec. 6, they exhibit poor performance on images with faint edges.

In this paper, we focus on the problem of detecting step edges at high levels of noise. As mentioned in the introduction, this is an important task in a variety of imaging domains. One of the first proposed methods for step edge detection was the Sobel operator [19], which does so by thresholding the gradient magnitude at each pixel separately. Marr & Hildreth [16] proposed to detect edges by identifying the zero crossings of a 2D Laplacian of a Gaussian filter applied to the image. These local approaches for edge detection are very efficient, essentially with linear-time complexity in the total number of pixels. However, due to their local nature, they are sensitive to noise and exhibit poor performance at low SNR. One exception is the algorithm suggested by Canny [7], which extends Sobel by hysteresis thresholding of the local gradient magnitudes. This post-processing operation significantly improves its robustness to noise. We thus choose the Canny algorithm as one of the baselines for comparison to our work.

A potentially promising approach to detect edges in noisy images is to first denoise the image and then apply some edge detection algorithm. The problem of image denoising received considerable attention in recent years, and various methods were proposed, including bilateral filtering [20], anisotropic diffusion [18], non-local means [6], BM3D [9] and more. These methods, however, are not optimized for edge detection and typically denoise the image based on small local patches. Hence, they may blur faint edges, making their detection even more difficult.

Another set of edge detection algorithms utilize a wavelet-based bank of filters that vary in length and width, see a review in [21]. Given the availability of fast wavelet transforms, these methods are also quite fast. However, a key difference from our approach is that these wavelet methods do not adapt to the shape of actual curved edges. Hence, their performance at low SNRs is sub-optimal.

Several recent studies proposed to use matched filters to improve the detection of faint edges. The method of Galun et al. [12] detects *straight* edges using $O(N \log N)$ operations for an input image with N pixels. A significantly faster method to detect long straight edges, with sub-linear run-time in the image size, was proposed in [14]. An algorithm for detecting faint *curved* edges, based on the Beamlet data structure [11] and its corresponding quad-tree partition of the input image. was proposed by Alpert et al. [1]. While their method is also multiscale, it suffers from a high complexity of $O(N^{2.5})$ operations, which translates into a non-practical runtime of several minutes on typical images.

In our work, we detect curved edges with a significantly reduced complexity. The two key ideas are to instead construct a *binary* tree partition, and perform sophisticated processing on it. We present two variants of our algorithm, which both require memory of $O(N \log N)$. The first, a more stringent variant has time complexity $O(N^{1.5})$, whereas the second faster one incurs a slight loss of detection accuracy, but has even faster runtime at $O(N \log N)$ operations. For illustrative purposes, the run time of our latter algorithm, implemented in C++, is roughly 1 second on a small 129×129 image and 5 seconds on a 257×257 image. In contrast, the runtime of [1], as reported in their

paper, is several minutes per image.

3. The Beam-Curve Binary Tree

In this paper, we develop edge detection algorithms that efficiently examine a huge set of possible candidate curves. This section describes in detail how we achieve this goal.

3.1. Setup and Notations

The input to our method is a noisy gray-level image I with $N = m \times n$ pixels containing an a-priori unknown number of curved edges at unknown locations and shapes. In developing the algorithm and its theoretical analysis, we assume for simplicity square images with $m = n$ where n is a power of 2, and an additive Gaussian noise model. Namely, the input image can be decomposed as $I = I_{clean} + I_{noise}$ where I_{clean} is a noise-free image, with step edges of constant contrast, and $I_{noise}(x, y) \overset{i.i.d.}{\sim} N(0, \sigma^2)$.

An *edge* is a non self-intersecting curve γ, with step discontinuity in the pixel intensities of the unobserved I_{clean}. Its SNR is defined as the absolute difference in image intensities across the step edge, divided by the noise level. To each candidate curve γ of total length L, passing through a set of pixels P, we associate the following response vector

$$\phi(\gamma) = [R, L, C, P]. \tag{1}$$

The response value R, determined by the matched filter corresponding to the curve γ, is the difference between the sums of intensities on the two sides of the curve. The variable $C = R/m(L)$ is the average contrast along the curve, where $m(L)$ is the total number of pixels of the matched filter of length L. Let γ_1 and γ_2 be two curves with a common endpoint, and with corresponding vectors $\phi(\gamma_i) = [R_i, L_i, C_i, P_i]$, $i = 1, 2$. The response vector of their *concatenation* is

$$\phi(\gamma_1 + \gamma_2) = [R_1 + R_2, L_1 + L_2, \frac{R_1 + R_2}{m(L_1) + m(L_2)}, P_1 \cup P_2]. \tag{2}$$

To detect statistically significant curved edges in the image I, we first construct its *beam-curve binary tree* and its corresponding data structure denoted BC. Geometrically, each node of the tree corresponds to a tile in the image (a region of a prescribed shape). A tile V of area A, is split into two sub-tiles V_1, V_2 of roughly equal area by an *interface* (a straight line in our implementation) whose length is proportional to $\sqrt{A}$. The root node, at level $j = 0$ represents the full image, whereas at each finer level j there are 2^j tiles. The tree is constructed bottom-up, starting from a fine level j_b, whose tiles all have an area approximately $n_{\min} \times n_{\min}$, where $n_{\min}$ (typically a few pixels) is a user-chosen parameter.

