Semi Global Boundary Detection

Roy Josef Jevnisek, Shai Avidan

Tel-Aviv University

Abstract

Semi Global Boundary Detection (SGBD) breaks the image into scan lines in multiple orientations, segments each one independently, and combines the results into a final probabilistic 2D boundary map.

The reason we break the image into scan lines is that they are a mid-level image representation that captures more information than a local patch. Luckily, scan lines are 1D signals that can be optimally segmented using dynamic programming, and we make the assumption that the border pixels between segments are boundary pixels in the image.

This leads to a simple and efficient algorithm for boundary detection. The entire algorithm requires little parameter tuning, works well across different data sets and modalities, and produces sharp boundaries. We report results on several benchmarks that include both color and depth.

Keywords: Edge / Boundary Detection, Semi Global Matching

1. Introduction

The goal of this work is to accurately detect and localize boundaries in natural scenes using image measurements along scan lines. We use Dynamic Programming (DP) to find an optimal segmentation for each scan line and take the boundaries between these 1D segments to determine boundary pixels in the image.

Boundary pixels can be recovered either locally or globally. Edge detection algorithms are local in the sense that they determine if a pixel is a boundary pixel based on a local patch around it. Global methods, on the other hand, partition the image into regions based on global considerations. The boundary pixels then are taken to be the boundaries between regions. The two approaches can be combined with local methods providing important cues to a global algorithm.

We propose a mid-level image representation that is based on scan lines in different directions (e.g., rows, columns). Scan lines are not local, because they

extend beyond a local patch. On the other hand, they are not global, because a scan line does not span the whole image. The reason we use scan lines is that they are a 1D signal and there are efficient and optimal algorithms for segmenting them.

Interestingly, a similar approach works quite well in stereo. There, Semi Global Matching (SGM) [1] solves stereo matching by breaking the image into multiple scan lines in different directions and finding an optimal stereo matching for each scan line independently. The results from all scan lines are later combined to give the final stereo matching solution. SGM-based algorithms are consistently ranked at the top of various stereo benchmarks such as the Middlebury data set [2] and the KITTI dataset [3].

Our algorithm, termed Semi Global Boundary Detection (SGBD), breaks an image into multiple scan lines in different directions. It then segments each scan line multiple times with different number of segments per scan line and different feature channels. Each such segmentation give rise to boundary pixels and because we collect statistics of each segment within a scan line we can determine the strength of boundary pixels. The results are then aggregated to obtain a final probability map.

The proposed algorithm enjoys a number of favorable features. It requires tuning just a small number of parameters and we tune them only once on the the BSDS500 training set. Once tuned, it performs consistently well across a wide range of image data sets and imaging conditions. This includes indoor as well as outdoor images, RGB as well as depth images and clean as well as noisy images. This is an important quality of the algorithm because we do not know who is going to use it and on what type of images. Finally, the algorithm is simple to implement, quite fast in practice, and can be easily made to run in parallel on a GPU (The source code will be released).

2. Background

The literature on edge detection is vast and will not be covered here. A popular and early edge detector that is still in use today is the Canny edge detector [4] that detects a peak in gradient magnitude in the direction normal to the edge.

A trend in recent years, though, is to learn edge detectors. Martin *et al.* learn to detect natural image boundaries using local brightness, color and texture cues [5]. Dollar *et al.* use a boosted classifier to classify each pixel based on its surrounding patch [6]. Mairal *et al.* use discriminative sparse image models to learn class specific edge detectors [7]. Ren and Bo learned sparse codes of patch gradients

[8]. Recently, Dollar and Zitnick [9] used structured forest for fast edge detection. Leordeanu *et al.* [10] proposed a closed-form solution to the generalized boundary detection problem. They assume that boundaries often coincide across multiple channels and fit them with a linear model. This is reduced to solving an eigenvector problem.

These different edge detectors fit the framework of Arbelaez *et al.* [11] that also proposed a contour detection based on multiple local cues. They further showed how the output of any contour detection can be fed into a globalization framework based on spectral clustering that leads to an automatically generated hierarchical segmentation.

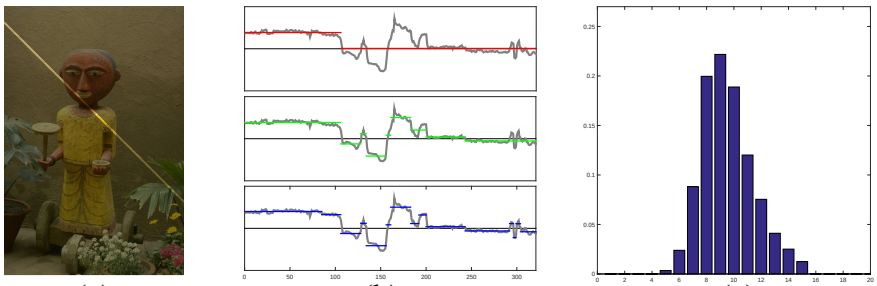
Recently, Isola *et al.* [12] introduced a crisp boundary detection algorithm that relies on pointwise mutual information. In particular, they observe that pixels belonging to the same object exhibit higher statistical dependencies than pixels belonging to different objects. These dependencies are used as local cues for spectral clustering [11].

