Efficient Bayesian Detection of Faint Curved Edges in Noisy Images

Nati Ofir

Department of Computer Science and Applied Mathematics
Weizmann Institute of Science
Rehovot 76100, Israel

Abstract. Detecting edges in images is a fundamental problem in computer vision with many applications. A large number of edge detection algorithms have been proposed during the past several decades. These algorithms can deal effectively with the problem, but they often face difficulties when applied to images taken under poor visual conditions in which faint edges and noisy backgrounds are produced. Such conditions occur frequently in a variety of imaging domains including biomedical, satellite, and high shutter speed, and may even occur in natural images as well. In this work, we introduce an efficient method to detect faint edges in noisy images. The first question we address is how to efficiently detect curved edges. Previous work showed that faint edges can be detected by applying a search over the space of possible curves. While this search space is exponentially large in the number of image pixels, our new multiscale algorithm manages to search through a large subset of the space in practical polynomial time. Our algorithm is based on a novel hierarchical partitioning of the image into triangular tiles. The second question we address is how to decide if a curve in the image indeed corresponds to a (possibly faint) edge. To that end, we developed a Bayesian approach that combines intensity and shape considerations. Our method takes advantage of statistical priors on edge contrast and shape. We further utilize natural images to derive a prior on-edge contrast. As our experiments demonstrate, in comparison to previous works our algorithm is more efficient and obtains higher quality of edge detection.

1 Introduction

This work addresses the problem of efficient detection of faint edges in noisy images. It extends the work of Galun et al. [1] and Alpert et al. [2], which respectively introduced algorithms for detecting faint straight and curved edges. Below we improve upon this work by proposing a practical complexity algorithm for detecting *curved* edges using an efficient multi-scale approach. Denote by N the number of pixels in an image. Alpert et al. [2] introduced a curved edge detection algorithm that works in $O(N^{2.5})$ computational complexity, while our method reduces the complexity to $O(N^{1.5})$. In addition, we introduce a Bayesian decision mechanism for edge detection that incorporates priors on the contrast and shape of an edge.

Noisy images with low signal-to-noise ratio (SNR) are common in a variety of imaging domains, significantly where images are captured under poor visibility conditions. Examples include biomedical, satellite, and high shutter-speed imaging. Noise poses a problem to edge detection, since the variability of local gradients, due to noise, can be relatively high. Methods like Canny [3] that use the local image gradients as a baseline will be disrupted by noise. Therefore, Canny, for example, performs noise reduction by convolving the image with a Gaussian filter before edge detection. Smoothing the noise with a Gaussian filter, however, reduces the noise on the one hand, but also reduces the edge contrast on the other hand. In general, common denoising methods do not reduce the noise sufficiently to improve the detection of faint edges.

To address this issue, we use a global filter that matches the edge curves and smooths the noise only along both sides of the edge. As this edge filter is spatially longer, the filter more aggressively reduces the noise allowing detection of fainter

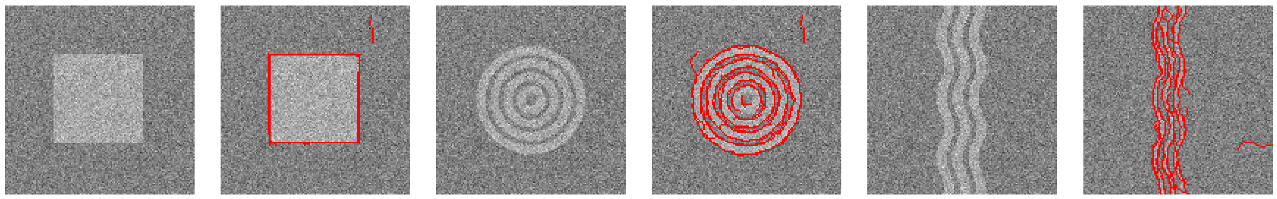

Fig. 1. Three examples of noisy images along with our detection results. The images include a pattern of a square, concentric circles, and sine waves.

edges. However, the edge curve is unknown apriori. Therefore, this method can be viewed as a search in the space of possible curved edges, where a global edge filter is applied to each curve.

Early edge detection algorithms include the Sobel operator [4], which performs edge detection by thresholding the gradient magnitude for each pixel separately. Canny [3] extended Sobel by tracing edge trajectories and by hysteresis thresholding of the local gradient magnitudes. Marr & Hildreth [5] apply a 2D Laplacian of a Gaussian filter and identify edges by detecting zero crossings. These local approaches for edge detection are on the one hand very efficient, but on the other hand sensitive to noise, and therefore typically exhibit poor performance at low SNR.

Several methods were developed to smooth image noise while avoiding the reduction of edge contrast. Tomasi & Manduchi [6] use bilateral filtering to smooth only non-edge pixels. Perona & Malik [7] introduced anisotropic diffusion to smooth with adaptive 2D Gaussian shape. A sophisticated edge filter was described in the state-of-the-art algorithm gPb [8], [9], [10] for image segmentation. This algorithm filter contains two half disks that are matched to each side of the edge. It is designed for natural image edges, and when applying this filter with a large disk radius it assumes a straight line edge. These approaches manage to detect edges in natural images, but the local filters limit their ability to overcome low SNR conditions.

As described by Wang & Yang [11], several edge detection methods are based on the wavelet transform and its variable width filters. An example of using a global filter for edge detection can be seen in the work of Felzenszwalb & McAllester [12]. This work introduces a dynamical programming algorithm for detecting a salient curve in an image, it searches only for the response of the best curve and produces its approximate localization. Galun et al. [1] use straight-line global filters of different lengths to perform faint edge detection. They scan all image lines efficiently in $O(N \log_2 N)$ operations, according to a method developed by Brandt & Dym [13]. Alpert et al. [2] utilize the Beamlet transform [14] to detect faint curved edges. Their method manages to perform faint edge detection with high quality but in a non-practical computational complexity. In our work, we show how their method can be improved in terms of quality and complexity.

Overcoming the noise by using a Bayesian approach can contribute to faint edge detection. Existing Bayesian approaches for image denoising use various strategies. Simoncelli [15] works in the wavelet domain and utilizes an empirical distribution of derivatives in natural images. Konishi et al. [16] offer an edge detection method that is based on a probabilistic decision function. They use an empirical distribution of the gradient magnitudes at the edge and compare it to a distribution of the gradient magnitudes in noise. Levin et al. [17] introduced a method for image deconvolution that takes advantage of several image derivative priors to overcome noise.

Prior knowledge of the shape of a curved edge may help to overcome the poor SNR conditions as well. Smoothness before curves was introduced in the literature in different contexts. Williams & Jacobs [18] infer the shape of an illusion contour as the most likely random walk with stochastic turns. Sharon et al. [19] introduce a similar idea of edge completion using the minimal energy assumption in terms of elastic energy.

In Section 2 we introduce a multi-scale algorithm for scanning a search space of curved edges efficiently. The algorithm applies a decision function on every curve in the search space to decide whether a curve belongs to an edge in the image. This method is based on a Triangle-Partition-Tree (TPT) that divides at each level an image tile of the shape of an isosceles right triangle (IRT) (Angles: $45°, 45°, 90°$) into two smaller IRTs. This TPT method works in $O(N^{1.5})$ operations. We further show empirically that it can detect edges more accurately than previous methods [1, 2].

In Section 3 below, we introduce a method for combining the global curve filter and the curve smoothness assumptions into a Bayesian decision function based on given distributions. We further introduce a parametric distribution that approximates the real distribution of image derivatives across edges and noise and contrast it with an empirical distribution derived from the Berkeley Segmentation Dataset [20]. We apply this Bayesian decision function in the TPT method, and together they both form an efficient algorithm for the detection of faint curved edges.

Our algorithm is designed to detect all curved edges in the image and to output their exact location. Moreover, it is designed to be adaptive to the noise level in the image. Our work makes several significant contributions to the study of edge detection. The main contribution is the introduction of an efficient algorithm for detecting faint curved edges. This challenging problem can be solved in $O(N^{1.5})$ operations. The second major contribution is the introduction of an empirical observation of the distribution of derivatives across edges in natural images. This empirical density can be used as a prior on natural images, and it is useful in particular for edge detection. The third contribution is the use of shape prior for curved edge detection. Our experiments show that the shape prior, integrated into our Bayesian decision mechanism, improves the accuracy of our detections.

2 Efficient Search for Curved Edges

Detection of faint edges in noisy images is a challenging problem. To detect edges at low SNRs, we use the approach described in [1] and [2] that applies filters that match the shape of the curves they trace. These filters smooth the noise without reducing the edge contrast, moreover, as the filter is longer it reduces the noise more aggressively. However, since the curved edge locations are unknown apriori, we must search through the set of possible curves in an image.

Our algorithm is designed to detect faint curved edges by scanning this search space. The total number of possible curves in an image is exponential in the number of pixels. Therefore, naive approaches, such as scanning all curves and computing an edge score for each curve cannot work in polynomial time. Moreover, when all curves are allowed, there is a significant probability of detection of false edges due to the noise. In other words, it may reduce the ability of an edge detection algorithm to distinguish between false and true curved edges.

For now, assume that we are working on a square image of size $N = n \times n$. Alpert et al. [2] developed an algorithm for scanning the curve search space in $O(N^{2.5})$ time complexity. Their algorithm is designed to scan a large subset of curves that contains the strongest filter responses. Their multiscale algorithm solves the problem of scanning all curves within a square by scanning all curves in 4 smaller squares. Then it concatenates the sub-curves of the smaller squares to a longer curve within the original square. This algorithm uses a data structure named Quad-Pyramid.

In this section, we introduce a significant improvement of the Quad-Pyramid algorithm. Our data structure, which we call a Triangle-Partition-Tree (TPT), enables the detection of faint curved edges in $O(N^{1.5})$ operations. The general idea behind the algorithm, like these of [2], is to work only on a large subset of curves and to concentrate the computational effort on sub-curves with high score responses. Geometrically, we solve the problem of scanning all curves in an isosceles right triangle (IRT) (Angles: $45°, 45°, 90°$) by firstly scanning all sub-curves in 2 smaller IRTs. Then we concatenate the sub-curves to longer curves within the original IRT. See Figure 2 for an illustration.

Denote by *EdgeScore* a scoring function result, such that for every curve in the search space of possible curves in an image, a curve belongs to an edge if its *EdgeScore* is greater than some threshold. For this section, we assume the existence of a function that maps from curves in the search space to a score $EdgeScore \in \mathbb{R}$. Moreover, we assume that once we calculated *EdgeScore* for two different sub-curves that can be stitched due to a shared endpoint, the *EdgeScore* of a curve induced by their stitching can be calculated in $O(1)$ operations. A simple example of such *EdgeScore* is the mean contrast along a curve. If we calculate the mean contrast of two sub-curves, we can easily compute the mean contrast of their stitching, as long as we store the sub-curve lengths.

