LSD: a Line Segment Detector

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

published • 2012-03-24 →BibTeX
reference • Rafael Grompone von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, Gregory Randall, *LSD: a Line Segment Detector*, Image Processing On Line, 2012. http://dx.doi.org/10.5201/ipol.2012.gjmr-lsd

Communicated by Lionel Moisan
Demo edited by Rafael Grompone

Authors

- Rafael Grompone von Gioi grompone@cmla.ens-cachan.fr
- Jérémie Jakubowicz jakubowi@telecom-paristech.fr
- Jean-Michel Morel morel@cmla.ens-cachan.fr
- Gregory Randall randall@fing.edu.uy

Summary

LSD is a linear-time Line Segment Detector giving subpixel accurate results. It is designed to work on any digital image without parameter tuning. It controls its own number of false detections: On average, one false alarms is allowed per image [1]. The method is based on Burns, Hanson, and Riseman's method [2], and uses an *a contrario* validation approach according to the Desolneux, Moisan, and Morel's theory [3,4]. The version described here includes some further improvement over the one described in [1].

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. preprint
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, *Meaningful Alignments*, International Journal of Computer Vision, vol. 40, no. 1, pp. 7-23, 2000. preprint
4. Agnès Desolneux, Lionel Moisan, Jean-Michel Morel, *From Gestalt Theory to Image Analysis, a Probabilistic Approach*, Springer 2008.
5. 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. preprint

Source Code

The ANSI C implementation of LSD, version 1.6, is the one which has been peer reviewed and accepted by IPOL. The code documentation, including the source code, is accessible here.

Supplementary material:

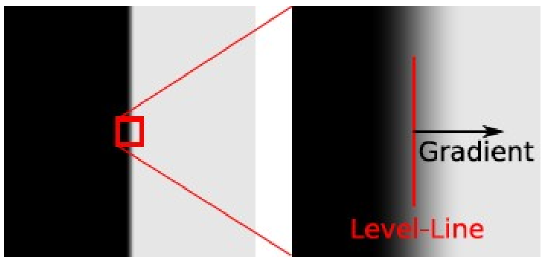
Version 1.0 of LSD corresponds better to the algorithm described in [1] and does not include the further improvements described here and included in the current version. It can be compiled as a pure C language code or using the Megawave2 framework.

Note that the interface changed from version 1.5 of LSD to the current version 1.6.

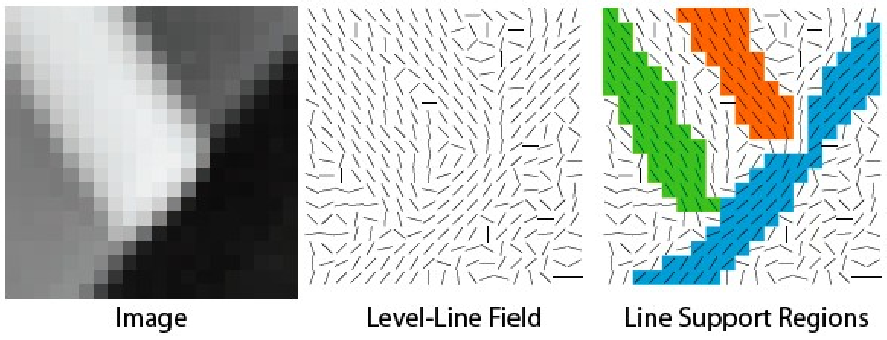
Versions 1.0 and 1.5 are both non reviewed material.

Overview

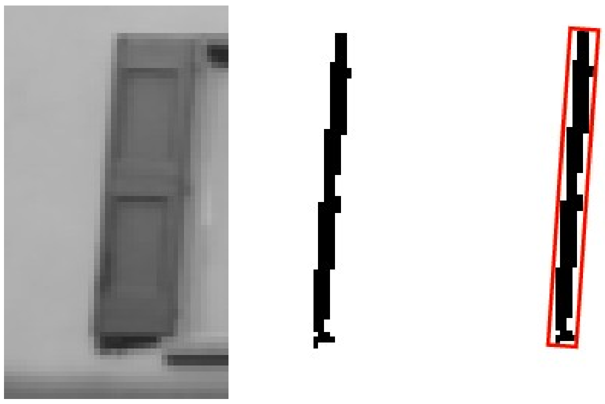
LSD is aimed at detecting locally straight contours on images. This is what we call *line segments*. Contours are zones of the image where the gray level is changing fast enough from dark to light or the opposite. Thus, the gradient and level-lines of the image are key concepts and are illustrated on the following figure:



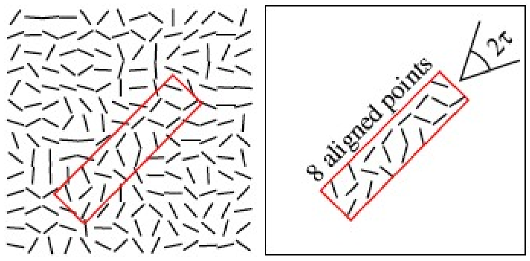
The algorithm starts by computing the level-line angle at each pixel to produce a *level-line field*, i.e., a unit vector field such that all vectors are tangent to the level line going through their base point. Then, this field is segmented into connected regions of pixels that share the same level-line 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*. 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 [3,4]. The so-called Helmholtz principle states that no perception (or detection) should be produced on an image of noise. Accordingly, the *a contrario* approach proposes to define a *noise* or *a contrario* model H_0 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 defined as being rare in the *a contrario* model.

In the case of *line segments*, we are interested in the number of *aligned points*. We consider the event that a *line segment* in the *a contrario* model has as many or more aligned points, as in the observed *line segment*. 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 *r*. Then, the expected number of events which are as good as the observed one is

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

where the *number of tests* N_{test} is the total number of possible rectangles being considered, P_{H_0} is the probability on the *a contrario* model H_0 (that is defined below), and *I* is a random image following H_0. The H_0 stochastic model fixes the distribution of the number of aligned points *k(r,I)*, which only depends on the distribution of the level-line field associated with *I*. Thus H_0 is a noise model for the image gradient orientation rather than a noise model for the image.