Andres *et al.* [13] treat the problem of image segmentation as a multi-cut problem that can be solved using Integer Linear Programming (ILP) with an exponential number of constraints. However, they can find the violating constraints in polynomial time and add them iteratively. This gives a practical solution to the multicut problem which is NP-hard.

Our algorithm is inspired by the work of Hirschmuller on Semi Global Matching (SGM) [1]. He used SGM to solve stereo matching by breaking the image into multiple scanlines in different directions and finding an optimal stereo matching for each scanline independently. The results from all scanlines are later combined to give the final stereo matching solution.

SGM uses Dynamic Programming to find the optimal disparity assignment per scanline. We use Dynamic Programming as well, but use it for segmentation and not labeling. In fact, our work closely follows the original work of Bellman that used Dynamic Programming for a piecewise linear approximation of curves [14].

It is instructive to relate our work on segmenting scan lines to work on image segmentation using graph cuts. For example, Zabih and Kolmogorov [15] proposed an EM procedure for unsupervised image segmentation. The M step divides the pixels into categories (without enforcing spatial coherency) using a standard clustering technique (i.e., k-means). The E step runs graph-cut [16] to segment the image into spatially coherent regions based on the current category estimates. The advantage of such an approach is that it works on the entire image at once. On the downside, observe that it combines two non-optimal steps (graph-cuts and k-means). In fact, this is also done in the case of interactive image segmentation



(a) (b) (c)

Figure 1: **1D Segmentation:** (a) Input image with a diagonal scan line superimposed. (b) Optimal segmentation to 2 (up), 9 (middle) and 15 (bottom) segments. (c) The estimated probability $p(k)$ of the number of segments for the scan line. Observe that segmenting the scan line into 9 segments gets a much higher probability score.

[17, 18] where user scribbles provide the unary potentials, based on a GMM of color distribution of the object, for graph-cut to segment the image.

We could have replaced the graph-cut step with a variant of SGM. After all, SGM was designed to optimize the same MRF based objective function as graph-cuts. Instead, we prefer to work on one scan line at a time. This lets us find the *optimal* segmentation in one step, albeit for a single scan line. The second step, of combining multiple 1D segmentation results into a global and consistent 2D boundary map, is done in a probabilistic way, as we show later in the paper.

Superpixels [19] can also be used in conjunction with graph-cuts to segment an image. For example, SLIC (Simple Linear Iterative Clustering) [20] propose a simple, linear and iterative clustering algorithm that over-segments an image into superpixels. The superpixels are then used in conjunction with graph-cuts to obtain the final image segmentation. The final boundary map is based on the boundaries between segments.

Our work is related to that of Han *et al.* [21] who also used Dynamic Programming to segment images. However, they show their method on a single synthetic image and do not report results on any real world images, nor do they report quantitative results. We, on the other hand, show how to deal with texture, how to merge results from multiple channels, and report results on several data sets.

3. The Method

Let $\mathbf{x} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N]$ denote a scan line where $\mathbf{x}_i$ is the value of the pixel at location i. We are interested in the probability $p(\mathbf{e}_i|\mathbf{x})$ of pixel i to be a boundary

pixel between two segments, when segmenting the entire scan line $\mathbf{x}$ into k segments. Since we don't know the number of segments k ahead of time we apply the law of total probability and obtain:

$$p(\mathbf{e}_i|\mathbf{x}) = \sum_{k=1}^{K} p(\mathbf{e}_i|\mathbf{x}, k)p(k|\mathbf{x}). \quad (1)$$

In words, the probability $p(\mathbf{e}_i|\mathbf{x})$ is marginalized over all possible segmentations of $\mathbf{x}$ into k segments. Clearly, there are many ways to segment $\mathbf{x}$ into k segments and we make the simplifying assumption that only the *optimal* segmentation matters. We show how to find the optimal segmentation in section 3.1. Since k is not given ahead of time we show how to compute $p(k|\mathbf{x})$ using the principal of Minimum Description Length (MDL) in section 3.2. In section 3.3 we show how to estimate $p(\mathbf{e}_i|\mathbf{x}, k)$ which is the local probability of a pixel to be an edge pixel, given that $\mathbf{x}$ is segmented into k segments.

3.1. Optimal Segmentation of 1D Signals

We wish to find an optimal segmentation of $\mathbf{x}$ into k segments, where each segment is represented by a model (e.g., constant, linear, etc...). To do so, define an error function $err(i, j)$ that determines how well the model approximates the points $\mathbf{x}_i, \cdots, \mathbf{x}_j$. Let $OPT_k(n)$ denote the optimal error of segmenting $[\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ into k segments. We are interested in $OPT_K(N)$ and can find it using the following recursive equation:

$$OPT_k(N) = \min_{1 \le m \le N-1} \{OPT_{k-1}(m) + err(m+1, N)\}$$

With the initial condition $OPT_1(m) = err(1, m)$. This can be solved using dynamic programming. By backtracking $OPT_K(N)$ we can find the set of transitions (i.e., edge pixels) as well.

The complexity of the algorithm is $O(N^3)$ because we need to compute $err(i, j)$ for $O(N^2)$ pairs and each such operation takes $O(N)$. This can be reduced to $O(N^2)$ by using an integral scan line (i.e., a 1D integral image). For an image of size $N \times N$ pixels, the overall complexity of the algorithm is $O(NKN^2) = O(N^3)$, where N is the length of the scan line (say, width or height of the image) and K is the maximum number of segments per scan line (K is a small constant).