2.1 Triangle-Partition-Tree Construction

Let $\Omega \subset \mathbb{R}^2$ be the discrete grid of image pixels, of size $N = n \times n$. In every tree level, we cover Ω with triangle tiles of the same size. Since the number of image pixels is N, and in every tree level we split a tile in the image into 2 equals sub-tiles, the height of the tree is $\log_2(N)$. We denote $j = 0, 1, ..., \log_2(N)$ as the levels of the tree, where $j = 0$ is the tree-root level, and $j = \log_2(N)$ is the leaf level. In terms of the grid Ω, a leaf is the minimal size triangle whose vertices lie on the grid, while the root corresponds to the whole image grid, divided into two IRTs. Every triangle at level j is subdivided into two child triangles at level $j + 1$. Both children have the identical shape of an IRT such that each one occupies half of the area of the parent triangle. We denote the length of the sides of a triangular tile by the number of pixels it traverses minus one, i.e., the length is the infinity norm of the side. We further parameterize an IRT by a triplet (a, b, c) with a and b representing the length of the two equal sides and c representing the length of the hypotenuse. At every level j, Ω is covered by 2^{j+1} triangle tiles. At every even level j, Ω is covered by triangle tiles of size $(n/2^{0.5j}, n/2^{0.5j}, n/2^{0.5j})$. At every odd level j, Ω is covered by triangle tiles of size $(n/2^{0.5(j+1)}, n/2^{0.5(j+1)}, n/2^{0.5(j-1)})$. See Figure 2 for the illustration of the 3 first levels of TPT. The splitting edge of a tile is the edge that splits a triangle tile of level j into 2 tiles of level $j + 1$. The length of the splitting edge of a tile in every even level j is $n/2^{0.5(j+2)}$, while in every odd level j, the length is $n/2^{0.5(j+1)}$.

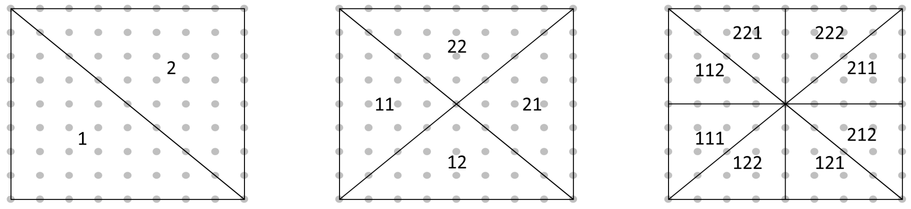

Fig. 2. The 3 topmost levels of the Triangle-Partition-Tree. Left: level 0 of TPT, a partition of $n \times n$ image into 2 triangles with edge length (n, n, n) in infinity norm. Middle: In level 1 of TPT, every triangle of level 0 is divided into two triangles of level 1. The whole image is partitioned into four triangles with edge length $(n/2, n/2, n)$. Right: In level 2 of TPT, every triangle of level 1 is divided into two triangles of level 2. The whole image is partitioned into 8 triangles with edge length $(n/2, n/2, n/2)$. The partitioning of the next levels, from level 4 up to level $\log_2 N$ is done recursively in the same way as described here.

Denote by S_j a triangle tile at level j. A curve at this level begins at a point p_1^j which lies on one of the sides of S_j and ends at a point p_2^j that lies on one of the other two sides of the same tile.

At the finest level $j = \log_2(N)$, a curve is a straight line that connects $p_1^{\log_2 N}$ and $p_2^{\log_2 N}$. In any coarser level $j < \log_2(N)$, a curve connecting p_1^j and p_2^j is constructed from two $j+1$ level curves such that:

1. Their concatenation produces a continuous curve that connects p_1^j and p_2^j.
2. From all the $j+1$ level pairs of curves connecting p_1^j to p_2^j that fulfill the first condition, the selected pair of curves achieves the maximal $EdgeScore$.

Figure 3 shows an illustration of a curve in the Triangle-Partition-Tree data structure.

2.2 The Algorithm

We present an algorithm for faint edge detection that uses the TPT data structure:

1. **Triangle-Partition-Tree Tiling**: Given an image of $N = n \times n$ pixels, divide the image into two IRTs. Then, recursively subdivide every IRT into two smaller IRTs until reaching the bottom level. In our implementation, the bottom level is constructed from triangles of edge lengths $(5, 5, 5)$ in the infinity norm.
2. **Initialization**: Construct the bottom level of the TPT by computing the scores of all straight edges of all of the triangular tiles at this level. Consider all the straight edges connecting a pixel on one side of a tile to a pixel on another side. Denote by $corners$ the number of corner pixels of a triangle, $sidePairs$ the number of pairs of triangle sides, $sideLength$ the length of a triangle side. The number of such straight lines in a tile is $sidePairs \times sideLength^2 - corners = 3 \cdot 5^2 - 3 = 72$. The number of operations done in the initialization step is $O(N)$.
3. **Tree Data construction**: Construct the upper level $j-1$ using the data computed at level j. Obtain curve edge score by stitching two curve edge scores of the lower level. Compute $EdgeScore$ for every possible curve path. Select for each two endpoints the curve made by valid stitching that maximizes $EdgeScore$.
4. **Edge Selection**: In post-processing, select curves in the TPT data in all tree levels that satisfy the criterion $EdgeScore$ if greater than a threshold. Discard curves whose local derivatives are far from constant and apply spatial non-maximal suppression.

2.3 Computational Complexity

Denote by P the number of image pixels in a tile, and by $T(P)$ the number of operations done by our algorithm running on a tile of P pixels. Therefore, the total complexity of our algorithm is $T(N)$. At the topmost level, we divide the image into 2 IRTs, each one containing roughly half of the image pixels. Then, we recursively sub-divide every IRT into two smaller IRTs, such that if the larger IRT contains P image pixels each smaller IRT contains roughly $P/2$ image pixels. Therefore, the recursion formula takes the general form $T(P) = 2T(P/2) + f(P)$ for a function f specified below. The set of curves we scan in every tile includes all the curves that connect two of the sides of the tile (See Figure 2). The number of curves we

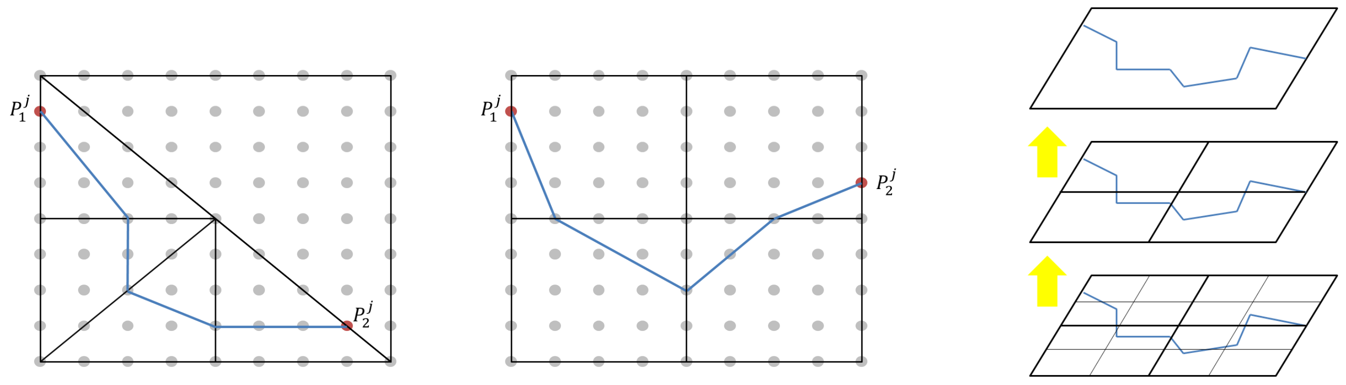

Fig. 3. From left to right: A curve (in blue), connecting points P_1^j and P_2^j on two sides of a triangle 1 in TPT, is obtained by concatenating two curves in triangles 12 and 11. In turn, these two curves are obtained by concatenating four straight lines of the finest level. Note that this curve begins on a point P_1^j that lies on one side of a tile and ends on a point P_2^j on another side of the same tile. In general, all non-self-intersecting curves that begin and end on two different sides of a tile are considered optional curves by TPT. In the middle, subdivision of 9×9 grid by the **Quad-Pyramid** [2] up to tiling of 5×5 squares. The root level includes the entire 9×9 grid. The subsequent level contains four 5×5 smaller squares. A blue curve at the root level begins on one side of the square and ends on another side. The curve is constructed by stitching four edges from the next lower level. On the right, we show the three topmost levels of the Quad-Pyramid. The computational complexity of constructing the Quad-Pyramid is $O(N^{2.5})$ for a square image with N pixels.

consider at every tile with P pixels, therefore, is the number of pairs of points along any two sides of a tile. Since a triangle has 3 sides, the number of pairs of sides is $\binom{3}{2} = 3$. Furthermore, since a triangle side is of length $O(\sqrt{P})$, the number of curves connecting two sides is $O(P)$. Therefore, the order of all the curves we consider in each tile is $O(P)$ as well.

Next, we consider the computational complexity needed to construct each curve. Note that each curve is constructed to achieve the maximal response out of all possible curves that share the same endpoints. For each curve in level j we maximize *EdgeScore* over the set of all possible stitching of curves from the level $j+1$. This set size is equal to the length of the splitting edge of a triangle tile, i.e., the edge that splits a large IRT into 2 smaller IRTs. If the number of pixels in a tile is P, then this edge length is $O(\sqrt{P})$. Therefore, the work in every level is $f(P) = O(P \cdot \sqrt{P})$.

In conclusion, the computational complexity of the Triangle-Partition-Tree construction can be written by the following recursive formula:

$$T(P) = 2T(P/2) + O(P \cdot \sqrt{P}) = O(P^{1.5}) \tag{1}$$

According to the master theorem [21], the total complexity of the algorithm is $T(N) = O(N^{1.5})$. See A.4 for a direct and more detailed complexity analysis including the multiplicative constants.

We can learn from this complexity analysis that the complexity of $O(N^{1.5})$ can be achieved also by other binary partitions of an image. In the analysis above we used only the following properties of our partition: the partition splits a tile into two equal tiles and it is done in each tile by a straight line of length $O(\sqrt{P})$ where P is the number of pixels in the tile. It means that the splitting line of a tile should decrease as we go down in the recursive hierarchy. Therefore, any algorithm that shares the same method of TPT except of using different partitions that fit the two properties, achieves the same complexity up to constant factors. For example, we can use the following partition method: split an image square tile into two equal rectangles, then split each rectangle into two equal squares, and repeat the splitting recursively. We can show that the above analysis holds for this binary partitioning as well. However, the advantage of partitioning by IRTs is its symmetry: there are two ways to split a square to equal rectangles, many ways to split a tile into two equal tiles, but only one way to split an IRT into two equal IRTs. Therefore, TPT is the only algorithm that is invariant to image rotation by 45° or 90°.

2.4 Runtime Complexity Analysis of the Quad-Pyramid

Alpert et al. [2] introduced the Quad-Pyramid data structure for computing efficiently the *EdgeScore* of all curves in the image. In this sub-section, we describe the Quad-Pyramid complexity and its differences relative to our TPT data structure.

Note that the general outline is very similar for both methods, the main inherent difference is that the Quad-Pyramid divides a square into 4 smaller squares in every recursive step. See Figure 3 for Quad-Pyramid visualization.