Note that *k(r,I)* is an abuse of notation as *I* does not corresponds to an image but to a *level-line field* following H_0. Nevertheless, there is no contradiction as *k(r,I)* only depends on the gradient orientations.

The *a contrario* model H_0 used for *line segment* detection is therefore defined as a stochastic model of the *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 [0,2π]

where *LLA(j)* is the level-line angle at pixel *j*. Under hypothesis H_0, the probability that a pixel on the *a contrario* model is an *aligned point* is

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

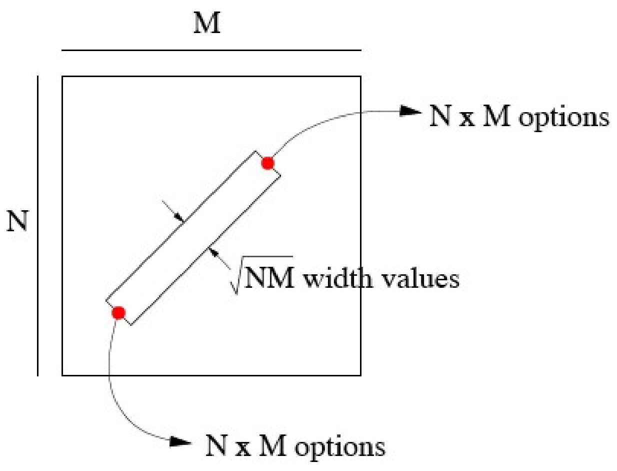
and, as a consequence of the independence of the random variables *LLA(j)*, *k(r,I)* follows a binomial distribution. Thus, the probability term $P_{H_0}[k(r,I) \geq k(r,i)]$ is given by

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

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 that could show an alignment at a fixed precision. Notice that the rectangles are *oriented*, meaning that the order of their starting and ending points is not arbitrary: it encodes which side of the *line segment* is darker. Thus, a rectangle from point A to point B is a different test from the rectangle from point B to point A. The exhaustive choice is to take all the rectangles starting and ending at image pixels. In a $N \times M$ image this gives $NM \times NM$ different rectangles. Also, $\sqrt{NM}$ different width values are considered for each one.



The number of rectangles considered is then

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

The *precision p* is initially set to the value τ/π; but other values are also tested to cover the relevant range of values; this is explained below in the section "Rectangle Improvement". We will note γ the number of different *p* values potentially tried. Each rectangle with each *p* value is a different test. Thus, the final *number of tests* is

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

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

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

This corresponds to the expected number of rectangles which have a sufficient number of *aligned points* to be as rare as *r* under H_0. When the NFA associated with an image rectangle is large, this 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 $NFA(r,i) \leq \varepsilon$ are called *ε-meaningful rectangles* and are the detections of the algorithm.

Theorem

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

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* under the *a contrario* model H_0 is less than ε. Thus, the number of detections on noise is controlled by ε and it can be made as small as desired. In other words, this shows that LSD satisfies the Helmholtz principle.

Proof

This proof is given here because it was not given in the original article [1].

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

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

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

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

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

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

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

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

which concludes the proof.

Following Desolneux, Moisan, and Morel [3,4], 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. Indeed, the detection limit (that is the minimal number of aligned points that could lead to a *ε-meaningful rectangle*) varies like $\sqrt{-\log\varepsilon}$. Setting ε to any reasonable value would produce very similar results.

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. 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 highest gradient magnitude, and with STATUS set to NOT USED, do:
    1. 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.
    2. Compute the rectangular approximation for the connected region R of pixels found.
    3. If the density of aligned points in the rectangle is less than D, cut the region, until the density restriction is satisfied.
    4. Compute the NFA value for the rectangle found.
    5. Try to modify the rectangle to improve the NFA value.
    6. If $\mathrm{NFA}(r) \leq \varepsilon$, add the rectangle to the output list.

LSD was designed as an automatic image analysis tool. As such it must work without requiring any parameter tuning. The algorithm actually depends on several numbers that determine its behavior; but their values were carefully devised to work on all images. (See their discussion below.) They are therefore part of LSD's design, internal parameters, and are not left to the user's choice. Changing their values would amount to define a new variant of the algorithm, in the same way as we could make variants by changing the gradient operator, or by switching from 8-neighborhood to 4-neighborhood in the region growing process.

Each step of the algorithm will be described in the following sections, as well as the design criteria for setting the six internal parameters: S, Σ, ρ, τ, D, and ε.

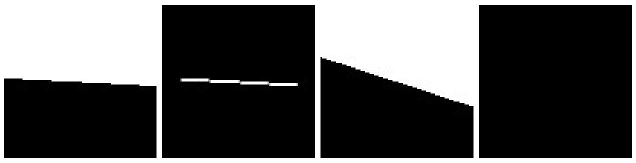
[The algorithm main function in LSD code is `LineSegmentDetection`.]

Image scaling

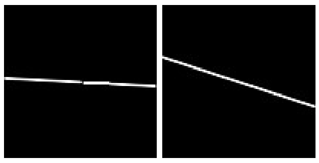
The result of LSD is different when the image is analyzed at different scales or if the algorithm is applied to a small part of the image. This is natural and corresponds to the different details that one can see if an image is observed from a distance or if attention is paid to a specific part. As a result of the *a contrario* validation step, the detections thresholds automatically adapt to the image size that is directly related to the number of tests. The scale of analysis is a choice left to the user, who can select it by cropping the image. Otherwise LSD processes automatically the entire image.

