LSD: a Line Segment Detector

Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, Gregory Randall

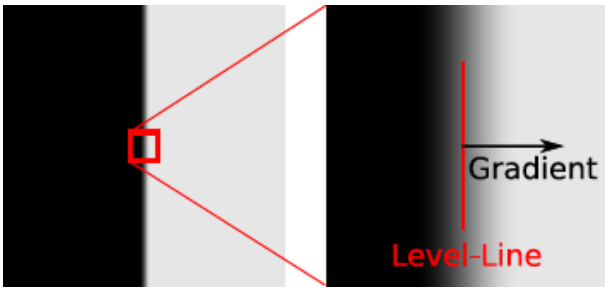
September 2011

Abstract

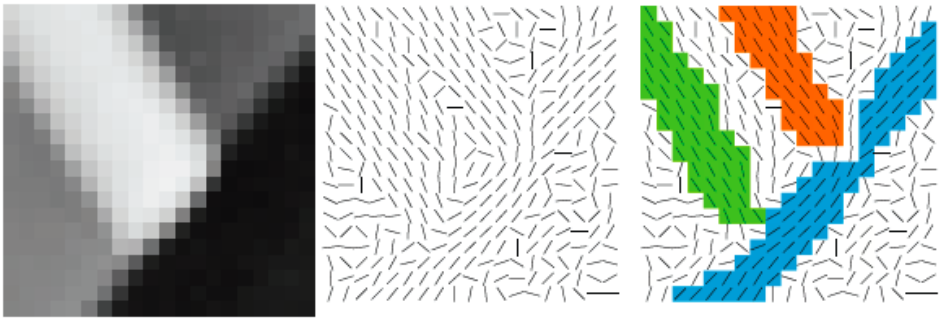
LSD is a linear-time Line Segment Detector that gives accurate results, a controlled number of false detections, and requires no parameter tuning [1]. The method is based in Burns, Hanson, and Riseman method [2], and uses an *a contrario* validation approach according to Desolneux, Moisan, and Morel theory [3].

1 Overview

LSD is aimed at detecting locally straight contours on images. That is what we call *line segments*. Contours are zones of the image where the gray level change rapidly from dark to light (or the opposite). Thus, the gradient and level-lines of the image are key concepts, as is shown on the following figure:

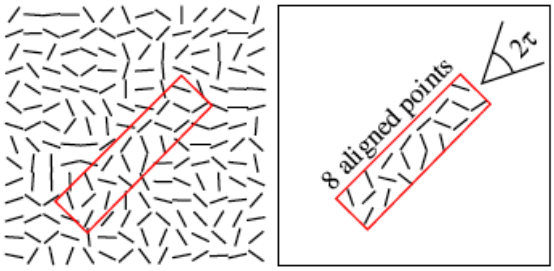


The algorithm starts by computing the level-line angle at each pixel to produce a *level-line field*. Then, this field is segmented into connected regions of pixels that share the same level-line angle up to a certain tolerance τ. These connected regions are called *line support regions*:



Each *line support region* (a set of pixels) is a candidate for a *line segment*. But the corresponding geometrical object (a rectangle in this case) must be associated with it. The principal inertial axis of the *line support region* is used as main rectangle direction; the size of the rectangle is chosen to cover the full region:

Each rectangle is subject to a validation procedure. The pixels in the rectangle whose level-line angle corresponds to the angle of the rectangle up to a tolerance τ are called *aligned points*. The total number of pixels in the rectangle, n, and its number of *aligned points*, k, are counted and used to validate or not the rectangle as a detected *line segment*.



The validation step is based on the *a contrario* approach and the Helmholtz principle proposed by Desolneux, Moisan, and Morel [3]. The so-called Helmholtz principle states that no perception (or detection) should be produced on noise images. Accordingly, the *a contrario* approach propose to define a *noise* or *a contrario* model where the desired structure is not present. Then, an event is validated if the expected number of events as good as the observed one is small on the *a contrario* model. In other words, structured events are rare on the *a contrario* model.

In the case of *line segments*, we are interested in the number of *aligned points* and we consider the events that has at least the same number of *aligned points* as the observed one. Given an image i and a rectangle r, we will note $k(r,i)$ the number of *aligned points* and $n(r)$ the total number of pixels in the r. Then, the expected number of events as good as the observed one are

$$N_{test} \cdot P_{H_0}[k(r,I) \geq k(r,i)]$$

where the *number of tests* N_{test} is the total number of possible rectangles considered, P_{H_0} is the probability on the *a contrario* model H_0, and I is a random image on H_0.

The *a contrario* model H_0 for *line segment* detection is a *level-line field* satisfying the following properties:

- $\{LLA(j)\}_{j \in \text{Pixels}}$ is composed of independent random variables
- $LLA(j)$ is uniformly distributed over $[0, 2\pi]$

where $LLA(j)$ is the level-line angle at pixel j. On this setting, the probability that a pixel on the *a contrario* model is an *aligned point* is

$$p = \frac{\tau}{\pi}$$

and, as a consequence of the independence of the random variables, $k(r,I)$ follows a binomial distribution. Then, the probability term $P_{H_0}[k(r,I) \geq k(r,i)]$ is given by

$$P_{H_0}\big[k(r,I) \geq k(r,i)\big] = B\big(n(r), k(r,i), p\big)$$

where $B(n,k,p)$ is the tail of the binomial distribution:

$$B(n,k,p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}$$

The *number of tests* N_{test} corresponds to the total number of rectangles considered. That is all the rectangles starting and ending in pixels of the image. In a $N \times M$ image that gives $NM \times NM$ different configurations. Also, $\sqrt{NM}$ different width values are considered for each one. Then, the *number of tests* is

$$(NM)^{5/2}.$$

Finally, we define the Number of False Alarms (NFA) associated with a rectangle r on image i as

$$\text{NFA}(r,i) = (NM)^{5/2} \cdot b\Big(n(r), k(r,i), p\Big).$$

That corresponds to the expected number of rectangles, as good as r, to be found on H_0. When the NFA value associated with a rectangle on an image is large, that means that such an event is expected on the *a contrario* model, i.e., common and thus not a relevant one. On the other hand, when the NFA value is small, the event is rare and probably a meaningful one. A threshold ε is selected and rectangles with $\text{NFA}(r,i) \leq \varepsilon$ are called ε*-meaningful rectangles* and are the detections of the algorithm.