3.2. Estimating Number of Segments

The optimal segmentation algorithm described above assumes the number of segments, K, is known. This is not the case in practice. We cannot choose a K that minimizes the segmentation error because the error is monotonically decreasing as K increases. Instead, we use the Minimal Descriptive Length (MDL) principle that penalizes the model as the number of parameters (i.e., segments) increases. It was shown in [21] that the MDL factor for a piecewise polynomial function with degree P and white additive Gaussian noise is proportional to $KPln(N)$. Therefore, we define the MDL error for k to be:

$$MDLE(k) = OPT_k(N) + \lambda kPln(N) \tag{2}$$

where λ is a regularization weight balancing the measured error with the number of segments used. Minimizing (2) implies that there is only one correct value for k. A better alternative is to estimate the probability distribution function (pdf) of k. To do so, we normalize it into a probability:

$$p(k|\mathbf{x}) = \frac{1}{Z} \cdot exp\left\{-\frac{MDLE(k) - MDLE_{min}}{\sigma^2}\right\} \tag{3}$$

Where $MDLE_{min}$ is the minimal value of (2) over all k, Z is a factor that normalizes the distribution $p(k)$ and σ^2 is a tunable parameter that controls the spread of $p(k)$. This gives us the probability of segmenting $\mathbf{x}$ into k segments. Figure 1 illustrates this. It shows an image with a diagonal scan line superimposed on it. Next to it we show the results of optimally segmenting this 1D line into different numbers of segments. As can be seen, $p(k)$, the probability of the number of segments, for this scan line peaks at $k = 9$.

3.3. Estimating Local Edge Strength

Once we segment $\mathbf{x}$ into k segments we need to assign a probability to each boundary pixel found by that segmentation. For example, we can set

$$p(\mathbf{e}_i|\mathbf{x}, k) = \mathbb{1}_k(\mathbf{x}_i) \tag{4}$$

where $\mathbb{1}_k(\mathbf{x}_i)$ is an indicator function that equals 1 if $\mathbf{x}_i$ is a transition pixel when optimally segmenting $\mathbf{x}$ into k segments and zero otherwise. This means that all edge pixels found for a particular value k have the same probability. Clearly, this is not true in practice and we must augment $\mathbb{1}_k(\mathbf{x}_i)$ with some local edge estimation. For example, we can approximate each segment with its mean and take

the difference of the means to the left and right of the edge pixel to measure local edge strength. This did not work well in practice. The reason that the difference of means failed is that it falsely enhanced edges inside textured areas, where boundaries do not exist. In also falsely punished edges between two textures that are very different but have similar means.

This is why we introduce texture as a way to measure edge strength. Specifically, we calculate a histogram of textons for every segment. We then take the local edge strength to be the chi-square distance between the texton histograms to the right and to the left of the boundary. Formally:

$$p(\mathbf{e}_i|\mathbf{x}, k) = \mathbb{1}_k(\mathbf{x}_i)\chi^2(h_{L,\mathbf{x}_i}, h_{R,\mathbf{x}_i}) \tag{5}$$

where $h_{L,\mathbf{x}_i}$ and $h_{R,\mathbf{x}_i}$ are the texton histograms to the left and right of the edge pixel $\mathbf{x}_i$, respectively.

We calculate the texton histograms in a method similar to [11]. Given an input image, we convert it to Lab space and work on the L channel. We convolve the L channel with 16 Gaussian derivative filters, then cluster the $16D$ vectors into 32 prototype textons using k-means and assign each pixel to the closest prototype.

It is worth comparing the approach we take to the one taken by Arbelaez *et al.* [11]. There, edge pixels are first detected using a fixed scale local operator and then merged into a global solution using normalized cuts. Here, on the other hand, edge pixels are first detected in a semi-global manner. Then we estimate the local edge strength at those points. This allows us to use adaptive scale. For each edge pixel we look at the segments to its left and right, extract a histogram of textons and measure the chi-squared distance between the two histograms.

4. The Algorithm

Section 3 described the method we use for a single scanline. Here we provide the overall algorithm. It consists of the following steps:

1. Semi-Global Smoothing the image.
2. Running the 1D method on each scan line.
3. Merging the 1D results to produce a 2D edge map.

As an optional post-processing step we run the Ultrametric contour map algorithm [11]. Algorithm 1 gives the pseudo-code.

4.1. Semi-Global Smoothing

We work in Lab space and observe noticeable jpeg compression artifacts, espcially in the a and b channels, that cause multiple spurious edges. Smoothing the image with a median, Gaussian or bilateral filter did not eliminate these artifacts. Therefore, we perform *semi-global smoothing*. Figure 2 illustrates semi-global smoothing on the L, a and b channels of an image.

Semi-global smoothing relies on 1D segmentation to determine the spatial extent of smoothing. Specifically, let $\mathbf{x}_i \in s_k(j)$ denote that pixel $\mathbf{x}_i$ belongs to the j-th segment when segmenting $\mathbf{x}$ into k segments. Then the smoothed value of $\mathbf{x}_i$ is set to:

$$\mathbf{x}_i^{Smooth} = \sum_k p(k|\mathbf{x}) \cdot \mu_{s_k(j)} \tag{6}$$

Where $\mu_{s_k(j)}$ is the mean of segment $s_k(j)$. This smoothing operation is quite strong, so for the a and b channels we average it with the original channels. The L channel was not smoothed. In addition, we normalize the mean and variance of the image to a global mean and variance that was computed once on the BSDS500 training set.