The first step of LSD is, nevertheless, to scale the input image to 80% of its size. This scaling helps to cope with aliasing and quantization artifacts (especially the staircase effect) present in many images. Blurring the image would produce the same effect but affecting statistics of an image in the *a contrario* model: some structures would be detected on a blurred white noise. When correctly sub-sampled, the white noise statistics are preserved. Note that the *a contrario* validation is applied to the scaled image and the $N \times M$ image size used in the NFA computation corresponds to an input image of size $1.25N \times 1.25M$.

The following images show two discrete edges at different angles, both presenting the staircase effect. Next to each image is the result of LSD without using the initial scaling. In the first case the edge is detected as four horizontal line segments instead of one; in the second case, no line segment is detected:



In both cases the result is reasonable, but it does not correspond to what we would expect. The following figures show the result of LSD, using the 80% scaling. Both edges are now detected and with the right orientation (even if the first one is still fragmented).



The scale factor of 80% (S=0.8), is the smallest image reduction that reasonably solves the staircase problem while producing almost the same result as a full scale analysis on images without artifacts. (A 80% scaling means here that the x and y axis are each reduced to 80%; the number of pixels is thus reduced to 64%.)

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, which gives a good balance between avoiding aliasing and avoiding image blurring.

[The scaling is performed in LSD code by the function `gaussian_sampler`.]

Gradient computation

The image 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 computed 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 magnitude as

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

This simple scheme uses the smallest possible mask size in its computation, thus reducing as much as 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 their starting and ending points is not arbitrary, since it encodes which side of the *line segment* is darker. For example, if the contrast of an image is reverted (changing black for white and white for black) the result of LSD would be the same but the starting and ending points would 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 actually never fully independent even if the image is a white noise. Indeed, adjacent pixel values are used to compute the gradient and therefore the gradients are (slightly) dependent. This does not prevent the use of an *a contrario* approach. Indeed, 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 [5].

[In the LSD code, the gradient computation, gradient threshold, and pseudo-ordering, are all performed by the function `ll_angle`.]

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 correspond to the more contrasted edges. In an edge, the central pixels usually have the highest gradient magnitude. So it makes sense to start looking for *line segments* at pixels with the highest gradient magnitude.

Sorting algorithms usually require $O(n \log n)$ operations to sort *n* values. However, a simple pixel pseudo-ordering is possible in linear-time. To this aim, 1024 bins are created corresponding to equal gradient magnitude intervals between zero and the largest observed value on the image. Pixels are classified into the bins according to their gradient magnitude. LSD uses first seed pixels from the bin of the largest gradient magnitudes; then it takes seed pixels from the second bin, and so on until exhaustion of all bins. 1024 bins are enough to sort almost strictly the gradient values when the gray level values are quantized in the integer range [0,255].

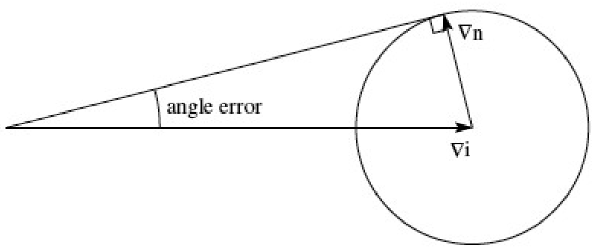
[In the LSD code, the gradient computation, gradient threshold, and pseudo-ordering, are all performed by the function `ll_angle`.]

Gradient threshold

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

Assuming a quantization noise *n* and an ideal image *i* we observe:

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



We have

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

where *q* is a bound on $|\nabla n|$. The criterion used is to reject pixels where the angle error is larger than the angle tolerance τ used in the region growing algorithm. That is, we impose $|\text{angle error}| \leq \tau$ and we get

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

The threshold ρ is set using the last expression where *q* is a bound on 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,...,255}. Thus, the maximal possible error in the gradient is 1 (when adjacent pixels have quantization errors of 0.5 that do not compensate). For empirical reasons, we preferred a more conservative bound and we set *q=2*. This value will not, however, give good results if the image intensity range differs significantly from the [0,255] interval.

[In the LSD code, the gradient computation, gradient threshold, and pseudo-ordering, are all performed by the function `ll_angle`.]

Region Growing

Starting from a pixel in the ordered 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 *region angle* θ_{region} up to a tolerance τ are added to the region. The initial *region angle* θ_{region} is 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

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

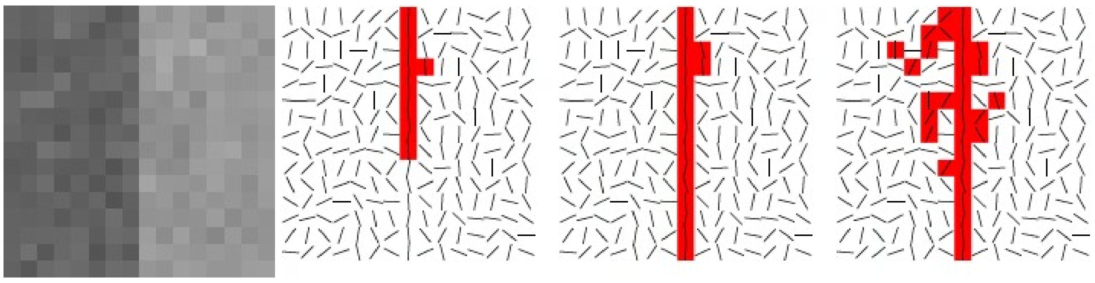
where the index *j* runs over the pixels in the region. If we associate to each pixel in the region a unitary vector with its level-line angle, the latter formula corresponds to the angle of the mean vector. The process is repeated until no other pixel can be added to the region. The following pseudo-code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

```
1. The initial point P is added to the Region
2. $\theta_{region}$ 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:
           $\text{Status}(Q) \neq \text{USED}$
```

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

An 8-connected neighborhood is used, 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)*.