Theorem:

$$E_{H_0}\left[\sum_{r\in\mathcal{R}} \mathbb{1}_{\text{NFA}(r,I)\leq\varepsilon}\right] \leq \varepsilon$$

where E is the expectation operator, $\mathbb{1}$ is the indicator function, $\mathcal{R}$ is the set of rectangles considered, and I is a random image on H_0.

The theorem states that the average number of ε*-meaningful rectangles* on *a contrario* model H_0 images is less than ε. ε controls the number of detections on noise and it can be made as small as desired. In other words, it shows that LSD satisfies the Helmholtz principle.

Proof.
We define $\hat{k}(r)$ as

$$\hat{k}(r) = \min\left\{ n \in \mathbb{N},\ P_{H_0}\big[k(r,I) \geq n\big] \leq \frac{\varepsilon}{(NM)^{5/2}} \right\}.$$

Then, $\text{NFA}(r,i) \leq \varepsilon$ is equivalent to $k(r,i) \geq \hat{k}(r)$. Now,

$$E_{H_0}\left[\sum_{r\in\mathcal{R}} \mathbb{1}_{\text{NFA}(r,I)\leq\varepsilon}\right] = \sum_{r\in\mathcal{R}} P_{H_0}\big[\text{NFA}(r,I) \leq \varepsilon\big] = \sum_{r\in\mathcal{R}} P_{H_0}\left[k(r,I) \geq \hat{k}(r)\right].$$

But, by definition of $\hat{k}(r)$ we know that

$$P_{H_0}\left[k(r,I) \geq \hat{k}(r)\right] \leq \frac{\varepsilon}{(NM)^{5/2}}.$$

and using that $\#\mathcal{R} = (NM)^{5/2}$ we get

$$E_{H_0}\left[\sum_{r\in\mathcal{R}} \mathbb{1}_{\text{NFA}(r,I)\leq\varepsilon}\right] \leq \sum_{r\in\mathcal{R}} \frac{\varepsilon}{(NM)^{5/2}} = \varepsilon.$$

□

Following Desolneux, Moisan, and Morel [3], we set ε=1 once for all. This corresponds to accepting on average one false detection per image in the *a contrario* model, which is reasonable. Also, the detection result is not sensitive to the value of ε so setting it to any reasonable value would produce very similar results.

2 Algorithm

The LSD algorithm takes a gray-level image as input and returns a list of detected *line segments*. The algorithm can be described by the following 12 steps that will be described in detail below. The auxiliary image STATUS has the same size as the scaled image, and is used to keep track of the pixels already used.

---

1. Scale the input image to scale S using Gaussian sub-sampling (σ=Σ/S).
2. Compute the gradient magnitude and level-line orientation at each pixel.
3. Build a list of pixels pseudo-ordered according to their image gradient magnitude.
4. Set all pixels in the auxiliary image STATUS to the value NOT USED.
5. Mark the STATUS of pixels whose gradient magnitude is less than ρ to the value USED.
6. For each pixel P in the list, starting with the ones with the higher gradient magnitude, and with STATUS set to NOT USED do:
   7. Starting from P as a seed pixel, grow a region R of connected and NOT USED pixels, that share the same level-line angle up to a tolerance τ. Mark the STATUS of the pixels in the region as USED.
   8. Compute the rectangular approximation for the connected region R of pixels found.
   9. If the density of aligned points in the rectangle is less than D, cut the region, until the density restriction is satisfied.
   10. Compute the NFA value for the rectangle found.
   11. Try to modify the rectangle to improve the NFA value.
   12. If NFA(r) $\leq \varepsilon$, add the rectangle to the output list.

The algorithm depends on six parameters: S, Σ, ρ, τ, D, and ε. As we will see, all of these parameters have reasonable default values that produce good results in most images. Except D that is set to an empirical value, all the other parameters have a justified default value. Thus, there is no need for parameter tuning.

2.1 Image scaling

The first step of LSD is to scale the input image. This step can be useful to analyze structures present at different scales of an image. This step can also be skipped (by selecting S=1) to process the image at full resolution. The scale S is, *a priori*, a choice of the user and in a sense a true parameter or tool to control the analysis performed by LSD. However, the default value sets a scaling to 80% of the image size (S=0.8); the reason is to avoid aliasing and quantization artifacts (especially the staircase effect) present in many images while producing almost the same result as a full scale analysis. 80% is the smallest image reduction that solves the staircase problem.

The scaling is performed by a Gaussian sub-sampling: the image is filtered with a Gaussian kernel to avoid aliasing and then sub-sampled. The standard deviation of the Gaussian kernel is determined by σ=Σ/S, where S is the scaling factor. The value of Σ is set to 0.6 that gives a good balance between aliasing and blurring of the image.

2.2 Gradient computation

The image's gradient is computed at each pixel using a 2x2 mask. Given

$\ddots$	$\vdots$	$\vdots$	$\cdots$
$\cdots$	$i(x,y)$	$i(x+1,y)$	$\cdots$
$\cdots$	$i(x,y+1)$	$i(x+1,y+1)$	$\cdots$
$\cdots$	$\vdots$	$\vdots$	$\ddots$

where $i(x,y)$ is the image gray level value at pixel (x,y), the image gradient is compute as

$$g_x(x,y) = \frac{i(x+1,y) + i(x+1,y+1) - i(x,y) - i(x,y+1)}{2},$$

$$g_y(x,y) = \frac{i(x,y+1) + i(x+1,y+1) - i(x,y) - i(x+1,y)}{2}.$$

The level-line angle is computed as

$$\arctan\left(\frac{g_x(x,y)}{-g_y(x,y)}\right)$$

and the gradient norm as

$$G(x,y) = \sqrt{g_x^2(x,y) + g_y^2(x,y)}.$$

The reason for using this simple schema is to use the smallest possible mask size in its computation, thus reducing to the minimum possible the dependence of the computed gradient values (thus, approaching the theoretical independence in the case of a noise image).