Within each tile V, for every pair of pixels p_1 and p_2 on its boundary ∂V, the beam curve data structure BC stores a response vector $\phi(\gamma)$. Postponing exact definitions to Sec. 5, this response vector corresponds to a single curve γ between p_1 and p_2, which is most likely to be an edge. See Alg. 1 for a pseudo-code of the tree construction, and Fig. 3 for its implementation based on a rectangular tile partition.

3.2. The Edge Detection Algorithm

As described in Alg. 1, the matched filter responses of the various curves are calculated in a bottom-up fashion, from the leaves of the tree to its root. For each leaf tile V at the bottom level j_b, we set the response of each pair of boundary pixels $p_1, p_2 \in \partial V$ to be the matched filter response of the straight line segment that connects them. See Alg. 2 for a pseudo-code of the bottom level processing.

Next, given the response vectors at level $j + 1$ we compute responses at the next coarser level j. We do this by concatenating curves from sibling tiles as follows. Let V_1, V_2 be sibling tiles at level $j + 1$, with a parent tile V at level j. We consider all pairs of pixels such that $p_1 \in \partial V_1 \cap \partial V$ and $p_2 \in \partial V_2 \cap \partial V$. Now, for each such pair of pixels (p_1, p_2), in the stringent variant of our algorithm, we consider all pixels in the joint interface $V_{12} = \partial V_1 \cap \partial V_2$. For each pixel $p_3 \in V_{12}$ we compute the response of the curve obtained by concatenating the two curves from p_1 to p_3 and p_3 to p_2. Among all of these concatenated curves, we store the one with highest response score (defined explicitly in Eq. (12) below). See Alg. 3 for pseudo-code of the coarser level construction.

The final output of our algorithm is a soft edge map image E. A value $E_{ij} = 0$ means that there is no detected edge passing through pixel (i, j), whereas the higher the value of E_{ij}, the stronger is the statistical evidence that an edge passes there. This edge map E is constructed as follows: We initially set all pixels of E to zero. For every response vector $\phi(\gamma)$, as detailed in Sec. 5, we assign a significance score that depends on its mean contrast C and its length L. A positive score indicates that the curve marks a statistically significant edge. Then, we sort all the curves in the tree whose score is positive from the highest score to lowest. For each curve γ in this sorted list, for each pixel $p \in P$ we set $E(p)$ to hold the score of γ. To deal with overlapping curves with positive scores, we apply the following simple non-maximal suppression procedure: First, if at some pixel p, the current value $E(p)$ is already positive, we do not decrease it; and, secondly, if most of the pixels in P were already marked by previous curves (of higher scores), we discard the current curve γ and do not add its pixels to E.

In principle, the generic algorithm above can be implemented with any binary partition of the image. In Sec. 4 we make a detailed complexity analysis, under the assumption that the children tiles are roughly half the area of their parent tile, and the length of their interface is roughly the square

root of their area. We prove that under these conditions, the algorithm complexity is $O(N^{1.5})$.

Algorithm 1 $BeamCurveTree(V)$

Require: Tile V whose maximal side length is n.
  if $n \leq n_{\min}$ **then**
    $BC \leftarrow BottomLevel(V)$
  else
    $V_1, V_2 \leftarrow SubTiles(V)$ { The tile is split into two sub-tiles of equal area}
    $BC_1 \leftarrow BeamCurveTree(V_1)$
    $BC_2 \leftarrow BeamCurveTree(V_2)$
    $BC \leftarrow CoarserLevel(V, V_1, V_2, BC_1, BC_2)$
  end if
  return BC

Algorithm 2 $BottomLevel(V)$

Require: Small tile V.
  $BC \leftarrow EmptySet$
  for $\forall p_1, p_2 \in \partial V$ **do**
    $\gamma \leftarrow$ straight line from p_1 to p_2
    $BC.add(\phi(\gamma))$
  end for
  return BC