Once we have smoothed the a and b channel, and have normalized all three channels for mean and variance, we run our method on each scan line in each channel in each direction.

4.2. Merging 1D Results to 2D Edge Map

After running our method on all scan lines we end up with four 2D probability maps per channel. These maps are the result of running the algorithm in different directions (i.e., rows, columns and both diagonals). Each map indicates the probability of a pixel to belong to a boundary (i.e., edge) that is in a direction perpendicular to the scan line.

We merge the edge maps in each direction by taking the maximum across the different channels, as each channel reflects different properties. This gives us four edge maps, one per direction that encodes edge information from all channels. Finally, we determine the orientation of each boundary pixel by projecting the four maps onto several orientations and picking the one that maximizes their combination.

As a final step we feed our edge maps to the Ultrametric contour map algorithm [11]. As an input to the oriented watershed transform we provide a smoothed version of the four probability maps that we calculated. In addition [11]'s code expects 8-channels, so we averaged the results of consecutive angles, 0° with 45°, 45° with 90° and so on, to extend the 4 directions into 8. A pseudo-code of the algorithm is given in Algorithm 1.

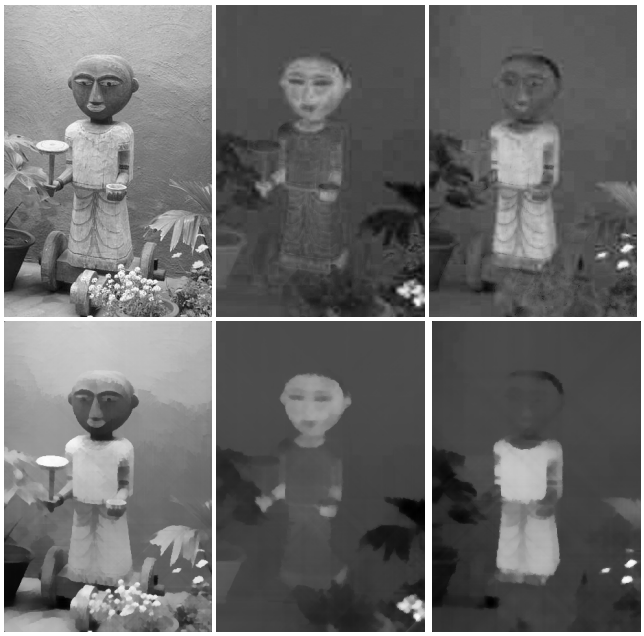


Figure 2: **Semi Global Smoothing:** top row shows L, a and b channel respectivly. Bottom row shows the semi-global smoothed versions. The jpeg artifacts, appearing mainly in the a and b channels mostly disappear while boundary information is preserved. (In practice we only smooth the a and b channels and keep the L channel intact.)

Algorithm 1

Input image: $I_{N\times M}$
Edge probability: $E_{N\times M}$

$I \leftarrow$ rgb2lab(I)
$I \leftarrow$ smooth(I) // using equation (6)
for direction $d \in \{rows, cols, diag_{45^\circ}, diag_{-45^\circ}\}$
  for channel $c \in \{l, a, b\}$:
    foreach scanline evaluate $p(k|x)$ and $p(e_i|x, k)$ // using equations (3),(5)
    $E_c^d(e_i) \leftarrow p(e_i|x, k)$ // using equation (1)
  $E_c \leftarrow min_c(E_c^d)$
for angle $a \in \{0^\circ, 45^\circ, 90^\circ, 135^\circ\}$
  $E_a \leftarrow \sum_c$(Project E_c on a)
$E(e_i) = max_a(E_a)$
$E_{ucm} \leftarrow im2ucm(E)$ (optional) // using [11]

5. Results

We report results on a number of datasets. The first is the original Berkeley Segmentation Data Set (BSDS300), the second is the new and larger BSDS500, and the third is the NYU RGBD data set. On the NYU dataset we report results on the RGB component as well as the depth component, in addition to results on RGBD. This dataset was used by Ren and Bo [8] to evaluate edge detection using RGB or RGBD images. Their algorithm requires training and they use a $60\%/40\%$ training/testing split. We don't use the training set and report results on the test set only. We also demonstrate the tolerance of the algorithm to noise, and perform an experiment on BSDS500 images that are corrupted with white Gaussian noise.

All experiments were carried out with the *same* parameters. The parameters were tuned using the 200 training set images of the BSDS500 data set. Once tuned, these parameters are fixed in all the experiments reported here. The parameters that needs to be tuned are γ, the tradeoff between segmentation error and the MDL factor, and σ, std of the Gaussian kernel for estimating $p(k)$. To do that, we started with a 3×3 grid of values and evaluated the F-measure. We found the point on the grid with maximal F-measure and repeated the process with a finer grid centered at it. We repeated the process until there was no significant increase in F-measure (3 iterations). It turned out that $\sigma^2 = 15$ was optimal across all channels and $\gamma_L = 0.1, \gamma_a = \gamma_b = 0.7, \gamma_{depth} = 0.1$ were optimal for the L, a, b and depth channels, respectively. All parameters were tuned with maximal K equals 20 (we observed that $p(k)$ for $k \geq 15$ is negligible).