The gradient and level-line angles encode the direction of the edge, that is the angle of the dark to light transition. Note that a dark to light transition and a light to dark transition are different, having a 180 degree angle difference between the corresponding gradient or level-line angles. This means that the resulting *line segments* detected by LSD are oriented and that the order of the start and end points is not arbitrary as it encode which side of the *line segment* is darker. For example, if an image is inverted (changing black for white and white for black) the result of LSD would be the same but the start and end points will be exchanged on every *line segment*.

Note that the computed value corresponds to the image gradient at coordinates (x+0.5,y+0.5) and not (x,y). This half-pixel offset is then added to the output rectangles coordinates to produce coherent results.

In the *a contrario* model, the *level-line field* is composed of independent random variables at each pixel. But the computed *level-line field* is never independent because adjacent pixel values are used to compute it. This does not prevent to use an *a contrario* approach, but requires a more involved analysis to compute the detection threshold. Actually, numerical simulations have shown that the same threshold deduced for the case of independent *level-line field* also controls the number of false detections when computed by the 2x2 mask, see [4].

2.3 Gradient Pseudo-ordering

LSD is a greedy algorithm and the order in which pixels are processed has an impact on the result. Pixels with high gradient magnitude corresponds to more contrasted edges, and in an edge, the center pixels usually have higher gradient magnitude. So it makes sense to start looking for *line segments* at pixels with the higher gradient magnitude.

But sorting pixels by gradient value is computationally more expensive than the rest of the LSD algorithm that can be done in linear-time. However, a pseudo-ordering of pixels is possible in linear-time and is what is done in LSD, so the full algorithm runs in linear-time relative to the number of pixels in the image.

Pixels are classified into bins according to their gradient value. Then, a list is made beginning with the pixels in the bin with highest gradient values, followed by the pixels on the second bin and so on, ending the list with the pixels in the bin with smallest gradient values.

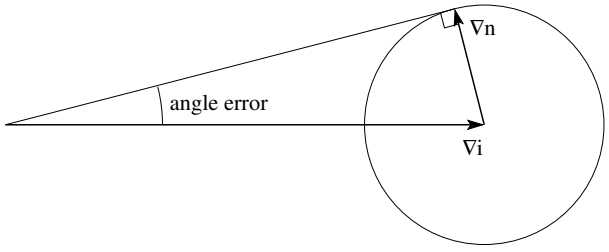
By default, 1024 bins are used, sorting pixels almost by strict gradient value when gray level values are quantized in the integer range [0,255].

2.4 Gradient threshold

Pixels with small gradient norm corresponds to flat zones or slow gradients. Also, they naturally present a higher error in the gradient computation due to the quantization of its values. In LSD the pixels with gradient magnitude smaller then ρ are rejected and are not used in the constructions of *line-support regions* or rectangles.

Assuming a quantization noise n, what we observe is the image

$$\tilde{i} = i + n \qquad \nabla\tilde{i} = \nabla i + \nabla n.$$



We have

$$|\text{angle error}| \leq \arcsin\left(\frac{q}{|\nabla i|}\right),$$

where q is a bound to $|\nabla n|$. Imposing $|\text{angle error}| \leq \tau$ we get

$$\rho = \frac{q}{\sin\tau}.$$

The threshold ρ is set using the last expression where q is a bound to the possible error in the gradient value due to quantization effects [1], and τ is the angle tolerance to be used in the region growing algorithm. In the usual case, the pixel values are quantized to integer values in $\{0, 1, \ldots, 255\}$. Thus, the maximum possible error in the gradient is 2 (when adjacent pixels have quantization errors of one that do not compensate). The default value is then $q = 2$.

2.5 Region Growing

Starting from a pixel in the list of unused pixels, the seed, a region growing algorithm is applied to form a *line-support region*. Recursively, the unused neighbors of the pixels already in the region are tested, and the ones whose level-line angle is equal to the so-called *region angle* θ_{region} up to a tolerance τ are added to the region. The initial *region angle* θ_{region} is assigned to the level-line angle of the seed point, and each time a pixel is added to the region the region angle value is updated to the following

$$\arctan\left(\frac{\sum_j \sin(\text{level-line-angle}_j)}{\sum_j \cos(\text{level-line-angle}_j)}\right)$$

where the index j runs over the pixels in the region. If we associate to each pixel in the region an unitary vector with its level-line angle, the latter formula corresponds to the angle of the mean vector. The process is repeated until no other pixel can be added to the region. The following pseudo-code gives a precise definition:

---

1. The seed point P is added to the Region
2. θ_{region} is set to the level-line angle of pixel P
3. $S_x \leftarrow \cos(\theta_{region})$
4. $S_y \leftarrow \sin(\theta_{region})$
5. For each pixel P in the Region do:
    1. For each pixel Q neighbor of P and Status$(Q) \neq$ USED do:
        1. If AngleDifference $(\theta_{region}, \text{level-line-angle}(Q)) < \tau$ do:
            1. Add Q to the Region
            2. Status$(Q) \leftarrow$ USED
            3. $S_x \leftarrow S_x + \cos(\text{LevelLineAngle}(Q))$
            4. $S_y \leftarrow S_y + \sin(\text{LevelLineAngle}(Q))$
            5. $\theta_{region} \leftarrow \arctan(S_y/S_x)$

The neighborhood used is 8-connected, so the neighbors of pixel $i(x, y)$ are $i(x-1, y-1)$, $i(x, y-1)$, $i(x+1, y-1)$, $i(x-1, y)$, $i(x+1, y)$, $i(x-1, y+1)$, $i(x, y+1)$, and $i(x+1, y+1)$.

An extensive set of experiments support the selection of the value used for τ. Its default value is 22.5 degree or $\pi/8$ radian, that corresponds to a 45 degree range or $1/8$ of the full range of orientations. It is near the largest possible value that still makes sense to call "oriented" along the rectangle. Regions that would be obtained with a smallest value are also obtained. In some cases, some extra pixels could be added. However, in the validation process, smallest values of precision p are tested, so this parameter value only affects the region growing algorithm and not the validation.

2.6 Rectangular Approximation

A *line segment* corresponds to a geometrical event, a rectangle. Before evaluating a *line-support region*, the rectangle associated with it must be found. The region of pixels is interpreted as a solid object and the gradient magnitude of each pixel is used as the "mass" of that point. Then, the center of mass of the region is selected as the center of the rectangle and the main direction of the rectangle is set to the first inertia axis of the region. Finally, the width and length of the rectangles are set to the smallest values that makes the rectangle to cover the full *line-support region*.