Algorithm 3 $CoarserLevel(V, V_1, V_2, BC_1, BC_2)$

Require: V is an image tile, V_1 and V_2 are its sub-tiles.
Require: BC_1 is a set of the responses of sub-tile V_1.
Require: BC_2 is a set of the responses of sub-tile V_2.
  $BC \leftarrow BC_1 \cup BC_2$
  if BasicMode **then**
    $InterfaceSet \leftarrow \partial V_1 \cap \partial V_2$
  else if OptimizedMode **then**
    $InterfaceSet \leftarrow BestPixels(\partial V_1 \cap \partial V_2)$
  end if
  for $\forall p_1, p_2 : p_1 \in \partial V \cap \partial V_1, p_2 \in \partial V \cap \partial V_2$ **do**
    $AllResponses \leftarrow EmptySet$
    for $\forall p_3 \in InterfaceSet$ **do**
      $\gamma_1 \leftarrow$ curve from p_1 to p_3 in set BC_1
      $\gamma_2 \leftarrow$ curve from p_3 to p_2 in set BC_2
      $\phi(\gamma) \leftarrow concatenate(\phi(\gamma_1), \phi(\gamma_2))$
      $AllResponses.add(\phi(\gamma))$
    end for
    $BC.add(AllResponses.bestResponse())$
  end for
  return BC

Rectangular partition. In this partition, we split the square image into two rectangles of equal size. Next, split each rectangle into two squares and continue recursively until squares of size $n_{\min} \times n_{\min}$ are obtained. Specifically, in our implementation, we used $n_{\min} = 5$. We call the obtained data structure *Rectangle-Partition-Tree* (RPT). See Fig. 3 for an illustration of the RPT levels.

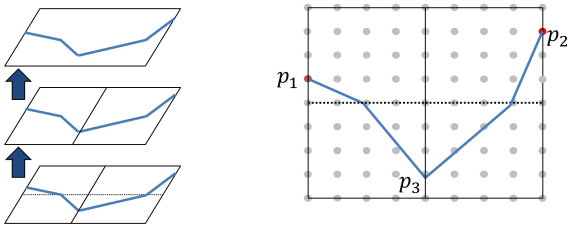


Figure 3. Left: The 3 topmost levels of a RPT. Right: The $n \times n$ image at level $j = 0$ is partitioned into two rectangles of size $n \times n/2$ at level $j = 1$. Each rectangle is then partitioned into two $n/2 \times n/2$ squares at level $j = 2$. A curve connecting two boundary pixels p_1, p_2 of level $j = 0$ is a concatenation of up to 2 curves of level $j = 1$, and up to 4 curves of level $j = 2$.

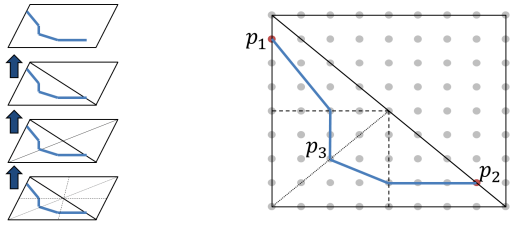


Figure 4. Triangle-Partition-Tree, an alternative implementation of the beam-curve binary-tree. Left: The 4 topmost levels of the TPT. Right: The partitioning embedded in the 2D image grid, every triangle is divided into two sub-triangles.

Triangular Partition. A second possible beam curve binary tree, which we denote as the *Triangle-Partition-Tree* (TPT), is based on triangular tiles, see Fig. 4. Here, at the topmost level, we split the image along its diagonal into two triangles. Then, at any subsequent level, we split each triangle into two sub-triangles recursively. For a square image, the triangles obtained are all right-angled isosceles. It can be shown that the TPT-based edge detection algorithm is slightly faster than the RPT-based one. Specifically, for a square image of N pixels, the TPT algorithm requires $\approx 14N^{1.5}$ operations (derivation omitted due page limit restrictions). In contrast, as we analyze in detail in Sec. 4 below, the RPT construction requires $\approx 18N^{1.5}$ operations. In addition, the set of potential curves scanned by both partitions is of comparable size. Hence, both partitions yield a similar edge detection performance.

3.3. An Optimized Version

A run-time complexity of $O(N^{1.5})$ operations may still be too slow when processing large images. Hence, it is of interest to develop even faster algorithms. Below we introduce an efficient variant of our algorithm with an $O(N \log N)$ complexity, at a statistical price of slightly

worse detection quality.

Let V be a tile at level j with children tiles V_1, V_2 and let $p_1, p_2 \in \partial V$. In this variant, the algorithm still seeks to compute the curve with the best response between p_1, p_2. However, instead of scanning all pixels in the joint interface $V_{12} = \partial V_1 \cap \partial V_2$, it only considers a subset of k pixels for some fixed constant k. To select this subset, for each pixel $p_3 \in V_{12}$ we look at the curve with highest response, that starts at either ∂V_1 or ∂V_2 and ends at p_3, as previously computed at level $j+1$. We then keep only the k pixels with the highest responses. As we prove in Sec. 4, the overall number of operations of this variant is significantly smaller. Our experiments in Sec. 6 indicate that in practice this results in a negligible decrease in edge detection quality.