The tolerance τ is set to 22.5 degree or π/8 radian, that corresponds to a 45 degree range or 1/8 of the full range of orientations. It was chosen because it is near the largest possible value that still makes sense to call a pixel "oriented like the rectangle". What is important is not the exact value but the order of magnitude, so it was set to obtain *p*=1/8. The next figure shows a typical example. On the left we see a detail of a noisy edge. Next to it is the result of the region growing algorithm for τ set to 11.25, 22.5, and 45 degree, respectively. The first case is too restrictive and the region is too small; with 45 degree regions often expand too far from the edge; 22.5 is a good compromise.



Regions that could be obtained with a smaller value are also obtained in this way. In the validation process, smaller values of the precision *p* are also tested, so the value of τ only affects the region growing algorithm and not the validation.

[The region growing step is done in the LSD code by the function `region_grow`.]

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 make 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 matrix

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

with

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

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

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

[In the LSD code, the rectangular approximation is computed by the function `region2rect`, using the function `get_theta` to compute the main rectangle's angle.]

NFA Computation

A key concept in the validation of a rectangle is that of *p-aligned points*, namely the pixels in the rectangle whose level-line angle is equal to the rectangle's main orientation, up to a tolerance pπ. The *precision p* is initially set to the value τ/π, but other values are also tested as is explained in the section "Rectangle Improvement"; a total of γ different values for *p* are tried. 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) associated with the rectangle *r* is

$$\mathrm{NFA}(r) = (NM)^{5/2}\gamma \cdot B(n,k,p)$$

where N and M are the number of columns and rows of the image (after scaling), and *B(n,k,p)* is the binomial tail

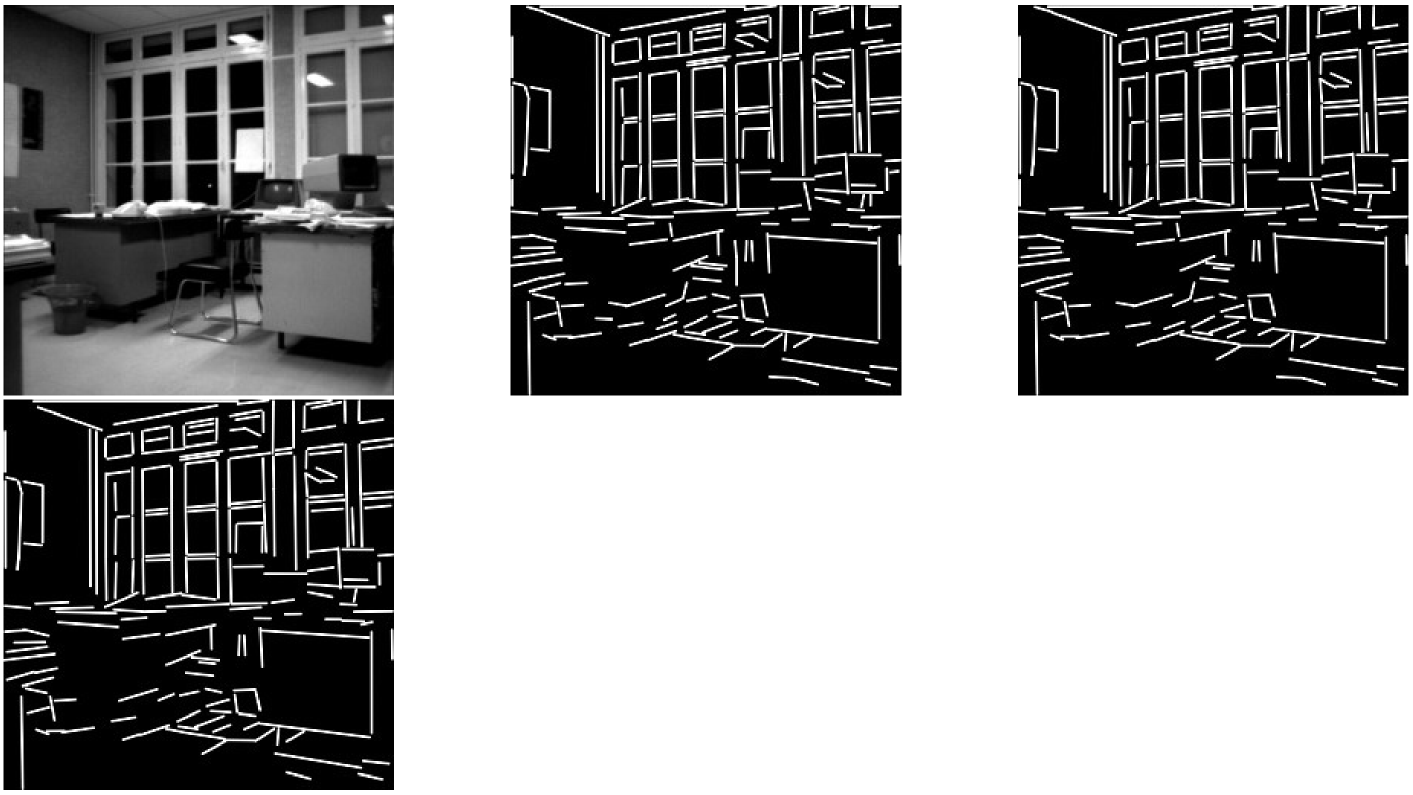
$$B(n,k,p) = \sum_{j=k}^{n} \binom{n}{j} p^j (1-p)^{n-j}.$$

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

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

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

As stated before, and following Desolneux, Moisan, and Morel [3,4], we set ε=1 once for all. Here we will only show an experiment illustrating the stability of the result relative to ε value. Below is the input image and the result of LSD with $\varepsilon = 1$, $\varepsilon = 10^{-1}$, and $\varepsilon = 10^{-2}$, respectively. Only a few small line segments disappear:



In our implementation, the computation of the binomial tail is performed using the the following relation to the Gamma function:

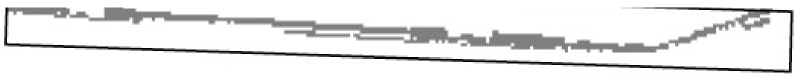
$$\binom{n}{k} = \frac{\Gamma(n+1)}{\Gamma(k+1)\cdot\Gamma(n-k+1)}.$$

The Gamma function can be efficiently computed. We use the methods by Lanczos and Windschitl as described on http://www.rskey.org/gamma.htm. To speed up the computations, the sum of the binomial tail is truncated when the error can be bounded to be less than 10%.