The center of the rectangle (c_x, c_y) is set to:

$$c_x = \frac{\sum_{j \in \text{Region}} G(j) \cdot x(j)}{\sum_{j \in \text{Region}} G(j)}$$

$$c_y = \frac{\sum_{j \in \text{Region}} G(j) \cdot y(j)}{\sum_{j \in \text{Region}} G(j)}$$

where $G(j)$ is the gradient magnitude of pixel j, and the index j runs over the pixels in the region. The main rectangle's angle is set to the angle of the eigenvector associated with the smallest eigenvalue of the following matrix:

$$M = \begin{pmatrix} m^{xx} & m^{xy} \\ m^{xy} & m^{yy} \end{pmatrix}$$

with

$$m^{xx} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (x(j) - c_x)^2}{\sum_{j \in \text{Region}} G(j)}$$

$$m^{yy} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (y(j) - c_y)^2}{\sum_{j \in \text{Region}} G(j)}$$

$$m^{xy} = \frac{\sum_{j \in \text{Region}} G(j) \cdot (x(j) - c_x)(y(j) - c_y)}{\sum_{j \in \text{Region}} G(j)}.$$

2.7 NFA Computation

A key concept in the validation of a rectangle is that of *p-aligned points*, that are pixels in the rectangle whose level-line angle is equal to the rectangle's main orientation, up to a tolerance $p\pi$. The *precision* p is initially set to the value τ/π but other values are also tested as we will show later. The total number of pixels in the rectangle is denoted by n and the number of *p-aligned points* is denoted by k (we drop r and i when they are implicit to simplify the notation). Then, the Number of False Alarms (NFA) value associated with the rectangle r is

$$\text{NFA}(r) = (NM)^{5/2} \cdot b(n, k, p)$$

where N and M are the number of columns and rows of the image, and $b(n, k, p)$ is the binomial tail defined by

$$b(n, k, p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

All in all, for each rectangle being evaluated and given a *precision* p, the numbers k and n are counted, and then the NFA value is computed by

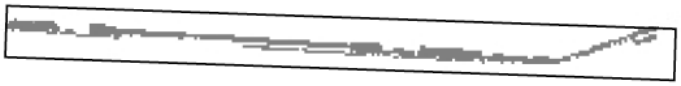
$$\text{NFA}(r) = (NM)^{5/2} \cdot \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

The rectangles with $\text{NFA}(r) \leq \varepsilon$ are validated as detections.

As stated before, and following Desolneux, Moisan, and Morel [3], we set ε=1 once for all.

2.8 *Aligned Points* Density

In some cases, the τ-angle-tolerance method produces a wrong interpretation. This problem can arise when two straight edges are present in the image forming an angle between them smaller than the tolerance τ. The following image shows an example of a *line-support region* found (in gray) and the rectangle corresponding to it.



This *line-support region* could be better interpreted as two thinner rectangles, one longer than the other, forming an obtuse angle.

In LSD this problem is handled by detecting problematic *line-support regions* and cutting them into two smaller regions, hoping to cut the region in the right place to solve the problem. Once a cut region is accepted, the rectangle associated is recomputed and the algorithm is resumed.

The detection of this "angle problem" is based on the density of *aligned points* in the rectangle. When this problem is not present, the rectangle is well adapted to the *line-support region* and the density of *aligned points* is high. On the other hand, when the "angle problem" is present, as can be seen on the previous figure, the density of *aligned points* is very low. Also, when a slightly curved edge is being locally approximated by a sequence of straight edges, the degree of the approximation (how many *line segments* are used to cover part of curve) is related to the density of *aligned points*; this parameter, then, also controls how curves are approximated by *line segments*.

The density of *aligned points* of a rectangle is computed as the ratio of the number of *aligned points* (k in the previous notation) to the area of the rectangle:

$$d = \frac{k}{\text{length}(r) \cdot \text{width}(r)}.$$

A threshold D is defined and rectangles should have a density d larger or equal to D to be accepted. The default value of this parameter is 0.7 (70 %) which provides good balance between solving the "angle problem", providing smooth approximations to curves, without over-cutting too much *line segments*.

When this threshold is not satisfied, two methods for cutting the region are tried: first the method called *reduce angle tolerance* and then *reduce region radius*. In both methods, part of the pixels in the region are kept, while the others are re-marked again as NOT USED, so they can be used again in future *line-support regions*. We will describe now these two methods:

2.8.1 Reduce angle tolerance

The first method, *reduce angle tolerance*, tries to guess a new tolerance τ' that adapts well to the region, and then the region growing algorithm is used again with the same seed but using the newly estimated tolerance value. When two straight regions that form an obtuse angle are present, this method is expected to get the tolerance that would get only one of these regions, the one containing the seed pixel.

If all the pixels in the region were used in the estimation of the tolerance, the new value would be such that all the pixels would still be accepted. Instead, only the pixels near the seed are used. Actually, the pixels used are those whose distance to the seed point is less than the width of the rectangle initially computed. In that way, the size of the neighborhood used in the estimation of τ' adapts to the size of the region.

All the pixels in that neighborhood of the seed point are evaluated, and the new tolerance τ' is set to twice the standard deviation of the level-line angles of these pixels. With this new value, the same region growing algorithm is applied, starting from the same seed point. Before that, all the pixels on the original regions are set to NOT USED, so the algorithm can use them again, and the discarded ones are available for further regions.

2.8.2 Reduce region radius

The previous method, *reduce angle tolerance*, is tried only once, and if the resulting *line-support region* fails to satisfy the density criterion a second method is repetitively tried. The idea of this second method is to gradually remove the pixels that are farther from the seed until the criterion is satisfied or the region is too small and rejected. This method work best when the *line-support region* corresponds to a curve and the region needs to be reduced until the density criterion is satisfied, usually meaning that a certain degree of approximation to the curve is obtained.

The distance from the seed point to the farther pixel in the region is called the *radius* of the region. Each iteration of this method removes the farthest pixels of the region to reduce the region's *radius* to 75 % of its value.