4. Complexity Analysis

In this section, we study the computational complexity of the beam-curve algorithm of Sec. 3.2. We begin with an analysis of the general scheme. Denote by $t(A)$ the number of operations performed by our algorithm on a tile V of area A. The step of populating the responses of V involves computing responses for every triplet of pixels, $p_1 \in \partial V_1$, $p_2 \in \partial V_2$ and $p_3 \in \partial V_1 \cap \partial V_2$ where V_1 and V_2 the children tiles of V. The length of each of these three boundaries is $O(\sqrt{A})$, and so the complexity of this step is proportional to $A^{1.5}$. The operation is then repeated for the sub-tiles V_1 and V_2 whose areas are $\approx A/2$. Therefore, $t(A)$ satisfies the following recursion,

$$t(A) = 2t(A/2) + O(A^{1.5}). \tag{3}$$

The total complexity $t(A)$ can be determined using the master theorem [8], which considers recursions of the form

$$t(n) = at(n/b) + f(n). \tag{4}$$

The asymptotic behavior of $t(n)$ is obtained by comparing $f(n)$ to $n^{\log_b a \pm \epsilon}$. Specifically, if $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$ then $t(n) = \Theta(n^{\log_b a})$, while if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$ and in addition if $af(n/b) \leq cf(n)$ for some constant $c < 1$ then $t(n) = \Theta(f(n))$. In (3), $a = b = 2$ and $A^{1.5} = \Omega(A^{\log_2 2 + 0.5})$. In addition, $2f(A/2) = (1/\sqrt{2})f(A)$, implying that $t(A) = \Theta(f(A)) = \Theta(A^{1.5})$. Finally, as the area of the root tile equals the total number of image pixels N, the overall complexity of the beam-curve binary tree algorithm is $O(N^{1.5})$ operations.

RPT Space and Time Complexity. Next, for the case of a rectangular partition we now bound the multiplicative constant hidden in the above O notation, and compute its required memory. Recall that in the RPT, the root level $j = 0$ represents the original $n \times n$ image, whereas the next level, $j = 1$, contains two equal rectangles of size $n \times n/2$. In general, every even level j includes squares of size $n/2^{j/2} \times n/2^{j/2}$ pixels, while every odd level j includes rectangles of size $n/2^{(j-1)/2} \times n/2^{(j+1)/2}$.

We next derive the number of stored responses per level. At every tile of even level j, we store the edge responses of curves that pass from one side of a tile to a different side. The number of stored responses thus equals the product of 4 choose 2 (pairs of rectangle sides) times the tile area,

$$\binom{4}{2} \cdot \frac{n}{2^{\frac{j}{2}}} \cdot \frac{n}{2^{\frac{j}{2}}} = \frac{6N}{2^j}. \tag{5}$$

Consequently, multiplying Eq. (5) by the number of tiles, gives that the total number of stored curves at level j is

$$\frac{6N}{2^j} \cdot 2^j = 6N. \tag{6}$$

At odd levels, it can be shown that the above equations hold with a constant 6.5 instead of 6. Finally, as the number of levels is bounded by $\log N$ and the storage per curve is constant, the total required space is $O(N \log N)$.

Stringent Algorithm Complexity. First, recall that at the bottom level, for every leaf we scan all the straight lines that begin on one side of the tile and end on another side. Since the number of operations required to compute each such response is bounded by a constant, the total number of operations at this level is $O(N)$.

Next, at every coarser level j, we build each curve by concatenating two shorter curves of level $j+1$, that share a common endpoint at the joint interface. The work done at level j is thus bounded by the number of calculated curves times the length of the joint interface. Since the former is $6N$, the overall number of operations in an even level is

$$6N \times n/2^{j/2} = 6N^{1.5}/2^{j/2}, \tag{7}$$

while the number of operations in an odd level is

$$6N \times n/2^{(j+1)/2} = 6N^{1.5}/2^{(j+1)/2}. \tag{8}$$

Denote by $C(N)$ the time complexity of the algorithm that considers all the possible points on the interface. By summing these over all scales we get the following,

$$C(N) \approx 6N^{1.5} \left[\sum_{j \text{ even}}^{j_b} 2^{-j/2} + \sum_{j \text{ odd}}^{j_b} 2^{-(j+1)/2} \right]. \tag{9}$$

Consequently,

$$C(N) \leq 6N^{1.5} \left[\sum_{l=0}^{\infty} 2^{-l} + \sum_{l=1}^{\infty} 2^{-l} \right] = 18N^{1.5}. \tag{10}$$

Optimized Algorithm Complexity. As described in Sec. 3.3, the optimized variant scans, for each of the $6N$ pairs of start and endpoints, only the best k pixels in the

joint interface. Overall, it goes over $6Nk$ curves at each level j. Additional work is required to select the best k pixels. Since the number of tiles equals 2^j and the interface length is $\approx n/2^{j/2}$, the number of operations of this step is bounded by $\log k \cdot 2^j \cdot n/2^{j/2} < N$ for every level j. To conclude, the total number of operations at each level is no more than $(6k+1)N$. Thus, the overall complexity is bounded by $(6k+1)N \log N$, which leads to a significantly faster runtime.