In this section, denote by $T_Q(P)$ the computational complexity of constructing the Quad-Pyramid on an image tile of P pixels. Therefore, the total complexity of the Quad-Pyramid is $T_Q(N)$. While in TPT a tile is a triangle, now a tile is a square. In every pyramid level, we divide a square tile of P pixels into 4 smaller square tiles that contain roughly $P/4$ pixels each. Therefore, the recursion formula is of the type $T_Q(P) = 4T_Q(P/4) + f(P)$. In every tile, we consider all the curves that begin on one side of a tile and end on another side. Since a square has 4 sides, the number of pairs of sides is $\binom{4}{2} = 6$. For each pair of sides, we consider $O(P)$ curves since a single edge of a square is of size $O(\sqrt{P})$. Therefore, the order of the number of curves to consider in every tile is $O(P)$, the same order of curves per tile as in the TPT.

The main difference between the two methods is the work needed to compute the maximal *EdgeScore* for each curve given fixed endpoints. A curve at level j in the Quad-Pyramid is constructed from all valid stitching of curves from level $j+1$. While in TPT each curve in level j is constructed from two curves of level $j+1$, in the Quad-Pyramid, we stitch four curves of level $j+1$ to construct a curve at level j. To compute the curve at level j, we maximize *EdgeScore* over the set of all possible stitchings. Once we fix the two endpoints of a curve, this set size is equivalent to the length of an edge of a tile at level $j+1$ to the power of 3. The explanation is that every curve can go through any pixel in the three splitting tile edges of the lower level. If the number of pixels in a tile is P, then the tile-side length of the lower level is $O(\sqrt{P}/2)$. Therefore, the work done in every tile is $f(P) = O(P \cdot (\sqrt{P})^3) = O(P^{2.5})$.

In conclusion, the computational complexity of the Quad-Pyramid construction can be written by the following recursive formula:

$$T_Q(P) = 4T_Q(P/4) + O(P \cdot (\sqrt{P}/2)^3) = O(P^{2.5}) \tag{2}$$

According to the master theorem [21], the total complexity of the algorithm is $T(N) = O(N^{2.5})$.

2.5 Search Space Size

To compute the search space size for every curve length L, we will continue the methodology and notations described in Section 2.1.

The number of curves computed at each level j is equal to the number of pairs of points along the sides of all triangles. At every level j, the image grid Ω, is covered by 2^{j+1} triangle tiles. The number of pairs of endpoints to consider at every tile S_j is roughly $3(n/2^{0.5j})^2$, since a triangle includes 3 edges, the number of pair of edges is also 3, and a triangle edge length in level j is roughly $n/2^{0.5j}$.

Therefore, the total number of pairs of endpoints to consider at level j is the number of tiles times the number of curves per tile which is: $2^{j+1} \cdot 3(n/2^{0.5j})^2 = 6n^2 = 6N$

To compute the total number of possible curves for each fixed endpoint in our search space, we should look deeply into the algorithm recursion. Therefore, we consider that each curve from level j achieved a maximal response out of a set of curves made by stitching of sub-curves of level $j+1$. Then, if we denote by $K(j)$ the number of curves that can go through two endpoints in level j in our search space, it equals the number of valid stitching of 2 curves from level $j+1$, which is equivalent to the splitting edge size. Moreover, each one of the two sub-curves of level $j+1$ was picked out of its own search space of possible curves. Since the splitting edge of level j is of length $n/2^{0.5(j+1)}$, the formulation of K(j) is as follows:

$$K(j) = K(j+1)^2 \cdot n/2^{0.5(j+1)} \approx 2^{2N/2^j} \tag{3}$$

It can be shown that $K(j) \approx 2^{2N/2^j}$, see Appendix A.5 for the derivation. We can assume that the average length of a curve at level j is $L = \alpha \times 2^{\log_2(N)-j}$ where α is a constant related to the average length of a straight edge in a triangle tile of the finest level. In our algorithm, α is approximately 3 pixels. Now, we can derive an expression for the search space size of curves of length L. This size equals the number of pairs of endpoints to consider in level j ($6N$) times the search space size of each endpoint in the level ($K(j)$):

$$K_L = 6N \cdot K(j) = 6N \cdot 2^{2L/\alpha} \approx 6N \cdot 2^{2L/3} \tag{4}$$

2.6 Space Complexity Analysis

As we showed in the search space analysis, the number of curves in any TPT level j is $6N$. The number of levels in the TPT data structure is $\log_2(N)$ since in every level we split a tile into two equal tiles. For each curve, we store $O(1)$ data that includes its *EdgeScore* and other relevant data for computing *EdgeScore* efficiently for further stitching. Therefore, the space complexity is as follows:

$$SpaceComplexity = 6N \cdot \log_2 N \cdot O(1) = O(N \log_2(N)) \tag{5}$$

2.7 3D edge detection

Edge detection algorithms are traditionally applied to 2D images. Nevertheless, edges are relevant features in 3D images as well. For instance, Magnetic Resonance Imaging (MRI) can produce medical 3D images where the edges may represent blood vessels or neurons. An interesting question, therefore, is whether we can modify our partition method to scan all curved edges in a 3D image. To that end, we use the 2D partition method that we mentioned briefly in Section 2.3. This method partitions a square tile into two equal rectangle tiles, and then partition each one of the rectangles into two equal square tiles. This method, like the TPT method, achieves an efficient scan in $O(N^{1.5})$ computational complexity, we decided to modify this one specifically since its 3D form is more intuitive.

Consider a 3D image as a cube in $\mathbb{R}^3$. Denote by N the number of pixels in the cube and by n the cube edge length, so that $N = n^3$. For simplicity, assume that the origin lies at the center of the cube, and each cube face is parallel to one of the axial plains (XY, XZ, or YZ). The first partition is done by a $n \times n$ square that lies on the XY plane. This square splits the cube into two equal cuboids of size $n \times n \times n/2$. The second partition is done by squares of size $n \times n/2$ that lie in the XZ plane. They split each cuboid into two equal cuboids of size $n \times n/2 \times n/2$. Next, we split each one of the new cuboids by $n/2 \times n/2$ square that lies in the YZ plane and remain with eight cubes of size $n/2 \times n/2 \times n/2$. Then, we split every cube recursively in the same way. In conclusion, we briefly introduce a partition of a 3D cube so that at each level we split a 3D tile into two 3D tiles that are smaller and equal. The separator is a 2D square (or rectangle) of size $O(n^2)$ pixels while the tile is of volume $O(N) = T(n^3)$ pixels.

This partition modifies our square and rectangle 2D partition into a 3D cube and cuboid partition. Note that we could modify the IRT's partition in a similar approach. We would like to compute the runtime complexity of our 3D method. Denote by $T_{3D}(N)$ the computational complexity of our 3D partition algorithm, $T_{3D}(P)$ the complexity of the algorithm running on a 3D tile of $P = p^3$ pixels. In each level, we split a 3D tile of P pixels into two equal tiles of roughly $P/2$ pixels each. Therefore, $T_{3D}(P) = 2T_{3D}(P/2) + f(P)$ like any balanced binary tree. A 3D tile face is of order $O(p^2)$ pixels. In each tile, we consider all the curves that begin on one face and end on another face. The number of curves we consider in every tile, therefore, is $O((p^2)^2) = O(p^4)$. Then a 2D square splits every tile into two smaller tiles, this square size is of order $O(p^2)$ as well. Therefore, for each curve in a tile, we consider $O(p^2)$ splitting points that lie on the splitting square. Once we fix the two endpoints of a curve at level j, each splitting point induces a stitching of two curves of level $j + 1$. For each two endpoints of level j, we select the splitting point that yields a curve with maximal *EdgeScore*. Then, the amount of work in a tile is the number of curves to consider times the work per curve: $f(P) = O(p^4 \cdot p^2) = O(P^2)$. Now, we can derive the computational complexity of our 3D partition according to the master theorem [21]:

$$T_{3D}(P) = 2T_{3D}(P/2) + O((p^2)^2 \cdot p^2) = O(p^6) = O(P^2) \tag{6}$$

In conclusion, our efficient 2D algorithm can be modified into an edge detection algorithm of 3D curved edges that works in $T(N) = O(N^2)$ operations.

3 Bayesian Edge Detection

Detecting faint edges in noisy images poses several difficulties. Galun et al. [1] described this problem in a synthetic theocratical scenario where the edges are step edges with low contrast and the noise in the image is Gaussian i.i.d, i.e. $I(x, y) = S(x, y) +$

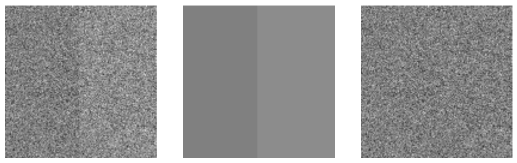

Fig. 4. From left to right: Given a synthetic noisy image $I = S + N$, the clean image contains one weak step edge, while the pure noise image N is composed of pixels whose intensity values are distributed normally.

$n(x, y)$, where I is the observed image, S is the noiseless signal of faint step edges and $n(x, y) \sim \mathcal{N}(0, \sigma_n^2)$ is the pixel independent additive noise (Figure 4). Due to the noise and the faintness of the edge, the contrast along the edge in I can be very faint and can even flip sign. Therefore, local approaches for edge detection will not be able to detect edges under these conditions.

Denote by μ_s the mean absolute contrast in the noiseless image $S(x, y)$. The classic definition of the SNR under this model is the ratio between the mean contrast of an edge in the clean image (μ_s) and the standard deviation (σ_n) of the noise:

$$SNR = \frac{\mu_s}{\sigma_n} \tag{7}$$

Alpert et al. [2] introduced an ideal filter for edge detection that matches the curve traced by an edge, and then computes the edge contrast by averaging the contrasts along the edge. As the filter becomes longer, the noise is averaged more aggressively and the filter can detect fainter edges.

This filter is designed to measure the contrast of edges. A curved edge is a continuous curve that produces local maxima of contrast. When detecting faint edges in noisy images, the ideal match filter is necessary to extract the maximal contrast of an edge in the sense that, if the filter does not match the exact edge trace, then we expect the filter response to be smaller. An additional reasonable assumption for edges is that their shape is smooth, i.e. low value of the elastic energy function. In this section, we introduce a curve descriptor for curved edges that uses both the curve shape and the pixel intensities along it. The ideal match filter can be computed from the curve descriptor. We describe a Bayesian approach to determine if a given curve represents an edge or noise using this descriptor. This approach combines both contrast and shape cues.

3.1 Curve Descriptor

We introduce a curve descriptor for edge detection, see Figure 5 for illustration. Denote by l_i the intensity on the left side of the curve in location i, r_i the corresponding right side intensity, and c_i the corresponding contrast in location i. Denote by θ_i the curve turns in location i, L the curve length, and w its width. The descriptor is defined as follows:

$$\mathcal{CD} \equiv \{c_1 = 0.5(l_1 - r_1), ..., c_L = 0.5(l_L - r_L); \theta_1, ..., \theta_{L-1}; w, L\} \tag{8}$$

The above $\mathcal{CD}$ can be used in different spatial lengths, different orientations, and different widths. Variable descriptor widths can be achieved by sampling l_i and r_i in a width of w pixels orthogonal to the curve. The descriptor should ideally match the curve traced by an edge and should be useful for distinguishing between edge and noise.