This process is repeated until the density criterion is satisfied or there are not enough pixels in the region to form a *meaningful rectangle*.

2.9 Rectangle Improvement

Before rejecting a *line-support region* for being not *meaningful* (NFA $> \varepsilon$), LSD tries some variations to the rectangle's configuration initially found with the aim to get a valid one.

The relevant factors tested are the *precision* p used and the width of the rectangle.

The initial *precision* used, corresponding to the region growing tolerance τ is large enough so only testing smaller values make sense. The same is true for width of the rectangle because the initial width was chosen to cover all the *line-support region*. But in some cases, for example, reducing in one pixel the width may reduce the number of *aligned points* only by one, while reducing the total number of pixels by a number equal to the length of the rectangle; the balance could improve the NFA value greatly.

The *rectangle improvement* routine of LSD consists of the following steps:

1. try finer *precisions*
2. try to reduce width
3. try to reduce one side of the rectangle
4. try to reduce the other side of the rectangle
5. try even finer precisions

If a *meaningful rectangle* is found (NFA $\leq \varepsilon$) the improvement routine will stop after the step that found it.

Step 1 tries the following *precision* values: $p/2$, $p/4$, $p/8$, $p/16$, and $p/32$, where p is the initial *precision* value. The value that produces the best NFA value (the smallest) is kept.

Step 2 tries up to five times to reduce the rectangle width in 0.5 pixels. That means that the width values tested are W, $W - 0.5$, $W - 1$, $W - 1.5$, $W - 2$, and $W - 2.5$, where W is the initial width value. Again, the value that produces the best NFA value is kept.

Step 3 tries five times to reduce only one side of the rectangle in 0.5 pixel. This implies reducing the width of the rectangle in 0.5 pixels but also moving the center of the rectangle in 0.25 pixels in order that the other side of the rectangle remains on its place. So the side displacements tested are 0.5, 1, 1.5, 2, and 2.5 pixels. As before, the value that produces the best NFA value is kept.

Step 4 do the same thing as step 3 but to the other side.

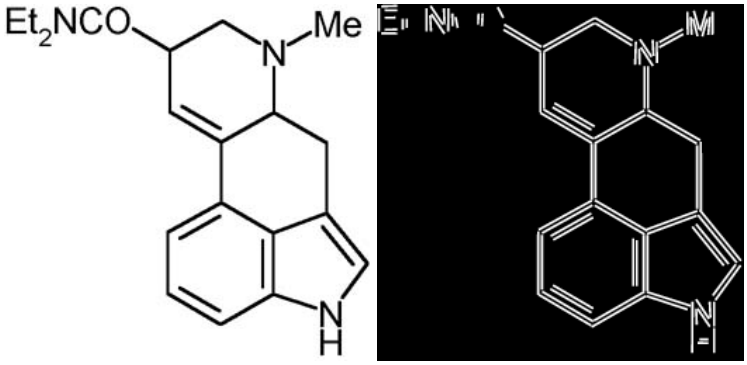
Step 5 tries again to reduce the *precision* even more. This step tests the precision values $\hat{p}/2$, $\hat{p}/4$, $\hat{p}/8$, $\hat{p}/16$, and $\hat{p}/32$, where $\hat{p}$ is the *precision* at the beginning of this step. The value that produces the best NFA value is kept.

3 Examples

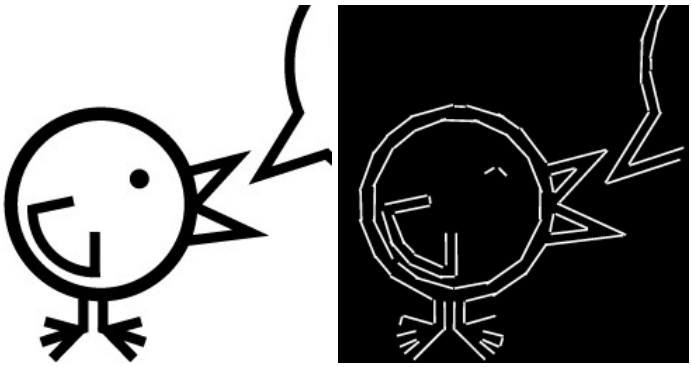
The following set of examples try to give an idea of the kind of results obtained with LSD, both good and and bad results:

Office: A good result. The *line segments* detected corresponds to straight structures in the image. The detection corresponds roughly with the expected result.

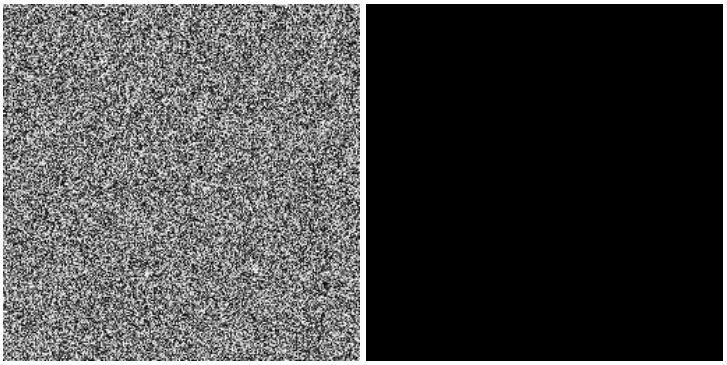
LSD: Note that LSD detect locally straight *edges*, so each black strokes produce *two* detections, one for each white to black transition. Also note that there is a minimal length that a line segment must have, and smaller ones cannot be detected. (For example, the base of the number '2'.) This minimal size adapts with the image size.



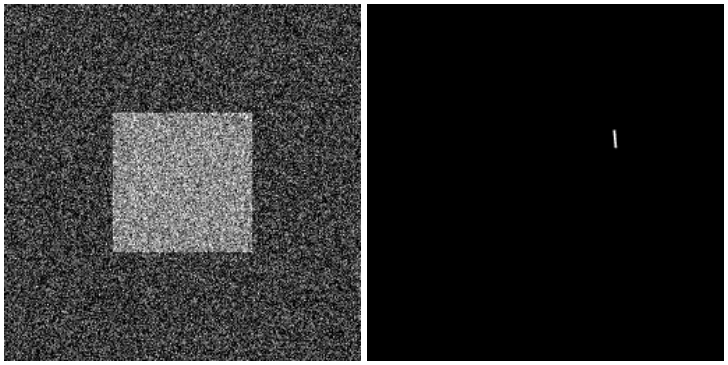
Pipi: Note that when curves are present, LSD produce short line segments corresponding to curve sections that are locally straight. The result is a polygonal approximation for curves.