5. Detection Threshold and Search Space Size

As described in Sec. 3, our method scans a huge set of candidate curves. For suitable pairs of start and end pixels, the algorithm stores in the data structure BC, the curved edge with the highest edge score, as defined by Eq. (12) below. Clearly, for most images, the vast majority of these responses do not trace actual image edges and should be discarded. This task requires the determination of a suitable threshold, possibly dependent on edge length, such that only edge responses above it are retained.

Previous work [1] introduced an approximate theoretical formula for this threshold, designed to control the average number of false-positive detections. The derivation of the threshold is related to the a-contrario approach, see, e.g., [13]. To this end, consider a pure noise image $I = I_{noise}$, where $I_{noise}(x,y) \sim N(0,\sigma^2)$. By definition, this image contains no real edges, and so, with high probability, all of its edge responses should be discarded. Suppose that there are K_L distinct candidate edges of length L. Then, the corresponding threshold, $T(L,K_L)$, is approximately the maximal edge contrast expected in I_{noise} among K_L statistically independent curves of length L. Alpert et al. [1] showed that

$$T(L,K_L) \approx \sigma\sqrt{\frac{2\ln K_L}{wL}}, \tag{11}$$

where w is the width of the matched filter. Thus, to each curve of length L, we assign an *edge score* φ, defined as the difference between its mean contrast and the threshold,

$$\varphi(\gamma) = C(\gamma) - T(L,K_L). \tag{12}$$

A positive score indicates that the candidate curve traces an edge. Moreover, higher scores represent stronger confidence for this indication.

To compute the edge score function of Eq. (12), we thus need to know, for each curve length L the size of the corresponding search space size, K_L. As shown below in Eq. (20), for the RPT construction, this quantity is approximately given by $K_L \approx 6N \times 2^{\beta L}$ for a suitable constant β. Inserting this expression into Eq. (11) gives

$$T(L) = \sigma\sqrt{\frac{2\ln(6N\times 2^{\beta L})}{wL}}. \tag{13}$$

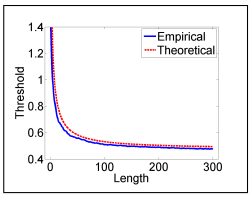


Figure 5. Contrast threshold as a function of curve length. The dashed theoretical curve, based on (13), is designed to have few false positive detections. The empirical threshold is produced by applying our algorithm on over 100 pure noise images, and storing the median of the maximal response for each curve length.

Minimal Detectable Contrast. An interesting question, already raised by [1], is how faint can an edge be and still be detected. In our case, note that as N and L tend to infinity, the threshold in Eq. (13) converges to a finite limit,

$$T_\infty = \Omega(\frac{\sigma}{\sqrt{w}}). \tag{14}$$

Namely, due to the exponential size of the search space, our threshold is bounded from below by a positive constant. Hence, our ability to detect faint edges of unknown shape and location in low SNR is limited. Fig. 5 compares the theoretical threshold of Eq. (13) to empirical results. The latter was computed by running our algorithm on over 100 pure noise images and storing for each curve length the median of the 100 maximal responses obtained. It can be seen that both curves are close to each other and that the graphs converge to $\approx 1/2$. This value is the asymptotic bound T_∞ for the selected parameters in this simulation: width $w=4$, image size $N=129^2$, noise level $\sigma=1$ and $\beta=0.65$.

5.1. Search Space Size

We now compute the size of the search space K_L of candidate curves of length L in the RPT. This quantity directly affects the contrast of threshold (11).

We first calculate the search space size at level j of the RPT, and then show its connection to K_L. Denote by $S(j)$ the upper bound of the total number of candidate curves at level j, and denote by $S'(j)$ the same number, but for given fixed start and end points. Since by Eq. (6), the total number of stored curves at any level is approximately $6N$, then

$$S(j) = 6N \times S'(j). \tag{15}$$

Next, to compute $S'(j)$, recall that in the RPT, we split a tile V of level j into two sub-tiles V_1, V_2 of level $j+1$, by a joint interface $\partial V_1 \cap \partial V_2$, whose length is $\approx n/2^{j/2}$. For fixed endpoints $p_1, p_2 \in \partial V$, the quantity $S'(j)$ satisfies the following recursive formula