[The function `rect_nfa` of the LSD code counts the number of *aligned points* and the total number of points, and then computes the NFA, calling the function `nfa` to compute the binomial tail.]

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 at 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 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*; the *aligned points* density is thus also related to the precision at which 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:

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

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

Two methods for cutting the region are actually tried: *reduce angle tolerance* and *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:

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, only the pixels whose distance to the seed point is less than the width of the rectangle initially computed are used. 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.

[The reduce angle tolerance method is implemented by the function `refine` in the LSD code.]

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 works 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 until there are not enough pixels in the region to form a *meaningful rectangle*. This is just a way of gradually reducing the region until the criterion is satisfied; it could be done one pixel at a time, but that would make the process slower.

[The *reduce region radius* method is implemented in the `reduce_region_radius` function of the LSD code.]

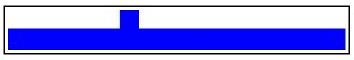
Rectangle Improvement

Before rejecting a *line-support region* for being not *meaningful* (NFA>ε), LSD tries some variations of 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 makes sense. If the pixels are well aligned, using a finer *precision* will keep the same number of aligned points, but a smaller *p* yields a smaller (and therefore better) NFA.

In a similar way, it only makes sense to try to reduce the rectangle's width because the initial width was chosen to cover the whole *line-support region*. Often, reducing by one pixel the width may reduce the number of *aligned points* by only a few units while reducing the total number of pixels by a number equal to the length of the rectangle, see the figure below. This may decrease significantly the binomial tail and therefore also the NFA.



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 ($\mathrm{NFA} \leq \varepsilon$) 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 (the smallest) value is kept.

Step 2 tries up to five times to reduce the rectangle width by 0.5 pixels. This means that the tested width values 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 by 0.5 pixel. This implies reducing the width of the rectangle by 0.5 pixels but also moving the center of the rectangle by 0.25 pixels to maintain the position of the other side of the rectangle. So the tested side displacements 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 does the same thing as step 3 on the other side of the rectangle.

Step 5 tries again to reduce the *precision* still 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.

In addition to the initial precision $p = \frac{\tau}{\pi}$, five more values are potentially tested in step 1 and still five more in step 5. Then $\gamma = 11$. The range of precisions covered is from $p = \frac{\tau}{\pi}$ to $p = \frac{\tau}{1024\pi}$ and is more than enough to consider any relevant case, the finer precision being about 0.02 degree. Five such steps, attaining a 1 degree precision, would be enough; this refinement, however, works better sometimes before and sometimes after the width refinement, and there is no serious caveat in performing both.

[The rectangle improvement routine is performed by the function `rect_improve` of LSD code.]