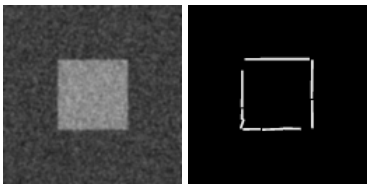
Noise: LSD was designed to provide a good false detection control. Its false detection control is based in automatically providing detections thresholds that prevent detections that could happen by chance on noise images.



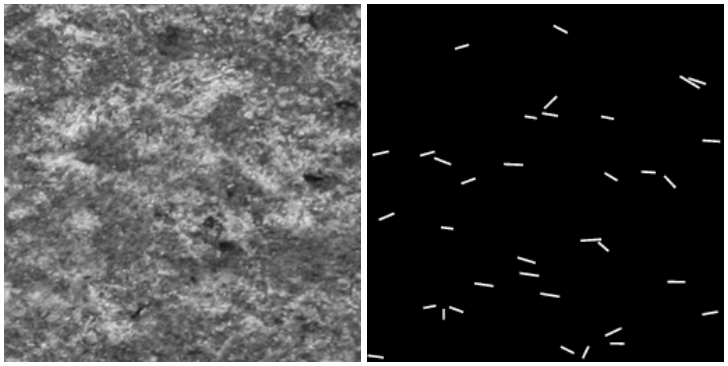
Square: The square is masked by noise and is not detected by LSD. But the square can be detected if the image is analyzed at a different scale by Gaussian sub-sampling.



Square-subsampling: When a Gaussian sub-sampling is applied to the previous image, the noise is partially removed and the expected line segments are detected.

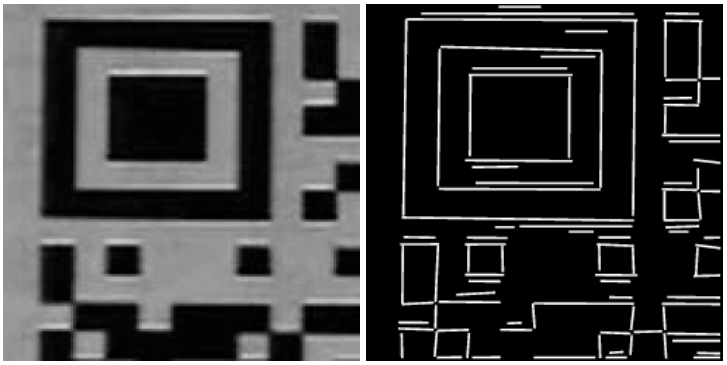


Wall: Some regions are partially anisotropic and partially straight. This regions can produce unexpected detections.



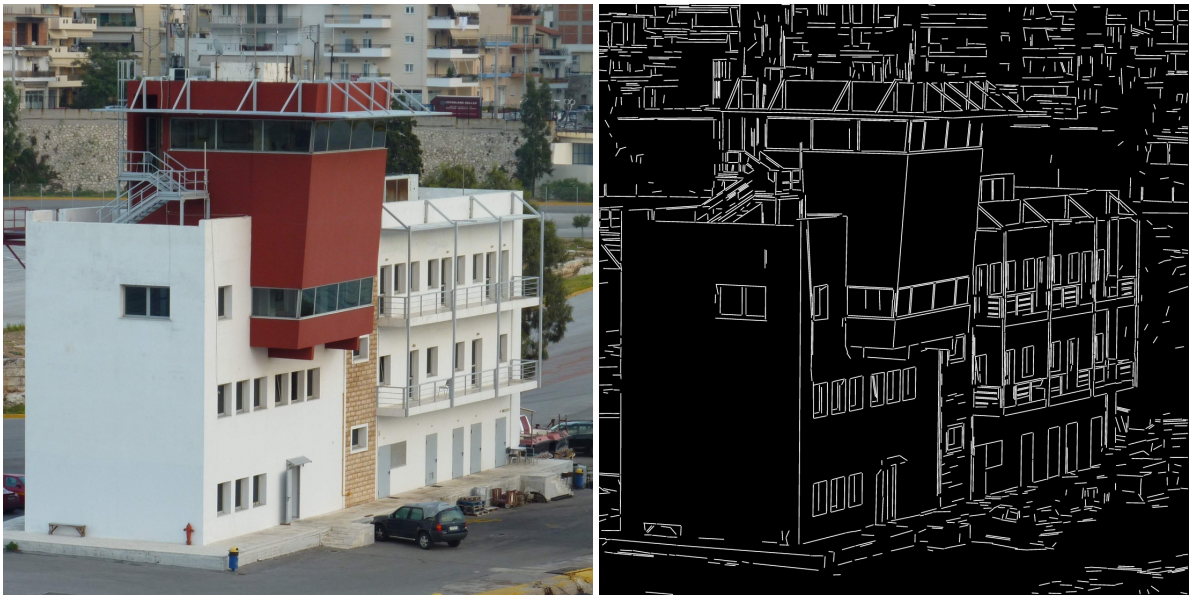
Map: In some images we get detections that seems strange at first sight, even if they are correct.

Gibbs: Image compression Gibbs effect is responsible of many unexpected detections.



Color: LSD is designed to work on gray-level images. Before applying LSD to a color image it must be converted to a gray-level image. However, some color edges could be lost in this conversion. For example, the following image presents a prominent color edge (left), but after the standard conversion to a gray-level image (middle) the edge is lost. The reason is that both, the red values and the green values are converted to the same gray value. Thus, LSD will produce no detection (right) because *none is present* in the input image to LSD (middle). The edge is lost on the color to gray image conversion and not at LSD. However, LSD can be blamed for not handling natively color images; such an extension is possible but was not done on the current implementation.

Natural scene: All in all, LSD usually produces a reasonable result on natural images.