$$S'(j) = S'(j+1) \times S'(j+1) \times n/2^{j/2}. \tag{16}$$

In order to apply the master theorem [8], we take logarithm

on both sides

$$\log S'(j) = 2\log S'(j+1) + \frac{1}{2}\log(N/2^j). \quad (17)$$

Substitute $A = N/2^j$, which is exactly the number of pixels in a tile at level j, and define $\tilde{S}(A) = S'(j)$. Then,

$$\log \tilde{S}(A) = 2\log \tilde{S}(A/2) + \frac{1}{2}\log A. \quad (18)$$

According to (4), denote $t(A) = \log \tilde{S}(A)$ and $f(A) = 0.5\log(A)$. Therefore, in this case $a = b = 2$ and $f(A) = O(A^{\log_2 2 - 0.5})$, and using the master theorem $t(A) = \log \tilde{S}(A) = \Theta(A)$. Subsequently, $S'(j) = 2^{O(N/2^j)}$ and combining this with Eq. (15),

$$S(j) = 6N \times 2^{O(N/2^j)}. \quad (19)$$

To derive an expression for K_L, it can be shown, that the average length over all candidate curves in a given tile at level j is proportional to the tile's area: $L = O(N/2^j)$. Therefore we can approximate K_L, by

$$K_L \approx 6N \times 2^{\beta L} \quad (20)$$

for some constant β. As mentioned above, we showed by empirical fitting that $\beta = 0.65$.

We remark that although the search space is exponential in L, there are nonetheless various curves that cannot be found by the RPT. Examples include closed curves like a circle, self-intersecting curves like a cross, and also any curve that begins and ends in the same side of a tile, like a narrow 'V' shape. Such geometric objects are detected by our algorithm as union of shorter valid curved edges.

6. Experiments

We tested our algorithm both on simulated artificial images as well as on challenging real images from several application domains. Our code is implemented in Matlab, yet we developed another preliminary implementation in C++. Furthermore, as our tree construction can be easily parallelized, our code utilizes multi-threading. We ran our experiments on a single 8-core Intel i7, 16 GB RAM machine. For an input image of size 129×129 pixels the run time in C++ is ≈ 0.6 second for the $O(N \log N)$ version and ≈ 0.9 seconds for the $O(N^{1.5})$, both including the post-processing step of computing the final edge map image E. For an input image of 257×257 pixels the corresponding C++ run-times are ≈ 5 and ≈ 8 seconds respectively. More CPU cores can reduce these run times significantly. The runtime graphs are shown in Fig. 6. Note, that our implementation outputs the edge image along with a list of all the curved edges that passed the statistical threshold. We compared our solution to the classical

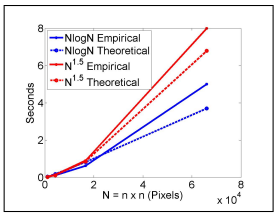


Figure 6. Empirical run-time of our implementation compared to the theoretical run-time for both $O(N \log N)$ and $O(N^{1.5})$ algorithms.

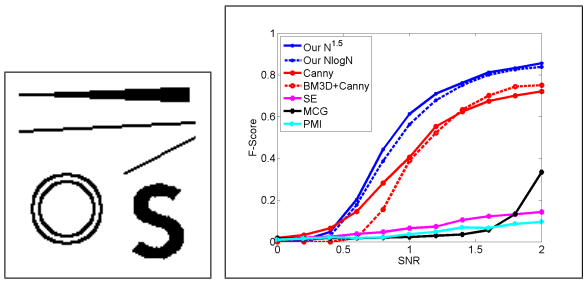


Figure 7. Simulation results. Left: A binary pattern used in simulation experiments. Right: F-measures obtained with various edge detection algorithms as a function of SNR.

Canny [7] algorithm, and to several states of the art algorithms for boundary detection in natural images, including Multiscale-Combinatorial-Grouping (MCG) [4], Crisp Pointwise-Mutual-Information (PMI) [15] and Dollar and Zitnick's Structured-Edges (SE) [10].

Simulations. For the simulations, we used a binary pattern of size 129×129 pixels that contains a narrow triangle, straight lines, concentric circles, and an 'S' shape, see Fig. 7. We then scaled the intensities and added additive Gaussian noise with 0.1 standard deviation to produce images with SNRs between 0 and 2 in intervals of 0.2. In addition, we added very weak salt and pepper noise that affects only 1% of the pixels in the images. We used the result of Canny on the clean binary pattern as the ground truth. We quantitatively compared the various edge detection algorithms by computing their Precision (P), Recall (R), and F-scores, using the popular F-measure for image segmentation described in [17].