3.2 Probabilistic Decision Function

By extracting the curve descriptor $\mathcal{CD}$, we examine if there exists an edge in the image whose location and shape match that descriptor. However, since the edge location and the edge shape are unknown apriori, we need to search the space of curved edges. During the search, we extract the descriptor from each curve and decide whether an instance in the search space is an edge or not. In conclusion, we use the following approach for edge detection: 1) scan all curves within the search space 2) for each curve, extract the descriptor $\mathcal{CD}$ 3) According to the descriptor, decide if this curve is an edge or noise. To efficiently scan the curves we use the Triangle-Partition-Tree described in Section 2.

To decide if $\mathcal{CD}$ belongs to an edge or noise/background we describe here a Bayesian approach. After extracting the curve descriptor from a curve, we accept it as an edge if the probability for the curve to be an edge given its descriptor is greater

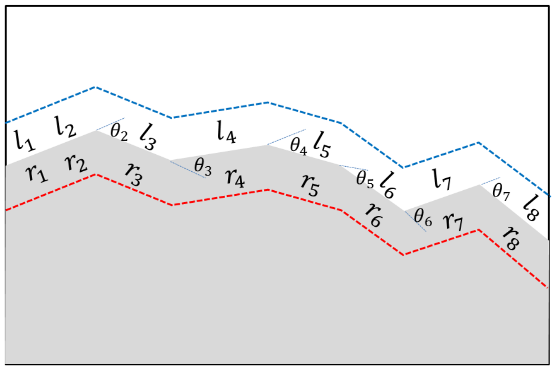

Fig. 5. An ideal matched filter samples the intensities and contrasts along the edge and describes the filter shape as a list of turn angles. Zero-degree turns are not marked.

than the probability for the curve to be noise given the same descriptor:

$$\frac{P(edge|\mathcal{CD})}{P(noise|\mathcal{CD})} > 1 \tag{9}$$

This test can be seen as requiring the a posteriori ratio (*AposterioriRatio*) test to have a score greater than 1. The probabilities of edge or noise given a curve descriptor are hard to measure directly. We therefore would like to apply the above *AposterioriRatio* test through a likelihood ratio (*LikelihoodRatio*) test. For that purpose, we should model or measure the likelihood functions $P(\mathcal{CD}|edge)$ and $P(\mathcal{CD}|noise)$, and the priors $P(edge)$ and $P(noise)$. We can model these probabilistic functions theoretically or measure them empirically from a given relevant training set.

To translate the *AposterioriRatio* test into a *LikelihoodRatio* test, we first discuss the prior $P(edge)$. Denote by K_L the number of curves of length L in the search space. Since we perform edge detection as a search in the space of image curves, we will use a sparsity assumption in this space, i.e., we expect to detect only $O(\sqrt{K_L})$ edges in the space for every curve length. Therefore, we assume the following sparsity assumption:

$$P(edge) \approx \frac{1}{\sqrt{K_L}} \tag{10}$$

Now, by assuming that every curve can be in either of two states: edge or noise, we derive from Equation (9) an equivalent *LikelihoodRatio* test:

$$\frac{P(\mathcal{CD}|edge)}{P(\mathcal{CD}|noise)} > \sqrt{K_L} \tag{11}$$

We introduce an approach for measuring these likelihood functions while assuming statistical independence between the contrast cue and the shape cue. In addition, we will examine the contrast cue only through the mean contrast along the edge. The mean contrast can be derived directly from the curve descriptor as follows:

$$\bar{c} = \frac{1}{L}\sum_i c_i \tag{12}$$

A curve descriptor that matches the shape of a curved edge, can be used to compute the mean contrast $\bar{c}$ and to perform an optimal denoising by averaging the noise along the curve. By assuming that contrast and shape are independent and by considering only the mean contrast, we derive from Equation (11) the following inequality:

$$\frac{P(\bar{c}|edge)P(\theta_1,...,\theta_{L-1}|edge)}{P(\bar{c}|noise)P(\theta_1,...,\theta_{L-1}|noise)} > \sqrt{K_L} \tag{13}$$

In other words, this equation requires the product of the likelihood ratios for the contrast and shape to exceed the square root of the search space size. It can thus be rewritten as:

$$\mathit{LikelihoodRatio}(\bar{c}) \cdot \mathit{LikelihoodRatio}(\theta) > \sqrt{K_L} \tag{14}$$

In Section 3.3 we discuss how *LikelihoodRatio*($\bar{c}$) and the corresponding contrast likelihood functions for the edge and noise are computed. We develop from the criterion $\mathit{LikelihoodRatio}(\bar{c}) > \sqrt{K_L}$ a closed form threshold of the mean contrast. Then, in Section 3.4, we discuss how *LikelihoodRatio*(θ) and the corresponding shape likelihood functions are computed. We develop from the criterion $\mathit{LikelihoodRatio}(\theta) > 1$ a closed form threshold of the turn angles. These two closed-form thresholds form two scores by taking the distances of the observations from the thresholds. Finally, in Section 3.5 we implement the multiplication criterion in (14) by averaging the contrast score and the shape score, this averaging is used as our *EdgeScore*.

3.3 Contrast Likelihood Functions

To compute *LikelihoodRatio*($\bar{c}$) we should study how the mean contrast $\bar{c}$ (12) is distributed when the filter is applied to edges and when the filter is applied to noise or background. We first discuss the application of the filter to noise.

Consider a pure noisy image $n(x, y)$ where every pixel is i.i.d. normal, i.e., $n(x, y) \sim \mathcal{N}(0, \sigma_n^2)$ for every pixel. Then every curve in $n(x, y)$ is a noise curve. A filter of length L and width w that is applied to that image sees wL pixels of pure noise. The statistic $\bar{c}$ averages these pixels and we use the properties of the normal distribution to derive its variance. Denote by σ_L^2 the variance of the mean contrast of noise edge of length L and width w, then: $\sigma_L^2 = \frac{\sigma_n^2}{wL}$. Therefore, the likelihood function of the mean contrast given noise is normally distributed with zero mean and σ_L^2 variance:

$$P(\bar{c}|\mathit{noise}) \sim \mathcal{N}(0, \sigma_L^2) \tag{15}$$

The edge likelihood function $P(\bar{c}|\mathit{edge})$ can take several forms for different types of inputs. For example, we expect this function to be different for natural images and medical images. In Section 4 we introduce an experiment that computes this likelihood function for natural images from the Berkeley Segmentation Dataset (BSD). In general, we expect this function to be symmetric and to have a negligible probability in an area around the zero contrast. The experiment in section 4 shows that in natural images this area is indeed very small.

We assume that the mean contrast is distributed normally with zero mean and higher standard deviation than the noise, i.e. $P(\bar{c}|\mathit{edge}) \sim N(0, \sigma_s^2)$ where $\sigma_s > \sigma_n$. This makes sense because the contrast is isotropic and because the edge contrast is expected to be stronger than the noise for the edge to be detectable. This assumption ignores the absence of edges with near-zero contrast. However, this interval is very narrow and does not significantly affect $P(\bar{c}|\mathit{edge})$ in images. Then, if we consider only the contrast cue and neglect the shape cue, we get the following inequality:

$$\mathit{LikelihoodRatio}(\bar{c}) > \sqrt{K_L} \tag{16}$$

Now we can derive a closed-form expression for the threshold T for the average contrast by comparing two normal distributions. See Appendix A.1 for details about the derivation of the contrast threshold. Denote by SNR^* the ratio between the standard deviations of the edge contrast and the noise, i.e. $SNR^* = \frac{\sigma_s}{\sigma_n}$. Then, the following threshold for the mean contrast is equivalent to the criterion in (16):

$$|\bar{c}| > T(SNR^*, \sigma_n, w, L) = \sigma_n \sqrt{\frac{\ln(K_L \cdot wL \cdot SNR^{*^2})}{wL}} \tag{17}$$

If we consider the classic definition of the SNR (7) as the ratio between the mean of absolute contrast of an edge (μ_s) to the noise standard deviation (σ_n), we can derive the relation between the classic SNR to SNR^*. For a random variable $X \sim \mathcal{N}(0, \sigma)$ it holds that $E[|X|] = \sigma\sqrt{\frac{2}{\pi}}$. Then using that the signal contrast is distributed normally, the following relation holds:

$$SNR^* = \frac{\sigma_s}{\sigma_n} = \sqrt{\frac{\pi}{2}}\frac{\mu_s}{\sigma_n} = \sqrt{\frac{\pi}{2}}SNR \tag{18}$$

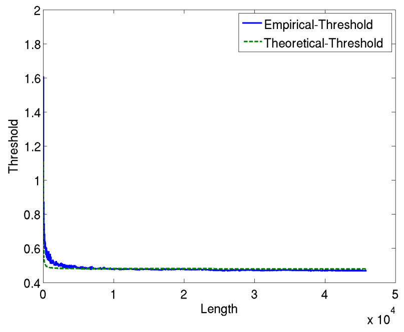

Fig. 6. Empirical versus theoretical contrast threshold of Triangle-Partition-Tree. Any curve of length L with a higher mean contrast than Threshold(L) is suspected as a curved edge.

Note that [1] and [2] developed a similar threshold for the average contrast. They designed their threshold to reject false positive edge detections. Their threshold did not take into consideration the signal distribution and produced similar results to the above threshold when substituting $SNR^* = 1$. This low SNR^* can be seen as a faint edge assumption. The following is their threshold criterion:

$$|\bar{c}| > T(\sigma_n, w, L) = \sigma_n \sqrt{\frac{\ln(K_L^2)}{wL}} \tag{19}$$

Note that this threshold is similar but higher than our threshold. Using that $K_L > N > wL$:

$$\frac{T(SNR^* = 1, \sigma_n, w, L)}{T(\sigma_n, w, L)} = \sqrt{\frac{\ln(K_L \cdot wL)}{\ln(K_L^2)}} = \sqrt{0.5 + 0.5\frac{\ln(wL)}{\ln(K_L)}} < 1 \tag{20}$$

Consequently, our approach on the one hand can detect fainter edges but will detect more false positive edges on the other hand.

In equation (4) we observed that in our Triangle-Partition-Tree algorithm, it holds that $K_L = 6N \cdot 2^{2L/3}$. Then, if we substitute this expression we infer that in the TPT algorithm, the contrast thresholds are as follows:

$$|\bar{c}|/\sigma_n > T(SNR^* = 1, \sigma_n, w, L)/\sigma_n \approx \sqrt{\frac{\ln(6N \cdot 2^{2L/3} \cdot wL)}{wL}} = \sqrt{\frac{\ln(6N \cdot wL)}{wL} + T_\infty^2} \tag{21}$$

If we denote by T_∞ the threshold where L goes to infinity, then in TPT it holds that $T_\infty > 0$. The conclusion is that our method for curved edge detection, due to its exponential search space, can detect only edges with mean contrast greater than a constant equals T_∞. Our experiments show that for our method, $T_\infty = 0.48$, i.e., curves with $SNR < 0.48$ cannot be detected by TPT. See Figure 6 for a comparison between the theoretical the empirical threshold of our method.