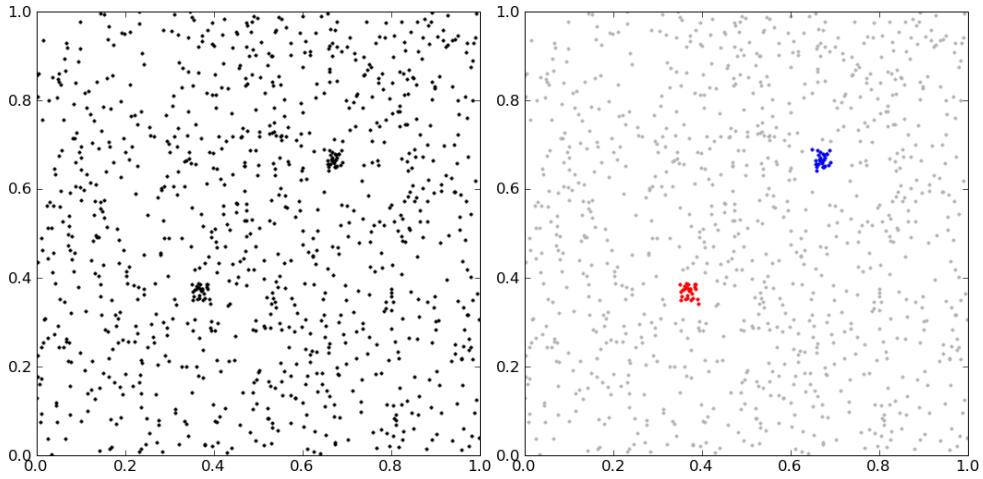


4 Projects

LSD was designed to be a parameterless automatic algorithm: given the input data, the result is produced without any human supervision. The projects presented here share the same goal. One of them use the same *a contrario* frameworks used by LSD; the others use a similar statistical tools to set detection thresholds that control the number of false positives. LSD detects locally straight edges on images; three of projects dealt with general edge detectors.

4.1 *A contrario* clustering

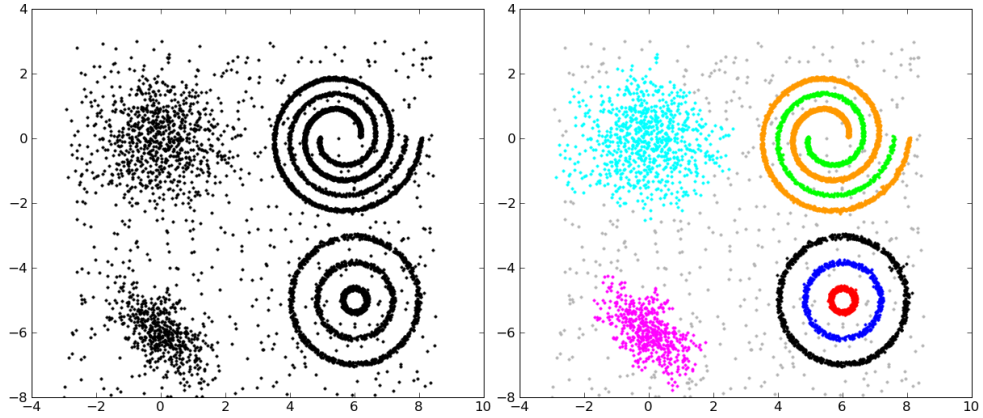
Clustering is the task of grouping subsets of a set of data points into *clusters*, so that the points in the a cluster differentiate in some sense from the rest. The following figure shows an example: on the left we see the set of data points; on the right, two clusters are highlighted on red and blue.



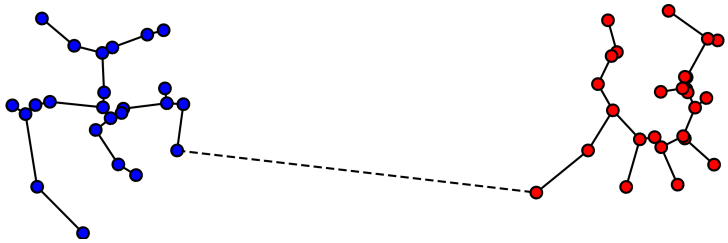
Clustering is an old problem in data analysis and hundreds of algorithms were proposed in the last 50 years. Here we propose to study a recent one based on the *a contrario* framework:

> "Meaningful Clustered Forest: an Automatic and Robust Clustering Algorithm" by Mariano Tepper, Pablo Musé, & Andrés Almansa, `http://arxiv.org/abs/1104.0651`.

The algorithm works parameterless and produce impressive state-of-the-art results as the following one:



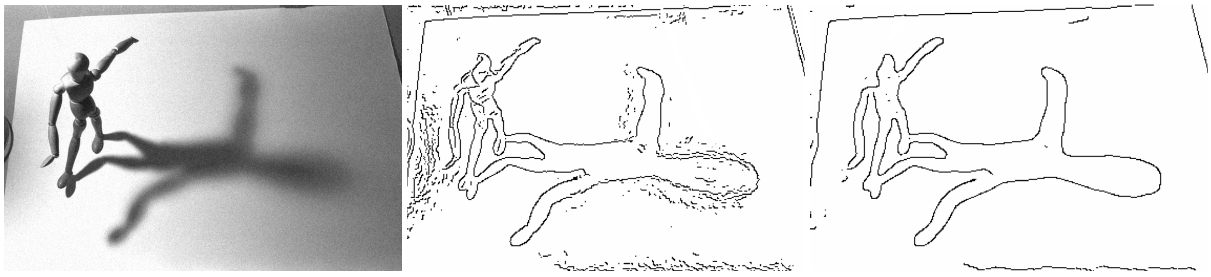
The algorithm depends on a notion of distance between points. With it, the Minimum Spanning Tree (MST) is computed (that is, an incomplete graph joining all points with the smallest sum of distances along its edges). A sub-graph of the MST will be considered meaningful when all the distances on the edges are too short to have arisen by random points with uniform distribution.



The project will be done in collaboration with the authors of the algorithm and part of the necessary code modules are already available.

4.2 Edge detection without parameters

Edge detection algorithms usually depends on at least two parameters: the scale of the analysis and a detection threshold. The following example shows an image and the result at two different scale sizes of a classic method.