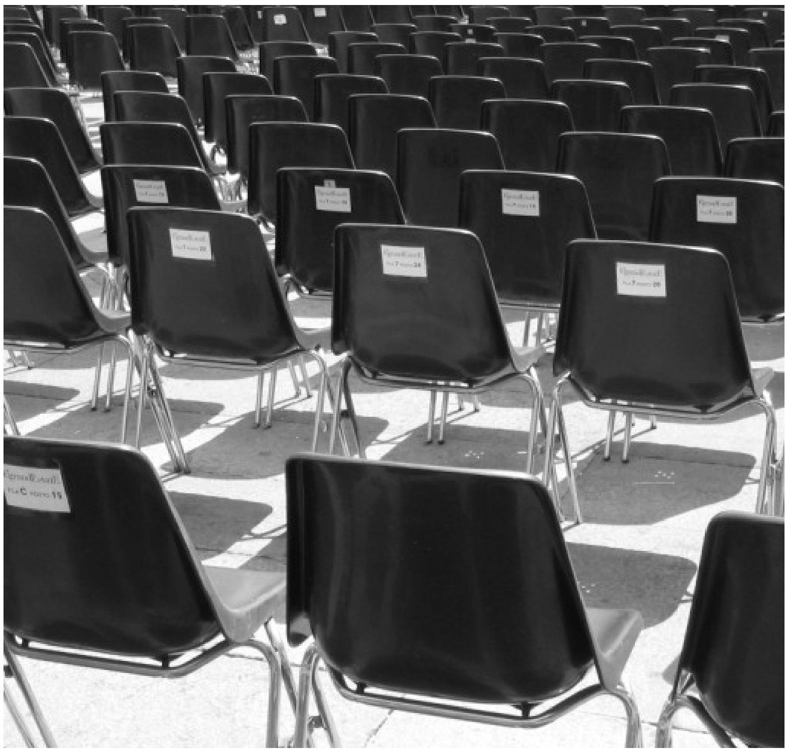
Computational complexity

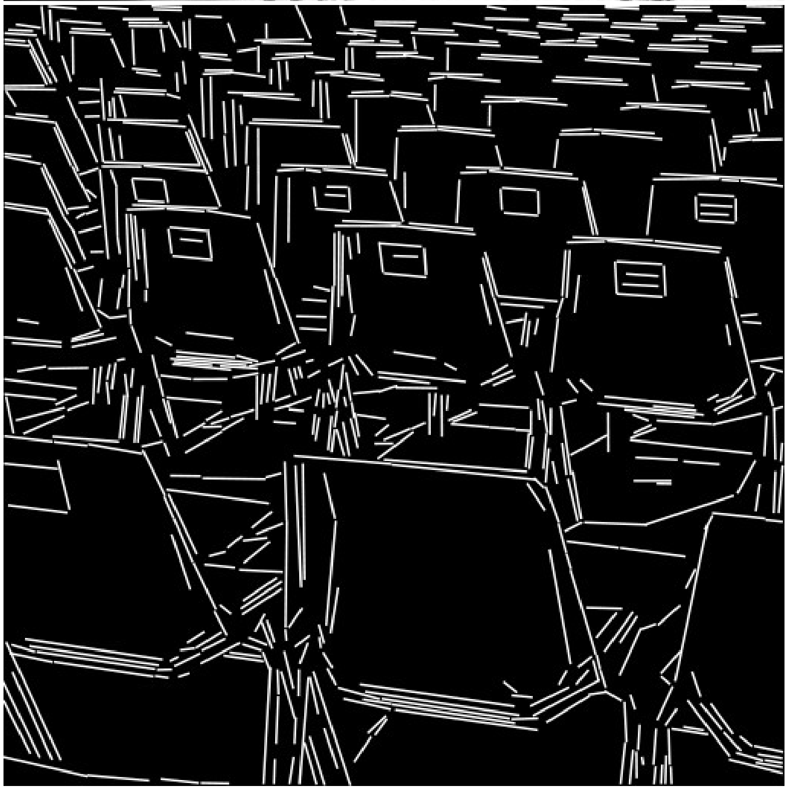
Performing a Gaussian sub-sampling and computing the image gradient, both can be performed with a number of operations proportional to the number of pixels in the image. Then, pixels are pseudo-ordered by a classification into bins, operation that can be done in linear time. The computational time of the line-support region finding algorithm is proportional to the number of visited pixels, and this number is equal to the total number of pixels in the regions plus the border pixels of each one. Thus, the number of visited pixels remains proportional to the total number of pixels of the image. The rest of the processing can be divided into two kind of tasks. The first kind, for example summing the region mass or counting aligned points, are proportional to the total number of pixels involved in all regions. The second kind, like computing inertia moments or computing the NFA value from the number of aligned points, are proportional to the number of regions. Both the total number of pixels involved and the number of regions are at most equal to the number of pixels. All in all, LSD has an execution time proportional to the number of pixels in the image.

Examples

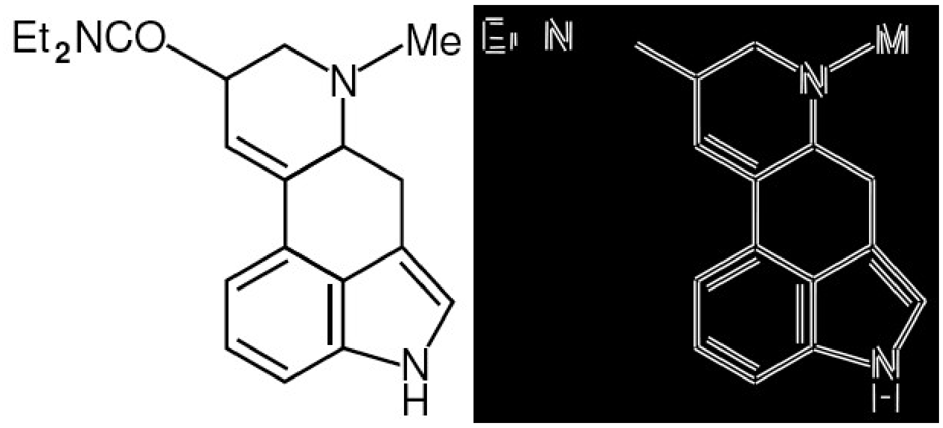
The following set of examples tries to give an idea of the kind of results obtained with LSD, both good and bad.

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

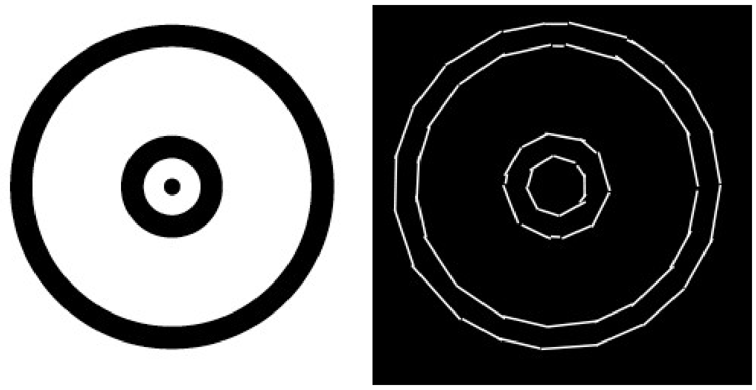




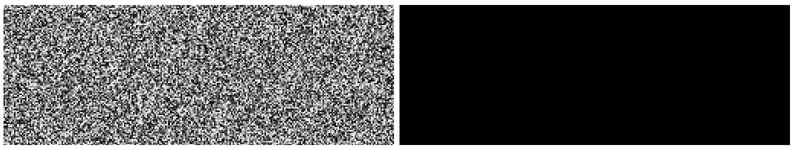
molecule-lsd: Note that LSD detects locally straight *edges*, so each black stroke produces *two* detections, one for each white to black transition. Also note that there is a minimal length that a line segment must have, and smaller ones cannot be detected. (For example, the base of the number '2'.) This minimal size for detection depends on the image size because the NFA increases with the image size.