3.4 Shape Likelihood Functions

Next, we would like to incorporate a shape before our decision function. Our method assumes that curved edges tend to be smooth. In other words, their elastic energy level is low, or their probability as a random walk path is high. Below we introduce a shape score designed to prefer smooth curves over non-smooth ones.

In Section 3.1 we presented a curve descriptor ($\mathcal{CD}$) that describes the pixel intensities along an edge and the edge shape. Equation (14) describes an inequality criterion for a curve descriptor to represent an edge that involves both cues: contrast and shape. To develop a shape score that conforms with the previous methodology, we require that the shape likelihood ratio of a curve descriptor that represents an edge should be greater than one:

$$LikelihoodRatio(\theta) > 1 \tag{22}$$

The product of the contrast *LikelihoodRatio* and shape *LikelihoodRatio* in (14) will be implemented heuristically by averaging between closed form scores of both cues. As was done for the contrast, we model the likelihood function of the shape conditioned on edge event and conditioned on noise event. For the noise, we assume that a noise edge does not "prefer" any shape. Therefore, the likelihood of each turn is distributed uniformly:

$$\forall j : P(\theta_j|noise) \sim \mathcal{U}(-\pi/2, \pi/2) \tag{23}$$

Thus, for every turn θ_j we set $P(\theta_j|noise) = \frac{1}{\pi}$. In addition, we assume that shape turns are statistically independent. Thus, we can derive an expression for the shape likelihood function of noise:

$$P(\theta|noise) = P(\theta_1, ...\theta_{L-1}|noise) = \prod_j P(\theta_j|noise) = \frac{1}{\pi^{L-1}} \tag{24}$$

For the edge shape likelihood, we assume we prefer each turn independently to be as close as possible to a zero. A distribution that achieves this property is the normal distribution centered around zero:

$$\forall j : P(\theta_j|edge) \sim \mathcal{N}(0, \sigma_\theta^2) \tag{25}$$

Under the above assumption, we can derive a closed-form inequality criterion for the shape of $\mathcal{CD}$ to belong to an edge:

$$\sqrt{\bar{\theta^2}} < T(\sigma_\theta) = \sigma_\theta\sqrt{2\ln\pi - \ln(2\pi\sigma_\theta^2)} \tag{26}$$

The statistic $\bar{\theta^2}$ captures the smoothness of the curve. It is defined by:

$$\bar{\theta^2} = \frac{1}{L-1}\sum_j \theta_j^2 \tag{27}$$

For the derivation of the shape threshold (26) see Appendix A.2.

3.5 Edge Score

So far we have derived closed-form inequality criteria, one for the contrast (17) and another for the shape (26). We convert these inequalities to scoring functions by subtracting the two sides of each inequality. Now we can combine these two functions into a single average score in the following way:

$$EdgeScore = \beta \cdot (|\bar{c}| - T(snr, \sigma_n, w, L)) + (1-\beta) \cdot (T(\sigma_\theta) - \sqrt{\bar{\theta^2}}) \tag{28}$$

with $\beta \in [0, 1]$. In our implementation, we represented the turn angles in radians, the mean contrast was extracted by a normalized filter from a grayscale image with intensities between 0 to 1 (in double precision). Under this choice of units, the parameter β that produces the best results in our experiments is $\beta = 0.9455$.

We used this *EdgeScore* definition as the scoring function for the Triangle-Partition-Tree algorithm (Section 2). In this algorithm, a curve has declared an edge if its *EdgeScore* is greater than zero. Throughout the development of TPT, we assumed that our scoring function could be used for the multiscale approach. In other words, once we compute the score of two sub-curves that share a single endpoint, we can compute the score of the curve induced by their stitching in $O(1)$ computational complexity. The above *EdgeScore* fulfills this criterion by storing only $O(1)$ data for each curve. In conclusion, the *EdgeScore* that we developed here fits the requirements of the efficient TPT algorithm. Both sections together form an efficient Bayesian detection of faint curved edges in noisy images.

4 Contrast Likelihood Functions in Natural Images

In Section 3.3 we introduced a theoretical approach to calculate *LikelihoodRatio*$(\bar{c})$ given an edge mean contrast $\bar{c}$. In this section, we address the same problem empirically by measuring distributions in natural images. Note that we do not use this measuring in our experiments, however, it contrasts the assumptions we made for the developing of the theoretical approach. Denote by c a single local contrast sample, in contrast to $\bar{c}$ which is a mean contrast along a curve. We design an experiment for estimating the density functions $P(c|edge)$ and $P(c|noise)$. We ran the experiment on the Berkely Segmentation Dataset (BSD), which contains 500 natural images with boundaries marked by human observers.

To estimate $P(c|edge)$ in the BSD, we need to detect the maximal edge response in any edge location. For that purpose, we used four Sobel filters for each direction in an oriented 8-connected lattice. For each pixel marked as the boundary in the data set, we stored the maximal signed response obtained by the four filters. Eventually, we collected a large set of edge contrasts that was produced by the unknown density function $P(c|edge)$ in natural images.

To estimate $P(c|noise)$ in natural images, we think of noise as background. We took the same maximum of the Sobel filter responses for all of the pixels that are not marked as boundaries in BSD. Then we collected a large set of contrast samples from the density function $P(c|noise)$. The histogram of these values appears to be very similar to the known density of derivatives in natural images. This density function, known as the sparse prior, was introduced in several studies such as [15] and [22]. Note that the density function of derivatives in natural images was calculated on all pixels, edges, and non-edges, while we eliminated the pixels that were marked as boundaries in the BSD. Both cases produced similar results because the portion of the marked boundaries is very small and because there are edges in the BSD that are not marked as boundaries. Therefore, we modeled $P(c|noise)$ in the same parametric model that was used for all derivatives in natural images, and known as the sparse prior:

$$P(c|noise; s, p) = \frac{e^{-|c/s|^p}}{Z(s,p)} \tag{29}$$

with the partition function $Z(s,p) = 2\frac{s}{p}\Gamma(\frac{1}{p})$, where Γ denotes the Gamma distribution. As mentioned in [15] and [22]: p is typically within the range $[0.5, 0.8]$, and s is typically within the range $[0.03, 0.1]$.

To model the edge contrast density $P(\bar{c}|edge)$, we found that a similar model can be used with the additional modification of assigning zero probability to near-zero contrasts. Denote by $\infty_{|c|\geq\alpha}$ the indicator function that equals 1 if $|c| > \alpha$, and 0 otherwise. We introduce the following parametric model:

$$P(c|edge; s, p, \alpha) = \frac{e^{-|c/s|^p}}{Z'(s,p)} \cdot \infty_{|c|\geq\alpha} \tag{30}$$

The new normalization function can be approximated by: $Z'(s,p) \approx Z(s,p) - \alpha(1 + e^{-(\alpha/s)^p})$, see Appendix A.3 for details on this approximation.

In our experiment, we fixed α and estimated the parameters p and s with maximum likelihood. We obtained the following estimates: $p = 1, s = 0.33, \alpha = 0.02$.

We repeated the same experiment with pixels that are not marked as boundaries in the BSD and obtained the following parameters: $p = 0.7, s = 0.07, \alpha = 0$. Note that the results of this experiment further confirm that the sparse prior correctly reflects the probability of a single contrast given noise.

In conclusion, we found that the distribution of derivatives across edges is wider (higher s parameter) and not as sharp (higher p parameter) as the derivatives at all pixels. Figure 7 shows the result of this experiment.

5 Experiments

To evaluate our TPT-based edge detection algorithm we applied it to both simulations and real images captured under poor imaging conditions. We used simulation data from [2], which includes 63 images of binary patterns contaminated by additive normal i.i.d noise. The images are of size 129×129 pixels each. The 63 images include three subsets of images, each includes 21 images. The first set of images shows a rectangle in one of three sizes. The second set includes images of sine waves of three different widths. The third set shows concentric circles of three different widths. Each subgroup of seven images of every different pattern contains the same pattern under different SNRs. The noise in all images remains at the same level of 0.1 standard deviation, while the contrast of the pattern changes from 0.08 to 0.2 in 0.02 intervals. Therefore, each subgroup of 7 images is tested under the following SNR levels: 0.8, 1, 1.2, 1.4, 1.6, 1.8, 2. Figure 8 shows the results obtained with these algorithms on several images from the simulations. The figure also shows the results obtained with our algorithm when the shape prior is not utilized ($\beta = 1$ in *EdgeScore* (28)). In addition, it shows the results obtained by several existing algorithms, including the faint curve detection with a Quad Pyramid [2], faint straight line detection [1], gPb [9], Canny [3]

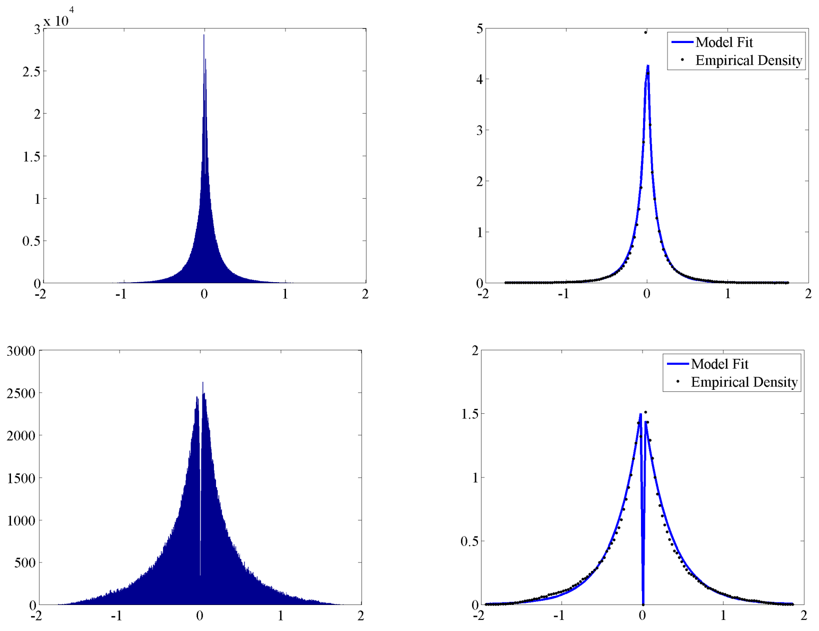

Fig. 7. Top row: A histogram plot of the maximal derivative measured empirically for non-boundary pixels of the BSD-500 images (left), and a maximum likelihood fit for $P(c|noise)$ (right). Bottom row: A histogram plot of the maximal derivative measured empirically for the boundary pixels (left). Note that in this histogram there is a cavity of low probabilities near zero. Right: A maximum likelihood fit for $P(c|edge)$.

and Sobel [4]. It can be readily seen that our method manages to detect the faint edges with only a few numbers of false positives detection.

To evaluate the results we use the F-measure, $F = \frac{2PR}{P+R}$ where P denotes precision and R denotes recall. Figure 9 shows the F-measures obtained under different SNRs. The F-measure achieved by our method increases with the SNR, yet it manages to obtain superior results relative to the local approaches (Sobel, Canny, gPb) at low SNRs (0.8 and 1). Figure 10 shows the F-measure scores for each tested algorithm. Our algorithm produced the best result in terms of average F-measure. Moreover, we see that the use of the shape score improves the F-measure, especially in the case of the square patterns.