5.1. Experiments

Table 1 show results of our algorithm on the BSDS300 and BSDS500 datasets and compares it to several other methods. On the BSDS300 data set, our algorithm achieves a result of $F = 0.69$. On the BSDS500 data set we achieve a score of $F = 0.72$ which is slightly worst than recent results of [11] ($F = 0.73$) and [8, 9] ($F = 0.74$). We report here the results on the test set but it is worth noting that the performance of our algorithm are the same on the train set as well. This is an indication that the algorithm is very stable and performs consistently well across a wide range of image data sets and imaging conditions, as will be described later in the section.

For the BSDS500 dataset we report the results of three variants of the algorithm. The first variant, termed SGBD-naive, only uses equation 4 and does not

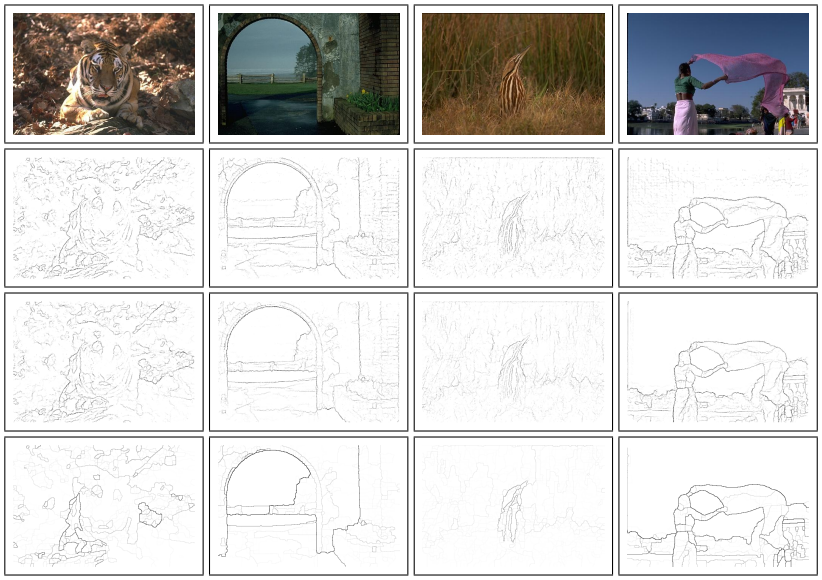


Figure 3: **BSDS500 results:** Top first row shows BSDS500 images. The bottom three bottom rows show the boundary probabilities for SGBD-naive, SGBD and SGBD-owt-ucm. Columns show image with (a) min recall, (b) max recall, (c) min precision, and (d) max precision. These are the probability maps produced by our method. Observe how crisp are the boundaries.

take local edge strength (i.e., texture information) into account. The second variant, termed SGBD, uses texture to estimate the local edge strength (i.e., equation 5). The third variant, termed SGBD-owt-ucm, applies the Ultrametric Contour Map on top of the results of the SGBD algorithm. As can be seen in table 1 the naive algorithm achieves a 0.68 F-score with no texture information whatsoever. Adding texture information improves the results to 0.7 F-score and adding the Ultrametric Contour Map method brings the overall performance to 0.72.

Figures 3 show results on the BSDS500 data set. In particular, we show the images with the lowest/highest precision/recall. Figure 5 shows the ROC curve of our method, along with several other methods.

Next we conduct an experiment to demonstrate the robustness of our algorithm to noise. In the experiment we add increasing amounts of noise to the BSDS500 test images and compare our method to that of [9] and [12]. Results are reported in Figure 6. As can be seen, our algorithm is quite robust to noise because of its

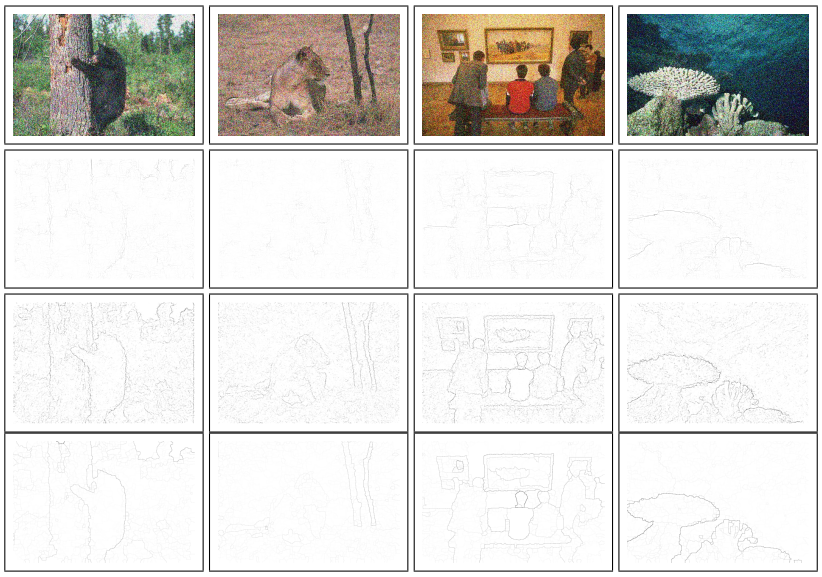


Figure 4: **Noise experiment:** The top row shows several noisy images from the BSDS500 data set, second row shows the results of [9], the bottom two rows show SGBD and SGBD-owt-ucm results, respectively. Observe our algorithm does a better job detecting boundaries in the image. SGBD detects most of the relevant boundary pixels. SGBD-owt-ucm cleans up the result.