Algorithm \ SNR Range	**0.0-0.8**	**1.0-1.6**	**1.8-2.0**
Our $O(N^{1.5})$ detector	**0.14**	**0.73**	**0.85**
Our $O(N \log N)$ detector	**0.12**	**0.7**	**0.83**
Canny	0.11	0.57	0.71
BM3D+Canny	0.04	0.57	0.75
SE	0.03	0.09	0.14
MCG	0.02	0.04	0.23
PMI	0.02	0.06	0.09

Table 1. Average F-measures obtained in simulations computed over images in three SNR ranges.

Table 1 shows the average F-scores obtained for each algorithm. A graph of the F-scores as a function of SNR is shown in Fig. 7. All methods are tuned to detect no edges

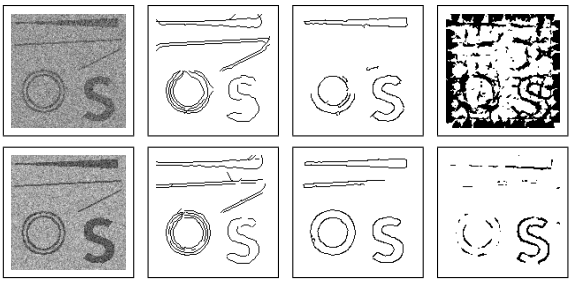
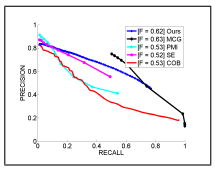


Figure 8. Result of applying various edge detection algorithms to the noisy simulation image, at SNR 2 (top), and SNR 3 (bottom). From left to right: Input image, our $O(N^{1.5})$, Canny and PMI.



Algorithm	ODS	OIS	AP
Ours	**0.60**	**0.62**	**0.51**
MCG	**0.63**	**0.63**	0.24
PMI	0.47	0.53	0.32
SE	0.52	0.52	**0.35**

Figure 9. Simulation results with the noisy BSDS-500 images. Left: Precision vs. Recall (PR) of contour detection by various algorithms. Right: Performance table. ODS refers to the F-Measure at the optimal threshold across the entire dataset, OIS to the best per image F-measure, and AP to the area under the PR curve.

when the image includes pure noise (SNR=0). For that sake, we modified the Canny's thresholds to be $[low, high] = [0.28, 0.7]$. For the other methods, we used a fixed threshold on the soft edge map. As expected, algorithms geared to detecting boundaries in natural images do not perform well in such noisy conditions. Canny manages to obtain fairly high F-scores. Still, our $O(N^{1.5})$ algorithm achieves the best F-scores at all SNR levels, while our $O(N \log N)$ variant achieves only slightly lower scores. These results suggest that high-quality detections can be obtained by our faster method, in spite of the significant reduction of complexity. Fig. 8 shows representative results of Canny, PMI, and our method at SNR=2 and 3.

We also made an extensive evaluation of our algorithm on the dataset of 63 artificial images containing various straight and curved edges, as studied by [1]. On this dataset our $O(N^{1.5})$ algorithm achieved an average F-score of ≈ 0.79, compared to ≈ 0.74 achieved by [1]. This demonstrates that our algorithm is not only significantly faster but also at least as accurate as [1].

We next applied our $O(N^{1.5})$ to the natural images of the BSDS-500 [17] to which we added Gaussian noise with $\sigma = 0.1$. The performance of the various methods is summarized in Fig. 9. Our results are superior to those of PMI [15] and SE [10] and are comparable to MCG [4]. We emphasize that while our method is better suited to deal with the noise it is currently not designed to handle natural textures that are ubiquitous in these images.

Real Images. We further tested our algorithms on various real images taken under relatively poor conditions, see