We implemented the Triangle-Partition-Tree algorithm in Matlab. Our implementation performs edge detection in the actual run time of approximately 15 seconds on the simulations images of size $N = 129^2$ on an Intel i7 3.4 GHz, 8GB RAM machine. The runtime obtained with our implementation is consistent with the theoretical time complexity of $O(N^{1.5})$, see Figure 10.

In conclusion, our algorithm achieves high-quality results in terms of F-Measure, relative to all the other tested algorithms. It manages to find the curved edges with very few false positives. The shape cue improves the F-measure results by roughly 2% and it is proved useful for edge detection.

We further tested our implementation on real images. Figure 11 and 12 shows the results of satellite and medical images. We managed to detect thin and faint edges with different curved shapes. Figure 13 shows some results on the BSD500. Note that our algorithm is designed for edge detection while the BSD ground truth reflects the desired segmentation results. However, we can see that our method manages to produce good visual results of edge detection for the input images of BSD.

6 Conclusions

We have studied the problem of faintly curved edge detection in noisy images. We have introduced an efficient algorithm that improves the complexity of the previous algorithm and achieves better detection quality. Our results demonstrate that the detection of faint, curved edges can be done in practical time. Moreover, we have introduced a formalism of Bayesian edge detection that considers a combination of contrast and shape cues. Our faint edge detection quality compares favorably with all the algorithms we have checked, and it shows that the smooth shape prior improves edge detection results. Finally, we

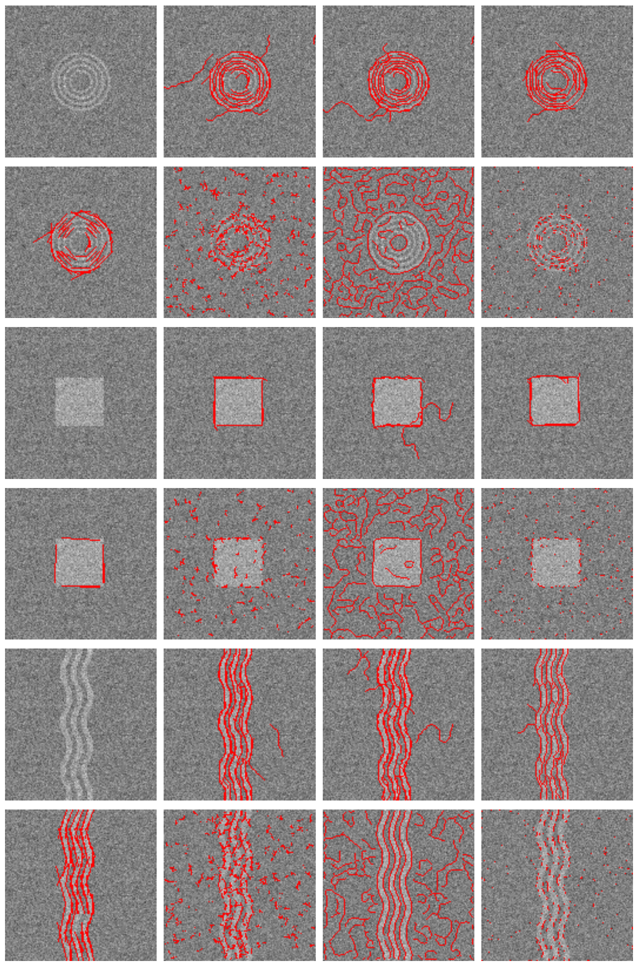

Fig. 8. Simulation examples: Three of the 63 simulation images are shown, each includes a simple binary pattern contaminated by significant noise (SNR 1.8). Each pair of rows shows the result of applying various edge detection algorithms to one of the noisy images. From left to right: (top row) input image, the result obtained with our algorithm, result obtained with our algorithm with no shape score, result obtained with the Quad-Pyramid, (bottom row) result obtained with the straight faint edge detector, gPb, Canny and Sobel.

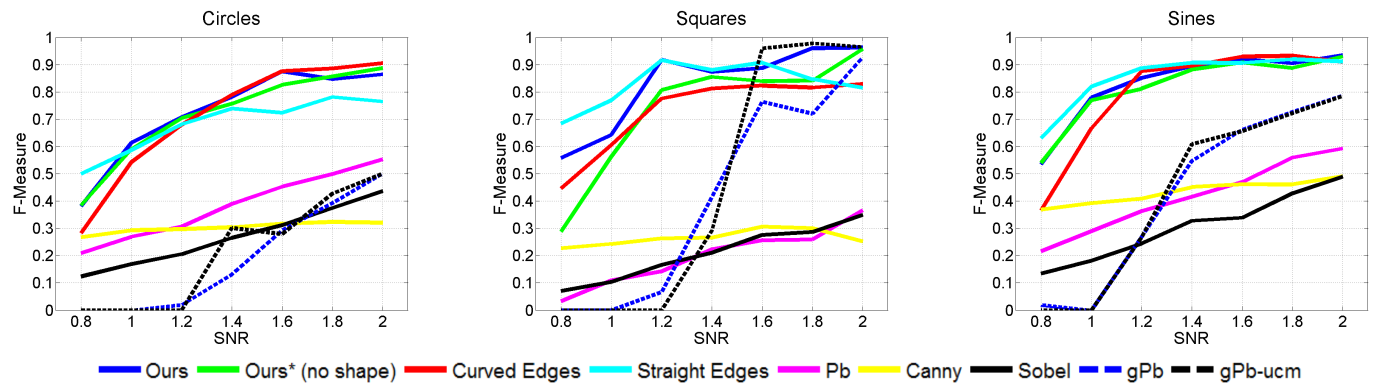

Fig. 9. Simulation results: For each tested edge detector we show the F-Measure obtained as a function of SNR.

Algorithm	Circles	Squares	Sines	Average
Ours	**0.72**	**0.83**	**0.83**	**0.8**
Lines	0.68	**0.83**	**0.86**	**0.79**
Ours*	**0.72**	0.74	0.82	0.76
Curves	0.71	0.73	0.8	0.75
gPb-ucm	0.22	0.46	0.43	0.37
gPb	0.19	0.41	0.43	0.35
Pb	0.38	0.2	0.42	0.33
Canny	0.3	0.27	0.43	0.33
Sobel	0.27	0.21	0.31	0.26

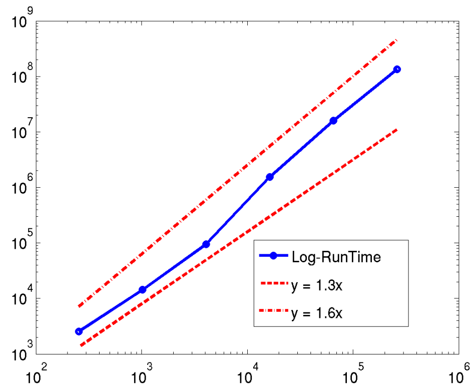

Fig. 10. Left: Average F-Measure scores obtained with the various algorithms on our simulation data set. The best two scores in every column are marked in boldface. The shape cue improves the accuracy of our algorithm, particularly in the case of the square images. Right: CPU run-time of our code as a function of the number of pixels in the input image (log-log plot). Our runtime graph is bounded between the $T_1(N) = O(N^{1.3})$ and $T_2(N) = O(N^{1.6})$.

studied the distribution of contrasts in edge pixels in natural images. This distribution can be useful as a prior for natural images.

7 Future Work

This work leaves several interesting open questions that we would like to further investigate.

In future work, we hope to understand the trade-off between detection quality and time complexity in the gap between $O(N^{1.5})$ to $O(N \log_2(N))$. We believe that we can reduce the algorithm complexity by storing only a subset of the edge data of every tile of every level. This subset will contain the edges that achieved the maximal *EdgeScore* out of all edges in the tile. An additional direction for reducing the complexity is using edge sparsity in images. If we have an efficient method for classifying tiles as ones that contain edges and ones that do not, we should be able to reduce the total amount of work on our solution. Moreover, we can take advantage of edge sparsity by using sparse data structures.

This paper focuses on detecting edges in 2D images. We hope to extend this approach to detect edges in 3D images. Such a result may be useful and practical in several application domains such as medical imaging.

Curves in images depict useful information about the underlying scene. We would like to explore how this information can contribute to other computer vision problems. For example, image registration methods have several limitations that may be overcome by using correspondence constraints between curves in the aligned images. These limitations may include non-rigid transformation between the input images, or even, no shared features between images as a result of a dramatic change in viewpoint.

As we have discussed in the paper, faintly curved edge detection is a complicated problem mainly because it requires the algorithm to perform a search in an exponentially large search space. By imposing several assumptions, the multiscale TPT

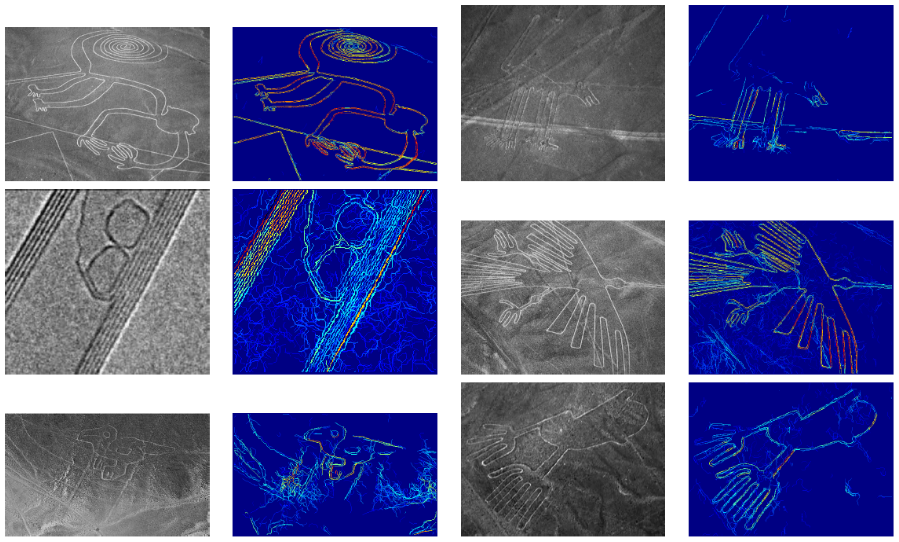

Fig. 11. From left to right: A satellite image with faint curved fibers along with our detection result. A retina image and our detection result.

algorithm was designed to scan this space in practical polynomial time. We believe other problems in computer vision too are considered hard because, by their nature, they involve search in an exponential search space. Such problems may include image segmentation or object detection. We hope to investigate in the future whether such problems can be solved with a similar approach, i.e., by reducing the exponential complexity to a practical polynomial while maintaining the quality of the results. To that end, we may want to generalize our method to handle mode general graphs or grids, and then adapt the generalized technique to the constraints of other computer vision (and maybe even other computer science) problems.

A Appendix

A.1 Derivation of the Contrast Threshold

We want to determine an explicit expression for $LikelihoodRatio(\bar{c}) > K_L$ under the assumptions: $P(\bar{c}|edge) \sim \mathcal{N}(0, \sigma_s^2)$ and $P(\bar{c}|noise) \sim \mathcal{N}(0, \sigma_L^2)$ where $\sigma_L^2 = \frac{\sigma_n^2}{wL}$ and $\sigma_s > \sigma_L$.