BSDS300			
Method	**OSD**	**OIS**	**AP**
Human	0.79	-	-
SCG[8]	0.71	-	-
gPb-owt-ucm[11]	0.71	0.74	0.73
Gb[5]	0.67	-	-
BEL [6]	0.66	-	-
Canny [4]	0.58	0.62	0.58
SGBD-owt-ucm	0.69	0.71	0.69

BSDS500			
Method	**OSD**	**OIS**	**AP**
Human	0.80	0.80	-
Crisp-MS, [12]	0.74	0.77	0.78
SE-MS, T=4 [9]	0.74	0.76	0.78
SCG [8]	0.74	0.76	0.77
gPb-owt-ucm [11]	0.73	0.76	0.73
Sketch Tokens [22]	0.73	0.75	0.78
gPb[11]	0.71	0.74	0.65
NCuts[23]	0.64	0.68	0.45
MeanShift[24]	0.64	0.68	0.56
Felz-Hutt[25]	0.61	0.64	0.56
Canny[4]	0.60	0.63	0.58
SGBD-owt-ucm	0.72	0.74	0.72
SGBD	0.70	0.72	0.73
SGBD-naive	0.68	0.69	0.71

Table 1: **Results on the BSDS dataset:** Our results (bottom row) compared to recent techniques.

semi-global nature. The method of [9] works better on clean images but degrades faster as the amount of noise increases. The method of [12] degrades drastically, probably because the pairwise pixel statistics are strongly corrupted by the noise. Even when the amount of noise is large (std of 38 intensity values) our algorithm manages to maintain a gap from [9] ($F = 0.57$ vs. $F = 0.53$). Some examples of the noisy images and the edge maps we produce can be seen in Figure 4.

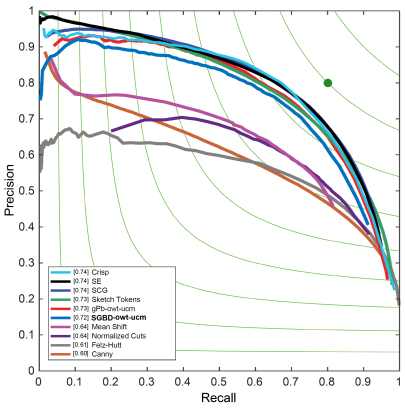


Figure 5: **Precision recall curve on BSDS500 Dataset:** figure copied from [15] with our curve overlaid.

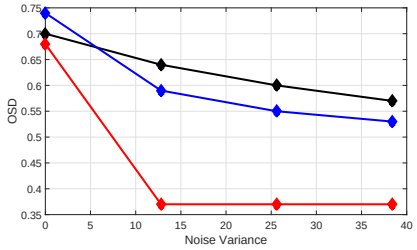


Figure 6: **Noise sensitivity:** F-measure (OSD) as a function of white Gaussian noise added to images. The x-axis is the noise variance (in the scale of $[0, 255]$). The y-axis is the OSD score. [9] (in blue) [12] in speedy mode (in red) and ours (in black). As can be seen, our algorithm degrades gracefully as the amount of noise increases.

The next set of experiments is on the NYU data set. Table 2 reports the results on the RGB images, the depth image, and the RGBD images. Again, our algorithm matches the results of [11, 8, 9] and achieves an F-measure of $F = 0.58$.