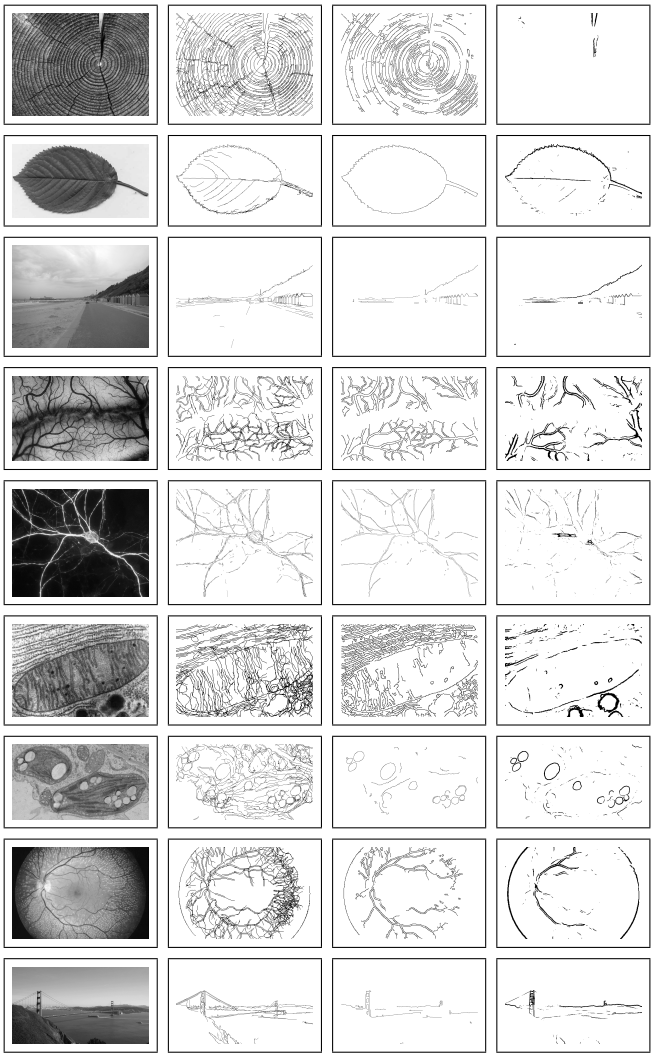


Figure 10. Real images: Each row shows the original image and a comparison of our binary result, with those obtained by Canny (middle) and by PMI (right).

results in Figure 10. It can be seen that our method manages to accurately detect details in these challenging images beyond those detected by existing approaches. Specifically, our method depicts nearly all the growth rings of the tree, the branchings of the nerve cell, and the blood vessels in the retina. Accurate detection of such structures can for example facilitate high throughput biomedical research.

7. Conclusion

We presented efficient algorithms for detecting faint curved edges in noisy images that achieve state-of-the-art results at low SNRs. Our novel approach detects curved edges in only $O(N \log N)$ operations. The algorithm is adaptive to various parameters such as edge length, shape, and SNR and can be applied with various filters. Thus it may be applicable in a variety of imaging domains.

References

[1] S. Alpert, M. Galun, B. Nadler, and R. Basri. Detecting faint curved edges in noisy images. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 750–763, Berlin, Heidelberg, 2010. Springer-Verlag. 2, 6, 8

[2] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *pami*, 33(5):896–916, 2011. 2

[3] P. Arbelaez, M. Maire, C. C. Fowlkes, and J. Malik. From contours to regions: An empirical evaluation. In *CVPR*, 2009. 2

[4] P. Arbeláez and J. Pont-tuset. Multiscale combinatorial grouping. In *CVPR*, 2014. 2, 7, 8

[5] H. Bi, S. Cook, H. Yu, M. Benfield, and E. Houde. Deployment of an imaging system to investigate fine-scale spatial distribution of early life stages of the ctenophore mnemiopsis leidyi in chesapeake bay. *Journal of Plankton Research*, 35:270–280, 2013. 1

[6] A. Buades, B. Coll, and J.-M. Morel. A non-local algorithm for image denoising. CVPR '05, pages 60–65, 2005. 2

[7] J. Canny. A computational approach to edge detection. *IEEE Trans. Pattern Anal. Mach. Intell.*, 8(6):679–698, June 1986. 1, 2, 7

[8] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press Cambridge, 2001. 5, 6

[9] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian. Image denoising with block-matching and 3d filtering. In *Electronic Imaging 06*, 2006. 1, 2

[10] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *ICCV*, 2013. 2, 7, 8

[11] D. L. Donoho and X. Huo. Beamlets and multiscale image analysis. In *in Multiscale and Multiresolution Methods*, volume 20, pages 149–196, 2001. 2

[12] M. Galun, R. Basri, and A. Brandt. Multiscale edge detection and fiber enhancement using differences of oriented means. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007. 2

[13] R. Grompone von Gioi, J. Jakubowicz, J.-M. Morel, and G. Randall. Lsd: A line segment detector. *Image Processing On Line*, 2:35–55, 2012. 6

[14] I. Horev, B. Nadler, E. Arias-Castro, M. Galun, and R. Basri. Detection of long edges on a computational budget: A sublinear approach. *SIAM Journal on Imaging Sciences*, 8(1):458–483, 2015. 2

[15] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*, 2014. 2, 7, 8

[16] D. Marr and E. Hildreth. Theory of Edge Detection. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 207(1167):187–217, 1980. 2

[17] D. R. Martin, C. C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(5):530–549, May 2004. 2, 7, 8

[18] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12:629–639, 1990. 2

[19] I. Sobel. Camera models and machine perception. In *PhD thesis, Electrical Engineering Department, Stanford University.*, 1970. 2

[20] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Computer Vision, 1998. Sixth International Conference on*, pages 839–846, 1998. 2

[21] P. S. P. Wang and J. Yang. A review of wavelet-based edge detection methods. *IJPRAI*, 26(7), 2012. 2