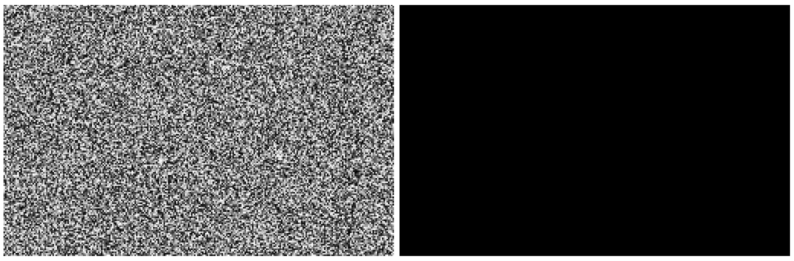


circles: Note that when curves are present, LSD produces short line segments corresponding to curve sections that are locally straight. The result is a polygonal approximation for curves. When the curvature is too strong LSD of course fails.

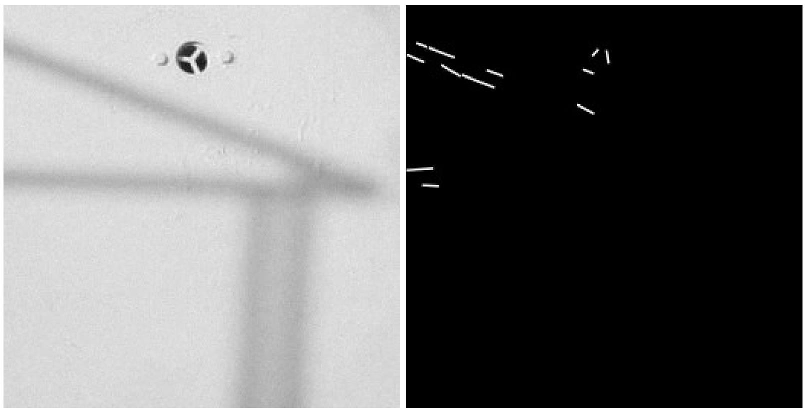


noise: LSD was designed to provide a good false detection control. Its false detection control is based on automatically providing detections thresholds that prevent detections that could happen by chance on images of noise.

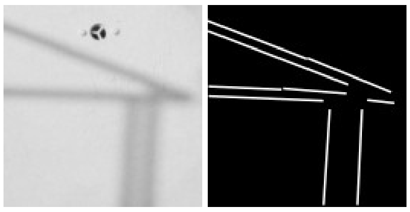




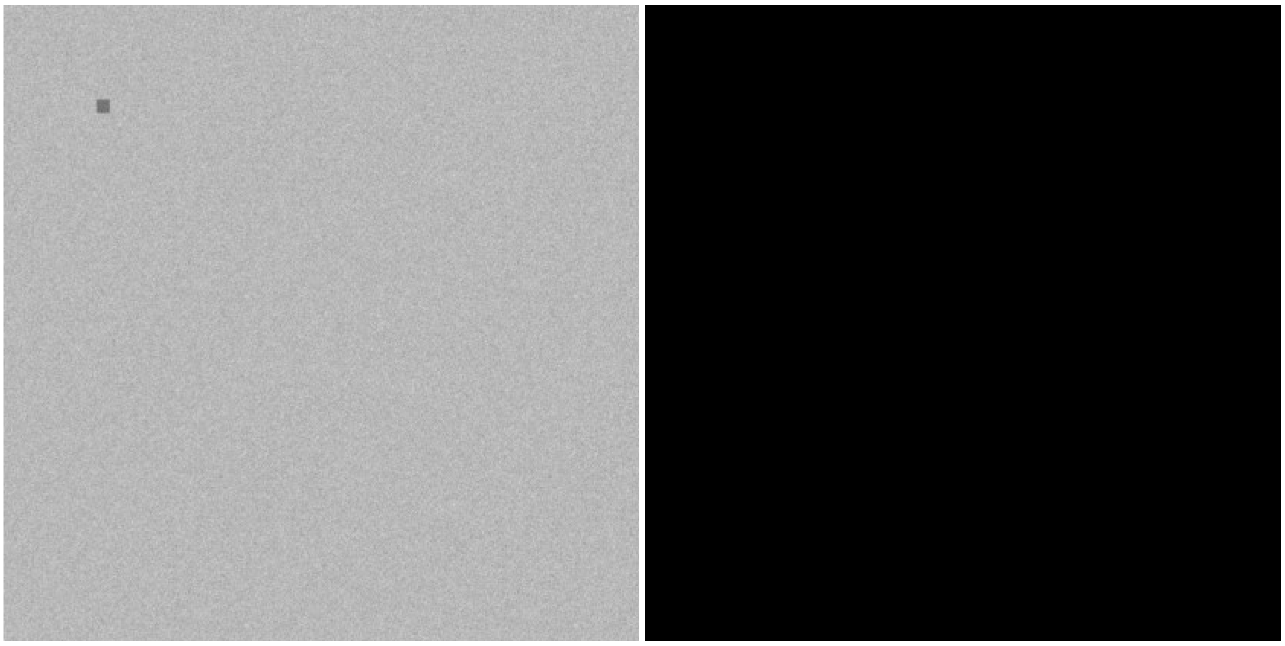
shadow and noise: A significant part of the visible straight structure in the following image is not detected. The reasons are the slow gradient in the shadow and the presence of noise. However, the structure can be detected by LSD at a different scale as is shown on the next example.



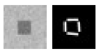
shadow and noise, subsampling: When a Gaussian sub-sampling is applied to the previous image, the noise is partially removed and the structure is analyzed at the right scale. The expected line segments are detected.



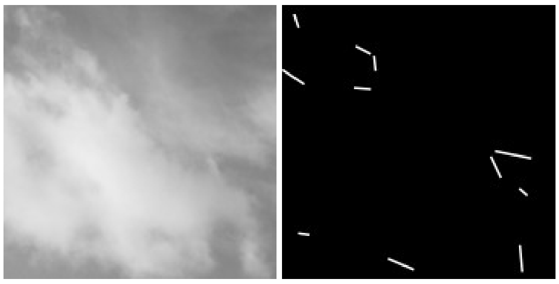
small: Due to the *a contrario* framework used by LSD to control the number of false detection, the result depends on the image size: the number of tests depends on it. As a consequence, the result of LSD may be locally different if the algorithm is applied to the full image or to a crop of it. The following image contains a little square just under the detection limit. No detection is thus produced. The next example, however, shows a crop of this same image and the square is indeed detected. This is the natural behavior of LSD and means that the detail level depends on the size of the whole data being analyzed. Human perception is similar: small details often go unnoticed unless attention is drawn to them.