We derive this expression using the following derivation

$$LikelihoodRatio(\bar{c}) > \sqrt{K_L}$$

Substituting the $LikelihoodRatio$ definition

$$\frac{P(\bar{c}|edge)}{P(\bar{c}|noise)} > \sqrt{K_L}$$

Taking the ln of both sides

$$\ln(P(\bar{c}|edge)) - \ln(P(\bar{c}|noise)) > \ln(\sqrt{K_L})$$

Substituting for the density functions

$$(-0.5\ln(2\pi\sigma_s^2) - \frac{\bar{c}^2}{2\sigma_s^2}) - (-0.5\ln(2\pi\sigma_L^2) - \frac{\bar{c}^2}{2\sigma_L^2}) > 0.5\ln(K_L)$$

Therefore,

$$\bar{c}^2(\frac{1}{2\sigma_L^2} - \frac{1}{2\sigma_s^2}) > 0.5\ln(K_L) + 0.5(\ln(2\pi\sigma_s^2) - \ln(2\pi\sigma_L^2))$$

Implying that

$$\bar{c}^2 > \frac{\sigma_s^2\sigma_L^2}{\sigma_s^2 - \sigma_L^2}\ln(K_L\frac{\sigma_L^2}{\sigma_s^2}) = \frac{\sigma_s^2}{\sigma_s^2/\sigma_L^2 - 1}\ln(K_L\frac{\sigma_L^2}{\sigma_s^2}) = \frac{\sigma_s^2}{wL\sigma_s^2/\sigma_n^2 - 1}\ln(K_L wL\frac{\sigma_n^2}{\sigma_s^2})$$

Ignoring the negligible -1 in the dominator:

$$\bar{c}^2 > \sigma_n^2 \frac{\ln(K_L \cdot wL \cdot \frac{\sigma_n^2}{\sigma_s^2})}{wL}$$

Taking the square root of both sides:

$$|\bar{c}| > \sigma_n \sqrt{\frac{\ln(K_L \cdot wL \cdot \frac{\sigma_n^2}{\sigma_s^2})}{wL}}$$

The standard deviation ratio $\frac{\sigma_s}{\sigma_n}$ is tightly related to the SNR of the image. If we consider the definition of SNR as the ratio between the mean edge contrast (μ_s) in the image to the noise level then $SNR = \frac{\mu_s}{\sigma_n}$. Since we assumed that $P(\bar{c}|edge) \sim \mathcal{N}(0, \sigma_s^2)$, then the mean edge contrast can be calculated by the mean of the absolute value of a normal distribution with zero means as follows:

$$\mu = \sigma_s \sqrt{\frac{2}{\pi}}$$

Therefore, the SNR according to our assumption is:

$$SNR = \sqrt{\frac{2}{\pi}} \frac{\sigma_s}{\sigma_n}$$

Denote by SNR^* the standard deviation ratio $\frac{\sigma_s}{\sigma_n}$ then $SNR^* = \sqrt{\frac{\pi}{2}} SNR$. Now, the threshold can by rewritten as

$$|\bar{c}| > \sigma_n \sqrt{\frac{\ln(K_L \cdot wL \cdot SNR^{*2})}{wL}} = \sigma_n \sqrt{\frac{\ln(K_L \cdot wL \cdot \frac{\pi}{2} SNR^2)}{wL}}$$

A.2 Derivation of the Shape Threshold

We want to solve the equation $LikelihoodRatio(\theta) > K_L$ under the assumption of: $\forall \theta_j : P(\theta_j|edge) \sim \mathcal{N}(0, \sigma_\theta^2), P(\theta_j|noise) \sim \mathcal{U}(-\pi/2, \pi/2)$. In addition, we assume that each turn is statistically independent.

Denote $L' = L - 1$. We derive this expression using the following derivation

$$LikelihoodRatio(\theta) > 1$$

Using the definition of the $LikelihoodRatio$

$$P(\theta_1, ...\theta_{L'}|edge) > P(\theta_1, ...\theta_{L'}|noise)$$

Taking the ln() of both sides:

$$\sum_{j=1}^{j=L'} \ln(P(\theta_j|signal)) > \sum_{j=1}^{j=L'} \ln(P(\theta_j|noise))$$

Substituting for the two density functions

$$\sum_{j=1}^{L'} \ln(\frac{1}{\sqrt{2\pi\sigma_\theta^2}} \exp(-\frac{\theta_j^2}{2\sigma_\theta^2})) > \sum_{j=1}^{L'} \ln(\frac{1}{\pi})$$

Therefore,

$$-0.5L' \ln(2\pi\sigma_\theta^2) - \sum_{j=1}^{L'} \frac{\theta_j^2}{2\sigma_\theta^2} > -L' \ln(\pi)$$

Implying that

$$\frac{\bar{\theta^2}}{2\sigma_\theta^2} < \ln(\pi) - 0.5 \ln(2\pi\sigma_\theta^2)$$

where $\bar{\theta^2} = \frac{1}{L'} \sum_{j=1}^{L'} \theta_j^2$. Therefore,

$$\bar{\theta^2} < \sigma_\theta^2 (2 \ln(\pi) - \ln(2\pi\sigma_\theta^2))$$

Taking the square root of both sides:

$$\sqrt{\bar{\theta^2}} < \sigma_\theta \sqrt{2\ln(\pi) - \ln(2\pi\sigma_\theta^2)}$$

So far we derived a closed-form expression for the shape threshold. Note that this shape criterion may hold only if the right-hand side is greater than zero:

$$0 < \sigma_\theta \sqrt{2\ln(\pi) - \ln(2\pi\sigma_\theta^2)}$$

Implying that

$$2\ln(\pi) > \ln(2\pi\sigma_\theta^2)$$

Take 2 to the power of the inequality sides:

$$\pi^2 > 2\pi\sigma_\theta^2$$

Therefore, it is feasible to be below the shape threshold only if

$$\sigma_\theta < \sqrt{\pi/2}$$

It means that the shape standard deviation should be an angle that is less than $\approx 72°$.

A.3 The Likelihood Function for the Density of Natural Images Derivatives

The density function of all derivatives in natural images is modeled by

$$P(c|noise; s, p) = \frac{e^{-|c/s|^p}}{Z(s,p)}$$

The partition function is known to be

$$Z(s,p) = 2\frac{s}{p}\Gamma(\frac{1}{p})$$

The empirical density function of edge derivatives in natural images is modeled by the same probabilistic function with the single modification that assigns a zero probability at an interval around the zero contrast, i.e.,

$$P(c|edge; s, p, \alpha) = \frac{e^{-|c/s|^p}}{Z'(s,p)} \cdot \infty_{|c|\geq\alpha}$$

To compute the new normalization value, we approximate the eliminated part of the Gaussian function by the sum of the areas of a rectangle and triangle as follows

$$Z'(s,p) = Z(s,p) - RectBottom - TriangleTop$$

RectBottom is a rectangle whose vertices are $(\alpha, 0), (-\alpha, 0), (\alpha, P(\alpha)), (-\alpha, P(-\alpha))$. $TraingleTop$ is a triangle whose vertices are $(0, P(0)), (\alpha, P(\alpha)), (-\alpha, P(-\alpha))$ According to these definitions

$$Z'(s,p) = Z(s,p) - (2\alpha \cdot P(c=\alpha|noise)) - (P(c=0|noise) - P(c=\alpha|noise) \cdot \alpha)$$

from which we obtain

$$Z'(s,p) = Z(s,p) - \alpha(P(c=0|noise) + P(c=\alpha|noise))$$

Finally, substituting the probability densities

$$Z'(s,p) = Z(s,p) - \alpha(1 + e^{-(\alpha/s)^p})$$

A.4 Complexity of Triangle Partition Tree Algorithm

In Section 2.3 we proved that the computational complexity of our algorithm is $O(N^{1.5})$ by the master theorem rule. Here we will prove it again directly.

In Section 2.5 we show that the number of curves (pairs of endpoints) to consider at every level j is roughly: $6N$. In Section 2.1 we express the triangle tile dimensions as a function of the level index j. The amount of computational work for computing the edge response of each curve at level j is the same as the edge length of a triangle tile at level $j+1$. Therefore, the computational complexity per curve calculation in level j is roughly $n/2^{0.5(j+1)}$.

Denote by $T(N)$ the computational complexity of our algorithm running on a square image of N pixels. The total complexity is the sum of the work done in each level j. Then, in each level, we count the number of curves, $6N$, times the work done per curve, $n/2^{0.5(j+1)}$. With these derivations, we can compute the time complexity directly

$$T(N) = \sum_{j=0}^{\log_2(N)} 6N \cdot n/2^{0.5(j+1)}$$

The above equation sums the work in all the levels as described in Section 2.3. Since this is the sum of a geometric series:

$$T(N) \leq \frac{6N^{1.5}}{\sqrt{2}} \cdot \frac{1}{1-2^{-0.5}}$$

and therefore,

$$T(N) \approx 14.5N^{1.5}$$

A.5 Search Space Calculation

In section 2.5 we computed K_L, the search space size of each curve of length L. The main part of the derivation was the computation of the number of possible curves that can go through fixed given endpoints in TPT. For that purpose, we defined the function $K(j)$ as the search space of curves given endpoints in level j in TPT.

According to the TPT algorithm definition, the size of the set of possible curves given endpoints ($K(j)$) equals to the size of the splitting edge of a tile in level j ($n/2^{0.5(j+1)}$), times the search space size of the sub-curves of level $j+1$ ($K(j+1)$). Since a curve in level j, is made out of stitching of two sub-curves of level $j+1$, we take $K(j)^2$ in the calculation as follows:

$$K(j) = K(j+1)^2 \cdot n/2^{0.5(j+1)}$$

Our goal is to derive an expression for $K(j)$ from this recursive formula. Firstly, we take the log of the equations:

$$\log_2 K(j) = 2\log_2 K(j+1) + \log_2(n/2^{0.5(j+1)}) = 2\log_2 K(j+1) + 0.5\log_2(N/2^{j+1})$$

Now, substitute $l = N/2^{j+1}$ and denote $LK(l)$ by the function $\log_2 K(j)$:

$$LK(l) = 2LK(l/2) + 0.5\log_2 l$$

According the the master theorem [21] and empirical evaluation of $LK()$:

$$LK(l) \approx 2l$$

Substitute back to K(J):

$$\log_2 K(J) = 2N/2^{j+1}$$

Taking 2 to the power of the equation:

$$K(J) = 2^{2N/2^j}$$

We assume that the average length of a curve of level j is $L = \alpha 2^{\log_2(N)-j}$, where α denotes the average length of a straight edge in the bottom level in the TPT, which is ≈ 3. Then, the search space of a curve of length L given fixed endpoints is $2^{2L/\alpha} = 2^{2L/3}$.