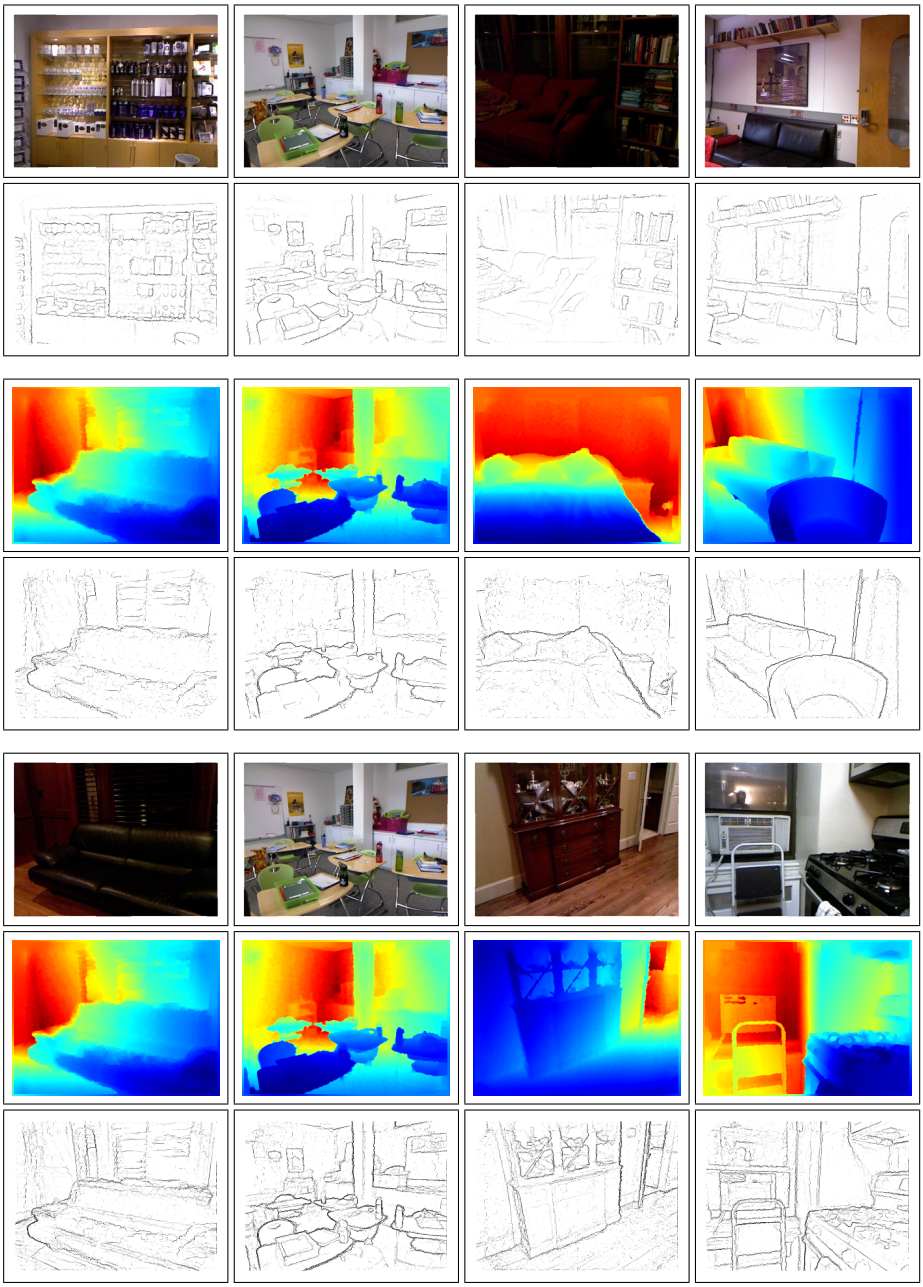


Figure 7: **NYU Results** top two rows RGB images, rows 3 and 4 depth images, rows 5 6 & 7 RGBD images. Columns show image with (a) min recall, (b) max recall, (c) min precision, and (d) max precision.

The method of [9] achieves top score of 0.60 compared to our score of 0.58. However, their method was specifically trained on the NYU dataset. When it was trained on the BSDS500 data set and tested on the NYU data set their results dropped to 0.55. Figure (7) show the results on the RGB only, depth only, and RGBD images. In each case we show the images with the lowest/highest precision/recall score. When dealing with RGBD images, we simply aggregate the results of both sources.

Our method is quite fast in practice. It runs for 8sec per BSDS RGB image (321x481pixels), 3sec per NYU RGB image (320x240 pixels), and 1 sec per NYU depth image. For comparison, Crisp [12] reports 15min per BSDS image, SCG [18] report 1-2sec and SE [9] reports 6hz. Parallelism is inherent to our method, as each line is processed separately. Therefore an efficient implementation, utilizing multiple threads or the GPU will surely boost our run time performance.

The experiments show that our algorithm is competitive across a wide range of images and imaging conditions. We tuned its parameters once on the BSDS500 training set images, which are mainly outdoor images. Then we tested it on the BSDS500 test set where it gave good results as well as on a noisy version of the BSDS500 test images as well as the NYU data set (which is mainly indoor images), where it gave state-of-the-art results. This is probably due to the fact that we need to tune only a small number of parameters. We believe this is an important quality of the algorithm because we do not know who is going to use it and on what type of images. Finally, we observe that our algorithm produces crisp boundaries across all data sets.

NYU			
Method	**OSD**	**OIS**	**AP**
gPb [11]	(0.51,0.44,0.53)	(0.52,0.46,0.54)	(0.37,0.28,0.40)
SCG [8]	(0.55,0.53,0.62)	(0.57,0.54,0.63)	(0.46,0.45,0.54)
SE-MS[9]	(0.60,0.58,0.64)	(0.61,0.59,0.65)	(0.56,0.57,0.63)
SE-MS[9]	(0.55, - , -)	(0.57, - , -)	(0.46, - , -)
SGBD	(0.58,0.53,0.60)	(0.59,0.55,0.61)	(0.52,0.45,0.58)

Table 2: **Results on the NYU dataset:** Our result (bottom row) compared to recent techniques. Each parenthesis holds the values for RGB, depth and RGBD. [9] report different results when training on NYU and testing on NYU, or training on BSDS500 and testing on NYU.

6. Conclusions

We presented a simple algorithm for boundary detection that relies on a Semi Global analysis of the image. The algorithm works on scan lines, which are a mid-level image representation, that is not purely local like patch based methods, yet is not global, as it does not work on the entire image. The reason for using scan lines is that they are 1D signals and as such we can use Dynamic Programming to efficiently find a globally optimal segmentation. This is based on the assumption we make that the boundaries between 1D segments are the boundary pixels in the image. We gave a probabilistic analysis that suggests how to combine all the 1D results into a single coherent probabilistic 2D boundary map, and evaluated it on a number of data sets. The algorithm is simple to implement, and works well across a wide range of data sets without any modification.