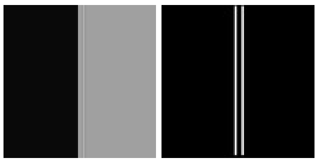
small, crop: A crop of the previous example, centered on the square. The square is now detected.



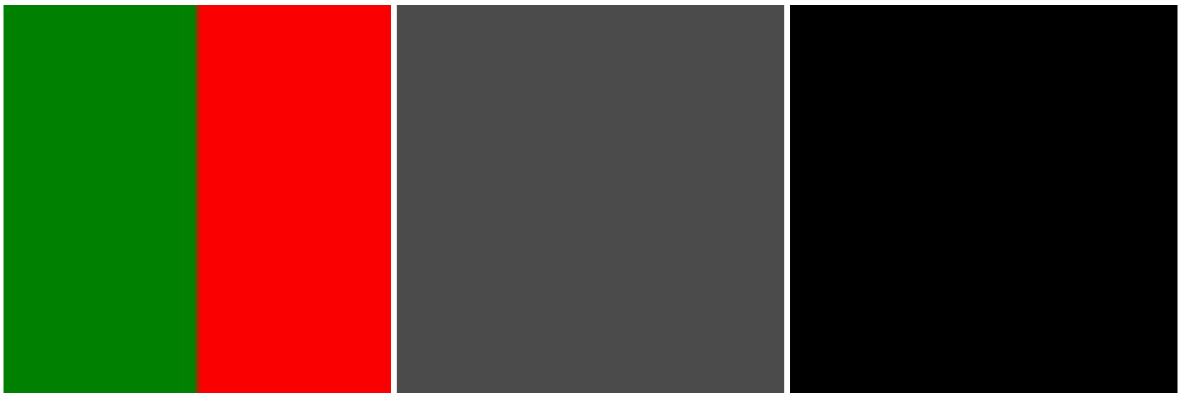
sky: Some regions are partially anisotropic and partially straight. Such regions can produce unexpected detections.



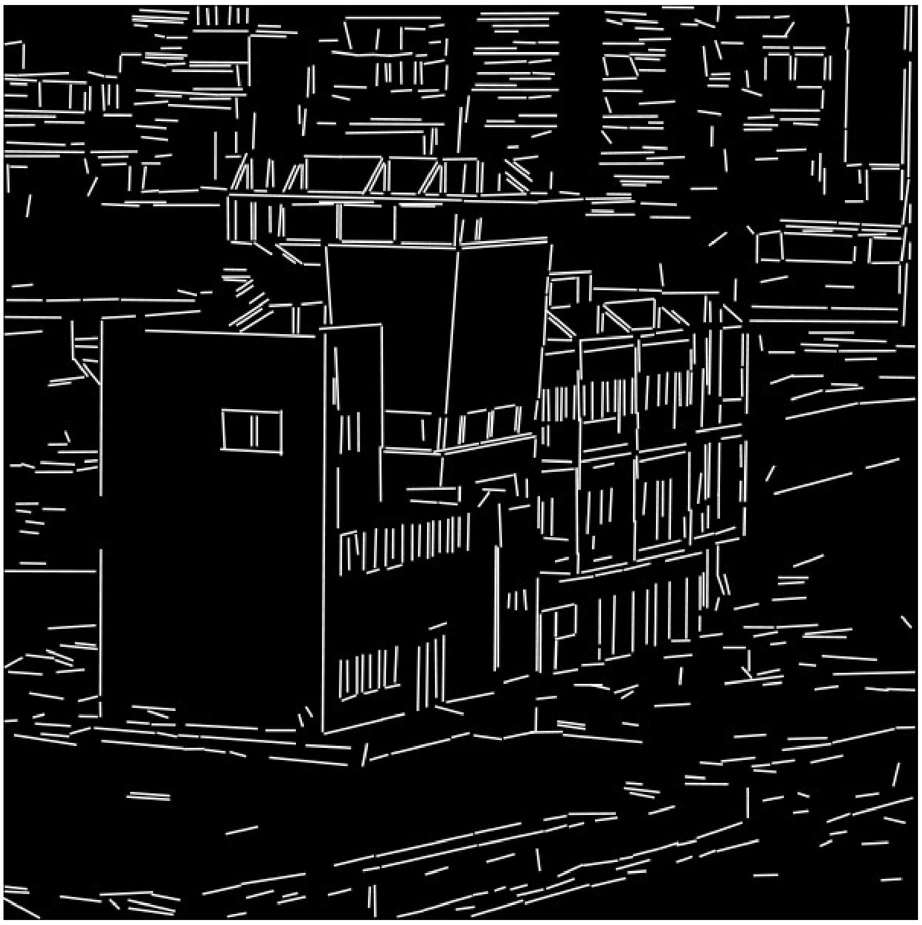
gibbs: Image compression, Gibbs effects are responsible for many unexpected detections.



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 clear edge (left), but after the standard conversion to a gray-level image (middle) the edge is lost. The reason is that the red value and the green value are both 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 by LSD. An extension of LSD to deal with this (relatively rare) event is possible but was not done in the current implementation.



real world scene: All in all, LSD usually produces a reasonable result on real images.



Video

This is an example of applying LSD, frame by frame, to a video: original(43Mb) lsd version (62Mb).

feeds & twitter sitemap contact privacy policy ISSN: 2105-1232 DOI: 10.5201/ipol
supported by CMLA, ENS Cachan DMI, Universitat de les Illes Balears Fing, Universidad de la República Gandi.net