None of the two results are satisfactory. On the first one we get a detailed analysis of the mannequin, but the shadow produces a noisy result. On the other hand, when a larger scale is used the shadow is better resolved but at the price of missing the details on the mannequin. The detection threshold didn't perform a good job either as the spurious detections show.

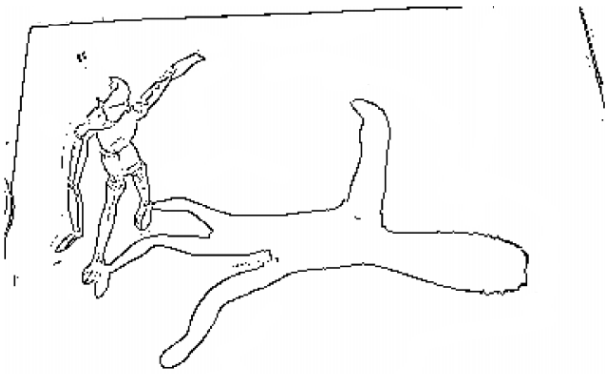
We propose the study and implementation of two algorithms designed to handle this problem and produce parameterless edge detection. Thus, two projects are proposed, one for each algorithm:

> "Local Scale Control for Edge Detection and Blur Estimation" by James H. Elder & Steven W. Zucker, IEEE PAMI, vol.20, no.7, pp.699-716, July 1998.

and

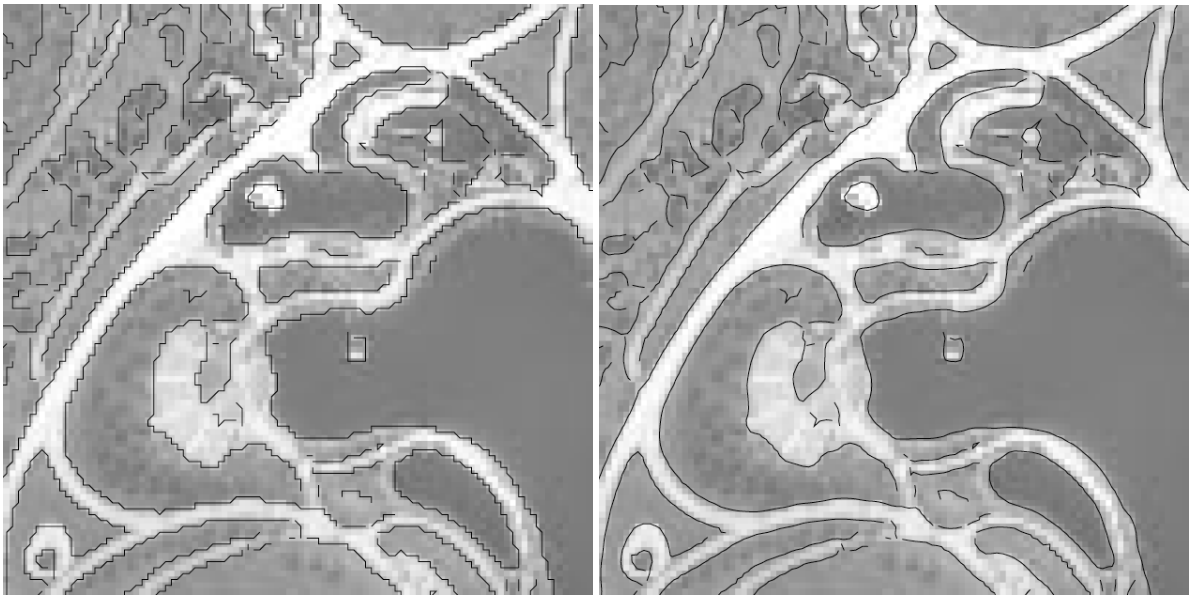
> "Multiscale Edge Detection and Fiber Enhancement Using Differences of Oriented Means" by Meirav Galun, Ronen Basri & Achi Brandt, ICCV 2007.

Both algorithms assume known the power of an additive Gaussian noise. This is reasonable as good algorithms for estimating the noise power are now available. Linear filters are used to measure the edge response at different positions and scales. Similar to what is done in the *a contrario* approach, the response of these filters is considered reliable when its value is larger than the values that could be expected due to noise without an edge. The two algorithms propose different approaches for handling detections at multiple scales. The following is the result of the first algorithm to the previous example.



4.3 Devernay's sub-pixel edge detector

Most edge detection algorithms produce results with pixel precision. But images usually contain enough information for finer localization as is shown below:



We propose to study and implement the algorithm proposed by Devernay:

> "A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy" by Frédéric Devernay. Rapport de recherche INRIA No.2724, November 1995.

It is based on the classic edge detector by Canny, to which a second refinement step is added. In spite of its simplicity, the algorithm produces precisions as low as a 10th or a 20th of a pixel.

References

[1] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, Gregory Randall, *LSD: A Fast Line Segment Detector with a False Detection Control*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 4, pp. 722-732, April 2010.

[2] J. Brian Burns, Allen R. Hanson, Edward M. Riseman, *Extracting Straight Lines*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 4, pp. 425-455, 1986.

[3] Agnès Desolneux, Lionel Moisan, Jean-Michel Morel, *From Gestalt Theory to Image Analysis, a Probabilistic Approach*, Springer 2008.

[4] Rafael Grompone von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, Gregory Randall, *On Straight Line Segment Detection*, Journal of Mathematical Imaging and Vision, vol. 32, no. 3, pp. 313-347, November 2008.

[5] ariano Tepper, Pablo Musé, & Andrés Almansa, *Meaningful Clustered Forest: an Automatic and Robust Clustering Algorithm*. `http://arxiv.org/abs/1104.0651`.

[6] ames H. Elder & Steven W. Zucker, *Local Scale Control for Edge Detection and Blur Estimation*, IEEE PAMI, vol.20, no.7, pp.699-716, July 1998.

[7] eirav Galun, Ronen Basri & Achi Brandt, *Multiscale Edge Detection and Fiber Enhancement Using Differences of Oriented Means*, ICCV 2007.

[8] rédéric Devernay, *A Non-Maxima Suppression Method for Edge Detection with Sub-Pixel Accuracy*. Rapport de recherche INRIA No.2724, November 1995.