References

1. Galun, M., Basri, R., Brandt, A.: Multiscale edge detection and fiber enhancement using differences of oriented means. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. (2007) 1–8
2. Alpert, S., Galun, M., Nadler, B., Basri, R.: Detecting faint curved edges in noisy images. In: Proceedings of the 11th European conference on Computer vision: Part IV. ECCV'10, Berlin, Heidelberg, Springer-Verlag (2010) 750–763
3. Canny, J.: A computational approach to edge detection. IEEE Trans. Pattern Anal. Mach. Intell. **8** (1986) 679–698
4. Sobel, I.: Camera models and machine perception. In: PhD thesis, Electrical Engineering Department, Stanford University. (1970)
5. Marr, D., Hildreth, E.: Theory of Edge Detection. Proceedings of the Royal Society of London. Series B, Biological Sciences **207** (1980) 187–217
6. Tomasi, C., Manduchi, R.: Bilateral filtering for gray and color images. In: Computer Vision, 1998. Sixth International Conference on. (1998) 839–846
7. Perona, P., Malik, J.: Scale-space and edge detection using anisotropic diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990) 629–639
8. Maire, M., Arbelaez, P., Fowlkes, C., Malik, J.: Using contours to detect and localize junctions in natural images. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. (2008) 1–8
9. Arbelaez, P., Maire, M., Fowlkes, C.C., Malik, J.: From contours to regions: An empirical evaluation. In: CVPR. (2009)
10. Arbelaez, P., Maire, M., Fowlkes, C., Malik, J.: Contour detection and hierarchical image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. **33** (2011) 898–916
11. Wang, P.S.P., Yang, J.: A review of wavelet-based edge detection methods. IJPRAI **26** (2012)
12. Felzenszwalb, P.F., McAllester, D.A.: The generalized a* architecture. CoRR **abs/1110.2216** (2011)
13. Brandt, A., Dym, J.: Fast calculation of multiple line integrals. SIAM J. Sci. Comput. **20** (1999) 1417–1429
14. Donoho, D.L., Huo, X., Jermyn, I., Jones, P., Lerman, G., Levi, O., Natterer, F.: Beamlets and multiscale image analysis. In: in Multiscale and Multiresolution Methods. Volume 20. (2001) 149–196
15. Simoncelli, E.P.: Bayesian denoising of visual images in the wavelet domain. In: LECTURE NOTES IN STATISTICS, Springer-Verlag (1999)
16. Konishi, S., Yuille, A.L., Coughlan, J., Zhu, S.C.: Fundamental bounds on edge detection: An information theoretic evaluation of different edge cues. In: Proc. IEEE Conf. Comput. Vision and Pattern Recognition. (1999) 573–579
17. Levin, A., Fergus, R., Durand, F., Freeman, W.T.: Deconvolution using natural image priors. ACM Trans. Graphics **26** (2007) 0–2
18. Williams, L.R., Jacobs, D.W.: Stochastic completion fields: A neural model of illusory contour shape and salience. Neural Computation **9** (1997) 837 – 858
19. Sharon, E., Brandt, A., Basri, R.: Completion energies and scale. Pattern Analysis and Machine Intelligence, IEEE Transactions on **22** (2000) 1117–1131
20. Martin, D.R., Fowlkes, C.C., Malik, J.: Learning to detect natural image boundaries using local brightness, color, and texture cues. IEEE Trans. Pattern Anal. Mach. Intell. **26** (2004) 530–549
21. Thomas H. Cormen, Charles E. Leiserson, R.L.R., Stein, C.: Introduction to algorithms, second edition. mit press and mcgraw-hill, 2001. isbn 0-262 03293-7. sections 4.3 and 4.4, pp. 73-90. (2001)
22. Weiss, Y., Freeman, W.: What makes a good model of natural images? In: Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on. (2007) 1–8

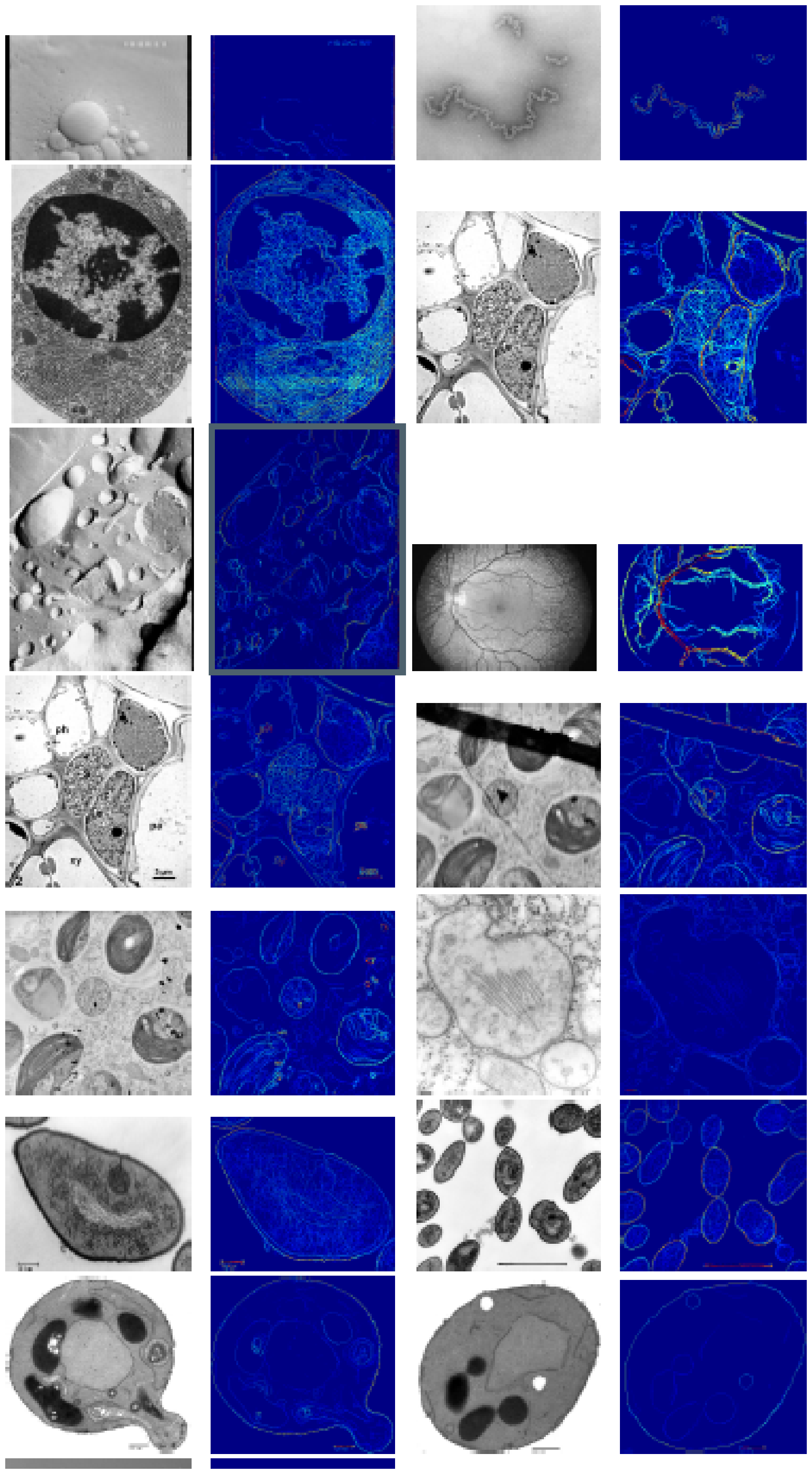

Fig. 12. From left to right: A satellite image with faint curved fibers along with our detection result. A retina image and our detection result.

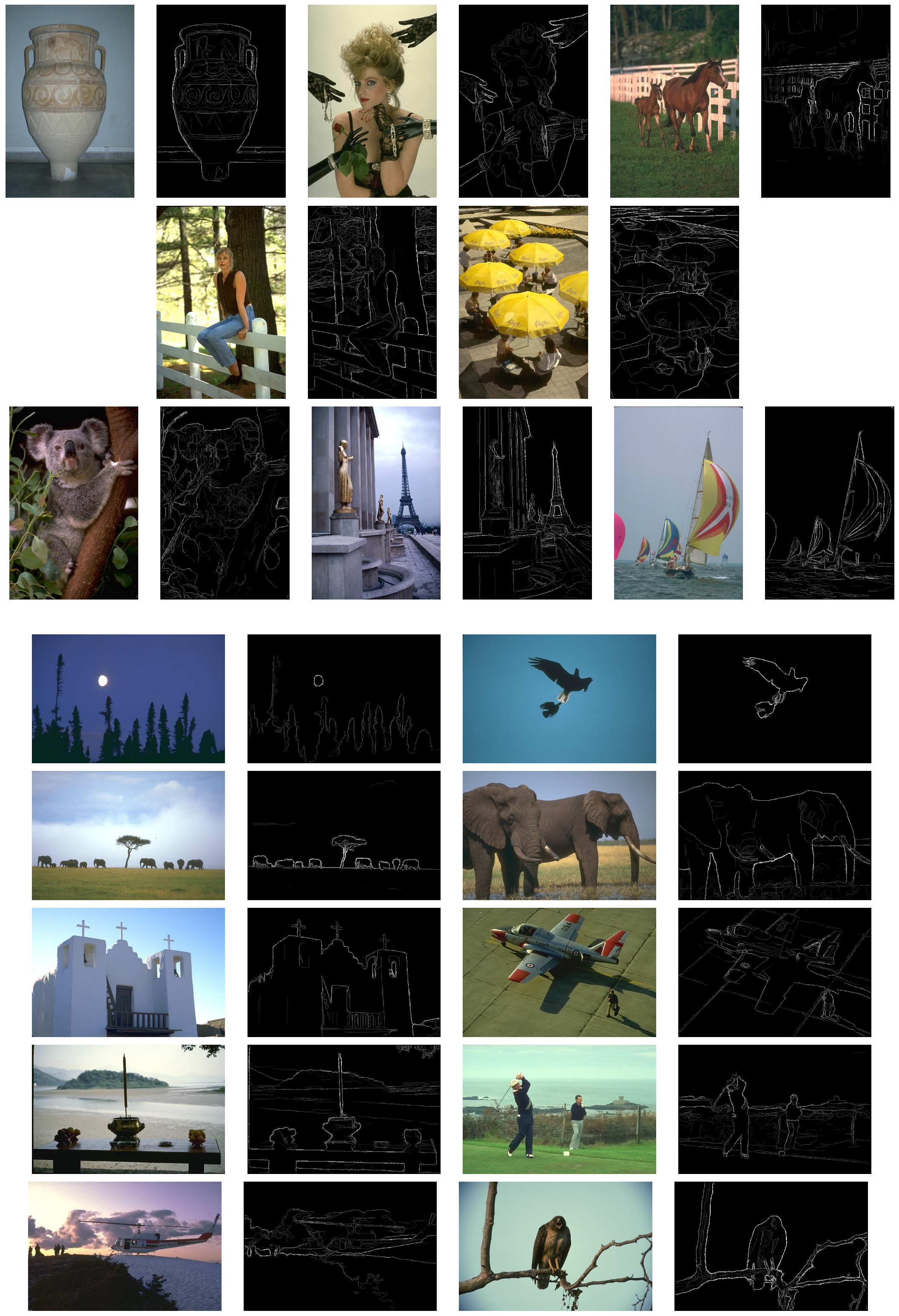

Fig. 13. Results obtained with our algorithm for the BSD500.