References

[1] H. Hirschmüller, Stereo processing by semiglobal matching and mutual information, IEEE Trans. Pattern Anal. Mach. Intell. 30 (2) (2008) 328–341.

[2] D. Scharstein, R. Szeliski, A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, Int. J. Comput. Vision 47 (1-3) (2002) 7–42. doi:10.1023/A:1014573219977.
URL http://dx.doi.org/10.1023/A:1014573219977

[3] A. Geiger, P. Lenz, R. Urtasun, Are we ready for autonomous driving? the kitti vision benchmark suite, in: Conference on Computer Vision and PatternRecognition (CVPR), 2012.

[4] J. Canny, A computational approach to edge detection, Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-8 (6) (1986) 679–698. doi:10.1109/TPAMI.1986.4767851.

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

[6] P. Dollár, Z. Tu, S. Belongie, Supervised learning of edges and object boundaries, in: CVPR (2), 2006, pp. 1964–1971.

[7] J. Mairal, M. Leordeanu, F. Bach, M. Hebert, J. Ponce, Discriminative sparse image models for class-specific edge detection and image interpretation, in: ECCV (3), 2008, pp. 43–56.

[8] X. Ren, L. Bo, Discriminatively trained sparse code gradients for contour detection, in: NIPS, 2012, pp. 593–601.

[9] P. Dollár, C. L. Zitnick, Structured forests for fast edge detection, in: ICCV, 2013.

[10] M. Leordeanu, R. Sukthankar, C. Sminchisescu, Efficient closed-form solution to generalized boundary detection, in: ECCV (4), 2012, pp. 516–529.

[11] P. Arbelaez, M. Maire, C. Fowlkes, J. Malik, Contour detection and hierarchical image segmentation, IEEE Trans. Pattern Anal. Mach. Intell. 33 (5) (2011) 898–916.

[12] P. Isola, D. Zoran, D. Krishnan, E. H. Adelson, Crisp boundary detection using pointwise mutual information., in: ECCV (3), 2014, pp. 799–814.

[13] B. Andres, J. H. Kappes, T. Beier, U. Köthe, F. A. Hamprecht, Probabilistic image segmentation with closedness constraints, in: ICCV, 2011, pp. 2611–2618.

[14] R. Bellman, On the approximation of curves by line segments using dynamic programming, Commun. ACM 4 (6) (1961) 284.

[15] R. Zabih, V. Kolmogorov, Spatially coherent clustering using graph cuts, in: CVPR (2), 2004, pp. 437–444.

[16] Y. Boykov, O. Veksler, R. Zabih, Fast approximate energy minimization via graph cuts, IEEE Trans. Pattern Anal. Mach. Intell. 23 (11) (2001) 1222–1239. doi:10.1109/34.969114.
URL http://doi.ieeecomputersociety.org/10.1109/34.969114

[17] Y. Boykov, M. Jolly, Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images, in: ICCV, 2001, pp. 105–112.

[18] C. Rother, V. Kolmogorov, A. Blake, "grabcut": interactive foreground extraction using iterated graph cuts, ACM Trans. Graph. 23 (3) (2004) 309–314.

[19] X. Ren, J. Malik, Learning a classification model for segmentation, in: Proc. 9th Int'l. Conf. Computer Vision, Vol. 1, 2003, pp. 10–17.

[20] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, S. Süsstrunk, SLIC superpixels compared to state-of-the-art superpixel methods, IEEE Trans. Pattern Anal. Mach. Intell. 34 (11) (2012) 2274–2282. doi:10.1109/TPAMI.2012.120.
URL http://dx.doi.org/10.1109/TPAMI.2012.120

[21] T. X. Han, S. Kay, T. S. Huang, Optimal segmentation of signals and its application to image denoising and boundary feature extraction, in: ICIP, 2004, pp. 2693–2696.

[22] J. J. Lim, C. L. Zitnick, P. Dollr, Sketch tokens: A learned mid-level representation for contour and object detection., in: CVPR, IEEE, 2013, pp. 3158–3165.
URL http://dblp.uni-trier.de/db/conf/cvpr/cvpr2013.html

[23] T. Cour, F. Bénézit, J. Shi, Spectral segmentation with multiscale graph decomposition, in: 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005), 20-26 June 2005, San Diego, CA, USA, 2005, pp. 1124–1131. doi:10.1109/CVPR.2005.332.
URL http://dx.doi.org/10.1109/CVPR.2005.332

[24] D. Comaniciu, P. Meer, Mean shift: A robust approach toward feature space analysis, IEEE Trans. Pattern Anal. Mach. Intell. 24 (5) (2002) 603–619. doi:10.1109/34.1000236.
URL http://doi.ieeecomputersociety.org/10.1109/34.1000236

[25] P. F. Felzenszwalb, D. P. Huttenlocher, Efficient graph-based image segmentation, International Journal of Computer Vision 59 (2) (2004) 167–181. doi:10.1023/B:VISI.0000022288.19776.77.
URL http://dx.doi.org/10.1023/B:VISI.0000022288.19776.77